FOSSEE Winter Internship Report

On

Development of New Installer method and Working on Report Generation of new modules for Osdag

Submitted by

Anuranjani

2nd Year B.Tech Student, Department of Electrical Engineering

Indian Institute of Technology

Jodhpur

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

January 8, 2025

Acknowledgments

- I would like to express my profound gratitude to all those who have supported and guided me throughout this enriching journey. This project has been a significant learning experience, made possible by the collective support of many individuals and institutions.
- My sincere thanks go to the Project staff at the Osdag team - Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia - for their constant support, technical expertise, and willingness to share their knowledge. Their mentorship has been invaluable in helping me overcome challenges and grow professionally.
- I am deeply grateful to Prof. Siddhartha Ghosh, Principal Investigator (PI) of the Osdag project at the Department of Civil Engineering, IIT Bombay, for his invaluable guidance and vision. His leadership has been instrumental in shaping this project and my understanding of the field.
- I extend my heartfelt appreciation to Prof. Kannan M. Moudgalya, FOSSEE PI from the Department of Chemical Engineering, IIT Bombay, for his oversight and direction.
- The FOSSEE managers, Usha Viswanathan and Vineeta Parmar, along with their entire team, have provided exceptional support and created an environment conducive to learning and growth.
- This project would not have been possible without the support of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India. Their commitment to advancing educational technology has made this opportunity possible.

- I am particularly thankful to my co-interns, whose collaboration and friendship made this journey both enjoyable and intellectually stimulating. The daily interactions, shared challenges, and mutual support have contributed significantly to my professional development.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
		Audio-Video e-content	
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
		Digital Content Access	
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
		Hands-on Learning	
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
		E-Governance	
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
		Tracking and Research Tools	
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.

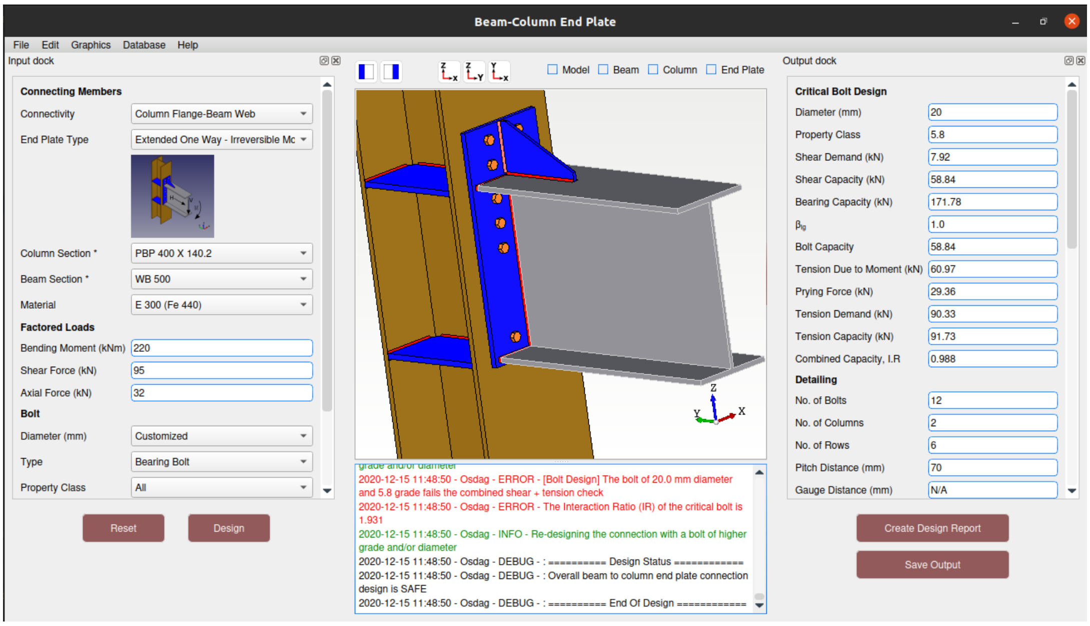

Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

We were provided with a 3D Prism Viewer application that needs to be tested, packaged, and documented. The task is divided into the following key areas:

1. **Unit Testing**: Develop comprehensive unit tests to validate the application's functionality, specifically focusing on the accuracy of surface area and volume calculations. The tests should adhere to Python's best practices for unit testing.

2. **Packaging**: Package the application using Conda, ensuring proper environment specifications and dependency management.

3. **Documentation**: Prepare and submit a detailed report in MS Word or LaTeX format. The report should document the testing methodology adopted and, packaging steps during the task.

2.2 Tasks Done

Below is a summary of the tasks performed as part of the screening process:

1. Developed and executed unit tests to verify the accuracy of surface area and volume calculations, ensuring the application adheres to its functional requirements.

Listing 2.1: Unit Tests File for the Prism Viewer

```python
import unittest
import sqlite3
import sys
from PyQt5.QtWidgets import QApplication
from prism_viewer.main import PrismViewer
from prism_viewer.prism_calculator import PrismCalculator

class TestPrismViewerApp(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        # Initialize the test database.
        cls.conn = sqlite3.connect('prisms.db')
        cls.cursor = cls.conn.cursor()

        # Create the application instance.
        cls.app = QApplication(sys.argv)
        cls.viewer = PrismViewer()

    def setUp(self):
        # Reset the application state before each test.
        self.viewer.designation_dropdown.setCurrentIndex(0)

    def test_program_initialization(self):
        self.assertEqual(self.viewer.windowTitle(), "Rectangular Prism Viewer")

        # Verify essential UI components.
        self.assertIsNotNone(self.viewer.designation_dropdown)
        self.assertIsNotNone(self.viewer.surface_area_label)
        self.assertIsNotNone(self.viewer.volume_label)
        self.assertIsNotNone(self.viewer.display_button)

        # Check the database connections.
        self.assertIsNotNone(self.viewer.conn)
        self.assertIsNotNone(self.viewer.cursor)

        # Check if the viewer has been created or not.
```

```python
        self.assertTrue(self.viewer.isHidden())

    @classmethod
    def tearDownClass(cls):
        cls.conn.close()

class TestPrismCalculator(unittest.TestCase):

    def test_surface_area_calculation(self):
        test_cases = [
            # (length, width, height, expected_area = 2 * (length * width + width * height + height * length))
            (10, 5, 2, 2 * (10 * 5 + 5 * 2 + 2 * 10)),    # Standard case
            (1, 1, 1, 6),                                 # Unit cube
            (10, 10, 10, 600),                            # Equal dimensions
        ]

        for length, width, height, expected in test_cases:
            with self.subTest(f"Testing surface area with L={length}, W={width}, H={height}"):
                result = PrismCalculator.surface_area(length, width, height)
                self.assertEqual(result, expected)

    def test_volume_calculation(self):
        test_cases = [
            # (length, width, height, expected_volume = length * width * height)
            (10, 5, 2, 10*5*2),   # Standard case
            (1, 1, 1, 1),         # Unit cube
            (10, 10, 10, 1000),   # Equal dimensions
        ]

        for length, width, height, expected in test_cases:
            with self.subTest(f"Testing volume with L={length}, W={width}, H={height}"):
```

```
                result = PrismCalculator.volume(length, width,
                    height)
                self.assertEqual(result, expected)

    def test_zero_value_calculation(self):
        # To check if zero is returned for zero dimensions.
        # Surface area tests
        self.assertEqual(PrismCalculator.surface_area(0, 0, 0), 0)
        self.assertEqual(PrismCalculator.surface_area(5, 0, 0), 0)
        self.assertEqual(PrismCalculator.surface_area(0, 5, 0), 0)
        self.assertEqual(PrismCalculator.surface_area(0, 0, 5), 0)

        # Volume tests
        self.assertEqual(PrismCalculator.volume(0, 0, 0), 0)
        self.assertEqual(PrismCalculator.volume(0, 5, 2), 0)
        self.assertEqual(PrismCalculator.volume(5, 0, 2), 0)
        self.assertEqual(PrismCalculator.volume(5, 2, 0), 0)

    def test_negative_value_calculation(self):
        # To ensure handling of errors for negative values.
        negative_cases = [
            (-10, 5, 2),
            (10, -5, 2),
            (10, 5, -2),
            (-1, -1, -1)
        ]

        for length, width, height in negative_cases:
            with self.subTest(f"Testing negative values L={length},
                W={width}, H={height}"):
                with self.assertRaises(ValueError):
                    PrismCalculator.surface_area(length, width,
                        height)
                with self.assertRaises(ValueError):
                    PrismCalculator.volume(length, width, height)

if __name__ == '__main__':
    unittest.main()
```

2. Packaged the Prism Viewer application using Conda, creating a well-defined environment with all dependencies managed effectively.

Listing 2.2: Conda-Recipe for the application

```
{% set name = "prism_viewer" %}
{% set version = "0.1.1" %}

package:
  name: {{ name|lower }}
  version: {{ version }}

source:
  path: ..

build:
  number: 0
  script: "{{ PYTHON }} -m pip install . -vv"
  entry_points:
    - prism_viewer = prism_viewer.main:main
  skip: true  # [py<311]

requirements:
  build:
    - python =3.11
    - {{ compiler('cxx') }}
  host:
    - python =3.11
    - pip
    - setuptools =75.3.0
    - numpy =1.26.4
    - swig
    - pythonocc-core =7.8.1
    - pyqt =5.15.9
  run:
    - python =3.11
    - {{ pin_compatible('numpy') }}
    - pyqt =5.15.9
    - pythonocc-core =7.8.1
    - sqlite
```

```
    - occt =7.8.1
    - six =1.16.0
    - svgwrite
    - qt =5.15.9

test:
  imports:
    - prism_viewer
    - OCC
    - PyQt5
  requires:
    - unittest-xml-reporting
  commands:
    - python -m unittest discover -s prism_viewer/tests/
  source_files:
    - prism_viewer/tests/

about:
  home: "https://github.com/anuranjani23/fossee-3D-rectangular-prism-viewer.git"
  license: MIT
  license_family: MIT
  license_file: LICENSE
  summary: "A PyQt5 and PythonOCC-based 3D rectangular prism viewer application"
  description: |
    A 3D viewer application for rectangular prisms built using PyQt5 and PythonOCC.
    Features include surface area and volume calculations, interactive 3D visualization,
    and SQLite database storage.
  doc_url: https://github.com/anuranjani23/fossee-3D-rectangular-prism-viewer/blob/main/README.md
  dev_url: https://github.com/anuranjani23/fossee-3D-rectangular-prism-viewer

extra:
  recipe-maintainers:
    - anuranjani23
```

```
  platforms:
    - linux
    - osx
    - win-64
```

Listing 2.3: Enviornment YML file

```
name: prism_viewer_env
channels:
  - conda-forge
dependencies:
  - python=3.11
  - pyqt=5.15.9
  - numpy=1.26.4
  - pythonocc-core=7.8.1
  - swig
  - sqlite
  - occt=7.8.1
  - six=1.16.0
  - svgwrite
  - qt=5.15.9
  - pip
```

3. Documented the entire process, including testing approaches, packaging methods, and the rationale behind critical decisions, in a structured report prepared using LaTeX. The report can be found here.

4. Demonstrated proficiency in Python programming, focusing on Object-Oriented Programming (OOP) principles, unit testing, and industry-standard packaging practices with Conda and PIP building and packaging.

Chapter 3

Internship Task 1: Resolve Import Path Issues and Float Error for Mac UI Template

3.1 Task 1: Problem Statement

The task involved resolving critical issues in the Mac UI template concerning float handling and import paths. The primary challenges involved debugging and fixing float-related errors that were affecting the UI rendering, along with restructuring the import system.

3.2 Task 1: Tasks Done

My responsibilities included identifying and resolving float computation errors, removing redundant imports that were cluttering the codebase, and standardizing the import paths to use relative references for better maintainability. This optimization task aimed to enhance the template's reliability and maintain consistent coding standards across the UI framework.

3.3 Task 1: Python Code

This section presents the changes made to the Python script for the UI template of Mac. The figures below illustrate these changes:

```diff
src/osdag/gui/ui_template_for_mac.py
 from ..design_type.connection.column_end_plate import ColumnEndPlate
 from ..design_type.connection.column_cover_plate_weld import ColumnCoverPlateWeld
 from ..design_type.connection.base_plate_connection import BasePlateConnection
@@ -78,7 +90,6 @@
 import logging
 import subprocess
 from ..get_DPI_scale import scale
-from ..cad.cad3dconnection import cadconnection
 from OCC.Display.backend import load_backend, get_qt_modules
 from ..osdagMainSettings import backend_name
 used_backend = load_backend(backend_name())
@@ -121,7 +132,7 @@ def __init__(self, main, folder, parent=None):
         resolution = QtWidgets.QDesktopWidget().screenGeometry()
         width = resolution.width()
         height = resolution.height()
-        self.resize(width * (0.75), height * (0.7))
+        self.resize(int(width * 0.75), int(height * 0.7))
         self.ui = Window()
         self.ui.setupUi(self, main, folder)
         # self.showMaximized()
@@ -146,21 +157,22 @@ def resize_dockComponents(self):
         # Input Dock
         width = self.ui.inputDock.width()
         self.ui.inputDock.resize(width, self.height())
-        self.ui.in_widget.resize(width, posi)
+        self.ui.in_widget.resize(int(width), int(posi))

-        self.ui.btn_Reset.move((width / 2) - 110, posi + 8)
-        self.ui.btn_Design.move((width / 2) + 17, posi + 8)
+
+        self.ui.btn_Reset.move(int((width / 2)) - 110, int(posi + 8))
+        self.ui.btn_Design.move(int((width / 2)) + 17, int(posi + 8))
         # self.ui.btn_Design.move(,posi+10)

         # Output Dock
         width = self.ui.outputDock.width()
         self.ui.outputDock.resize(width, self.height())
-        self.ui.out_widget.resize(width, posi)
-        self.ui.btn_CreateDesign.move((width / 2) - (186 / 2), posi + 8)
-        self.ui.save_outputDock.move((width / 2) - (186 / 2), posi + 52)
+        self.ui.out_widget.resize(int(width), int(posi))
+        self.ui.btn_CreateDesign.move(int((width / 2) - (186 / 2)), int(posi + 8))
+        self.ui.save_outputDock.move(int((width / 2) - (186 / 2)), int(posi + 52))

         # Designed model
-        self.ui.splitter.setSizes([0.85 * posi, 0.15 * posi])
+        self.ui.splitter.setSizes([int(0.85 * posi), int(0.15 * posi)])
         self.ui.modelTab.setFocus()
         self.ui.display.FitAll()

@@ -235,7 +247,7 @@ def open_summary_popup(self, main):
```

```diff
src/osdag/gui/ui_template_for_mac.py
@@ -235,7 +247,7 @@ def open_summary_popup(self, main):

         self.new_window = QtWidgets.QDialog(self)
         self.new_ui = Ui_Dialog1(main.design_status,loggermsg=self.textEdit.toPlainText())
-        self.new_ui.setupUi(self.new_window, main)
+        self.new_ui.setupUi(self.new_window, main, main)
         self.new_ui.btn_browse.clicked.connect(lambda: self.getLogoFilePath(self.new_window, self.new_ui.lbl_browse))
         self.new_ui.btn_saveProfile.clicked.connect(lambda: self.saveUserProfile(self.new_window))
         self.new_ui.btn_useProfile.clicked.connect(lambda: self.useUserProfile(self.new_window))
@@ -733,8 +745,9 @@ def setupUi(self, MainWindow, main,folder):
         maxi_width += 82
         print('maxiwidth',maxi_width)
         maxi_width = max(maxi_width, scale*350)    # In case there is no widget
-        self.inputDock.setFixedWidth(maxi_width)
-        self.in_widget.setFixedWidth( maxi_width)
+        self.inputDock.setFixedWidth(int(maxi_width))
+
+        self.in_widget.setFixedWidth(int(maxi_width))
         for option in option_list:
             key = self.dockWidgetContents.findChild(QtWidgets.QWidget, option[0])

@@ -812,7 +825,7 @@ def setupUi(self, MainWindow, main,folder):
                     self.on_change_connect(key_changed, updated_list, data, main)

         self.btn_Reset = QtWidgets.QPushButton(self.dockWidgetContents)
-        self.btn_Reset.setGeometry(QtCore.QRect((maxi_width/2)-110, 650, 100, 35))
+        self.btn_Reset.setGeometry(QtCore.QRect(int((maxi_width / 2) - 110), 650, 100, 35))
         font = QtGui.QFont()
         font.setPointSize(10)
         font.setBold(True)
@@ -822,7 +835,7 @@ def setupUi(self, MainWindow, main,folder):
         self.btn_Reset.setObjectName("btn_Reset")

         self.btn_Design = QtWidgets.QPushButton(self.dockWidgetContents)
-        self.btn_Design.setGeometry(QtCore.QRect((maxi_width/2)+10, 650, 100, 35))
+        self.btn_Design.setGeometry(QtCore.QRect(int((maxi_width / 2) + 10), 650, 100, 35))
         font = QtGui.QFont()
         font.setPointSize(10)
         font.setBold(True)
@@ -995,8 +1008,8 @@ def setupUi(self, MainWindow, main,folder):
         out_scroll.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
         out_scroll.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)

-        self.outputDock.setFixedWidth(maxi_width)
-        self.out_widget.setFixedWidth(maxi_width)
+        self.outputDock.setFixedWidth(int(maxi_width))
+        self.out_widget.setFixedWidth(int(maxi_width))
         self.outputDock.setSizePolicy(QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,QtWidgets.QSizePolicy.Maximum))
         self.out_widget.setSizePolicy(QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,QtWidgets.QSizePolicy.Maximum))
```

src/osdag/gui/ui_template_for_mac.py

```
        out_scroll.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
        out_scroll.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)

-       self.outputDock.setFixedWidth(maxi_width)
-       self.out_widget.setFixedWidth(maxi_width)
+       self.outputDock.setFixedWidth(int(maxi_width))
+       self.out_widget.setFixedWidth(int(maxi_width))
        self.outputDock.setSizePolicy(QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,QtWidgets.QSizePolicy.Maximum))
        self.out_widget.setSizePolicy(QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,QtWidgets.QSizePolicy.Maximum))
        # common_button = QtWidgets.QPushButton()
@@ -1052,7 +1065,7 @@ def setupUi(self, MainWindow, main,folder):
        self.save_outputDock.setObjectName("save_outputDock")
        self.save_outputDock.setText("Save Output")
        self.save_outputDock.clicked.connect(self.save_output_to_csv(main))
-       # self.btn_CreateDesign.clicked.connect(self.createDesignReport(main))
+       #self.btn_CreateDesign.clicked.connect(self.createDesignReport(main))

        ##################################
        # Menu UI
@@ -1407,7 +1420,7 @@ def setupUi(self, MainWindow, main,folder):
                out_titles.append(title_name)
        self.ui_loaded = True

-       from osdagMainSettings import backend_name
+       from ..osdagMainSettings import backend_name
        self.display, _ = self.init_display(backend_str=backend_name(),window=MainWindow)
        self.connectivity = None
        self.fuse_model = None
@@ -1816,8 +1829,8 @@ def return_class(self,name):
            return BeamCoverPlate
        elif name == KEY_DISP_BEAMCOVERPLATEWELD:
            return BeamCoverPlateWeld
-       elif name == KEY_DISP_BEAMENDPLATE:
-           return BeamEndPlate
+       elif name == KEY_DISP_BB_EP_SPLICE:
+           return BeamBeamEndPlateSplice
        elif name == KEY_DISP_COLUMNENDPLATE:
            return ColumnEndPlate
        elif name == KEY_DISP_BASE_PLATE:
```

Chapter 4

Internship Task 2 Title: Resolve LaTeX Report Issues for Compression Members

4.1 Task 2: Problem Statement

Fixing and completing the Report Generation for both modules of Compression Members, "Columns with known support conditions" and "Struts in Trusses".

4.2 Task 2: Tasks Done

The task primarily focused on debugging the Python source files associated with both modules to ensure the successful generation of the report. This involved addressing errors in the code and ensuring all components worked seamlessly together. The next step was to complete and refine the code responsible for generating the content of the report, including integrating CAD-generated images into the appropriate sections of the document.

To achieve this, several new functions were written, while existing functions were carefully reviewed and fixed to address any issues. Additionally, meticulous attention was given to verifying calculations, ensuring accurate results. The LaTeX formatting of the report was also reviewed and corrected to maintain a cleaner appearance, including proper alignment, spacing, and consistency. The integration of images into the report was carefully managed to ensure they were placed in the correct order and contextually aligned with the content. With this, addition of the failed design Report was also ensured, i.e. if the CAD generation fails, the Report is able to generate still.

Another critical aspect of the task was fixing the design log. This involved resolving issues with excessive or overflowing logger messages, which were being inappropriately printed in the report. By implementing these fixes, the task aimed to ensure that the report generation process was robust, accurate, and met all requirements.

4.3 Task 2: Python Code

The figures below describe the changes made, and the new functions that were added accordingly:

4.3.1 Report_functions.py

Firstly in the `Report_functions.py`, the functions that were added are as follows: Both the `cl_3_7_2_section_classification_angle_required` function and the `cl_3_7_2_section_classification_angle_provided` function were responsible for the Section Classification check in the "Struts in Trusses" module. Then in the same module, the `cl_7_5_1_2_effective_slenderness_ratio` function was added which is used to print Slenderness Ration when load_type is not "Concentric Load". Then after that, `calculate_buckling_class` was added to calculate the Buckling Class curve for the "Column with known support conditions" module, using this function, both the functions `comp_column_class_section_check_required` and `comp_column_class_section_check_provided` were responsible for the addition of Buckling Class - Compatibility check in the Design Check of the Report of same module.

```
src/osdag/Report_functions.py

@@ -128,6 +128,84 @@ def cl_3_7_2_section_classification_flange(d,t,result,epsilon,class_of_section=N
         eqn.append(NoEscape(r'\begin{aligned} & \textbf{Slender} \end{aligned}'))
     # eqn.append(NoEscape(r'& [\text{Ref: Table 2, Cl.3.7.2 and 3.7.4, IS 800:2007}] \end{aligned}'))
     return eqn
+
+
+def cl_3_7_2_section_classification_angle_required(ratio_type, class_of_section=None):
+    """
+    Provide the required conditions for angle section classification based on IS 800:2007, Cl.3.7.2.
+
+    Args:
+        ratio_type: Type of ratio to be calculated ('b/t', 'd/t', 'b+d/t')
+        class_of_section: Expected classification ('Plastic', 'Compact', 'Semi-Compact', 'Slender')
+        epsilon: Material constant (float)
+
+    Returns:
+        A LaTeX equation showing the required classification conditions.
+    """
+    eqn = Math(inline=True)
+
+    if class_of_section in ["Plastic", "Compact"]:
+        eqn.append(NoEscape(r'\begin{aligned} \text{For ' + class_of_section + r' Section:} \\'))
+        eqn.append(NoEscape(r'\text{No Specific Ratio Limit} \end{aligned}'))
+
+    elif class_of_section == "Semi-Compact":
+        if ratio_type == 'b/t':
+            eqn.append(NoEscape(r'\begin{aligned} \\'))
+            eqn.append(NoEscape(r'\frac{b}{t} \leq 15.7\varepsilon'))
+            eqn.append(NoEscape(r'\end{aligned}'))
+        elif ratio_type == 'd/t':
+            eqn.append(NoEscape(r'\begin{aligned} \\'))
+            eqn.append(NoEscape(r'\frac{d}{t} \leq 15.7\varepsilon'))
+            eqn.append(NoEscape(r'\end{aligned}'))
+        elif ratio_type == '(b+d)/t':
+            eqn.append(NoEscape(r'\begin{aligned}  \\'))
+            eqn.append(NoEscape(r'\frac{b+d}{t} \leq 25\varepsilon'))
+            eqn.append(NoEscape(r'\end{aligned}'))
+
+    else:
+        raise ValueError("Invalid section classification. Choose from 'Plastic', 'Compact', 'Semi-Compact'.")
+
+    return eqn
+
+def cl_3_7_2_section_classification_angle_provided(b, d, t, ratio_value, ratio_type, epsilon, class_of_section=None):
+    """
+    Provide the numerical values for angle section classification based on IS 800:2007, Cl.3.7.2.
+
+    Args:
+        b: Width of the angle leg (float)
+        d: Depth of the angle (float)
+        t: Thickness of the leg (float)
+        ratio_type: Type of ratio to be calculated ('b/t', 'd/t', 'b+d/t')
+        ratio_value: The pre-calculated ratio value (float)
```

Figure 4.1: From Report_functions.py

```
src/osdag/Report_functions.py
+        d: Depth of the angle (float)
+        t: Thickness of the leg (float)
+        ratio_type: Type of ratio to be calculated ('b/t', 'd/t', 'b+d/t')
+        ratio_value: The pre-calculated ratio value (float)
+        class_of_section: Expected classification ('Plastic', 'Compact', 'Semi-Compact', 'Slender')
+        epsilon: Material constant (float)
+
+    Returns:
+        A LaTeX equation showing the numerical values and classification.
+    """
+    eqn = Math(inline=True)
+
+    if ratio_type == 'b/t':
+        eqn.append(NoEscape(r'\begin{aligned}'))
+        eqn.append(NoEscape(r'\frac{b}{t} = \frac{' + str(b) + '}{' + str(t) + '} = ' + str(ratio_value) + r' \leq 15.7\varepsilon \\'))
+        eqn.append(NoEscape(r'& \textbf{' + class_of_section + r'} \end{aligned}'))
+
+    elif ratio_type == 'd/t':
+        eqn.append(NoEscape(r'\begin{aligned}'))
+        eqn.append(NoEscape(r'\frac{d}{t} = \frac{' + str(d) + '}{' + str(t) + '} = ' + str(ratio_value) + r' \leq 15.7\varepsilon \\'))
+        eqn.append(NoEscape(r'& \textbf{' + class_of_section + r'} \end{aligned}'))
+
+    elif ratio_type == '(b+d)/t':
+        eqn.append(NoEscape(r'\begin{aligned}'))
+        eqn.append(NoEscape(r'\frac{b+d}{t} = \frac{' + str(b+d) + '}{' + str(t) + '} = ' + str(ratio_value) + r' \leq 25\varepsilon \\'))
+        eqn.append(NoEscape(r'& \textbf{' + class_of_section + r'} \end{aligned}'))
+
+    else:
+        raise ValueError("Invalid ratio type. Choose from 'b/t', 'd/t', '(b+d)/t'.")
+
+    return eqn
+
+
 def cl_5_4_1_table_4_5_gamma_value(v, t):
     """
     Calculate gamma value
@@ -448,6 +526,40 @@ def cl_7_1_2_effective_slenderness_ratio(K, L, r, slender):
     slender_eqn.append(NoEscape(r'& [\text{Ref. IS 800:2007, Cl.7.1.2}] \end{aligned}'))
     return slender_eqn
 
+def cl_7_5_1_2_effective_slenderness_ratio(k1, k2, k3, lmb_v, lmb_phi, slender):
+    """
+    Calculate effective slenderness ratio based on given parameters.
+
+    Args:
+        k1: Constant k1 (float)
+        k2: Constant k2 (float)
+        k3: Constant k3 (float)
+        lmb_v: Slenderness parameter λv (float)
+        lmb_phi: Slenderness parameter λϕ (float)
+        slender: Effective slenderness ratio λe (float)
+
```

Figure 4.2: From Report_functions.py

```
src/osdag/Report_functions.py
+        lmb_v: Slenderness parameter λv (float)
+        lmb_phi: Slenderness parameter λϕ (float)
+        slender: Effective slenderness ratio λe (float)
+
+    Returns:
+        LaTeX representation of the effective slenderness ratio calculation.
+
+    Note:
+        Reference:
+        IS 800:2007, Cl.7.5.1.2
+    """
+    k1 = str(k1)
+    k2 = str(k2)
+    k3 = str(k3)
+    lmb_v = str(lmb_v)
+    lmb_phi = str(lmb_phi)
+    slender = str(slender)
+
+    slender_eqn = Math(inline=True)
+    slender_eqn.append(NoEscape(r'\begin{aligned} \lambda_e &= \sqrt{k_1 + k_2 \cdot \lambda_v^2 + k_3 \cdot \lambda_\phi^2} \\'))
+    slender_eqn.append(NoEscape(r'&= \sqrt{' + k1 + r' + ' + k2 + r' \cdot ' + lmb_v + r'^2 + ' + k3 + r' \cdot ' + lmb_phi + r'^2} \\'))
+    slender_eqn.append(NoEscape(r'&= ' + slender + r' \\ \\'))
+    slender_eqn.append(NoEscape(r'& [\text{Ref. IS 800:2007, Cl.7.5.1.2}] \end{aligned}'))
+
+    return slender_eqn
+

 def cl_8_2_moment_capacity_member(Pmc, Mdc, M_c):
     """
@@ -1325,10 +1437,9 @@ def cl_8_7_1_5_buckling_curve(sub = 'c'):

     """

-    sub = str(sub)
+    sub = str(sub).upper()
     slender_eqn = Math(inline=True)
-    slender_eqn.append(NoEscape(r'\begin{aligned} &= c \\'))
-
+    slender_eqn.append(NoEscape(r'\begin{aligned} &= ' + sub + r' \\'))
     slender_eqn.append(NoEscape(r'& [\text{Ref. IS 800:2007, Cl.8.7.3.1}] \end{aligned}'))
     return slender_eqn

@@ -4066,7 +4177,7 @@ def lever_arm_end_plate(lever_arm, bolt_row, ep_type=''):
     return display_eqn


-def get_pass_fail(required, provided, relation='',M1 = '', M2 = ''):
+def get_pass_fail(required, provided, relation='',M1=''):
     if provided == 0 or required == 'N/A' or provided == 'N/A' or required == 0:
         return ''
     else:
```

Figure 4.3: From Report_functions.py

```diff
src/osdag/Report_functions.py
            return ''
        else:
@@ -8778,65 +8889,122 @@ def prov_moment_load_bp(moment_input, min_mc, app_moment_load, moment_capacity,
    return app_moment_load_eqn


-
-def comp_column_class_section_check_required( bucklingclass , h , bf ):
+def calculate_buckling_class(h, bf, tf, axis):
+    """
+    Determines the buckling class using IS 800:2007 checks.
+    axis: Can be "ZZ" or "YY"
+    """
+    h_bf_ratio = h / bf
+    tf_limit = tf
+
+    if axis == "ZZ":
+        if h_bf_ratio <= 1.2 and tf_limit > 100:
+            return "D"
+        elif h_bf_ratio <= 1.2 and tf_limit <= 100:
+            return "C"
+        elif h_bf_ratio > 1.2 and tf_limit <= 40:
+            return "B"
+        elif h_bf_ratio > 1.2 and 40 < tf_limit <= 100:
+            return "C"
+
+    elif axis == "YY":
+        if h_bf_ratio > 1.2 and tf_limit <= 40:
+            return "A"
+        elif h_bf_ratio > 1.2 and 40 < tf_limit <= 100:
+            return "B"
+        elif h_bf_ratio <= 1.2 and tf_limit <= 100:
+            return "B"
+        elif h_bf_ratio <= 1.2 and tf_limit > 100:
+            return "D"
+    else:
+        return "Invalid Axis"
+def comp_column_class_section_check_required(h, bf, tf, axis):
    """
    Args:
-        h:Depth of section(mm)                                                   (float)
-        bf: Breadth of section(mm)                                               (float)
-        bucklingclass: buckling class                                            (float)
+        h: Depth of section (mm)                                  (float)
+        bf: Breadth of section (mm)                               (float)
+        tf: Thickness of flange (mm)                              (float)
+        axis: Axis for buckling class ("YY" or "ZZ")              (str)
    Returns:
-            bucklingclass_eq
-    Note:
-        Reference: IS 800 Cl.7.1.2.2
-        @author:Rutvik Joshi
```

Figure 4.4: From Report_functions.py

```diff
src/osdag/Report_functions.py
-    Note:
-        Reference: IS 800 Cl.7.1.2.2
-        @author:Rutvik Joshi
+        bucklingclass_eq: LaTeX formatted buckling class equation        (Math object)
     """
+    bucklingclass_eq = Math(inline=True)
+
+    # Calculate buckling class
+    calculated_buckling_class = calculate_buckling_class(h, bf, tf, axis)
+
     bucklingclass_eq=Math(inline=True)
-    if bucklingclass==0.34:
-        bucklingclass_eq.append(NoEscape(r'\begin{aligned} frac{h}{b_\text{f}}>1.2\\'))
+    if calculated_buckling_class=="A":
+        bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}>1.2\\'))
         bucklingclass_eq.append(NoEscape(r'                 t_\text{f}<=40\\'))
-        bucklingclass_eq.append(NoEscape(r'                 \end(aligned)'))
-    elif bucklingclass==0.49:
+        bucklingclass_eq.append(NoEscape(r'                 \end{aligned}'))
+    elif calculated_buckling_class=="B":
         if h/bf>1.2:
-            bucklingclass_eq.append(NoEscape(r'\begin{aligned} frac{h}{b_\text{f}}>1.2\\'))
-            bucklingclass_eq.append(NoEscape(r'40 <= t_\text{f} <= 100'))
-            bucklingclass_eq.append(NoEscape(r'                 \end(aligned)'))
+            if axis=="YY":
+                bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}>1.2\\'))
+                bucklingclass_eq.append(NoEscape(r'40 < t_\text{f} <= 100'))
+                bucklingclass_eq.append(NoEscape(r'                 \end{aligned}'))
+            else:
+                bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}>1.2\\'))
+                bucklingclass_eq.append(NoEscape(r'                 t_\text{f}<=40\\'))
+                bucklingclass_eq.append(NoEscape(r'                 \end{aligned}'))
         else:
-            bucklingclass_eq.append(NoEscape(r'\begin{aligned} frac{h}{b_\text{f}}<=1.2\\'))
+            bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}<=1.2\\'))
             bucklingclass_eq.append(NoEscape(r' t_\text{f} <= 100'))
-            bucklingclass_eq.append(NoEscape(r'                 \end(aligned)'))
-    elif bucklingclass==0.76:
-        bucklingclass_eq.append(NoEscape(r'\begin{aligned} frac{h}{b_\text{f}}<=1.2\\'))
-        bucklingclass_eq.append(NoEscape(r' t_\text{f} > 100'))
-        bucklingclass_eq.append(NoEscape(r'                 \end(aligned)'))
+            bucklingclass_eq.append(NoEscape(r'                 \end{aligned}'))
+    elif calculated_buckling_class=="C":
+        if h/bf>1.2:
+            bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}>1.2\\'))
+            bucklingclass_eq.append(NoEscape(r'40 < t_\text{f} <= 100'))
+            bucklingclass_eq.append(NoEscape(r'                 \end{aligned}'))
+        else:
+            bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}<=1.2\\'))
+            bucklingclass_eq.append(NoEscape(r' t_\text{f} <= 100'))
+            bucklingclass_eq.append(NoEscape(r'                 \end{aligned}'))
+    elif calculated_buckling_class=="D":
+        if axis=="YY":
```

Figure 4.5: From Report_functions.py

```diff
src/osdag/Report_functions.py
+    elif calculated_buckling_class=="D":
+        if axis=="YY":
+            bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}<=1.2\\'))
+            bucklingclass_eq.append(NoEscape(r' t_\text{f} > 100'))
+            bucklingclass_eq.append(NoEscape(r'                  \end{aligned}'))
+        else:
+            bucklingclass_eq.append(NoEscape(r'\begin{aligned} \frac{h}{b_\text{f}}<=1.2\\'))
+            bucklingclass_eq.append(NoEscape(r' t_\text{f} > 100'))
+            bucklingclass_eq.append(NoEscape(r'                  \end{aligned}'))
     return bucklingclass_eq

-
-def comp_column_class_section_check_provided( bucklingclass , h , bf , tf , var_h_bf ):
+def comp_column_class_section_check_provided(h, bf, tf, var_h_bf, axis):
     """
     Args:
-        h:Depth of section(mm)                                                    (float)
-        bf: Breadth of section(mm)                                                (float)
-        bucklingclass: buckling class                                              (float)
+        h: Depth of section (mm)                             (float)
+        bf: Breadth of section (mm)                           (float)
+        tf: Thickness of flange (mm)                          (float)
+        var_h_bf: Calculated h/bf ratio                       (float)
     Returns:
-            bucklingclass_eq
-    Note:
-        Reference: IS 800 Cl.7.1.2.2
-        @author:Rutvik Joshi
-    """
-    bucklingclass_eq=Math(inline=True)
-    h=str(h)
-    bf=str(bf)
-    tf=str(tf)
-    var_h_bf=str(var_h_bf)
-
-    bucklingclass_eq.append(NoEscape(r'\begin{aligned} frac{h}{b_\text{f}}&= frac{'+ h +r'}{' + bf + r'}\\'))
+        bucklingclass_eq: LaTeX formatted buckling class equation  (Math object)
+    """
+    bucklingclass_eq = Math(inline=True)
+
+    # Calculate buckling class first
+    calculated_buckling_class = calculate_buckling_class(h, bf, tf, axis)
+
+    # Convert values to strings
+    h = str(h)
+    bf = str(bf)
+    tf = str(tf)
+    var_h_bf = str(var_h_bf)
+
+    # Append the LaTeX formatted equations including buckling class
+    bucklingclass_eq.append(NoEscape(r'\begin{aligned} \text{Buckling Class: }' + calculated_buckling_class + r'\\'))
+    bucklingclass_eq.append(NoEscape(r'\frac{h}{b_\text{f}}&= \frac{'+ h +r'}{' + bf + r'}\\'))
     bucklingclass_eq.append(NoEscape(r'                                &='+ var_h_bf +r'\\'))
```

Figure 4.6: From Report_functions.py

4.3.2 reportGenerator_latex.py

Custom table formatting for Imperfection Factor Check in "Column with known support conditions" module.

```
src/osdag/design_report/reportGenerator_latex.py
@@ -232,6 +232,19 @@ def save_latex(self, uiObj, Design_Check, reportsummary, filename, rel_path, Dis
                            table.end_table_header()
                            table.add_hline()
                            count = count + 1
+                elif check[0] == 'NewTable':
+                    if count >= 1:
+                        # doc.append(NewPage())
+                        doc.append(pyl.Command('Needspace', arguments=NoEscape(r'10\baselineskip')))
+
+                    with doc.create(Subsection(check[1])):
+                        with doc.create(LongTable(check[2], row_height=1.2)) as table:
+                            table.add_hline()
+                            table.add_row(('Axes', 'Buckling Class', 'Imperfection Factor', ''), color='OsdagGreen')
+                            table.add_hline()
+                            table.end_table_header()
+                            table.add_hline()
+                            count = count + 1
                elif check[0] == "Selected":
                    if count >=1:
                        # doc.append(NewPage())
@@ -471,4 +484,4 @@ def save_latex(self, uiObj, Design_Check, reportsummary, filename, rel_path, Dis

def color_cell(cellcolor,celltext):
    string = NoEscape(r'\cellcolor{'+cellcolor+r'}{'+celltext+r'}')
-    return string
+    return string
```

Figure 4.7: From reportGenerator_latex.py

4.3.3 Column.py

In the `Column.py`, `section_classification` and `save_design` functions were modified and `common_result` function was added, a failure dictionary was added as well to fall back to for report generation when design_status fails. The last 4 images were integrated as well.

```
src/osdag/design_type/compression_member/Column.py
@@ -573,6 +573,18 @@ def func_for_validation(self, design_dictionary):
         if flag:
             print(f"\n design_dictionary{design_dictionary}")
             self.set_input_values(self, design_dictionary)
+            if self.design_status ==False and len(self.failed_design_dict)>0:
+                logger.error(
+                    "Design Failed, Check Design Report"
+                )
+                return # ['Design Failed, Check Design Report'] @TODO
+            elif self.design_status:
+                pass
+            else:
+                logger.error(
+                    "Design Failed. Slender Sections Selected"
+                )
+                return # ['Design Failed. Slender Sections Selected']
         else:
             return all_errors

@@ -677,7 +689,7 @@ def set_input_values(self, design_dictionary):
         # initialize the design status
         self.design_status_list = []
         self.design_status = False
-
+        self.failed_design_dict = {}
         flag = self.section_classification(self)
         print(flag)
         if flag:
@@ -775,11 +787,8 @@ def section_classification(self):
```

Figure 4.8: From Column.py

```
src/osdag/design_type/compression_member/Column.py

@@ -21,7 +21,7 @@
 from ..member import Member
 from ...Report_functions import *
 from ...design_report.reportGenerator_latex import CreateLatex
-
+from pylatex.utils import NoEscape

 class ColumnDesign(Member):

@@ -731,19 +731,28 @@ def section_classification(self):
                 self.web_class = IS800_2007.Table2_iii((self.section_property.depth - (2 * self.section_property.flange_thickness)),
                                                        self.section_property.web_thickness, self.material_property.fy,
                                                        classification_type='Axial compression')
+                web_ratio = (self.section_property.depth - 2 * (
+                        self.section_property.flange_thickness + self.section_property.root_radius)) / self.section_property.web_thickness
+                flange_ratio = self.section_property.flange_width / 2 / self.section_property.flange_thickness

             elif (self.sec_profile == VALUES_SEC_PROFILE[1]):  # RHS and SHS
                 self.flange_class = IS800_2007.Table2_iii((self.section_property.depth - (2 * self.section_property.flange_thickness)),
                                                           self.section_property.flange_thickness, self.material_property.fy,
                                                           classification_type='Axial compression')
                 self.web_class = self.flange_class
+                web_ratio = (self.section_property.depth - 2 * (
+                        self.section_property.flange_thickness + self.section_property.root_radius)) / self.section_property.web_thickness
+                flange_ratio = self.section_property.flange_width / 2 / self.section_property.flange_thickness

             elif self.sec_profile == VALUES_SEC_PROFILE[2]:  # CHS
                 self.flange_class = IS800_2007.Table2_x(self.section_property.out_diameter, self.section_property.flange_thickness,
                                                         self.material_property.fy, load_type='axial compression')
                 self.web_class = self.flange_class  #Why?
+                web_ratio = (self.section_property.depth - 2 * (
+                        self.section_property.flange_thickness + self.section_property.root_radius)) / self.section_property.web_thickness
+                flange_ratio = self.section_property.flange_width / 2 / self.section_property.flange_thickness
                 # print(f"self.web_class{self.web_class}")
-
+
             if self.flange_class == 'Slender' or self.web_class == 'Slender':
                 self.section_class = 'Slender'
             else:
```

Figure 4.9: From Column.py

```diff
src/osdag/design_type/compression_member/Column.py
@@ -775,11 +787,8 @@ def section_classification(self):
                 elif self.flange_class == 'Semi-Compact' and self.web_class == 'Semi-Compact':
                     self.section_class = 'Semi-Compact'

-        logger.info("The flange of the trial section ({}) is {} and web is {}. The section is {} [Reference: Cl 3.7, IS 800:2007].".
-                    format(trial_section, self.flange_class, self.web_class, self.section_class))
-
             # 2.2 - Effective length
-        self.effective_length_zz = IS800_2007.cl_7_2_2_effective_length_of_prismatic_compression_members(
+            self.effective_length_zz = IS800_2007.cl_7_2_2_effective_length_of_prismatic_compression_members(
                 self.length_zz,
                 end_1=self.end_1_z,
                 end_2=self.end_2_z)
@@ -789,7 +798,7 @@ def section_classification(self):
             #     self.sec_profile) / self.length_yy  # mm
             # print(f"self.effective_length {self.effective_length_yy} ")

-        self.effective_length_yy = IS800_2007.cl_7_2_2_effective_length_of_prismatic_compression_members(
+            self.effective_length_yy = IS800_2007.cl_7_2_2_effective_length_of_prismatic_compression_members(
                 self.length_yy,
                 end_1=self.end_1_y,
                 end_2=self.end_2_y)
@@ -810,16 +819,17 @@ def section_classification(self):
             # print("++++++++++++++++++++++++++++++++++++++++++++++++")

             # 2.3 - Effective slenderness ratio
-        self.effective_sr_zz = self.effective_length_zz / self.section_property.rad_of_gy_z
-        self.effective_sr_yy = self.effective_length_yy / self.section_property.rad_of_gy_y
+            self.effective_sr_zz = self.effective_length_zz / self.section_property.rad_of_gy_z
+            self.effective_sr_yy = self.effective_length_yy / self.section_property.rad_of_gy_y

-        limit = IS800_2007.cl_3_8_max_slenderness_ratio(1)
-        if self.effective_sr_zz > limit and self.effective_sr_yy > limit:
+            limit = IS800_2007.cl_3_8_max_slenderness_ratio(1)
+            if self.effective_sr_zz > limit and self.effective_sr_yy > limit:
                 logger.warning("Length provided is beyond the limit allowed. [Reference: Cl 3.8, IS 800:2007]")
                 logger.error("Cannot compute. Given Length does not pass.")
                 local_flag = False
-        else:
-            logger.info("Length provided is within the limit allowed. [Reference: Cl 3.8, IS 800:2007]")
+            #else:
+            #    logger.info("Length provided is within the limit allowed. [Reference: Cl 3.8, IS 800:2007]")
+

             # if len(self.allowed_sections) == 0:
             #     logger.warning("Select at-least one type of section in the design preferences tab.")
@@ -828,13 +838,11 @@ def section_classification(self):
             #     self.design_status_list.append(self.design_status)

             #TODO: @danish check this part
-            # if self.section_class in self.allowed_sections:
-            #     self.input_section_list.append(trial_section)
```

Figure 4.10: From Column.py

```diff
src/osdag/design_type/compression_member/Column.py
+            #    logger.info("Length provided is within the limit allowed. [Reference: Cl 3.8, IS 800:2007]")
+

             # if len(self.allowed_sections) == 0:
             #     logger.warning("Select at-least one type of section in the design preferences tab.")
@@ -828,13 +838,11 @@ def section_classification(self):
             #     self.design_status_list.append(self.design_status)

             #TODO: @danish check this part
-            # if self.section_class in self.allowed_sections:
-            #     self.input_section_list.append(trial_section)
-            #     self.input_section_classification.update({trial_section: self.section_class})
+            if self.section_class in self.allowed_sections:
+                self.input_section_list.append(trial_section)
+                self.input_section_classification.update({trial_section: [self.section_class, self.flange_class, self.web_class, flange_ratio, web_ratio]})
             # print(f"self.section_class{self.section_class}")

-        self.input_section_list.append(trial_section)
-        self.input_section_classification.update({trial_section: [self.section_class, self.flange_class, self.web_class, flange_ratio, web_ratio]})

         return local_flag

@@ -865,19 +873,33 @@ def design_column(self):
         #     self.design_status = False
         #     self.design_status_list.append(self.design_status)
         self.epsilon = math.sqrt(250 / self.material_property.fy)
-        if len(self.input_section_list) > 0:
+        #if len(self.input_section_list) > 0:
+
+        # initializing lists to store the optimum results based on optimum UR and cost
+
+        # 1- Based on optimum UR
+        self.optimum_section_ur_results = {}
+        self.optimum_section_ur = []

-            # initializing lists to store the optimum results based on optimum UR and cost
+        # 2 - Based on optimum cost
+        self.optimum_section_cost_results = {}
+        self.optimum_section_cost = []
+        self.flag = self.section_classification(self)

-            # 1- Based on optimum UR
-            self.optimum_section_ur_results = {}
-            self.optimum_section_ur = []
+        print('self.flag:',self.flag)
+        # reduction of the area based on the connection requirements (input from design preferences)
+        if self.effective_area_factor < 1.0:
+            self.effective_area = round(self.effective_area * self.effective_area_factor, 2)

-            # 2 - Based on optimum cost
-            self.optimum_section_cost_results = {}
-            self.optimum_section_cost = []
```

Figure 4.11: From Column.py

```
src/osdag/design_type/compression_member/Column.py
-            # 2 - based on optimum cost
-            self.optimum_section_cost_results = {}
-            self.optimum_section_cost = []
+            #logger.warning("Reducing the effective sectional area as per the definition in the Design Preferences tab.")
+            #logger.info("The actual effective area is {} mm2 and the reduced effective area is {} mm2 [Reference: Cl. 7.3.2, IS 800:2007]".
+            #             format(round((self.effective_area / self.effective_area_factor), 2), self.effective_area))
+        #else:
+            #if self.section_class != 'Slender':
+            #    logger.info("The effective sectional area is taken as 100% of the cross-sectional area [Reference: Cl. 7.3.2, IS 800:2007].")
 
-            i = 1
+        #print('self.input_section_list:',self.input_section_list)
+        if self.flag:
             for section in self.input_section_list:  # iterating the design over each section to find the most optimum section
 
                 # fetching the section properties of the selected section
@@ -900,7 +922,7 @@ def design_column(self):
                     self.section_property = Column(designation=section, material_grade=self.material)
 
                 self.material_property.connect_to_database_to_get_fy_fu(self.material, max(self.section_property.flange_thickness,
-                                                                                            self.section_property.web_thickness))
+                                                                                         self.section_property.web_thickness))
 
 
 
@@ -912,33 +934,22 @@ def design_column(self):
                 self.list_yy.append(section)
 
                 # Step 1 - computing the effective sectional area
-                self.section_class = self.input_section_classification[section]
+                self.section_class = self.input_section_classification[section][0]
 
                 if self.section_class == 'Slender':
-                    logger.warning("The trial section ({}) is Slender. Computing the Effective Sectional Area as per Sec. 9.7.2, "
-                                   "Fig. 2 (B & C) of The National Building Code of India (NBC), 2016.".format(section))
+                    #logger.warning("The trial section ({}) is Slender. Computing the Effective Sectional Area as per Sec. 9.7.2, "
+                    #               "Fig. 2 (B & C) of The National Building Code of India (NBC), 2016.".format(section))
 
                     if (self.sec_profile == VALUES_SEC_PROFILE[0]):  # Beams and Columns
                         self.effective_area = (2 * ((31.4 * self.epsilon * self.section_property.flange_thickness) *
                                                     self.section_property.flange_thickness)) + \
-                                              (2 * ((21 * self.epsilon * self.section_property.web_thickness) * self.section_property.web_thickness))
+                                             (2 * ((21 * self.epsilon * self.section_property.web_thickness) * self.section_property.web_thickness))
                     elif (self.sec_profile == VALUES_SEC_PROFILE[1]):
                         self.effective_area = (2 * 21 * self.epsilon * self.section_property.flange_thickness) * 2
                 else:
                     self.effective_area = self.section_property.area  # mm2
                     # print(f"self.effective_area{self.effective_area}")
 
-                # reduction of the area based on the connection requirements (input from design preferences)
-                if self.effective_area_factor < 1.0:
-                    self.effective_area = round(self.effective_area * self.effective_area_factor, 2)
-
-                    logger.warning("Reducing the effective sectional area as per the definition in the Design Preferences tab.")
```

Figure 4.12: From Column.py

```diff
src/osdag/design_type/compression_member/Column.py
                 # initialize lists for updating the results dictionary
-                list_zz = []
-                list_yy = []
+                self.list_zz = []
+                self.list_yy = []
 
-                list_zz.append(section)
-                list_yy.append(section)
+                self.list_zz.append(section)
+                self.list_yy.append(section)
 
                 # Step 1 - computing the effective sectional area
                 self.section_class = self.input_section_classification[section]
@@ -928,11 +939,11 @@ def design_column(self):
                     if self.section_class != 'Slender':
                         logger.info("The effective sectional area is taken as 100% of the cross-sectional area [Reference: Cl. 7.3.2, IS 800:2007].")
 
-                list_zz.append(self.section_class)
-                list_yy.append(self.section_class)
+                self.list_zz.append(self.section_class)
+                self.list_yy.append(self.section_class)
 
-                list_zz.append(self.effective_area)
-                list_yy.append(self.effective_area)
+                self.list_zz.append(self.effective_area)
+                self.list_yy.append(self.effective_area)
 
                 # Step 2 - computing the design compressive stress
 
@@ -968,11 +979,11 @@ def design_column(self):
                 self.imperfection_factor_zz = IS800_2007.cl_7_1_2_1_imperfection_factor(buckling_class=self.buckling_class_zz)
                 self.imperfection_factor_yy = IS800_2007.cl_7_1_2_1_imperfection_factor(buckling_class=self.buckling_class_yy)
 
-                list_zz.append(self.buckling_class_zz)
-                list_yy.append(self.buckling_class_yy)
+                self.list_zz.append(self.buckling_class_zz)
+                self.list_yy.append(self.buckling_class_yy)
 
-                list_zz.append(self.imperfection_factor_zz)
-                list_yy.append(self.imperfection_factor_yy)
+                self.list_zz.append(self.imperfection_factor_zz)
+                self.list_yy.append(self.imperfection_factor_yy)
 
                 # 2.2 - Effective length
                 self.effective_length_zz = IS800_2007.cl_7_2_2_effective_length_of_prismatic_compression_members(self.length_zz ,
@@ -982,45 +993,45 @@ def design_column(self):
                                                                                                                  end_1=self.end_1_y,
                                                                                                                  end_2=self.end_2_y)  # mm
 
-                list_zz.append(self.effective_length_zz)
-                list_yy.append(self.effective_length_yy)
```

Figure 4.13: From Column.py

```
src/osdag/design_type/compression_member/Column.py
+                self.list_zz.append(self.f_cd_1_zz)
+                self.list_yy.append(self.f_cd_1_yy)

-                list_zz.append(self.f_cd_2)
-                list_yy.append(self.f_cd_2)
+                self.list_zz.append(self.f_cd_2)
+                self.list_yy.append(self.f_cd_2)

-                list_zz.append(self.f_cd_zz)
-                list_yy.append(self.f_cd_yy)
+                self.list_zz.append(self.f_cd_zz)
+                self.list_yy.append(self.f_cd_yy)

-                list_zz.append(self.f_cd)
-                list_yy.append(self.f_cd)
+                self.list_zz.append(self.f_cd)
+                self.list_yy.append(self.f_cd)

                 # 2.7 - Capacity of the section

                 self.section_capacity = self.f_cd * self.effective_area  # N

-                list_zz.append(self.section_capacity)
-                list_yy.append(self.section_capacity)
+                self.list_zz.append(self.section_capacity)
+                self.list_yy.append(self.section_capacity)

                 # 2.8 - UR
                 self.ur = round(self.load.axial_force / self.section_capacity, 3)

-                list_zz.append(self.ur)
-                list_yy.append(self.ur)
+                self.list_zz.append(self.ur)
+                self.list_yy.append(self.ur)
                 self.optimum_section_ur.append(self.ur)

                 # 2.9 - Cost of the section in INR
                 self.cost = (self.section_property.unit_mass * self.section_property.area * 1e-4) * min(self.length_zz, self.length_yy) *
                             self.steel_cost_per_kg

-                list_zz.append(self.cost)
-                list_yy.append(self.cost)
+                self.list_zz.append(self.cost)
+                self.list_yy.append(self.cost)
                 self.optimum_section_cost.append(self.cost)
                 # print(f"list_zz{list_zz},list_yy{list_yy} ")

@@ -1076,7 +1087,7 @@ def design_column(self):
                 # 1- Based on optimum UR
                 self.optimum_section_ur_results[self.ur] = {}

-                list_2 = list_zz + list_yy
+                list_2 = self.list_zz + self.list_yy
                 for i in list_1:
```

Figure 4.14: From Column.py

```
src/osdag/design_type/compression_member/Column.py
                            break
-        else:
-            logger.warning("The section(s) defined for performing the column design is/are not selected based on the selected Inputs and/or "
-                           "Design Preferences")
-            logger.error("Cannot compute!")
-            logger.info("Change the inputs provided and re-design.")
-            self.design_status = False
-            self.design_status_list.append(self.design_status)
+        #else:
+        #    logger.warning("The section(s) defined for performing the column design is/are not selected based on the selected Inputs and/or "
+        #                   "Design Preferences")
+        #    logger.error("Cannot compute!")
+        #    logger.info("Change the inputs provided and re-design.")
+        #    self.design_status = False
+        #    self.design_status_list.append(self.design_status)
            # print(f"design_status_list{self.design_status_list}")

    def results(self):
        """ """
-        # sorting results from the dataset
-
+        _ = [i for i in self.optimum_section_ur if i > 1.0]
+        print( '_ ',_)
+        if len(_)==1:
+            temp = _[0]
+        elif len(_)==0:
+            temp = None
+        else:
+            temp = sorted(_)[0]
+        self.failed_design_dict = self.optimum_section_ur_results[temp] if temp is not None else None
+        print('self.failed_design_dict ',self.failed_design_dict)
+
        # results based on UR
        if self.optimization_parameter == 'Utilization Ratio':
            filter_UR = filter(lambda x: x <= min(self.allowable_utilization_ratio, 1.0), self.optimum_section_ur)
            self.optimum_section_ur = list(filter_UR)

            self.optimum_section_ur.sort()
-            # print(f"self.optimum_section_ur{self.optimum_section_ur}")
-            #print(f"self.result_UR{self.result_UR}")
+            print(f"self.optimum_section_ur{self.optimum_section_ur} \n self.optimum_section_ur_results{self.optimum_section_ur_results}")
+
+            # print(f"self.result_UR{self.result_UR}")

            # selecting the section with most optimum UR
            if len(self.optimum_section_ur) == 0:  # no design was successful
@@ -1134,138 +1155,166 @@ def results(self):
                logger.error("The solver did not find any adequate section from the defined list.")
                logger.info("Re-define the list of sections or check the Design Preferences option and re-design.")
                self.design_status = False
-                self.design_status_list.append(self.design_status)
+                if len(self.failed_design_dict)>0:
+                    logger.info(
```

Figure 4.15: From Column.py

src/osdag/design_type/compression_member/Column.py

```diff
-            #print(f"self.result_UR{self.result_UR}")
+            print(f"self.optimum_section_ur{self.optimum_section_ur} \n self.optimum_section_ur_results{self.optimum_section_ur_results}")
+
+            # print(f"self.result_UR{self.result_UR}")

             # selecting the section with most optimum UR
             if len(self.optimum_section_ur) == 0:  # no design was successful
@@ -1134,138 +1155,166 @@ def results(self):
                 logger.error("The solver did not find any adequate section from the defined list.")
                 logger.info("Re-define the list of sections or check the Design Preferences option and re-design.")
                 self.design_status = False
-                self.design_status_list.append(self.design_status)
+                if len(self.failed_design_dict)>0:
+                    logger.info(
+                    "The details for the best section provided is being shown"
+                )
+                    self.result_UR = self.failed_design_dict['UR'] #temp
+                    self.common_result(
+                        self,
+                        list_result=self.failed_design_dict,
+                        result_type=None,
+                    )
+                    logger.warning(
+                    "Re-define the list of sections or check the Design Preferences option and re-design."
+                )
+                    return
+                #self.design_status_list.append(self.design_status)

             else:
-                self.result_UR = self.optimum_section_ur[-1]  # optimum section which passes the UR check
+                self.failed_design_dict = None
+                self.result_UR = self.optimum_section_ur[
+                    -1
+                ]  # optimum section which passes the UR check
                 print(f"self.result_UR{self.result_UR}")
                 self.design_status = True
+                self.common_result(
+                    self,
+                    list_result=self.optimum_section_ur_results,
+                    result_type=self.result_UR,
+                )

         else:  # results based on cost
             self.optimum_section_cost.sort()

             # selecting the section with most optimum cost
             self.result_cost = self.optimum_section_cost[0]
-
+            self.design_status = True
         # print results
-        if len(self.optimum_section_ur) == 0:
-            logger.warning(
-                "The sections selected by the solver from the defined list of sections did not satisfy the Utilization Ratio (UR) "
                 "criteria")
```

Figure 4.16: From Column.py

```diff
+
+    def common_result(self, list_result, result_type):
+
+        self.result_designation = list_result[result_type]['Designation']
+        self.section_class = self.input_section_classification[self.result_designation][0]
+
+        if self.section_class == 'Slender':
+            logger.warning("The trial section ({}) is Slender. Computing the Effective Sectional Area as per Sec. 9.7.2, "
+                           "Fig. 2 (B & C) of The National Building Code of India (NBC), 2016.".format(self.result_designation))
+        if self.effective_area_factor < 1.0:
+            self.effective_area = round(self.effective_area * self.effective_area_factor, 2)
+
+            logger.warning("Reducing the effective sectional area as per the definition in the Design Preferences tab.")
+            logger.info("The actual effective area is {} mm2 and the reduced effective area is {} mm2 [Reference: Cl. 7.3.2, IS 800:2007]".
+                           format(round((self.effective_area / self.effective_area_factor), 2), self.effective_area))
```

Figure 4.17: From Column.py

```
src/osdag/design_type/compression_member/Column.py

+            if self.section_class != 'Slender':
+                logger.info("The effective sectional area is taken as 100% of the cross-sectional area [Reference: Cl. 7.3.2, IS 800:2007].")
+        logger.info(
+            "The section is {}. The {} section  has  {} flange({}) and  {} web({}).  [Reference: Cl 3.7, IS 800:2007].".format(
+                self.input_section_classification[self.result_designation][0] ,
+                self.result_designation,
+                self.input_section_classification[self.result_designation][1], round(self.input_section_classification[self.result_designation][3],2),
+                self.input_section_classification[self.result_designation][2], round(self.input_section_classification[self.result_designation][4],2)
+            ))

+        self.result_section_class = list_result[result_type]['Section class']
+        self.result_effective_area = list_result[result_type]['Effective area']
+
+        self.result_bc_zz = list_result[result_type]['Buckling_curve_zz']
+        self.result_bc_yy = list_result[result_type]['Buckling_curve_yy']
+
+        self.result_IF_zz = list_result[result_type]['IF_zz']
+        self.result_IF_yy = list_result[result_type]['IF_yy']
+
+        self.result_eff_len_zz = list_result[result_type]['Effective_length_zz']
+        self.result_eff_len_yy = list_result[result_type]['Effective_length_yy']
+
+        self.result_eff_sr_zz = list_result[result_type]['Effective_SR_zz']
+        self.result_eff_sr_yy = list_result[result_type]['Effective_SR_yy']
+
+        self.result_ebs_zz = list_result[result_type]['EBS_zz']
+        self.result_ebs_yy = list_result[result_type]['EBS_yy']
+
+        self.result_nd_esr_zz = list_result[result_type]['ND_ESR_zz']
+        self.result_nd_esr_yy = list_result[result_type]['ND_ESR_yy']
+
+        self.result_phi_zz = list_result[result_type]['phi_zz']
+        self.result_phi_yy = list_result[result_type]['phi_yy']
+
+        self.result_srf_zz = list_result[result_type]['SRF_zz']
+        self.result_srf_yy = list_result[result_type]['SRF_yy']
+
+        self.result_fcd_1_zz = list_result[result_type]['FCD_1_zz']
+        self.result_fcd_1_yy = list_result[result_type]['FCD_1_yy']
+        self.result_fcd_2 = list_result[result_type]['FCD_2']
+        self.result_fcd_zz = list_result[result_type]['FCD_zz']
+        self.result_fcd_yy = list_result[result_type]['FCD_yy']
+        self.result_fcd = list_result[result_type]['FCD']
+
+        self.result_capacity = list_result[result_type]['Capacity']
+        self.result_cost = list_result[result_type]['Cost']
+
+
+    def save_design(self, popup_summary):
+
+        if (self.design_status and self.failed_design_dict is None) or (not self.design_status and len(self.failed_design_dict)>0):
+            if self.sec_profile=='Columns' or self.sec_profile=='Beams' or self.sec_profile == VALUES_SEC_PROFILE[0]:
+                self.report_column = {KEY_DISP_SEC_PROFILE: "ISection",
```

Figure 4.18: From Column.py

```
src/osdag/design_type/compression_member/Column.py
+    def save_design(self, popup_summary):
+
+        if (self.design_status and self.failed_design_dict is None) or (not self.design_status and len(self.failed_design_dict)>0):
+            if self.sec_profile=='Columns' or self.sec_profile=='Beams' or self.sec_profile == VALUES_SEC_PROFILE[0]:
+                self.report_column = {KEY_DISP_SEC_PROFILE: "ISection",
+                                      KEY_DISP_SECSIZE: (self.section_property.designation, self.sec_profile),
+                                      KEY_DISP_COLSEC_REPORT: self.section_property.designation,
+                                      KEY_DISP_MATERIAL: self.section_property.material,
+    #                                   KEY_DISP_APPLIED_AXIAL_FORCE: self.section_property.,
+                                      KEY_REPORT_MASS: self.section_property.mass,
+                                      KEY_REPORT_AREA: round(self.section_property.area * 1e-2, 2),
+                                      KEY_REPORT_DEPTH: self.section_property.depth,
+                                      KEY_REPORT_WIDTH: self.section_property.flange_width,
+                                      KEY_REPORT_WEB_THK: self.section_property.web_thickness,
+                                      KEY_REPORT_FLANGE_THK: self.section_property.flange_thickness,
+                                      KEY_DISP_FLANGE_S_REPORT: self.section_property.flange_slope,
+                                      KEY_REPORT_R1: self.section_property.root_radius,
+                                      KEY_REPORT_R2: self.section_property.toe_radius,
+                                      KEY_REPORT_IZ: round(self.section_property.mom_inertia_z * 1e-4, 2),
+                                      KEY_REPORT_IY: round(self.section_property.mom_inertia_y * 1e-4, 2),
+                                      KEY_REPORT_RZ: round(self.section_property.rad_of_gy_z * 1e-1, 2),
+                                      KEY_REPORT_RY: round(self.section_property.rad_of_gy_y * 1e-1, 2),
+                                      KEY_REPORT_ZEZ: round(self.section_property.elast_sec_mod_z * 1e-3, 2),
+                                      KEY_REPORT_ZEY: round(self.section_property.elast_sec_mod_y * 1e-3, 2),
+                                      KEY_REPORT_ZPZ: round(self.section_property.plast_sec_mod_z * 1e-3, 2),
+                                      KEY_REPORT_ZPY: round(self.section_property.plast_sec_mod_y * 1e-3, 2)}
+            else:
+                #Update for section profiles RHS and SHS, CHS by making suitable elif condition.
+                self.report_column = {KEY_DISP_COLSEC_REPORT: self.section_property.designation,
+                                      KEY_DISP_MATERIAL: self.section_property.material,
+                                      #                                  KEY_DISP_APPLIED_AXIAL_FORCE: self.section_property.,
+                                      KEY_REPORT_MASS: self.section_property.mass,
+                                      KEY_REPORT_AREA: round(self.section_property.area * 1e-2, 2),
+                                      KEY_REPORT_DEPTH: self.section_property.depth,
+                                      KEY_REPORT_WIDTH: self.section_property.flange_width,
+                                      KEY_REPORT_WEB_THK: self.section_property.web_thickness,
+                                      KEY_REPORT_FLANGE_THK: self.section_property.flange_thickness,
+                                      KEY_DISP_FLANGE_S_REPORT: self.section_property.flange_slope}
+
+
+            self.report_input = \
+                {#KEY_MAIN_MODULE: self.mainmodule,
+                KEY_MODULE: self.module, #"Axial load on column "
+                    KEY_DISP_AXIAL: self.load.axial_force * 10 ** -3,
+                    KEY_DISP_ACTUAL_LEN_ZZ: self.length_zz,
+                    KEY_DISP_ACTUAL_LEN_YY: self.length_yy,
+                    KEY_DISP_SEC_PROFILE: self.sec_profile,
+                    KEY_DISP_SECSIZE: self.result_section_class,
+                    KEY_DISP_END1: self.end_1_z,
+                    KEY_DISP_END2: self.end_2_z,
+                    KEY_DISP_END1_Y: self.end_1_y,
+                    KEY_DISP_END2_Y: self.end_2_y,
+                    "Column Section - Mechanical Properties": "TITLE"
```

Figure 4.19: From Column.py

```
src/osdag/design_type/compression_member/Column.py                                    +491 -349

        self.report_input = \
            {#KEY_MAIN_MODULE: self.mainmodule,
            KEY_MODULE: self.module, #"Axial load on column "
                KEY_DISP_AXIAL: self.load.axial_force * 10 ** -3,
                KEY_DISP_ACTUAL_LEN_ZZ: self.length_zz,
                KEY_DISP_ACTUAL_LEN_YY: self.length_yy,
                KEY_DISP_SEC_PROFILE: self.sec_profile,
                KEY_DISP_SECSIZE: self.result_section_class,
                KEY_DISP_END1: self.end_1_z,
                KEY_DISP_END2: self.end_2_z,
                KEY_DISP_END1_Y: self.end_1_y,
                KEY_DISP_END2_Y: self.end_2_y,
                "Column Section - Mechanical Properties": "TITLE",
            KEY_MATERIAL: self.material,
                KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.material_property.fu,
                KEY_DISP_YIELD_STRENGTH_REPORT: self.material_property.fy,
                KEY_DISP_EFFECTIVE_AREA_PARA: self.effective_area_factor, #To Check
                KEY_DISP_SECSIZE:  str(self.sec_list),
                "Selected Section Details": self.report_column,
            }

        self.report_check = []
        t1 = ('Selected', 'Selected Member Data', '|p{5cm}|p{2cm}|p{2cm}|p{2cm}|p{4cm}|')
        self.report_check.append(t1)

        self.h = (self.section_property.depth - 2 * (self.section_property.flange_thickness + self.section_property.root_radius))
        self.h_bf_ratio = self.h / self.section_property.flange_width


        # 2.2 CHECK: Buckling Class - Compatibility Check
        t1 = ('SubSection', 'Buckling Class - Compatibility Check', '|p{4cm}|p{3.5cm}|p{6.5cm}|p{2cm}|')
        self.report_check.append(t1)

        # YY axis row
        t1 = (
            "h/bf and tf for YY Axis",
            comp_column_class_section_check_required(self.h, self.section_property.flange_width, self.section_property.flange_thickness, "YY"),
            comp_column_class_section_check_provided(self.h, self.section_property.flange_width, self.section_property.flange_thickness, round(self.h_bf_ratio, 2), "YY"),
'Compatible'
        )
        self.report_check.append(t1)

        # ZZ axis row
        t1 = (
            "h/bf and tf for ZZ Axis",
            comp_column_class_section_check_required(self.h, self.section_property.flange_width, self.section_property.flange_thickness, "ZZ"),
            comp_column_class_section_check_provided(self.h, self.section_property.flange_width, self.section_property.flange_thickness, round(self.h_bf_ratio, 2), "ZZ"),
'Compatible'
        )
        self.report_check.append(t1)

```

Figure 4.20: From Column.py

```
src/osdag/design_type/compression_member/Column.py

        t1 = ('SubSection', 'Section Classification', '|p{3cm}|p{3.5cm}|p{8.5cm}|p{1cm}|')
        self.report_check.append(t1)
        t1 = ('Web Class', 'Axial Compression',
                cl_3_7_2_section_classification_web(round(self.h, 2), round(self.section_property.web_thickness, 2),
round(self.input_section_classification[self.result_designation][4], 2),
                                                   self.epsilon, self.section_property.type,
                                                   self.input_section_classification[self.result_designation][2]),
                ' ')
        self.report_check.append(t1)
        t1 = ('Flange Class', self.section_property.type,
                cl_3_7_2_section_classification_flange(round(self.section_property.flange_width/2, 2),
                                                       round(self.section_property.flange_thickness, 2),
                            round(self.input_section_classification[self.result_designation][3], 2),
                                                       self.epsilon,
                                                       self.input_section_classification[self.result_designation][1]),
                ' ')
        self.report_check.append(t1)
        t1 = ('Section Class', ' ',
                cl_3_7_2_section_classification(
                                                       self.input_section_classification[self.result_designation][0]),
                ' ')
        self.report_check.append(t1)


        t1 = ('NewTable', 'Imperfection Factor', '|p{3cm}|p{5 cm}|p{5cm}|p{3 cm}|')
        self.report_check.append(t1)

        t1 = (
            'YY',
            self.list_yy[3].upper(),
            self.list_yy[4], ''
        )
        self.report_check.append(t1)

        t1 = (
            'ZZ',
            self.list_zz[3].upper(),
            self.list_zz[4], ''
        )
        self.report_check.append(t1)


        K_yy = self.result_eff_len_yy / self.length_yy
        K_zz= self.result_eff_len_zz / self.length_zz
        t1 = ('SubSection', 'Slenderness Ratio', '|p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
        self.report_check.append(t1)
        t1 = ("Effective Slenderness Ratio (For YY Axis)", ' ',
                            cl_7_1_2_effective_slenderness_ratio(K_yy,self.length_yy, self.section_property.rad_of_gy_y, round(self.result_eff_sr_yy, 2)),
                            ' ')
        self.report_check.append(t1)
        t1 = ("Effective Slenderness Ratio (For ZZ Axis)", ' ',
                            cl_7_1_2_effective_slenderness_ratio(K_zz,self.length_zz, self.section_property.rad_of_gy_z, round(self.result_eff_sr_zz, 2)),
```

Figure 4.21: From Column.py

```diff
src/osdag/design_type/compression_member/Column.py    +491 -349
+            t1 = (
+                'ZZ',
+                self.list_zz[3].upper(),
+                self.list_zz[4], ''
+            )
+            self.report_check.append(t1)
+
+
+            K_yy = self.result_eff_len_yy / self.length_yy
+            K_zz= self.result_eff_len_zz / self.length_zz
+            t1 = ('SubSection', 'Slenderness Ratio', '|p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
+            self.report_check.append(t1)
+            t1 = ("Effective Slenderness Ratio (For YY Axis)", ' ',
+                            cl_7_1_2_effective_slenderness_ratio(K_yy,self.length_yy, self.section_property.rad_of_gy_y, round(self.result_eff_sr_yy, 2)),
+                            ' ')
+            self.report_check.append(t1)
+            t1 = ("Effective Slenderness Ratio (For ZZ Axis)", ' ',
+                            cl_7_1_2_effective_slenderness_ratio(K_zz,self.length_zz, self.section_property.rad_of_gy_z, round(self.result_eff_sr_zz, 2)),
+                            ' ')
+            self.report_check.append(t1)
+
+
+
+            t1 = ('SubSection', 'Checks', '|p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
+            self.report_check.append(t1)
+
+            t1 = (r'$\phi_{yy}$', ' ',
+                cl_8_7_1_5_phi(self.result_IF_yy, round(self.non_dim_eff_sr_yy, 2), round(self.result_phi_yy, 2)),
+                ' ')
+            self.report_check.append(t1)
+
+            t1 = (r'$\phi_{zz}$', ' ',
+                cl_8_7_1_5_phi(self.result_IF_zz, round(self.non_dim_eff_sr_zz, 2), round(self.result_phi_zz, 2)),
+                ' ')
+            self.report_check.append(t1)
+
+            t1 = (r'$F_{cd,yy} \, \left( \frac{N}{\text{mm}^2} \right)$', ' ',
+                cl_8_7_1_5_Buckling(self.material_property.fy, self.gamma_m0, round(self.non_dim_eff_sr_yy, 2), round(self.result_phi_yy, 2), round(self.result_fcd_2, 2), round(self.result_fcd_yy, 2)),
+                ' ')
+            self.report_check.append(t1)
+
+            t1 = (r'$F_{cd,zz} \, \left( \frac{N}{\text{mm}^2} \right)$', ' ',
+                cl_8_7_1_5_Buckling(self.material_property.fy, self.gamma_m0, round(self.non_dim_eff_sr_zz, 2), round(self.result_phi_zz, 2), round(self.result_fcd_2, 2), round(self.result_fcd_zz, 2)),
+                ' ')
+            self.report_check.append(t1)
+
+            t1 = (r'Design Compressive Strength (\( P_d \)) (For the most critical value of \( F_{cd} \))', self.load.axial_force * 10 ** -3,
+                        cl_7_1_2_design_compressive_strength(round(self.result_capacity / 1000, 2), self.section_property.area, round(self.result_fcd, 2),self.load.axial_force * 10 ** -3),
+                        get_pass_fail(self.load.axial_force * 10 ** -3, round(self.result_capacity, 2), relation="leq"))
+            self.report_check.append(t1)
```

Figure 4.22: From Column.py

```diff
+        else:
+            self.report_input = \
+                {#KEY_MAIN_MODULE: self.mainmodule,
+                KEY_MODULE: self.module, #"Axial load on column "
+                    KEY_DISP_AXIAL: self.load.axial_force * 10 ** -3,
+                    KEY_DISP_ACTUAL_LEN_ZZ: self.length_zz,
+                    KEY_DISP_ACTUAL_LEN_YY: self.length_yy,
+                    KEY_DISP_SEC_PROFILE: self.sec_profile,
+                    KEY_DISP_SECSIZE:  str(self.sec_list),
+                    #KEY_DISP_SECSIZE: self.result_section_class,
+                    KEY_DISP_END1: self.end_1_z,
+                    KEY_DISP_END2: self.end_2_z,
+                    KEY_DISP_END1_Y: self.end_1_y,
+                    KEY_DISP_END2_Y: self.end_2_y,
+                    "Column Section - Mechanical Properties": "TITLE",
+                KEY_MATERIAL: self.material,
+                    KEY_DISP_ULTIMATE_STRENGTH_REPORT: self.material_property.fu,
+                    KEY_DISP_YIELD_STRENGTH_REPORT: self.material_property.fy,
+                    KEY_DISP_EFFECTIVE_AREA_PARA: self.effective_area_factor, #To Check
+
+                    # "Failed Section Details": self.report_column,
+                }
+            self.report_check = []
+
+            t1 = ('Selected', 'All Members Failed', '|p{5cm}|p{2cm}|p{2cm}|p{2cm}|p{4cm}|')
+            self.report_check.append(t1)


        Disp_2d_image = []
@@ -1481,4 +1622,5 @@ def save_design(self, popup_summary):
        rel_path = rel_path.replace("\\", "/")
        fname_no_ext = popup_summary['filename']
        CreateLatex.save_latex(CreateLatex(), self.report_input, self.report_check, popup_summary, fname_no_ext,
-                              rel_path, Disp_2d_image, Disp_3D_image, module=self.module)
+                              rel_path, Disp_2d_image, Disp_3D_image, module=self.module)
+
```

Figure 4.23: From Column.py

```
+
+        Disp_2d_image = []
+        Disp_3D_image = "/ResourceFiles/images/3d.png"
+
         print(sys.path[0])
         rel_path = str(sys.path[0])
         rel_path = os.path.abspath(".") # TEMP
         rel_path = rel_path.replace("\\", "/")
         fname_no_ext = popup_summary['filename']
         CreateLatex.save_latex(CreateLatex(), self.report_input, self.report_check, popup_summary, fname_no_ext,
-                               rel_path, module=self.module)
-
-
-
+                               rel_path, Disp_2d_image, Disp_3D_image, module=self.module)
```

Figure 4.24: From Column.py

4.3.4 compression.py

In the `compression.py`, `section_classification` and `save_design` functions were modified, a failure dictionary was added as well to fall back to for report generation when design_status fails. The last 4 images were integrated as well similar to `compression.py`.

```diff
src/osdag/design_type/compression_member/compression.py
@@ -11,6 +11,12 @@
 from ...utils.common import is800_2007
 from ...utils.common.component import *
 
+import logging
+from ..connection.moment_connection import MomentConnection
+from ...utils.common.material import *
+from ...Report_functions import *
+from ...design_report.reportGenerator_latex import CreateLatex
+from pylatex.utils import NoEscape
 
 
 class Compression(Member):
@@ -249,7 +255,7 @@ def get_values_for_design_pref(self, key, design_dictionary):
     ####################################
 
     def module_name(self):
-        return KEY_DISP_COMPRESSION_STRUT
+        return KEY_DISP_COMPRESSION_Strut
 
     def set_osdaglogger(key):
 
@@ -312,7 +318,7 @@ def input_values(self):
         '''
 
         # @author: Amir, Umair
-
+        self.module = KEY_DISP_COMPRESSION_Strut
         options_list = []
 
         t1 = (KEY_MODULE, KEY_DISP_COMPRESSION_Strut, TYPE_MODULE, None, True, 'No Validator')
@@ -840,6 +846,16 @@ def func_for_validation(self, design_dictionary):
             return all_errors
         print(f"func_for_validation done")
 
+
+    def get_3d_components(self):
+
+        components = []
+
+        t1 = ('Model', self.call_3DModel)
+        components.append(t1)
+
+        return components
+
     def fn_conn_type(self):
 
         "Function to populate section size based on the type of section "
@@ -856,6 +872,7 @@ def set_input_values(self, design_dictionary):
```

Figure 4.25: From compression.py

```diff
src/osdag/design_type/compression_member/compression.py
@@ -840,9 +839,21 @@ def func_for_validation(self, design_dictionary):
             flag = True

         #print(f'flag = {flag}')
-        if flag and flag1 and flag2:
+        if flag:
+            print(f"\n design_dictionary{design_dictionary}")
             self.set_input_values(self, design_dictionary)
-            # print(design_dictionary)
+            if self.design_status ==False and len(self.failed_design_dict)>0:
+                logger.error(
+                    "Design Failed, Check Design Report"
+                )
+                return # ['Design Failed, Check Design Report'] @TODO
+            elif self.design_status:
+                pass
+            else:
+                logger.error(
+                    "Design Failed. Slender Sections Selected"
+                )
+                return # ['Design Failed. Slender Sections Selected']
         else:
             return all_errors
         print(f"func_for_validation done")
@@ -872,7 +883,6 @@ def set_input_values(self, design_dictionary):
         super(Compression,self).set_input_values(self, design_dictionary)
         #self.sizelist == self.sec_list
         # section properties
-        self.mainmodule = 'Struts in Trusses'
         self.module = design_dictionary[KEY_MODULE]
         self.mainmodule = 'Struts in Trusses'
         self.sizelist = design_dictionary[KEY_SECSIZE]
@@ -1005,6 +1015,7 @@ def set_input_values(self, design_dictionary):
         print("K = {}.\n The input values are set. Performing preliminary member check(s).".format(self.K))
         # self.i = 0
         # checking input values
+        self.failed_design_dict = {}
         flag = self.section_classification(self)
         if len(self.input_section_list) == 0:
             flag == False
@@ -1225,85 +1236,78 @@ def set_input_values(self, design_dictionary):
     #         logger.error(": Design is unsafe. \n ")
     #         logger.info(" :=========End Of design===========")

+
     def section_classification(self):
-        """ Classify the sections based on Table 2 of IS 800:2007 """
-        # print(f"Inside section_classification")
-        first_section_logged = False
+        """Classify the sections based on Table 2 of IS 800:2007"""
```

Figure 4.26: From compression.py

```
src/osdag/design_type/compression_member/compression.py
+        self.failed_design_dict = {}
         flag = self.section_classification(self)
         if len(self.input_section_list) == 0:
             flag == False
@@ -1225,85 +1236,78 @@ def set_input_values(self, design_dictionary):
     #        logger.error(": Design is unsafe. \n ")
     #        logger.info(" :=========End Of design===========")

+
     def section_classification(self):
-        """ Classify the sections based on Table 2 of IS 800:2007 """
-        # print(f"Inside section_classification")
-        first_section_logged = False
+        """Classify the sections based on Table 2 of IS 800:2007"""
+        print(f"Inside section_classification")
         local_flag = True
+        self.input_modified = []
         self.input_section_list = []
         self.input_section_classification = {}
+        lambda_check = False

-
         for section in self.sec_list:
             trial_section = section.strip("'")
             # print(f"trial_section {trial_section}")

             # section_classification_subchecks(trial_section, self.material)

-            # Fetch the section properties
-            self.section_property = self.section_classification_subchecks(self, trial_section)
-            # print(f"Type of section{type(section)}")
+            # fetching the section properties
+            self.section_property = self.section_classification_subchecks(self,trial_section)
+            print(f"Type of section{type(section)}")

+            # section classification
             if (self.sec_profile in VALUES_SEC_PROFILE_Compression_Strut[:3]):  # Angles or Back to Back or 'Star Angle'
+
                 # updating the material property based on thickness of the thickest element
                 self.material_property.connect_to_database_to_get_fy_fu(self.material, self.section_property.thickness)
-
                 if self.section_property.type == 'Rolled':
                     if self.sec_profile == VALUES_SEC_PROFILE_Compression_Strut[0] or self.sec_profile == VALUES_SEC_PROFILE_Compression_Strut[2]:
-                        list_Table2_vi = IS800_2007.Table2_vi(self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
+                        list_Table2_vi= IS800_2007.Table2_vi(self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
                                                              self.material_property.fy, "Axial Compression")
                     elif self.sec_profile == VALUES_SEC_PROFILE_Compression_Strut[1]:
                         list_Table2_vi = IS800_2007.Table2_vii(self.section_property.min_leg,
-                                                               self.section_property.max_leg,
-                                                               self.section_property.thickness,
-                                                               self.material_property.fy, "Axial Compression")
+                                                                self.section_property.max_leg,
```

Figure 4.27: From compression.py

src/osdag/design_type/compression_member/compression.py

```diff
-        if self.section_property.section_class in self.allowed_sections and local_flag == True:
+            if self.section_property.section_class in self.allowed_sections and local_flag == True:
                 self.input_section_list.append(trial_section)
                 self.input_section_classification.update({trial_section: self.section_property.section_class})
                 # if self.sec_profile != Profile_name_1:
                 self.sec_prop_initial_dict.update({trial_section : (self.section_property.section_class, min_radius_gyration, slenderness, effective_area)}) #,
 self.width_thickness_ratio,self.depth_thickness_ratio,self.width_depth_thickness_ratio
         # print(f" sectopn class done {self.sec_list}")
         return local_flag
             # print(f"self.section_property.section_class{self.section_property.section_class}")
+

     #  ======Calculations start here====== #
     def optimization_tab_check(self):
@@ -1566,7 +1571,22 @@ def common_checks_1(self, section, step = 1, list_result = [], list_1 = []):


     def common_result(self, list_result,result_type):
-            self.result_designation = list_result[result_type]['Designation']
+
+            self.result_designation = list_result[result_type]["Designation"] # TODO debug
+            limit = IS800_2007.cl_3_8_max_slenderness_ratio(1)
+            if self.sec_prop_initial_dict[self.result_designation][2] > limit:
+                logger.warning("Length provided is beyond the limit allowed. [Reference: Cl 3.8, IS 800:2007]")
+                logger.error("Cannot compute. Given Length does not pass for this section.")
+                # self.sec_list.remove(self.section_property.designation )
+            else:
+                logger.info("Length provided is within the limit allowed. [Reference: Cl 3.8, IS 800:2007]" )
+            logger.info(
+                            "The section is {}. The b/t of the section ({}) is {} and d/t is {} and (b+d)/t is {}.  [Reference: Cl 3.7, IS 800:2007].".format(
+                               self.input_section_classification[self.result_designation], self.result_designation,
+                                round(self.width_thickness_ratio, 2), round_up(self.depth_thickness_ratio),
+                                round(self.width_depth_thickness_ratio, 2)))
+
+
             self.result_section_class = list_result[result_type]['Section class']
             self.result_effective_area = list_result[result_type]['Effective area']

@@ -1732,125 +1752,134 @@ def design_strut(self):
         # 2 - Based on optimum cost
         self.optimum_section_cost_results = {}
         self.optimum_section_cost = []
+        self.flag = self.section_classification(self)

-        if self.effective_area_factor == 1.0:
+        print('self.flag:',self.flag)
+        if self.effective_area_factor < 1.0:
+            logger.warning(
+                "Reducing the effective sectional area as per the definition in the Design Preferences tab."
+            )
```

Figure 4.28: From compression.py

src/osdag/design_type/compression_member/compression.py

```diff
+                print("Section result: \n",self.sec_profile, list_result)
+                # Step 3 - Storing the optimum results to a list in a descending order
+
+                list_1 = ['Designation','Section class', 'Effective area', 'Buckling_class', 'IF',
+                        'Effective_length', 'Effective_SR', 'EBS', 'lambda_vv', 'lambda_psi', 'ND_ESR', 'phi', 'SRF',
+                        'FCD_formula', 'FCD_max', 'FCD', 'Capacity', 'UR', 'Cost']
+                self.common_checks_1(self, section, 5, list_result, list_1)
+                print(f"\n self.optimum_section_cost_results {self.optimum_section_cost_results}"
+                f"\n self.optimum_section_ur_results {self.optimum_section_ur_results}")

     def min_rad_gyration_calc_strut(self,designation, material_grade,key,subkey, D_a=0.0,B_b=0.0,T_t=0.0,t=0.0):
         if key == Profile_name_1 and (subkey == loc_type1 or subkey == loc_type2):
@@ -2020,6 +2049,16 @@ def strength_of_strut(self):

     def results(self):
         """ """
+        _ = [i for i in self.optimum_section_ur if i > 1.0]
+        print( '_ ',_)
+        if len(_)==1:
+            temp = _[0]
+        elif len(_)==0:
+            temp = None
+        else:
+            temp = sorted(_)[0]
+        self.failed_design_dict = self.optimum_section_ur_results[temp] if temp is not None else None
+        print('self.failed_design_dict ',self.failed_design_dict)
         # sorting results from the dataset
         #if len(self.input_section_list) > 1 : #TODO: Parth to confirm if this code is needed
             #if design_dictionary[KEY_AXIAL] != '':  #TODO: Parth to confirm if this code is needed
@@ -2039,29 +2078,44 @@ def results(self):
                 logger.error("The solver did not find any adequate section from the defined list.")
                 logger.info("Re-define the list of sections or check the Design Preferences option and re-design.")
                 self.design_status = False
-                self.design_status_list.append(self.design_status)
+                if len(self.failed_design_dict)>0:
+                    logger.info(
+                    "The details for the best section provided is being shown"
+                )
+                    self.result_UR = self.failed_design_dict['UR'] #temp
+                    self.common_result(
+                        self,
+                        list_result=self.failed_design_dict,
+                        result_type=None,
+                    )
+                    logger.warning(
+                    "Re-define the list of sections or check the Design Preferences option and re-design."
+                )
+                    return
+                #self.design_status_list.append(self.design_status)

             else:
-                self.result_UR = self.optimum_section_ur[-1]  # optimum section which passes the UR check
```

Figure 4.29: From compression.py

```
src/osdag/design_type/compression_member/compression.py
@@ -2118,7 +2145,7 @@ def results(self,design_dictionary):
     ### start writing save_design from here!
     def save_design(self, popup_summary):

-        if self.connectivity == 'Hollow/Tubular Column Base':
+        """if self.connectivity == 'Hollow/Tubular Column Base':
             if self.dp_column_designation[1:4] == 'SHS':
                 select_section_img = 'SHS'
             elif self.dp_column_designation[1:4] == 'RHS':
@@ -2140,23 +2167,51 @@ def save_design(self, popup_summary):
             else:
                 section_type = 'Circular Hollow Section (CHS)'
         else:
-            section_type = 'I Section'
+            section_type = 'I Section' """
+
+        if self.section_property.max_leg == self.section_property.min_leg:
+            if self.sec_profile == "Back to Back Angles":
+                if self.loc == "Long Leg":
+                    image = "bblequaldp"
+                else:
+                    image = "bbsequaldp"
+            elif self.sec_profile == "Star Angles":
+                if self.loc == "Long Leg":
+                    image = "salequaldp"
+                else:
+                    image = "sasequaldp"
+            else:
+                image = "equaldp"

+        else:
+            if self.sec_profile == "Back to Back Angles":
+                if self.loc == "Long Leg":
+                    image = "bblunequaldp"
+                else:
+                    image = "bbsunequaldp"
+            elif self.sec_profile == "Star Angles":
+                if self.loc == "Long Leg":
+                    image = "salunequaldp"
+                else:
+                    image = "sasunequaldp"
+            else:
+                image = "unequaldp"
+

-        if self.section_property=='Columns' or self.section_property=='Beams':
-            self.report_column = {KEY_DISP_SEC_PROFILE: "ISection",
-                                  KEY_DISP_COLSEC_REPORT: self.section_property.designation,
```

Figure 4.30: From compression.py

```diff
src/osdag/design_type/compression_member/compression.py
+                #self.design_status_list.append(self.design_status)
 
             else:
-                self.result_UR = self.optimum_section_ur[-1]  # optimum section which passes the UR check
+                self.failed_design_dict = None
+                self.result_UR = self.optimum_section_ur[
+                    -1
+                ]  # optimum section which passes the UR check
                 print(f"self.result_UR{self.result_UR}")
                 self.design_status = True
+                self.common_result(
+                    self,
+                    list_result=self.optimum_section_ur_results,
+                    result_type=self.result_UR,
+                )
 
         else:  # results based on cost
             self.optimum_section_cost.sort()
 
             # selecting the section with most optimum cost
             self.result_cost = self.optimum_section_cost[0]
 
-        # print results
         if len(self.optimum_section_ur) == 0:
-            logger.warning(
-                "The sections selected by the solver from the defined list of sections did not satisfy the Utilization Ratio (UR) "
-                "criteria")
-            logger.error("The solver did not find any adequate section from the defined list.")
-            logger.info("Re-define the list of sections or check the Design Preferences option and re-design.")
             self.design_status = False
-            self.design_status_list.append(self.design_status)
-            pass
+
         else:
             if self.optimization_parameter == 'Utilization Ratio':
                 print(f" self.optimum_section_ur_results {self.optimum_section_ur_results}")
@@ -2198,154 +2252,177 @@ def save_design(self, popup_summary):
             else:
                 image = "unequaldp"
 
+        if (self.design_status and self.failed_design_dict is None) or (not self.design_status and len(self.failed_design_dict)>0):
+            if self.sec_profile == "Angles" or self.sec_profile == VALUES_SEC_PROFILE_2[0]:
+                self.report_column = {KEY_DISP_SEC_PROFILE: image,
+                                        KEY_DISP_SECSIZE: (self.section_property.designation, self.sec_profile),
+                                        KEY_DISP_MATERIAL: self.section_property.material,
+        #                                 KEY_DISP_APPLIED_AXIAL_FORCE: self.section_property.,
+                                        KEY_REPORT_MASS: self.section_property.mass,
+                                        KEY_REPORT_AREA: round(self.section_property.area * 1e-2, 2),
+                                        KEY_REPORT_MAX_LEG_SIZE: round(self.section_property.max_leg,2),
+                                        KEY_REPORT_MIN_LEG_SIZE: round(self.section_property.min_leg,2),
+                                        KEY_REPORT_ANGLE_THK: round(self.section_property.thickness,2),
+                                        KEY_REPORT_R1: self.section_property.root_radius,
```

Figure 4.31: From compression.py

```
src/osdag/design_type/compression_member/compression.py
```

```
+        self.report_input = \
+            {#KEY_MAIN_MODULE: self.mainmodule,
+                KEY_MODULE: self.module, #"Axial load on column "
+                KEY_DISP_AXIAL: self.load.axial_force/1000,
+                KEY_DISP_LENGTH: self.length,
+                KEY_DISP_SEC_PROFILE: self.sec_profile,
+                KEY_DISP_END1: self.end_1,
+                KEY_DISP_END2: self.end_2,
+                KEY_DISP_SECSIZE: self.result_section_class,
+                "Strut Section - Mechanical Properties": "TITLE",
+                KEY_DISP_ULTIMATE_STRENGTH_REPORT: round(self.section_property.fu, 2),
+                KEY_DISP_YIELD_STRENGTH_REPORT: round(self.section_property.fy, 2),
+                KEY_MATERIAL: self.material,
+                KEY_DISP_EFFECTIVE_AREA_PARA: self.effective_area_factor,
+                KEY_DISP_SECSIZE:  str(self.sec_list),
+                "Selected Section Details": self.report_column,
+                }
+
+        self.report_check = []
+
+        t1 = ('Selected', 'Selected Member Data', '|p{5cm}|p{2cm}|p{2cm}|p{2cm}|p{4cm}|')
+        self.report_check.append(t1)
+
+
+        t1 = ('SubSection', 'Buckling Class & Imperfection factor', '|p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
+        self.report_check.append(t1)
+        t1 = (KEY_DISP_BUCKLING_CURVE_ZZ, ' ',
+                    cl_8_7_1_5_buckling_curve(),
+                    ' ')
+        self.report_check.append(t1)
+
+        t1 = (KEY_DISP_IMPERFECTION_FACTOR_ZZ + r' ($\alpha$)', ' ',
+                    cl_8_7_1_5_imperfection_factor(self.result_IF),
+                    ' ')
+        self.report_check.append(t1)
+
+        t1 = ('SubSection', 'Section Classification', '|p{5cm}|p{3cm}|p{6.5cm}|p{1.5cm}|')
+        self.report_check.append(t1)
+        self.h = (self.section_property.leg_a_length - 2 * (self.section_property.thickness + self.section_property.root_radius))
+        t1 = ('Single Angle',
+            cl_3_7_2_section_classification_angle_required("b/t", self.section_property.section_class),
+            cl_3_7_2_section_classification_angle_provided(
+                self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
+                round(self.width_thickness_ratio, 2), "b/t", self.epsilon, self.section_property.section_class),
+            get_pass_fail(15.7 * self.epsilon, round(self.width_thickness_ratio, 2), relation="geq")
+        )
+        self.report_check.append(t1)
+
+        t1 = (
+            'Double Angles with the components separated',
+            cl_3_7_2_section_classification_angle_required("d/t", self.section_property.section_class),
+            cl_3_7_2_section_classification_angle_provided(
+                self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
```

Figure 4.32: From compression.py

```
src/osdag/design_type/compression_member/compression.py
+            t1 = (
+                'Double Angles with the components separated',
+                cl_3_7_2_section_classification_angle_required("d/t", self.section_property.section_class),
+                cl_3_7_2_section_classification_angle_provided(
+                    self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
+                    round(self.depth_thickness_ratio, 2), "d/t", self.epsilon, self.section_property.section_class),
+                get_pass_fail(15.7 * self.epsilon, round(self.depth_thickness_ratio, 2), relation="geq")
+            )
+            self.report_check.append(t1)
+
+            t1 = (
+                'Axial Compression',
+                cl_3_7_2_section_classification_angle_required("(b+d)/t", self.section_property.section_class),
+                cl_3_7_2_section_classification_angle_provided(
+                    self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
+                    round(self.width_depth_thickness_ratio, 2), "(b+d)/t", self.epsilon, self.section_property.section_class),
+                get_pass_fail(25 * self.epsilon, round(self.width_depth_thickness_ratio, 2), relation="geq")
+            )
+            self.report_check.append(t1)

-        self.report_input = \
+            t1 = ('(All the above three criteria should be satisfied)', '', '', '')
+            self.report_check.append(t1)
+
+
+            t1 = ('Section Class', ' ',
+                              cl_3_7_2_section_classification(
+                                                                    self.section_property.section_class),
+                              ' ')
+            self.report_check.append(t1)
+
+            t1 = ('SubSection', 'Effective Slenderness Ratio', '|p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
+            self.report_check.append(t1)
+            if self.load_type == 'Concentric Load':
+                K= self.result_eff_len / self.length
+                t1 = ("Effective Slenderness Ratio", ' ',
+                                    cl_7_1_2_effective_slenderness_ratio(K,self.length,round(self.min_radius_gyration, 2),self.result_eff_sr),
+                                    ' ')
+                self.report_check.append(t1)
+            else:
+                t1 = ("Effective Slenderness Ratio", ' ',
+                                cl_7_5_1_2_effective_slenderness_ratio(self.k1,self.k2,self.k3,self.lambda_vv,self.lambda_psi,self.result_eff_sr),
+                                ' ')
+                self.report_check.append(t1)
+
+
+            t1 = ('SubSection', 'Checks for Strength', '|p{4cm}|p{2 cm}|p{7cm}|p{3 cm}|')
+            self.report_check.append(t1)
+            t1 = (KEY_DISP_EULER_BUCKLING_STRESS_ZZ, ' ',
+                                cl_8_7_1_5_buckling_stress(self.section_property.modulus_of_elasticity,self.result_eff_sr, round(self.result_ebs, 2)),
 symbol to lanbdae
+                                ' ')
+            self.report_check.append(t1)
```

Figure 4.33: From compression.py

```
+
+            t1 = (r'$F_{cd} \, \left( \frac{N}{\text{mm}^2} \right)$', ' ',
+                                cl_8_7_1_5_Buckling(self.material_property.fy,self.gamma_m0,round(self.nondimensional_effective_slenderness_ratio, 2),round(self.phi,
 2),round(self.design_compressive_stress_max, 2),round(self.design_compressive_stress, 2)), '')
+            self.report_check.append(t1)
+
+            t1 = ('P_d', self.load.axial_force * 10 ** -3,
+                                cl_7_1_2_design_compressive_strength(round(self.result_capacity * 10 ** -3, 2),round(self.result_effective_area, 2),
 round(self.design_compressive_stress, 2),self.load.axial_force * 10 ** -3),
+                                get_pass_fail(self.load.axial_force * 10 ** -3, round(self.result_capacity * 10 ** -3, 2), relation="leq"))
+            self.report_check.append(t1)
+
+        else:
+            self.report_input = \
            {#KEY_MAIN_MODULE: self.mainmodule,
                KEY_MODULE: self.module, #"Axial load on column "
                KEY_DISP_AXIAL: self.load.axial_force/1000,
                KEY_DISP_LENGTH: self.length,
                KEY_DISP_SEC_PROFILE: self.sec_profile,
                KEY_DISP_END1: self.end_1,
                KEY_DISP_END2: self.end_2,
-               KEY_DISP_SECSIZE: self.result_section_class,
+               #KEY_DISP_SECSIZE: self.result_section_class,
                "Strut Section - Mechanical Properties": "TITLE",
                KEY_DISP_ULTIMATE_STRENGTH_REPORT: round(self.section_property.fu, 2),
                KEY_DISP_YIELD_STRENGTH_REPORT: round(self.section_property.fy, 2),
                KEY_MATERIAL: self.material,
                KEY_DISP_EFFECTIVE_AREA_PARA: self.effective_area_factor,
                KEY_DISP_SECSIZE:  str(self.sec_list),
-               "Selected Section Details": self.report_column,
-               }
-
```

Figure 4.34: From compression.py

```
+
+        Disp_2d_image = []
+        Disp_3D_image = "/ResourceFiles/images/3d.png"
         print(sys.path[0])
         rel_path = str(sys.path[0])
         rel_path = os.path.abspath(".") # TEMP
         rel_path = rel_path.replace("\\", "/")
         fname_no_ext = popup_summary['filename']
         CreateLatex.save_latex(CreateLatex(), self.report_input, self.report_check, popup_summary, fname_no_ext,
-                              rel_path, [], '', module=self.module)
+                              rel_path, Disp_2d_image, Disp_3D_image, module=self.module)


     # def memb_pattern(self, status):
@@ -2388,4 +2390,5 @@ def save_design(self, popup_summary):
     #             ['./ResourceFiles/images/L.png',400,202, "Plate Block Shear Pattern"])  # [image, width, height, caption]
     #     pattern.append(t99)
     #
-    #     return pattern
+    #     return pattern
+
```

Figure 4.35: From compression.py

Chapter 5

Internship Task 3 Title: Miscellaneous

5.1 Task 3: Problem Statement

Several issues and errors were encountered across various functionalities and features, including image rendering problems, unexpected float type errors, and ineffective input/output docks. Specific challenges were noted in the BC-End Plate Typical Sketch and the compression members' output dock, which failed to perform as intended. Additionally, discrepancies in selected member data and ineffective lengths for flexural members further complicated the workflow, requiring corrections and updates to ensure accuracy and functionality.

5.2 Task 3: Tasks Done

The image issues and float type errors were resolved by type-casting the problem causing variables to int(). Adjustments were made to the input dock image paths by converting them to absolute path instead of relative, ensuring seamless operation and addressing any image-related discrepancies. The BC-End Plate Typical Sketch was causing an error due to a similar float-type error in the `ui_template.py` file and was rectified again by type-casting the problem causing variables to int() to reflect the required design sketch correctly. The output dock for compression members and the selected member data functionality were fixed to provide accurate results of the most optimum section instead of a random section from the input list as it was doing before. Effective lengths for flexural member modules were fixed to show the lengths for the most optimum section instead of a random section, which was varying each time to ensure compliance with design standards. These fixes collectively enhanced the reliability and precision of the system.

5.3 Task 3: Python Code

The bug-fixes for some of these miscellaneous tasks are described below:

5.3.1 For fixing Output Dock Designation and Selected Member Data

```
src/osdag/design_type/compression_member/Column.py
-            #logger.info("The actual effective area is {} mm2 and the reduced effective area is {} mm2 [Reference: Cl. 7.3.2, IS 800:2007]".
-            #             format(round((self.effective_area / self.effective_area_factor), 2), self.effective_area))
-        #else:
-            #if self.section_class != 'Slender':
-            #    logger.info("The effective sectional area is taken as 100% of the cross-sectional area [Reference: Cl. 7.3.2, IS 800:2007].")

         #print('self.input_section_list:',self.input_section_list)
         if self.flag:
@@ -923,7 +913,7 @@ def design_column(self):

                 self.material_property.connect_to_database_to_get_fy_fu(self.material, max(self.section_property.flange_thickness,
                                                                                             self.section_property.web_thickness))
-
+                self.epsilon = math.sqrt(250 / self.material_property.fy)


                 # initialize lists for updating the results dictionary
@@ -937,9 +927,6 @@ def design_column(self):
                 self.section_class = self.input_section_classification[section][0]

                 if self.section_class == 'Slender':
-                    #logger.warning("The trial section ({}) is Slender. Computing the Effective Sectional Area as per Sec. 9.7.2, "
-                    #             "Fig. 2 (B & C) of The National Building Code of India (NBC), 2016.".format(section))
-
                     if (self.sec_profile == VALUES_SEC_PROFILE[0]):  # Beams and Columns
                         self.effective_area = (2 * ((31.4 * self.epsilon * self.section_property.flange_thickness) *
                                               self.section_property.flange_thickness)) + \
@@ -950,6 +937,9 @@ def design_column(self):
                     self.effective_area = self.section_property.area  # mm2
                     # print(f"self.effective_area{self.effective_area}")

+                if self.effective_area_factor < 1.0:
+                    self.effective_area = round(self.effective_area * self.effective_area_factor, 2)
+
                 self.list_zz.append(self.section_class)
                 self.list_yy.append(self.section_class)

@@ -1416,6 +1406,11 @@ def save_design(self, popup_summary):

         if (self.design_status and self.failed_design_dict is None) or (not self.design_status and len(self.failed_design_dict)>0):
             if self.sec_profile=='Columns' or self.sec_profile=='Beams' or self.sec_profile == VALUES_SEC_PROFILE[0]:
+                try:
+                    result = Beam(designation=self.result_designation, material_grade=self.material)
+                except:
+                    result = Column(designation=self.result_designation, material_grade=self.material)
+                self.section_property = result
                 self.report_column = {KEY_DISP_SEC_PROFILE: "ISection",
                                       KEY_DISP_SECSIZE: (self.section_property.designation, self.sec_profile),
                                       KEY_DISP_COLSEC_REPORT: self.section_property.designation,
```

