Import Libraries

In [251]:
```
!pip install googlemaps
!pip install geopandas
!pip3 install geopy
```

```
Requirement already satisfied: googlemaps in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(4.10.0)
Requirement already satisfied: requests<3.0,>=2.20.0 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site
-packages (from googlemaps) (2.28.1)
Requirement already satisfied: charset-normalizer<3,>=2 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/s
ite-packages (from requests<3.0,>=2.20.0->googlemaps) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site
-packages (from requests<3.0,>=2.20.0->googlemaps) (1.26.11)
Requirement already satisfied: certifi>=2017.4.17 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-pa
ckages (from requests<3.0,>=2.20.0->googlemaps) (2022.9.24)
Requirement already satisfied: idna<4,>=2.5 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from requests<3.0,>=2.20.0->googlemaps) (3.3)
Requirement already satisfied: geopandas in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(0.12.2)
Requirement already satisfied: packaging in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages (f
rom geopandas) (21.3)
Requirement already satisfied: pyproj>=2.6.1.post1 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-p
ackages (from geopandas) (3.4.1)
Requirement already satisfied: pandas>=1.0.0 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-package
s (from geopandas) (1.4.4)
Requirement already satisfied: fiona>=1.8 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from geopandas) (1.9.1)
Requirement already satisfied: shapely>=1.7 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from geopandas) (2.0.1)
Requirement already satisfied: click-plugins>=1.0 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-pa
ckages (from fiona>=1.8->geopandas) (1.1.1)
Requirement already satisfied: click~=8.0 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from fiona>=1.8->geopandas) (8.0.4)
Requirement already satisfied: setuptools in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from fiona>=1.8->geopandas) (63.4.1)
Requirement already satisfied: munch>=2.3.2 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from fiona>=1.8->geopandas) (2.5.0)
Requirement already satisfied: cligj>=0.5 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from fiona>=1.8->geopandas) (0.7.2)
Requirement already satisfied: certifi in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages (fro
m fiona>=1.8->geopandas) (2022.9.24)
Requirement already satisfied: attrs>=19.2.0 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-package
s (from fiona>=1.8->geopandas) (21.4.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/sit
e-packages (from pandas>=1.0.0->geopandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages
(from pandas>=1.0.0->geopandas) (2022.1)
Requirement already satisfied: numpy>=1.18.5 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-package
s (from pandas>=1.0.0->geopandas) (1.21.5)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/s
ite-packages (from packaging->geopandas) (3.0.9)
Requirement already satisfied: six in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages (from mu
nch>=2.3.2->fiona>=1.8->geopandas) (1.16.0)
Requirement already satisfied: geopy in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/site-packages (2.3.
0)
Requirement already satisfied: geographiclib<3,>=1.52 in /Users/venkatasaisumanthsadu/opt/anaconda3/lib/python3.9/sit
e-packages (from geopy) (2.0)
```

In [252]:
```
import pandas as pd
import numpy as np
import geopandas
from googlemaps import Client as GoogleMaps
import geopy.distance
```

Download datasets:

Read trips dataset

In [253]:
```
df_trips = pd.read_csv("data/trip_data.csv")
df_trips.columns
```

Out[253]:
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
       'fare_amount', 'tip_amount', 'total_amount'],
      dtype='object')
```

As it was stated that trips during lunchtime (11:30 am - 2 pm) and dinnertime (5 pm - 9 pm) were to restaurants. Let us consider only those datapoints as trips data.

In [254]:
```
df_trips_timed = df_t[((df_t.tpep_dropoff_datetime.str.split(" ",expand=True)[1] > '11:30:00') &
                       (df_t.tpep_dropoff_datetime.str.split(" ",expand=True)[1] < '14:00:00')) |
                      ((df_t.tpep_dropoff_datetime.str.split(" ",expand=True)[1] > '17:00:00') &
                       (df_t.tpep_dropoff_datetime.str.split(" ",expand=True)[1] < '21:00:00'))]

print(len(df_trips_timed))

df_trips_timed = df_trips_timed.reset_index(drop=True)
# df_trips_timed consists of only trips during lunchtime and dinnertime.
df_trips_timed.head()
```

```
33471
```

Out[254]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	t
0	2	2015-01-15 19:05:39 +00:00	2015-01-15 19:23:42 +00:00	1	1.59	-73.993896	40.750111	-73.974785	40.750618	
1	2	2015-01-15 19:05:39 +00:00	2015-01-15 19:32:00 +00:00	1	2.38	-73.976425	40.739811	-73.983978	40.757889	
2	2	2015-01-15 19:05:40 +00:00	2015-01-15 19:21:00 +00:00	5	2.83	-73.968704	40.754246	-73.955124	40.786858	
3	2	2015-01-15 19:05:40 +00:00	2015-01-15 19:28:18 +00:00	5	8.33	-73.863060	40.769581	-73.952713	40.785782	
4	2	2015-01-15 19:05:41 +00:00	2015-01-15 19:20:36 +00:00	1	2.37	-73.945541	40.779423	-73.980850	40.786083	

Reading resturants dataset

In [3]:
```
# google geocoding api key to convert address of a resturant to latitude and longitude
gmaps = GoogleMaps('AIzaSyC0fHL6fvCIdw01BQFqsMX9x7Ol6pKVyZg')
```

In [260]:
```
# read data
df_r = pd.read_csv("data/restaurant_data.csv")
df_r.columns
```

Out[260]:
```
Index(['CAMIS', 'DBA', 'BORO', 'BUILDING', 'STREET', 'ZIPCODE', 'PHONE',
       'CUISINE DESCRIPTION', 'INSPECTION DATE', 'ACTION', 'VIOLATION CODE',
       'VIOLATION DESCRIPTION', 'CRITICAL FLAG', 'SCORE', 'GRADE',
       'GRADE DATE', 'RECORD DATE', 'INSPECTION TYPE'],
      dtype='object')
```

In [261]:
```
# we dont want NaN values primarily in ['BUILDING', 'STREET', 'ZIPCODE', 'CAMIS', 'DBA','BORO', 'CUISINE DESCRIPTION',
# as later these are used for modeling.
df_r_unique = df_r.dropna()
df_r_unique = df_r_unique.reset_index()
```

In [262]:
```
len(df_r_unique)
```

Out[262]:
```
191636
```

In [263]:
```
# concatenating 'Building', 'street' and 'zipcode' to get complete address of resturant.
# This is reuqired to find latitude and longitude using google geocoding api.

df_r_unique['ADDRESS'] = ''
for i in range(len(df_r_unique)):
    df_r_unique['ADDRESS'][i] = df_r_unique['BUILDING'][i] + ' ' + df_r_unique['STREET'][i] + ' ' + str(int(df_r_unique
```

```
/var/folders/dz/k6x51lvd2jv050r966v_h2br0000gn/T/ipykernel_19884/117765336.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df_r_unique['ADDRESS'][i] = df_r_unique['BUILDING'][i] + ' ' + df_r_unique['STREET'][i] + ' ' + str(int(df_r_unique
['ZIPCODE'][i]))
```

In [265]:
```
# in order to join the above two datasets we only require unique addresses and their location.
df_r_unique = df_r_unique.drop_duplicates(subset=['ADDRESS'], keep='last')
df_r_unique.reset_index(inplace = True)
len(df_r_unique)
```

Out[265]: 22489

In [266]:
```
df_r_unique['LONG_GCP'] = ""
df_r_unique['LAT_GCP'] = ""
```

In [287]:
```
for x in range(len(df_r_unique)):
    try:
        geocode_result = gmaps.geocode(df_r_unique['ADDRESS'][x])
        df_r_unique['LAT_GCP'][x] = geocode_result[0]['geometry']['location'] ['lat']
        df_r_unique['LONG_GCP'][x] = geocode_result[0]['geometry']['location']['lng']
    except IndexError:
        print("Address was wrong...")
    except Exception as e:
        print("Unexpected error occurred.", e )
```

In [268]:
```
# lets remove rows whose address couldn't be converted into latitude and longitude as the address is either not
# properly clear or is not found.
df_r_unique2 = df_r_unique[(df_r_unique['LONG_GCP'] != "") | (df_r_unique['LAT_GCP'] != "")].copy(deep=True)
len(df_r_unique2)
```

Out[268]: 22489

Converting trip and resturant dataframes into GeoDataframe using Latitude and Longitude. This is done to join both dataframes on geometry column based on closest proximity

In [270]:
```
df_r_unique2.dropna(subset=['LONG_GCP','LAT_GCP'], inplace = True)
df_r_unique2.reset_index(drop=True, inplace = True)
```

Link the two datasets spatially

The two geodataframes are joined based on the haversine formula, which assumes the earth is a sphere, which results in errors of up to about 0.5% (according to help(geopy.distance))

In [276]:
```
# convert trip dataframe to geodataframe
gdf_trips = geopandas.GeoDataFrame(
    df_t_timed, geometry=geopandas.points_from_xy(df_t_timed.dropoff_longitude, df_t_timed.dropoff_latitude))

