# Analyze Helpdesk tickets

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as stats
from sklearn.metrics import confusion_matrix
from sklearn.metrics import RocCurveDisplay
from sklearn.preprocessing import RobustScaler
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

In [ ]:
```
tickets = pd.read_csv("C:/Users/HP/Desktop/tickets/WA_Fn-UseC_-IT-Help-Desk.csv")
```

In [ ]:
```
tickets.columns
```

Out[ ]:
```
Index(['ticket', 'requestor', 'RequestorSeniority', 'ITOwner', 'FiledAgainst',
       'TicketType', 'Severity', 'Priority', 'daysOpen', 'Satisfaction'],
      dtype='object')
```

In [ ]:
```
tickets = tickets[['ticket', 'requestor', 'RequestorSeniority', 'ITOwner', 'FiledAgainst',
       'TicketType', 'Priority', 'daysOpen', 'Satisfaction']]
```

In [ ]:
```
tickets.head(3)
```

Out[ ]:

| | ticket | requestor | RequestorSeniority | ITOwner | FiledAgainst | TicketType | Priority | daysOpen | Satisfaction |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1929 | 1 - Junior | 50 | Systems | Issue | 0 - Unassigned | 3 | 1 - Unsatisfied |
| **1** | 2 | 1587 | 2 - Regular | 15 | Software | Request | 1 - Low | 5 | 1 - Unsatisfied |
| **2** | 3 | 925 | 2 - Regular | 15 | Access/Login | Request | 0 - Unassigned | 0 | 0 - Unknown |

## Exploratory Data Science

In [ ]:
```
tickets.shape
```

Out[ ]:
```
(100000, 9)
```

In [ ]:
```
tickets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   ticket              100000 non-null  int64 
 1   requestor           100000 non-null  int64 
 2   RequestorSeniority  100000 non-null  object
 3   ITOwner             100000 non-null  int64 
 4   FiledAgainst        100000 non-null  object
 5   TicketType          100000 non-null  object
 6   Priority            100000 non-null  object
 7   daysOpen            100000 non-null  int64 
 8   Satisfaction        100000 non-null  object
dtypes: int64(4), object(5)
memory usage: 6.9+ MB
```

In [ ]:
```
train, test = train_test_split(tickets, test_size=0.25, random_state=42, shuffle=True)
```

## Missing Value Imputation

In [ ]:
```
train.size, test.size
```

Out[ ]:
```
(675000, 225000)
```

In [ ]:
```
tickets.isnull().sum()
```

Out[ ]:
```
ticket                0
requestor             0
RequestorSeniority    0
ITOwner               0
FiledAgainst          0
TicketType            0
Priority              0
daysOpen              0
Satisfaction          0
dtype: int64
```

In [ ]:
```
print(tickets.isnull().sum()/tickets.shape[0] *100)
```

```
ticket                0.0
requestor             0.0
RequestorSeniority    0.0
ITOwner               0.0
FiledAgainst          0.0
TicketType            0.0
Priority              0.0
daysOpen              0.0
Satisfaction          0.0
dtype: float64
```

In [ ]:
```
tickets.Satisfaction.value_counts()
```

Out[ ]:
```
0 - Unknown             30211
3 - Highly satisfied    29063
1 - Unsatisfied         21124
2 - Satisfied           19602
Name: Satisfaction, dtype: int64
```

In [ ]:
```
tickets.Satisfaction = tickets.Satisfaction.astype('category').cat.codes
```

In [ ]:
```
tickets.head(2)
```

Out[ ]:

| | ticket | requestor | RequestorSeniority | ITOwner | FiledAgainst | TicketType | Priority | daysOpen | Satisfaction |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1929 | 1 - Junior | 50 | Systems | Issue | 0 - Unassigned | 3 | 1 |
| **1** | 2 | 1587 | 2 - Regular | 15 | Software | Request | 1 - Low | 5 | 1 |

In [ ]:
```
tickets.Priority = tickets.Priority.astype('category').cat.codes
# tickets.Severity = tickets.Severity.astype('category').cat.codes
tickets.TicketType = tickets.TicketType.astype('category').cat.codes
tickets.FiledAgainst = tickets.FiledAgainst.astype('category').cat.codes
tickets.RequestorSeniority = tickets.RequestorSeniority.astype('category').cat.codes
```

## Scaling

In [ ]:
```
tickets.dtypes
```

Out[ ]:
```
ticket               int64
requestor            int64
RequestorSeniority    int8
ITOwner              int64
FiledAgainst          int8
TicketType            int8
Priority              int8
daysOpen             int64
Satisfaction          int8
dtype: object
```

In [ ]:
```
#Lets import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as st
import sklearn.datasets as dts
import matplotlib.pyplot as plt
from itertools import permutations
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
X = (tickets.drop(columns=tickets[['Priority']],axis=0)).values
Y = (tickets.iloc[:,-1:]).values
```

In [ ]:
```
print(X.shape, Y.shape)
```

```
(100000, 8) (100000, 1)
```

In [ ]:
```
scaler =StandardScaler()
X=scaler.fit_transform(X)
```

In [ ]:
```
X  = pd.DataFrame(X)
```

## Gaussian Or Normalized

In [ ]:
```
import scipy.stats as stat
import pylab
```

In [ ]:
```
#### If you want to check whether feature is guassian or normal distributed
#### Q-Q plot
def plot_data(df,feature):
    plt.figure(figsize=(10,6))
    plt.subplot(1,2,1)
    df[feature].hist()
    plt.subplot(1,2,2)
    stat.probplot(df[feature],dist='norm',plot=pylab)
    plt.show()
```

In [ ]:
```
X.head(3)
```

Out[ ]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **0** | -1.732033 | 1.610322 | -1.361357 | 1.698449 | 1.023391 | -1.735475 | -0.520861 | -0.396683 |
| **1** | -1.731999 | 1.018120 | -0.378236 | -0.724050 | 0.234851 | 0.576211 | -0.249779 | -0.396683 |
| **2** | -1.731964 | -0.128191 | -0.378236 | -0.724050 | -1.342229 | 0.576211 | -0.927484 | -1.231507 |

In [ ]:
```
X.rename(columns={0:"ticket",
1:"requestor", 2:"RequestorSeniority", 3:"ITOwner", 4:"FiledAgainst", 5:"TicketType",
6:"Severity", 7:"daysOpen", 8:"Satisfaction"}, inplace=True)
```

In [ ]:
```
X.head(2)
```

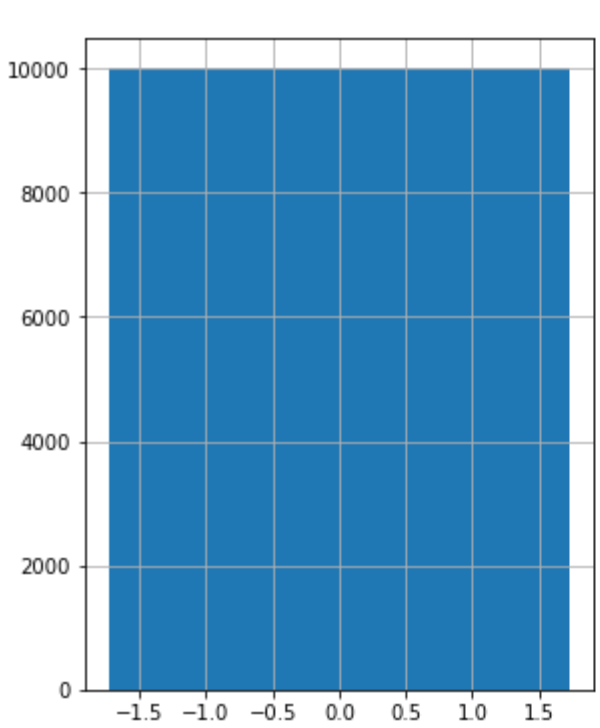
Out[ ]:

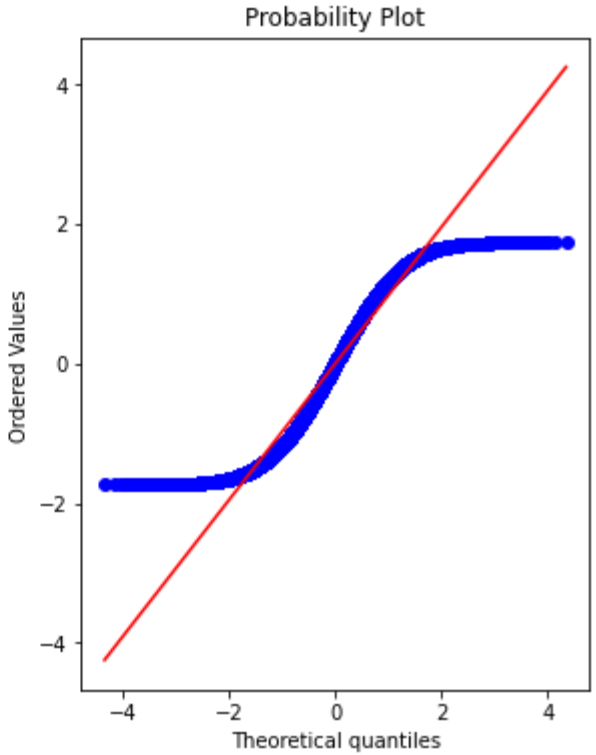
| | ticket | requestor | RequestorSeniority | ITOwner | FiledAgainst | TicketType | Severity | daysOpen |
|---|---|---|---|---|---|---|---|---|
| **0** | -1.732033 | 1.610322 | -1.361357 | 1.698449 | 1.023391 | -1.735475 | -0.520861 | -0.396683 |
| **1** | -1.731999 | 1.018120 | -0.378236 | -0.724050 | 0.234851 | 0.576211 | -0.249779 | -0.396683 |

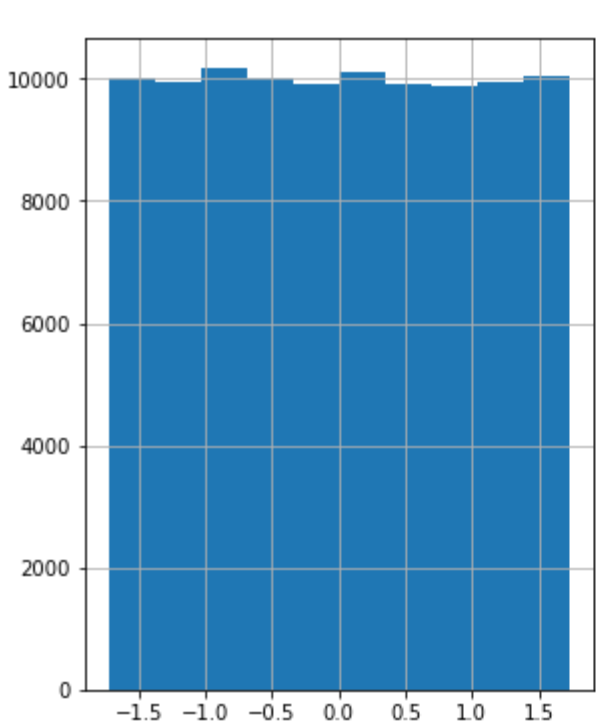
In [ ]:
```
X.columns
```

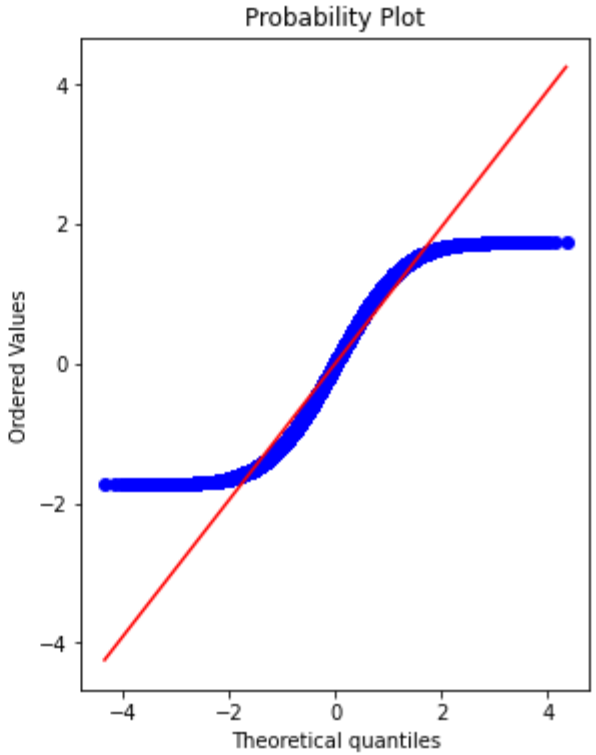
Out[ ]:
```
Index(['ticket', 'requestor', 'RequestorSeniority', 'ITOwner', 'FiledAgainst',
       'TicketType', 'Severity', 'daysOpen'],
      dtype='object')
```

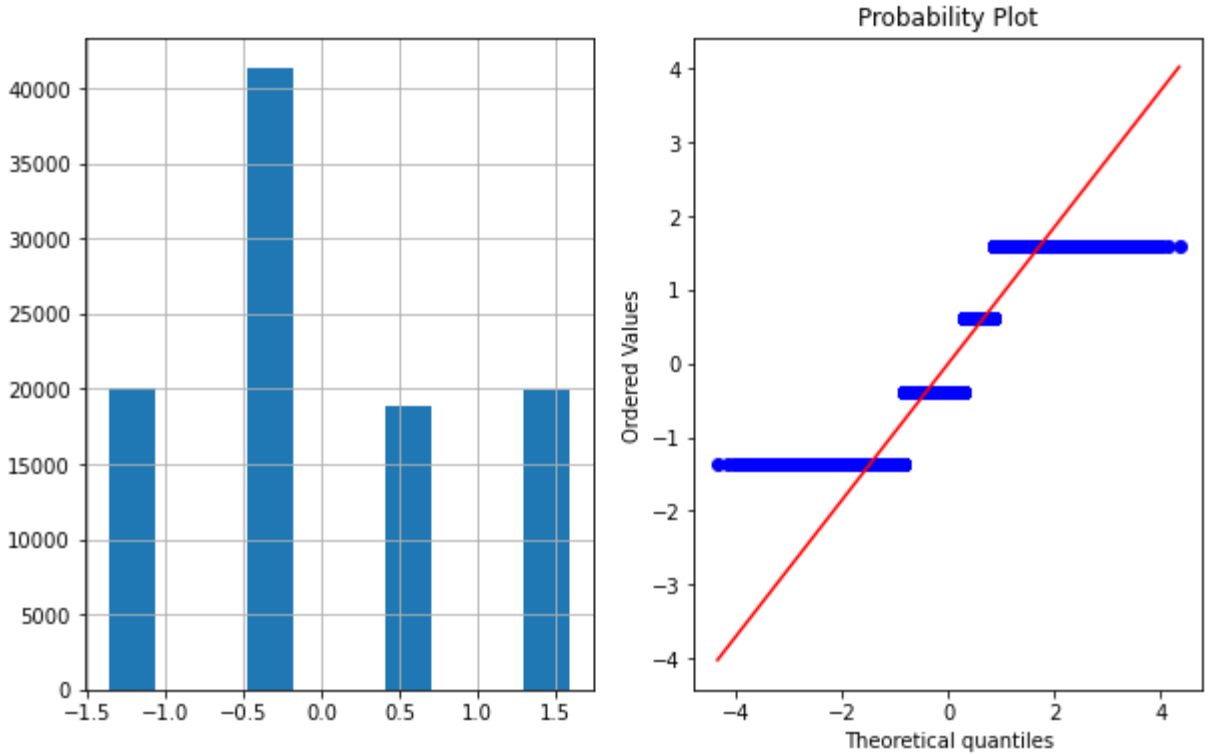
In [ ]:
```
plot_data(X,'ticket')
plot_data(X,'requestor')
plot_data(X,'RequestorSeniority')
plot_data(X, 'ITOwner')
plot_data(X, 'FiledAgainst')
plot_data(X, 'TicketType')
plot_data(X, 'Severity')
plot_data(X, 'daysOpen')
## plot_data(X, 'Satisfaction')
```

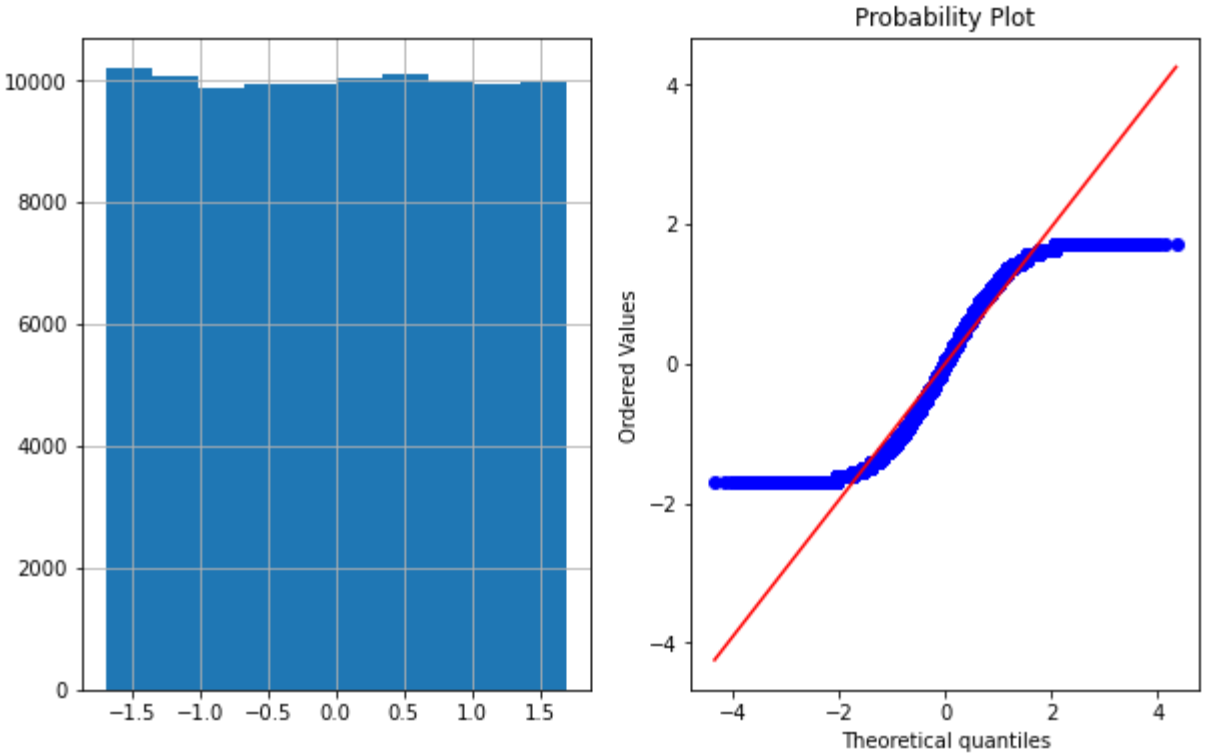

10000
8000
6000
4000
2000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

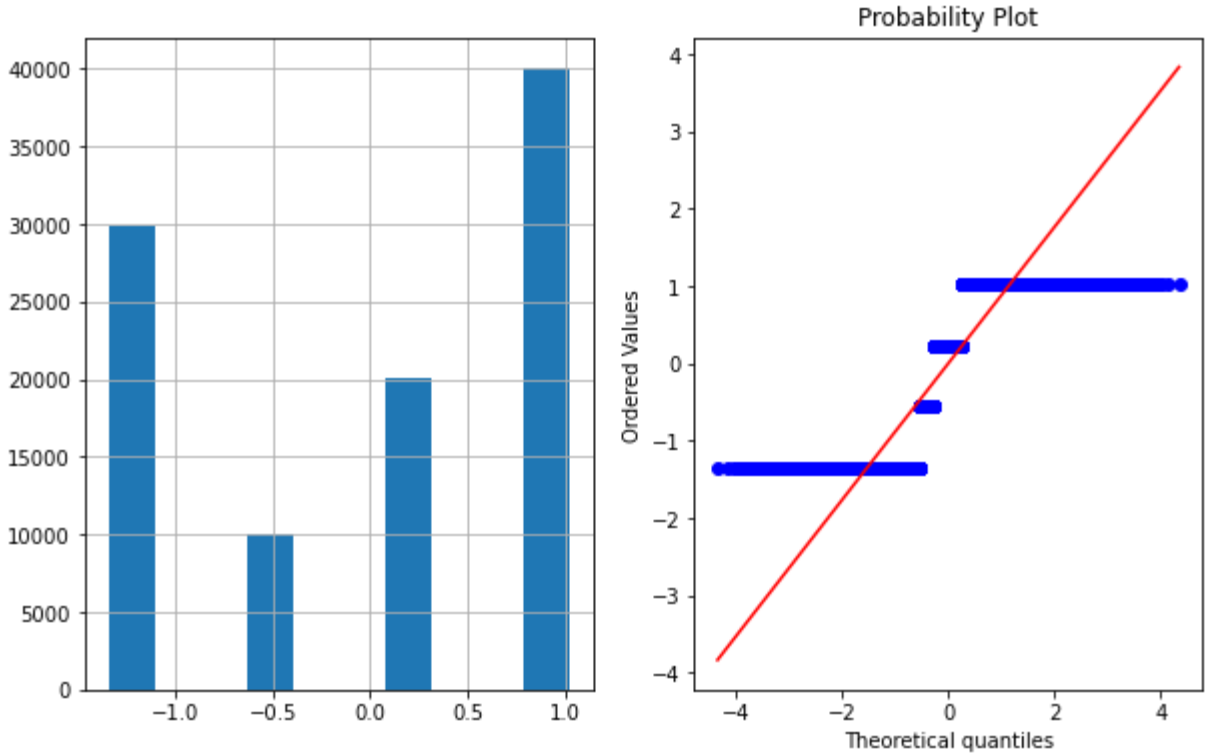

Probability Plot
Ordered Values
Theoretical quantiles
4
2
0
−2
−4
−4
−2
0
2
4

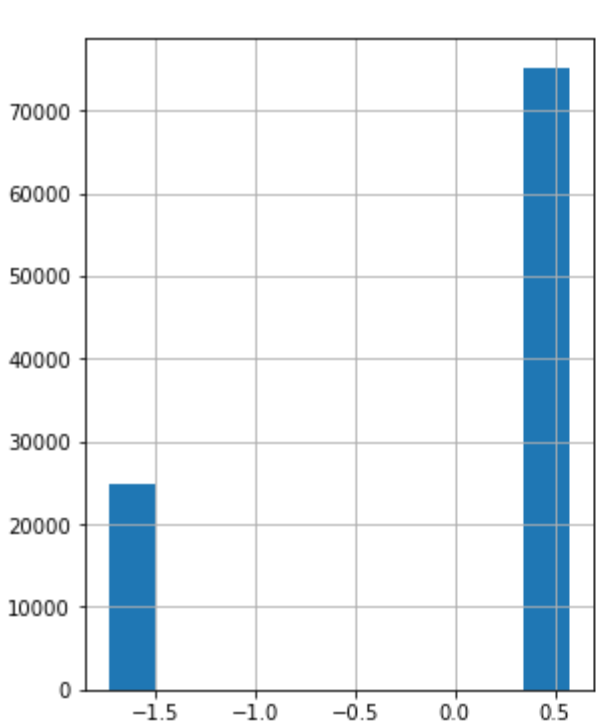

10000
8000
6000
4000
2000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

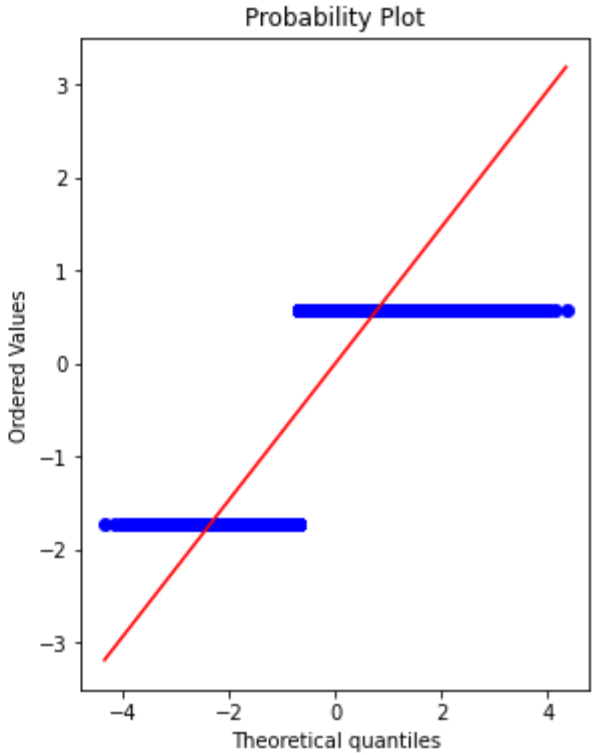

Probability Plot
Ordered Values
Theoretical quantiles
4
2
0
−2
−4
−4
−2
0
2
4

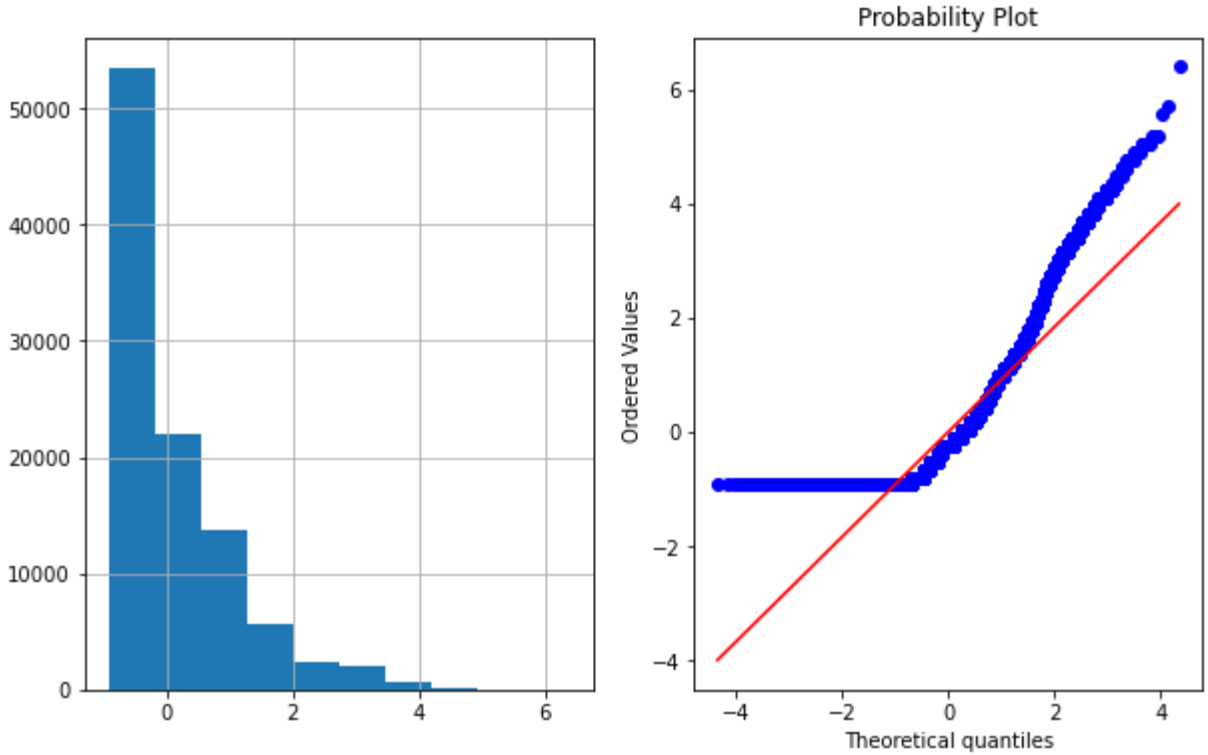

40000
35000
30000
25000
20000
15000
10000
5000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Probability Plot
Ordered Values
Theoretical quantiles
4
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

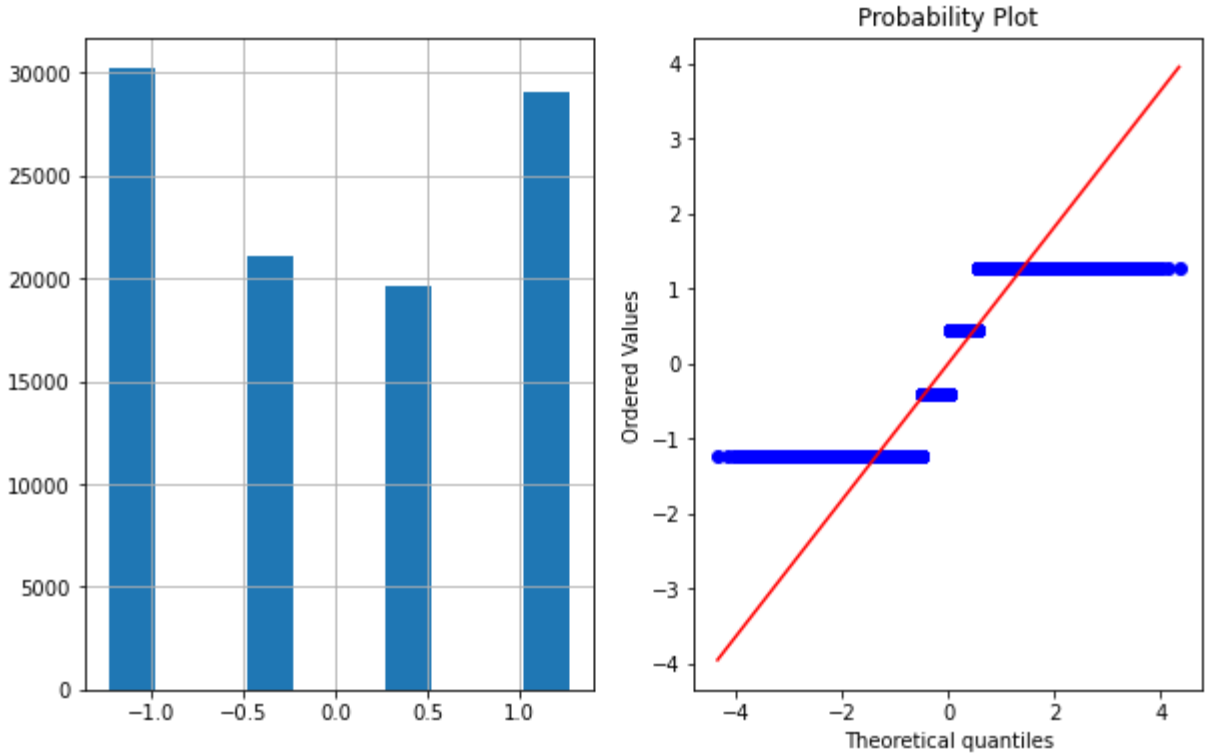

Probability Plot
Ordered Values
Theoretical quantiles
10000
8000
6000
4000
2000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
4
2
0
−2
−4
−4
−2
0
2
4


Probability Plot
Ordered Values
Theoretical quantiles


70000
60000
50000
40000
30000
20000
10000
0
−1.5
−1.0
−0.5
0.0
0.5


Probability Plot
Ordered Values
Theoretical quantiles
3
2
1
0
−1
−2
−3
−4
−2
0
2
4


Probability Plot
Ordered Values
Theoretical quantiles
50000
40000
30000
20000
10000
0
6
4
2
0
−2
−4
−4
−2
0
2
4
6




In [ ]:
```
Y = pd.DataFrame(Y)
```

In [ ]:
```
Y.rename(columns={0:"Priority"}, inplace=True)
```

In [ ]:
```
Y.value_counts()
```

Out[ ]:
```
Priority
0           30211
3           29063
1           21124
2           19602
dtype: int64
```

## Feature Selection

In [ ]:
```
from sklearn.ensemble import ExtraTreesClassifier
```

In [ ]:
```
Ext = ExtraTreesClassifier()
Ext.fit(X,Y)
```

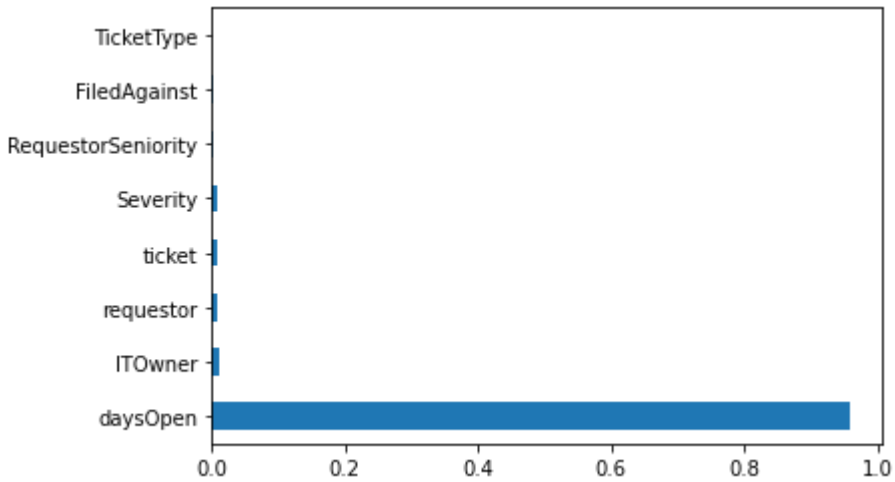
Out[ ]:
```
ExtraTreesClassifier()
```

In [ ]:
```
print(Ext.feature_importances_)
```

```
[8.40263519e-03 8.50517310e-03 2.18916789e-03 1.15485811e-02
 1.99961561e-03 6.57980509e-04 8.18007896e-03 9.58516768e-01]
```

In [ ]:
```
feature = pd.Series(Ext.feature_importances_,index=X.columns)
feature.sort_values(ascending=True).nlargest(15).plot(kind='barh')
```

Out[ ]:
```
<AxesSubplot:>
```




In [ ]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y)
```

In [ ]:
```
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

```
(75000, 8) (25000, 8) (75000, 1) (25000, 1)
```

## Correlation - To Check `Multicollinearity`

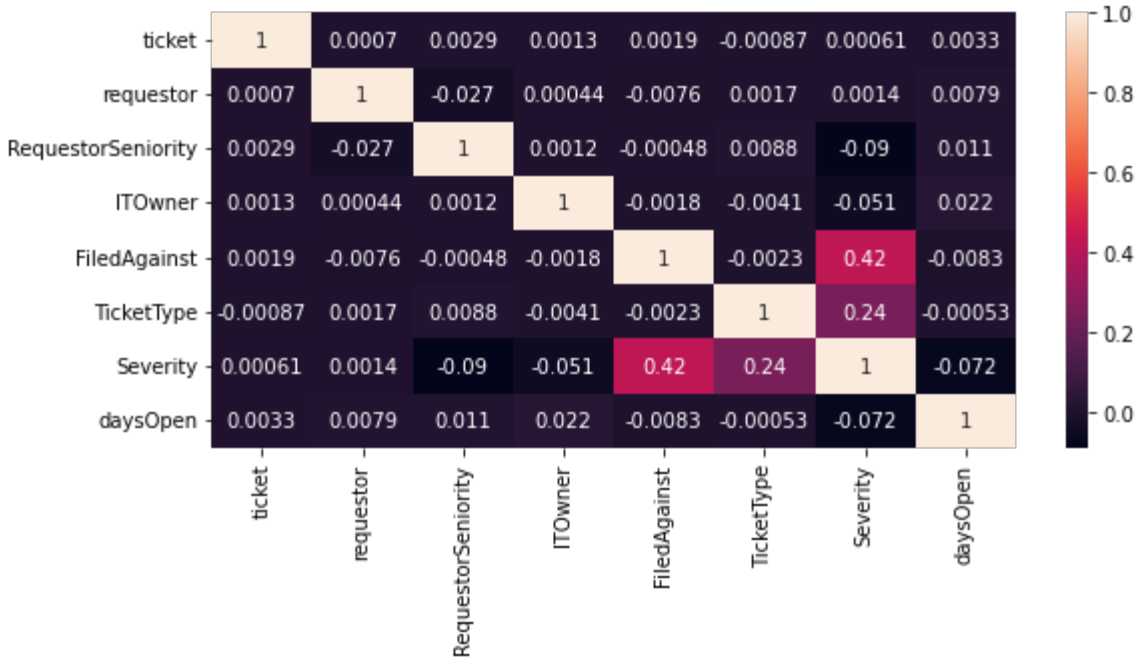
In [ ]:
```
X_train.corr()
```

Out[ ]:

| | ticket | requestor | RequestorSeniority | ITOwner | FiledAgainst | TicketType | Severity | daysOpen |
|---|---|---|---|---|---|---|---|---|
| **ticket** | 1.000000 | 0.000698 | 0.002947 | 0.001265 | 0.001888 | -0.000871 | 0.000612 | 0.003305 |
| **requestor** | 0.000698 | 1.000000 | -0.027485 | 0.000436 | -0.007603 | 0.001678 | 0.001363 | 0.007854 |
| **RequestorSeniority** | 0.002947 | -0.027485 | 1.000000 | 0.001227 | -0.000484 | 0.008806 | -0.090215 | 0.011447 |
| **ITOwner** | 0.001265 | 0.000436 | 0.001227 | 1.000000 | -0.001766 | -0.004079 | -0.051396 | 0.022118 |
| **FiledAgainst** | 0.001888 | -0.007603 | -0.000484 | -0.001766 | 1.000000 | -0.002309 | 0.422771 | -0.008321 |
| **TicketType** | -0.000871 | 0.001678 | 0.008806 | -0.004079 | -0.002309 | 1.000000 | 0.243748 | -0.000534 |
| **Severity** | 0.000612 | 0.001363 | -0.090215 | -0.051396 | 0.422771 | 0.243748 | 1.000000 | -0.072246 |
| **daysOpen** | 0.003305 | 0.007854 | 0.011447 | 0.022118 | -0.008321 | -0.000534 | -0.072246 | 1.000000 |

In [ ]:
```
import seaborn as sns
corr=X_train.corr()
top_features=corr.index
plt.figure(figsize=(9,4))
sns.heatmap(X_train[top_features].corr(),annot=True)
```

Out[ ]:
```
<AxesSubplot:>
```




In [ ]:
```
threshold=0.7
```

In [ ]:
```
# find and remove correlated features
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

In [ ]:
```
correlation(X_train,threshold)
```

Out[ ]:
```
set()
```

## Pipeline Creation

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
##from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
```

In [ ]:
```
pipeline_lr=Pipeline([('scalar1',RobustScaler()),
                      ('pca1',PCA(n_components=2)),
                      ('lr_classifier',LogisticRegression(random_state=0))])
```

In [ ]:
```
pipeline_dt=Pipeline([('scalar2',RobustScaler()),
                      ('pca2',PCA(n_components=2)),
                      ('dt_classifier',DecisionTreeClassifier())])
```

In [ ]:
```
pipeline_randomforest=Pipeline([('scalar3',RobustScaler()),
                      ('pca3',PCA(n_components=2)),
                      ('rf_classifier',RandomForestClassifier())])
```

In [ ]:
```
pipeline_gradient_boost=Pipeline([('scalar4',RobustScaler()),
                      ('pca4',PCA(n_components=2)),
                      ('gb_classifier',GradientBoostingClassifier())])
```

In [ ]:
```
pipeline_Adaboost=Pipeline([('scalar5',RobustScaler()),
                          ('pca5',PCA(n_components=2)),
                          ('xgb_classifier',AdaBoostClassifier())])
```

In [ ]:
```
pipeline_knn=Pipeline([('scalar6',RobustScaler()),
                          ('pca6',PCA(n_components=2)),
                          ('knn_classifier',KNeighborsClassifier())])
```

In [ ]:
```
## LEts make the list of pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_randomforest,pipeline_gradient_boost,pipeline_Adaboost, pipeline_knn]
```

In [ ]:
```
best_accuracy=0.0
best_classifier=0
best_pipeline=""
```

## For `Target` Label Category

In [ ]:
```
# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'RandomForest', 3: 'Gradient Boost', 4: 'Ada Boost', 5:

# Fit the pipelines
for pipe in pipelines:
        pipe.fit(X_train,Y_train["Priority"])
```

In [ ]:
```
for i,model in enumerate(pipelines):
    print("{} Accuracy: {}".format(pipe_dict[i],model.score(X_train,Y_train["Priority"])))
```

```
Logistic Regression Accuracy: 0.30616
Decision Tree Accuracy: 1.0
RandomForest Accuracy: 1.0
Gradient Boost Accuracy: 0.3792933333333333
Ada Boost Accuracy: 0.3414
KNN Accuracy: 0.5719333333333333
```

## Hyper Parameter Tuning

In [ ]:
```
gradient_grid = {
                'max_depth': [8, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 150, None],
                'max_features': ['auto', 'sqrt'],
                'min_samples_leaf': [10, 20, 40, 50, 60, 70, 80, 90],
                'min_samples_split': [200, 500, 800, 1000]
                }
```

In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
gb_random = RandomizedSearchCV(estimator = gb,
                               param_distributions = gradient_grid,
                               n_iter = 2,
                               cv =2,
                               verbose=2,
                               random_state=42)
```

In [ ]:
```
# Fit the random search model
gb_random.fit(X_train, Y_train)
```

Out[ ]:
```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  28.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  27.5s
[CV] END max_depth=60, max_features=sqrt, min_samples_leaf=60, min_samples_split=1000; total time=  35.5s
[CV] END max_depth=60, max_features=sqrt, min_samples_leaf=60, min_samples_split=1000; total time=  33.4s
RandomizedSearchCV(cv=2, estimator=GradientBoostingClassifier(), n_iter=2,
                   param_distributions={'max_depth': [8, 10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110, 120,
                                                      150, None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [10, 20, 40, 50, 60,
                                                             70, 80, 90],
                                        'min_samples_split': [200, 500, 800,
                                                              1000]},
                   random_state=42, verbose=2)
```

In [ ]:
```
gb_random.best_params_
```

Out[ ]:
```
{'min_samples_split': 800,
 'min_samples_leaf': 20,
 'max_features': 'sqrt',
 'max_depth': 10}
```

In [ ]:
```
best_random_grid=gb_random.best_estimator_
```

In [ ]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
y_pred_random=best_random_grid.predict(X_test)
```

In [ ]:
```
print(classification_report(Y_test, y_pred_random))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7566
           1       1.00      1.00      1.00      5318
           2       1.00      1.00      1.00      4932
           3       1.00      1.00      1.00      7184

    accuracy                           1.00     25000
   macro avg       1.00      1.00      1.00     25000
weighted avg       1.00      1.00      1.00     25000
```

## Grid Search

In [ ]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'min_samples_split': [gb_random.best_params_['min_samples_split'] - 40,
    gb_random.best_params_['min_samples_split'],
    gb_random.best_params_['min_samples_split'] + 40],

    'min_samples_leaf': [gb_random.best_params_['min_samples_leaf'] - 10,
    gb_random.best_params_['min_samples_leaf'],
    gb_random.best_params_['min_samples_leaf'] + 10],

    'max_features': [gb_random.best_params_['max_features']],

    'max_depth': [gb_random.best_params_['max_depth'] - 10,
    gb_random.best_params_['max_depth'],
    gb_random.best_params_['max_depth'] + 10 ]
}

print(param_grid)
```

```
{'min_samples_split': [760, 800, 840], 'min_samples_leaf': [10, 20, 30], 'max_features': ['sqrt'], 'max_depth': [0,
10, 20]}
```

In [ ]:
```
grid_search=GridSearchCV(estimator=best_random_grid,param_grid=param_grid,cv=3,verbose=2)
grid_search.fit(X_train,Y_train)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=   0.1s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=   0.1s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=   0.0s
[CV] END max_depth=0, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=  41.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=  36.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=  35.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=  36.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=  37.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=  37.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=  36.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=  36.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=  36.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=  37.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=  40.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=  40.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  50.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  36.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  34.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=  34.4s
```

```
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=  37.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=  35.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=  38.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=  41.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=  41.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=  41.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=  36.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=  40.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=  38.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=  35.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=  40.1s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=  51.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=  49.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=760; total time=  50.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=  47.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=  49.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=800; total time=  49.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=  47.8s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=  52.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=840; total time=  48.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=  46.8s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=  46.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=760; total time=  47.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  45.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  46.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=800; total time=  48.9s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=  45.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=  47.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=20, min_samples_split=840; total time=  48.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=  58.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time= 1.1min
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=760; total time=  50.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=  49.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=  50.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=800; total time=  53.1s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=  51.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=  53.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=30, min_samples_split=840; total time=  49.0s
```

Out[ ]:
```
GridSearchCV(cv=3,
             estimator=GradientBoostingClassifier(max_depth=10,
                                                  max_features='sqrt',
                                                  min_samples_leaf=20,
                                                  min_samples_split=800),
             param_grid={'max_depth': [0, 10, 20], 'max_features': ['sqrt'],
                         'min_samples_leaf': [10, 20, 30],
                         'min_samples_split': [760, 800, 840]},
             verbose=2)
```

In [ ]:
```
grid_search.best_estimator_
```

Out[ ]:
```
GradientBoostingClassifier(max_depth=10, max_features='sqrt',
                           min_samples_leaf=10, min_samples_split=760)
```

In [ ]:
```
best_grid=grid_search.best_estimator_
```

In [ ]:
```
print(best_grid)
```

```
GradientBoostingClassifier(max_depth=10, max_features='sqrt',
                           min_samples_leaf=10, min_samples_split=760)
```

In [ ]:
```
y_pred_grid=best_grid.predict(X_test)
```

In [ ]:
```
y_pred_grid=pd.DataFrame(y_pred_grid)
```

## Classification Report

In [ ]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score
```

In [ ]:
```
print(classification_report(Y_test, y_pred_grid))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7566
           1       1.00      1.00      1.00      5318
           2       1.00      1.00      1.00      4932
           3       1.00      1.00      1.00      7184

    accuracy                           1.00     25000
   macro avg       1.00      1.00      1.00     25000
weighted avg       1.00      1.00      1.00     25000
```

In [ ]:
```
print(y_pred_grid.value_counts())
```

```
0    7566
3    7184
1    5318
2    4932
dtype: int64
```

In [ ]:
```
y_pred_grid.rename(columns={0:"Predict_grid"}, inplace=True)
```

In [ ]:
```
y_pred_random = pd.DataFrame(y_pred_random)
print(y_pred_random.value_counts())
```

```
0    7566
3    7184
1    5318
2    4932
dtype: int64
```

In [ ]:
```
y_pred_random.rename(columns={0:"Predict_random"}, inplace=True)
```

In [ ]:
```
print(X_test.shape, y_pred_grid.shape)
```

```
(25000, 8) (25000, 1)
```

In [ ]:
```
y_pred_grid = pd.DataFrame(y_pred_grid)
```

In [ ]:
```
y_pred_grid.value_counts()
```

Out[ ]:
```
Predict_grid
0               7566
3               7184
1               5318
2               4932
dtype: int64
```

In [ ]:
```
y_pred_random.value_counts()
```

Out[ ]:
```
Predict_random
0                 7566
3                 7184
1                 5318
2                 4932
dtype: int64
```

In [ ]:
```
print(confusion_matrix(Y_test, y_pred_grid))
```

```
[[7566    0    0    0]
 [   0 5318    0    0]
 [   0    0 4932    0]
 [   0    0    0 7184]]
```

In [ ]:
```
print(confusion_matrix(Y_test, y_pred_random))
```

```
[[7566    0    0    0]
 [   0 5318    0    0]
 [   0    0 4932    0]
 [   0    0    0 7184]]
```

In [ ]:
```
c = test.Priority.astype('category')
d = dict(enumerate(c.cat.categories))
```

In [ ]:
```
print(d)
```

```
{0: '0 - Unassigned', 1: '1 - Low', 2: '2 - Medium', 3: '3 - High'}
```

In [ ]:
```
y_pred_grid = y_pred_grid["Predict_grid"].map(d)
```

In [ ]:
```
y_pred_grid.value_counts()
```

Out[ ]:
```
0 - Unassigned    7566
3 - High          7184
1 - Low           5318
2 - Medium        4932
Name: Predict_grid, dtype: int64
```

In [ ]:
```
y_pred_random = y_pred_random["Predict_random"].map(d)
```

In [ ]:
```
y_pred_random.value_counts()
```

Out[ ]:
```
0 - Unassigned    7566
3 - High          7184
1 - Low           5318
2 - Medium        4932
Name: Predict_random, dtype: int64
```

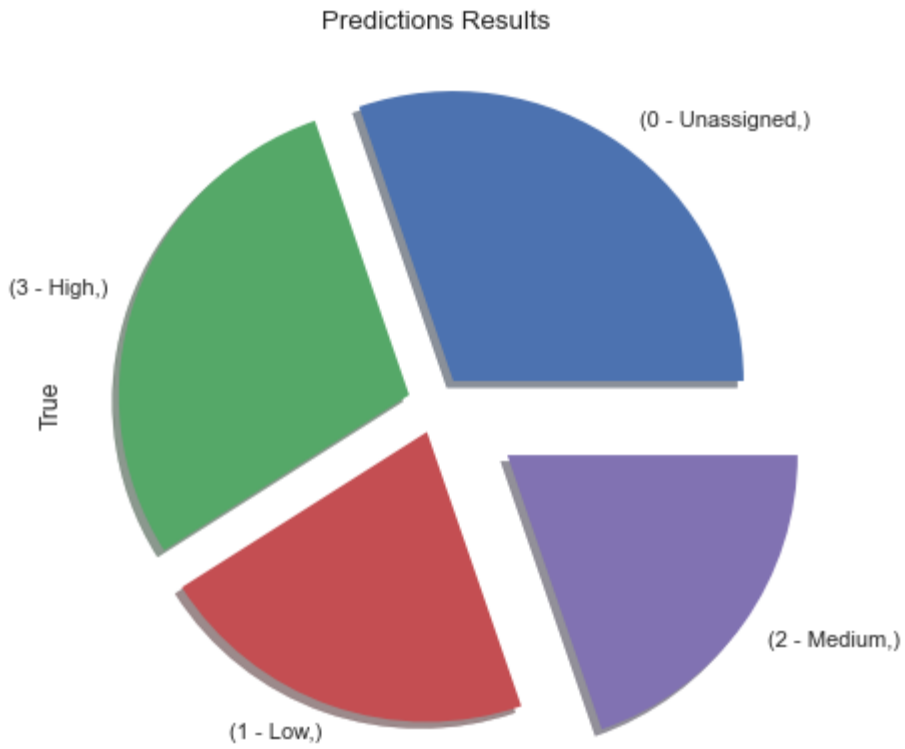
In [ ]:
```
y_pred_random = pd.DataFrame(y_pred_random)
```

In [ ]:
```
y_pred_random.value_counts()
```

Out[ ]:
```
Predict_random
0 - Unassigned    7566
3 - High          7184
1 - Low           5318
2 - Medium        4932
dtype: int64
```

In [ ]:
```
bw = y_pred_random.value_counts()
```

In [ ]:
```
plt.figure(figsize=(10,6), dpi=80, facecolor='white')
explode=(0.1, 0.1, 0.1, 0.3)
bw.plot.pie(shadow=True, explode=explode, label=True)
plt.title("Predictions Results")
plt.show()
```

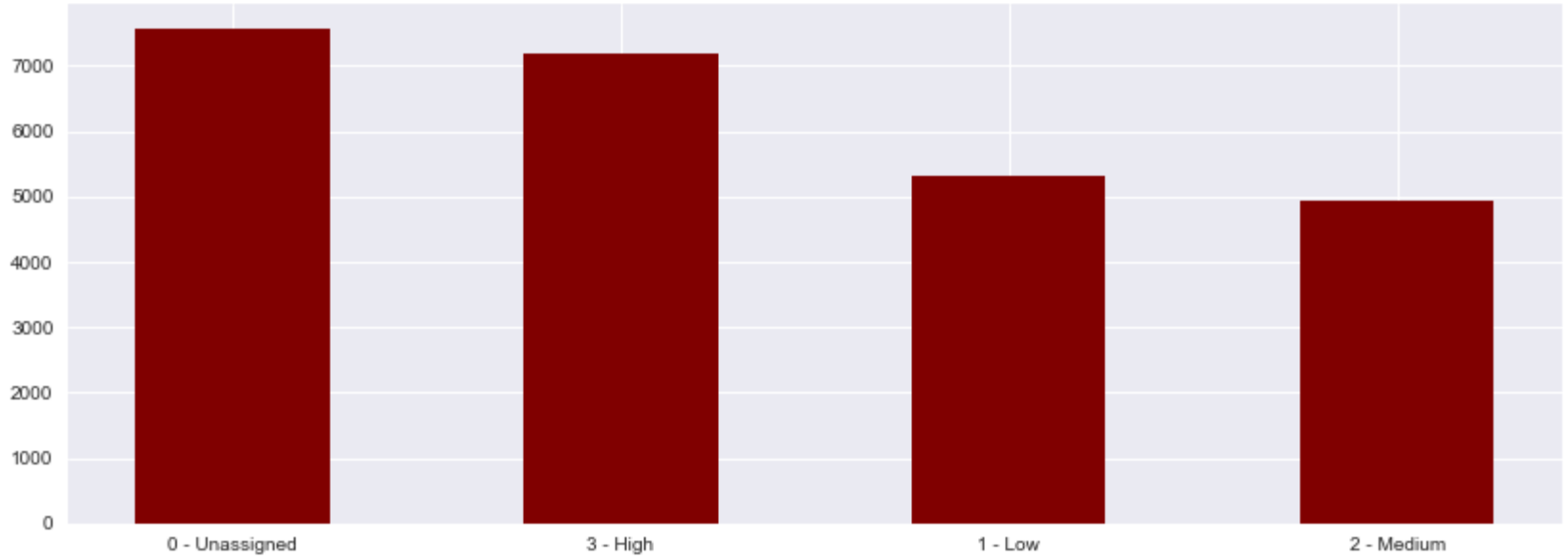



In [ ]:
```
dict = {
'0 - Unassigned' :   7566,
'3 - High'       :   7184,
'1 - Low'        :   5318,
'2 - Medium'     :   4932
}

names = list(dict.keys())
values = list(dict.values())

fig = plt.figure(figsize = (14, 5))

plt.style.use("seaborn")
# creating the bar plot
plt.bar(names, values, color ='maroon', width = 0.5)
plt.show()
```


7000
6000
5000
4000
3000
2000
1000
0
0 - Unassigned
3 - High
1 - Low
2 - Medium

In [ ]: