This is just an initial block to insert an intor

```python
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem.snowball import SnowballStemmer
from nltk.tokenize import word_tokenize, sent_tokenize

import pandas as pd

df = pd.read_csv("census-block-group-data.csv")
#df.drop([0], axis=0, inplace=True)
Arrival_df = pd.read_csv("MBTA-Bus-Arrival-Departure-Times_2023-
01.csv")

df = df.drop(columns = ['FILEID','STUSAB','SUMLEV','GEOCODE','REGION',
'DIVISION','STATE','COUNTY','COUSUB'])

Neighborhood_df =
pd.read_csv("redistricting_data_tract20_nbhd_hhpopsize_ab-1.csv")
Neighborhood_df.columns = Neighborhood_df.iloc[0]

# Optionally, drop the first row from the DataFrame
Neighborhood_df = Neighborhood_df.drop(Neighborhood_df.index[0])
Bus_stops = pd.read_csv("MBTA_Systemwide_GTFS_Map.csv")

Bus_stops.head()
```

```
           X          Y  OBJECTID stop_id  stop_code  \
0 -71.082754  42.330957    647997       1        1.0
1 -71.068787  42.330555    647998      10       10.0
2 -71.062911  42.355692    647999   10000    10000.0
3 -71.076237  42.331591    648000   10003    10003.0
4 -71.071280  42.335017    648001   10005    10005.0

                       stop_name stop_desc platform_code platform_name
\
0   Washington St opp Ruggles St       NaN           NaN           NaN

1  Theo Glynn Way @ Newmarket Sq       NaN           NaN           NaN

2       Tremont St opp Temple Pl       NaN           NaN           NaN

3       Albany St opp Randall St       NaN           NaN           NaN

4     Albany St opp E Concord St       NaN           NaN           NaN
```

```
    stop_lat  ...  Sidewalk_Condition Sidewalk_Material
Current_Shelter  \
0  42.330957  ...                Good             Brick
JCD
1  42.330555  ...                Good          Concrete
0
2  42.355692  ...                Good          Concrete
0
3  42.331591  ...                Good          Concrete
0
4  42.335017  ...                Good             Brick
JCD

                     Routes Municipality_1  Neighborhood created_user
\
0  1|8|10|47|19|170|191|171         BOSTON       Roxbury    DOT_ADMIN

1           9|8|10|9702|171         BOSTON  South Boston    DOT_ADMIN

2         193|192|43|191|55         BOSTON   Beacon Hill    DOT_ADMIN

3                      1|47         BOSTON       Roxbury    DOT_ADMIN

4                   8|10|47         BOSTON     South End    DOT_ADMIN


             created_date last_edited_user        last_edited_date
0  2023/05/30 14:10:22+00        DOT_ADMIN  2023/05/30 14:10:22+00
1  2023/05/30 14:10:22+00        DOT_ADMIN  2023/05/30 14:10:22+00
2  2023/05/30 14:10:22+00        DOT_ADMIN  2023/05/30 14:10:22+00
