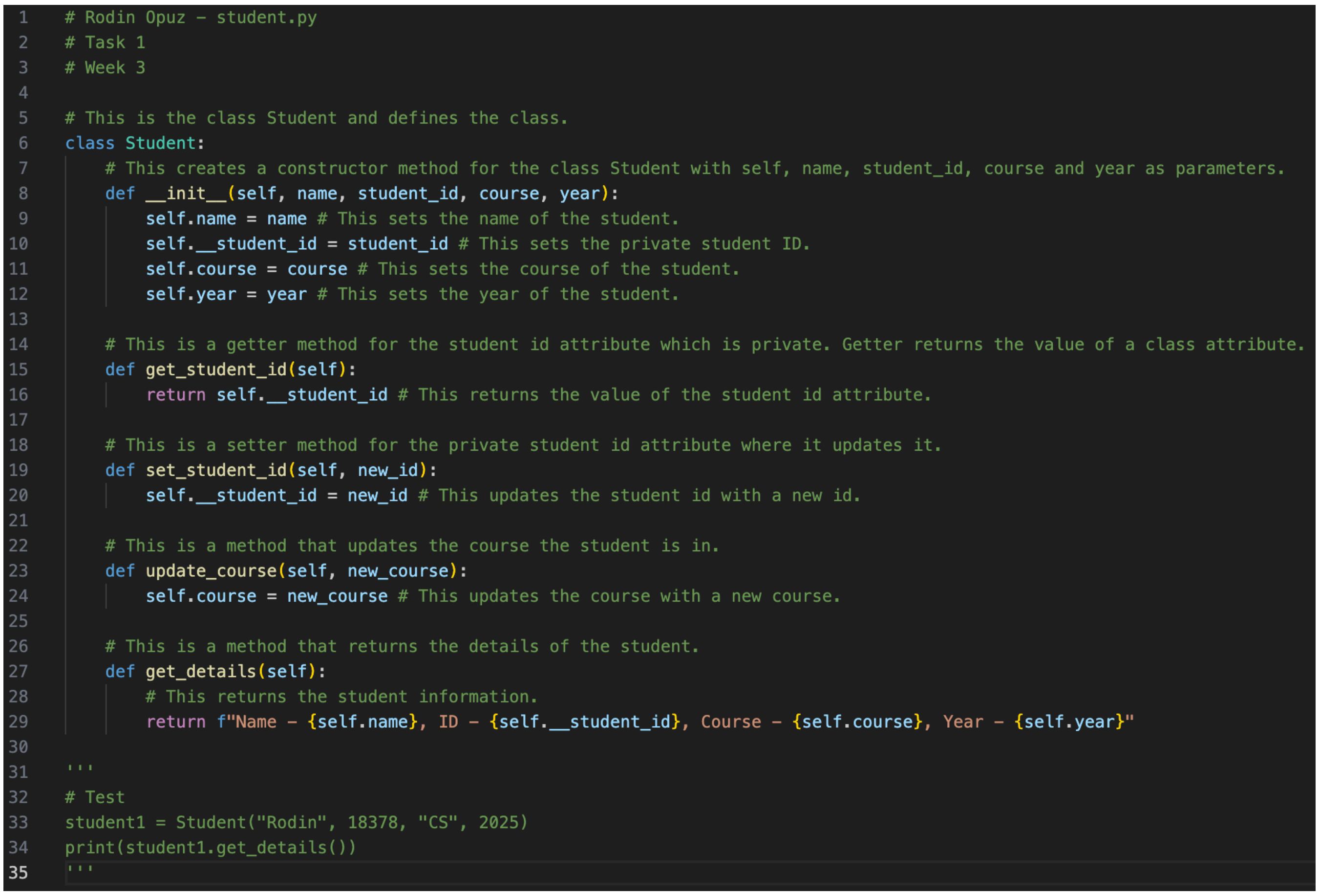
IN0009 Coursework Project

This report includes the system design, implementation, challenges faced and recommendations for future improvements. Rodin Opuz - Coursework

Part 1 Project

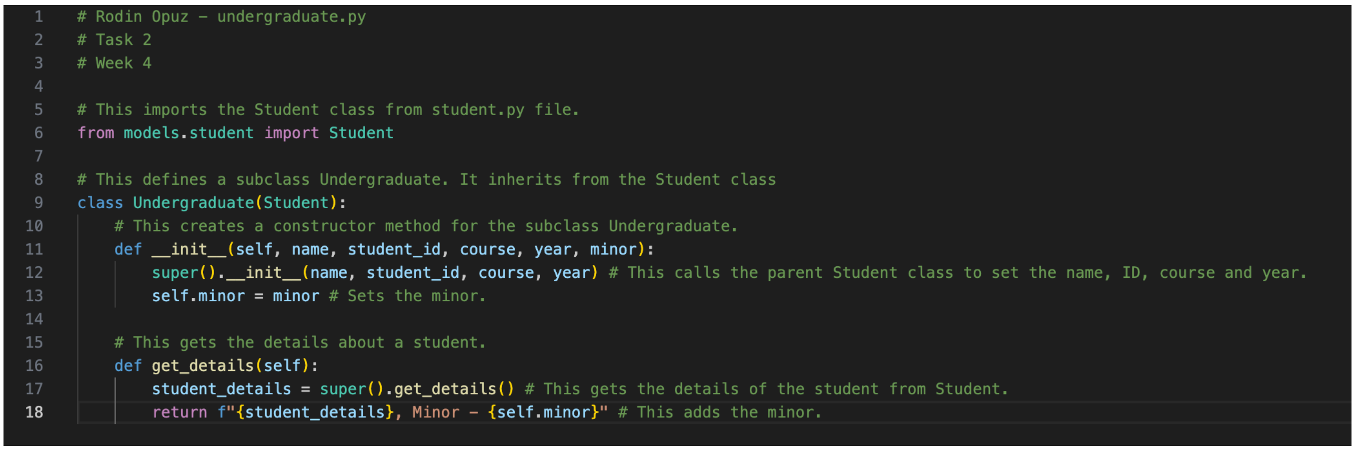
Task 1



```python
# Rodin Opuz - student.py
# Task 1
# Week 3

# This is the class Student and defines the class.
class Student:
    # This creates a constructor method for the class Student with self, name, student_id, course and year as parameters.
    def __init__(self, name, student_id, course, year):
        self.name = name # This sets the name of the student.
        self.__student_id = student_id # This sets the private student ID.
        self.course = course # This sets the course of the student.
        self.year = year # This sets the year of the student.

    # This is a getter method for the student id attribute which is private. Getter returns the value of a class attribute.
    def get_student_id(self):
        return self.__student_id # This returns the value of the student id attribute.

    # This is a setter method for the private student id attribute where it updates it.
    def set_student_id(self, new_id):
        self.__student_id = new_id # This updates the student id with a new id.

    # This is a method that updates the course the student is in.
    def update_course(self, new_course):
        self.course = new_course # This updates the course with a new course.

    # This is a method that returns the details of the student.
    def get_details(self):
        # This returns the student information.
        return f"Name - {self.name}, ID - {self.__student_id}, Course - {self.course}, Year - {self.year}"

'''
# Test
student1 = Student("Rodin", 18378, "CS", 2025)
print(student1.get_details())
'''
```

This is the Student class which defines the attributes of a student. This includes ID, name, course and year. The student ID is a private attribute. There are getter methods and setter methods. This Student class has get_student_id function which returns the student ID attribute, set_student_id function which updates an ID, update_course function which updates the course and get_details function which returns the details of the student. It returns in a string that is formatted.

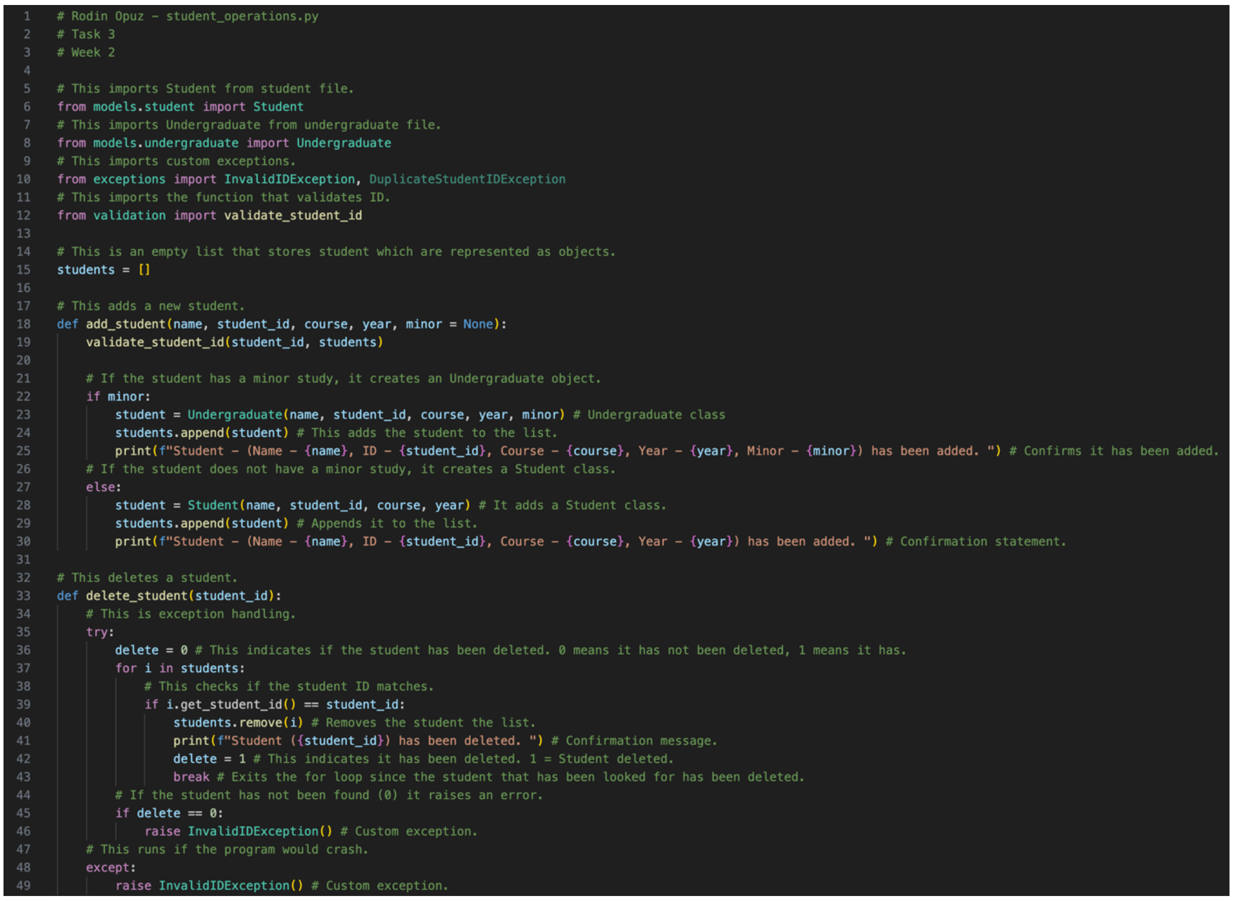
Part 2



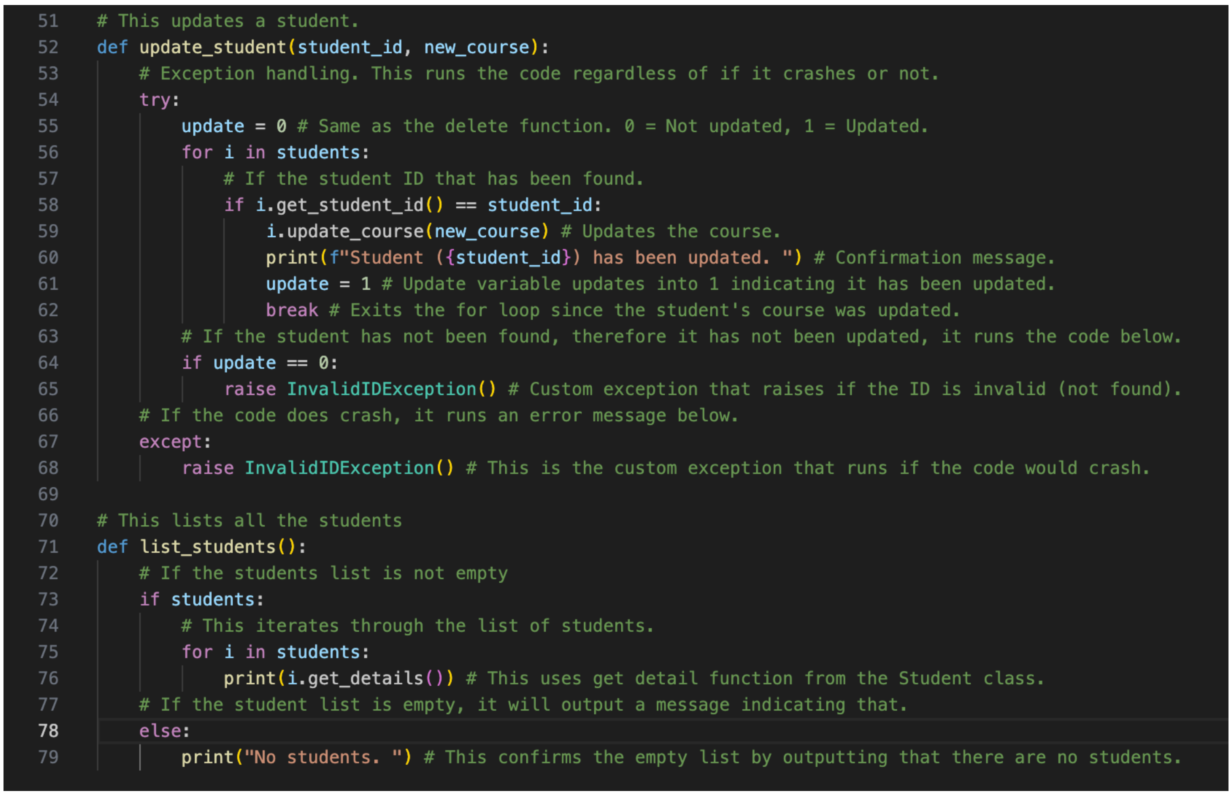
```python
# Rodin Opuz - undergraduate.py
# Task 2
# Week 4

# This imports the Student class from student.py file.
from models.student import Student

# This defines a subclass Undergraduate. It inherits from the Student class
class Undergraduate(Student):
    # This creates a constructor method for the subclass Undergraduate.
    def __init__(self, name, student_id, course, year, minor):
        super().__init__(name, student_id, course, year) # This calls the parent Student class to set the name, ID, course and year.
        self.minor = minor # Sets the minor.

    # This gets the details about a student.
    def get_details(self):
        student_details = super().get_details() # This gets the details of the student from Student.
        return f"{student_details}, Minor - {self.minor}" # This adds the minor.
```

