Decision Tree

Methodology

- **Data Cleaning:** Checking for null values and based on their number either droping them or replacing with mean, median, mode based on the type and description of data. Droping decscrete and catagorical variables that have highly skewed histograms.
- **Data Visualization:** This step helps understand the understand the data in a visually. We can understand normality of the data as well. This helps us to decide whether to normalize the data. In case of catagorical variables it also helps in feature selection.
- **Feature Selection:** Based on the Pearson correlation between the labeled column and rest of the features. In general, a very great correlation should have an absolute value greater than 0.75. When the labeled column is depended on multiple columns, the correlation with one column may be less. But combined features may have higher effect.
- **Train Test Split:** We split the data into 80:20 ratio for tarining testing respectively.
- **Model Selection:** Based on the data visualization and data correlation, we need to select a model that would best suit. Here we need to use XGBOOST.
- **Evalution:** In this case we are using RMSE, R2 Score to determine the accuracy of the predicting model.

importing libraries

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

Reading data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", for
```

```
df=pd.read_csv(r"drive/My Drive/biddings.csv")
```

Null value percentages

```
Null=[]
for i in df:
    Null.append((i,df[i].isna().mean()*100))
Null=pd.DataFrame(Null,columns=['class','per'])
Null
```

	class	per
0	0	0.0
1	1	0.0
2	2	0.0

ALL the columns are having nonull values

```
df.dtypes
```

```
0          float64
1          float64
2          float64
3          float64
4          float64
            ...
84         float64
85         float64
86         float64
87         float64
convert      int64
Length: 89, dtype: object
```

All the Data types are float

Under Sampling

```
from imblearn.under_sampling import RandomUnderSampler
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning: The module is deprecated in
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.neighbors.b
  warnings.warn(message, FutureWarning)
```

```
c=df.columns
df[c[-1]].value_counts()/len(df)*100
```

```
0    99.8092
1     0.1908
Name: convert, dtype: float64
```

```
df[c[-1]].value_counts()/len(df)*100
```

```
0    99.8092
1     0.1908
Name: convert, dtype: float64
```

```
cols = [col for col in df.columns if col not in ["convert"]]
X = df[cols]
```

```
y=df["convert"]
rus = RandomUnderSampler(random_state=0)
rus.fit(X, y)
X, y = rus.fit_sample(X, y)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing i
  warnings.warn(msg, category=FutureWarning)
```

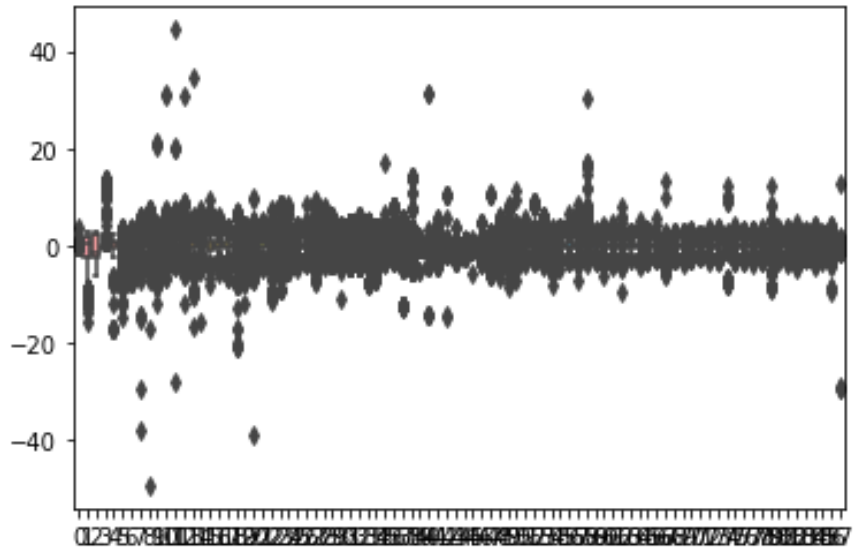
```
f = pd.DataFrame(X)
```

```
df=f
```

Outliers detection and removal

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f24fe9966a0>
```



most of he data has outliers

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
df1 = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
0     0.0200
1     4.3100
2     4.5925
3     0.5800
4     1.3600
       ...
83    0.6100
84    0.3800
85    0.2700
86    0.6200
87    0.1900
Length: 88, dtype: float64
```

```
df1.shape
```

```
(476, 88)
```

removing columns wll only give 476 data points which is less for learning

```
mask=(((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()/len(df) > 0.1)
```

```
for i,col in zip(mask,c):
  if i:
    df=df.drop(float(col),axis=1)
```

removing columns with 10% outliers

```
df
```

	0	1	2	3	4	7	11	13	15	16	17	18	20	21	23	24	25	
0	-0.02	1.99	2.40	-0.61	0.89	0.16	0.37	-0.57	0.07	0.07	1.18	-0.40	0.01	0.22	0.14	0.35	0.19	0.
1	0.00	-3.72	-4.22	0.00	-1.04	-0.68	0.09	1.47	1.03	-0.40	0.75	-0.47	-0.04	2.46	0.53	-0.09	0.32	-0.
2	0.01	-2.46	2.20	0.71	-1.37	-1.11	-2.04	0.74	-3.97	-0.76	-0.50	-0.03	-0.52	-0.93	-2.56	-3.16	-2.32	-0

i

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.2)
```

3811	0.01	-4.20	2.66	-0.06	-0.03	-1.67	-0.94	-0.24	-0.26	-3.76	0.51	-0.12	0.05	-1.51	-1.02	-1.25	0.56	0

```
from sklearn.linear_model import LogisticRegression as LR
model =LR()
model.fit(X_train, y_train)
model.score(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
0.6592398427260813
```

```
model.score(X_test, y_test)
```

```
0.6518324607329843
```

Decision Tree

```
from sklearn import tree
model =tree.DecisionTreeClassifier()
```

```
model.fit(X_train, y_train)
model.score(X_train, y_train)
```

```
0.9891874180865007
```

```
model.score(X_test, y_test)
```

```
0.5575916230366492
```