cu.png

School of Computing, Engineering and Mathematics (CEM)
Faculty of Engineering, Environment and Computing (EEC)

5011CEM BIG DATA PROGRAMMING PROJECT | 2024 PORTFOLIO OF CODE

NAME: SEAN MCHUGH | ID 12138845

April 11, 2024

Contents

1 GitHub Repository

The full repository can be found at https://github.com/Seannmc/Big-Data-Project

2 taskpandas.py

```
import pandas as pd
import matplotlib.pyplot as plt

def pandas_task1ap1(filepath):
    newDataTypes = {'County Name': 'object',
                    'Number of Trips': 'float64',
                    'Number of Trips 1-3': 'float64',
                    'Number of Trips 10-25': 'float64',
                    'Number of Trips 100-250': 'float64',
                    'Number of Trips 25-50': 'float64',
                    'Number of Trips 250-500': 'float64',
                    'Number of Trips 3-5': 'float64',
                    'Number of Trips 5-10': 'float64',
                    'Number of Trips 50-100': 'float64',
                    'Number of Trips <1': 'float64',
                    'Number of Trips >=500': 'float64',
                    'Population Not Staying at Home': 'float64',
                    'Population Staying at Home': 'float64',
                    'State Postal Code': 'object',
                    'Week': 'int64',
                    'Month': 'int64'}

    # Read the dataset into a pandas DataFrame
    df1 = pd.read_csv(filepath, dtype=newDataTypes)

    # Filtering by National level and ascending week
    df1= df1[df1['Level'] == 'National']
    df1 = df1.sort_values(by='Week')

    # Filtering unique values for columns needed by removing duplicates
    df1 = df1.drop_duplicates(subset=['Week'])

    # Removing null values
    df1 = df1.dropna(subset=['Week'])
    df1 = df1.dropna(subset=['Population Staying at Home'])

    # Grouping by week and calculating the average number of people
        staying at home
    mean_population_at_home_by_week = df1.groupby('Week')['Population
        Staying at Home'].mean()

    # Sort this average in ascending order
    sorted_mean_population_at_home_by_week =
        mean_population_at_home_by_week.sort_values()

    return mean_population_at_home_by_week,
        sorted_mean_population_at_home_by_week

def pandas_task1ap2(filepath):

    # Read the dataset into a pandas DataFrame
    df2 = pd.read_csv(filepath)
```

```
    # Extract distances from the 'Trips' columns
    distance_columns = [col for col in df2.columns if 'Trips' in col
        and col != 'Trips']

    # Calculate the mean number of people traveling each distance
    mean_people_per_distance = df2[distance_columns].mean()

    # Sort this average in ascending order
    sorted_mean_people_per_distance =
        mean_people_per_distance.sort_values()

    distance_columns = ['Trips <1 Mile', 'Trips 1-3 Miles', 'Trips 3-5
        Miles', 'Trips 5-10 Miles',
                        'Trips 10-25 Miles', 'Trips 25-50 Miles',
                            'Trips 50-100 Miles',
                        'Trips 100-250 Miles', 'Trips 250-500 Miles',
                            'Trips 500+ Miles']

    # Compute the average for each distance category
    new_mean_people_per_distance = df2[distance_columns].mean()

    # Sort this average in ascending order
    sorted_new_mean_people_per_distance =
        new_mean_people_per_distance.sort_values()

    return mean_people_per_distance, sorted_mean_people_per_distance,
        new_mean_people_per_distance, sorted_new_mean_people_per_distance

def pandas_task1b(filepath):

    # Reading data
    df3 = pd.read_csv(filepath)

    # Filtering by National level
    df3 = df3[df3['Level'] == 'National']

    # Filter the data for trips where more than 10,000,000 people
        conducted 10-25 trips
    trips_10_25 = df3[df3['Number of Trips 10-25'] > 10000000][['Date',
        'Number of Trips 10-25']]

    # Filter the data for trips where more than 10,000,000 people
        conducted 50-100 trips
    trips_50_100 = df3[df3['Number of Trips 50-100'] >
        10000000][['Date', 'Number of Trips 50-100']]

    # Returning values for graphing
    return trips_10_25, trips_50_100

def pandas_graph_task1ap1(task1ap1_result):
    mean_population_at_home_by_week,
        sorted_mean_population_at_home_by_week = task1ap1_result

    # Increasing figure size to prevent overlap
    plt.figure(figsize=(10, 12))

    # Plotting the bar chart
    plt.subplot(3, 1, 1)  # 3 rows, 1 column, subplot 1
    mean_population_at_home_by_week.plot(kind='bar', color='blue',
        zorder=2)
    plt.title('Average Number of People Staying at Home per Week Using
        Serial Processing')
```

```
    plt.xlabel('Week')
    plt.ylabel('Average Population Staying at Home')
    plt.xticks(rotation=45)
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the sorted bar chart
    plt.subplot(3, 1, 2)  # 3 rows, 1 column, subplot 2
    sorted_mean_population_at_home_by_week.plot(kind='bar',
        color='red', zorder=2)
    plt.title('Sorted average Number of People Staying at Home per Week
        Using Serial Processing')
    plt.xlabel('Week')
    plt.ylabel('Average Population Staying at Home')
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the histogram
    plt.subplot(3, 1, 3)  # 3 rows, 1 column, subplot 3
    plt.hist(mean_population_at_home_by_week, bins=10, color='blue',
        zorder=2)
    plt.title('Distribution of Average Number of People Staying at Home
        per Week Using Serial Processing')
    plt.xlabel('Average Population Staying at Home')
    plt.ylabel('Frequency')
    plt.grid(True, axis='y')  # Grid only on y-axis

    plt.subplots_adjust(hspace=0.5)

    # Show the plot
    plt.show()


def pandas_graph_task1ap2(task1ap2_result):
    mean_people_per_distance, sorted_mean_people_per_distance,
        new_mean_people_per_distance,
        sorted_new_mean_people_per_distance = task1ap2_result
    # Creating the subplots
    plt.figure(figsize=(12, 10))

    # Plotting the histogram for mean people per distance
    plt.subplot(2, 2, 1)
    plt.bar(mean_people_per_distance.index,
        mean_people_per_distance.values, color='blue', zorder=2)
    plt.title('Mean Number of People Traveling vs. Distance (Without
        Reorganised Columns ) Using Serial Processing')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the histogram for the sorted mean people per distance
    plt.subplot(2, 2, 2)
    plt.bar(sorted_mean_people_per_distance.index,
        sorted_mean_people_per_distance.values, color='red', zorder=2)
    plt.title('Mean Number of People Traveling (Without Reorganised
        Columns) in Order Using Serial Processing')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis
```

```
    # Plotting the histogram for the mean people per distance with
        amended columns
    plt.subplot(2, 2, 3)
    plt.bar(new_mean_people_per_distance.index,
        new_mean_people_per_distance.values, color='blue', zorder=2)
    plt.title('Mean Number of People Traveling (With Reorganised
        Columns) Using Serial Processing')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the histogram for the mean people per distance with
        amended columns
    plt.subplot(2, 2, 4)
    plt.bar(sorted_new_mean_people_per_distance.index,
        sorted_new_mean_people_per_distance.values, color='red',
        zorder=2)
    plt.title('Mean Number of People Traveling (With Reorganised
        Columns) in Order Using Serial Processing')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Adjust layout to prevent label overlap
    plt.tight_layout()

    # Show the plot
    plt.show()

def pandas_graph_task1b(trips_10_25, trips_50_100):
    # Create figure and axes
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    # Scatter plot for 10-25 mile trips
    axs[0].scatter(trips_10_25['Date'], trips_10_25['Number of Trips
        10-25'], color='blue')
    axs[0].set_title('10-25 Mile Trips Using Serial Processing')
    axs[0].set_xlabel('Date')
    axs[0].set_ylabel('Number of Trips')

    # Set ticks for every month
    axs[0].xaxis.set_major_locator(plt.MaxNLocator(12))  # Set maximum
        number of ticks to 12 for months

    # Scatter plot for 50-100 mile trips
    axs[1].scatter(trips_50_100['Date'], trips_50_100['Number of Trips
        50-100'], color='red')
    axs[1].set_title('50-100 Mile Trips Using Serial Processing')
    axs[1].set_xlabel('Date')
    axs[1].set_ylabel('Number of Trips')

    # Set ticks for every month
    axs[1].xaxis.set_major_locator(plt.MaxNLocator(12))  # Set maximum
        number of ticks to 12 for months

    # Adjust layout
    plt.tight_layout()
```

```
    # Show plot
    plt.show()

if __name__ == "__main__":
    # Example calls
    distance_path = "Trips_by_Distance.csv"
    trips_path = "Trips_Full Data.csv"

    # 1a
    pandas_task1ap1_result, pandas_task1ap2_result =
        pandas_task1ap1(distance_path), pandas_task1ap2(trips_path)
    # 1b
    trips_10_25, trips_50_100 = pandas_task1b(distance_path)
    print(trips_10_25)
    print(trips_50_100)

    # Graphs
    pandas_graph_task1ap1(pandas_task1ap1_result)
    pandas_graph_task1ap2(pandas_task1ap2_result)
    pandas_graph_task1b(trips_10_25, trips_50_100)
```

Code Listing 1: taskpandas.py

3 taskdask.py

```
import dask.dataframe as dd
import matplotlib.pyplot as plt
import dask
import dask.config
def display_10_entries_dask(*dfs):
    '''Function for peeking at DataFrames'''
    for i, df in enumerate(dfs):
        print(f"First 10 entries for DataFrame {i+1}:")
        print(df.head(10))

def dask_task1ap1(filepath, blocksize=322000000):
    newDataTypes = {'County Name': 'object',
                    'Number of Trips': 'float64',
                    'Number of Trips 1-3': 'float64',
                    'Number of Trips 10-25': 'float64',
                    'Number of Trips 100-250': 'float64',
                    'Number of Trips 25-50': 'float64',
                    'Number of Trips 250-500': 'float64',
                    'Number of Trips 3-5': 'float64',
                    'Number of Trips 5-10': 'float64',
                    'Number of Trips 50-100': 'float64',
                    'Number of Trips <1': 'float64',
                    'Number of Trips >=500': 'float64',
                    'Population Not Staying at Home': 'float64',
                    'Population Staying at Home': 'float64',
                    'State Postal Code': 'object',
                    'Week': 'int64',
                    'Month': 'int64'}

    # Read the dataset into a Dask DataFrame
    df1 = dd.read_csv(filepath, dtype=newDataTypes, blocksize=blocksize)

    # Filtering by National level
```

```
    df1 = df1[df1['Level'] == 'National']

    # Removing duplicate values
    df1 = df1.drop_duplicates(subset=['Week'])

    # Removing null values
    df1 = df1.dropna(subset=['Week'])
    df1 = df1.dropna(subset=['Population Staying at Home'])

    # Sorting by 'Week' column in ascending order
    df1 = df1.sort_values(by='Week')

    # Grouping by week and calculating the average number of people
        staying at home
    mean_population_at_home_by_week = df1.groupby('Week')['Population
        Staying at Home'].mean()

    # Sort this average in ascending order
    sorted_mean_population_at_home_by_week =
        mean_population_at_home_by_week.compute().sort_values()

    return mean_population_at_home_by_week,
        sorted_mean_population_at_home_by_week

def dask_task1ap2(filepath):
    # Read the dataset into a Dask DataFrame
    df2 = dd.read_csv(filepath)

    # Extract distances from the 'Trips' columns
    distance_columns = [col for col in df2.columns if 'Trips' in col
        and col != 'Trips']

    # Calculate the mean number of people traveling each distance
    mean_people_per_distance = df2[distance_columns].mean()

    # Sort this average in ascending order
    sorted_mean_people_per_distance =
        mean_people_per_distance.compute().sort_values()

    distance_columns = ['Trips <1 Mile', 'Trips 1-3 Miles', 'Trips 3-5
        Miles', 'Trips 5-10 Miles',
                        'Trips 10-25 Miles', 'Trips 25-50 Miles',
                            'Trips 50-100 Miles',
                        'Trips 100-250 Miles', 'Trips 250-500 Miles',
                            'Trips 500+ Miles']

    # Compute the average for each distance category
    new_mean_people_per_distance = df2[distance_columns].mean()

    # Sort this average in ascending order
    sorted_new_mean_people_per_distance =
        new_mean_people_per_distance.compute().sort_values()
    return mean_people_per_distance, sorted_mean_people_per_distance,
        new_mean_people_per_distance, sorted_new_mean_people_per_distance

def dask_task1b(filepath, blocksize = 322000000):

    dtype={'County Name': 'object',
       'Number of Trips': 'float64',
       'Number of Trips 1-3': 'float64',
       'Number of Trips 10-25': 'float64',
       'Number of Trips 100-250': 'float64',
```

```
        'Number of Trips 25-50': 'float64',
        'Number of Trips 250-500': 'float64',
        'Number of Trips 3-5': 'float64',
        'Number of Trips 5-10': 'float64',
        'Number of Trips 50-100': 'float64',
        'Number of Trips <1': 'float64',
        'Number of Trips >=500': 'float64',
        'Population Not Staying at Home': 'float64',
        'Population Staying at Home': 'float64',
        'State Postal Code': 'object'}

    # Reading data
    df3 = dd.read_csv(filepath, dtype=dtype, blocksize=blocksize)

    # Filtering by National level
    df3 = df3[df3['Level'] == 'National']

    # Filter the data for trips where more than 10,000,000 people
        conducted 10-25 trips
    trips_10_25 = df3[df3['Number of Trips 10-25'] > 10000000][['Date',
        'Number of Trips 10-25']]

    # Filter the data for trips where more than 10,000,000 people
        conducted 50-100 trips
    trips_50_100 = df3[df3['Number of Trips 50-100'] >
        10000000][['Date', 'Number of Trips 50-100']]

    # Returning values for graphing
    return trips_10_25, trips_50_100

def dask_graph_task1ap1(task1ap1_result):
    mean_population_at_home_by_week,
        sorted_mean_population_at_home_by_week = task1ap1_result
    # Increasing figure size to prevent overlap
    plt.figure(figsize=(10, 12))

    # Plotting the bar chart
    plt.subplot(3, 1, 1)  # 3 rows, 1 column, subplot 1
    mean_population_at_home_by_week.compute().plot(kind='bar',
        color='blue', zorder=2)
    plt.title('Average Number of People Staying at Home per Week Using
        Parallel Processing')
    plt.xlabel('Week')
    plt.ylabel('Average Population Staying at Home')
    plt.xticks(rotation=45)
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the sorted bar chart
    plt.subplot(3, 1, 2)  # 3 rows, 1 column, subplot 2
    sorted_mean_population_at_home_by_week.plot(kind='bar',
        color='red', zorder=2)
    plt.title('Sorted average Number of People Staying at Home per Week
        Using Parallel Processing')
    plt.xlabel('Week')
    plt.ylabel('Average Population Staying at Home')
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the histogram
    plt.subplot(3, 1, 3)  # 3 rows, 1 column, subplot 3
    plt.hist(mean_population_at_home_by_week.compute(), bins=10,
        color='blue', zorder=2)
```

```
    plt.title('Distribution of Average Number of People Staying at Home
        per Week Using Parallel Processing')
    plt.xlabel('Average Population Staying at Home')
    plt.ylabel('Frequency')
    plt.grid(True, axis='y')  # Grid only on y-axis

    plt.subplots_adjust(hspace=0.5)

    # Show the plot
    plt.show()

def dask_graph_task1ap2(task1ap2_result):
    mean_people_per_distance, sorted_mean_people_per_distance,
        new_mean_people_per_distance,
        sorted_new_mean_people_per_distance = task1ap2_result
    # Creating the subplots
    plt.figure(figsize=(12, 10))

    # Plotting the histogram for mean people per distance
    plt.subplot(2, 2, 1)
    mean_people_per_distance.compute().plot.bar(color='blue', zorder=2)
    plt.title('Mean Number of People Traveling vs. Distance (Without
        Reorganised Columns) Using Parallel Processing')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the histogram for the sorted mean people per distance
    plt.subplot(2, 2, 2)
    sorted_mean_people_per_distance.plot.bar(color='red', zorder=2)
    plt.title('Mean Number of People Traveling (Without Reorganised
        Columns) in Order Using Parallel Processing')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the histogram for the mean people per distance with
        amended columns
    plt.subplot(2, 2, 3)
    new_mean_people_per_distance.compute().plot.bar(color='blue',
        zorder=2)
    plt.title('Mean Number of People Traveling (With Reorganised
        Columns)')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Plotting the histogram for the mean people per distance with
        amended columns
    plt.subplot(2, 2, 4)
    sorted_new_mean_people_per_distance.plot.bar(color='red', zorder=2)
    plt.title('Mean Number of People Traveling (With Reorganised
        Columns) in Order Using Parallel Processing')
    plt.xlabel('Distance')
    plt.ylabel('Mean Number of People')
```

```python
    plt.xticks(rotation=90)  # Rotate x-axis labels for better
        visibility
    plt.grid(True, axis='y')  # Grid only on y-axis

    # Adjust layout to prevent label overlap
    plt.tight_layout()

    # Show the plot
    plt.show()

def dask_graph_task1b(trips_10_25, trips_50_100):
    # Create figure and axes
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    # Scatter plot for 10-25 mile trips
    dask.delayed(lambda trips, ax:
        trips.compute().plot.scatter(x='Date', y='Number of Trips
        10-25', color='blue', ax=ax))(
         trips_10_25, axs[0])
    axs[0].set_title('10-25 Mile Trips Using Serial Processing')
    axs[0].set_ylabel('Number of Trips')
    axs[0].set_xlabel('Date')
    axs[0].xaxis.set_major_locator(plt.MaxNLocator(12))  # Set maximum
        number of ticks to 12 for months

    # Scatter plot for 50-100 mile trips
    dask.delayed(lambda trips, ax:
        trips.compute().plot.scatter(x='Date', y='Number of Trips
        50-100', color='red', ax=ax))(
         trips_50_100, axs[1])
    axs[1].set_title('50-100 Mile Trips Using Serial Processing')
    axs[1].set_ylabel('Number of Trips')
    axs[1].set_xlabel('Date')
    axs[1].xaxis.set_major_locator(plt.MaxNLocator(12))  # Set maximum
        number of ticks to 12 for months

    # Compute the delayed operations in parallel
    dask.compute(*dask.persist(axs))

    # Explicitly compute the delayed computations
    trips_10_25, trips_50_100 = dask.compute(trips_10_25, trips_50_100)

    # Plot the computed data
    trips_10_25.plot.scatter(x='Date', y='Number of Trips 10-25',
        color='blue', ax=axs[0])
    trips_50_100.plot.scatter(x='Date', y='Number of Trips 50-100',
        color='red', ax=axs[1])

    # Adjust layout
    plt.tight_layout()

    # Show plot
    plt.show()

if __name__ == "__main__":

    distance_path = "Trips_by_Distance.csv"
    trips_path = "Trips_Full Data.csv"
    blocksize = 20000000

    # Getting returns from part 1 and part 2
```

```
    dask_task1ap1_result, dask_task1ap2_result =
        dask_task1ap1(distance_path, blocksize),
        dask_task1ap2(trips_path)

    trips_10_25, trips_50_100 = dask_task1b(distance_path)

    dask_graph_task1ap1(dask_task1ap1_result)
    dask_graph_task1ap2(dask_task1ap2_result)
    dask_graph_task1b(trips_10_25, trips_50_100)
```

Code Listing 2: taskdask.py

4 processing.py

```
import time
import matplotlib.pyplot as plt
from taskdask import dask_task1ap1, dask_task1ap2, dask_task1b
from taskpandas import pandas_task1ap1, pandas_task1ap2, pandas_task1b
from dask.distributed import Client, LocalCluster
import datetime

def execute_parallel_tasks(processor, distance_path, trips_path,
    loc_directory, cluster, client):
    # Calculating time to run parallel tasks
    start_time_parallel = time.time()
    dask_task1ap1(distance_path)
    dask_task1ap2(trips_path)
    dask_task1b(distance_path)
    parallel_time = time.time() - start_time_parallel

    # Returning parallel time
    return parallel_time


def plot_execution_times(n_processors, n_processors_time, serial_time,
    fastest_processor):
    plt.bar(n_processors, n_processors_time.values(), label='Parallel
        Time')
    plt.bar(fastest_processor, n_processors_time[fastest_processor],
        color='red', label='Fastest Processor')
    plt.axhline(y=serial_time, color='r', label='Serial Time')
    plt.xlabel('Number of Processors')
    plt.ylabel('Time (seconds)')
    plt.title('Execution Time Comparison')
    plt.legend()
    plt.grid(True)
    plt.show()

if __name__ == '__main__':
    whole_start_time = time.time()
    # Change local directory if needed
    local_directory = r'C:\Users\seanm\Desktop\Uni Work\Year 2\Term
        2\Big data project\Assesment\Code'
    distance_path = "Trips_by_Distance.csv"
    trips_path = "Trips_Full Data.csv"

    # Define the range of processor numbers
    n_processors = [10, 20]
```

```
    # Repeat the simulation 'n' times
    n = 5  # Set 'n' to the desired number of repetitions

    # Initialize cluster and client outside the loop
    cluster = LocalCluster(local_directory=local_directory,
        memory_limit='4GB')
    client = Client(cluster)

    n_processors_time = {}

    for i in range(n):
        print(f"PASS {i+1}")
        for processor in n_processors:
            if processor not in n_processors_time:
                n_processors_time[processor] = []
            parallel_time = execute_parallel_tasks(processor,
                distance_path, trips_path, local_directory, cluster,
                client)
            n_processors_time[processor].append(parallel_time)
            print(f"Parallel execution time for {processor}
                processor(s): {parallel_time} seconds")
        print("\n")
    # Calculate the average parallel time for each processor
    for processor, times in n_processors_time.items():
        n_processors_time[processor] = sum(times) / len(times)

    print("Executing tasks Serially")
    start_time_serial = time.time()
    pandas_task1ap1_result = pandas_task1ap1(distance_path)
    pandas_task1ap2_result = pandas_task1ap2(trips_path)
    pandas_task1b_result = pandas_task1b(distance_path)
    serial_time = time.time() - start_time_serial

    # Finding the fastest processor
    fastest_processor = min(n_processors_time,
        key=n_processors_time.get)
    fastest_time = n_processors_time[fastest_processor]
    print(f"\nFastest processor: {fastest_processor} processors,
        Average execution time: {fastest_time} seconds")

    print("\nAverage Parallel Times:")
    for processor, time_taken in n_processors_time.items():
        print(f"Number of Processors: {processor}, Average Parallel
            Time: {time_taken}")

    whole_runtime = time.time() - whole_start_time
    # Converting to h/m/s format
    whole_runtime = datetime.timedelta(seconds=whole_runtime)
    print(f"Total run time for program: {whole_runtime}")

    plot_execution_times(n_processors, n_processors_time, serial_time,
        fastest_processor)
```

Code Listing 3: processing.py

5 optimizing_processing.py

```
import time
```

```python
import matplotlib.pyplot as plt
from taskdask import dask_task1ap1, dask_task1ap2, dask_task1b
from dask.distributed import Client, LocalCluster
from dask.dataframe import read_csv
import datetime

def execute_loading_of_distance_dataset(filepath, blocksize):
    stime = time.time()
    df = read_csv(filepath, blocksize=blocksize)
    etime = time.time()-stime
    return df, etime

def execute_parallel_tasks(processor, distance_path, trips_path,
        loc_directory, cluster, client):
    # Calculating time to run parallel tasks
    start_time_parallel = time.time()
    dask_task1ap1(distance_path)
    dask_task1ap2(trips_path)
    dask_task1b(distance_path)
    parallel_time = time.time() - start_time_parallel

    # Returning parallel time
    return parallel_time

def plot_execution_times(n_processors, n_processors_time,
        fastest_processor):
    processors_str = [str(p) for p in n_processors]
    plt.bar(processors_str, n_processors_time.values(), label='Parallel
        Time')

    # Highlighting the fastest processor bar in red
    plt.bar(str(fastest_processor),
        n_processors_time[fastest_processor], color='red',
        label='Fastest Processor Number')

    plt.xlabel('Number of Processors')
    plt.ylabel('Time (seconds)')
    plt.title('Execution Time Comparison')
    plt.legend()
    plt.grid(True)
    plt.show()

def optimize_loading_of_distance_dataset():
    filepath = "Trips_by_Distance.csv"
    blocksize_increment = 1000000  # Increment by 1 million each pass
    min_blocksize = 300000000
    max_blocksize = 400000000
    block_sizes = list(range(min_blocksize, max_blocksize + 1,
        blocksize_increment))
    execution_times = [0] * len(block_sizes)  # Initialize list for
        accumulated times

    for _ in range(100):  # Run the loop 20 times
        print(f"PASS {_}")
        for i, blocksize in enumerate(block_sizes):
            print(f"Loading dataset with blocksize: {blocksize}")
            _, etime = execute_loading_of_distance_dataset(filepath,
                blocksize)
            execution_times[i] += etime
        print("\n")

    # Calculate the average execution times
```

```
    execution_times = [time / 100 for time in execution_times]

    # Finding the index of the fastest execution time
    best_index = execution_times.index(min(execution_times))
    best_blocksize = block_sizes[best_index]

    # Plotting block sizes and execution times
    plt.figure(figsize=(10, 6))
    plt.plot(block_sizes, execution_times, color='blue', label='Average
        Execution Time')
    plt.scatter(best_blocksize, min(execution_times), color='red',
        label=f'Fastest Block Size: {best_blocksize}')
    plt.xlabel('Block Size')
    plt.ylabel('Execution Time (seconds)')
    plt.title('Average Execution Time vs. Block Size')
    plt.grid(True)

    plt.legend()
    plt.show()

def optimize_parallel_tasks(local_dir):
    whole_start_time = time.time()
    # Change local directory if needed
    local_directory = local_dir
    distance_path = "Trips_by_Distance.csv"
    trips_path = "Trips_Full Data.csv"

    # Define the range of processor numbers
    n_processors = list(range(1, 50))
    # Repeat the simulation 'n' times
    n = 10  # Set 'n' to the desired number of repetitions

    # Initialize cluster and client outside the loop
    cluster = LocalCluster(local_directory=local_directory,
        memory_limit='4GB')
    client = Client(cluster)

    n_processors_time = {}

    for i in range(n):
        print(f"PASS {i+1}")
        for processor in n_processors:
            if processor not in n_processors_time:
                n_processors_time[processor] = []
            parallel_time = execute_parallel_tasks(processor,
                distance_path, trips_path, local_directory, cluster,
                client)
            n_processors_time[processor].append(parallel_time)
            print(f"Parallel execution time for {processor}
                processor(s): {parallel_time} seconds")
        print("\n")
    # Calculate the average parallel time for each processor
    for processor, times in n_processors_time.items():
        n_processors_time[processor] = sum(times) / len(times)

    # Finding the fastest processor
    fastest_processor = min(n_processors_time,
        key=n_processors_time.get)
    fastest_time = n_processors_time[fastest_processor]
    print(f"\nFastest processor: {fastest_processor} processors,
        Average execution time: {fastest_time} seconds")
```

```
    print("\nAverage Parallel Times:")
    for processor, time_taken in n_processors_time.items():
        print(f"Number of Processors: {processor}, Average Parallel
            Time: {time_taken}")

    whole_runtime = time.time() - whole_start_time
    # Converting to h/m/s format
    whole_runtime = datetime.timedelta(seconds=whole_runtime)
    print(f"Total run time for program: {whole_runtime}")

    plot_execution_times(n_processors, n_processors_time,
        fastest_processor)

if __name__ == '__main__':
    whole_start_time = time.time()
    # Change local directory if needed
    local_directory = r'C:\Users\seanm\Desktop\Uni Work\Year 2\Term
        2\Big data project\Assesment\Code'
    distance_path = "Trips_by_Distance.csv"
    trips_path = "Trips_Full Data.csv"
    #optimize_loading_of_distance_dataset()
    optimize_parallel_tasks(local_directory)
```

Code Listing 4: optimizing_processing.py

6 model.py

```
import dask.array as da
import dask.dataframe as dd
from sklearn.linear_model import LinearRegression
from dask_ml.model_selection import train_test_split
from dask.diagnostics import ProgressBar
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

dtype = {'County Name': 'object',
         'Number of Trips': 'float64',
         'Number of Trips 1-3': 'float64',
         'Number of Trips 10-25': 'float64',
         'Number of Trips 100-250': 'float64',
         'Number of Trips 25-50': 'float64',
         'Number of Trips 250-500': 'float64',
         'Number of Trips 3-5': 'float64',
         'Number of Trips 5-10': 'float64',
         'Number of Trips 50-100': 'float64',
         'Number of Trips <1': 'float64',
         'Number of Trips >=500': 'float64',
         'Population Not Staying at Home': 'float64',
         'Population Staying at Home': 'float64',
         'State Postal Code': 'object'}

# Importing datasets using dask
trips_by_distance = dd.read_csv('Trips_by_Distance.csv', dtype=dtype)
trips_full_data = dd.read_csv('Trips_Full Data.csv')

# Converting Date in Trips_by_Distance to m/d/y format
trips_by_distance['Date'] = dd.to_datetime(trips_by_distance['Date'])
```

```
# Filtering the dataset by year = 2019, week = 32, level = National
filtered_distance_data = trips_by_distance[
    (trips_by_distance['Date'].dt.year == 2019) &
    (trips_by_distance['Week'] == 32) &
    (trips_by_distance['Level'] == 'National')
]

# Defining columns for distance
distance_columns = [
    'Number of Trips 1-3',
    'Number of Trips 3-5',
    'Number of Trips 5-10',
    'Number of Trips 10-25'
]
# Defining columns for trips
trips_columns = ['Trips 1-25 Miles']

# Filtering datasets based on defined columns
x= filtered_distance_data[distance_columns]
y = trips_full_data[trips_columns]

# Training model
model = LinearRegression()
model.fit(x, y)
y_pred = model.predict(x)

# Calculating model information
r_sq = model.score(x, y)
intercept = model.intercept_
coefficients = model.coef_[0]
equation = f'y = {intercept:}'
for i, coef in enumerate(coefficients):
    equation += f' + {coef:} * x{i+1}'

# Displaying model information
print(f"Coefficient of determination: {r_sq}")
print(f"Intercept: {intercept}")
print(f"Coefficients: {coefficients}")
print(f"Equation: {equation}")
print(f"predicted response:\n{y_pred}")

# Plotting the predicted response against the actual values
plt.scatter(y.compute(), y_pred, color='black', marker='x')
plt.plot(y.compute(), y.compute(), color='red', linewidth=3)
plt.xlabel('Actual Trips 1-25 Miles')
plt.ylabel('Predicted Trips 1-25 Miles')
plt.title('Actual vs Predicted Trips 1-25 Miles')
plt.show()
```

Code Listing 5: model.py

7 trips_visualisation.py

```
import dask.dataframe as dd
import matplotlib.pyplot as plt

# Reading dataset
df = dd.read_csv('Trips_Full Data.csv')
```

```
# Extracting distances from the 'Trips' columns
distance_columns = [col for col in df.columns if 'Trips' in col and col
    != 'Trips']


# Defining new distance columns for mean
distance_columns = ['Trips <1 Mile', 'Trips 1-3 Miles', 'Trips 3-5
    Miles', 'Trips 5-10 Miles',
                    'Trips 10-25 Miles', 'Trips 25-50 Miles', 'Trips
                        50-100 Miles',
                    'Trips 100-250 Miles', 'Trips 250-500 Miles',
                        'Trips 500+ Miles']

# Compute the average for each distance category
df = df[distance_columns].mean()

# Plotting bar chart
plt.figure(figsize=(10, 6))
df[distance_columns].compute().plot(kind='bar', color='blue')
plt.title('Number of Participants')
plt.xlabel('Days')
plt.ylabel('Number of Participants')
plt.grid(axis='y')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels
plt.show()

# Plotting histogram
plt.figure(figsize=(10, 6))
plt.hist(df[distance_columns].compute(), bins=10, color='red',
    edgecolor='black')
plt.title('Histogram of Number of Participants')
plt.xlabel('Number of Participants')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()
```

Code Listing 6: trips_visualisation.py

References

Coventry University. (2024). *Curve fitting using regression$_s$olutions. Retrievedfrom* (Regression modelling)

Dask Development Team. (2022). *Dask documentation.* Retrieved from https://docs.dask.org/en/stable/ (Accesed for Dask enquiries)

Matplotlib Development Team. (2022). *Matplotlib documentation.* Retrieved from https://matplotlib.org/stable/index.html (Accesed for Matplotlib enquiries)

pandas development team. (2022). *pandas documentation.* Retrieved from https://pandas.pydata.org/docs/ (Accesed for Pandas enquiries)

Dask Development Team (2022) pandas development team (2022) Matplotlib Development Team (2022) Coventry University (2024)