In [1]:

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the data

In [2]:

```
# Importing Data
data = pd.read_csv('nyc_taxi_trip_duration.csv')
data.head()
```

Out[2]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.771164	N	400
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.694931	N	1100
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.774918	N	1635
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.780628	N	1141
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.740631	N	848

In [3]:

```
# creating an instance(date) of DatetimeIndex class using "pickup_datetime"
date_pick = pd.DatetimeIndex(data['pickup_datetime'])
# creating an instance(date) of DatetimeIndex class using "dropoff_datetime"
date_drop = pd.DatetimeIndex(data['dropoff_datetime'])

# extracting new columns from "pick datetime"

# last day of year when pickup was done
data['doy_pick'] = date_pick.dayofyear

# week of year when pickup was done
data['woy_pick'] = date_pick.weekofyear

# month of year when pickup was done
data['moy_pick'] = date_pick.month

# day of week when pickup was done
data['dow_pick'] = date_pick.dayofweek

# hour of day when pickup was done
data['hod_pick'] = date_pick.hour


# extracting new columns from "dropoff datetime"

# last day of year dropoff was done
data['doy_drop'] = date_drop.dayofyear

# week of year when dropoff was done
data['woy_drop'] = date_drop.weekofyear

# month of year when dropoff was done
data['moy_drop'] = date_drop.month

# day of week when dropoff was done
data['dow_drop'] = date_drop.dayofweek

# hour of day when dropoff was done
data['hod_drop'] = date_drop.hour
```

```
C:\Users\vempa\AppData\Local\Temp/ipykernel_19632/1098005334.py:12: FutureWarning: weekofyear and week have been deprecated, please use DatetimeIndex.isocalendar().wee
k instead, which returns a Series.  To exactly reproduce the behavior of week and weekofyear and return an Index, you may call pd.Int64Index(idx.isocalendar().week)
  data['woy_pick'] = date_pick.weekofyear
C:\Users\vempa\AppData\Local\Temp/ipykernel_19632/1098005334.py:30: FutureWarning: weekofyear and week have been deprecated, please use DatetimeIndex.isocalendar().wee
k instead, which returns a Series.  To exactly reproduce the behavior of week and weekofyear and return an Index, you may call pd.Int64Index(idx.isocalendar().week)
  data['woy_drop'] = date_drop.weekofyear
```

In [4]:

```
data.dtypes
```

Out[4]:

```
id                     object
vendor_id               int64
pickup_datetime        object
dropoff_datetime       object
passenger_count         int64
pickup_longitude      float64
pickup_latitude       float64
dropoff_longitude     float64
dropoff_latitude      float64
store_and_fwd_flag     object
trip_duration           int64
doy_pick                int64
woy_pick                int64
moy_pick                int64
dow_pick                int64
hod_pick                int64
doy_drop                int64
woy_drop                int64
moy_drop                int64
dow_drop                int64
hod_drop                int64
dtype: object
```

In [5]:

```
data = pd.get_dummies(data.drop('id',axis=1), columns = ['store_and_fwd_flag'])
```

In [6]:
```
data.tail()
```

Out[6]:

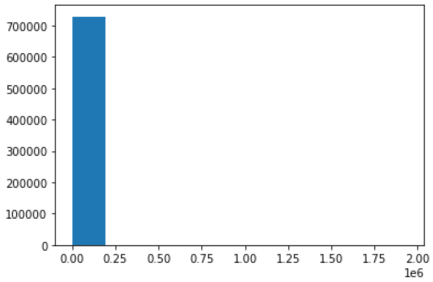
	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration	doy_pick	...	moy_pick	dow_pick	hod_pick	doy_
729317	2	2016-05-21 13:29:38	2016-05-21 13:34:34	2	-73.965919	40.789780	-73.952637	40.789181	296	142	...	5	5	13	
729318	1	2016-02-22 00:43:11	2016-02-22 00:48:26	1	-73.996666	40.737434	-74.001320	40.731911	315	53	...	2	0	0	
729319	1	2016-04-15 18:56:48	2016-04-15 19:08:01	1	-73.997849	40.761696	-74.001488	40.741207	673	106	...	4	4	18	
729320	1	2016-06-19 09:50:47	2016-06-19 09:58:14	1	-74.006706	40.708244	-74.013550	40.713814	447	171	...	6	6	9	
729321	2	2016-01-01 17:24:16	2016-01-01 17:44:40	4	-74.003342	40.743839	-73.945847	40.712841	1224	1	...	1	4	17	

