nyc_taxi_duration_prediction

December 4, 2023

0.1 1. Introduction

We aim to tackle the challenge of predicting taxi ride durations in New York City based on starting and stopping coordinates. This research question is very relevant for urban commuters and transportation systems and incorporates the myriad factors influencing taxi ride times, including traffic patterns, congestion, time of day, and external variables such as events or weather conditions. The strengths of machine learning methods align well with the intricacies of this problem, and allow us to uncover more nuanced relationships within the data. Beyond individual convenience, the ability to predict taxi ride times carries practical implications for optimizing taxi fleet management, resource allocation, and enhancing overall traffic flow in the city.

Kaggle published the dataset we are working on for a coding competition, so over one thousand other groups have investigated this research question. Successful teams performed feature engineering to create fields such as month, day, hour, day of the week, and used models such as Random Forest Regression, Extra Trees Regression, PCA, XGBoost, linear regression, and Light GBM.

The original dataset also includes fields for the taxi driver, the number of passengers, and whether the trip time was recorded in real time. The second dataset contains weather information with timestamps, temperature, precipitation, cloud cover, and wind information at every hour. The NYC taxi cab dataset was published by NYC Taxi and Limousine Commission (TLC) in Big Query on Google Cloud Platform and is well documented and densely populated with over one million data points. The weather dataset is much more sparse and has a much larger time range than needed to match the NYC Taxi data. These datasets provide a good source for addressing our research questions because they extensively cover NYC taxi travel for a significant time period. In short, the data allows us to effectively identify significant features impacting taxi trip duration and develop a robust predictive model for accurate estimations.

While our primary investigation is focused on determining taxi cab trip duration, we are also interested in several other questions. What dates, days of the week, and times of day are most busy? Where are the most popular destinations? What factors influence taxi trip time the most? Furthermore, the multifaceted exploration of temporal, spatial, and environmental influences on travel durations in NYC presents a well-rounded analysis to reveal comprehensive insights into travel behaviors. Our data is well suited for our research question in exploring and predicting taxi trip duration, and has a single, clear answer. A process of machine learning model development will help us go one step further and develop a robust predictive model to estimate and understand trip durations accurately.

0.2 2. Feature Engineering

In our pursuit to enhance the predictive power of our model, we identified the necessity for additional features in our dataset. This realization led to the development of three distinct feature groups: datetime, distance, and weather.

0.2.1 2.0 Data Cleaning

The taxi cab duraction dataset from Kaggle contained several outlier data that required to removal. For example, some trips lasted 1 second, and others lasted over 980 days. To prevent erroneous data, we removed all rows where trip_duration was in the .005 quantile or less than 60 seconds. The dataset also contained outliers in the pick up and drop off locations that were far outside New York City, which we fixed by removed any point outside of city limits.

0.2.2 2.1 Datetime

One of the most important features in our dataset is passenger pickup time, originally represented in a string in the format `YYYY-MM-DD HH:MM:SS`. We created multiple time features from this column including `pickup_month`, one-hot encoded `pickup_day`, `pickup_hour`, and `pickup_minute`. We also added other versions of these data points including one-hot encoded `pickup_period`, `pickup_hour_sin`, `pickup_hour_cos`, and `pickup_datetime_norm`.

2.0.0 Pickup Period The feature `pickup_period` captures the time of day when passengers were picked up in one of four periods: morning (6:00 AM to 12:00 PM), afternoon (12:00 PM to 6:00 PM), evening (6:00 PM to 12:00 AM), and night (12:00 AM to 6:00 AM). These divisions align intuitively with significant periods of the day for taxi services, such as morning rush hours and evening nightlife.

df[‘pickup_period’] = pd.cut(df[‘pickup_hour’], bins=[-1, 6, 12, 18, 24], labels=[‘night’, ‘morning’, ‘afternoon’, ‘evening’]) df = pd.get_dummies(df, columns=[‘pickup_period’], drop_first=True)

2.0.1 Pickup Period Sine/Cosine We applied a circular encoding to the hour of the day to account for the cyclical nature of the hours of the day. We created `pickup_hour_sin` and `pickup_hour_cos` features using sine and cosine transformations, that avoid discontinuities (such as the start and end of a day).

$$\text{hour\_sin} = \sin\left(\frac{2\pi \cdot \text{pickup\_hour}}{24}\right) \quad \text{hour\_cos} = \cos\left(\frac{2\pi \cdot \text{pickup\_hour}}{24}\right). \tag{1}$$

2.0.2 Pickup Datetime Norm The final feature we created in the datetime feature grouping was `pickup_datetime_norm` to represent the normalized pickup datetime. This feature converts the pickup datetime from nanoseconds to seconds, then scaled the value by the maximum to place all the values between 0 and 1.

df[‘pickup_date_time_norm’] = pd.to_datetime(df[‘pickup_date_time_norm’]).view(‘int64’) // 10**9 df[‘pickup_date_time_norm’] = (df[‘pickup_date_time_norm’]- df[‘pickup_date_time_norm’].min()) / (df[‘pickup_date_time_norm’].max() - df[‘pickup_date_time_norm’].min()).

0.2.3 2.1 Distance

We created two features that estimate distance between pickup and drop off locations: the Manhattan distance and the average distances between local coordinate clusters.

2.1.0 Manhattan Distance We include the Manhattan Distance feature because of its grid based metric. Many of the streets of New York are laid out in a grid-like fashion, so this metric can better approximate road distances than the Euclidean distance. The Manhattan distance is also more simple and interpretable, since it computes the sum of the absolute values of the differences between the x and y coordinates of the two points. We calculated the Manhattan distance by first converting the pickup and dropoff coordinate points into radians and using the following formula:

$$\text{Manhattan Distance} = R \cdot (|\text{pickup\_latitude} - \text{dropoff\_latitude}| + |\text{pickup\_longitude} - \text{dropoff\_longitude}|) \tag{2}$$

where R is the radius of the Earth in kilometers (6371 km).

2.1.1 KMeans Clustering Average Duration Along with incorporating distance metrics and weather information, we also added a duration feature that acts as an initial estimate for the model's actual trip duration prediction. To do this, we fit a pickup and dropoff KMeans clustering model with 200 clusters. Images of these are shown in section `4. Data Visualization`. We then labeled each pickup location and drop off location in the data with its respective cluster label. By grouping the data by these cluster pairs, we computed the average trip_duration between each cluster pair and merged this onto the original dataframe. See Appendix for the full code implementation.

0.2.4 2.2 Weather

When considering potential features to add to our dataset, accounting for the effect of local weather on taxi ride tme was one of the most obvious additions to include. For example, if it is raining or snowing, there will be more traffic on the roads, and more people who would normally walk would prefer a taxi. A dataset created by Kaggle user @Aadam contains a miriad of weather data for New York City between 2016 and 2022 on an hourly basis. This dataset includes features such as temperature (in Celcius), precipition (in mm), cloud cover (low, mid, high, and total), wind speed (in km/h), and wind direction. We decided to use temperature, precipitation, and total cloud cover as features in our dataset with a simple join on the pickup datetime, rounded to the nearest whole hour.

[4]:
```
X, y = get_X_y(force_clean=True)    # All feature engineering is done in the␣
 ↪function get_X_y, found in appendix
feature_X = generate_features(X, y) # generates features adds more features to␣
 ↪the data, including weather
feature_X.shape, y.shape
```

[4]:
```
((1441615, 27), (1441615,))
```

[36]:
```
print("Example of the features for the first row of the data:")
print("{:>20} = {:<20} {:>20} = {:<20}".format("Feature", "Value", "Feature",␣
 ↪"Value"))
```

```
for i in range(0, len(feature_X.columns)-1, 2):
    col1, val1, col2, val2 = ( feature_X.columns[i], feature_X.iloc[0][i],␣
 ↪feature_X.columns[i + 1], str(feature_X.iloc[0][i + 1]))
    print(f"{col1:>20} = {val1:<20} {col2:>20} = {val2:<20}")
```

```
Example of the features for the first row of the data:
            Feature = Value                          Feature = Value
           vendor_id = 1                      pickup_datetime = 2016-03-14
17:24:55
     passenger_count = 1                     pickup_longitude =
-73.98215484619139
     pickup_latitude = 40.76793670654297    dropoff_longitude =
-73.96463012695312
    dropoff_latitude = 40.765602111816406          pickup_month = 3
   pickup_day_Monday = 1                   pickup_day_Saturday = 0
   pickup_day_Sunday = 0                   pickup_day_Thursday = 0
  pickup_day_Tuesday = 0                  pickup_day_Wednesday = 0
         pickup_hour = 17                        pickup_minute = 24
pickup_period_morning = 0                 pickup_period_afternoon = 1
pickup_period_evening = 0                      pickup_hour_sin =
-0.9659258262890683
     pickup_hour_cos = -0.25881904510252063 pickup_datetime_norm =
0.40508581306349817
         distance_km = 2.2082549595298886    temperature_2m (°C) = 6.4
  precipitation (mm) = 0.2                       cloudcover (%) = 100.0
```

0.3 3. Feature Selection

As seen is section 2, we created several new features in addition to those already in the dataset. We also consider which features are unnecessary or unhelpful.

0.3.1 3.1 L^1 Regularization

To figure out which of our features is most important, we utilized L^1 regularization since L^1 regularization naturally sets unneeded feature coefficents to 0, and typically out performs step-wise feature removal.

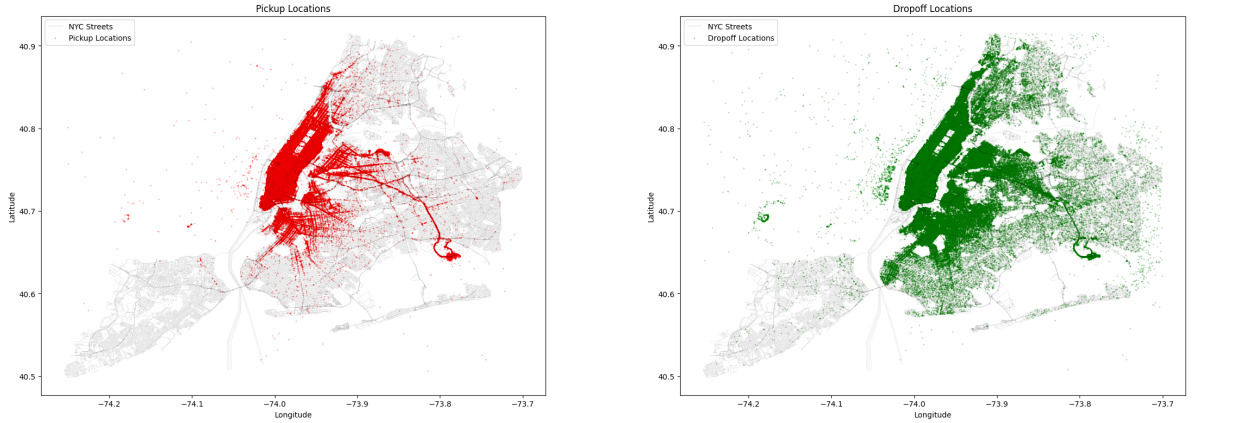
[2]:
```
# lasso_feature_selection performs feature selection using the LassoLarsIC␣
 ↪method
lasso_feature_selection(X, y)
```

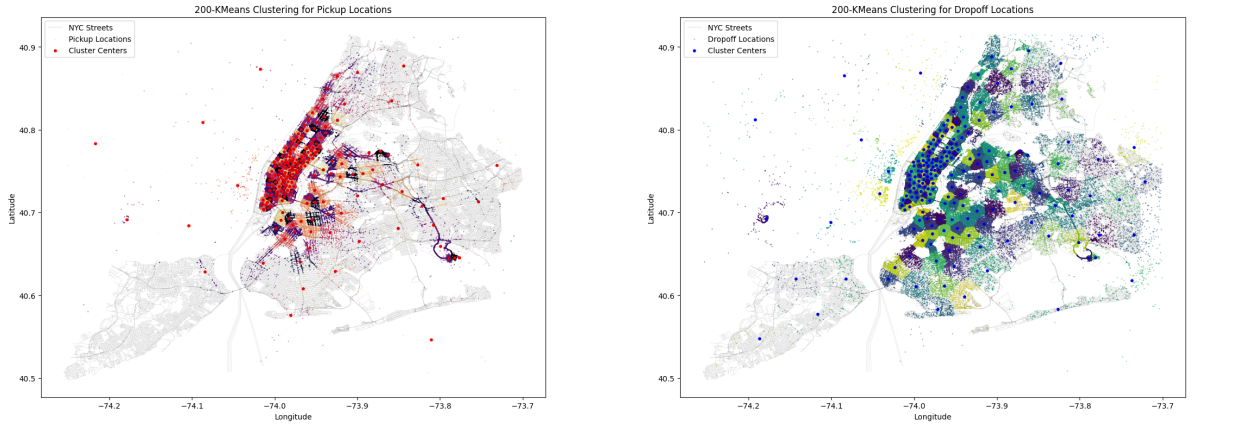
```
Optimal Alpha     : 1.0806
Optimal BIC       : 18056884.5229
Lasso Coefficients: [ 0.    , 0.    , 0.    ,  0.    ,  0.    ,  0.    ,  0.
, 0.    , 0.    , 0.    ,
                      0.    , 0.    , 0.    ,  0.0169,  0.    ,  0.    ,  0.
, 0.    , 0.    , 0.    ,
                     -0.0345, -0.1089, 0.    ,  0.0147,  0.0043]
```

```
Important Features: ['pickup_minute', 'distance_km', 'temperature_2m (°C)',
'cloudcover (%)', 'avg_cluster_duration']
```

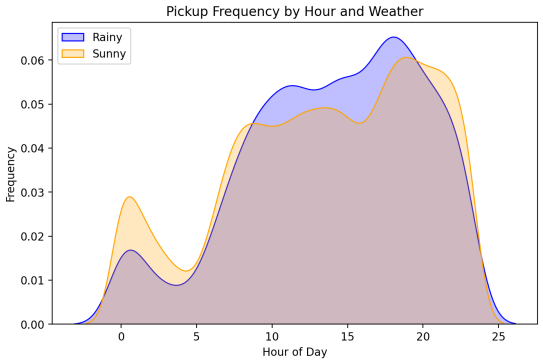
0.4 4. Data Visualization

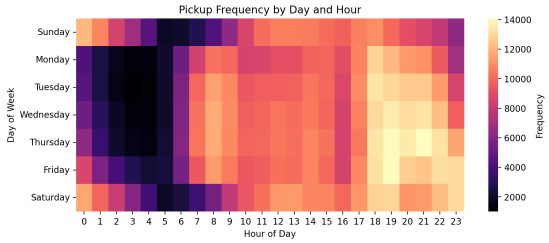
In the following figure, the top two graphs visualize the pickup and dropoff locations overlaid over a map of NYC. The bottom two graphs shows the pickup and dropoff locations clustered into groups using K-means clustering. The pickup locations are more heavily clustered around downtown (Manhattan), while the dropoff locations are more evenly distributed throughout the city.





We can also learn about the factors that influence a New Yorker's decision to take a taxi from the data. For example, in the figure below, the graph on the left displays the most popular times of day to hail a taxi, which peeks around 6:00 PM. We also see that poor weather encourages more people to take a taxi during the day when people are more likely to be returning from their daily activities, but less likely to choose to go out at night in the first place. The graph on the right shows the most popular days of the week for taxis, which peeks on Friday and Saturday and during the evenings of the weekdays.





0.5 5. Modeling and Results

1. Analyze the data using the techniques discussed in class.
2. (Markdown) Explain what research questions you can answer using the techniques presented this semester.

Our primary research question is to predict the duration of a taxi ride in New York City. To achieve this goal, we implemented a variety of machine learning models, including Lasso regression, random forest regression, XGBoost, and LightGBM. We explain our implementation, training, optimization, and results for each model below.

0.5.1 5.1 Light GBM with Grid Search

We implemented a grid search over hyperparameters on LightGBM to identify the best version of this model to predict taxi trip duration. This model that runs quickly with the high dimensional data in our dataset.

[4]:
```
# The above function takes 45 minutes to run and produces the following results
lightgbm_hyperparameter_search(X, y)
```

```
{'Best parameters from grid search': {'boosting_type': 'gbdt', 'learning_rate':
0.01, 'max_depth': 20,
                                     'n_estimators': 100, 'num_leaves': 30,
'reg_alpha': 0.1, 'reg_lambda': 0.5},
'Best RMSE': 606.9699}
```

0.5.2 5.2 Lasso Regression

[6]:
```
# Lasso Regression is quite quick. This took only a few seconds.
lasso_regression_model(optimal_alpha=1.0806)
```

```
{'RMSE: 606.9680'}
```

0.5.3 5.3 XGBoost

0.5.4 5.4 Random Forests

0.5.5 5.5 Model Hyperparameters

Parameter	LightGBM Values	XGBoost Values	RF Values	Lasso Values
boosting_type	gbdt			
learning_rate	0.01			
max_depth	20			
n_estimators	100			
num_leaves	30			
reg_alpha	0.1			1.0806
reg_lambda	0.5			
Best MSE	606.9699			606.9680

0.6 6. Interpretation

(Add some nonsense here about interpretation and stuff.) 1. (Markdown) Analyze the data, draw conclusions, and effectively communicate your main observations and results. 2. (Markdown) Explain the results of your analysis, whether the results are meaningful, and why you chose the tools that you used.

[ ]:

0.7 7. Ethical Implications

Our research involves analyzing a large dataset created by tracking some of the life-style patterns of real people living in New York during 2016, raising concerns about privacy and responsible data usage for individual behaviors, locations, and travel patterns. Our dataset and model protect this data by excluding all personally identifiable information to ensure that only aggregate information can be meaningful, and the patterns of individuals remain indiscernible.

The risk of misinterpretation or misuse of our predictive models is very real. Users could misunderstand predictions, leading to inappropriate decision-making. Our predictive model may contain and inadvertently perpetuate biases that we are unaware of, such as inappropriate associations with certain neighborhoods and taxis. Furthermore, users might misunderstand the predictive and uncertain nature of the model and treat its estimates as certainties. For instance, if taxi companies or transportation authorities were to make decisions solely based on the model's predictions without considering broader traffic management strategies, it could inadvertently lead to concentrated traffic, worsening congestion in certain areas. If our model were deployed in conjunction with algorithms influencing taxi availability, the system might inadvertently create self-fulfilling feedback loops, disproportionately affecting certain areas or demographics.

To address these issues, clear communication about the model's limitations, potential biases, and intended use is crucial. Providing educational resources, user-friendly interfaces, adequate documentation, and implementing fairness-aware algorithms can contribute to responsible and ethical deployment. Regular assessments, periodic audits, and interventions are necessary to avoid reinforcing existing biases. We have also considered ethical responsibilities such as the responsible

disclosure of findings, ensuring the public benefits from the research, and avoiding any unintentional harm. Active engagement with potential stakeholders and the community can further help address concerns and foster ethical practices.

0.8 Appendix

0.8.1 A.1 Code

[ ]:
```
# python imports
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
import pickle
from copy import deepcopy
from sklearn.cluster import KMeans

# Native imports
from py_files.features import generate_features
from py_files.data_manager import get_X_y, get_nyc_gdf
```

[ ]:
```
# KMeans Clustering code
def kmeans_pickup_dropoff_model(df, n_clusters=200):
    # fit the kmeans models and label each pickup and dropoff location by its␣
 ↪cluster
    kmeans_pickup = (KMeans(n_clusters=n_clusters)
        .fit(df.loc[:, ['pickup_longitude', 'pickup_latitude']].values))
    kmeans_dropoff = (KMeans(n_clusters=n_clusters)
        .fit(df.loc[:, ['dropoff_longitude', 'dropoff_latitude']].values))
    df['pickup_cluster'] = kmeans_pickup.predict(df[['pickup_longitude',␣
 ↪'pickup_latitude']].values)
    df['dropoff_cluster'] = kmeans_dropoff.predict(df[['dropoff_longitude',␣
 ↪'dropoff_latitude']].values)

    # compute the average duration between each cluster and merge this onto the␣
 ↪original dataframe
    group_durations = (df
        .groupby(['pickup_cluster', 'dropoff_cluster'])['trip_duration']
        .mean()
        .reset_index()
        .rename(columns={'trip_duration': 'avg_cluster_duration'}))
    df = pd.merge(
        left=df, right=group_durations, how='left',
        left_on=['pickup_cluster', 'dropoff_cluster'],␣
 ↪right_on=['pickup_cluster', 'dropoff_cluster'])

    # fill the missing values with the mean of the average duration from␣
 ↪cluster to cluster
```

```
    df['avg_cluster_duration'] = df['avg_cluster_duration'].
 ↪fillna(df['avg_cluster_duration'].mean())
    df.drop(columns=['pickup_200_cluster', 'dropoff_200_cluster',␣
 ↪'trip_duration'], inplace=True)
    return df
```

[ ]:
```
# lasso feature selection
def lasso_feature_selection(X, y):
    lasso_lars_ic = make_pipeline(StandardScaler(with_mean=False),␣
 ↪LassoLarsIC(criterion="bic", normalize=False)).fit(X, y)

    results = pd.DataFrame(
        {
            "alphas": lasso_lars_ic[-1].alphas_,
            "BIC criterion": lasso_lars_ic[-1].criterion_,
        }
    ).set_index("alphas")

    optimal_alpha = results[results['BIC criterion'] == results['BIC␣
 ↪criterion'].min()].index

    # Train a Lasso model with the optimal alpha for feature selection
    lasso = linear_model.Lasso(alpha=optimal_alpha)
    lasso.fit(X, y)

    return {'Optimal Alpha': optimal_alpha.values[0], 'Optimal BIC': results.
 ↪loc[optimal_alpha].values[0].tolist()[0],
            'Lasso Coeffs': lasso.coef_.round(4), 'Important Features': X.
 ↪columns[lasso.coef_ != 0].values}
```

[ ]:
```
# LightGBM Hyperparameter Selection
def lighhtgbm_hyperparameter_search(X, y):
    # Train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # Create param grid
    param_grid = {
        'boosting_type': ['gbdt', 'dart'],
        'num_leaves': [30, 40],
        'learning_rate': [0.01, 0.05],
        'n_estimators': [100, 200],
        'max_depth': [10, 20],
        'reg_alpha': [0.1, 0.5],
        'reg_lambda': [0.1, 0.5],
    }

    # LightGBM
```

```
    lgb_train = lgb.LGBMRegressor()

    # Grid search
    grid_search = GridSearchCV(estimator=lgb_train, param_grid=param_grid,␣
 ↪cv=3, scoring='neg_root_mean_squared_error', verbose=1)
    grid_search.fit(X_train, y_train)

    # Validate
    y_pred = grid_search.predict(X_test)

    return {'Best parameters from grid search': grid_search.best_params_, 'Best␣
 ↪RMSE': mean_squared_error(y_test, y_pred, squared=False)}
```

[ ]:
```
# Lasso Regression Model
def lasso_regression_model(optimal_alpha):
    # Important features selected from Lasso Lars IC Feature Selection
    important_features = ['pickup_minute', 'distance_km', 'temperature_2m␣
 ↪(°C)', 'cloudcover (%)', 'avg_cluster_duration']

    # Create dataframe of important features
    X2 = X[important_features]

    # Get test train split
    X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size=0.2,␣
 ↪random_state=42)

    # Create and fit the model.
    model = linear_model.Lasso(alpha=optimal_alpha)
    model.fit(X_train, y_train)

    # Predict on test data and compute RMSE
    y_pred = model.predict(X_test)
    return {'RMSE': mean_squared_error(y_test, y_pred, squared=False)}
```

[ ]:
```
# constants/parameters for this code cell
SHOW_PLOTS = True
LOAD_SAVED_KMEANS_MODELS = True

# load in the cleaned training data and the NYC geopandas dataframe
# with all of the NYC streets
X, y = get_X_y(force_clean=True)
nyc_gdf = get_nyc_gdf()

#########################
# PLOT PICKUP LOCATIONS #
#########################
def plot_pickup_locations(X):
```

```
    # plot the nyc streets
    plt.gcf().set_dpi(500)
    nyc_gdf.plot(linewidth=0.1, edgecolor='black', figsize=(12, 12), alpha=0.5,␣
 ↪label="NYC Streets")

    # plot the pickup locations as a scatter plot on top of the nyc streets
    plt.scatter(X['pickup_longitude'], X['pickup_latitude'], c='red', alpha=0.
 ↪75, s=0.1, label="Pickup Locations")
    leg = plt.legend(loc='upper left')
    for lh in leg.legend_handles:
        lh.set_alpha(1)
    plt.title("Pickup Locations")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")

    # save the plot
    plt.savefig("images/pickup_locations_save.png")
    plt.show() if SHOW_PLOTS else plt.clf()

############################
# PLOT DROPOFF LOCATIONS #
############################
def plot_dropoff_locations(X):
    # plot the nyc streets
    plt.gcf().set_dpi(500)
    nyc_gdf.plot(linewidth=0.1, edgecolor='black', figsize=(12, 12), alpha=0.5,␣
 ↪label="NYC Streets")

    # plot the dropoff locations as a scatter plot on top of the nyc streets
    plt.scatter(X['dropoff_longitude'], X['dropoff_latitude'], c='green',␣
 ↪alpha=0.75, s=0.1, label="Dropoff Locations")
    leg = plt.legend(loc='upper left')
    for lh in leg.legend_handles:
        lh.set_alpha(1)
    plt.title("Dropoff Locations")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")

    # save the plot
    plt.savefig("images/dropoff_locations_save.png")
    plt.show() if SHOW_PLOTS else plt.clf()

######################
# KMEANS CLUSTERING #
######################
def kmeans_pickup_dropoff_predict(df, n_clusters=200):
    df = deepcopy(X)
```

```
    # load kmeans_pickup and kmeans_dropoff from the models folder using pickle
    if LOAD_SAVED_KMEANS_MODELS:
        with open("models/kmeans_200_pickup.pkl", "rb") as file:
            kmeans_200_pickup = pickle.load(file)
        with open("models/kmeans_200_dropoff.pkl", "rb") as file:
            kmeans_200_dropoff = pickle.load(file)


    # fit kmeans_pickup and kmeans_dropoff with 200 clusters
    else:
        n_clusters = 200
        kmeans_pickup = (KMeans(n_clusters=n_clusters)
            .fit(df.loc[:, ['pickup_longitude', 'pickup_latitude']].values))
        kmeans_dropoff = (KMeans(n_clusters=n_clusters)
            .fit(df.loc[:, ['dropoff_longitude', 'dropoff_latitude']].values))

        # save the models to pickle files for loading later
        with open("models/kmeans_200_pickup.pkl", "wb") as file:
            pickle.dump(kmeans_pickup, file)
        with open("models/kmeans_200_dropoff.pkl", "wb") as file:
            pickle.dump(kmeans_dropoff, file)

    # predict the clusters for each pickup and dropoff location
    df['pickup_200_cluster'] = kmeans_200_pickup.
 ↪predict(df[['pickup_longitude', 'pickup_latitude']].values)
    df['dropoff_200_cluster'] = kmeans_200_dropoff.
 ↪predict(df[['dropoff_longitude', 'dropoff_latitude']].values)

    # get the centers
    pickup_200_centers = kmeans_200_pickup.cluster_centers_
    dropoff_200_centers = kmeans_200_dropoff.cluster_centers_
    return df, pickup_200_centers, dropoff_200_centers

#######################################
# PLOT PICKUP LOCATIONS WITH CLUSTERS #
#######################################
def plot_cluster_pickup(df, pickup_200_centers):
    # plot the nyc streets
    plt.gcf().set_dpi(500)
    nyc_gdf.plot(linewidth=0.1, edgecolor='black', figsize=(12, 12), alpha=0.5,␣
 ↪label="NYC Streets")

    # plot the cluster locations and the pickup locations color-coded
    # to their associated cluster
```

```
    plt.scatter(df['pickup_longitude'], df['pickup_latitude'],␣
 ↪c=df['pickup_200_cluster'], cmap='magma', alpha=1.0, s=0.1, label="Pickup␣
 ↪Locations")
    plt.scatter(pickup_200_centers[:, 0], pickup_200_centers[:, 1], c='red',␣
 ↪alpha=1, s=10, label="Cluster Centers")
    leg = plt.legend(loc='upper left')
    for lh in leg.legend_handles:
        lh.set_alpha(1)
    plt.title("200-KMeans Clustering for Pickup Locations")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")

    # save the plot
    plt.savefig("images/kmeans_200_pickup_save.png")
    plt.show() if SHOW_PLOTS else plt.clf()

########################################
# PLOT DROPOFF LOCATIONS WITH CLUSTERS #
########################################
def plot_cluster_dropoff(df, dropoff_200_centers):
    # plot the nyc streets
    plt.gcf().set_dpi(500)
    nyc_gdf.plot(linewidth=0.1, edgecolor='black', figsize=(12, 12), alpha=0.5,␣
 ↪label="NYC Streets")

    # plot the cluster locations and the pickup locations color-coded
    # to their associated cluster
    plt.scatter(df['dropoff_longitude'], df['dropoff_latitude'],␣
 ↪c=df['dropoff_200_cluster'], cmap='viridis', alpha=1.0, s=0.1,␣
 ↪label="Dropoff Locations")
    plt.scatter(dropoff_200_centers[:, 0], dropoff_200_centers[:, 1], c='blue',␣
 ↪alpha=1, s=10, label="Cluster Centers")
    leg = plt.legend(loc='upper left')
    for lh in leg.legend_handles:
        lh.set_alpha(1)
    plt.title("200-KMeans Clustering for Dropoff Locations")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")

    # save the plot
    plt.savefig("images/kmeans_200_dropoff_save.png")
    plt.show() if SHOW_PLOTS else plt.clf()
```