

Project Description: ¶

This project is based New York Taxi Fare Prediction on Kaggle. The objective of this project is predicting the fare of a taxi ride given a pickup and a drop off location.

1. Cleaning the Data:

Cleaning the data:

1. The minimal fare_amount is negative. As this does not seem to be realistic rows with negative fares are dropped.
2. Removed missing data: As this dataset is huge, removing datapoints with missing data probably has no effect on the models being trained.
3. Removed the datapoints with latitude and longitude values outside the boundary of NYC. Approximate boundaries obtained by observing test data boundaries and wikipedia:
`'min_lng':-74.27,'min_lat':40.58,'max_lng':-72.99, 'max_lat':41.8`
4. Removed rows with count of passengers outside the range of 1 to 6.

```
In [1]: # Helper functions
import math
# This function is based on https://stackoverflow.com/questions/27928/
# calculate-distance-between-two-latitude-longitude-points-haversine-formula
# return distance in miles
def distance(lat1, lon1, lat2, lon2):
    p = math.pi/180 # Pi/180
    a = 0.5 - np.cos((lat2 - lat1) * p)/2 + np.cos(lat1 * p) * np.cos(lat2 * p)
    ) * (1 - np.cos((lon2 - lon1) * p)) / 2
    return 0.6213712 * 12742 * np.arcsin(np.sqrt(a))

def filter_on_location(df, loca):
    return (df.pickup_longitude >= loca[0]) & (df.pickup_longitude <= loca[1])
& \
    (df.pickup_latitude >= loca[2]) & (df.pickup_latitude <= loca[3]) &
\
    (df.dropoff_longitude >= loca[0]) & (df.dropoff_longitude <= loca[1])
] & \
    (df.dropoff_latitude >= loca[2]) & (df.dropoff_latitude <= loca[3])

def filter_passenger_count(df):
    return (df.passenger_count >= 1) & (df.pickup_longitude <= 6)

def prepare_time_features(df):
    df['pickup_datetime'] = df['pickup_datetime'].str.replace(" UTC", "")
    df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'], format='%Y-%
m-%d %H:%M:%S')
    df['hour_of_day'] = df.pickup_datetime.dt.hour
    df['week'] = df.pickup_datetime.dt.week
    df['month'] = df.pickup_datetime.dt.month
    df["year"] = df.pickup_datetime.dt.year
    df['day_of_year'] = df.pickup_datetime.dt.dayofyear
    df["weekday"] = df.pickup_datetime.dt.weekday
    df["day_of_month"] = df.pickup_datetime.dt.day
    # To check
    df['night_surcharge'] = np.where((df['hour_of_day']>=20) | (df['hour_of_da
y'] <= 6), 1, 0)
    # df.loc[(df['hour_of_day'] >= 20) & (df['hour_of_day'] <= 6), 'night_surchar
ge'] = 1
    # df.loc[df['hour_of_day'] >6 & df['hour_of_day'] < 20, 'night_surcharge'] =
    0
    return df

def time_slicer(df, timeframes, value, color="purple"):

# Function to count observation occurrence through different Lenses of tim
e.
    f, ax = plt.subplots(len(timeframes), figsize = [12,12])
    for i,x in enumerate(timeframes):
        df.loc[:,[x,value]].groupby([x]).mean().plot(ax=ax[i],color=color)
        ax[i].set_ylabel(value.replace("_", " ").title())
        ax[i].set_title("{} by {}".format(value.replace("_", " ").title(), x.r
eplace("_", " ").title()))
        ax[i].set_xlabel("")
    ax[len(timeframes)-1].set_xlabel("Time Frame")
    plt.tight_layout(pad=0)
```

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import time
from math import sqrt
df_train = pd.read_csv('train.csv', nrows = 2000000)
print('Old size: %d' % len(df_train))
# Drop rows with -ve fare
df_train = df_train[df_train.fare_amount>=0]
print(df_train.isnull().sum())
# Drop rows with no data
df_train = df_train.dropna(how = 'any', axis = 'rows')
print('New size: %d' % len(df_train))

loca = (-74.27, -72.99, 40.58, 41.8)
print('Old size: %d' % len(df_train))
# Drop rows with Location out of NYC
df_train = df_train[filter_on_location(df_train, loca)]
print('New size: %d' % len(df_train))

print('Old size: %d' % len(df_train))
# Drop rows with invalid number of passengers
df_train = df_train[filter_passenger_count(df_train)]
print('New size: %d' % len(df_train))

# df_train['pickup_datetime'] = pd.to_datetime(df_train['pickup_datetime'])
df_train.dtypes
```

Old size: 2000000

key	0
fare_amount	0
pickup_datetime	0
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	14
dropoff_latitude	14
passenger_count	0
dtype:	int64

New size: 1999909

Old size: 1999909

New size: 1957132

Old size: 1957132

New size: 1950180

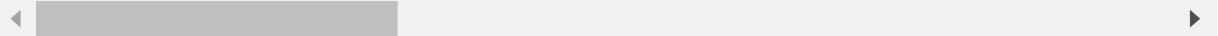
```
Out[2]: key          object
fare_amount      float64
pickup_datetime   object
pickup_longitude float64
pickup_latitude   float64
dropoff_longitude float64
dropoff_latitude   float64
passenger_count     int64
dtype: object
```

2. Pairwise Pearson correlation:

```
In [3]: # add new column to dataframe with distance in miles
df_train['distance_miles'] = distance(df_train.pickup_latitude, df_train.pickup_longitude, \
                                         df_train.dropoff_latitude, df_train.dropoff_longitude)
df_train['latdiff'] = df_train['pickup_latitude'] - df_train['dropoff_latitude']
df_train['londiff'] = df_train['pickup_longitude'] - df_train['dropoff_longitude']
df_train['euclid'] = np.sqrt((df_train['latdiff'] * df_train['latdiff']) + (df_train['londiff'] * df_train['londiff']))
# add new columns to dataframe with for time features
df_train = prepare_time_features(df_train)
df_train.head()
```

Out[3]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21	-73.844311	40.721319
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16	-74.016048	40.711303
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00	-73.982738	40.761270
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42	-73.987130	40.733143
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00	-73.968095	40.768008

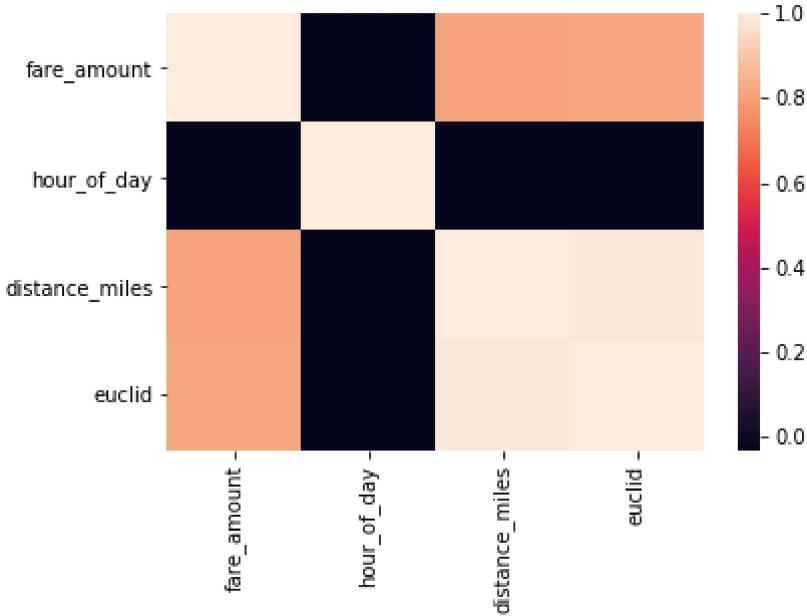


```
In [4]: needed_variables = ['fare_amount','hour_of_day','distance_miles','euclid']
df_corr = df_train[needed_variables].corr()
print(df_corr)
sns.heatmap(df_corr, xticklabels=df_corr.columns.values, yticklabels=df_corr.columns.values)

# plt.scatter(x=df_train['distance_miles'],y=df_train['fare_amount'])
# plt.xlabel("Euclidean distance of the ride")
# plt.ylabel("Fare Amount")
# plt.title("Ride Distance vs Fare Amount")
```

	fare_amount	hour_of_day	distance_miles	euclid
fare_amount	1.000000	-0.018127	0.817420	0.825804
hour_of_day	-0.018127	1.000000	-0.029139	-0.029670
distance_miles	0.817420	-0.029139	1.000000	0.993916
euclid	0.825804	-0.029670	0.993916	1.000000

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x16e98513320>



Observation on Pearson correlation:

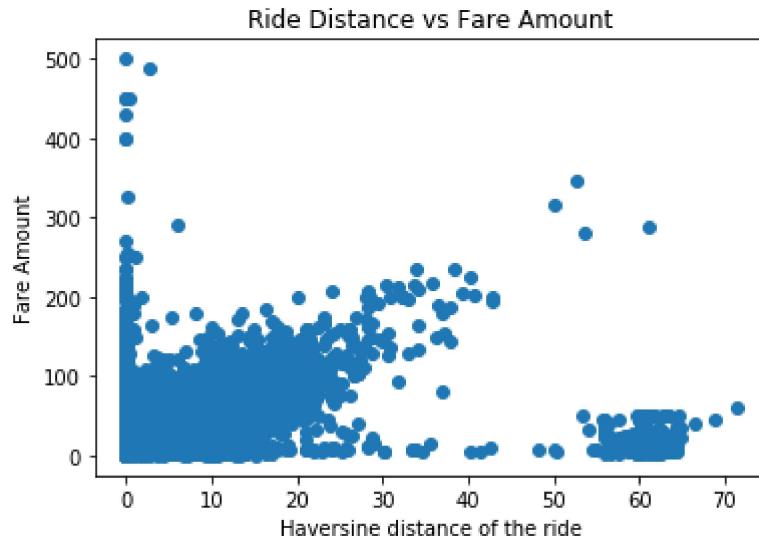
The distance in miles has the highest correlation with fare amount. Hour of a day does not give a strong correlation with fare amount or distance travelled but the time of day gives useful patterns in predicting fare amount as well as ride distance.

3. Visualization the correlations between variables:

3.1 Ride Distance Vs Fare Amount

```
In [5]: plt.scatter(x=df_train['distance_miles'],y=df_train['fare_amount'])
plt.xlabel("Haversine distance of the ride")
plt.ylabel("Fare Amount")
plt.title("Ride Distance vs Fare Amount")
```

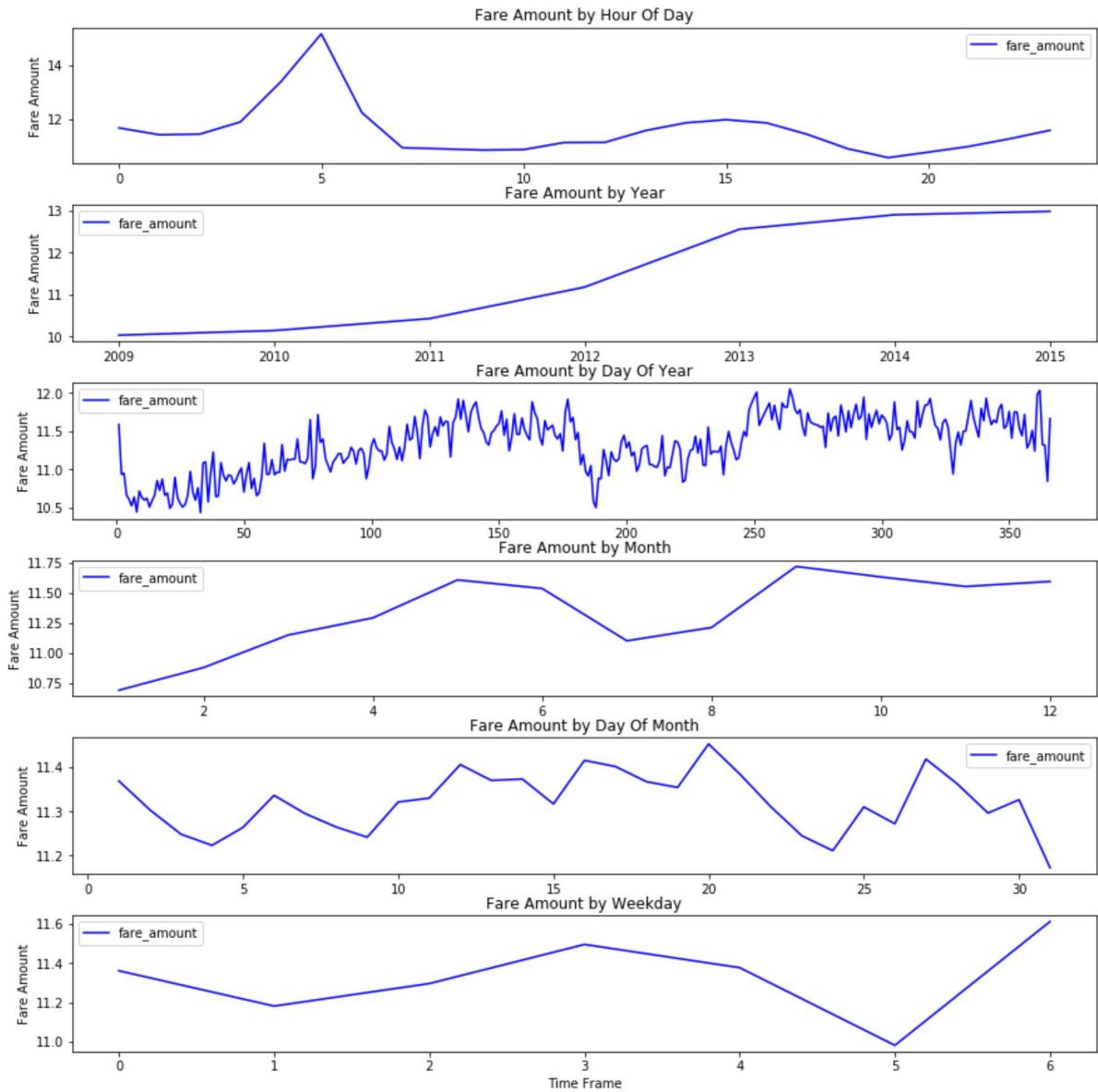
```
Out[5]: Text(0.5,1,'Ride Distance vs Fare Amount')
```



Observation: In the plot above, we can see a fairly linear realtionship between taxi fare and distance. But for trip distances >50miles, though a linear relationship exists,the fare amount is very low. The fare seems to be fixed for trip distances > 50 miles. Normally airports pickup or dropoff have fixed prices.

3.2 Time of Day Vs Fare Amount

```
In [6]: time_slicer(df=df_train, timeframes=['hour_of_day', 'year', "day_of_year", "month", "day_of_month", "weekday"], value = "fare_amount", color="blue")
```

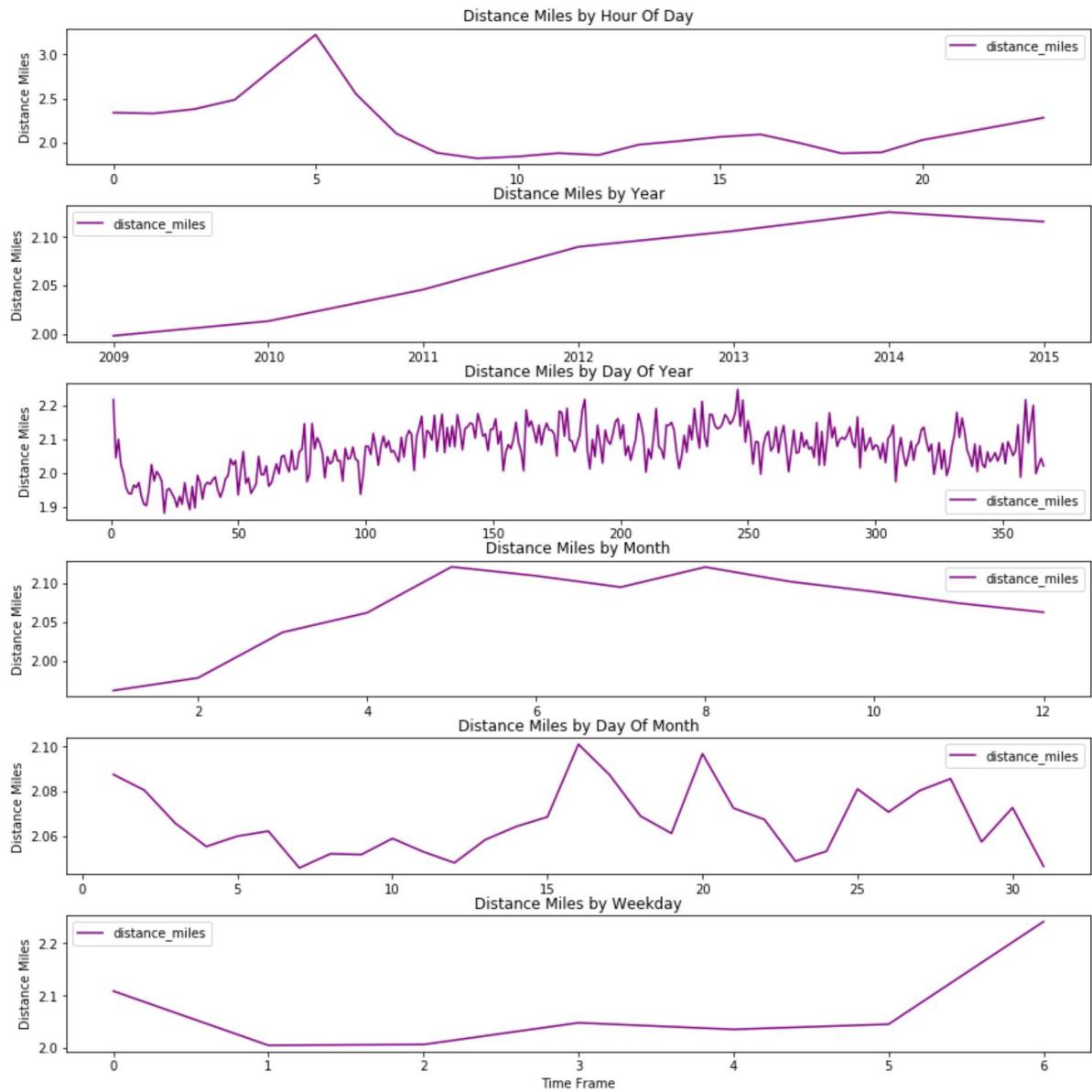


Observations on time of ride and fare amount:

- The fares, however, seem to be high between 5AM and 10AM, and 2PM to 4PM. It is possible that people who live far away prefer to leave earlier to avoid rush hour traffic.
- The avg fare amount at 5am is the higher which would be because of lower number of taxis available. Most passengers would be travelling towards the airport at this time.
- The highest fares seem to be on a Sunday and Monday, and the lowest on Wednesday and Friday. Possibly a lot of people travel longer distances on Sunday and Monday (visiting family living far away and returning back home), and hence, the high fares. And guess people just want to stay at home on a Friday after a hectic week at work, or grab a drink from close by.
- The fares throughout the month mostly seem uniform, with the maximum fare received on the 12th

3.3 Time of Day Vs Distance Travelled

```
In [7]: time_slicer(df=df_train, timeframes=['hour_of_day', 'year',"day_of_year", "month", "day_of_month", "weekday"], value = "distance_miles", color = "purple")
```



Observations on time of ride and distance travelled:

- Longer trips are taken around 5am which maybe to airports.
- Short trips are taken between 8am to 10 am, which may be to workplaces.
- The distances travelled are shorter in colder months. This may be due to reduction in number of tourists in extreme cold.
- Passengers travel longer distances on Sundays, probably taking cross town trips.

4 An exciting plot using the dataset that reveals something interesting

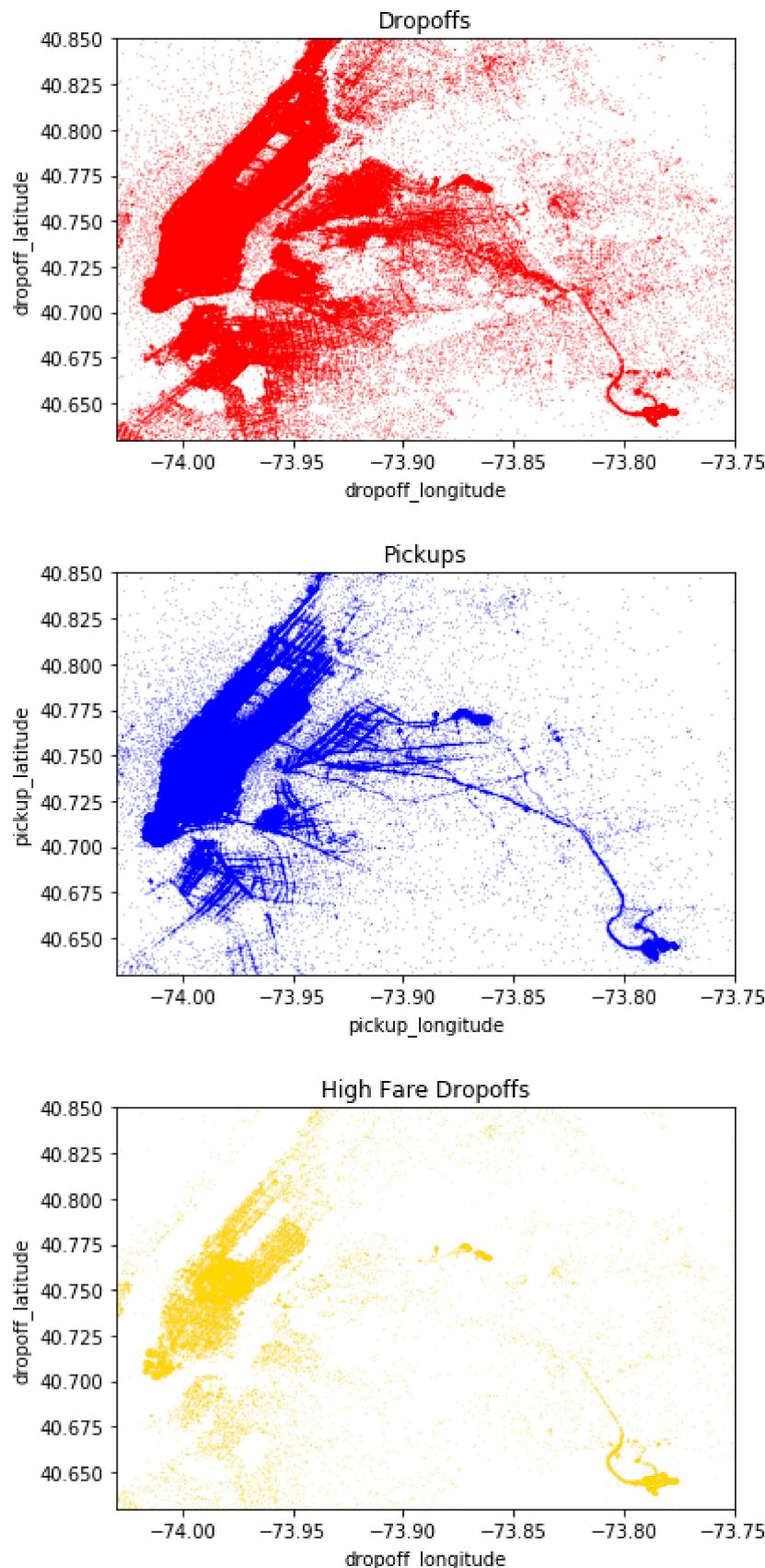
Plotting pickup and drop off points on a map on NYC. Also, I have tried plotting points from which higher than normal fares are charged. If the fare more than 3 standard deviation more than the mean fare, I am labelling it as a high fare.

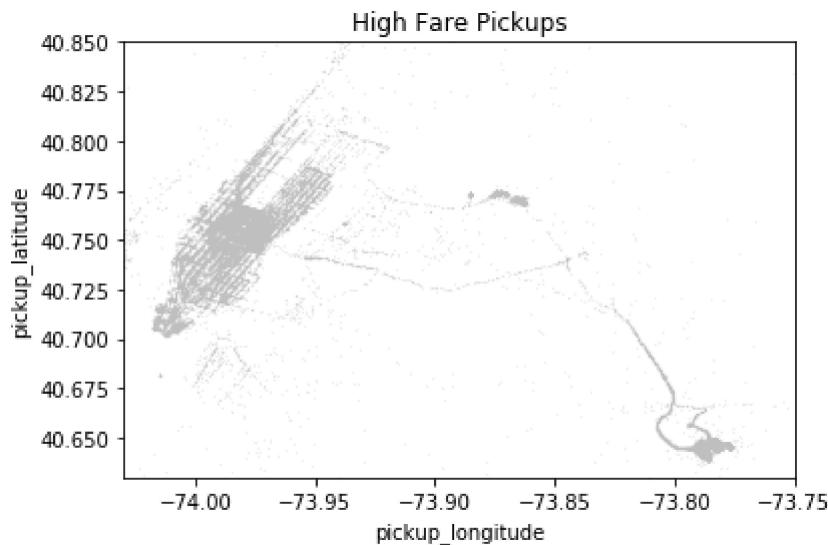
```
In [8]: lat_border = (40.63, 40.85) # Defining approximate boundaries of NYC
long_border = (-74.03, -73.75)
df_train.plot(kind='scatter', x='dropoff_longitude', y='dropoff_latitude',
              color='red', s=.02, alpha=.6)
plt.title("Dropoffs")
plt.ylim(lat_border)
plt.xlim(long_border)

df_train.plot(kind='scatter', x='pickup_longitude', y='pickup_latitude',
              color='blue', s=.02, alpha=.6)
plt.title("Pickups")
plt.ylim(lat_border)
plt.xlim(long_border)
plt.show()

high_fares = df_train[df_train['fare_amount'] > df_train.fare_amount.mean() +
3* df_train.fare_amount.std()]
high_fares.plot(kind='scatter', x='dropoff_longitude', y='dropoff_latitude',
                 color='gold', s=.02, alpha=.6)
plt.title("High Fare Dropoffs")
plt.ylim(lat_border)
plt.xlim(long_border)
plt.show()

high_fares = df_train[df_train['fare_amount'] > df_train.fare_amount.mean() +
3* df_train.fare_amount.std()]
high_fares.plot(kind='scatter', x='pickup_longitude', y='pickup_latitude',
                 color='silver', s=.02, alpha=.6)
plt.title("High Fare Pickups")
plt.ylim(lat_border)
plt.xlim(long_border)
plt.show()
```





Observations on pickup and dropoff location plots:

- Majority of the pickup and drop off points are in lower Manhattan.
- The two locations that appear to stand out in the high fare drop off and pick up plots. These location are in vicinity of John F. Kennedy International Airport and LaGuardia Airport. This information can be used to add new features with respect to distance of pickup or dropoff point from an airport.

5 Generating additional features from existing dataset

The following useful features have been added to the existing dataset based on the above plots and Pearson Coeffiecents:

- `distance_miles`: This is the Haversine distance between the pickup and dropoff locations.
- `hour_of_day`
- `month`
- `year`
- `night_surcharge`
- Distance from JFK Airport: Minimum of the dropoff or pickup location is chosen
- Distance from Newark Airport: Minimum of the dropoff or pickup location is chosen
- Distance from LGA Airport: Minimum of the dropoff or pickup location is chosen

```
In [9]: # Return minumum distance from pickup or dropoff coordinates to hotspots like airports.
#      JFK: John F. Kennedy International Airport
#      EWR: Newark Liberty International Airport
#      LGA: LaGuardia Airport
def add_airport_dist(dataset):
    jfk_coord = (40.639722, -73.778889)
    ewr_coord = (40.6925, -74.168611)
    lga_coord = (40.77725, -73.872611)

    pickup_lat = dataset['pickup_latitude']
    dropoff_lat = dataset['dropoff_latitude']
    pickup_lon = dataset['pickup_longitude']
    dropoff_lon = dataset['dropoff_longitude']

    pickup_jfk = distance(pickup_lat, pickup_lon, jfk_coord[0], jfk_coord[1])
    dropoff_jfk = distance(jfk_coord[0], jfk_coord[1], dropoff_lat, dropoff_lon)
    pickup_ewr = distance(pickup_lat, pickup_lon, ewr_coord[0], ewr_coord[1])
    dropoff_ewr = distance(ewr_coord[0], ewr_coord[1], dropoff_lat, dropoff_lon)
    pickup_lga = distance(pickup_lat, pickup_lon, lga_coord[0], lga_coord[1])
    dropoff_lga = distance(lga_coord[0], lga_coord[1], dropoff_lat, dropoff_lon)

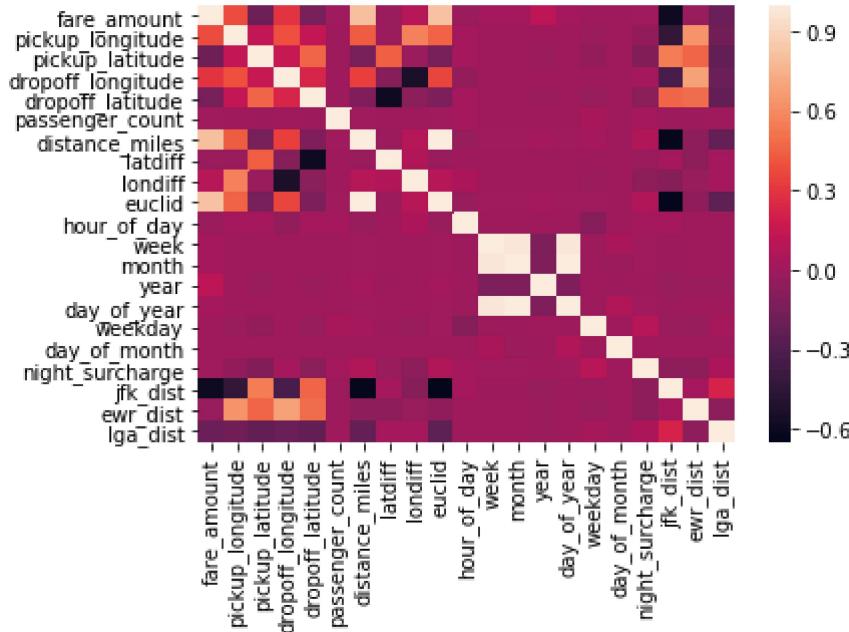
    dataset['jfk_dist'] = pd.concat([pickup_jfk, dropoff_jfk], axis=1).min(axis=1)
    dataset['ewr_dist'] = pd.concat([pickup_ewr, dropoff_ewr], axis=1).min(axis=1)
    dataset['lga_dist'] = pd.concat([pickup_lga, dropoff_lga], axis=1).min(axis=1)

    return dataset

#Run the functions to add the features to the dataset
df_train = add_airport_dist(df_train)

df_corr = df_train.corr()
# print(df_corr)
sns.heatmap(df_corr, xticklabels=df_corr.columns.values, yticklabels=df_corr.columns.values)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x16e8e36b4e0>



6 Linear regression model to predict taxi fare:

Using Linear Regression to predict taxi fare using features: year, hour_of_day, month, distance_miles, passenger_count, night_surcharge, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, jfk_dist, ewr_dist, lga_dist.

```
In [10]: from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression

idx = (df_train.passenger_count!=0)

features = ['year', 'hour_of_day', 'week', 'month','distance_miles', 'passenger_count', 'night_surcharge', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
            'dropoff_latitude', 'jfk_dist', 'ewr_dist', 'lga_dist']
# features = ['year', 'hour_of_day', 'distance_miles']
X = df_train[idx][features].values
y = df_train[idx]['fare_amount'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model_LR = LinearRegression()
model_LR.fit(X_train, y_train)
y_predicted = model_LR.predict(X_test)

# The coefficients of the Linear equation aX1 + bX2 + cX3 + ... = Y
print('Coefficients of Linear Regression : ', [(x,y) for x,y in zip(features,model_LR.coef_)])
```

The mean squared error of the Linear regression model's predictions

```
print("Mean squared error: %.5f" % mean_squared_error(y_test, y_predicted))

# Variance of the prediction
print('Variance : %.5f' % r2_score(y_test, y_predicted))
```

df_test = pd.read_csv('test.csv')

add new column to dataframe with distance in miles

```
df_test['distance_miles'] = distance(df_test.pickup_latitude, df_test.pickup_longitude, \
                                         df_test.dropoff_latitude, df_test.dropoff_longitude)
```

add new columns to dataframe with for time features

```
df_test = prepare_time_features(df_test)
df_test = add_airport_dist(df_test)
# df_test = add_ride_durations(df_test)
# df_test['duration'].to_csv('durations_test.csv', index=False)
```

define test dataset

```
XTEST = df_test[features].values
y_pred_final = model_LR.predict(XTEST)

submission = pd.DataFrame(
    {'key': df_test.key, 'fare_amount': y_pred_final},
    columns = ['key', 'fare_amount'])
submission.to_csv('submission_lr.csv', index = False)
```

```
Coefficients of Linear Regression : [('year', 0.531868318052891), ('hour_of_day', 0.01243677939514244), ('week', 0.012538068608420504), ('month', 0.020952189847604684), ('distance_miles', 3.445314304570997), ('passenger_count', 0.036461174588833266), ('night_surcharge', -1.0091035908891743), ('pickup_longitude', -16.137634598582597), ('pickup_latitude', -33.27002997013331), ('dropoff_longitude', -18.890396778007027), ('dropoff_latitude', -23.70873371911323), ('jfk_dist', -0.0810333011909446), ('ewr_dist', 1.0407753698546698), ('lga_dist', -0.3410691324024978)]
Mean squared error: 28.09567
Variance : 0.69449
```

Observations on Linear Regression:

Most important features from the coefficients of Linear Regression:

1. distance_miles: As suggested by Pearson Coefficient, the haversine distance is directly proportional to ride fare.
2. year: As seen before the fares are increasing year after year.
3. month: Peak tourist months and Christmas time seem to increase the taxi fares
4. passenger_count: Increase in number of passengers increases the taxi fare marginally.
5. Distance from airports: Lesser the distance from an airport the more is the taxi fare.

7 External Datasets:

1. Duration of Ride: Our current dataset is missing the dropoff timestamp, so the duration of ride is not known. The duration of ride should be directly proportional to the taxi fare. I was able to get RIDE DURATION by passing start and end coordinates to distance_matrix API provided by Google Maps. The time of travel is seen to be linearly increasing with haversine distance. Source:
<https://developers.google.com/maps/documentation/distance-matrix/intro#DistanceMatrixResponses>
<https://www.geeksforgeeks.org/python-calculate-distance-duration-two-places-using-google-distance-matrix-api/>
2. Density of taxis available near pickup point: The more the number of taxis available at a pickup point, lesser should be the fare. This information would be available for taxi providers like Uber and Lyft.
3. Weather: Rain and Snow can potentially make the taxi fares to go up. This information can potentially be obtained for the city of New York. We can add features like Rainy(0,1) and Snow (0,1).

The external could be obtained for a small training set(5k). The performance does not increase substantially as the train data is less. Also, duration of ride for distances might vary depending on a lot of external factors including driver's expertise. Hence even for fixed distances this might not be constant across all rides. This may be why the external data did not help much. But a near linear relation between duration and fare is undisputable.

```
In [11]: # AN EXPERIMENT USING EXTERNAL DATASET to make predictions of fare.
# Using Google Maps to get the duration of a journey by a taxi in seconds between given source and destination coordinates.
# The REST API of Google takes time, hence training on 10K rows.
# importing required libraries

import googlemaps
import pandas as pd
# METHOD TO GET EXTRENAL DATA: DURATION OF TRIPS FROM GOOGLE MAPS
def add_ride_durations(df):
    gmaps = googlemaps.Client(key='AIzaSyDIAmvyppS8WYvY9vjZz2-DWAku-KqRw0K8')
    origins = []
    destinations = []
    ride_durations = []
    count = 0;
    for index, row in df.iterrows():
        origins = []
        destinations = []
        origins.append(str(row['pickup_latitude']) + ' ' + str(row['pickup_longitude']))
        destinations.append(str(row['dropoff_latitude']) + ' ' + str(row['dropoff_longitude']))
        matrix = gmaps.distance_matrix(origins, destinations)
        if matrix is not None:
            if matrix.get('rows')[0].get('elements')[0].get('duration') is not None:
                ride_durations.append(matrix.get('rows')[0].get('elements')[0].get('duration').get('value'))
            else:
                ride_durations.append("-1")
        if matrix is None:
            ride_durations.append("-1")
        count+=1
    #     if count%1000==0:
    #         print (count)
    #     if count>=len(df):
    #         break
    #     print(ride_durations)
    df['duration'] = ride_durations
    return df
# Ran once and stored results in a CSV.
df_external = df_train[:5000]
df_external= df_external.reset_index()
# df_external = add_ride_durations(df_external)
# df_external['duration'].to_csv('durations.csv', index=False)
df_new = pd.read_csv('durations.csv')
df_external['duration'] = df_new

plt.scatter(x=df_external['duration'],y=df_external['fare_amount'])
plt.xlabel("Duration of the ride in secs from Google Maps API")
plt.ylabel("Fare Amount")
plt.title("Duration of the ride in secs vs Fare Amount")
plt.show()

# Train models on external dataset
```

```

features_ext = ['year', 'hour_of_day', 'week', 'month','distance_miles', 'passenger_count', 'night_surcharge', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
                'dropoff_latitude', 'jfk_dist', 'ewr_dist', 'lga_dist', 'duration']
# features = ['year', 'hour_of_day', 'distance_miles']
X_ext = df_external[features_ext].values
y_ext = df_external['fare_amount'].values
X_train_ext, X_test_ext, y_train_ext, y_test_ext = train_test_split(X_ext, y_ext, test_size=0.2)
model_LR_ext = LinearRegression()

model_LR_ext.fit(X_train_ext, y_train_ext)
y_predicted_ext = model_LR_ext.predict(X_test_ext)

# The coefficients of the Linear equation aX1 + bX2 + cX3 + ... = Y
print('Coefficients of Linear Regression : ', [(x,y) for x,y in zip(features_ext,model_LR_ext.coef_)]) 

# The mean squared error of the Linear regression model's predictions
print("Mean squared error: %.5f" % mean_squared_error(y_test_ext, y_predicted_ext))

# Variance of the prediction
print('Variance : %.5f' % r2_score(y_test_ext, y_predicted_ext))

from sklearn.ensemble import RandomForestRegressor
rf_ext = RandomForestRegressor(n_estimators=25, max_features=None, max_depth=25, min_samples_split=3,
                                min_samples_leaf=2, random_state=0)
rf_ext.fit(X_train_ext, y_train_ext)
y_predict_ext = rf_ext.predict(X_test_ext)
print("Mean squared error RF_ext: %.5f" % mean_squared_error(y_test_ext, y_predict_ext))
print("Root Mean squared error RF_ext: %.5f" % sqrt(mean_squared_error(y_test_ext, y_predict_ext)))
# Variance of the prediction
print('Variance RF : %.5f' % r2_score(y_test_ext, y_predict_ext))

# -----
# RUNNING RF ON THE TEST DATA.

df_test_ext = pd.read_csv('test.csv')

# add new column to dataframe with distance in miles
df_test_ext['distance_miles'] = distance(df_test_ext.pickup_latitude, df_test_ext.pickup_longitude, \
                                            df_test_ext.dropoff_latitude, df_test_ext.dropoff_longitude)

# add new columns to dataframe with for time features
df_test_ext = prepare_time_features(df_test_ext)
df_test_ext = add_airport_dist(df_test_ext)

# df_test_ext = add_ride_durations(df_test_ext)
# df_test_ext['duration'].to_csv('durations_test.csv', index=False)

```

```

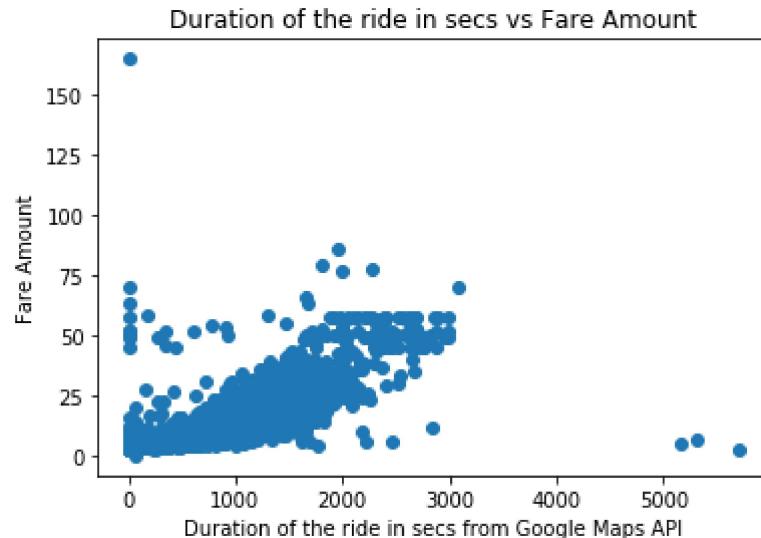
df_test_ext= df_test_ext.reset_index()
df_new = pd.read_csv('durations_test.csv')

# df_test_ext.insert(2, "duration", 0)
df_test_ext['duration'] = df_new
# REPLACE NAN WITH MEAN
df_test_ext['duration'] = df_test_ext['duration'].fillna(df_test_ext['duration'].mean())
# define test dataset
XTEST_ext = df_test_ext[features_ext].values

y_pred_final_rf_ext = rf_ext.predict(XTEST_ext)

submission = pd.DataFrame(
    {'key': df_test_ext.key, 'fare_amount': y_pred_final_rf_ext},
    columns = ['key', 'fare_amount'])
submission.to_csv('submission_rf_ext.csv', index = False)

```



Coefficients of Linear Regression : [('year', 0.4772139240798721), ('hour_of_day', 0.0036709988367102033), ('week', -0.00025119760114886277), ('month', 0.06993762479638393), ('distance_miles', 2.4127443530343373), ('passenger_count', -0.03162111639896925), ('night_surcharge', -1.0064017115223145), ('pickup_longitude', -24.58550961290238), ('pickup_latitude', -38.08502343092172), ('dropoff_longitude', -6.637393425301386), ('dropoff_latitude', -14.663113755831608), ('jfk_dist', -0.07343397057094207), ('ewr_dist', 1.4602279416221835), ('lga_dist', 0.015878420648935958), ('duration', 0.00519441064046215)]
Mean squared error: 59.63087
Variance : 0.22576
Mean squared error RF_ext: 31.51006
Root Mean squared error RF_ext: 5.61338
Variance RF : 0.59088

Kaggle Score for RF with duration of ride: 3.99

8 Better prediction models:

The following techniques were tried on our dataset:

- Random Forest Regressor
- Ridge Regression
- SVM

```
In [12]: # RIDGE
```

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=1e-4, normalize=True)
model_ridge.fit(X_train,y_train)
y_predicted = model_ridge.predict(X_test)
print("Mean squared error Ridge: %.12f"
      % mean_squared_error(y_test, y_predicted))
df_train['distance_miles'].describe()

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=25, max_features=None, max_depth=25, m
in_samples_split=3,
                           min_samples_leaf=2, random_state=0)
rf.fit(X_train, y_train)
y_predict = rf.predict(X_test)
print("Mean squared error RF: %.5f" % mean_squared_error(y_test, y_predict))
print("Root Mean squared error RF: %.5f" % sqrt(mean_squared_error(y_test, y_p
redict)))
# Variance of the prediction
print('Variance RF : %.5f' % r2_score(y_test, y_predict))

y_pred_final_rf = rf.predict(XTEST)

submission = pd.DataFrame(
    {'key': df_test.key, 'fare_amount': y_pred_final_rf},
    columns = ['key', 'fare_amount'])
submission.to_csv('submission_rf.csv', index = False)
```

Mean squared error Ridge: 28.096059381583

Mean squared error RF: 13.67290

Root Mean squared error RF: 3.69769

Variance RF : 0.85132

```
In [14]: # Creating a Support Vector Machine model for prediction
from sklearn.svm import SVR

model_svm = SVR(cache_size=7000)

# Training the SVM Model
model_svm.fit(X_train[:10000], y_train[:10000])

# Using the model to predict the test dataset
y_predict = model_svm.predict(X_test)

# The mean squared error of the KNN model's predictions
print("Mean squared error SVM: %.5f" % mean_squared_error(y_test, y_predict))

# Variance of the prediction
print('Variance SVM: %.5f' % r2_score(y_test, y_predict))

y_pred_final_svm = model_svm.predict(XTEST)

submission = pd.DataFrame(
    {'key': df_test.key, 'fare_amount': y_pred_final_svm},
    columns = ['key', 'fare_amount'])
submission.to_csv('submission_svm.csv', index = False)
```

Mean squared error SVM: 60.12889

Variance SVM: 0.34617

Clearly Random Forest performs the best. Random Forest is able to discover more complex dependencies at the cost of more time for fitting. Since linear regression is performing not too well it shows the dependency is something different from linear, there is no possibility for an algorithm to come up with a linear equation for predicting the ride fare. Parameter Tuning in Random Forest further improved the accuracy. n_estimators=25: Increasing the number of trees in the forest(n_estimators) to 25 from default of 25 improved the accuracy.

Credits:

- Fare VS Time and Distance VS time of day plots ideas inspired from:
[\(https://www.kaggle.com/nicapotato/taxi-rides-time-analysis-and-oof-lgbm\)](https://www.kaggle.com/nicapotato/taxi-rides-time-analysis-and-oof-lgbm)