5 rows × 21 columns

In [8]:
```
data_cleaned = data.drop(['pickup_datetime','dropoff_datetime'], axis=1)
```

In [9]:
```
plt.hist(data["trip_duration"])
```

Out[9]:
```
(array([7.29321e+05, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 1.00000e+00]),
 array([1.0000000e+00, 1.9397450e+05, 3.8794800e+05, 5.8192150e+05,
        7.7589500e+05, 9.6986850e+05, 1.1638420e+06, 1.3578155e+06,
        1.5517890e+06, 1.7457625e+06, 1.9397360e+06]),
 <BarContainer object of 10 artists>)
```



In [10]:
```
def UVA_outlier(data, var):
#     import pdb
#     pdb.set_trace()
    # calculating descriptives of variable
    quant25 = data[var].quantile(0.25)
    quant75 = data[var].quantile(0.75)
    IQR = quant75 - quant25
    med = data[var].median()
    whis_low = quant25-(1.5*IQR)
    whis_high = quant75+(1.5*IQR)

    ls = data.index[(data[var] < whis_low) | (data[var] > whis_high)]

    return ls
```

In [11]:
```
def remove(df,ls):
    ls = sorted(set(ls))
    df = df.drop(ls)
    return df
```

In [12]:
```
# import pdb
index_list1 = []

# for j in data.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','store_and_fwd_flag'], axis=1).columns:
for j in ['trip_duration','pickup_longitude','dropoff_longitude','pickup_latitude','dropoff_latitude']:
# for j in data.columns:
#     pdb.set_trace()
    for i in [j]:
        index_list1.extend(UVA_outlier(data,i))
        data_cleaned = remove(data,index_list1)
        index_list1.clear()
```

In [13]:
```
data = data_cleaned
```

In [14]:
```
#seperating independent and dependent variables
x = data.drop(['trip_duration','pickup_datetime','dropoff_datetime'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[14]:
```
((693076, 18), (693076,))
```

In [15]:
```
# Importing the train test split function
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 56)
```

In [16]:
```
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as mse
```

In [17]:

```
from sklearn.tree import DecisionTreeRegressor

reg = DecisionTreeRegressor()
reg.fit(train_x, train_y)

# Predicting over the Train Set and calculating error
train_predict = reg.predict(train_x)
k = mae(train_predict, train_y)
print('Training Mean Absolute Error', k )
R_squared = r2_score(train_predict,train_y)
print('Training R2 score', R_squared )
k2 = mse(train_predict, train_y)
print('Training Mean squared Error', k2 )
k3 = np.sqrt(mse(train_predict, train_y))
print('Training Root Mean squared Error    ', k3 )
```

