Documentation

Task	Status
Load and process review data	Completed
View all reviews for a specific park	Completed
Count reviews from a specific location	Completed
Calculate average rating for a park in a specific year	Completed
Display average score per park by location	Completed
Export park data to text, CSV, and JSON formats	Completed
Visualize data with pie charts and bar charts	Completed
Show top 10 locations by average rating for a park	Completed
Display average rating by month for a park	Completed
Conform to PEP8 standards	Completed
Implement a class diagram for OOP	Completed

Main.py

```
import process
import visual
import tui


def main():\

    filename = 'disneyland_reviews.csv'
    data = process.load_data(filename)

    while True:
        choice = tui.display_main_menu()

        if choice == 'A':
            process_viewer = process.DataViewer(data)
            process_viewer.handle_sub_menu_a()
        elif choice == 'B':
            visualizer = visual.DataVisualizer(data)
            visualizer.handle_sub_menu_b()
        elif choice == 'C':
            park_name = tui.get_park_name()
            format_choice = tui.get_export_format()
            filename = tui.get_filename()
            exporter = process.ParkDataExporter(data, park_name)
```

```
            if format_choice == 'txt':
                exporter.export_to_txt(filename)
            elif format_choice == 'csv':
                exporter.export_to_csv(filename)
            elif format_choice == 'json':
                exporter.export_to_json(filename)
            else:
                tui.display_error("Invalid format choice. Please try again.")
        elif choice == 'D':
            tui.display_message("Exiting the program.")
            break
        else:
            tui.display_error("Invalid choice. Please try again.")


if __name__ == "__main__":
    main()
```

Process.py:

```
import csv
import json
import tui


class DataViewer:
    def __init__(self, data):
        self.data = data

    def display_reviews_for_park(self, park):
        """Display all reviews for a specific park."""
        park_reviews = [review for review in self.data if review['Branch'] ==
park]
        if not park_reviews:
            tui.display_message(f"No reviews found for park: {park}")
        else:
            tui.display_message(f"Displaying reviews for {park}:")
            for review in park_reviews:
                tui.display_message(
                    f"Review ID: {review['Review_ID']}, Rating:
{review['Rating']}, "
                    f"Reviewer: {review['Reviewer_Location']}"
                )

    def count_reviews_from_location(self, park, location):
        """Count and display the number of reviews from a specific
location."""
        count = sum(
            1 for review in self.data if review['Branch'] == park and
review['Reviewer_Location'] == location
        )
        tui.display_message(f"Number of reviews from {location} for {park}:
{count}")

    def average_rating_for_year(self, park, year):
```

```
        """Calculate and display the average rating for a park in a specific
year."""
        reviews = [
            review for review in self.data if review['Branch'] == park and
review['Year_Month'].startswith(year)
        ]
        if not reviews:
            tui.display_message(f"No reviews found for {park} in year
{year}.")
            return

        total_rating = sum(int(review['Rating']) for review in reviews)
        average_rating = total_rating / len(reviews)
        tui.display_message(f"Average rating for {park} in {year}:
{average_rating:.2f}")

    def average_score_per_park_by_location(self):
        """Display the average score per park by reviewer location."""
        scores_by_location = {}

        for review in self.data:
            park = review['Branch']
            location = review['Reviewer_Location']
            rating = int(review['Rating'])

            if park not in scores_by_location:
                scores_by_location[park] = {}
            if location not in scores_by_location[park]:
                scores_by_location[park][location] = []
            scores_by_location[park][location].append(rating)

        for park, locations in scores_by_location.items():
            tui.display_message(f"\n{park}:")
            for location, scores in locations.items():
                average_score = sum(scores) / len(scores)
                tui.display_message(f"Location: {location}, Average Rating:
{average_score:.2f}")

    def handle_sub_menu_a(self):
        """Handle the sub-menu for viewing data (Option A)."""
        while True:
            choice = tui.display_sub_menu_a()

            if choice == '1':
                park = tui.get_park_name()
                self.display_reviews_for_park(park)
            elif choice == '2':
                park = tui.get_park_name()
                location = tui.get_location_name()
                self.count_reviews_from_location(park, location)
            elif choice == '3':
                park = tui.get_park_name()
                year = tui.get_year()
                self.average_rating_for_year(park, year)
            elif choice == '4':
                self.average_score_per_park_by_location()
            elif choice == '5':
```

```
                break
            else:
                tui.display_error("Invalid choice. Please try again.")


class ParkDataExporter:
    def __init__(self, data, park):
        self.data = data
        self.park = park
        self.aggregated_data = self.aggregate_data()

    def aggregate_data(self):
        total_reviews = 0
        positive_reviews = 0
        total_score = 0
        countries = set()

        for review in self.data:
            if review['Branch'] == self.park:
                total_reviews += 1
                rating = int(review['Rating'])
                if rating >= 4:
                    positive_reviews += 1
                total_score += rating
                countries.add(review['Reviewer_Location'])

        if total_reviews > 0:
            average_score = total_score / total_reviews
        else:
            average_score = 0

        return {
            "Total Reviews": total_reviews,
            "Positive Reviews": positive_reviews,
            "Average Score": average_score,
            "Unique Countries": len(countries)
        }

    def export_to_txt(self, filename):
        with open(filename, 'w') as f:
            for key, value in self.aggregated_data.items():
                f.write(f"{key}: {value}\n")

    def export_to_csv(self, filename):
        with open(filename, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(self.aggregated_data.keys())
            writer.writerow(self.aggregated_data.values())

    def export_to_json(self, filename):
        with open(filename, 'w') as f:
            json.dump(self.aggregated_data, f, indent=4)


def load_data(filename):
    """Load data from a CSV file into a list of dictionaries."""
    try:
```

```
        with open(filename, newline='', encoding='utf-8') as csvfile:
            reader = csv.DictReader(csvfile)
            data = [row for row in reader]
        tui.display_message(f"Dataset loaded successfully. Total rows:
{len(data)}")
        return data
    except FileNotFoundError:
        tui.display_error(f"The file '{filename}' was not found.")
        return []
```

