Greetings! Today we are going to discuss the NYC Taxi Trip dataset. Further we will work on steps such as Data Exploration, Data Preprocessing, Data Visualization, Data Modelling and finally the conclusion so that we can finally predict Taxi Trip Duration.

Import necessary libraries

In [ ]:
```
import pandas as pd
import numpy as np
```

Load the dataset

In [ ]:
```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
```

Exploring the dataset

In [ ]:
```
print(train_df.head())
print(train_df.describe())
```

```
         id  vendor_id      pickup_datetime     dropoff_datetime  \
0  id2875421          2  2016-03-14 17:24:55  2016-03-14 17:32:30
1  id2377394          1  2016-06-12 00:43:35  2016-06-12 00:54:38
2  id3858529          2  2016-01-19 11:35:24  2016-01-19 12:10:48
3  id3504673          2  2016-04-06 19:32:31  2016-04-06 19:39:40
4  id2181028          2  2016-03-26 13:30:55  2016-03-26 13:38:10

   passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
0                1        -73.982155        40.767937         -73.964630
1                1        -73.980415        40.738564         -73.999481
2                1        -73.979027        40.763939         -74.005333
3                1        -74.010040        40.719971         -74.012268
4                1        -73.973053        40.793209         -73.972923

   dropoff_latitude store_and_fwd_flag  trip_duration
0         40.765602                  N            455
1         40.731152                  N            663
2         40.710087                  N           2124
3         40.706718                  N            429
4         40.782520                  N            435
          vendor_id  passenger_count  pickup_longitude  pickup_latitude  \
count  1.458644e+06     1.458644e+06      1.458644e+06     1.458644e+06
mean   1.534950e+00     1.664530e+00     -7.397349e+01     4.075092e+01
std    4.987772e-01     1.314242e+00      7.090186e-02     3.288119e-02
min    1.000000e+00     0.000000e+00     -1.219333e+02     3.435970e+01
25%    1.000000e+00     1.000000e+00     -7.399187e+01     4.073735e+01
50%    2.000000e+00     1.000000e+00     -7.398174e+01     4.075410e+01
75%    2.000000e+00     2.000000e+00     -7.396733e+01     4.076836e+01
max    2.000000e+00     9.000000e+00     -6.133553e+01     5.188108e+01

       dropoff_longitude  dropoff_latitude  trip_duration
count       1.458644e+06      1.458644e+06   1.458644e+06
mean       -7.397342e+01      4.075180e+01   9.594923e+02
std         7.064327e-02      3.589056e-02   5.237432e+03
min        -1.219333e+02      3.218114e+01   1.000000e+00
25%        -7.399133e+01      4.073588e+01   3.970000e+02
50%        -7.397975e+01      4.075452e+01   6.620000e+02
75%        -7.396301e+01      4.076981e+01   1.075000e+03
max        -6.133553e+01      4.392103e+01   3.526282e+06
```

Preprocessing (if needed): Drop irrelevant columns if any etc

In [ ]:
```
train_df.vendor_id.value_counts()
```

Out[ ]:
```
2    780302
1    678342
Name: vendor_id, dtype: int64
```

In [ ]:
```
train_df= train_df.sample(frac = 0.15,random_state=1)
test_df= test_df.sample(frac = 0.15,random_state=1)
```

Organising the date and time columns

In [ ]:
```
train_df['pickup_datetime'] = pd.to_datetime(train_df['pickup_datetime'])
train_df['hour'] = train_df['pickup_datetime'].dt.hour
train_df['minute'] = train_df['pickup_datetime'].dt.minute
train_df['minute_oftheday'] = train_df['hour']*60 + train_df['minute']
train_df["day_week"] =train_df["pickup_datetime"].dt.dayofweek
train_df["month"] = train_df["pickup_datetime"].dt.month
```

```
test_df["pickup_datetime"] = pd.to_datetime(test_df["pickup_datetime"])
test_df["hour"] = test_df["pickup_datetime"].dt.hour
test_df["minute"] = test_df["pickup_datetime"].dt.minute
test_df["minute_oftheday"] = test_df["hour"] * 60 + test_df["minute"]
test_df["day_week"] =test_df["pickup_datetime"].dt.dayofweek
test_df["month"] = test_df["pickup_datetime"].dt.month
```

Dropping Original DateTime Columns (we have new ones now)

In [ ]:
```
train_df.drop(["pickup_datetime", "dropoff_datetime"], axis=1, inplace=True)
test_df.drop(["pickup_datetime"], axis=1, inplace=True)
```

In [ ]:
```
train_df.head(5)
```

Out[ ]:

	id	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude
1457636	id0880738	2	1	-73.981728	40.749500	-73.945915
615369	id2002545	2	1	-73.979088	40.771606	-73.946518
491096	id0289724	2	1	-73.989700	40.738651	-73.997772
82632	id3767649	2	1	-73.988441	40.723267	-73.995880
71403	id2530846	2	2	-73.985733	40.752598	-73.969231

Drop the ID columns because it's difficult for them to be converted into Float values

In [ ]:
```
train_df = train_df.drop("id", axis=1)
test_df = test_df.drop("id", axis=1)
```

In [ ]:
```
pip install geopy
```

```
Requirement already satisfied: geopy in c:\users\aindri\anaconda3\envs\ml_one\lib\sit
e-packages (2.3.0)
Requirement already satisfied: geographiclib<3,>=1.52 in c:\users\aindri\anaconda3\en
vs\ml_one\lib\site-packages (from geopy) (2.0)
Note: you may need to restart the kernel to use updated packages.
```

Using geopy to make the distance value easier to read and predict for later

In [ ]:
```
from geopy import distance

def get_distance(row):
    pick = (row.pickup_latitude, row.pickup_longitude)
    drop = (row.dropoff_latitude, row.dropoff_longitude)
    dist = distance.geodesic(pick, drop).km
    return dist

train_df["distance"] = train_df.apply(get_distance, axis=1)
test_df["distance"] = test_df.apply(get_distance, axis=1)
```

In [ ]:
```
print(train_df.store_and_fwd_flag.value_counts())
train_df["store_and_fwd_flag"].replace({'N':0, 'Y':1}, inplace=True)
```

```
test_df["store_and_fwd_flag"].replace({'N':0,'Y':1}, inplace=True)
print(train_df.store_and_fwd_flag.value_counts())
```

```
N    217639
Y      1158
Name: store_and_fwd_flag, dtype: int64
0    217639
1      1158
Name: store_and_fwd_flag, dtype: int64
```

In [ ]:
```
train_df.drop('minute_oftheday', axis=1, inplace=True)
test_df.drop('minute_oftheday', axis=1, inplace=True)
```

In [ ]:
```
label = 'trip_duration'
```

In [ ]:
```
features = list(train_df.columns)
features.remove(label)
```

In [ ]:
```
df = train_df
```

Predictive Models

importing necessary libraries

In [ ]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score,mean_absolute_error
```

In [ ]:
```
from sklearn.linear_model import LinearRegression,Ridge,Lasso,ElasticNet
from sklearn.neighbors import KNeighborsRegressor
```

In [ ]:
```
import math
import time
def model_report(model,training_x,training_y,testing_x,testing_y,name) :
    start = time.time()


    model.fit(training_x,training_y)
    predictions  = model.predict(testing_x)
    mae  = mean_absolute_error(y_test,predictions)
    r2=  r2_score(y_test,predictions)
    duration=(time.time() - start)
    df = pd.DataFrame({"Model"           : [name],
                       "MAE"  : [mae],
                       "R2 Score"    : [r2],
                        "Duration"    : [duration],
                      })
    return df,model
```

In [ ]:
```
X = train_df.drop('trip_duration', axis = 1)
Y = train_df.trip_duration
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.15)
```

LINEAR REGRESSION

In [ ]:
```
model = LinearRegression()
model0, model_lr = model_report(model, X_train, y_train, X_test,y_test,'Linear Regress
model_lr
```

Out[ ]:
```
LinearRegression
LinearRegression()
```

LASSO

In [ ]:
```
model = Lasso(alpha=0.001)
model1, model_lasso=model_report(model, X_train, y_train, X_test, y_test, 'Lasso')
model_lasso
```

Out[ ]:
```
Lasso
Lasso(alpha=0.001)
```

ELASTIC NET

In [ ]:
```
model = ElasticNet(alpha=0.001)
model_1, model_elasticnet = model_report(model, X_train, y_train, X_test, y_test, 'Ela
model_elasticnet
```

Out[ ]:
```
ElasticNet
ElasticNet(alpha=0.001)
```

KNEIGHBORS REGRESSOR

In [ ]:
```
model_knn = KNeighborsRegressor(n_neighbors=3)
model2, model_knn = model_report(model_knn, X_train, y_train, X_test, y_test, 'KNeighb
```

SUMMARY

In [ ]:
```
model_sum = pd.concat([model0, model1, model2], axis = 0).reset_index()

model_sum = model_sum.drop(columns = "index", axis=1)

model_sum
```

Out[ ]:

	Model	MAE	R2 Score	Duration
0	Linear Regression	499.451402	0.002226	0.096066
1	Lasso	499.451329	0.002226	0.063243
2	KNeighbors Regressor	567.353545	-0.023442	2.858535

Hence in this context, KNeighbors Regressor is the best performer