```
Training Mean Absolute Error 0.0026259746405877566
Training R2 score 0.9999999660164457
Training Mean squared Error 0.5649202492463549
Training Root Mean squared Error     0.7516117676342986
```

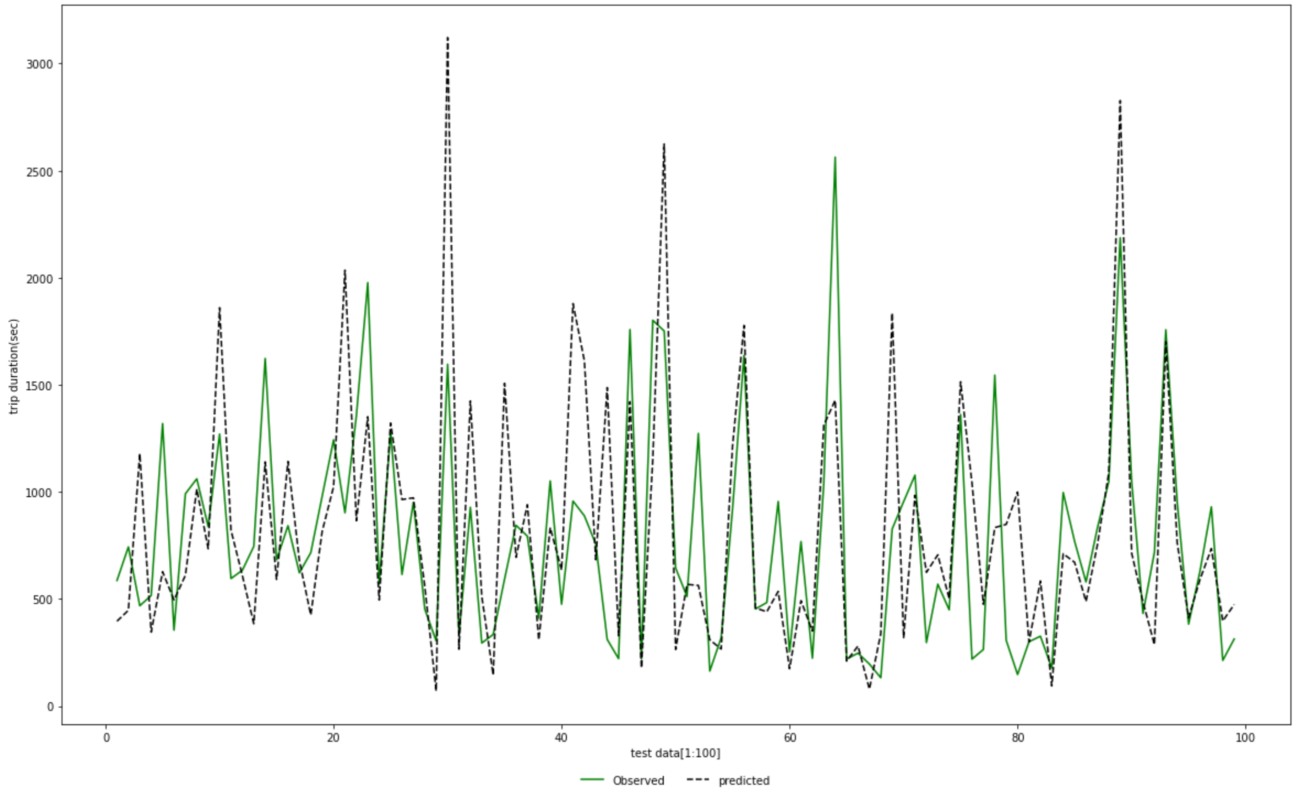
In [18]:

```
# Predicting over the Test Set and calculating error
test_predict = reg.predict(test_x)
k = mae(test_predict, test_y)
print('Test Mean Absolute Error    ', k )
R_squared = r2_score(test_predict,test_y)
print('R2 score on test set', R_squared )
k2 = mse(test_predict, test_y)
print('Test Mean squared Error    ', k2 )
k3 = np.sqrt(mse(test_predict, test_y))
print('Test Root Mean squared Error    ', k3 )
```

```
Test Mean Absolute Error     428.37537008928314
R2 score on test set -0.605175119904328
Test Mean squared Error     13555575.596191471
Test Root Mean squared Error     3681.789727318967
```

In [19]:

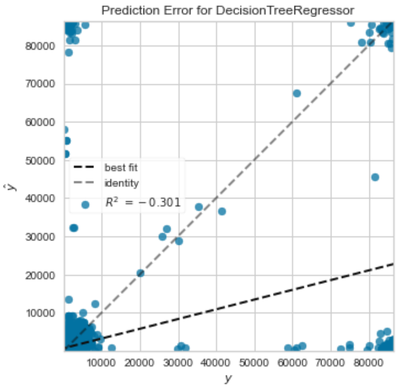
```
plt.rcParams['figure.figsize'] = (20,12)
x_ax = range(len(test_x))
plt.plot(x_ax[1:100],test_y[1:100],label= 'Observed',color = 'g', linestyle = '-')
plt.plot(x_ax[1:100],test_predict[1:100],label= 'predicted',color = 'k', linestyle = '--')
plt.xlabel('test data[1:100]')
plt.ylabel('trip duration(sec)')
plt.legend(bbox_to_anchor = (0.5,-0.1), loc = 'lower center', ncol = 2, frameon = False)
plt.show()
```



In [20]:

```
from yellowbrick.regressor import PredictionError
visualizer = PredictionError(reg)
visualizer.fit(train_x,train_y)
visualizer.score(test_x,test_y)
visualizer.poof()
```

```
C:\Users\vempa\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeRegressor was fitted with feature nam
es
  warnings.warn(
```



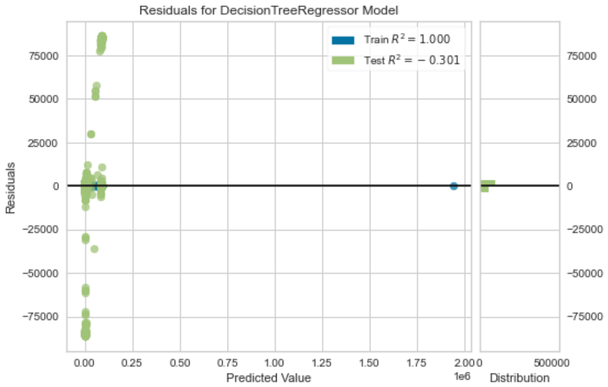
Out[20]:

```
<AxesSubplot:title={'center':'Prediction Error for DecisionTreeRegressor'}, xlabel='$y$', ylabel='$\\hat{y}$'>
```

In [21]:

```
from yellowbrick.regressor import ResidualsPlot
visualizer = ResidualsPlot(reg)
visualizer.fit(train_x,train_y)
visualizer.score(test_x,test_y)
visualizer.poof()
```

```
C:\Users\vempa\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeRegressor was fitted with feature nam
es
  warnings.warn(
```



Out[21]:

```
<AxesSubplot:title={'center':'Residuals for DecisionTreeRegressor Model'}, xlabel='Predicted Value', ylabel='Residuals'>
```

In [23]:

```
 #testing model evaluation
from sklearn.model_selection import cross_val_score
```

In [25]:

```
score_train = cross_val_score(reg, x, y , cv =10)
score_train
```

Out[25]:

```
array([-17115358.1925896 , -16710419.92389527, -14365576.85880418,
       -19289788.14772705, -17073197.69539351, -86884167.42002366,
       -14465013.76985822, -11682629.55852251, -17164371.54217007,
       -17846392.53518661])
```

In [26]:

```
score_train = np.mean(score_train)
```

In [28]:

```
score_train = np.absolute(score_train)
score_train
```

Out[28]:

```
23259691.56441707
```

In [29]:

```
score_test = cross_val_score(reg, test_x,test_y, cv =10)
score_test

score_test = np.mean(score_test)

score_test = np.absolute(score_test)
score_test
```

Out[29]:

```
16891851.67895218
```

Importance of Variables

In [53]:

```
df = pd.DataFrame({'Feature Names': x.columns, 'Importances': reg.feature_importances_})
df_sort = df.sort_values(by='Importances',ascending = False)
df_sort
```

Out[53]:

	Feature Names	Importances
11	doy_drop	0.455497
9	dow_pick	0.093580
3	pickup_latitude	0.072006
2	pickup_longitude	0.068859
5	dropoff_latitude	0.067797
14	dow_drop	0.066083
4	dropoff_longitude	0.057027
6	doy_pick	0.031181
10	hod_pick	0.030345
15	hod_drop	0.024778
12	woy_drop	0.008782
1	passenger_count	0.007822
7	woy_pick	0.006276
8	moy_pick	0.004769
13	moy_drop	0.004636
0	vendor_id	0.000550
16	store_and_fwd_flag_N	0.000007
17	store_and_fwd_flag_Y	0.000005

Drop day of the year has high importance

Store and forward flag has low importance

In [64]:

```
# remove the lowest 3 and plot a bar graph
df_upd = df_sort.drop([0,16,17])
```

In [65]:

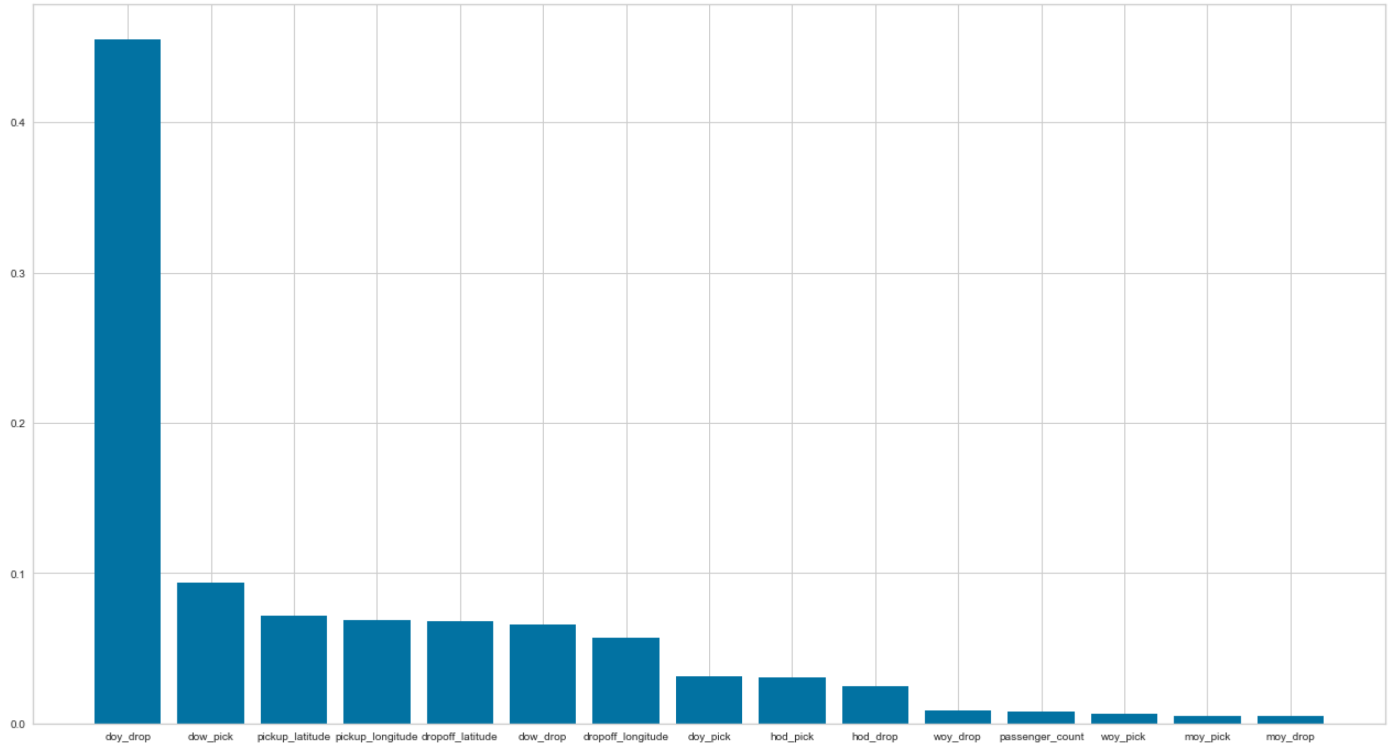
```
df_upd
```

Out[65]:

	Feature Names	Importances
11	doy_drop	0.455497
9	dow_pick	0.093580
3	pickup_latitude	0.072006
2	pickup_longitude	0.068859
5	dropoff_latitude	0.067797
14	dow_drop	0.066083
4	dropoff_longitude	0.057027
6	doy_pick	0.031181
10	hod_pick	0.030345
15	hod_drop	0.024778
12	woy_drop	0.008782
1	passenger_count	0.007822
7	woy_pick	0.006276
8	moy_pick	0.004769
13	moy_drop	0.004636

In [66]:

```
plt.rcParams['figure.figsize'] = (22,12)
plt.bar(df_upd['Feature Names'],df_upd['Importances'])
plt.show()
```



Drop day of the year has high importance and the nearest neighbour is approximately 5 times less than this variable

Month and Week of the pickup and month of the drop has low importance comparitively.

In [ ]: