Import libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics
from xgboost import XGBClassifier
from xgboost import XGBRegressor
import seaborn as sns
```

In [2]:
```
# load data
```

In [3]:
```
diamonds = sns.load_dataset('diamonds')
diamonds.head()
```

Out[3]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [4]:
```
# Data dictionary
```

price price in US dollars ($326--$18,823)

carat weight of the diamond (0.2--5.01)

cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)

color diamond colour, from J (worst) to D (best)

clarity a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

x length in mm (0--10.74)

y width in mm (0--58.9)

z depth in mm (0--31.8)

depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)

table width of top of diamond relative to widest point (43--95)

In [5]:
```
# shape

diamonds.shape
```

Out[5]:
```
(53940, 10)
```

In [6]:
```
# null value

diamonds.isnull().sum()
```

Out[6]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

In [7]:
```
# basic EDA
# describe
# outlier
# convert categorical to numerical features
# try scaling
```

In [17]:
```
# selecting 1st 15000 values for further steps
df = diamonds.copy()
#df = diamonds.iloc[0:15000,:]
df.shape
```

Out[17]:
```
(53940, 10)
```

In [18]:
```
df.head()
```

Out[18]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [19]:
```
df.columns
```

Out[19]:
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
       'z'],
      dtype='object')
```

In [20]:
```
# separate data into features and target

X = df.drop(['cut', 'color', 'clarity', 'price'],axis = 1)
# dropping categorical features for time sake,
# or else we can convert into numerical to get more features

y = df['price']
```

In [21]:
```
X.shape
```

Out[21]:
```
(53940, 6)
```

In [22]:
```
y.shape
```

Out[22]:
```
(53940,)
```

In [23]:
```python
# split the data in training and testing set

X_train, X_test, y_train,y_test =
train_test_split(X,y, test_size = 0.30, random_state = 42)
```

In [24]:
```python
print("X_train shape : " , X_train.shape)
print("X_test shape : " , X_test.shape)
print("y_train shape : " , y_train.shape)
print("y_test shape : " , y_test.shape)
```

```
X_train shape :  (37758, 6)
X_test shape :  (16182, 6)
y_train shape :  (37758,)
y_test shape :  (16182,)
```

In [25]:
```python
# fit model on training data
model = XGBRegressor()
model.fit(X_train,y_train)
```

Out[25]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [26]:
```python
# make predictions for test data

y_pred = model.predict(X_test)
print(y_pred)
```

```
[  516.4466   1773.6815   1004.8858  ... 11196.855    2670.2507
  1020.07666]
```

In [28]:
```python
# Mean squared error
print("MSE: ",metrics.mean_squared_error(y_test,y_pred))
```

```
MSE:  1850744.056558767
```

In [29]:
```python
rmse = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
print(rmse)
```

```
1360.420544007906
```

In [ ]:
```python

```

XGB on house prediction data

In [30]:
```python
from sklearn.datasets import load_boston
boston_dataset = load_boston()
boston = pd.DataFrame(boston_dataset.data,
                      columns = boston_dataset.feature_names)
boston.head()
```

Out[30]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [31]:
```python
# target variable

boston['MEDV'] = boston_dataset.target
```

In [32]:
```
X1 = pd.DataFrame(boston.iloc[:,:-1])
y1 = pd.DataFrame(boston.iloc[:,-1])

print("Features set : ",X1.shape)
print("Target : ",y1.shape)
```

```
Features set :  (506, 13)
Target :  (506, 1)
```

In [33]:
```
# split the data in training and testing set

X_train1, X_test1, y_train1,y_test1 =
train_test_split(X1,y1, test_size = 0.20, random_state = 42)
```

In [34]:
```
# fit model on training data
model1 = XGBRegressor()
model1.fit(X_train1,y_train1)
```

Out[34]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [35]:
```
# make predictions for test data

y_pred1 = model1.predict(X_test1)
print(y_pred1)
```

```
[23.25328   30.024755  15.632249  23.313478  17.775118  21.142563
 20.19583   15.010124  21.23614   22.242369  20.457346  19.209145
  8.551788  21.210636  20.696491  26.74365   18.824339  10.525872
 45.68885   14.116162  26.618996  24.94542   13.3510275 20.87231
 15.400073  15.636547  22.324673  12.777009  20.726126  22.56401
 20.346395  22.303246  18.523277  21.764612  15.568828  15.683646
 33.073547  19.115112  21.955132  22.399914  18.998787  31.328337
 43.464993  18.20766   22.09233   14.353467  14.607512  22.716745
 19.700527  27.072327  22.579268  35.133675  16.241447  25.214682
 46.013332  21.89786   15.043295  32.93268   20.53731   16.568089
 24.07178   34.34796   28.542194  16.977676  25.867334  15.649837
 13.039615  23.00082   27.26897   15.414835  21.546648  31.72919
 10.665012  20.770847  21.848396   6.475782  20.939093  46.59454
 12.456056   8.739085  22.215406  13.390212  20.454681  10.45914
 19.722834  27.327946  16.254663  23.860172  25.414312  17.06042
 22.9362     8.106883  19.001764  18.869307  24.129864  19.66075
 40.517284  13.981451  11.416717  15.428753  19.41982   24.281776 ]
```

In [36]:
```
# Mean squared error
print("MSE: ",metrics.mean_squared_error(y_test1,y_pred1))
```

```
MSE:  6.560527271813469
```

In [37]:
```
rmse = np.sqrt(metrics.mean_squared_error(y_test1,y_pred1))
print(rmse)
```

```
2.561352625433185
```

In [ ]:

Apply xgboost on classification problem

In [40]:
```
df_1 = pd.read_csv('diabetes.csv')
df_1.head()
```

Out[40]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [41]:
```
# seperate out features and target value from dataset

X_1 = df_1.drop(['Outcome'],axis = 1).values
y_1 = df_1['Outcome'].values
```

In [42]:
```
# split the data in training and testing set

X_train_1, X_test_1, y_train_1,y_test_1 =
train_test_split(X_1,y_1, test_size = 0.25, random_state = 42)
```

In [43]:
```
print("X_train shape : " , X_train_1.shape)
print("X_test shape : " , X_test_1.shape)
print("y_train shape : " , y_train_1.shape)
print("y_test shape : " , y_test_1.shape)
```

```
X_train shape :  (576, 8)
X_test shape :  (192, 8)
y_train shape :  (576,)
y_test shape :  (192,)
```

In [44]:
```
# fit model on training data
xgb_clf = XGBClassifier()
```

In [45]:
```
xgb_clf.fit(X_train_1,y_train_1)
```

```
/Users/kunalshriwas/opt/anaconda3/lib/python3.8/site-packages/xgboost/sklearn.py:888: UserWarning: The
use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove
this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier
object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[21:23:21] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgbo
ost-split_1619728204606/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eva
l_metric if you'd like to restore the old behavior.
```

Out[45]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [46]:
```
# make predictions for test data

y_pred_1 = xgb_clf.predict(X_test_1)
```

In [47]:
```
acc_xgb = metrics.accuracy_score(y_test_1,y_pred_1)
print("Accuracy : ",acc_xgb)
```

```
Accuracy :  0.75
```

In [ ]: