Exploring 311 dataset for department resource utilization

Modelling

To understand and predict whether additional resources are required, we've approached in two ways

1. Regression
2. Classification

We have some pre-processing techniques involved in these two sections to additionally refine our dataset.

Regression

In Regression, we predict the **Resolution Time** for given 311 complaint record. We can further use rule base approach to identify whether this predicted resolution time would be above the average resolution time of that particular department, thus providing an insight into whether additional resources could be required for this complaint.

Pre-processing

In this section, we apply one hot encoding to convert categorical data into numerical data. Moreover, we also validate our target variable for only values greater than 0. As part of this section, we only retain the top 5 departments for which we want to predict.

To make a comparative study, we applied different regression models

1. Ridge Regression
2. ElasticNet Model
3. Random Forest Regressor
4. Gradient Boosting Regressor
5. XG Boost

In [1]:
```
# Importing all the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
import random
import numpy as np
import matplotlib as mpl
import seaborn as sns
%matplotlib inline
plt.style.use(['fivethirtyeight'])
mpl.rcParams['lines.linewidth'] = 3
```
executed in 791ms, finished 23:02:33 2023-12-11

In [2]:
```
df_reg = pd.read_csv('311_preprocessed_dataset.csv')
```
executed in 235ms, finished 23:02:34 2023-12-11

In [3]:
```
# Removing Unspecified BBL values
df_reg = df_reg[df_reg['BBL'] != 'Unspecified']
df_reg['BBL'] = df_reg['BBL'].astype('float')
```
executed in 35ms, finished 23:02:34 2023-12-11

In [4]:
```
df_reg = pd.get_dummies(df_reg, columns=['Agency','Complaint Type','Location Type','Address Type','City','Community Bo
```
executed in 532ms, finished 23:02:34 2023-12-11

In [5]:
```
#Checking if any rows with zero resolution time
count_zero_resolution_time = (df_reg['Resolution Time'] == 0.0).sum()

print(f"The number of rows with 'Resolution Time' equal to 0.0 is: {count_zero_resolution_time}")
```
executed in 3ms, finished 23:02:34 2023-12-11

```
The number of rows with 'Resolution Time' equal to 0.0 is: 5814
```

In [6]:
```
#Removing those rows with zero resolution time
df_reg = df_reg[df_reg['Resolution Time'] != 0.0]
df_reg = df_reg.reset_index(drop=True)
count_zero_resolution_time = (df_reg['Resolution Time'] == 0.0).sum()
count_zero_resolution_time
```
executed in 651ms, finished 23:02:35 2023-12-11

Out[6]:
```
0
```

In [7]:
```
# Scaling the column value to days by dividing it with 86400 seconds each day
df_reg['Resolution Time'] = df_reg['Resolution Time'] / 86400
```
executed in 2ms, finished 23:02:35 2023-12-11

In [8]:
```
df_reg
```
executed in 27ms, finished 23:02:35 2023-12-11

Out[8]:

	Unnamed: 0	POPULATION	BBL	Latitude	Longitude	Resolution Time	Created_Date_Year	Created_Date_Month	Created_Date_Day	Agency_3-1-1	...	I
0	0	18681.0	4.119740e+09	40.683600	-73.799361	3.302083	2022	2	7	0	...	
1	1	18681.0	4.119900e+09	40.683172	-73.796164	10.042465	2022	2	18	0	...	
2	2	18681.0	4.120550e+09	40.671346	-73.800461	0.048333	2022	2	22	0	...	
3	3	18681.0	4.121070e+09	40.671214	-73.789485	1.463889	2022	2	23	0	...	
4	4	18681.0	4.120010e+09	40.679596	-73.798464	2.904688	2022	3	2	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
223760	243404	28481.0	3.021560e+09	40.707584	-73.966824	0.025764	2021	12	16	0	...	
223761	243405	28481.0	3.023790e+09	40.715262	-73.963177	3.432639	2021	12	20	0	...	
223762	243406	28481.0	3.023170e+09	40.720320	-73.960252	3.224306	2021	12	27	0	...	
223763	243409	28481.0	3.024050e+09	40.713747	-73.963365	2.000000	2022	1	29	0	...	
223764	243410	28481.0	3.021560e+09	40.707548	-73.966514	1.587500	2022	2	6	0	...	