Tui.py

```
def display_main_menu():
    """Display the main menu and return the user's choice."""
    print("Main Menu")
    print("A. View data")
    print("B. Visualize data")
    print("C. Export Park Data")
    print("D. Exit")
    choice = input("Enter your choice: ")
    return choice.upper()


def display_sub_menu_a():
    """Display sub-menu for viewing data and return the user's choice."""
    print("\nSub-Menu A: View Data")
    print("1. View all reviews for a specific park")
    print("2. Count reviews from a specific location")
    print("3. Average rating for a park in a given year")
    print("4. Display average score per park by reviewer location")
    print("5. Return to main menu")
    return input("Enter your choice: ").strip()


def display_sub_menu_b():
    """Display sub-menu for visualizing data and return the user's choice."""
    print("\nSub-Menu B: Visualize Data")
    print("1. Pie chart of reviews by park")
    print("2. Bar chart of average review scores")
    print("3. Top 10 locations for a park")
    print("4. Average rating by month")
    print("5. Return to main menu")
    return input("Enter your choice: ").strip()


def display_message(message):
    """Display a message to the user."""
    print(message)


def display_error(message):
    """Display an error message to the user."""
    print(f"Error: {message}")
```

```
def get_park_name():
    """Prompt the user to enter a park name."""
    return input("Enter park name: ").strip()


def get_location_name():
    """Prompt the user to enter a reviewer location."""
    return input("Enter reviewer location: ").strip()


def get_year():
    """Prompt the user to enter a year."""
    return input("Enter year (e.g., 2023): ").strip()


def get_export_format():
    """Prompt the user to enter the export format."""
    return input("Enter the format (txt, csv, json): ").lower()


def get_filename():
    """Prompt the user to enter the filename."""
    return input("Enter the filename: ")
```

Visual.py

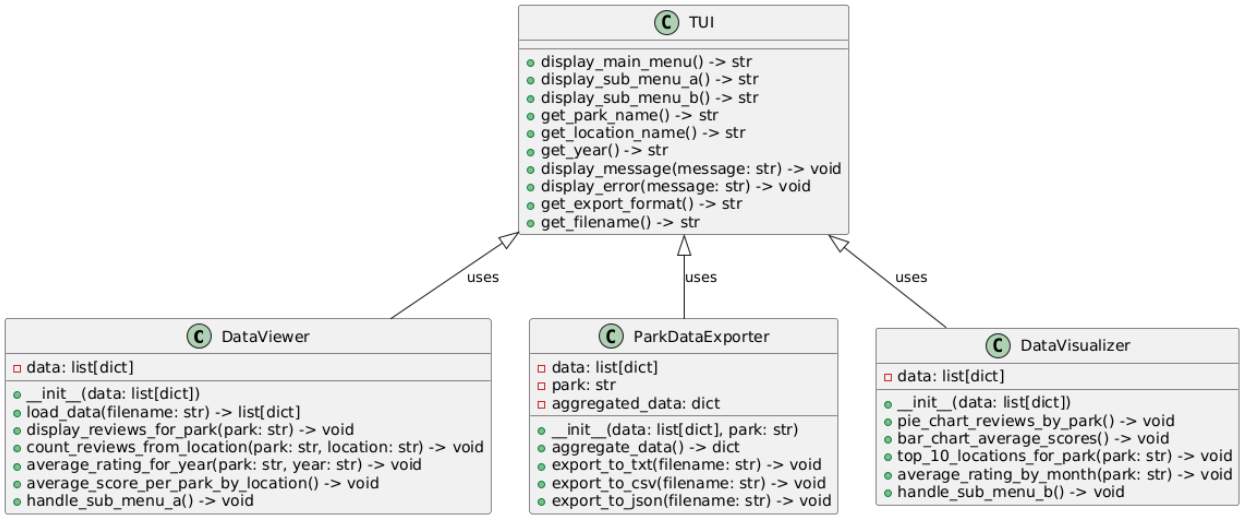
```
import calendar
import matplotlib.pyplot as plt
from collections import Counter
from datetime import datetime
import tui


class DataVisualizer:
    def __init__(self, data):
        self.data = data

    def pie_chart_reviews_by_park(self):
        """Display a pie chart showing the number of reviews for each
park."""
        park_counts = Counter(review['Branch'] for review in self.data)
        parks = list(park_counts.keys())
        counts = list(park_counts.values())

        plt.figure(figsize=(8, 8))
        plt.pie(counts, labels=parks, autopct='%1.1f%%', startangle=140)
        plt.title('Number of Reviews by Park')
        plt.axis('equal')
        plt.show()

    def bar_chart_average_scores(self):
        """Display a bar chart of average review scores for each park."""
        park_scores = {}

        for review in self.data:
```

```
            park = review['Branch']
            rating = int(review['Rating'])
            if park not in park_scores:
                park_scores[park] = []
            park_scores[park].append(rating)

        parks = list(park_scores.keys())
        averages = [sum(scores) / len(scores) for scores in
park_scores.values()]

        plt.figure(figsize=(10, 6))
        plt.bar(parks, averages, color='skyblue')
        plt.title('Average Review Scores by Park')
        plt.xlabel('Park')
        plt.ylabel('Average Score')
        plt.ylim(0, 5)

        plt.xticks(rotation=45)
        plt.show()

    def top_10_locations_for_park(self, park):
        """Display a bar chart of the top 10 locations with the highest
average rating for a park."""
        location_scores = {}

        for review in self.data:
            if review['Branch'] == park:
                location = review['Reviewer_Location']
                rating = int(review['Rating'])
                if location not in location_scores:
                    location_scores[location] = []
                location_scores[location].append(rating)

        average_scores = {location: sum(scores) / len(scores) for location,
scores in location_scores.items()}
        top_10 = sorted(average_scores.items(), key=lambda x: x[1],
reverse=True)[:10]

        locations, scores = zip(*top_10)

        plt.figure(figsize=(12, 6))
        plt.bar(locations, scores, color='orange')
        plt.title(f'Top 10 Locations by Average Rating for {park}')
        plt.xlabel('Location')
        plt.ylabel('Average Rating')
        plt.ylim(0, 5)
        plt.xticks(rotation=45)
        plt.show()

    def average_rating_by_month(self, park):
        """Display a bar chart showing the average rating by month for a
park."""
        monthly_scores = {month: [] for month in range(1, 13)}  # Initialize
dictionary for months

        for review in self.data:
            if review['Branch'] == park:
```

```
                year_month = review['Year_Month']

                # Skip records with invalid or missing Year_Month
                if not year_month or '-' not in year_month:
                    continue

                try:
                    # Ensure the date is valid
                    date_obj = datetime.strptime(year_month, "%Y-%m")
                except ValueError:
                    # Skip invalid dates
                    continue

                rating = int(review['Rating'])
                month = date_obj.month  # Extract month
                monthly_scores[month].append(rating)

        # Calculate averages
        averages = []
        for month in range(1, 13):
            if monthly_scores[month]:
                avg_rating = sum(monthly_scores[month]) /
len(monthly_scores[month])
            else:
                avg_rating = 0
            averages.append(avg_rating)

        # Prepare months for x-axis
        months = [calendar.month_name[i] for i in range(1, 13)]

        # Plot the bar chart
        plt.figure(figsize=(15, 6))
        plt.bar(months, averages, color='skyblue')
        plt.title(f'Average Rating by Month for {park}')
        plt.xlabel('Month')
        plt.ylabel('Average Rating')
        plt.ylim(0, 5)

        plt.grid(True, axis='y', linestyle='--', linewidth=0.7)
        plt.xticks(rotation=45)
        plt.show()

    def handle_sub_menu_b(self):
        """Handle the sub-menu for visualizing data (Option B)."""
        while True:
            choice = tui.display_sub_menu_b()

            if choice == '1':
                self.pie_chart_reviews_by_park()
            elif choice == '2':
                self.bar_chart_average_scores()
            elif choice == '3':
                park = tui.get_park_name()
                self.top_10_locations_for_park(park)
            elif choice == '4':
                park = tui.get_park_name()
                self.average_rating_by_month(park)
```

```
            elif choice == '5':
                break
            else:
                tui.display_error("Invalid choice. Please try again.")
```

class Diagram for class implemented

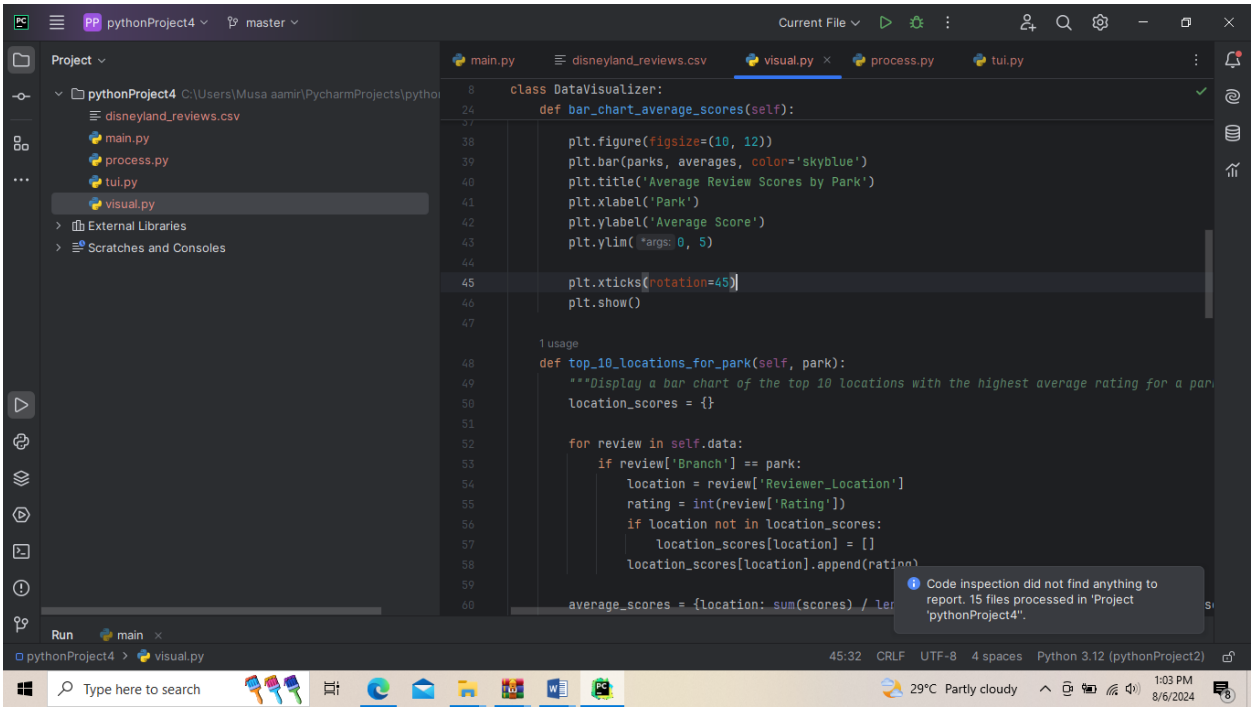


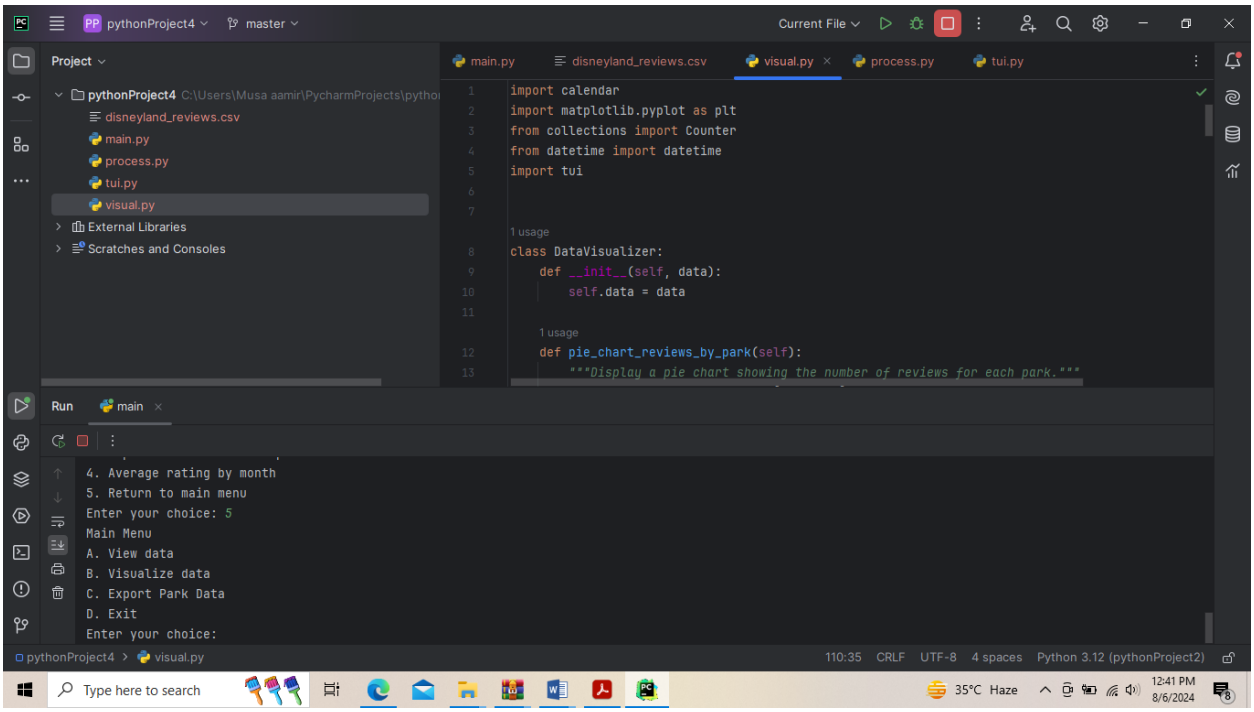
PEP8 Conformance Statement

As part of the development process, all code was checked for PEP8 compliance using PyCharm's built-in code inspection tools. The inspection revealed no PEP8 errors or warnings, indicating that the code adheres to the PEP8 standards. The following steps were taken to ensure compliance:

1. **Code Inspection**: The entire project was inspected using PyCharm's Inspect Code feature.
2. **Automatic and Manual Fixes**: Any minor issues detected during development were promptly fixed using PyCharm's quick fixes or manually, ensuring clean and readable code.

The screenshot below shows the results of the final code inspection, confirming that there are no PEP8 violations.

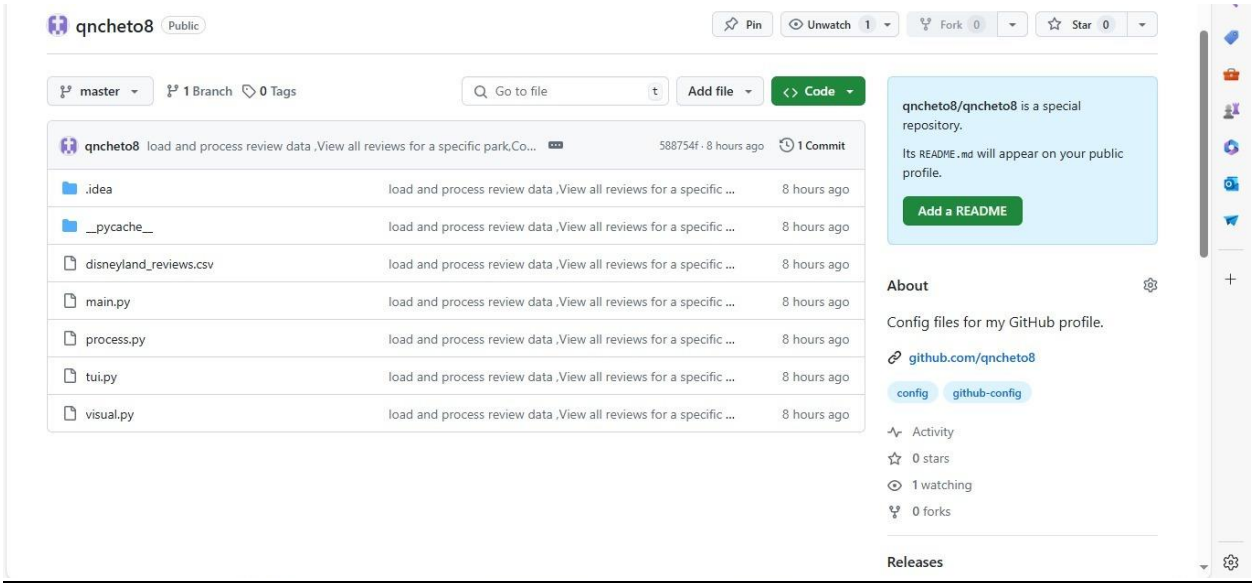




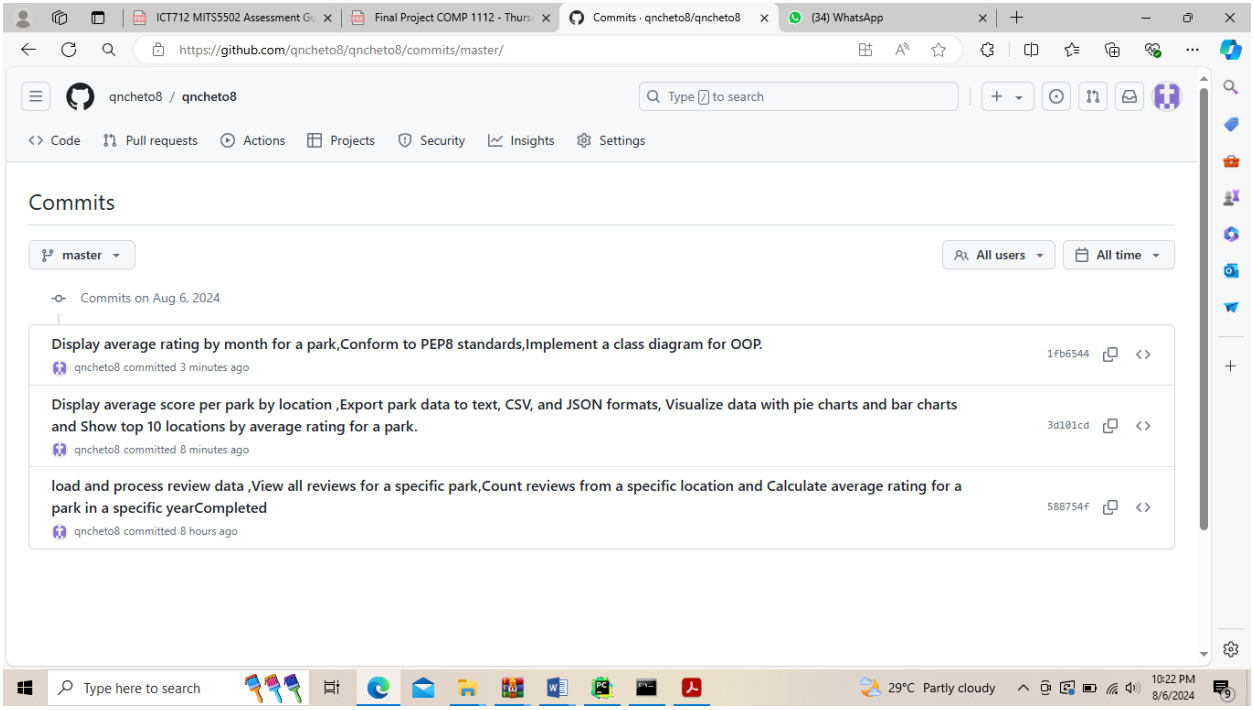
Github Evidence:

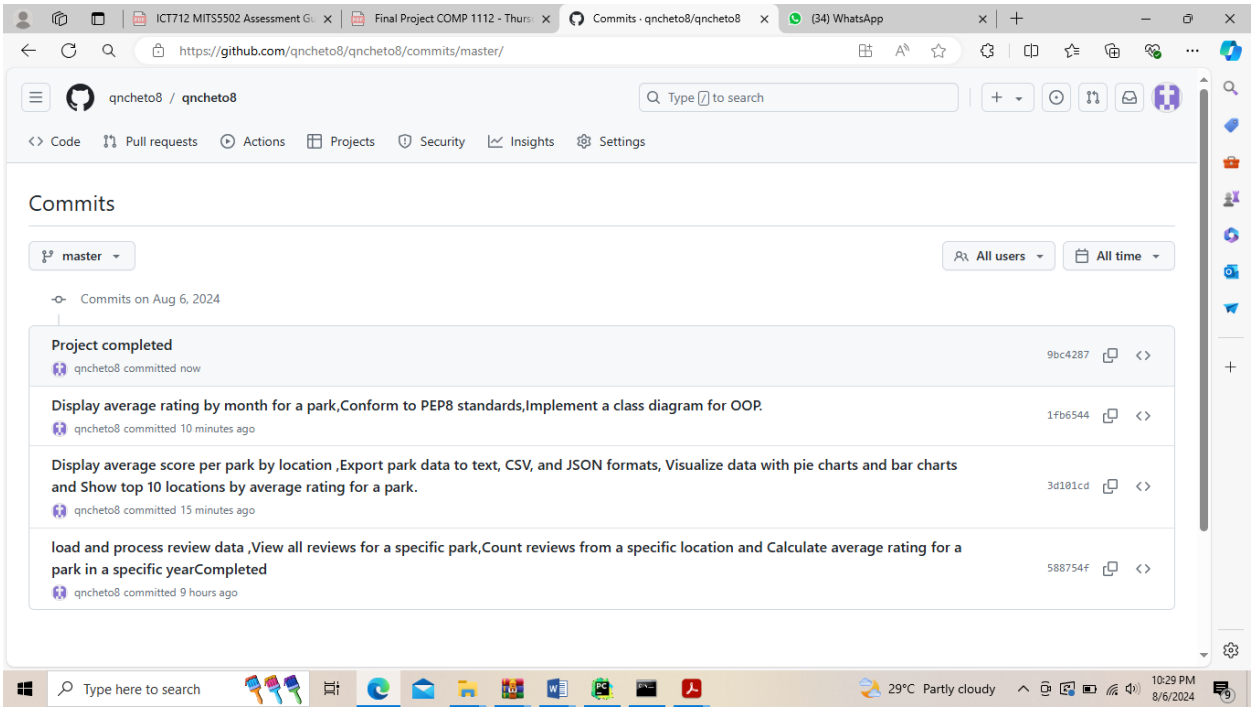
screenshot:

This screenshot is showing that code is pushed into github



This is commits history




ICT712 MITS5502 Assessment G
Final Project COMP 1112 - Thurs
Commits · qncheto8/qncheto8
(34) WhatsApp
https://github.com/qncheto8/qncheto8/commits/master/
qncheto8 / qncheto8
Type / to search
Code
Pull requests
Actions
Projects
Security
Insights
Settings
Commits
master
All users
All time
Commits on Aug 6, 2024
Project completed
qncheto8 committed now
9bc4287
Display average rating by month for a park,Conform to PEP8 standards,Implement a class diagram for OOP.
qncheto8 committed 10 minutes ago
1fb6544
Display average score per park by location ,Export park data to text, CSV, and JSON formats, Visualize data with pie charts and bar charts and Show top 10 locations by average rating for a park.
qncheto8 committed 15 minutes ago
3d101cd
load and process review data ,View all reviews for a specific park,Count reviews from a specific location and Calculate average rating for a park in a specific yearCompleted
qncheto8 committed 9 hours ago
588754f
Type here to search
29°C Partly cloudy
10:29 PM
8/6/2024