Figure 5.1: From Column.py

src/osdag/design_type/compression_member/compression.py

```
        if (self.design_status and self.failed_design_dict is None) or (not self.design_status and len(self.failed_design_dict)>0):
+           if self.sec_profile == Profile_name_1 or self.sec_profile == Profile_name_2 or self.sec_profile == Profile_name_3:  # Angles and Back to Back Angles
+               self.section_property = Angle(designation = self.result_designation, material_grade = self.material)
            if self.sec_profile == "Angles" or self.sec_profile == VALUES_SEC_PROFILE_2[0]:
                self.report_column = {KEY_DISP_SEC_PROFILE: image,
                                      KEY_DISP_SECSIZE: (self.section_property.designation, self.sec_profile),
@@ -2326,30 +2328,30 @@ def save_design(self, popup_summary):
            self.report_check.append(t1)
            self.h = (self.section_property.leg_a_length - 2 * (self.section_property.thickness + self.section_property.root_radius))
            t1 = ('Single Angle',
-               cl_3_7_2_section_classification_angle_required("b/t", self.section_property.section_class),
+               cl_3_7_2_section_classification_angle_required("b/t", self.input_section_classification[self.result_designation]),
                cl_3_7_2_section_classification_angle_provided(
                    self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
-                   round(self.width_thickness_ratio, 2), "b/t", self.epsilon, self.section_property.section_class),
+                   round(self.width_thickness_ratio, 2), "b/t", self.epsilon, self.input_section_classification[self.result_designation]),
                get_pass_fail(15.7 * self.epsilon, round(self.width_thickness_ratio, 2), relation="geq")
            )
            self.report_check.append(t1)

            t1 = (
                'Double Angles with the components separated',
-               cl_3_7_2_section_classification_angle_required("d/t", self.section_property.section_class),
+               cl_3_7_2_section_classification_angle_required("d/t", self.input_section_classification[self.result_designation]),
                cl_3_7_2_section_classification_angle_provided(
                    self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
-                   round(self.depth_thickness_ratio, 2), "d/t", self.epsilon, self.section_property.section_class),
+                   round(self.depth_thickness_ratio, 2), "d/t", self.epsilon, self.input_section_classification[self.result_designation]),
                get_pass_fail(15.7 * self.epsilon, round(self.depth_thickness_ratio, 2), relation="geq")
            )
            self.report_check.append(t1)

            t1 = (
                'Axial Compression',
-               cl_3_7_2_section_classification_angle_required("(b+d)/t", self.section_property.section_class),
+               cl_3_7_2_section_classification_angle_required("(b+d)/t", self.input_section_classification[self.result_designation]),
                cl_3_7_2_section_classification_angle_provided(
                    self.section_property.min_leg, self.section_property.max_leg, self.section_property.thickness,
-                   round(self.width_depth_thickness_ratio, 2), "(b+d)/t", self.epsilon, self.section_property.section_class),
+                   round(self.width_depth_thickness_ratio, 2), "(b+d)/t", self.epsilon, self.input_section_classification[self.result_designation]),
                get_pass_fail(25 * self.epsilon, round(self.width_depth_thickness_ratio, 2), relation="geq")
            )
            self.report_check.append(t1)
@@ -2359,9 +2361,7 @@ def save_design(self, popup_summary):


            t1 = ('Section Class', ' ',
-                         cl_3_7_2_section_classification(
-                                                             self.section_property.section_class),
-                         ' ')
+               cl_3_7_2_section_classification(self.input_section_classification[self.result_designation]), ' ')
            self.report_check.append(t1)
```

Figure 5.2: From compression.py

5.3.2 For fixing Input and Output Dock Images and Sketch

These path changes were made to all the python source files for each module of the Connection and Compression Members, wherever suitable.

```diff
src/osdag/design_type/connection/beam_beam_end_plate_splice.py
@@ -37,6 +37,7 @@
 from ...design_report.reportGenerator import save_html
 from ...Report_functions import *
 from ...design_report.reportGenerator_latex import CreateLatex
+from importlib.resources import files

 import logging
 import math
@@ -224,7 +225,7 @@ def input_values(self):
         t2 = (KEY_ENDPLATE_TYPE, KEY_DISP_ENDPLATE_TYPE, TYPE_COMBOBOX, VALUES_ENDPLATE_TYPE, True, 'No Validator')
         options_list.append(t2)

-        t15 = (KEY_IMAGE, None, TYPE_IMAGE, "./ResourceFiles/images/flush_ep.png", True, 'No Validator')
+        t15 = (KEY_IMAGE, None, TYPE_IMAGE, str(files("osdag.data.ResourceFiles.images").joinpath("flush_ep.png")), True, 'No Validator')
         options_list.append(t15)

         t4 = (KEY_SUPTDSEC, KEY_DISP_BEAMSEC, TYPE_COMBOBOX, connectdb("Beams"), True, 'No Validator')
@@ -284,19 +285,21 @@ def input_value_changed(self):

         return lst

+
     def fn_conn_image(self):
-        """ display representative images of end plate type """
+        """ Display representative images of end plate type """

         ep_type = self[0]
         if ep_type == VALUES_ENDPLATE_TYPE[0]:
-            return './ResourceFiles/images/flush_ep.png'
+            return str(files("osdag.data.ResourceFiles.images").joinpath("flush_ep.png"))
         elif ep_type == VALUES_ENDPLATE_TYPE[1]:
-            return './ResourceFiles/images/owe_ep.png'
+            return str(files("osdag.data.ResourceFiles.images").joinpath("owe_ep.png"))
         elif ep_type == VALUES_ENDPLATE_TYPE[2]:
-            return './ResourceFiles/images/extended.png'
+            return str(files("osdag.data.ResourceFiles.images").joinpath("extended.png"))
         else:
             return ''

+
     # create customized input for UI
     def customized_input(self):
         """ list of values available with customize option"""
@@ -483,15 +486,15 @@ def stiffener_detailing(self, status):
         detailing = []

         if self.endplate_type == VALUES_ENDPLATE_TYPE[0]:  # Flush EP
-            detailing_path = './ResourceFiles/images/BB_Stiffener_FP.png'
+            detailing_path = str(files("osdag.data.ResourceFiles.images").joinpath("BB_Stiffener_FP.png"))
```

Figure 5.3: From compression.py

```
src/osdag/gui/ui_template.py
@@ -2273,7 +2273,7 @@ def output_button_dialog(self, main, button_list, button):

                        if value is not None and value != "":
                            im = QtWidgets.QLabel(image_widget)
-                           im.setFixedSize(value[1], value[2])
+                           im.setFixedSize(int(value[1]), int(value[2]))
                            pmap = QPixmap(value[0])
                            im.setScaledContents(1)
                            im.setStyleSheet("background-color: white;")
@@ -2282,7 +2282,7 @@ def output_button_dialog(self, main, button_list, button):
                            caption = QtWidgets.QLabel(image_widget)
                            caption.setAlignment(Qt.AlignCenter)
                            caption.setText(value[3])
-                           caption.setFixedSize(value[1], caption.sizeHint().height())
+                           caption.setFixedSize(int(value[1]), caption.sizeHint().height())
                            image_layout.addWidget(caption)
                            max_image_width = max(max_image_width, value[1])
                            max_image_height = max(max_image_height, value[2])
@@ -2312,15 +2312,15 @@ def output_button_dialog(self, main, button_list, button):
                    if option_type == TYPE_IMAGE:
                        im = QtWidgets.QLabel(image_widget)
                        im.setScaledContents(True)
-                       im.setFixedSize(value[1], value[2])
+                       im.setFixedSize(int(value[1]), int(value[2]))
                        pmap = QPixmap(value[0])
                        im.setStyleSheet("background-color: white;")
                        im.setPixmap(pmap)
                        image_layout.addWidget(im)
                        caption = QtWidgets.QLabel(image_widget)
                        caption.setAlignment(Qt.AlignCenter)
                        caption.setText(value[3])
-                       caption.setFixedSize(value[1], 12)
+                       caption.setFixedSize(int(value[1]), 12)
                        image_layout.addWidget(caption)
                        max_image_width = max(max_image_width, value[1])
                        max_image_height = max(max_image_height, value[2])
@@ -2344,8 +2344,8 @@ def output_button_dialog(self, main, button_list, button):
                dialog_height = max(dialog_height, max_image_height+125)
                if not no_note:
                    dialog_height += 40
-               dialog.resize(dialog_width, dialog_height)
-               dialog.setMinimumSize(dialog_width, dialog_height)
+               dialog.resize(int(dialog_width), int(dialog_height))
+               dialog.setMinimumSize(int(dialog_width), int(dialog_height))

                if no_note:
                    layout1.removeWidget(note_widget)
```

Figure 5.4: From compression.py

Chapter 6

Internship Task 4 Title: Documentation of Report Generation and the New Installer Method

6.1 Task 3: Problem Statement

Effective documentation is required to support the ongoing development and maintenance of Osdag, focusing on two critical areas: the report generation process and the new installer method. For report generation, there is a need to document how reports are created, including the commands, scripts, and files involved, to provide a clear reference for new interns and project staff. Additionally, the philosophy and implementation of the new installer method need to be articulated to ensure a comprehensive understanding of its purpose and functionality, enabling seamless onboarding and troubleshooting.

6.2 Task 3: Tasks Done

The report generation process in Osdag has been meticulously documented, outlining the sequence of steps, key files, and commands utilized. This includes details about the specific scripts that control report formatting, how each data is processed, and output generation, providing a practical guide for newcomers. For the new installer method, the underlying philosophy and its implementation steps have been clearly articulated. This documentation highlights the rationale behind the new approach, the changes made from previous methods, and the specific functionalities added. Together, these efforts create a robust resource to assist future contributors in understanding and improving these areas.

6.3 Task 3: Documentation

The links for documentation are as follows:

- For Report Generation: Click here.
- For New Installer Method: Click here.

Chapter 7

Conclusions

7.1 Tasks Accomplished

The issues related to the database, themes, and image paths have been successfully addressed for the new installer method. Additionally, report generation has been completed for the "Beams and Column" section profile in the "Column with Known Support Conditions" module, as well as for the "Angles" section profile in the "Struts in Trusses" module. The issues regarding co-relation of CAD generated with the optimum section results has been fixed, and the design log and output dock functionality have been fully resolved for all section profiles across both modules. Comprehensive documentation has been prepared detailing the report generation process and the new installer method.

7.2 Skills Developed

Throughout the internship, I strengthened my technical capabilities and collaborative skillset. I gained more knowledge about GitHub's advanced features for version control, including branch management, pull request workflows, and merge conflict resolution. Using LaTeX, I created comprehensive technical documentation and reports while adhering to professional formatting standards. I significantly expanded my Python programming expertise by implementing the use of different libraries, and optimizing code performance. In the area of software development practices, I gained hands-on experience with code review methodologies, receiving constructive feedback from my mentors while maintaining code quality standards. Through working with Conda, I developed proficiency in managing virtual environments, handling package dependencies, and building distributable applications. The collaborative nature of the internship enhanced my ability to work effectively in cross-functional teams, communicate technical concepts clearly, and coordinate multiple project timelines simultaneously.

Chapter A

Appendix

A.1 Work Reports

Sr. No.	Date	Day	Task	Hours Spent
1	12 November 2024	Tuesday	Joining | Install instructions for Osdag | Session with Ajinkya to discuss new intended process and the current stage of the application.	2 to 3 hours.
2	13 November 2024	Wednesday	Attempt to install on personal devices and record behaviour.	4 to 5 hours. (Kept running into issues.)
3	14 November 2024	Thursday	Meeting with Ajinkya to finalize schedule.	1 hour.
4	15 November 2024	Friday	Attempt installing using a Windows VM, and try on Mac again.	5 hours. (Found the correct process-flow to follow and successfully installed.)
5	16 November 2024	Saturday	Weekly Holiday | Successfully installed and got the application working on both the machines.	
6	17 November 2024	Sunday	Weekly Holiday	
7	18 November 2024	Monday	Study break for examinations (Followed up with the progress of the team, tried debugging few errors in the report generation.)	Spent the time according to the breaks in between the exams.
8	19 November 2024	Tuesday		
9	20 November 2024	Wednesday		
10	21 November 2024	Thursday		
11	22 November 2024	Friday		
12	23 November 2024	Saturday		
13	24 November 2024	Sunday		
14	25 November 2024	Monday		
15	26 November 2024	Tuesday		
16	27 November 2024	Wednesday	Tested the working of the application after implementing the workarounds for database and image issues in report generation | Review status of conda install and figure out schedule related to NSIS.	3 hours. (+1 and half hour of the follow-up meeting with Ajinkya and Nandagopal regarding NSIS and review status of conda install.)
17	28 November 2024	Thursday	Resolve few path issues | See how temp image files (the last 4 images in the report generated) are created, tried implementing the tempfile module.	4 hours.
18	29 November 2024	Friday		4 to 4 and a half hours.
19	30 November 2024	Saturday	Weekly Holiday	
20	1 December 2024	Sunday	Weekly Holiday | Read about NSIS to familirize myself | Tried tinkering with the existing Windows Installer script.	
21	2 December 2024	Monday	Tested all the modules of the application, resolved few unexpected float errors, and checked the PyQt-GUI, rendering and report generation of these modules.	4 hours. (Took a bit of time in doing trial and error for the input specifications of the modules.)
22	3 December 2024	Tuesday	Travel	
23	4 December 2024	Wednesday	Follow up with Parth regarding the latex report generation of Compression Members.	
24	5 December 2024	Thursday	Worked on the Column with know support conditions LaTeX Report Generation and Integration wth CAD.	Spent 6 to 7 hours as per work required.
25	6 December 2024	Friday		Spent 6 to 7 hours as per work required.
26	7 December 2024	Saturday		3 to 4 hours.
27	8 December 2024	Sunday		3 to 4 hours.
28	9 December 2024	Monday		Spent 6 to 7 hours as per work required.
29	10 December 2024	Tuesday		Spent 6 to 7 hours as per work required.
30	11 December 2024	Wednesday		Spent 6 to 7 hours as per work required.
31	12 December 2024	Thursday		Spent 6 to 7 hours as per work required.
32	13 December 2024	Friday		Spent 6 to 7 hours as per work required.
33	14 December 2024	Saturday	Worked on the Struts in Trusses LaTeX Report Generation and Integration with CAD. Also spent time on working with the new installer method, and NSIS part.	3 to 4 hours.
34	15 December 2024	Sunday		3 to 4 hours.
35	16 December 2024	Monday		Spent 4 to 5 hours as per work required.
36	17 December 2024	Tuesday		Spent 4 to 5 hours as per work required.
37	18 December 2024	Wednesday		Spent 4 to 5 hours as per work required.
38	19 December 2024	Thursday		Spent 4 to 5 hours as per work required.
39	20 December 2024	Friday	Fixed the image issues related to Input Dock and resolved conflicts with merging.	4 hours.
40	21 December 2024	Saturday	Weekly Holiday	
41	22 December 2024	Sunday	Weekly Holiday | Work on Documentation for New Installer.	2 hours.
42	23 December 2024	Monday	Resolved Output Dock and Design Log for Compression Members.	4 hours.
43	24 December 2024	Tuesday		2 hours.
44	25 December 2024	Wednesday	Fixed Effective Length Issue in Flexure Members.	3 hours.
45	26 December 2024	Thursday	Fixed Selected Member Data Issue in Compression Members.	4 hours.
46	27 December 2024	Friday	Worked on Documentation for Report Generation and New Installer Method.	3 hours.
47	28 December 2024	Saturday		5 hours.

Bibliography

[1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.

[3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.