Extension Project will address Bluebikes correlation with Bus performance.

With this, we will be using Bluebikes station and trip datasets along with previous data about bus given in Bus performance project document.

Subproblem 1: Worst on-time Performance Routes correlation with Bluebikes usage

```
import pandas as pd
import numpy as np
import matplotlib as plt
from collections import defaultdict
import os

from google.colab import drive
drive.mount('/content/MyDrive')
dir_path = '/content/MyDrive/MyDrive/City of Boston: Transit & Performance A/Data Files/deliverable_3_data'
files = os.listdir(dir_path)
```

```
Mounted at /content/MyDrive
```

```
blue_bikes_trips = pd.read_csv(os.path.join(dir_path, '202201-bluebikes-tripdata.csv'))
mbta_prediction_accuracy = pd.read_csv(os.path.join(dir_path, 'Bus_Prediction_Accuracy.csv'))
census_neighbourhood = pd.read_csv(os.path.join(dir_path, 'Census-Boston-Neighborhood.csv'))
blue_bikes_stations = pd.read_csv(os.path.join(dir_path, 'current_bluebikes_stations.csv'))
mbta_reliability = pd.read_csv(os.path.join(dir_path, 'MBTA_Bus_Reliability.csv'))
mbta_gtfs = pd.read_csv(os.path.join(dir_path, 'MBTA_Systemwide_GTFS_Map.csv'))

# folder_path = "../../data/"
# blue_bikes_trips = pd.read_csv(f"{folder_path}/202201-bluebikes-tripdata.csv")
# blue_bikes_stations = pd.read_csv(f"{folder_path}/current_bluebikes_stations.csv")
# census_neighbourhood = pd.read_csv(f"{folder_path}/Census-Boston-Neighborhood.csv")
# mbta_gtfs = pd.read_csv(f"{folder_path}/MBTA_Systemwide_GTFS_Map.csv")
# mbta_reliability = pd.read_csv(f"{folder_path}/MBTA_Bus_Reliability.csv")
# mbta_prediction_accuracy = pd.read_csv(f"{folder_path}/Bus_Prediction_Accuracy.csv")
```

```python
def process_blue_bikes_trips(blue_bikes_trips):
     processed_blue_bikes_trips = blue_bikes_trips
     processed_blue_bikes_trips["tripduration"] =
blue_bikes_trips["tripduration"] / 60
     return processed_blue_bikes_trips

processed_blue_bikes_trips =
process_blue_bikes_trips(blue_bikes_trips)
processed_blue_bikes_trips.head()
```

```
   tripduration                 starttime                  stoptime  \
0      9.950000  2022-01-01 00:00:25.1660  2022-01-01 00:10:22.1920
1      6.850000  2022-01-01 00:00:40.4300  2022-01-01 00:07:32.1980
2      7.933333  2022-01-01 00:00:54.8180  2022-01-01 00:08:51.6680
3      7.766667  2022-01-01 00:01:01.6080  2022-01-01 00:08:48.2350
4     12.533333  2022-01-01 00:01:06.0520  2022-01-01 00:13:38.2300

   start station id               start station name  start station
latitude  \
0               178  MIT Pacific St at Purrington St
42.359573
1               189                        Kendall T
42.362428
2                94             Main St at Austin St
42.375603
3                94             Main St at Austin St
42.375603
4                19            Park Dr at Buswell St
42.347241

   start station longitude  end station id  \
0               -71.101295              74
1               -71.084955             178
2               -71.064608             356
3               -71.064608             356
4               -71.105301              41

                                    end station name  end station
latitude  \
0            Harvard Square at Mass Ave/ Dunster
42.373268
1                MIT Pacific St at Purrington St
42.359573
2                          Charlestown Navy Yard
42.374125
3                          Charlestown Navy Yard
42.374125
4  Packard's Corner - Commonwealth Ave at Brighto...
42.352261
```

```
   end station longitude  bikeid    usertype postal code
0              -71.118579    4923  Subscriber       02139
1              -71.101295    3112  Subscriber       02139
2              -71.054812    6901    Customer       02124
3              -71.054812    5214    Customer       02124
4              -71.123831    2214  Subscriber       02215
```

```
# print(blue_bikes_stations.head())

def process_blue_bikes_stations(blue_bikes_stations,
blue_bikes_trips):
      # Fixing first row as the column names
      new_column_names = blue_bikes_stations.iloc[0]  # Get the first
row to use as column names
      blue_bikes_stations.columns = new_column_names  # Set new column
names
      blue_bikes_stations =
blue_bikes_stations.iloc[1:].reset_index(drop=True)

      # Extracting the unique start station names and IDs
      start_stations = blue_bikes_trips[['start station id', 'start
station name']].drop_duplicates()
      start_stations = start_stations.rename(columns={'start station
id': 'station_id', 'start station name': 'station_name'})

      # Extracting the unique end station names and IDs.
      end_stations = blue_bikes_trips[['end station id', 'end station
name']].drop_duplicates()
      end_stations = end_stations.rename(columns={'end station id':
'station_id', 'end station name': 'station_name'})

      # Combining the start and end station information.
      combined_stations = pd.concat([start_stations,
end_stations]).drop_duplicates().set_index('station_name')
      blue_bikes_stations['station_id'] =
blue_bikes_stations['Name'].map(combined_stations['station_id'])
      blue_bikes_stations = blue_bikes_stations.dropna(subset =
["station_id"])
      return blue_bikes_stations

processed_blue_bikes_stations =
process_blue_bikes_stations(blue_bikes_stations,
processed_blue_bikes_trips)
processed_blue_bikes_stations.head()
```

```
0  Number                                 Name     Latitude
Longitude  \
0  K32015                       1200 Beacon St  42.34414899  -
71.11467361
1  W32006                          160 Arsenal  42.36466403  -
```

```
71.17569387
2  A32019                     175 N Harvard St  42.36447457  -
71.12840831
3  S32035                        191 Beacon St  42.38032335  -
71.10878613
4  C32094  2 Hummingbird Lane at Olmsted Green     42.28887    -
71.095003

0    District Public Total docks Deployment Year  station_id
0   Brookline    Yes            1            2021       452.0
1   Watertown    Yes           11            2021       502.0
2      Boston    Yes           17            2014       149.0
3  Somerville    Yes           19            2018       378.0
4      Boston    Yes           17            2020       493.0
```

```
census_neighbourhood.head()
```

```
    tract20_nbhd P0020001     P0020005
P0020006  \
0  field concept   Total:  White alone  Black or African American
alone
1        Allston    24904        12536
1326
2       Back Bay    18190        13065
690
3    Beacon Hill     9336         7521
252
4       Brighton    52047        32694
2414

             P0020002
P002aapi  \
0  Hispanic or Latino  Asian, Native Hawaiian and Pacific Islander
al...
1                3259
6271
2                1208
2410
3                 537
630
4                5376
8703

                                 P002others P0040001     P0040005  \
0  Other Races or Multiple Races,  all ages   Total:  White alone
1                                      1512    23140        11976
2                                       817    17042        12349
3                                       396     8603         6980
4                                      2860    47657        30752
```

```
                             P0040006  ...  \
0  Black or African American alone  ...
1                              1184  ...
2                               641  ...
3                               231  ...
4                              2076  ...

                                          P0050005  \
0  Nursing facilities/Skilled-nursing facilities
1                                              0
2                                            269
3                                              0
4                                            266

                            P0050006                          P0050007  \
0  Other institutional facilities  Noninstitutionalized population:
1                                0                              3281
2                                0                              1610
3                                0                                33
4                               56                              3796

                                P0050008          P0050009  \
0  College/University student housing  Military quarters
1                                3214                 0
2                                1487                 0
3                                   0                 0
4                                3493                 0

                               P0050010 H0010001  H0010002 H0010003  \
0  Other noninstitutional facilities   Total:  Occupied   Vacant
1                                 67    10748     10027      721
2                                123    11524     10006     1518
3                                 33     6037      5485      552
4                                303    23653     22535     1118

           hhsize
0  household size
1     2.156477511
2     1.630121927
3     1.696080219
4     2.126292434

[5 rows x 34 columns]

mbta_gtfs.head()

def process_gtfs(MBTA_data):
     MBTA_data = MBTA_data[MBTA_data['Neighborhood'].notnull()]
     MBTA_data = MBTA_data[MBTA_data['Routes'] != '#N/A']
     MBTA_data = MBTA_data[MBTA_data['Routes'].notnull()]
```

```
    # Split routes column to separate routes
    MBTA_data['Routes'] = MBTA_data['Routes'].str.split('|')
    MBTA_data = MBTA_data.explode('Routes')
    df = MBTA_data[["stop_id", "stop_name", "stop_lat", "stop_lon",
"Neighborhood", "Routes"]]

    return df

processed_mbta_gtfs = process_gtfs(mbta_gtfs)
processed_mbta_gtfs.head()
```

```
  stop_id                    stop_name   stop_lat   stop_lon
Neighborhood  \
0       1  Washington St opp Ruggles St  42.330957 -71.082754
Roxbury
0       1  Washington St opp Ruggles St  42.330957 -71.082754
Roxbury
0       1  Washington St opp Ruggles St  42.330957 -71.082754
Roxbury
0       1  Washington St opp Ruggles St  42.330957 -71.082754
Roxbury
0       1  Washington St opp Ruggles St  42.330957 -71.082754
Roxbury

  Routes
0      1
0      8
0     10
0     47
0     19
```

```
mbta_prediction_accuracy.head()
```

```
                   weekly mode route_id        bin
arrival_departure  \
0  2021/08/13 04:00:00+00  bus      NaN    0-3 min         departure

1  2021/08/13 04:00:00+00  bus      NaN    3-6 min         departure

2  2021/08/13 04:00:00+00  bus      NaN   6-12 min         departure

3  2021/08/13 04:00:00+00  bus      NaN  12-30 min         departure

4  2021/08/20 04:00:00+00  bus      NaN    0-3 min         departure


   num_predictions  num_accurate_predictions  ObjectId
0           293039                    233562         1
1           285817                    229090         2
2           561098                    472923         3
```

```
3          1594830                         1405620         4
4           285591                          228653         5
```

```
mbta_reliability.head()

# Code taken from Base Question 2 code
def process_reliability(df):
      new_df = df[df["mode_type"]=="Bus"] # taking only buses
      new_df = new_df.dropna(subset=['otp_denominator',
'otp_numerator','cancelled_numerator']) # No NaN / Null
      new_df['ot_rate'] =
new_df['otp_numerator']/new_df['otp_denominator']
      grouped_route = new_df.groupby('gtfs_route_id')
      grouped_rate = grouped_route['ot_rate'].mean().reset_index()
      rate_sorted = grouped_rate.sort_values(by='ot_rate',
ascending=False)
      return rate_sorted


reliability_rate_sorted = process_reliability(mbta_reliability)

reliability_rate_sorted.head() # best ot_rate
reliability_rate_sorted.tail() # worst ot_rate
```

	gtfs_route_id	ot_rate
150	747	0.458202
106	459	0.429970
99	448	0.406302
100	449	0.402552
178	9703	0.320094

We have the best and worst on-time performance data extracted from base question 2 - Utilizes the MBTA Reliability Dataset:

Best 10:

image-2.png

Worst 10:

image.png

```
merged_data_on_routes = pd.merge(processed_mbta_gtfs,
reliability_rate_sorted, left_on = "Routes", right_on =
"gtfs_route_id")

print(merged_data_on_routes['gtfs_route_id'].isna().sum()) # checking
no bus routes are not included in the relability dataset.
print(merged_data_on_routes['Routes'].isna().sum()) # checking no bus
routes are not included in the GTFS dataset.
```

```
merged_data_on_routes.head()

0
0

  stop_id                          stop_name   stop_lat   stop_lon  \
0       1       Washington St opp Ruggles St  42.330957 -71.082754
1   10003           Albany St opp Randall St  42.331591 -71.076237
2   10100             Albany St @ Randall St  42.331675 -71.076347
3   10101    Melnea Cass Blvd @ Harrison Ave  42.332066 -71.079147
4   10590  Massachusetts Ave @ Washington St  42.336621 -71.076956

  Neighborhood Routes gtfs_route_id   ot_rate
0      Roxbury      1             1  0.744301
1      Roxbury      1             1  0.744301
2      Roxbury      1             1  0.744301
3      Roxbury      1             1  0.744301
4    South End      1             1  0.744301

# Group by 'Routes'
grouped_by_routes = merged_data_on_routes.groupby('Routes')

grouped_by_routes.head()

# # Aggregate 'ot_rate' for each route, then sort to find the worst 10
# # Assuming 'worst' means the highest values
worst_routes =
grouped_by_routes['ot_rate'].mean().sort_values(ascending=True)
best_routes =
grouped_by_routes['ot_rate'].mean().sort_values(ascending=False)

# # Print the worst 10 routes based on ot_rate
print(worst_routes.head(10))
# # Print the best 10 routes based on ot_rate
print(best_routes.head(10))

Routes
9703    0.320094
449     0.402552
448     0.406302
459     0.429970
747     0.458202
41      0.488934
19      0.493452
70A     0.494182
14      0.509825
701     0.515090
Name: ot_rate, dtype: float64
Routes
742    0.837185
```

```
502    0.813195
32     0.807782
749    0.807251
111    0.803600
751    0.801902
741    0.801389
746    0.800187
7      0.792366
31     0.786380
Name: ot_rate, dtype: float64
```

```
worst_routes_loc = pd.merge(worst_routes, merged_data_on_routes,
left_on = ["Routes", "ot_rate"], right_on = ["Routes", "ot_rate"])
worst_routes_loc.rename(columns={"Routes": "route"}, inplace=True)
worst_routes_loc.head(25) # rows are per stop, so showing more rows
ensures the visibility of other routes here beyond route 9703
# print(worst_routes_loc.shape)
```

```
   route   ot_rate stop_id
stop_name  \
0   9703  0.320094    1111                      Cambridge St opp Hano
St
1   9703  0.320094    1112                     Cambridge St @ Harvard
St
2   9703  0.320094    1113                      Cambridge St @ Linden
St
3   9703  0.320094    1114                   Cambridge St @ N Harvard
St
4   9703  0.320094   11388                  Huntington Ave @ Belvidere
St
5   9703  0.320094    1257                      Tremont St @ Prentiss
St
6   9703  0.320094    1258          Tremont St @ Roxbury Crossing
Station
7   9703  0.320094    1260                   Columbus Ave @ New Cedar
St
8   9703  0.320094    1262                       Columbus Ave @ Heath
St
9   9703  0.320094    1784                    Ruggles St @ Huntington
Ave
10  9703  0.320094    1785                  Ruggles St @ Annunciation
Rd
11  9703  0.320094   31391               Huntington Ave @ Gainsborough
St
12  9703  0.320094   41391                      Huntington Ave @ Opera
Pl
13  9703  0.320094   61391                    Huntington Ave @ Forsyth
Way
14  9703  0.320094   71391                Huntington Ave @ Louis Prang
St
```

```
15  9703  0.320094     922                      Cambridge St opp Dustin
St
16  9703  0.320094     924                        Cambridge St @ Gordon
St
17  9703  0.320094     925                       Cambridge St @ Barrows
St
18   448  0.406302   16535                             Otis St @ Summer
St
19   448  0.406302    4727                  McClellan Highway @ Addison
St
20   448  0.406302    4728                 McClellan Highway @ Boardman
St
21   448  0.406302    6535                     Franklin St @ Devonshire
St
22   448  0.406302    6564  Summer St @ South Station - Red Line
entrance
23   448  0.406302    7094                   Terminal C - Departures
Level
24   448  0.406302     892                        Summer St @ Atlantic
Ave

     stop_lat   stop_lon  Neighborhood gtfs_route_id
0   42.353931 -71.136365       Allston          9703
1   42.355641 -71.132361       Allston          9703
2   42.355943 -71.131448       Allston          9703
3   42.357758 -71.126505       Allston          9703
4   42.345344 -71.082045      Back Bay          9703
5   42.332930 -71.092638       Roxbury          9703
6   42.331311 -71.094831       Roxbury          9703
7   42.328067 -71.097310       Roxbury          9703
8   42.325028 -71.098483       Roxbury          9703
9   42.337416 -71.095079  Mission Hill          9703
10  42.336729 -71.093223  Mission Hill          9703
11  42.341443 -71.086788        Fenway          9703
12  42.340553 -71.088908        Fenway          9703
13  42.339219 -71.092168        Fenway          9703
14  42.337684 -71.096046        Fenway          9703
15  42.350692 -71.145688       Allston          9703
16  42.352276 -71.140761       Allston          9703
17  42.353091 -71.138430       Allston          9703
18  42.354243 -71.058557      Downtown           448
19  42.386142 -71.019171   East Boston           448
20  42.391562 -71.012888   East Boston           448
21  42.355521 -71.057253      Downtown           448
22  42.352253 -71.054774      Downtown           448
23  42.366635 -71.017167   East Boston           448
24  42.352480 -71.054849      Downtown           448
```

From here, we will be comparing locations of bus stations of the worst routes and the locations of bluebikes going along those routes. We will then see the average number of rides in that station.

```
best_routes_loc = pd.merge(best_routes, merged_data_on_routes, left_on
= ["Routes", "ot_rate"], right_on = ["Routes", "ot_rate"])
best_routes_loc.rename(columns={"Routes": "route"}, inplace=True)
best_routes_loc.head(25) # rows are per stop, so showing more rows
ensures the visibility of other routes here beyond route 9703
# print(worst_routes_loc.shape)
```

```
   route   ot_rate stop_id                         stop_name
stop_lat  \
0    502  0.813195     178     Saint James Ave @ Dartmouth St
42.349505
1    502  0.813195   71855           Stuart St @ Dartmouth St
42.348245
2     32  0.807782   10522               Circuit Dr @ Glen Ln
42.305104
3     32  0.807782   11131             Centre St @ Roseway St
42.318993
4     32  0.807782    1128             South St @ Sedgwick St
42.308588
5     32  0.807782    1129           Centre St @ Seaverns Ave
42.312198
6     32  0.807782    1130          Centre St @ Saint John St
42.314462
7     32  0.807782    1132          Centre St opp Beaufort Rd
42.316493
8     32  0.807782   11587               Circuit Dr @ Glen Ln
42.305236
9     32  0.807782   11780   Ave Louis Pasteur @ Longwood Ave
42.337969
10    32  0.807782    1315   Huntington Ave @ Parker Hill Ave
42.333092
11    32  0.807782    1317      Huntington Ave opp Fenwood Rd
42.333494
12    32  0.807782    1319     Tremont St opp Wigglesworth St
42.333785
13    32  0.807782    1325           Humboldt Ave @ Seaver St
42.310100
14    32  0.807782    1326        Humboldt Ave @ Hutchings St
42.311245
15    32  0.807782    1327        Humboldt Ave @ Homestead St
42.311784
16    32  0.807782    1328         Humboldt Ave @ Crawford St
42.313354
17    32  0.807782    1330         Humboldt Ave @ Waumbeck St
42.314496
18    32  0.807782    1331          Humboldt Ave @ Wyoming St
```

```
42.316043
19    32  0.807782    1332         Humboldt Ave @ Townsend St
42.316934
20    32  0.807782    1346         Humboldt Ave @ Townsend St
42.317134
21    32  0.807782    1350         Humboldt Ave @ Waumbeck St
42.314699
22    32  0.807782    1351         Humboldt Ave @ Crawford St
42.313142
23    32  0.807782    1352       Humboldt Ave @ Homestead St
42.311990
24    32  0.807782    1353       Humboldt Ave @ Hutchings St
42.311237

     stop_lon   Neighborhood gtfs_route_id
0  -71.076639       Back Bay           502
1  -71.076218       Back Bay           502
2  -71.094684        Roxbury            32
3  -71.111932  Jamaica Plain            32
4  -71.115487  Jamaica Plain            32
5  -71.114144  Jamaica Plain            32
6  -71.114046  Jamaica Plain            32
7  -71.113660  Jamaica Plain            32
8  -71.094725        Roxbury            32
9  -71.102457       Longwood            32
10 -71.109678   Mission Hill            32
11 -71.106036   Mission Hill            32
12 -71.103909   Mission Hill            32
13 -71.091880        Roxbury            32
14 -71.090936        Roxbury            32
15 -71.090515        Roxbury            32
16 -71.089240        Roxbury            32
17 -71.088338        Roxbury            32
18 -71.087061        Roxbury            32
19 -71.086503        Roxbury            32
20 -71.086550        Roxbury            32
21 -71.088322        Roxbury            32
22 -71.089559        Roxbury            32
23 -71.090502        Roxbury            32
24 -71.091087        Roxbury            32
```

We also do the same for the best routes to provide a point of comparison.

```
# This formula is used to take distances between locations (using
longitude and latitude)

def haversine(lon1, lat1, lon2, lat2):
    R = 6371  # Earth radius in km
    dlon = np.radians(lon2 - lon1)
```

```
    dlat = np.radians(lat2 - lat1)
    a = np.sin(dlat/2)**2 + np.cos(np.radians(lat1)) *
np.cos(np.radians(lat2)) * np.sin(dlon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    distance = R * c
    return distance


# This formula will be used to test if bluebikes stations are within a
10 minute walk away from any of the worst route stops.
def no_more_than_x_mins(distance, x):
    max_walking_distance = x / 60  * 5 # assuming a walking speed of 5
km/h.
    return distance <= max_walking_distance

MAX_WALKING_DISTANCE = 10 # in minutes

close_blue_bikes_list = defaultdict(list)
# Comparing the locations:
for _, bus_stop in worst_routes_loc.iterrows():
    # Extract latitude and longitude for the bus stop
    bus_stop_lat, bus_stop_lon = float(bus_stop['stop_lat']),
float(bus_stop['stop_lon'])

    # Iterate through each blue bike station
    for _, bike_station in processed_blue_bikes_stations.iterrows():
        # Extract latitude and longitude for the bike station
        bike_station_lat, bike_station_lon =
float(bike_station['Latitude']), float(bike_station['Longitude'])

        # Calculate the distance between the bus stop and the bike
station
        distance = haversine(bus_stop_lon, bus_stop_lat,
bike_station_lon, bike_station_lat)

        if no_more_than_x_mins(distance, MAX_WALKING_DISTANCE):
            if (bike_station['station_id'] not in
close_blue_bikes_list[bus_stop["route"]]): # taking only the distinct
stops

close_blue_bikes_list[bus_stop["route"]].append(bike_station["station_
id"])


close_blue_bikes_list_best = defaultdict(list)
# Comparing the locations:
for _, bus_stop in best_routes_loc.iterrows():
    # Extract latitude and longitude for the bus stop
    bus_stop_lat, bus_stop_lon = float(bus_stop['stop_lat']),
float(bus_stop['stop_lon'])
```

```
    # Iterate through each blue bike station
    for _, bike_station in processed_blue_bikes_stations.iterrows():
        # Extract latitude and longitude for the bike station
        bike_station_lat, bike_station_lon =
float(bike_station['Latitude']), float(bike_station['Longitude'])

        # Calculate the distance between the bus stop and the bike
station
        distance = haversine(bus_stop_lon, bus_stop_lat,
bike_station_lon, bike_station_lat)

        if no_more_than_x_mins(distance, MAX_WALKING_DISTANCE):
            if (bike_station['station_id'] not in
close_blue_bikes_list_best[bus_stop["route"]]): # taking only the
distinct stops

close_blue_bikes_list_best[bus_stop["route"]].append(bike_station["sta
tion_id"])
```

```
print(len(close_blue_bikes_list_best['111']))
```

```
13
```

```
start_trip_count = blue_bikes_trips.groupby('start station id')["start
station id"].count()
end_trip_count = blue_bikes_trips.groupby('end station id')["end
station id"].count()

trip_counts = pd.concat([start_trip_count, end_trip_count], axis = 1)
trip_counts.columns = ['start_trip_count', 'end_trip_count']
trip_counts["difference"] = trip_counts["end_trip_count"] -
trip_counts["start_trip_count"] # Negative means more stations that
people pick up bikes from.
print(trip_counts.head(10))
```

```
    start_trip_count  end_trip_count  difference
3              199.0           211.0        12.0
4              349.0           374.0        25.0
6              644.0           601.0       -43.0
7               69.0            78.0         9.0
8              269.0           271.0         2.0
9              800.0           822.0        22.0
10             428.0           434.0         6.0
11             609.0           638.0        29.0
12             502.0           492.0       -10.0
14             586.0           620.0        34.0
```

```
# Get the list of best routes (e.g., top 10)
top_best_routes = best_routes.head(12).index.tolist()
print(top_best_routes)
```

```
# Filter close_blue_bikes_list for these routes
best_route_stations = {route: stations for route, stations in
close_blue_bikes_list_best.items() if route in top_best_routes}
print(len(best_route_stations))
# Flatten the dictionary to a list of tuples (route, station_id)
route_station_pairs = [(route, station_id) for route, stations in
best_route_stations.items() for station_id in stations]

# Convert to DataFrame
route_station_df = pd.DataFrame(route_station_pairs, columns=['route',
'station_id'])

# Merge with trip_counts
relevant_trip_counts_best = route_station_df.merge(trip_counts,
left_on='station_id', right_index=True)
```

```
['742', '502', '32', '749', '111', '751', '741', '746', '7', '31',
'15', '39']
9
```

```
# Get the list of best routes (e.g., top 10)
top_worst_routes = worst_routes.head(12).index.tolist()
print(top_worst_routes)
# Filter close_blue_bikes_list for these routes
worst_route_stations = {route: stations for route, stations in
close_blue_bikes_list.items() if route in top_worst_routes}
print(len(worst_route_stations))
# Flatten the dictionary to a list of tuples (route, station_id)
route_station_pairs_worst = [(route, station_id) for route, stations
in worst_route_stations.items() for station_id in stations]

# Convert to DataFrame
route_station_df_worst = pd.DataFrame(route_station_pairs_worst,
columns=['route', 'station_id'])

# Merge with trip_counts
relevant_trip_counts_worst = route_station_df_worst.merge(trip_counts,
left_on='station_id', right_index=True)
```

```
['9703', '449', '448', '459', '747', '41', '19', '70A', '14', '701',
'8', '354']
10
```

```
# Example: Calculate the mean difference for each route
mean_differences_best = relevant_trip_counts_best.groupby('route')
['difference'].mean()
print(mean_differences_best)
```

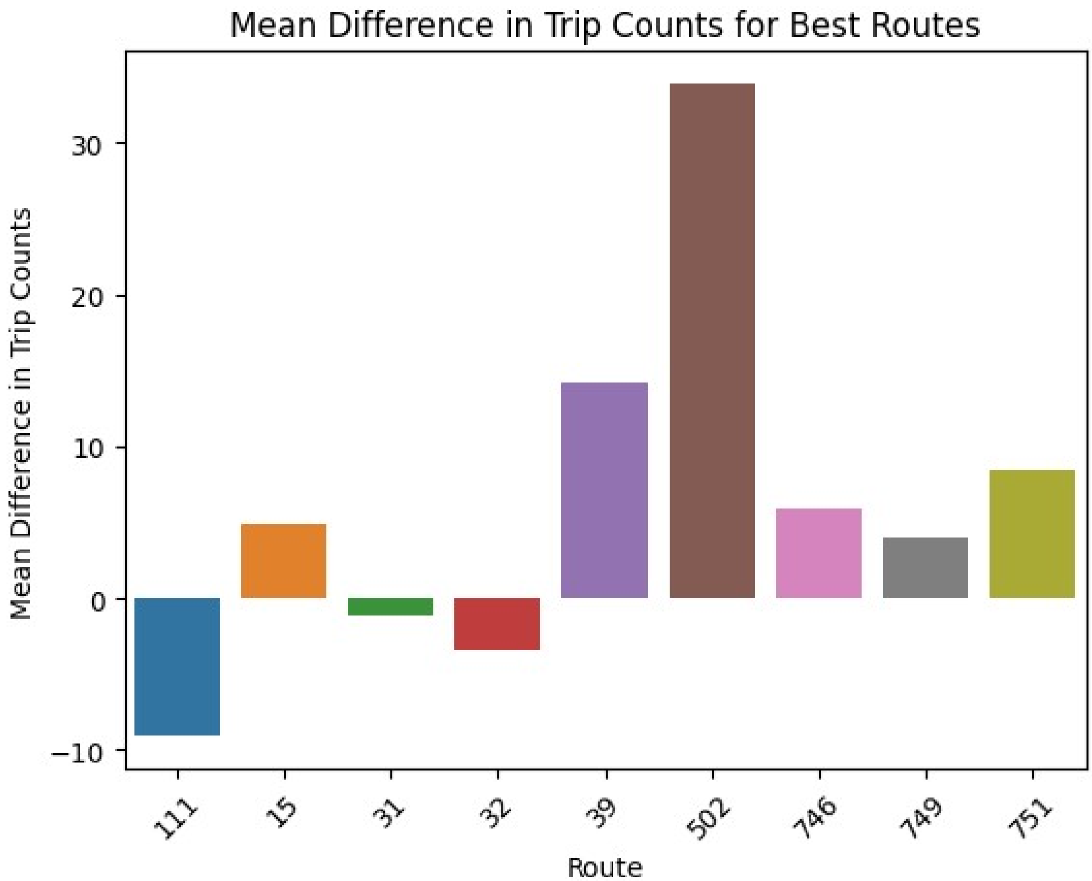
```
route
111    -9.076923
15      4.833333
```

```
31     -1.100000
32     -3.472222
39     14.173913
502    33.857143
746     5.909091
749     4.025641
751     8.459459
Name: difference, dtype: float64
```

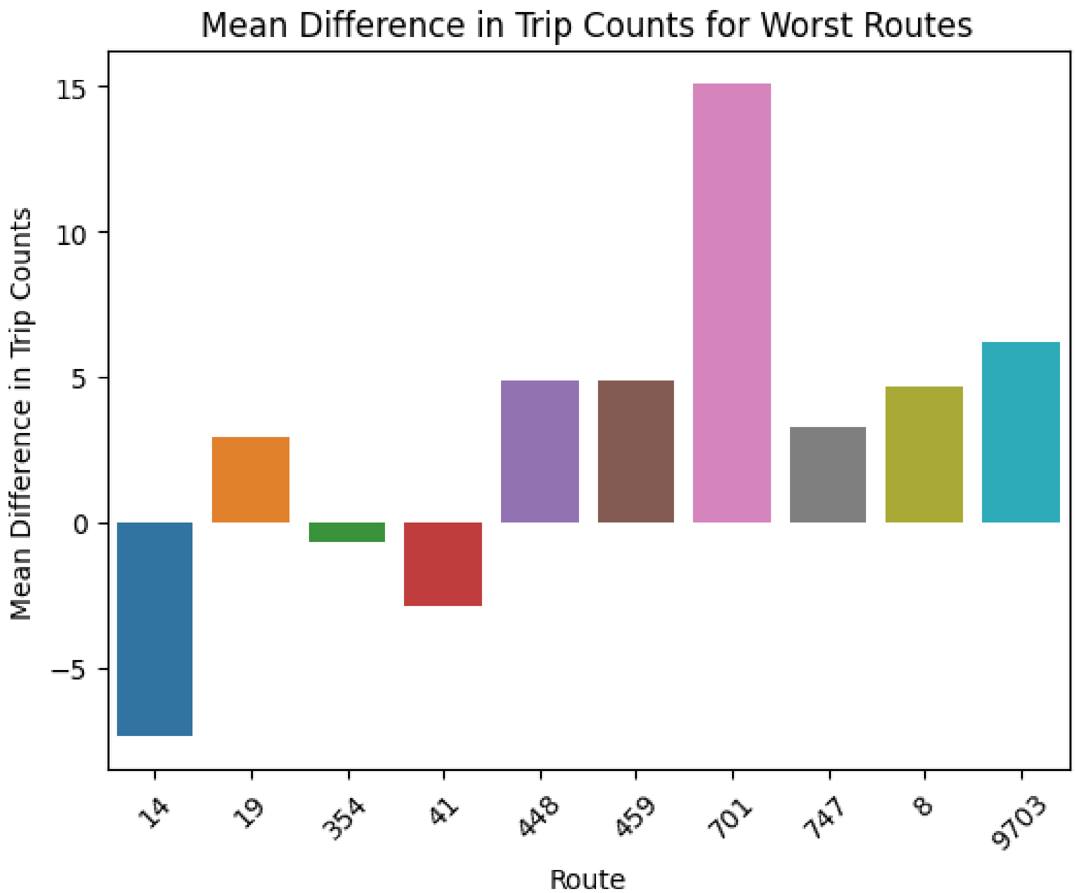
```
# Example: Calculate the mean difference for each route
mean_differences_worst = relevant_trip_counts_worst.groupby('route')
['difference'].mean()
print(mean_differences_worst)
```

```
route
14      -7.384615
19       2.916667
354     -0.684211
41      -2.904762
448      4.833333
459      4.833333
701     15.076923
747      3.240000
8        4.682927
9703     6.181818
Name: difference, dtype: float64
```

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.barplot(x=mean_differences_best.index,
y=mean_differences_best.values)
plt.xticks(rotation=45)
plt.xlabel('Route')
plt.ylabel('Mean Difference in Trip Counts')
plt.title('Mean Difference in Trip Counts for Best Routes')
plt.show()
```

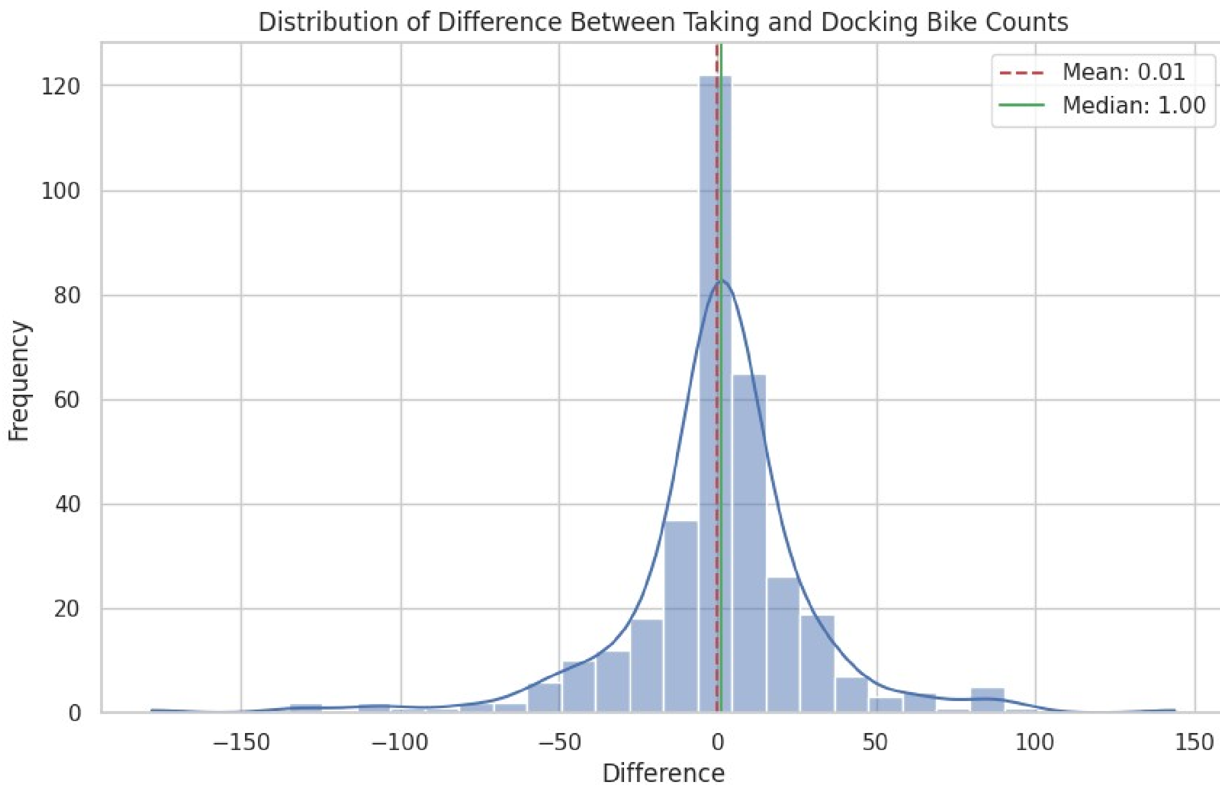



```
#Plot mean difference for worst routes
sns.barplot(x=mean_differences_worst.index,
y=mean_differences_worst.values)
plt.xticks(rotation=45)
plt.xlabel('Route')
plt.ylabel('Mean Difference in Trip Counts')
plt.title('Mean Difference in Trip Counts for Worst Routes')
plt.show()
```




```
mean_diff = trip_counts['difference'].mean()
median_diff = trip_counts['difference'].median()
std_diff = trip_counts['difference'].std()

print("Mean difference:", mean_diff)
print("Median difference:", median_diff)
print("Standard deviation:", std_diff)

Mean difference: 0.005747126436781609
Median difference: 1.0
Standard deviation: 30.98414526724328

# Set the style of seaborn
sns.set(style="whitegrid")

# Create a distribution plot
plt.figure(figsize=(10, 6))
sns.histplot(trip_counts['difference'], kde=True, bins=30)

# Add titles and labels
plt.title('Distribution of Difference Between Taking and Docking Bike
Counts')
```

```
plt.xlabel('Difference')
plt.ylabel('Frequency')

# Show mean and median in the plot
plt.axvline(mean_diff, color='r', linestyle='--', label=f"Mean:
{mean_diff:.2f}")
plt.axvline(median_diff, color='g', linestyle='-', label=f"Median:
{median_diff:.2f}")

# Add legend
plt.legend()

# Show the plot
plt.show()
```




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

df = pd.DataFrame(blue_bikes_trips)

# Create a box plot for the 'tripduration' column
plt.figure(figsize=(10, 6))
plt.boxplot(df['tripduration'], vert=False)  # 'vert=False' makes the
box plot horizontal
plt.title('Box plot of Trip Durations')
```

```
plt.xlabel('Duration (in hours)')
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-d5d373f86e4a> in <cell line: 4>()
      2 import matplotlib.pyplot as plt
      3
----> 4 df = pd.DataFrame(blue_bikes_trips)
      5
      6 # Create a box plot for the 'tripduration' column

NameError: name 'blue_bikes_trips' is not defined
```

###Blue Bike Station Duration

```
avg_trip_duration_start = blue_bikes_trips.groupby('start station id')['tripduration'].mean().reset_index()
avg_trip_duration_start.rename(columns={'start station id': 'station_id', 'tripduration': 'avg_start_duration'}, inplace=True)

# Calculate average trip duration for end stations
avg_trip_duration_end = blue_bikes_trips.groupby('end station id')['tripduration'].mean().reset_index()
avg_trip_duration_end.rename(columns={'end station id': 'station_id', 'tripduration': 'avg_end_duration'}, inplace=True)

# Merge the two dataframes on station_id
merged_avg_durations = pd.merge(avg_trip_duration_start, avg_trip_duration_end, on='station_id', how='outer')

# Calculating the mean of the two averages, handling cases where one might be NaN
merged_avg_durations['avg_trip_duration'] = merged_avg_durations[['avg_start_duration', 'avg_end_duration']].mean(axis=1, skipna=True)

bike_station_avg_usage = merged_avg_durations[['station_id', 'avg_trip_duration']].set_index('station_id').to_dict()['avg_trip_duration']

print(bike_station_avg_usage)
```

```
{3: 17.033690569752395, 4: 13.714920526689447, 6: 16.092184695624614,
7: 24.113113154960978, 8: 26.39300207593154, 9: 13.660425359894566,
10: 12.3289057452948, 11: 22.37370254266806, 12: 15.193988072101838,
14: 14.413497926529413, 15: 17.79285580524345, 16: 15.107208225034721,
17: 26.065753492642497, 19: 32.4173069813575, 20: 19.292263427109976,
```

```
21: 22.845110646958013, 22: 34.416340305892945, 23:
14.552135733169958, 24: 61.77415149602463, 25: 10.362235343575632, 26:
10.782945493277047, 27: 11.519996476391825, 29: 12.60202026901826, 30:
14.781514656544417, 31: 11.868469877234421, 32: 13.769010866910428,
33: 17.422139941654457, 36: 14.546961805555554, 37:
212.60039062499996, 39: 17.213605736100774, 40: 14.827959016855516,
41: 18.475649803892978, 42: 28.56688916321633, 43: 24.48252290448343,
44: 108.77078105847573, 46: 14.699977289479595, 47:
14.630674511398581, 49: 18.785899901735213, 51: 11.948253560126583,
54: 72.41685397935399, 55: 13.833070552811138, 56: 142.15872774397116,
58: 21.47957939884796, 59: 16.493282278863518, 60: 18.598612474429466,
63: 14.254332546108863, 65: 16.685637352875858, 66:
13.222459526398453, 67: 10.473578861598295, 68: 10.717546423405803,
69: 25.45218320278503, 70: 16.489238197449737, 71: 12.44133192175842,
72: 12.155125252130912, 73: 19.34624716553288, 74: 14.019971771418994,
75: 16.965181630298908, 76: 11.591324498069785, 77:
19.416725427978218, 78: 14.611416945742203, 80: 9.477270925293363, 81:
21.48075779415298, 82: 18.84125991348234, 84: 19.351344621831082, 85:
16.12650818125137, 86: 19.401852346999405, 87: 12.707698861219988, 89:
13.684821856963438, 90: 17.772099569838055, 91: 20.32076806983195, 92:
24.28092948717949, 93: 82.33660929951691, 94: 30.351137349254444, 95:
14.685466032030721, 96: 13.34569706742346, 97: 14.197270697490616, 98:
19.24468487300082, 99: 41.51385371095249, 100: 13.105872378937727,
101: 208.60589181286548, 102: 18.61619845188843, 103:
12.830303242687599, 104: 17.766283143939397, 105: 13.280714085297419,
107: 9.69295728585166, 108: 15.613916420377492, 109:
21.602296521445457, 110: 15.531137017485058, 111: 56.681064553480965,
115: 13.531856446525115, 116: 13.364847148446184, 117:
10.442484628897674, 118: 63.373032254897396, 119: 55.262973586000456,
121: 18.525030525030527, 124: 16.47491051898986, 125:
29.979343971631202, 126: 36.444355088153415, 131: 19.243008330911557,
133: 15.683061594202899, 135: 8.15861111111111, 136:
12.57087542087542, 137: 16.976463963963965, 138: 218.4180819151668,
139: 11.388227923408401, 140: 117.96740033833478, 141:
13.910692951015534, 142: 12.441478604020224, 143: 15.770200395429107,
144: 13.900491854636591, 145: 17.177182904411765, 146:
25.117361111111112, 149: 13.181809351785446, 150: 33.171113520086266,
151: 69.35559530804096, 152: 21.941122581601856, 156:
14.755833333333335, 157: 11.879062347707073, 159: 18.239738562091503,
160: 15.720307584572291, 161: 11.73120931739189, 162:
16.981986531986532, 163: 35.7806747311828, 169: 27.862722870349987,
170: 30.414409722222224, 171: 13.349680975572923, 173:
25.118753623188407, 174: 20.97297943567671, 175: 18.34290986991534,
176: 17.73381036201132, 177: 10.088180353693676, 178:
12.299685133818876, 179: 15.657235609463386, 180: 18.791921041921043,
181: 17.82758689077297, 182: 48.385882981467155, 183:
12.830549984905335, 184: 12.944133777973763, 185: 14.817075733082405,
186: 20.284472544998863, 187: 13.838586651218312, 188:
63.01822941084545, 189: 12.283346723068682, 190: 27.282928581080956,
```

191: 13.495178025338575, 192: 24.170863199532764, 193:
14.926521739130434, 194: 13.356277472527472, 195: 18.42533754490276,
196: 16.23124686716792, 197: 25.4379884004884, 200: 17.99629442573887,
201: 19.136493318887688, 202: 19.744756944444447, 203:
20.545833333333334, 205: 23.348492063492063, 206: 46.538987134502925,
208: 21.764282962028865, 210: 16.31319444444444, 212:
53.129487179487185, 213: 8.693738932456363, 214: 32.18661236424394,
215: 14.570869990224828, 217: 19.33696581196581, 218:
19.97229021073731, 219: 32.05049242424243, 221: 15.572908695986625,
222: 27.42879170416292, 224: 24.194884057971016, 225:
12.120843243133924, 226: 24.32893411022518, 228: 16.208991959951458,
232: 29.566666666666666, 233: 16.646004497234063, 234:
16.61979098982023, 235: 17.70306613756614, 236: 25.49570634920635,
239: 18.620762251334302, 258: 33.38267973856209, 271:
15.603993055555556, 272: 607.7558976715686, 273: 18.278729494190024,
279: 209.41047361729179, 280: 32.65900438310512, 282:
19.91863692067736, 296: 38.10569444444445, 318: 25.484860019646366,
319: 25.8104166666667, 327: 71.21119320957499, 328:
9.374698092031426, 329: 20.19794823232323, 330: 33.40914892652056,
331: 22.24611883214824, 332: 13.67711177701337, 333:
21.89619397456476, 334: 14.395225694444443, 335: 11.003697297679683,
336: 34.72465277777778, 337: 14.259567698046961, 338:
15.498533754796217, 340: 12.816666666666666, 341: 12.334375000000001,
342: 33.833821807349466, 344: 20.085129802699896, 345:
11.481003968253969, 346: 108.29430555555557, 348: 2365.2374999999997,
349: 21.695238095238096, 350: 10.814102564102564, 351:
38.43070926657883, 352: 23.6158701923247, 353: 27.621296296296293,
355: 39.47083333333333, 356: 24.023932801200708, 358:
161.12218471136384, 359: 13.289965606990148, 360: 21.032118055555557,
361: 19.040195085010012, 362: 197.59695921985815, 363:
10.251777956069034, 364: 32.07448226988234, 365: 56.009320257993274,
367: 16.653125, 370: 55.758498001336996, 371: 22.723437500000003, 372:
39.25083636291396, 373: 25.041964285714286, 374: 17.92089086226908,
376: 18.880636451550988, 377: 14.899654267845929, 378:
15.894895609639704, 379: 11.322622005323868, 380: 10.370901606296307,
381: 12.643952290700692, 385: 12.480829358220994, 386:
11.423927782728294, 387: 34.29440476190476, 389: 18.41813725490196,
390: 12.999870414673047, 391: 29.03888888888889, 392:
24.346296296296295, 393: 31.68888888888889, 394: 611.3434027777778,
395: 17.138403263403262, 396: 15.69, 397: 28.195183982683986, 398:
9.953435236048087, 399: 13.226490242460084, 400: 11.574624307316615,
401: 116.45742589231702, 402: 17.15681276088253, 403:
16.633195970695972, 404: 14.35032979384895, 405: 20.090826719576718,
407: 14.503923536070802, 408: 16.569025157232705, 413:
12.937317413003505, 414: 11.079043778801843, 415: 21.801787050208098,
417: 11.424718079922027, 419: 22.089636752136755, 422:
47.46666666666667, 424: 15.29375, 425: 13.167361111111111, 426:
12.640535149938218, 432: 11.713148148148148, 433: 16.373674242424244,
434: 14.658333333333335, 436: 19.42693236714976, 437:

```
15.165008506868219, 440: 13.328142941573793, 441: 114.46779146141218,
442: 27.792845471521943, 443: 28.15715876641919, 445:
116.41289682539681, 446: 13.20158160856676, 447: 25.19611872146119,
448: 14.833333333333334, 452: 19.73916998053981, 455:
18.83587187666135, 456: 13.528797430083145, 458: 24.28521739130435,
461: 31.076585365853656, 462: 18.259099698170157, 463:
16.619619236583524, 466: 16.08888888888889, 467: 37.784325396825395,
468: 16.496666666666666, 469: 22.458928571428572, 470:
24.89861111111111, 471: 10.416688861693402, 472: 9.148631788265934,
473: 30.323333333333334, 475: 22.416950312989044, 478:
9.738294314381271, 479: 9.500974077226118, 480: 17.76343954248366,
481: 14.79969696969697, 482: 17.91813725490196, 483:
14.500891265597147, 485: 49.15, 486: 21.12569444444444, 487:
12.090930018416206, 488: 18.718704535729568, 489: 17.40766081871345,
490: 26.345, 491: 13.668734371289272, 492: 12.417676127463105, 493:
17479.316666666666, 494: 17.856649831649833, 495: 20.426260115358126,
496: 13.087136598964555, 497: 18.205300332383665, 498:
16.202734778121776, 499: 42.91843564965182, 502: 20.534127286585367,
503: 13.561507936507937, 505: 14.751111111111111, 506:
25.596825396825395, 507: 22.2658547008547, 508: 23.309722222222224,
509: 15.971755599472992, 515: 11.633013960216784, 518:
7.500555555555556, 519: 3.4722222222222228, 520: 10.34431216931217,
522: 51.45, 523: 25.32328904991948, 524: 50.833333333333336, 525:
49.5, 526: 30.616666666666667, 527: 13.425, 528: 49.66444444444444,
529: 15.8759494978388, 530: 19.055500637755102, 531:
11.943669948537206, 532: 21.698434684684685, 533: 18.105456069751845,
535: 62.43888888888889, 536: 49.31666441289159, 537:
27.123809523809523, 538: 14.845833333333335, 539: 21.972430555555555,
540: 7.696933229813665, 541: 18.174977011494253, 544:
11.53458388620183, 545: 22.6, 547: 98.56428571428572, 548:
27.91111111111111, 549: 23.582329654317405, 550: 14.48267156862745,
553: 12.234724178403756, 554: 13.976830662596079, 557:
9.027083333333334, 1: 10929.633333333331, 504: 30.616666666666667,
546: 23.483333333333334, 555: 1019.875}

# Calculate the average trip duration for each best route
best_route_avg_durations = {}
for route, station_ids in best_route_stations.items():
    total_duration = 0
    count = 0
    for station_id in station_ids:
        if station_id in bike_station_avg_usage:
            total_duration += bike_station_avg_usage[station_id]
            count += 1
    if count > 0:
        best_route_avg_durations[route] = total_duration / count

# Convert to a DataFrame for easy plotting
best_route_durations_df =
```

```
pd.DataFrame(list(best_route_avg_durations.items()), columns=['route',
'avg_duration'])

# Sort and select the top 10 best routes
top_10_best_routes =
best_route_durations_df.sort_values(by='avg_duration').head(10)

# Plot
plt.figure(figsize=(12, 6))
plt.bar(top_10_best_routes['route'],
top_10_best_routes['avg_duration'], color='green')
plt.xlabel('Route')
plt.ylabel('Average Trip Duration (minutes)')
plt.y
plt.title('Average Trip Duration for Top 10 Best Routes')
plt.xticks(rotation=45)
plt.show()

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call
last)
<ipython-input-33-c8cf68177185> in <cell line: 6>()
      4 plt.xlabel('Route')
      5 plt.ylabel('Average Trip Duration (minutes)')
----> 6 plt.y
      7 plt.title('Average Trip Duration for Top 10 Best Routes')
      8 plt.xticks(rotation=45)

AttributeError: module 'matplotlib.pyplot' has no attribute 'y'
```