This is the Undergraduate class which inherits from the Student class which is imported. This adds a minor attribute. It calls the main class's constructor from which it inherits from and adds a new attribute (minor). The get_details includes the minor in the details. This includes information about the minor study and the details that are inherited from the main class Student.

Part 3



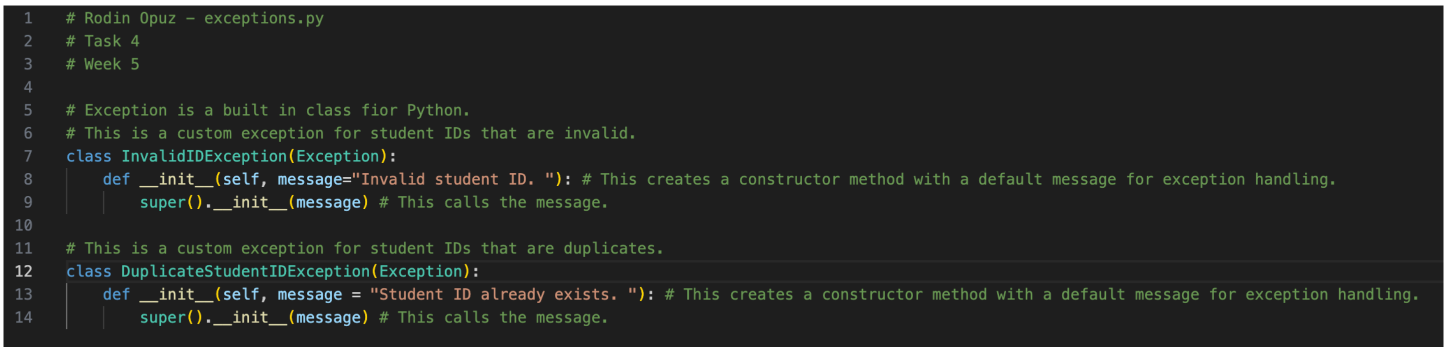
```python
# Rodin Opuz - student_operations.py
# Task 3
# Week 2

# This imports Student from student file.
from models.student import Student
# This imports Undergraduate from undergraduate file.
from models.undergraduate import Undergraduate
# This imports custom exceptions.
from exceptions import InvalidIDException, DuplicateStudentIDException
# This imports the function that validates ID.
from validation import validate_student_id

# This is an empty list that stores student which are represented as objects.
students = []

# This adds a new student.
def add_student(name, student_id, course, year, minor = None):
    validate_student_id(student_id, students)

    # If the student has a minor study, it creates an Undergraduate object.
    if minor:
        student = Undergraduate(name, student_id, course, year, minor) # Undergraduate class
        students.append(student) # This adds the student to the list.
        print(f"Student - (Name - {name}, ID - {student_id}, Course - {course}, Year - {year}, Minor - {minor}) has been added. ") # Confirms it has been added.
    # If the student does not have a minor study, it creates a Student class.
    else:
        student = Student(name, student_id, course, year) # It adds a Student class.
        students.append(student) # Appends it to the list.
        print(f"Student - (Name - {name}, ID - {student_id}, Course - {course}, Year - {year}) has been added. ") # Confirmation statement.

# This deletes a student.
def delete_student(student_id):
    # This is exception handling.
    try:
        delete = 0 # This indicates if the student has been deleted. 0 means it has not been deleted, 1 means it has.
        for i in students:
            # This checks if the student ID matches.
            if i.get_student_id() == student_id:
                students.remove(i) # Removes the student the list.
                print(f"Student ({student_id}) has been deleted. ") # Confirmation message.
                delete = 1 # This indicates it has been deleted. 1 = Student deleted.
                break # Exits the for loop since the student that has been looked for has been deleted.
        # If the student has not been found (0) it raises an error.
        if delete == 0:
            raise InvalidIDException() # Custom exception.
    # This runs if the program would crash.
    except:
        raise InvalidIDException() # Custom exception.
```



```python
# This updates a student.
def update_student(student_id, new_course):
    # Exception handling. This runs the code regardless of if it crashes or not.
    try:
        update = 0 # Same as the delete function. 0 = Not updated, 1 = Updated.
        for i in students:
            # If the student ID that has been found.
            if i.get_student_id() == student_id:
                i.update_course(new_course) # Updates the course.
                print(f"Student ({student_id}) has been updated. ") # Confirmation message.
                update = 1 # Update variable updates into 1 indicating it has been updated.
                break # Exits the for loop since the student's course was updated.
        # If the student has not been found, therefore it has not been updated, it runs the code below.
        if update == 0:
            raise InvalidIDException() # Custom exception that raises if the ID is invalid (not found).
    # If the code does crash, it runs an error message below.
    except:
        raise InvalidIDException() # This is the custom exception that runs if the code would crash.

# This lists all the students
def list_students():
    # If the students list is not empty
    if students:
        # This iterates through the list of students.
        for i in students:
            print(i.get_details()) # This uses get detail function from the Student class.
    # If the student list is empty, it will output a message indicating that.
    else:
        print("No students. ") # This confirms the empty list by outputting that there are no students.
```

This includes the functions in order for the user to manage the system. It includes add students function which adds students to the system, delete students which deletes students from the system, update students which updates students from the system and list students which outputs all the students in the system. The student information is stored in a list. It uses custom exceptions to manage errors.

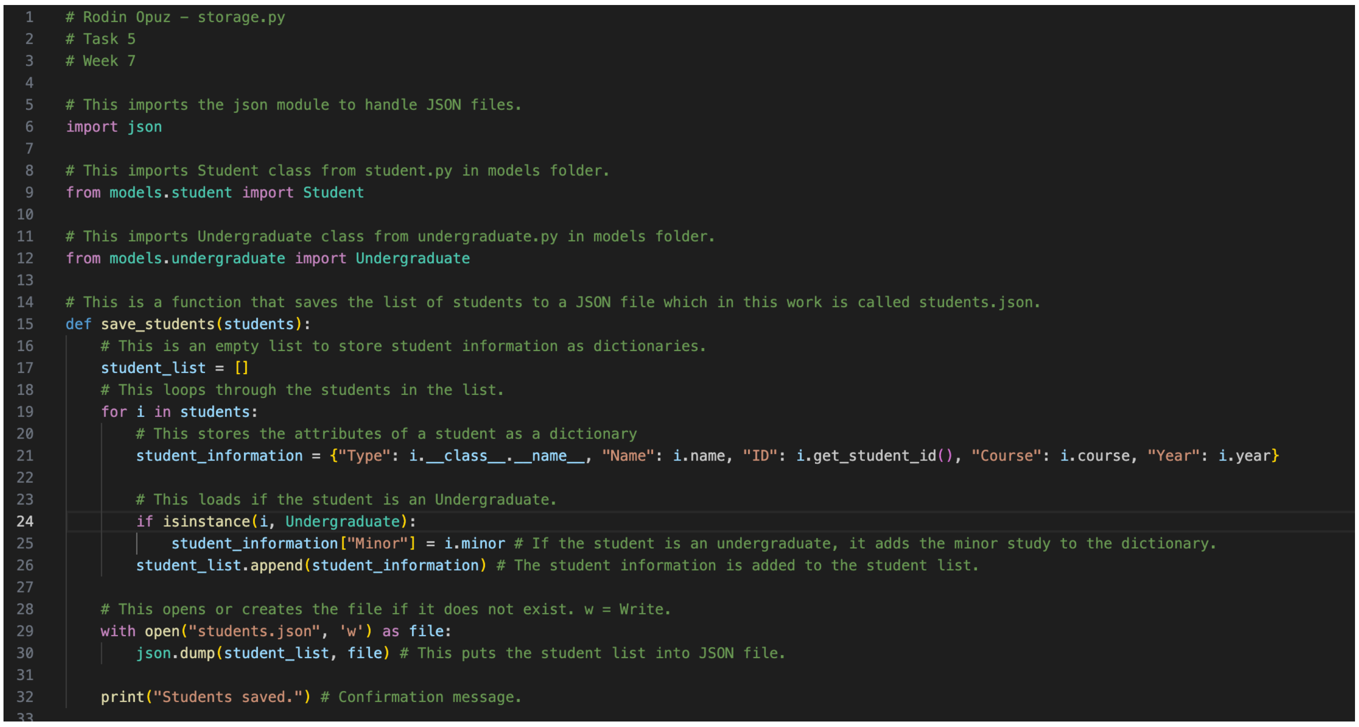
Part 4



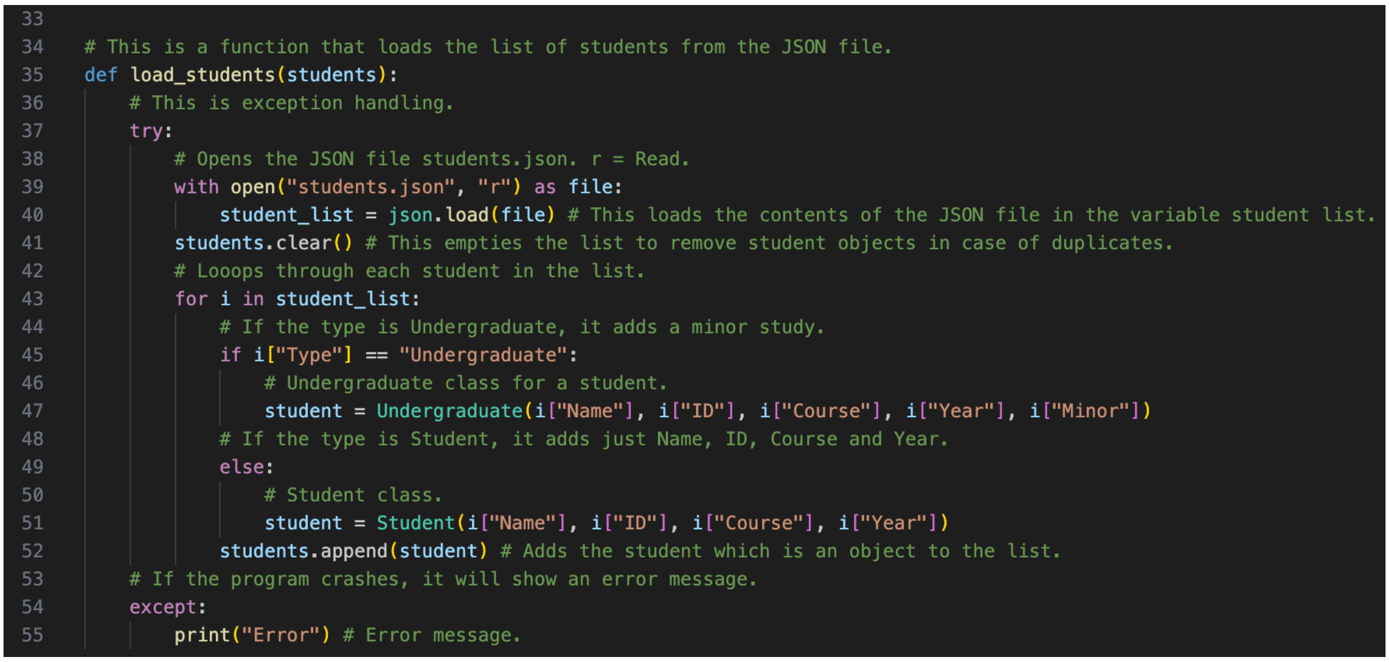
```python
# Rodin Opuz - exceptions.py
# Task 4
# Week 5

# Exception is a built in class fior Python.
# This is a custom exception for student IDs that are invalid.
class InvalidIDException(Exception):
    def __init__(self, message="Invalid student ID. "): # This creates a constructor method with a default message for exception handling.
        super().__init__(message) # This calls the message.

# This is a custom exception for student IDs that are duplicates.
class DuplicateStudentIDException(Exception):
    def __init__(self, message = "Student ID already exists. "): # This creates a constructor method with a default message for exception handling.
        super().__init__(message) # This calls the message.
```

These are custom exceptions that are used for exception handling. When an ID already exists, it will raise the DuplicateStudentIDException and if the ID is invalid or does not exist it will raise InvalidIdException error. These are just exception errors that occur when a problem that would crash the program. These help with debugging.

Part 5



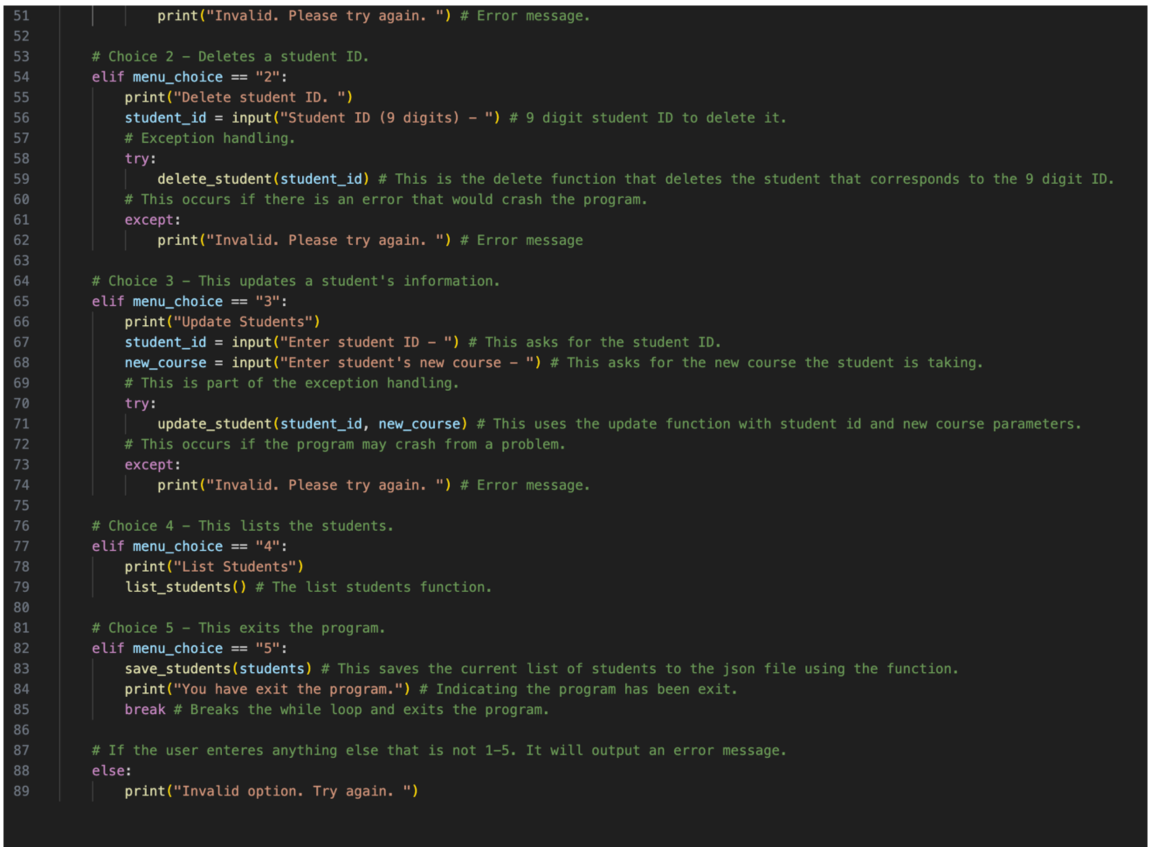
```python
# Rodin Opuz - storage.py
# Task 5
# Week 7

# This imports the json module to handle JSON files.
import json

# This imports Student class from student.py in models folder.
from models.student import Student

# This imports Undergraduate class from undergraduate.py in models folder.
from models.undergraduate import Undergraduate

# This is a function that saves the list of students to a JSON file which in this work is called students.json.
def save_students(students):
    # This is an empty list to store student information as dictionaries.
    student_list = []
    # This loops through the students in the list.
    for i in students:
        # This stores the attributes of a student as a dictionary
        student_information = {"Type": i.__class__.__name__, "Name": i.name, "ID": i.get_student_id(), "Course": i.course, "Year": i.year}

        # This loads if the student is an Undergraduate.
        if isinstance(i, Undergraduate):
            student_information["Minor"] = i.minor # If the student is an undergraduate, it adds the minor study to the dictionary.
        student_list.append(student_information) # The student information is added to the student list.

    # This opens or creates the file if it does not exist. w = Write.
    with open("students.json", 'w') as file:
        json.dump(student_list, file) # This puts the student list into JSON file.

    print("Students saved.") # Confirmation message.
```



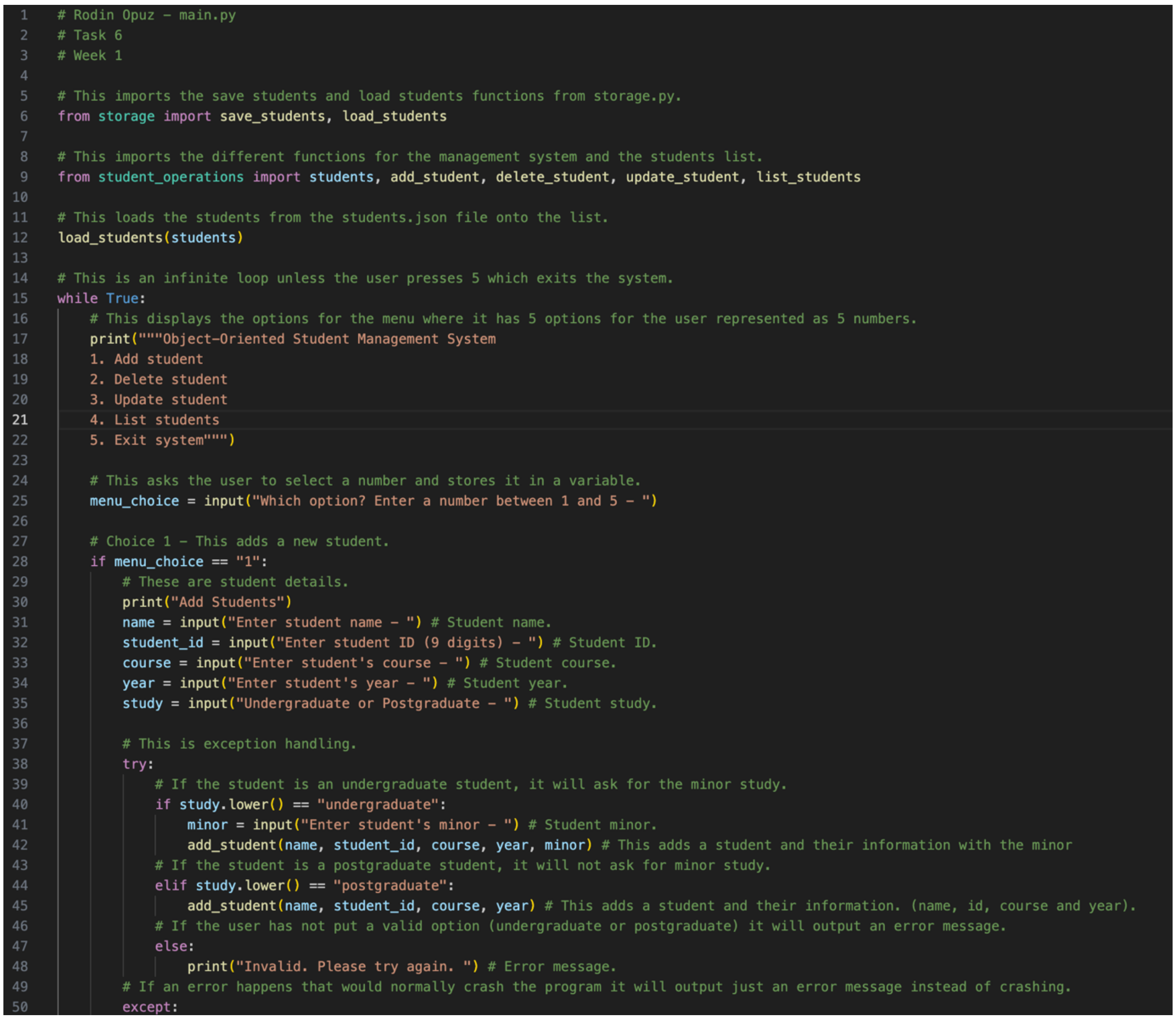
```python

# This is a function that loads the list of students from the JSON file.
def load_students(students):
    # This is exception handling.
    try:
        # Opens the JSON file students.json. r = Read.
        with open("students.json", "r") as file:
            student_list = json.load(file) # This loads the contents of the JSON file in the variable student list.
        students.clear() # This empties the list to remove student objects in case of duplicates.
        # Looops through each student in the list.
        for i in student_list:
            # If the type is Undergraduate, it adds a minor study.
            if i["Type"] == "Undergraduate":
                # Undergraduate class for a student.
                student = Undergraduate(i["Name"], i["ID"], i["Course"], i["Year"], i["Minor"])
            # If the type is Student, it adds just Name, ID, Course and Year.
            else:
                # Student class.
                student = Student(i["Name"], i["ID"], i["Course"], i["Year"])
            students.append(student) # Adds the student which is an object to the list.
    # If the program crashes, it will show an error message.
    except:
        print("Error") # Error message.
```

This manages saving and loading students to and from a JSON file. It converts student objects into dictionaries and stores them in a file (students. json). These can then represent storage of students and their information records.

Part 6



```python
                print("Invalid. Please try again. ") # Error message.

        # Choice 2 - Deletes a student ID.
        elif menu_choice == "2":
            print("Delete student ID. ")
            student_id = input("Student ID (9 digits) - ") # 9 digit student ID to delete it.
            # Exception handling.
            try:
                delete_student(student_id) # This is the delete function that deletes the student that corresponds to the 9 digit ID.
            # This occurs if there is an error that would crash the program.
            except:
                print("Invalid. Please try again. ") # Error message

        # Choice 3 - This updates a student's information.
        elif menu_choice == "3":
            print("Update Students")
            student_id = input("Enter student ID - ") # This asks for the student ID.
            new_course = input("Enter student's new course - ") # This asks for the new course the student is taking.
            # This is part of the exception handling.
            try:
                update_student(student_id, new_course) # This uses the update function with student id and new course parameters.
            # This occurs if the program may crash from a problem.
            except:
                print("Invalid. Please try again. ") # Error message.

        # Choice 4 - This lists the students.
        elif menu_choice == "4":
            print("List Students")
            list_students() # The list students function.

        # Choice 5 - This exits the program.
        elif menu_choice == "5":
            save_students(students) # This saves the current list of students to the json file using the function.
            print("You have exit the program.") # Indicating the program has been exit.
            break # Breaks the while loop and exits the program.

        # If the user enteres anything else that is not 1-5. It will output an error message.
        else:
            print("Invalid option. Try again. ")
```



```python
# Rodin Opuz - main.py
# Task 6
# Week 1

# This imports the save students and load students functions from storage.py.
from storage import save_students, load_students

# This imports the different functions for the management system and the students list.
from student_operations import students, add_student, delete_student, update_student, list_students

# This loads the students from the students.json file onto the list.
load_students(students)

# This is an infinite loop unless the user presses 5 which exits the system.
while True:
    # This displays the options for the menu where it has 5 options for the user represented as 5 numbers.
    print("""Object-Oriented Student Management System
    1. Add student
    2. Delete student
    3. Update student
    4. List students
    5. Exit system""")

    # This asks the user to select a number and stores it in a variable.
    menu_choice = input("Which option? Enter a number between 1 and 5 - ")

    # Choice 1 - This adds a new student.
    if menu_choice == "1":
        # These are student details.
        print("Add Students")
        name = input("Enter student name - ") # Student name.
        student_id = input("Enter student ID (9 digits) - ") # Student ID.
        course = input("Enter student's course - ") # Student course.
        year = input("Enter student's year - ") # Student year.
        study = input("Undergraduate or Postgraduate - ") # Student study.

        # This is exception handling.
        try:
            # If the student is an undergraduate student, it will ask for the minor study.
            if study.lower() == "undergraduate":
                minor = input("Enter student's minor - ") # Student minor.
                add_student(name, student_id, course, year, minor) # This adds a student and their information with the minor
            # If the student is a postgraduate student, it will not ask for minor study.
            elif study.lower() == "postgraduate":
                add_student(name, student_id, course, year) # This adds a student and their information. (name, id, course and year).
            # If the user has not put a valid option (undergraduate or postgraduate) it will output an error message.
            else:
                print("Invalid. Please try again. ") # Error message.
        # If an error happens that would normally crash the program it will output just an error message instead of crashing.
        except:
```

This is the interface for the student management system which allows users to add, delete, update and list students with a menu. It loads data from students.json and saves any changes when the user exits the program. There are exception handling to ensure the program does not crash and runs well.

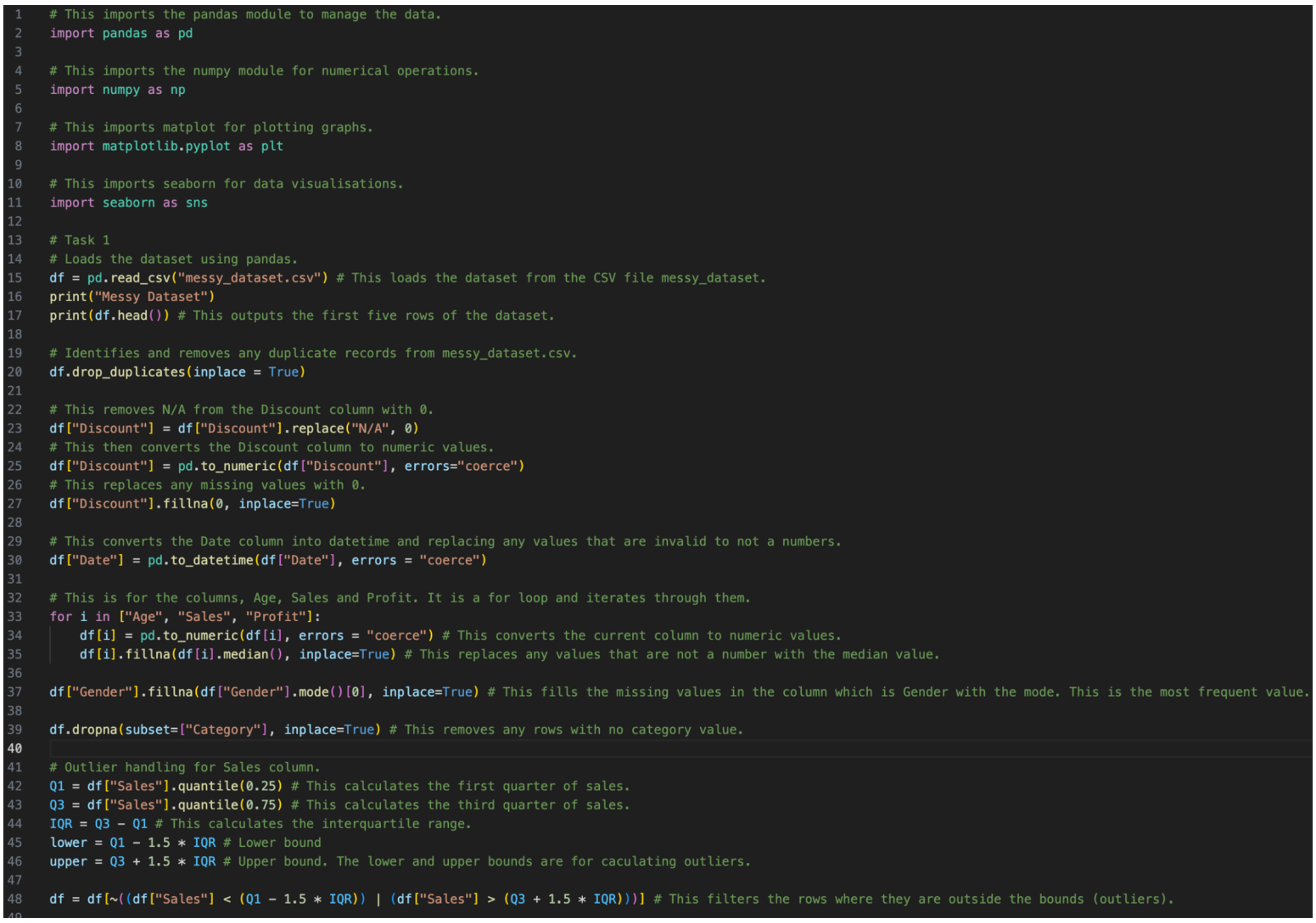
Challenges faced

I faced some problems with saving the records to a JSON file. I found it quite difficult until I watched some YouTube videos and looked at the IN0009 presentations for other information.

File managing was quite difficult in general but once I figured the basics through tutorials and other resources it went quite well.

Part 2 Project

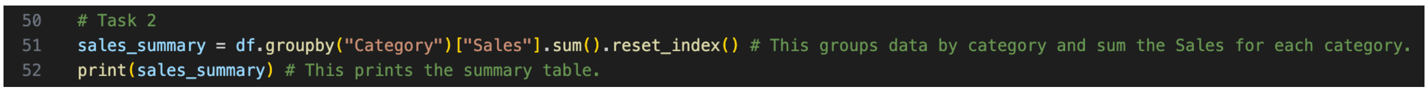
Task 1



```python
# This imports the pandas module to manage the data.
import pandas as pd

# This imports the numpy module for numerical operations.
import numpy as np

# This imports matplot for plotting graphs.
import matplotlib.pyplot as plt

# This imports seaborn for data visualisations.
import seaborn as sns

# Task 1
# Loads the dataset using pandas.
df = pd.read_csv("messy_dataset.csv") # This loads the dataset from the CSV file messy_dataset.
print("Messy Dataset")
print(df.head()) # This outputs the first five rows of the dataset.

# Identifies and removes any duplicate records from messy_dataset.csv.
df.drop_duplicates(inplace = True)

# This removes N/A from the Discount column with 0.
df["Discount"] = df["Discount"].replace("N/A", 0)
# This then converts the Discount column to numeric values.
df["Discount"] = pd.to_numeric(df["Discount"], errors="coerce")
# This replaces any missing values with 0.
df["Discount"].fillna(0, inplace=True)

# This converts the Date column into datetime and replacing any values that are invalid to not a numbers.
df["Date"] = pd.to_datetime(df["Date"], errors = "coerce")

# This is for the columns, Age, Sales and Profit. It is a for loop and iterates through them.
for i in ["Age", "Sales", "Profit"]:
    df[i] = pd.to_numeric(df[i], errors = "coerce") # This converts the current column to numeric values.
    df[i].fillna(df[i].median(), inplace=True) # This replaces any values that are not a number with the median value.

df["Gender"].fillna(df["Gender"].mode()[0], inplace=True) # This fills the missing values in the column which is Gender with the mode. This is the most frequent value.

df.dropna(subset=["Category"], inplace=True) # This removes any rows with no category value.


# Outlier handling for Sales column.
Q1 = df["Sales"].quantile(0.25) # This calculates the first quarter of sales.
Q3 = df["Sales"].quantile(0.75) # This calculates the third quarter of sales.
IQR = Q3 - Q1 # This calculates the interquartile range.
lower = Q1 - 1.5 * IQR # Lower bound
upper = Q3 + 1.5 * IQR # Upper bound. The lower and upper bounds are for caculating outliers.

df = df[~((df["Sales"] < (Q1 - 1.5 * IQR)) | (df["Sales"] > (Q3 + 1.5 * IQR)))] # This filters the rows where they are outside the bounds (outliers).
```

This involves removing duplicates, replacing missing or invalid values with central modes of tendencies, columns converting to correct format, removing rows where there were missing categories, and outliers which were extreme values.