# convert resturant dataframe to geodataframe
gdf_resturants = geopandas.GeoDataFrame(
    df_r_unique2, geometry=geopandas.points_from_xy(df_r_unique3.LONG_GCP, df_r_unique3.LAT_GCP))

# join both geodataframes using proximity distance of 50 meters based on geometry column
gdf_combined = gdf_trips.sjoin_nearest(gdf_resturants, how='inner', max_distance=50)

gdf_combined.reset_index(drop=True, inplace = True)

gdf_combined
```

Out[276]:

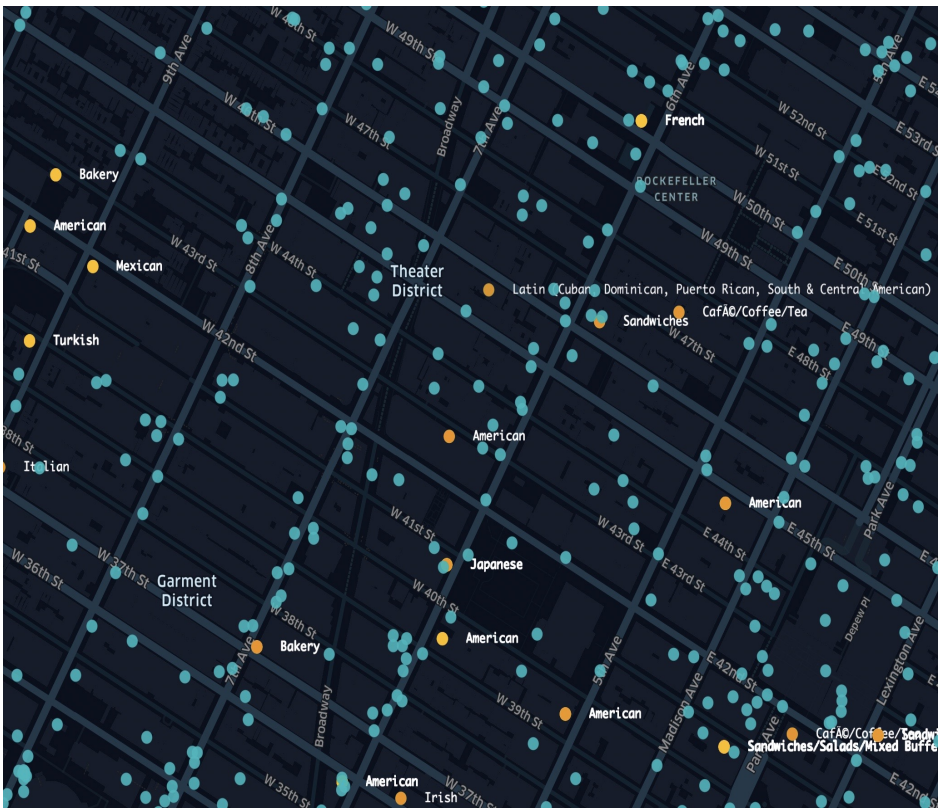
	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitu
0	2	2015-01-15 19:05:39 +00:00	2015-01-15 19:23:42 +00:00	1	1.59	-73.993896	40.750111	-73.974785	40.7506
1	2	2015-01-15 14:00:57 +00:00	2015-01-16 13:47:27 +00:00	1	0.95	-73.990692	40.755806	-73.974998	40.7505
2	1	2015-01-15 12:09:49 +00:00	2015-01-15 12:20:50 +00:00	2	1.10	-73.975876	40.761192	-73.974884	40.7501
3	2	2015-01-15 17:11:55 +00:00	2015-01-15 17:29:33 +00:00	1	1.53	-73.987465	40.732830	-73.974648	40.7503
4	2	2015-01-15 16:38:41 +00:00	2015-01-15 17:40:45 +00:00	1	17.05	-73.782112	40.644650	-73.974907	40.7505
...	...	...	...	...	...	...	...	...	
38832	1	2015-01-15 20:05:57 +00:00	2015-01-15 20:11:00 +00:00	1	1.20	-73.995666	40.724792	-74.007347	40.7114
38833	1	2015-01-15 20:05:58 +00:00	2015-01-15 20:12:40 +00:00	1	1.50	-73.982323	40.767231	-73.972862	40.7859
38834	1	2015-01-15 20:16:20 +00:00	2015-01-15 20:49:15 +00:00	2	5.30	-73.957726	40.773052	-74.001648	40.7184
38835	1	2015-01-15 20:16:21 +00:00	2015-01-15 20:36:12 +00:00	2	11.50	-73.872993	40.774178	-74.004105	40.7209
38836	1	2015-01-15 20:16:21 +00:00	2015-01-15 20:42:17 +00:00	1	10.00	-73.873024	40.774143	-73.957230	40.6728

38837 rows × 35 columns

Creating an exploratory map

The blue dots show the pick location where as the yellow dots show the resturant location with the type of cuisine

they serve.



Exploratory data analysis questions

How far do people travel based on different types of cuisine?

In [278]:
```
len(gdf_combined['CUISINE DESCRIPTION'].unique())
```

Out[278]:
```
82
```

In [279]:
```
gdf_combined['DISTANCE_TRAVELLED'] = ""

for x in range(len(gdf_combined)):
    try:
        coords_1 = (gdf_combined['pickup_latitude'][x], gdf_combined['pickup_longitude'][x])
        coords_2 = (gdf_combined['LAT_GCP'][x], gdf_combined['LONG_GCP'][x])
        gdf_combined['DISTANCE_TRAVELLED'][x] = geopy.distance.geodesic(coords_1, coords_2).miles
    except IndexError:
        print("IndexError", geopy.distance.geodesic(coords_1, coords_2).miles)
    except Exception as e:
        print("Unexpected error occurred.", e)
```

```
/var/folders/dz/k6x51lvd2jv050r966v_h2br0000gn/T/ipykernel_19884/2050848742.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  gdf_combined['DISTANCE_TRAVELLED'][x] = geopy.distance.geodesic(coords_1, coords_2).miles
```

In [281]:
```
gdf_combined.sort_values("DISTANCE_TRAVELLED", ascending=False)['DISTANCE_TRAVELLED']
# we found an extreme outlier in the data, this should be removed.
```

Out[281]:
```
8329     346.794997
34171     20.156912
33283     19.632348
33282     19.632348
26341     19.478686
              ...
36028      0.003003
36027      0.003003
20648      0.002821
20647      0.002821
7713       0.002611
Name: DISTANCE_TRAVELLED, Length: 38837, dtype: object
```

In [282]:
```
gdf_combined = gdf_combined[gdf_combined['DISTANCE_TRAVELLED'] != gdf_combined['DISTANCE_TRAVELLED'].max()]
gdf_combined.reset_index(inplace = True)
```

In [283]:
```
gdf_combined.sort_values("DISTANCE_TRAVELLED", ascending=False)['DISTANCE_TRAVELLED']
```

Out[283]:
```
34170    20.156912
33281    19.632348
33282    19.632348
26340    19.478686
38296    19.314463
             ...
36026     0.003003
36027     0.003003
20647     0.002821
20646     0.002821
7713      0.002611
Name: DISTANCE_TRAVELLED, Length: 38836, dtype: object
```

Average distance travelled

In [258]:
```
gdf_combined['DISTANCE_TRAVELLED'].mean()
```

Out[258]:
```
1.9809642620689494
```

Standard deviatioon of distance travelled

In [259]:
```
gdf_combined['DISTANCE_TRAVELLED'].std()
```

Out[259]:
```
2.350000290970313
```

Based on the mean and standard deviation 67% of the distance travelled are within 4.33 miles

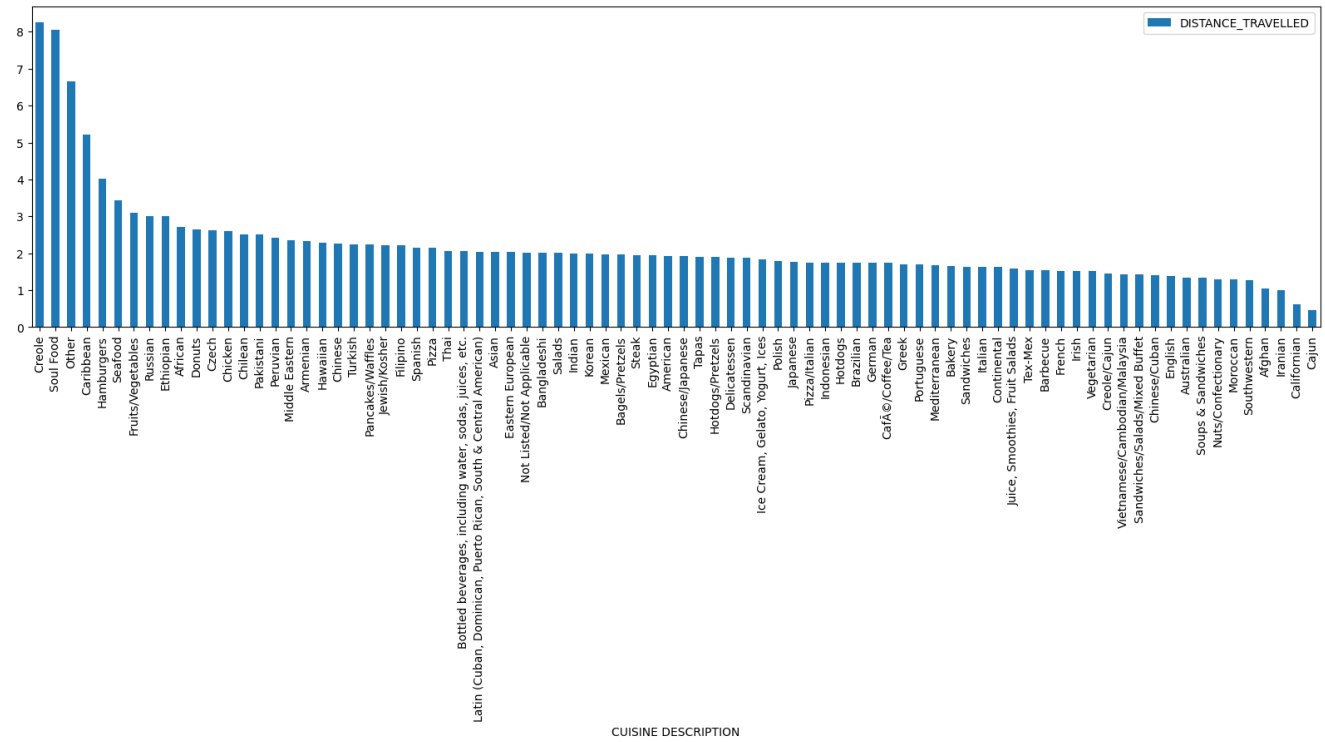
In [242]:
```
# To answer - How far do people travel based on different types of cuisine?
graph_tmp_cusine = gdf_combined.groupby(['CUISINE DESCRIPTION'], as_index = False)['DISTANCE_TRAVELLED'].mean().copy(de
```

In [243]:
```
graph_tmp_cusine.columns
```

Out[243]:
```
Index(['CUISINE DESCRIPTION', 'DISTANCE_TRAVELLED'], dtype='object')
```

In [244]:
```
graph_tmp_cusine.sort_values("DISTANCE_TRAVELLED", ascending=False).plot.bar(x = 'CUISINE DESCRIPTION', y = 'DISTANCE_T
```

Out[244]:
```
<AxesSubplot:xlabel='CUISINE DESCRIPTION'>
```



In [285]:
```
graph_tmp_cusine.sort_values("DISTANCE_TRAVELLED", ascending=False)
```

Out[285]:

	CUISINE DESCRIPTION	DISTANCE_TRAVELLED
22	Creole	8.262501
71	Soul Food	8.049429
57	Other	6.667293
15	Caribbean	5.205806
36	Hamburgers	4.027879
...	...	...
73	Southwestern	1.275113
0	Afghan	1.047403
43	Iranian	0.992074
14	Californian	0.623562
13	Cajun	0.468801

82 rows × 2 columns

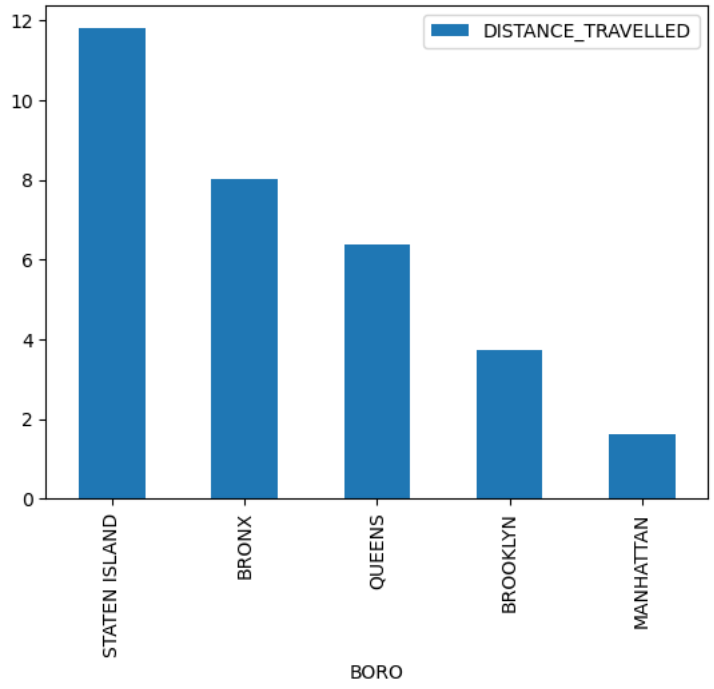
As seen from above graph - on an average the highest distance traveled is 8.262501 miles. But the acutal highest distance traveled is close to 20 miles. The bar graph shows the average distance travelled by people based on cuisine.

How far do people travel based on the borough where the restaurant is located ?

In [245]:
```
graph_tmp_boro = gdf_combined.groupby(['BORO'], as_index = False)['DISTANCE_TRAVELLED'].mean()
```

In [246]:
```
graph_tmp_boro.sort_values("DISTANCE_TRAVELLED", ascending=False).plot.bar(x = 'BORO', y = 'DISTANCE_TRAVELLED')
```

Out[246]:
```
<AxesSubplot:xlabel='BORO'>
```



In [247]:
```
gdf_combined.groupby(['BORO'], as_index = False)['DISTANCE_TRAVELLED'].mean()
```

Out[247]:

	BORO	DISTANCE_TRAVELLED
0	BRONX	8.032041
1	BROOKLYN	3.726855
2	MANHATTAN	1.640221
3	QUEENS	6.384624
4	STATEN ISLAND	11.799483

The Staten Island have more people travelling long distances on average. Then followed by Bronx.

How far do people travel based on meal time?

In [248]:
```
gdf_combined.groupby((gdf_combined.tpep_dropoff_datetime.str.split(" ",expand=True)[1] > '11:30:00') & (gdf_combined.tp
```

Out[248]:
```
1
False    2.075457
True     1.799271
Name: DISTANCE_TRAVELLED, dtype: float64
```

True - corresponds to Lunch time and False to Dinner.

As per the above results, people travel long distances for dinner with respect to luch time.

Describe how you would set up a predictive model

To set up a predictive model given the two datasets of trips and restuarants, we can follow the following steps:

1. Define the problem - also state the assumptions made
2. Gather data
3. Clean and preprocess the data
4. Split the data
5. Choose a model
6. Train the model
7. Evaluate the model
8. Adjust the model
9. Use the model

Define the problem

The problem statement is to predict cuisine type to be visited by a taxi rider based on information avilable in trip and restaurant datasets. The final combined dataframe contains 33,405 rows.

Not to forget that the data is collected on 01/15/2015 for trips and 2014 - 2017 records are present for restaurants. Also collected from 5 Borough's in USA. The final predictive model works well is prediction is made on data point drawn from this distribution.

Gather data

Collect the data you need to train your model. This may involve finding a dataset online, extracting data from a database, or collecting data using surveys or other means.

we can predict cuisine based on weather, price range of restaurant, pickup location, drop location, distance travelled, passenger count, restaurant name.

- `weather` : Given the drop time and location, we could fetch the weather information as a discrete variable from public APIs. One such API is the OpenWeather (https://openweathermap.org/api)
- `restaurant name` : Given the drop-location, we can infer the closest restaurant around the 50m radius. Restaurant name indicates standard knowledge about cuisines. Google's Places (https://developers.google.com/maps/documentation/places/web-service/search-nearby) API is the best fit for this task.
- `price range` : As the price range of a restaurant is a direct indicator of its primary cuisine, we fetch the price range of the restaurant from the same API.
- `locations` : The pickup and drop locations are mapped to the geohash using the geo-hasing libraries.
- `passenger count` : Passenger count indicates the number of people riding and visiting the restaurant.
- `distance travelled` : Distance tarvelled is discretized into three ordinaly categories which provides a more robust feature representation.

Clean and preprocess the data

Several possible cleaning startegies help for a simple and faster training.

1. Removing the outliers for the distance travelled improves the feature value which can be done by a blox plot.
2. The restaurant names are standardised by lower casing and removing any special characters
3. Any samples contaning empty feature values are ommited.

Split the data

I will split data into 70% for train, 15% for validation and 15% for test. As the data is not huge this split is considered.

Choose a model

Few models which I would try:

Random Forest: As the features mentioned above which would be used for predictions does not necessarily have category type data in all we can random forest which is an ensemble model that uses multiple decision trees to make predictions. Random forests can handle large datasets with high dimensionality and are robust against overfitting. It works by creating a large number of decision trees and averaging their predictions to obtain a more accurate prediction.

Naive Bayes: Naive Bayes assumes that the predictors are independent, all features in our case are indenpendent of each other.

Support Vector Machines (SVMs): They work well with high-dimensional datasets and can handle non-linear data.

Neural Networks: They can be used for classification problems and can handle complex relationships between variables.

Neural networks require a large amount of data to train and can be prone to overfitting, but they are capable of achieving high accuracy.

Selected Methodology

- I would first discretize the above mentioned features. For instance, the distance travelled could be categorised into three categories. The pickup and drop locations can be categorised by geo-hash.
- Each feature has its own micro-modeling process. The restaurant name is better processed by a character level encoding model. The goehash can be processed by a word2vec type model. All the micro-models produce an embedding which are then concatenated or processed using attention to give a final representation vector of the features.
- The final embedding is then passed on to a classification head to predict the cuisine type.

Train the model

Once you've chosen a model, you'll need to train it using the training set. This involves tuning the hyperparameters to adjust the mode until it fits the data as closely as possible.

Evaluate the model

We initially evaluate the model on validation dataset and tune the hyperparameters until we get better results, later the final evaluation is done on test set to check the performance.

I will use F1 score metric as the classes are imbalanced.

Adjust the model

If the model isn't performing well, you may need to adjust it. This could involve trying different types of models, adjusting the parameters of the existing model, or modifying the data to make it more suitable for the model.

Use the model

Once you're satisfied with the performance of the model, you can start using it to make predictions.

In [ ]:

In [ ]: