**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**import** pylab

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn **import** metrics

**from** sklearn.ensemble **import** RandomForestRegressor

**from** sklearn **import** metrics

**from** sklearn **import** preprocessing

df **=** pd**.**read\_csv('uber.csv')

df**.**info()

df**.**head()

df**.**describe()

df **=** df**.**drop(['Unnamed: 0', 'key'], axis**=**1)

df**.**isna()**.**sum()

df**.**dropna(axis**=**0,inplace**=True**)

df**.**dtypes

df**.**pickup\_datetime **=** pd**.**to\_datetime(df**.**pickup\_datetime, errors**=**'coerce')

df**=** df**.**assign(

second **=** df**.**pickup\_datetime**.**dt**.**second,

minute **=** df**.**pickup\_datetime**.**dt**.**minute,

hour **=** df**.**pickup\_datetime**.**dt**.**hour,

day**=** df**.**pickup\_datetime**.**dt**.**day,

month **=** df**.**pickup\_datetime**.**dt**.**month,

year **=** df**.**pickup\_datetime**.**dt**.**year,

dayofweek **=** df**.**pickup\_datetime**.**dt**.**dayofweek

)

df **=** df**.**drop('pickup\_datetime',axis**=**1)

df**.**info()

df**.**head()

incorrect\_coordinates **=** df**.**loc[

(df**.**pickup\_latitude **>** 90) **|**(df**.**pickup\_latitude **<** **-**90) **|**

(df**.**dropoff\_latitude **>** 90) **|**(df**.**dropoff\_latitude **<** **-**90) **|**

(df**.**pickup\_longitude **>** 180) **|**(df**.**pickup\_longitude **<** **-**180) **|**

(df**.**dropoff\_longitude **>** 90) **|**(df**.**dropoff\_longitude **<** **-**90)

]

df**.**drop(incorrect\_coordinates, inplace **=** **True**, errors **=** 'ignore')

**def** distance\_transform(longitude1, latitude1, longitude2, latitude2):

long1, lati1, long2, lati2 **=** map(np**.**radians, [longitude1, latitude1, longitude2, latitude2])

dist\_long **=** long2 **-** long1

dist\_lati **=** lati2 **-** lati1

a **=** np**.**sin(dist\_lati**/**2)**\*\***2 **+** np**.**cos(lati1) **\*** np**.**cos(lati2) **\*** np**.**sin(dist\_long**/**2)**\*\***2

c **=** 2 **\*** np**.**arcsin(np**.**sqrt(a)) **\*** 6371

*# long1,lati1,long2,lati2 = longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]*

*# c = sqrt((long2 - long1) \*\* 2 + (lati2 - lati1) \*\* 2)asin*

**return** c

df['Distance'] **=** distance\_transform(

df['pickup\_longitude'],

df['pickup\_latitude'],

df['dropoff\_longitude'],

df['dropoff\_latitude']

)

df**.**head()

plt**.**scatter(df['Distance'], df['fare\_amount'])

plt**.**xlabel("Distance")

plt**.**ylabel("fare\_amount")

plt**.**figure(figsize**=**(20,12))

sns**.**boxplot(data **=** df)

df**.**drop(df[df['Distance'] **>=** 60]**.**index, inplace **=** **True**)

df**.**drop(df[df['fare\_amount'] **<=** 0]**.**index, inplace **=** **True**)

df**.**drop(df[(df['fare\_amount']**>**100) **&** (df['Distance']**<**1)]**.**index, inplace **=** **True** )

df**.**drop(df[(df['fare\_amount']**<**100) **&** (df['Distance']**>**100)]**.**index, inplace **=** **True** )

plt**.**scatter(df['Distance'], df['fare\_amount'])

plt**.**xlabel("Distance")

plt**.**ylabel("fare\_amount")

#correlation

corr **=** df**.**corr()

corr**.**style**.**background\_gradient(cmap**=**'BuGn')

#standardization

X **=** df['Distance']**.**values**.**reshape(**-**1, 1) *#Independent Variable*

y **=** df['fare\_amount']**.**values**.**reshape(**-**1, 1) *#Dependent Variable*

**from** sklearn.preprocessing **import** StandardScaler

std **=** StandardScaler()

y\_std **=** std**.**fit\_transform(y)

print(y\_std)

x\_std **=** std**.**fit\_transform(X)

print(x\_std)

#linear regression

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(x\_std, y\_std, test\_size**=**0.2, random\_state**=**0)

**from** sklearn.linear\_model **import** LinearRegression

l\_reg **=** LinearRegression()

l\_reg**.**fit(X\_train, y\_train)

print("Training set score: {:.2f}"**.**format(l\_reg**.**score(X\_train, y\_train)))

print("Test set score: {:.7f}"**.**format(l\_reg**.**score(X\_test, y\_test)))

y\_pred **=** l\_reg**.**predict(X\_test)

result **=** pd**.**DataFrame()

result[['Actual']] **=** y\_test

result[['Predicted']] **=** y\_pred

result**.**sample(10)

print('Mean Absolute Error:', metrics**.**mean\_absolute\_error(y\_test, y\_pred))

print('Mean Absolute % Error:', metrics**.**mean\_absolute\_percentage\_error(y\_test, y\_pred))

print('Mean Squared Error:', metrics**.**mean\_squared\_error(y\_test, y\_pred))

print('Root Mean Squared Error:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_pred)))

print('R Squared (R²):', np**.**sqrt(metrics**.**r2\_score(y\_test, y\_pred)))

rf\_reg **=** RandomForestRegressor(n\_estimators**=**100, random\_state**=**10)

*# fit the regressor with training dataset*

rf\_reg**.**fit(X\_train, y\_train)

*# predict the values on test dataset using predict()*

y\_pred\_RF **=** rf\_reg**.**predict(X\_test)

result **=** pd**.**DataFrame()

result[['Actual']] **=** y\_test

result['Predicted'] **=** y\_pred\_RF

result**.**sample(10)

print('Mean Absolute Error:', metrics**.**mean\_absolute\_error(y\_test, y\_pred\_RF))

print('Mean Absolute % Error:', metrics**.**mean\_absolute\_percentage\_error(y\_test, y\_pred\_RF))

print('Mean Squared Error:', metrics**.**mean\_squared\_error(y\_test, y\_pred\_RF))

print('Root Mean Squared Error:', np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_pred\_RF)))

print('R Squared (R²):', np**.**sqrt(metrics**.**r2\_score(y\_test, y\_pred\_RF)))

*# Build scatterplot*

plt**.**scatter(X\_test, y\_test, c **=** 'b', alpha **=** 0.5, marker **=** '.', label **=** 'Real')

plt**.**scatter(X\_test, y\_pred\_RF, c **=** 'r', alpha **=** 0.5, marker **=** '.', label **=** 'Predicted')

plt**.**xlabel('Carat')

plt**.**ylabel('Price')

plt**.**grid(color **=** '#D3D3D3', linestyle **=** 'solid')

plt**.**legend(loc **=** 'lower right')

plt**.**tight\_layout()

plt**.**show()

*# compile the required information*

random\_forest\_metrics **=** pd**.**DataFrame([[

"Random Forest Regressor model",

np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_pred\_RF)),

np**.**sqrt(metrics**.**r2\_score(y\_test, y\_pred\_RF))

]], columns **=** cols)

result\_tabulation **=** pd**.**concat([result\_tabulation, random\_forest\_metrics], ignore\_index**=True**)

result\_tabulation