3  2023/05/30 14:10:22+00        DOT_ADMIN  2023/05/30 14:10:22+00
4  2023/05/30 14:10:22+00        DOT_ADMIN  2023/05/30 14:10:22+00

[5 rows x 34 columns]
```

```
Bus_stops = Bus_stops[["stop_name","Neighborhood","Routes"]]
Bus_stops.describe()
```

```
              stop_name Neighborhood Routes
count              6879         1787   6046
unique             6082           26    726
top     Sullivan Square   Dorchester    230
freq                 14          315    115
```

```
Neighborhood_df = Neighborhood_df.iloc[:, :7]

Neighborhood_df.head()
```

```
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
5   Charlestown  19120       13626                             990

0 Hispanic or Latino  \
1                3259
2                1208
3                 537
4                5376
5                2075

0 Asian, Native Hawaiian and Pacific Islander alone, all ages  \
1                                               6271
2                                               2410
3                                                630
4                                               8703
5                                               1650

0 Other Races or Multiple Races,  all ages
1                                     1512
2                                      817
3                                      396
4                                     2860
5                                      779
```

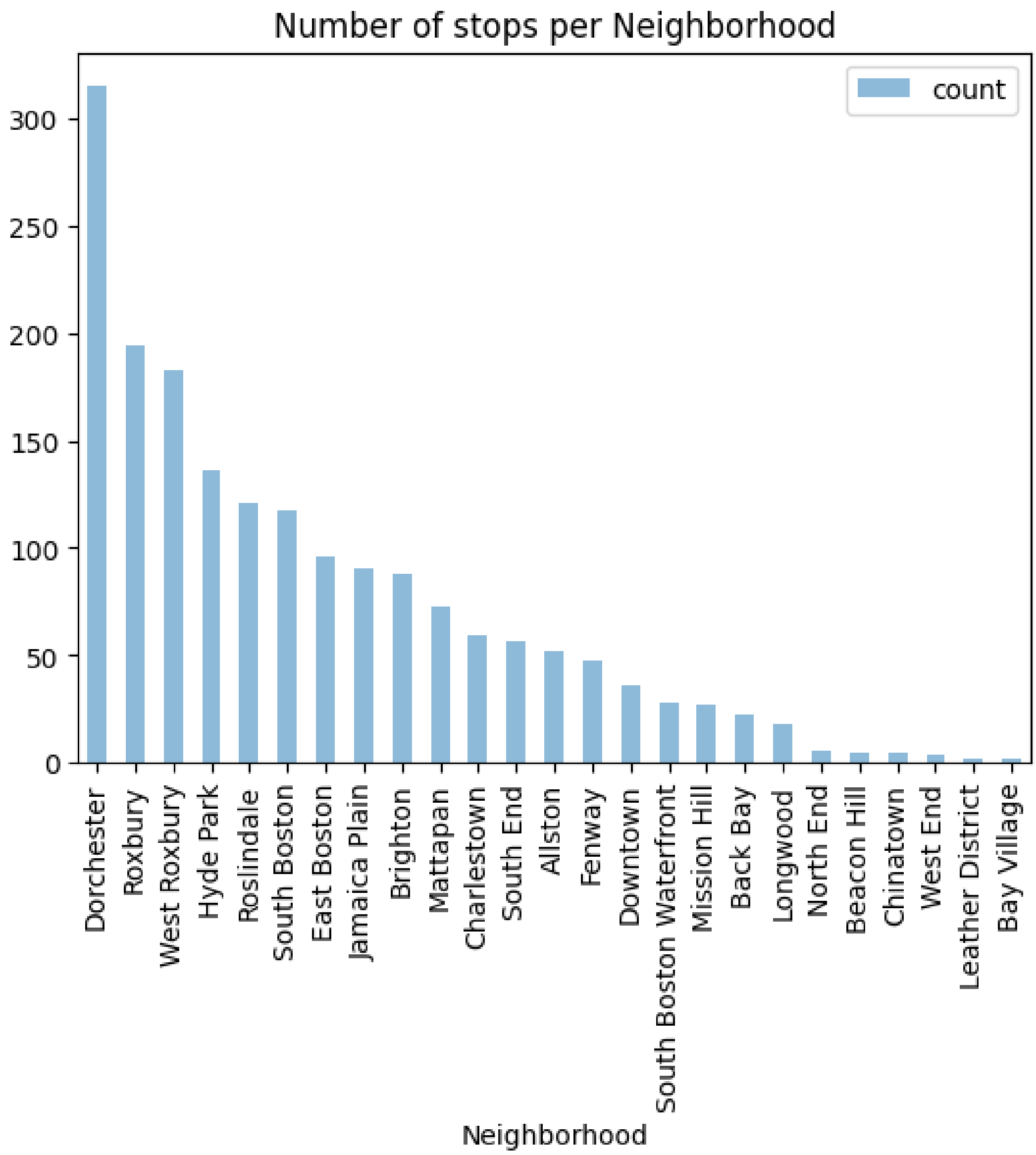
```
Neighborhood_df['Total:'] =
pd.to_numeric(Neighborhood_df['Total:'].str.replace(',', ''),
errors='coerce')
Neighborhood_df['White alone'] = pd.to_numeric(Neighborhood_df['White
alone'].str.replace(',', ''), errors='coerce')
Neighborhood_df['Black or African American alone'] =
pd.to_numeric(Neighborhood_df['Black or African American
alone'].str.replace(',', ''), errors='coerce')
Neighborhood_df['Hispanic or Latino'] =
pd.to_numeric(Neighborhood_df['Hispanic or Latino'].str.replace(',',
''), errors='coerce')
Neighborhood_df['Asian, Native Hawaiian and Pacific Islander alone,
all ages'] = pd.to_numeric(Neighborhood_df['Asian, Native Hawaiian and
Pacific Islander alone, all ages'].str.replace(',', ''),
errors='coerce')
# Neighborhood_df['Other Races or Multiple Races, all ages '] =
pd.to_numeric(Neighborhood_df['Other Races or Multiple Races, all ages
'].str.replace(',', ''), errors='coerce')

Neighborhood_df.head()
```

```
0 field concept  Total:  White alone  Black or African American alone
\
1       Allston   24904        12536                             1326
```

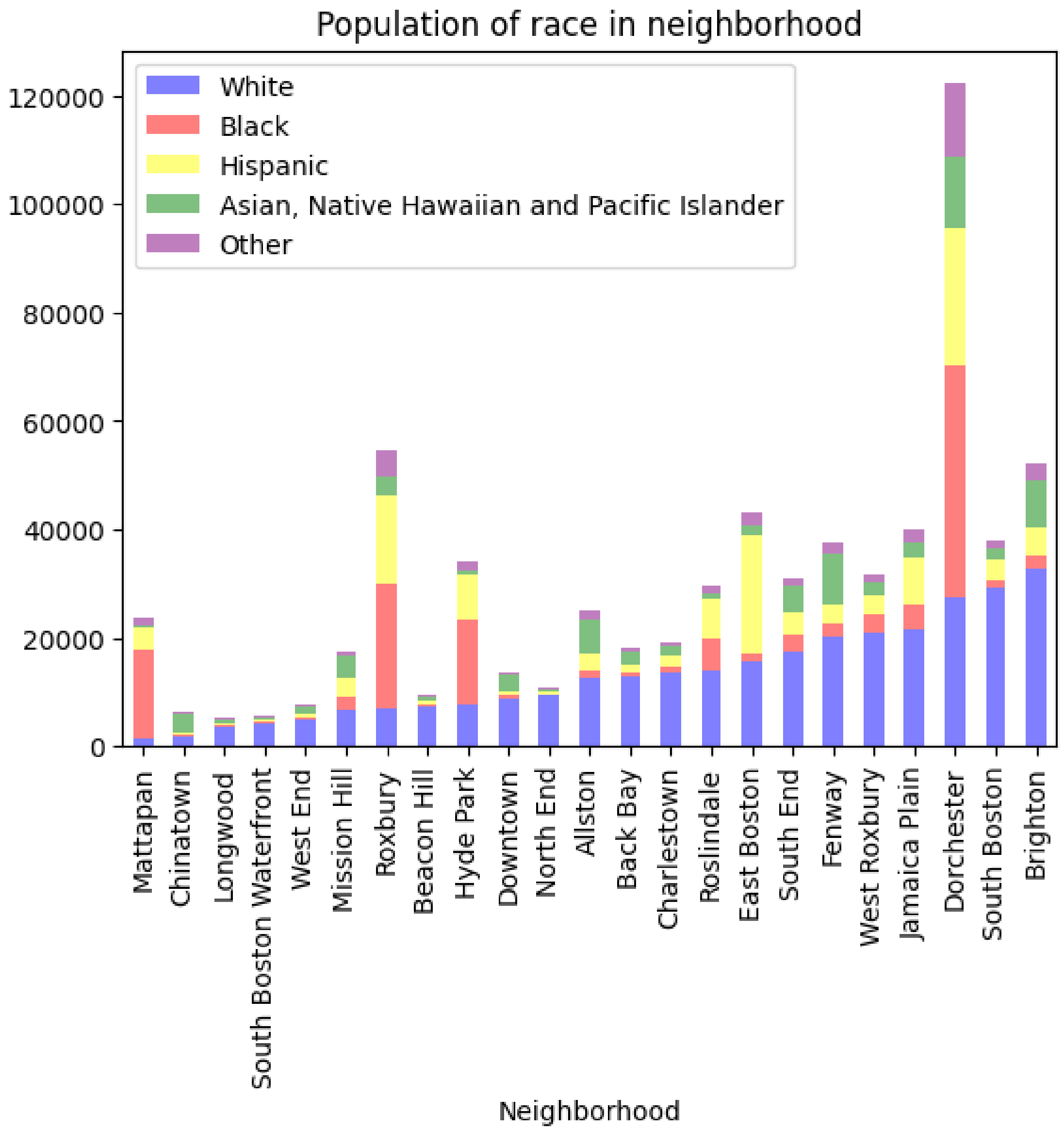
```
2      Back Bay   18190        13065                              690

3   Beacon Hill    9336         7521                              252

4      Brighton   52047        32694                             2414

5   Charlestown   19120        13626                              990


0  Hispanic or Latino  \
1                3259
2                1208
3                 537
4                5376
5                2075

0  Asian, Native Hawaiian and Pacific Islander alone, all ages  \
1                                               6271
2                                               2410
3                                                630
4                                               8703
5                                               1650

0 Other Races or Multiple Races,  all ages
1                                     1512
2                                      817
3                                      396
4                                     2860
5                                      779

# if Neighborhood_df['Total:'].dtype == 'object':
#     Neighborhood_df['Total:'] =
pd.to_numeric(Neighborhood_df['Total:'].str.replace(',', ''),
errors='coerce')

Neighborhood_Percentages_df = pd.DataFrame()
Neighborhood_Ints_df = pd.DataFrame()

Neighborhood_Percentages_df["Neighborhood"] = Neighborhood_df["field
concept"]
Neighborhood_Percentages_df["White"] = np.nan
Neighborhood_Percentages_df["Black"] = np.nan
Neighborhood_Percentages_df["Hispanic"] = np.nan
Neighborhood_Percentages_df["Asian, Native Hawaiian and Pacific
Islander"] = np.nan
Neighborhood_Percentages_df["Other"] = np.nan

Neighborhood_Ints_df["Neighborhood"] = Neighborhood_df["field
concept"]
Neighborhood_Ints_df["White"] = np.nan
```

```
Neighborhood_Ints_df["Black"] = np.nan
Neighborhood_Ints_df["Hispanic"] = np.nan
Neighborhood_Ints_df["Asian, Native Hawaiian and Pacific Islander"] =
np.nan
Neighborhood_Ints_df["Other"] = np.nan

race_categories = {
    "White": "White alone",
    "Black": "Black or African American alone",
    "Hispanic": "Hispanic or Latino",
    "Asian, Native Hawaiian and Pacific Islander": "Asian, Native
Hawaiian and Pacific Islander alone, all ages",
    "Other": "Other Races or Multiple Races,  all ages"
}

for i, row in Neighborhood_df.iterrows():
    total_population = row['Total:']  # This should be a numeric
value, not a string
    for new_col, old_col in race_categories.items():
        if isinstance(row[old_col], str):
            count = pd.to_numeric(row[old_col].replace(',', ''),
errors='coerce')
        else:
            count = pd.to_numeric(row[old_col], errors='coerce')

        percentage = (count / total_population) * 100 if
total_population else np.nan
        # percentage = count

        # Assign the percentage to the new DataFrame
        Neighborhood_Percentages_df.at[i, new_col] = percentage

for i, row in Neighborhood_df.iterrows():
    total_population = row['Total:']  # This should be a numeric
value, not a string
    for new_col, old_col in race_categories.items():
        if isinstance(row[old_col], str):
            count = pd.to_numeric(row[old_col].replace(',', ''),
errors='coerce')
        else:
            count = pd.to_numeric(row[old_col], errors='coerce')

        # percentage = (count / total_population) * 100 if
total_population else np.nan
        percentage = count

        # Assign the percentage to the new DataFrame
        Neighborhood_Ints_df.at[i, new_col] = percentage
```

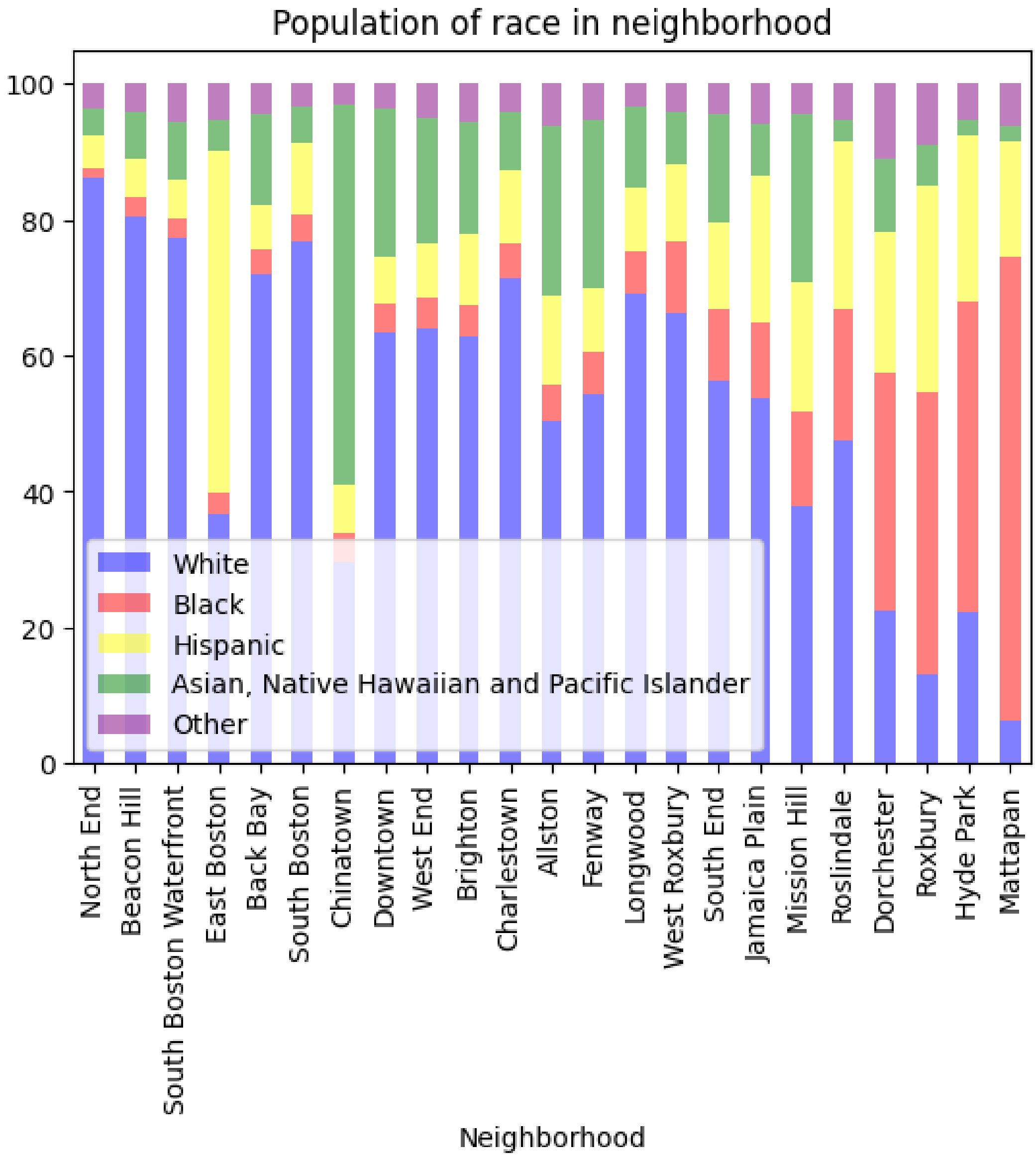
```
Bus_stops['Neighborhood'].value_counts().nlargest(25).plot(kind='bar',
legend=True, alpha=.5)
plt.title("Number of stops per Neighborhood")
plt.show()
```



```
stacked_df = Neighborhood_Ints_df.sort_values(by = ["White"])
# races = ["White alone", "Black or African American alone","Hispanic
or Latino","Asian, Native Hawaiian and Pacific Islander alone, all
ages", "Other Races or Multiple Races,  all ages" ]
ax = stacked_df.plot(kind = 'bar',x='Neighborhood', stacked=True,
color=['blue','red','yellow','green','purple'],legend=True, alpha=.5)
plt.title("Population of race in neighborhood")
plt.show()
```



```
stacked_percentage_df = Neighborhood_Percentages_df.sort_values(by =
["Black"])
# races = ["White alone", "Black or African American alone","Hispanic
or Latino","Asian, Native Hawaiian and Pacific Islander alone, all
ages", "Other Races or Multiple Races,  all ages" ]
ax = stacked_percentage_df.plot(kind = 'bar',x='Neighborhood',
stacked=True,
color=['blue','red','yellow','green','purple'],legend=True, alpha=.5)
plt.title("Population of race in neighborhood")
plt.show()
```



```
Bus_stops = Bus_stops.loc[worst_on_time_routes]
merged = pd.merge(Bus_stops, Neighborhood_Percentages_df,
left_on='Neighborhood', right_on='Neighborhood')
merged.head()

----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
Cell In[68], line 1
----> 1 Bus_stops = Bus_stops.loc[worst_on_time_routes]
      2 merged = pd.merge(Bus_stops, Neighborhood_Percentages_df,
left_on='Neighborhood', right_on='Neighborhood')
      3 merged.head()
```

```
File
/opt/homebrew/lib/python3.11/site-packages/pandas/core/indexing.py:115
3, in _LocationIndexer.__getitem__(self, key)
   1150 axis = self.axis or 0
   1152 maybe_callable = com.apply_if_callable(key, self.obj)
-> 1153 return self._getitem_axis(maybe_callable, axis=axis)

File
/opt/homebrew/lib/python3.11/site-packages/pandas/core/indexing.py:138
2, in _LocIndexer._getitem_axis(self, key, axis)
   1379     if hasattr(key, "ndim") and key.ndim > 1:
   1380         raise ValueError("Cannot index with multidimensional
key")
-> 1382     return self._getitem_iterable(key, axis=axis)
   1384 # nested tuple slicing
   1385 if is_nested_tuple(key, labels):

File
/opt/homebrew/lib/python3.11/site-packages/pandas/core/indexing.py:132
2, in _LocIndexer._getitem_iterable(self, key, axis)
   1319 self._validate_key(key, axis)
   1321 # A collection of keys
-> 1322 keyarr, indexer = self._get_listlike_indexer(key, axis)
   1323 return self.obj._reindex_with_indexers(
   1324     {axis: [keyarr, indexer]}, copy=True, allow_dups=True
   1325 )

File
/opt/homebrew/lib/python3.11/site-packages/pandas/core/indexing.py:152
0, in _LocIndexer._get_listlike_indexer(self, key, axis)
   1517 ax = self.obj._get_axis(axis)
   1518 axis_name = self.obj._get_axis_name(axis)
-> 1520 keyarr, indexer = ax._get_indexer_strict(key, axis_name)
   1522 return keyarr, indexer

File
/opt/homebrew/lib/python3.11/site-packages/pandas/core/indexes/base.py
:6114, in Index._get_indexer_strict(self, key, axis_name)
   6111 else:
   6112     keyarr, indexer, new_indexer =
self._reindex_non_unique(keyarr)
-> 6114 self._raise_if_missing(keyarr, indexer, axis_name)
   6116 keyarr = self.take(indexer)
   6117 if isinstance(key, Index):
   6118     # GH 42790 - Preserve name from an Index

File
/opt/homebrew/lib/python3.11/site-packages/pandas/core/indexes/base.py
:6175, in Index._raise_if_missing(self, key, indexer, axis_name)
```

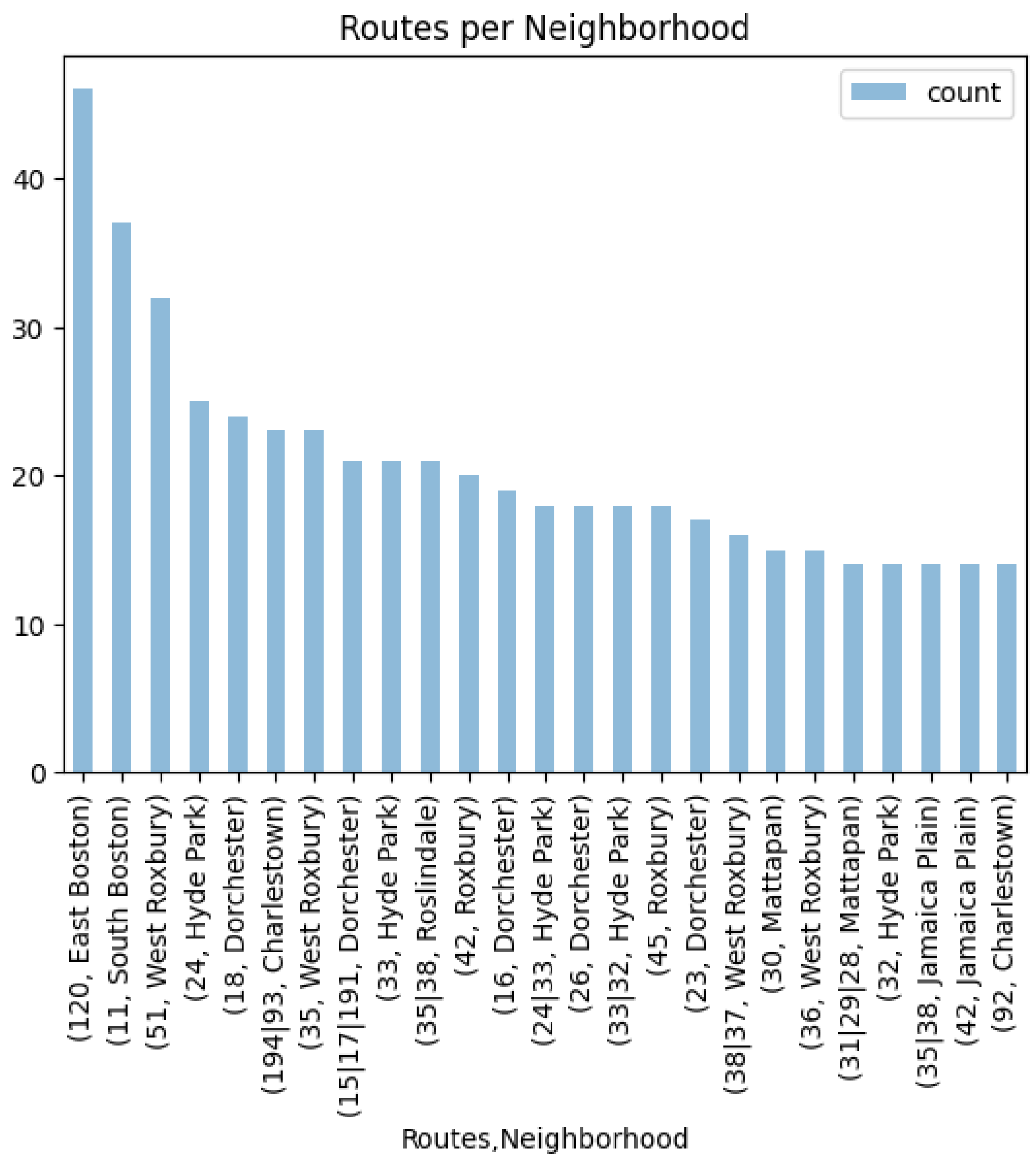
```
   6173     if use_interval_msg:
   6174         key = list(key)
-> 6175     raise KeyError(f"None of [{key}] are in the
[{axis_name}]")
   6177 not_found = list(ensure_index(key)[missing_mask.nonzero()
[0]].unique())
   6178 raise KeyError(f"{not_found} not in index")

KeyError: "None of [Index(['14', '70A', '19', '701', '41', '747',
'459', '448', '449', '9703'], dtype='object')] are in the [index]"
```

```
average_neighborhood = merged.groupby('Routes')[['White', 'Black',
'Hispanic', 'Asian, Native Hawaiian and Pacific Islander',
'Other']].mean()
average_neighborhood.head()
```

```
                 White      Black   Hispanic  \
Routes
1            34.158051  24.935417  20.861718
10           56.180430  10.545560  12.936837
104|105|109  71.265690   5.177824  10.852510
109|104|105  71.265690   5.177824  10.852510
10|170       61.395346   8.294805  10.838228

             Asian, Native Hawaiian and Pacific Islander     Other
Routes
1                                              13.038985  7.005828
10                                             15.774657  4.562516
104|105|109                                     8.629707  4.074268
109|104|105                                     8.629707  4.074268
10|170                                         14.932784  4.538837
```

```
average_neighborhood = merged.groupby('Routes')['Neighborhood']
```

```
average_neighborhood.value_counts().nlargest(25).plot(kind='bar',
legend=True, alpha=.5)
plt.title("Routes per Neighborhood")
plt.show()
```



```
worst_on_time_routes = ['14', '70A', '19', '701', '41', '747', '459',
'448', '449', '9703']

worst_routes_neighborhood =
average_neighborhood.loc[worst_on_time_routes]

----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[23], line 3
      1 worst_on_time_routes = ['14', '70A', '19', '701', '41', '747',
'459', '448', '449', '9703']
----> 3 worst_routes_neighborhood =
average_neighborhood.loc[worst_on_time_routes]

NameError: name 'average_neighborhood' is not defined

import pandas as pd

# Replace 'your_dataset.csv' with the path to your dataset file
Bus_map = pd.read_csv("MBTA_Systemwide_GTFS_Map.csv")
Bike_map = pd.read_csv("current_bluebikes_stations.csv")
# Check the first few rows of the dataframe

#adjust first row
Bike_map.columns = Bike_map.iloc[0]

Bike_map = Bike_map.drop(Bike_map.index[0])
Bike_map.head()

0  Number                                Name     Latitude
Longitude  \
1  K32015                      1200 Beacon St  42.34414899  -
71.11467361
2  W32006                         160 Arsenal  42.36466403  -
71.17569387
3  A32019                    175 N Harvard St  42.36447457  -
71.12840831
4  S32035                       191 Beacon St  42.38032335  -
71.10878613
5  C32094  2 Hummingbird Lane at Olmsted Green     42.28887    -
71.095003

0    District Public Total docks Deployment Year
1   Brookline    Yes           1            2021
2   Watertown    Yes          11            2021
3      Boston    Yes          17            2014
4  Somerville    Yes          19            2018
5      Boston    Yes          17            2020

Bus_map['latitude'] = pd.to_numeric(Bus_map['stop_lat'],
errors='coerce')
Bus_map['longitude'] = pd.to_numeric(Bus_map['stop_lon'],
errors='coerce')

Bike_map['latitude'] = pd.to_numeric(Bike_map['Latitude'],
errors='coerce')
Bike_map['longitude'] = pd.to_numeric(Bike_map['Longitude'],
errors='coerce')

# Drop rows with missing or invalid values
Bus_map.dropna(subset=['latitude', 'longitude'], inplace=True)
Bike_map.dropna(subset=['latitude', 'longitude'], inplace=True)
```

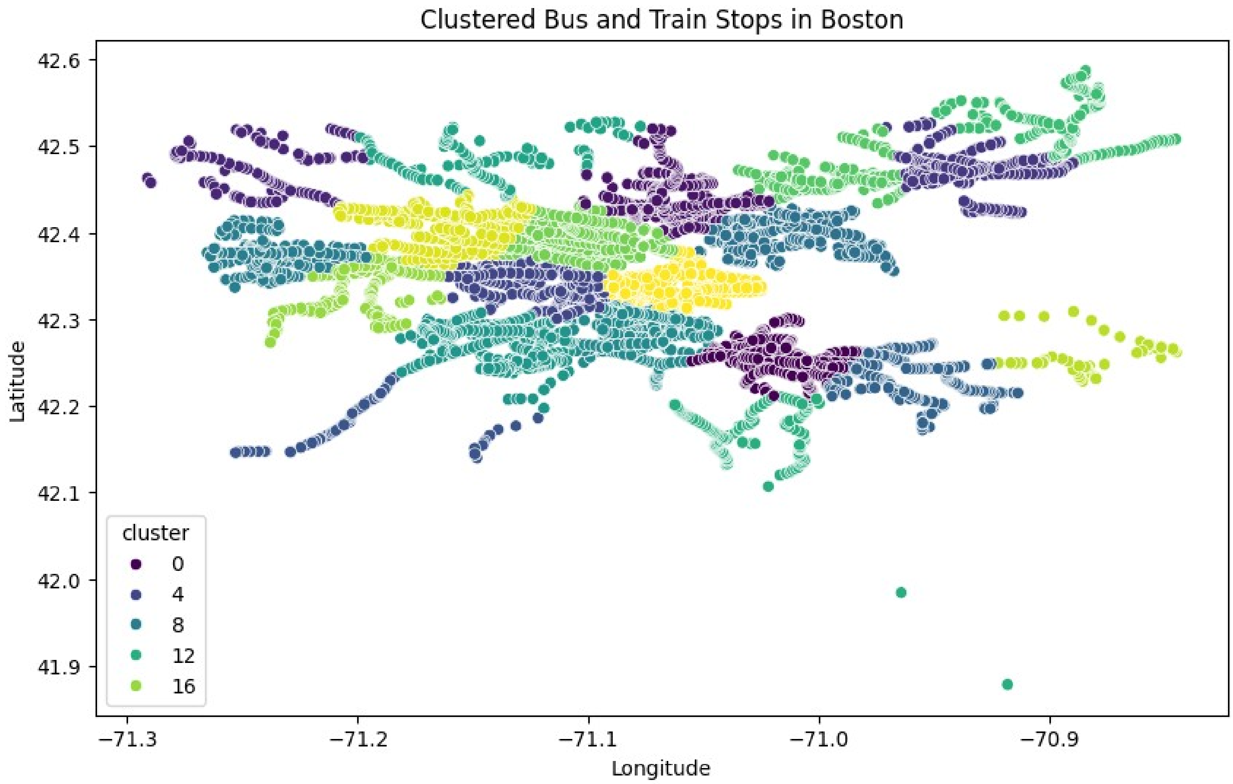
```python
from sklearn.cluster import KMeans

# Number of clusters
k = 20  # You can choose a different number based on your analysis

# Selecting the features (latitude and longitude)
features = Bus_map[['latitude', 'longitude']]

# Running KMeans
kmeans = KMeans(n_clusters=k, random_state=0).fit(features)

# Adding cluster labels to your original dataframe
Bus_map['cluster'] = kmeans.labels_
```

```
/opt/homebrew/lib/python3.11/site-packages/sklearn/cluster/
_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

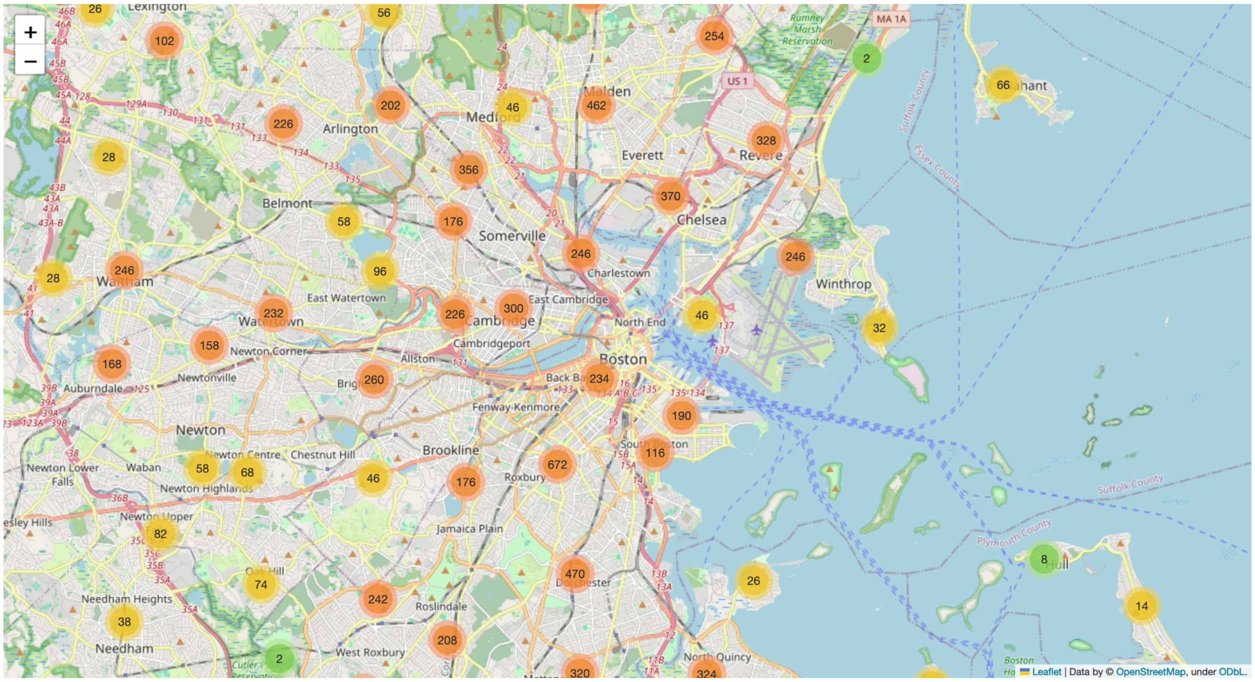
```python
# Number of clusters
k = 20  # You can choose a different number based on your analysis

# Selecting the features (latitude and longitude)
features_bike = Bike_map[['latitude', 'longitude']]

# Running KMeans
kmeans_bike = KMeans(n_clusters=k, random_state=0).fit(features_bike)

# Adding cluster labels to your original dataframe
Bike_map['cluster'] = kmeans_bike.labels_
```

```
/opt/homebrew/lib/python3.11/site-packages/sklearn/cluster/
_kmeans.py:1416: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

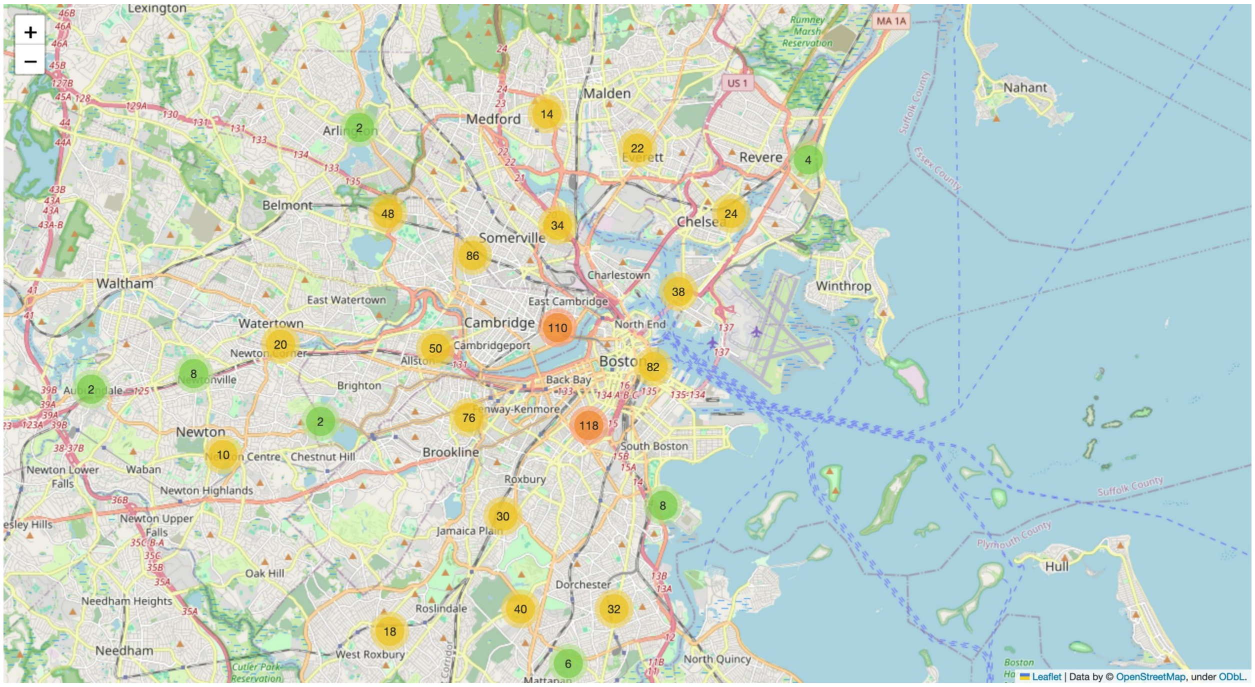
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='longitude', y='latitude', hue='cluster',
palette='viridis')
plt.title('Clustered Bus and Train Stops in Boston')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



```
# Do not edit this cell
import pandas as pd
import numpy as np
import folium #install if you haven't already
import selenium #install if you haven't already
from IPython.display import Image #install if you haven't already

def convert_map_to_png(map, filename):
    """
    Method to convert a folium map to a png file by
    saving the map as an html file and then taking a
    screenshot of the html file on the browser.

    map : folium map object
        The map to be converted to a png file
    filename : str, does not include file type
    """
    import os
    import time
    from selenium import webdriver

    html_filename=f'{filename}.html'
    map.save(html_filename)

    tmpurl=f'file://{os.getcwd()}/{html_filename}'
    try:
        try:
            browser = webdriver.Firefox()
        except:
            browser = webdriver.Chrome()
    except:
        browser = webdriver.Safari()

    browser.get(tmpurl)
    time.sleep(5)
    browser.save_screenshot(f'{filename}.png')
    browser.quit()
    os.remove(html_filename)

    return Image(f'{filename}.png')

import folium
import pandas as pd
from folium.plugins import FastMarkerCluster

# Initialize the map at a central point in Boston
boston_map = folium.Map(location=[42.3601, -71.0589], zoom_start=12)
# Latitude and longitude of Boston

# Load your dataset here
bus_train_stops = Bus_map

# Create a FastMarkerCluster
marker_cluster = FastMarkerCluster(data=bus_train_stops[['latitude',
'longitude']].values.tolist()).add_to(boston_map)

# Add popup (optional, customize as needed)
for _, stop in bus_train_stops.iterrows():
    popup = f"{stop['stop_name']}"  # Assuming your dataset has a
column 'stop_name'
    folium.Marker((stop['latitude'], stop['longitude']),
popup=popup).add_to(marker_cluster)

# Save map
boston_map.save("bus_train_stops_clusters.html")

# The function `convert_map_to_png` is not a standard function and
would need to be defined if you wish to use it.
# convert_map_to_png(boston_map, 'boston_bus_train_stops')
convert_map_to_png(boston_map, 'Boston_Bus_stops')
```



```python
import folium
import pandas as pd
from folium.plugins import FastMarkerCluster

# Initialize the map at a central point in Boston
boston_map = folium.Map(location=[42.3601, -71.0589], zoom_start=12)
# Latitude and longitude of Boston

# Load your dataset here
bike_stops = Bike_map

# Create a FastMarkerCluster
marker_cluster = FastMarkerCluster(data=bike_stops[['latitude',
'longitude']].values.tolist()).add_to(boston_map)

# Add popup (optional, customize as needed)
for _, stop in bike_stops.iterrows():
    popup = f"{stop['District']}"
    folium.Marker((stop['Latitude'], stop['Longitude']),
popup=popup).add_to(marker_cluster)

# Save map
boston_map.save("bus_train_stops_clusters.html")

# The function `convert_map_to_png` is not a standard function and
would need to be defined if you wish to use it.
# convert_map_to_png(boston_map, 'boston_bus_train_stops')
convert_map_to_png(boston_map, 'Boston_Bike_stations')
```

+
−
Lexington
Malden
Medford
Arlington
Everett
Revere
Nahant
Belmont
Somerville
Chelsea
Charlestown
Winthrop
Waltham
East Watertown
Watertown
Cambridge
East Cambridge
North End
Cambridgeport
Allston
Boston
Newton Corner
Newtonville
Auburndale
Brighton
Back Bay
Fenway-Kenmore
Newton
Newton Centre
Chestnut Hill
Brookline
South Boston
Newton Lower Falls
Waban
Roxbury
Newton Highlands
Newton Upper Falls
Wellesley Hills
Jamaica Plain
Oak Hill
Dorchester
Needham Heights
Roslindale
Hull
Needham
West Roxbury
North Quincy
Mattapan
Suffolk County
Essex County
Plymouth County
Rumney Marsh Reservation
Cutler Park Reservation
MA 1A
US 1
2
14
22
4
48
34
24
86
38
110
20
50
82
8
2
76
118
10
8
30
40
32
18
6
2
Leaflet | Data by © OpenStreetMap, under ODbL.