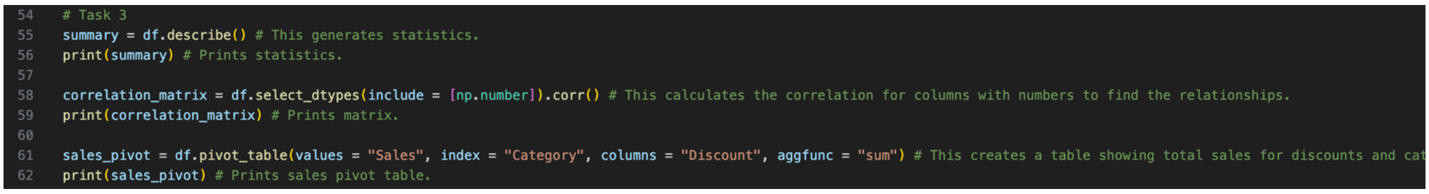
Task 2



```python
# Task 2
sales_summary = df.groupby("Category")["Sales"].sum().reset_index() # This groups data by category and sum the Sales for each category.
print(sales_summary) # This prints the summary table.
```

The dataset was grouped by the category column and calculates total sales for each category. Then a summary table generates which allows for comparison between categories.

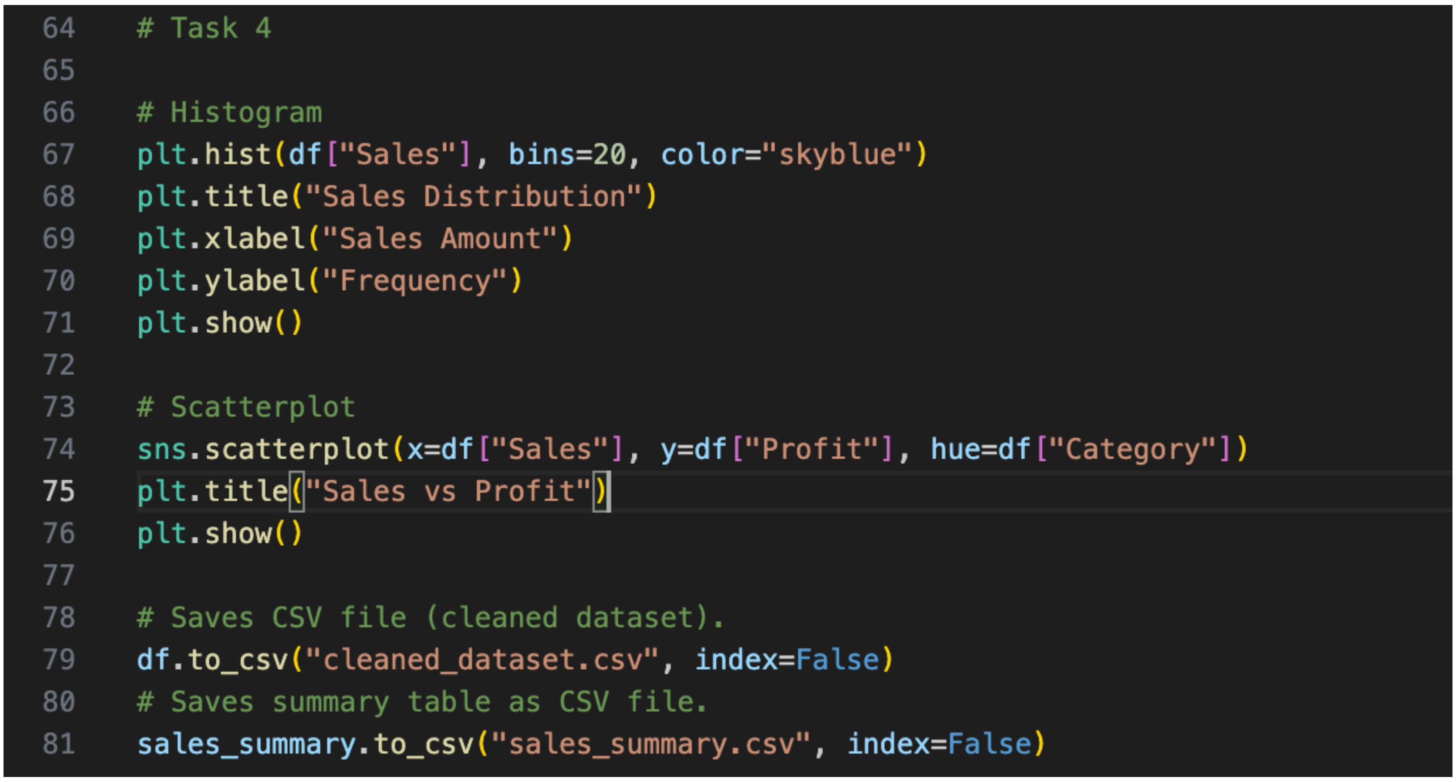
Task 3



```python
# Task 3
summary = df.describe() # This generates statistics.
print(summary) # Prints statistics.

correlation_matrix = df.select_dtypes(include = [np.number]).corr() # This calculates the correlation for columns with numbers to find the relationships.
print(correlation_matrix) # Prints matrix.

sales_pivot = df.pivot_table(values = "Sales", index = "Category", columns = "Discount", aggfunc = "sum") # This creates a table showing total sales for discounts and cat
print(sales_pivot) # Prints sales pivot table.
```

These are statistics which provide a summary for columns. It includes a correlation matrix to find the relationship and correlation between numerical values. A pivot table is also generated which shows the variation of sales.

Task 4



```python
# Task 4

# Histogram
plt.hist(df["Sales"], bins=20, color="skyblue")
plt.title("Sales Distribution")
plt.xlabel("Sales Amount")
plt.ylabel("Frequency")
plt.show()

# Scatterplot
sns.scatterplot(x=df["Sales"], y=df["Profit"], hue=df["Category"])
plt.title("Sales vs Profit")
plt.show()

# Saves CSV file (cleaned dataset).
df.to_csv("cleaned_dataset.csv", index=False)
# Saves summary table as CSV file.
sales_summary.to_csv("sales_summary.csv", index=False)
```

This creates a histogram t show the distribution of sales and a scatterplot to show the relationship between sales and profits with categories. This finds a relationship and correlation with the data. The cleaned dataset has around 3900 rows of data.

Challenges faced

I had a lot of trouble with trying to fix the missing values because there were so many pieces of data and columns which was very confusing. However, I watched more YouTube tutorials and looked at the presentations from different weeks as recommended by the assessment description which was very useful. I also had trouble with the inconsistent data types.

Future Improvements

Due to time constraints, not everything I wanted to implement was possible. Including this report as if I had additional time, I would be able to go more into depth.

For project 1, I would want to implement an actual GUI possibly using Tkinter to implement it. I would also want to add a file to track all the events that occur such as deletes, updates, adds, lists, and any errors to improve debugging

For project 2, I would have liked to do the optional tasks if I had additional time. That is how I would improve project 2 as even though I did the required tasks, it would have been more successful if I had time to complete the optional tasks

Thank you for your time