223765 rows × 530 columns

In [9]:
```
#Fetching all the agencies
agency_columns = [col for col in df_reg.columns if col.startswith('Agency_')]

print("Columns starting with 'Agency_':")
print(agency_columns)
```
executed in 2ms, finished 23:02:35 2023-12-11

```
Columns starting with 'Agency_':
['Agency_3-1-1', 'Agency_DCA', 'Agency_DCWP', 'Agency_DEP', 'Agency_DEPARTMENT OF CONSUMER AND WORKER PROTECTION', 'A
gency_DHS', 'Agency_DOB', 'Agency_DOE', 'Agency_DOF', 'Agency_DOHMH', 'Agency_DOITT', 'Agency_DOT', 'Agency_DPR', 'Ag
ency_DSNY', 'Agency_EDC', 'Agency_HPD', 'Agency_NYC311-PRD', 'Agency_NYPD', 'Agency_OTI', 'Agency_TLC']
```

In [10]:
```
#Choosing the top 5 agencies according to data analysis
agency_top_5 = ['Agency_HPD', 'Agency_NYPD', 'Agency_DSNY', 'Agency_DEP', 'Agnecy_DOB']
```
executed in 2ms, finished 23:02:35 2023-12-11

In [11]:
```
# Dropping the unnecessary departments from the dataset
columns_to_choose = [item for item in agency_columns if item not in agency_top_5]
df_reg = df_reg.drop(columns=columns_to_choose)
```
executed in 154ms, finished 23:02:35 2023-12-11

In [12]:
```
columns_to_choose
```
executed in 2ms, finished 23:02:35 2023-12-11

Out[12]:
```
['Agency_3-1-1',
 'Agency_DCA',
 'Agency_DCWP',
 'Agency_DEPARTMENT OF CONSUMER AND WORKER PROTECTION',
 'Agency_DHS',
 'Agency_DOB',
 'Agency_DOE',
 'Agency_DOF',
 'Agency_DOHMH',
 'Agency_DOITT',
 'Agency_DOT',
 'Agency_DPR',
 'Agency_EDC',
 'Agency_NYC311-PRD',
 'Agency_OTI',
 'Agency_TLC']
```

In [13]:
```
#Splitting data into training and testing
from sklearn.model_selection import train_test_split

y = df_reg["Resolution Time"]

X = df_reg.drop('Resolution Time', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 42)
```
executed in 531ms, finished 23:02:35 2023-12-11

In [14]:
```
print(X_train.shape)
print(X_test.shape)
```
executed in 2ms, finished 23:02:35 2023-12-11

```
(179012, 513)
(44753, 513)
```

Ridge Regression

```
In [15]: from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Create a Ridge Regression model
ridge_model = Ridge(alpha=1.0)  # Using this alpha hyperparameter

ridge_model_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('ridge', Ridge(alpha=1.0))
])

# Fit the model to the training data
ridge_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_ridge = ridge_model.predict(X_test)

# Evaluate the model
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f'Ridge Regression - R^2 Score: {r2_ridge}')
```

executed in 2.19s, finished 23:02:38 2023-12-11

```
Ridge Regression - R^2 Score: 0.7072389064515083

/Users/nikhilsoni/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_ridge.py:200: LinAlgWarning: Ill-conditioned matrix (rcond=3.24606e-24): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

ElasticNet Model

```
In [16]: from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the ElasticNet model over chosen hyperparameters
alpha = 0.5  # Regularization strength
l1_ratio = 0.5  # Mix ratio between L1 and L2 regularization
elastic_net = ElasticNet(alpha=alpha, l1_ratio=l1_ratio, random_state=42)
elastic_net.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = elastic_net.predict(X_test_scaled)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print(f'R^2 Score: {r2}')

#print the coefficients
print('Coefficients:', elastic_net.coef_)
```

executed in 6.19s, finished 23:02:44 2023-12-11

```
R^2 Score: 0.6700292281070743
Coefficients: [ 0.00000000e+00 -4.86636887e-01  5.91670297e-01 -0.00000000e+00
  2.09056588e-01  1.80794028e+00  1.14585290e-01  1.09234761e+00
 -2.18075733e+00 -8.00358100e-02  1.81186965e+00 -1.11626946e+00
 -0.00000000e+00  7.05104926e-01 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -1.69528434e-01  1.61906745e-01  0.00000000e+00
 -4.56333141e-01  0.00000000e+00  0.00000000e+00  1.04116075e+00
  0.00000000e+00  4.30594740e-01 -0.00000000e+00 -0.00000000e+00
  1.07043862e+00  1.32470566e+00  0.00000000e+00 -3.03760708e-02
 -8.52930270e-02  0.00000000e+00 -0.00000000e+00  1.17453006e+01
  0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  1.29026605e+00  1.37436440e-01 -0.00000000e+00  9.03442776e-01
  5.59108901e-01 -0.00000000e+00  2.58025699e-02  8.36625607e-01
 -1.85606871e-01  0.00000000e+00  1.41393697e+00  1.98889183e+00
 -9.53943842e-03  7.78148091e-01  3.59653008e+00 -8.51052790e-02
  0.00000000e+00 -1.19810552e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00  2.99966780e-01  0.00000000e+00
```

Random Forest Regressor

In [17]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

# Create and train the Random Forest Regression model with chosen hyperparameters
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)

# Make predictions on the test set
y_pred = random_forest.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)

print(f'R^2 Score: {r2}')
```
executed in 10m 23s, finished 23:13:07 2023-12-11

```
R^2 Score: 0.7230142689610124
```

Gradient Boosting Regression

In [18]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

# Create and train the Gradient Boosting Regression model with chosen hyperparameters
gradient_boosting = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
gradient_boosting.fit(X_train, y_train)

# Make predictions on the test set
y_pred = gradient_boosting.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)
print(f'R^2 Score: {r2}')
```
executed in 1m 13.2s, finished 23:14:20 2023-12-11

```
R^2 Score: 0.7005894202972796
```

XGBoost

In [19]:
```
import xgboost as xgb

# Create and train the XGBoost Regression model with chosen hyperparameters
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 100, random_state=42)

xg_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = xg_reg.predict(X_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)

print(f'R^2 Score: {r2}')
```
executed in 1.67s, finished 23:14:21 2023-12-11

```
R^2 Score: 0.694706224577087
```

Classification

Pre-processing

In this section, we firstly create our target variable and then pre-process the dataset to refine it more. In addition we also apply one hot encoding to convert the categorical values into numeric for modelling.

To make a comparative study, we applied different regression models

1. Decision Tree Classifier
2. Random Forest Classifier
3. Gradient Boosting Classifier
4. KNearest Neighbour Classifier

We chose the above models as they perform better on a combination of categorical and numeric data.

In [20]:
```
import pandas as pd
df_class = pd.read_csv('311_preprocessed_dataset.csv')
```
executed in 235ms, finished 23:14:22 2023-12-11

In [21]:
```python
df_class.dtypes
```
executed in 2ms, finished 23:14:22 2023-12-11

Out[21]:
```
Unnamed: 0                  int64
POPULATION                float64
Agency                     object
Complaint Type             object
Location Type              object
Address Type               object
City                       object
Community Board            object
BBL                        object
Borough                    object
Open Data Channel Type     object
Latitude                  float64
Longitude                 float64
Resolution Time           float64
Created_Date_Year           int64
Created_Date_Month          int64
Created_Date_Day            int64
dtype: object
```

Feature Engineering

Our target variable for classification

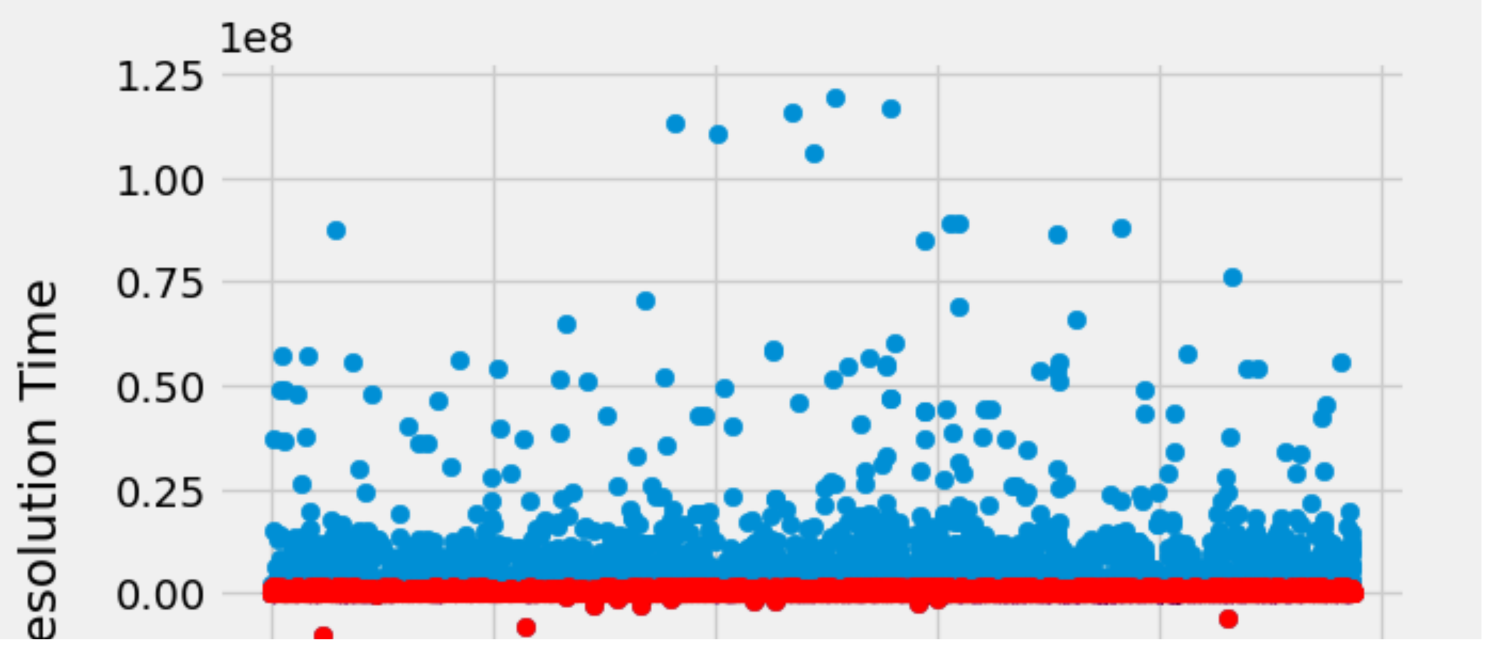
1. For the classification model, we created a new class label *Additional Resources Required*, which was derived from the *Resolution Time* attribute by considering department-wise the average resolution time, and labeled it as 1 if the *Resolution Time* was greater than the average, else we label it as 0.

In [22]:
```python

def CreateAdditionalResourcesRequiredFeature(df):
    """ To create AdditionalResourcesRequired target variable for classification models  """
    df_agency = df.groupby('Agency')['Resolution Time'].mean()

    dict = {}
    for i in range(len(df_agency)):
        dict[df_agency.index[i]] = df_agency[i]
    print (dict)

    df['AdditionalResourcesRequired'] = -1
    for key in dict:
        df['AdditionalResourcesRequired'] = np.where(np.logical_and(df['Agency'] == key, df['Resolution Time'] >= dict
    df['AdditionalResourcesRequired'] = np.where(df['AdditionalResourcesRequired'] != 1, 0, df['AdditionalResourcesReq

    return df

def RemoveResolutionTimeOutliers(df):
    """ To remove Resolution Time outliers """
    departments = list(df['Agency'].unique())
    resolutionTimeMaxList = []
    for department in departments:
        # We remove extreme outliers from the dataset as it could influence our target variable creation
        # We remove the 90th percentile, as once we remove we can see the red plots on the graph which
        # are uniform when compared to blue plots which were before removing the outliers.
        df_temp = df[df['Agency'] == department]
        plt.scatter(df_temp.index, df_temp['Resolution Time'])
        print ("Removing 90th percentile outliers..")
        df_temp = df_temp[df_temp['Resolution Time'] < np.percentile(df_temp['Resolution Time'], 90)]
        plt.scatter(df_temp.index, df_temp['Resolution Time'], color='r')
        plt.xlabel('Unique Key')
        plt.ylabel('Resolution Time')
        resolutionTimeMaxList.append(df_temp['Resolution Time'].max())
        plt.show()

    for i in range(len(departments)):
        df['Resolution Time'] = np.where(np.logical_and(df['Agency'] == departments[i], df['Resolution Time'] > resolu

    print ("Filtering out negative RT")
    df = df[df['Resolution Time'] > 0]

    return df
```
executed in 3ms, finished 23:14:22 2023-12-11

In [23]:
```python
# Filtering the departments out.
df_temp = df_class['Agency'].value_counts()
departments = list(df_temp.head(5).index)
df_class = df_class[df_class['Agency'].isin(departments)]
```
executed in 28ms, finished 23:14:22 2023-12-11

In [24]:
```python
# Removing Resolution Time outliers for classification modelling as we are considering the mean
# for calculating the average resource required per department and the outliers would skew the results.
df_class = RemoveResolutionTimeOutliers(df_class)
```
executed in 511ms, finished 23:14:22 2023-12-11

```
Removing 90th percentile outliers..
```



In [25]:
```python
# Removing Unspecified BBL values
df_class = df_class[df_class['BBL'] != 'Unspecified']
df_class['BBL'] = df_class['BBL'].astype('float')
```
executed in 25ms, finished 23:14:22 2023-12-11

In [26]:
```python
# Creating target variable for classification task
df_class = CreateAdditionalResourcesRequiredFeature(df_class)
```
executed in 37ms, finished 23:14:22 2023-12-11

```
{'DEP': 167439.39450149055, 'DOT': 532499.3357467935, 'DSNY': 283252.9480330257, 'HPD': 668321.2686835218, 'NYPD': 71
47.797402554265}
```

In [27]:
```python
# Removing Resolution Time column for classification dataset
df_class = df_class.drop(columns = ['Resolution Time'])
```
executed in 12ms, finished 23:14:22 2023-12-11

In [28]:
```python
import pandas as pd

# Applying one-hot encoding to the dataset
df_class = pd.get_dummies(df_class, columns=['Agency','Complaint Type','Location Type','Address Type','City','Communit

from sklearn.model_selection import train_test_split
Y = df_class['AdditionalResourcesRequired']
X = df_class.drop(columns = ['AdditionalResourcesRequired'])
df_train_X, df_test_X, df_train_Y, df_test_Y = train_test_split(X, Y, random_state=42, test_size=0.20)
```
executed in 506ms, finished 23:14:23 2023-12-11

Decision Tree Classifier

In [29]:
```python
accuracy_scores = []
# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# Creating the DecisionTreeClassifier instance
clf = DecisionTreeClassifier(criterion='gini', random_state=42, max_depth=13)
clf.fit(df_train_X, df_train_Y)

# Storing the predictions and accuracy scores for later evaluation to avoid later processing
accuracy_scores.append(clf.score(df_test_X, df_test_Y))
clf_pred = clf.predict(df_test_X)
```
executed in 1.93s, finished 23:14:25 2023-12-11

Random Forest Classifier

In [30]:
```python
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# Creating the RandomForestClassifier instance to train the model
rfc = RandomForestClassifier(max_depth=13, random_state=42, max_features=None)
rfc.fit(df_train_X, df_train_Y)

# Storing the predictions and accuracy scores for later evaluation to avoid later processing
accuracy_scores.append(rfc.score(df_test_X, df_test_Y))
rfc_pred = rfc.predict(df_test_X)
```
executed in 1m 53.7s, finished 23:16:19 2023-12-11

Gradient Boosting Classifier

In [31]:
```
# Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier

# Creating the GradientBoostingClassifier instance to train the model
gbc = GradientBoostingClassifier(n_estimators=1000, learning_rate=0.5, max_depth=1, random_state=42)
gbc.fit(df_train_X, df_train_Y)

# Storing the predictions and accuracy scores for later evaluation to avoid later processing
accuracy_scores.append(gbc.score(df_test_X, df_test_Y))
gbc_pred = gbc.predict(df_test_X)
```
executed in 2m 37s, finished 23:18:55 2023-12-11

KNN Classifier

In [32]:
```
df_train_X = df_train_X.values
df_test_X = df_test_X.values
```
executed in 80ms, finished 23:18:55 2023-12-11

In [33]:
```
# KNN classifier
from sklearn.neighbors import KNeighborsClassifier

import numpy as np

# Creating the KNNClassifier instance to train the model
knc = KNeighborsClassifier(n_neighbors=15)
knc.fit(df_train_X, df_train_Y)

# Storing the predictions and accuracy scores for later evaluation to avoid later processing
accuracy_scores.append(knc.score(df_test_X, df_test_Y))
knc_pred = knc.predict(df_test_X)
```
executed in 26.3s, finished 23:19:21 2023-12-11

Evaluation Metrics

To evaluate our models, we perform a comprehensive analysis using different evaluation metrics. Here are the following metrics used

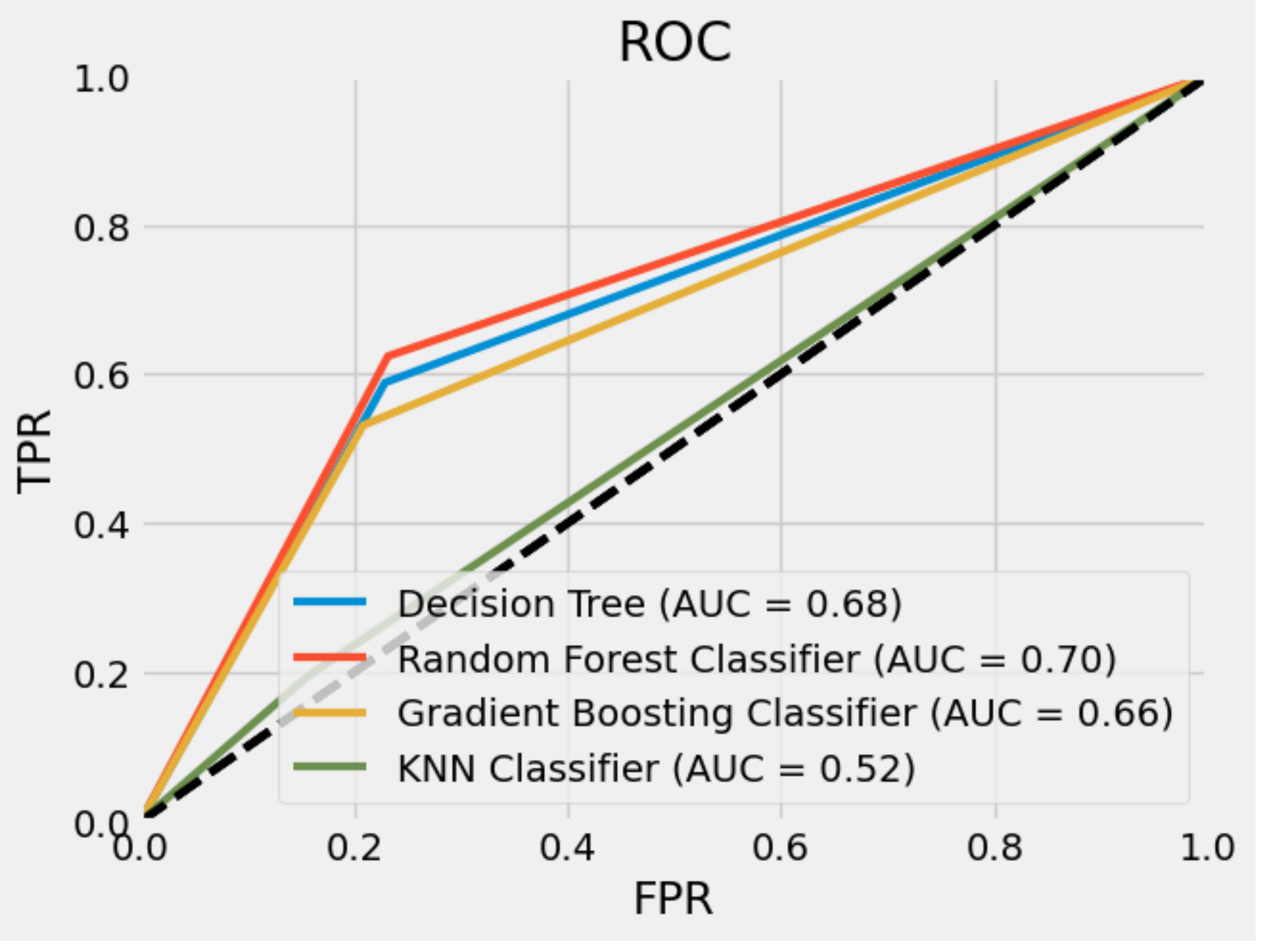
1. ROC AUC Curve
2. F1 Score
3. Accuracy
4. Precision
5. Recall
6. Confusion Matrix

With all the above metrics, we understand whether our model is performing on par expectations and what must be done to improve our modelling.

ROC AUC Curve

In [34]:
```
from sklearn import metrics
import matplotlib.pyplot as plt
def roc_curve(ytrue, ypred, model_name):
    """ To plot the AUC-ROC curve for all the models """
    fpr, tpr, thresholds = metrics.roc_curve(ytrue, ypred)
    auc = metrics.roc_auc_score(ytrue, ypred)
    plt.plot(fpr, tpr, label= model_name+' (AUC = %0.2f)' % auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.title('ROC')
    plt.legend(loc="lower right")

models = ['Decision Tree','Random Forest Classifier','Gradient Boosting Classifier','KNN Classifier']
def print_scores(table_scores):
    for i in range(len(models)):
        print (models[i] + " : %0.2f" % table_scores[i])
```
executed in 3ms, finished 23:19:21 2023-12-11

In [35]:
```
# Plotting ROC AUC curves for all the models
roc_curve(df_test_Y, clf_pred, 'Decision Tree')
roc_curve(df_test_Y, rfc_pred, 'Random Forest Classifier')
roc_curve(df_test_Y, gbc_pred, 'Gradient Boosting Classifier')
roc_curve(df_test_Y, knc_pred, 'KNN Classifier')
```
executed in 98ms, finished 23:19:22 2023-12-11



F1 Score

In [36]:
```
from sklearn.metrics import f1_score
def F1_Score(ytrue, ypred):
    return f1_score(ytrue, ypred)
```
executed in 2ms, finished 23:19:22 2023-12-11

In [37]:
```
# Printing the F1 scores for all the models
f1_table = []
f1_table.append(F1_Score(df_test_Y, clf_pred))
f1_table.append(F1_Score(df_test_Y, rfc_pred))
f1_table.append(F1_Score(df_test_Y, gbc_pred))
f1_table.append(F1_Score(df_test_Y, knc_pred))

print_scores(f1_table)
```
executed in 28ms, finished 23:19:22 2023-12-11

```
Decision Tree : 0.59
Random Forest Classifier : 0.61
Gradient Boosting Classifier : 0.56
KNN Classifier : 0.26
```

Accuracy

In [38]:
```
# Printing the accuracy scores for all the models
print_scores(accuracy_scores)
```
executed in 2ms, finished 23:19:22 2023-12-11

```
Decision Tree : 0.71
Random Forest Classifier : 0.72
Gradient Boosting Classifier : 0.70
KNN Classifier : 0.61
```

Precision

In [39]:
```
from sklearn.metrics import precision_score
def Precision(ytrue, ypred):
    return precision_score(ytrue, ypred)

# Printing the Precision scores for all the models
precision_table = []
precision_table.append(Precision(df_test_Y, clf_pred))
precision_table.append(Precision(df_test_Y, rfc_pred))
precision_table.append(Precision(df_test_Y, gbc_pred))
precision_table.append(Precision(df_test_Y, knc_pred))

print_scores(precision_table)
```
executed in 28ms, finished 23:19:22 2023-12-11

```
Decision Tree : 0.59
Random Forest Classifier : 0.60
Gradient Boosting Classifier : 0.59
KNN Classifier : 0.41
```

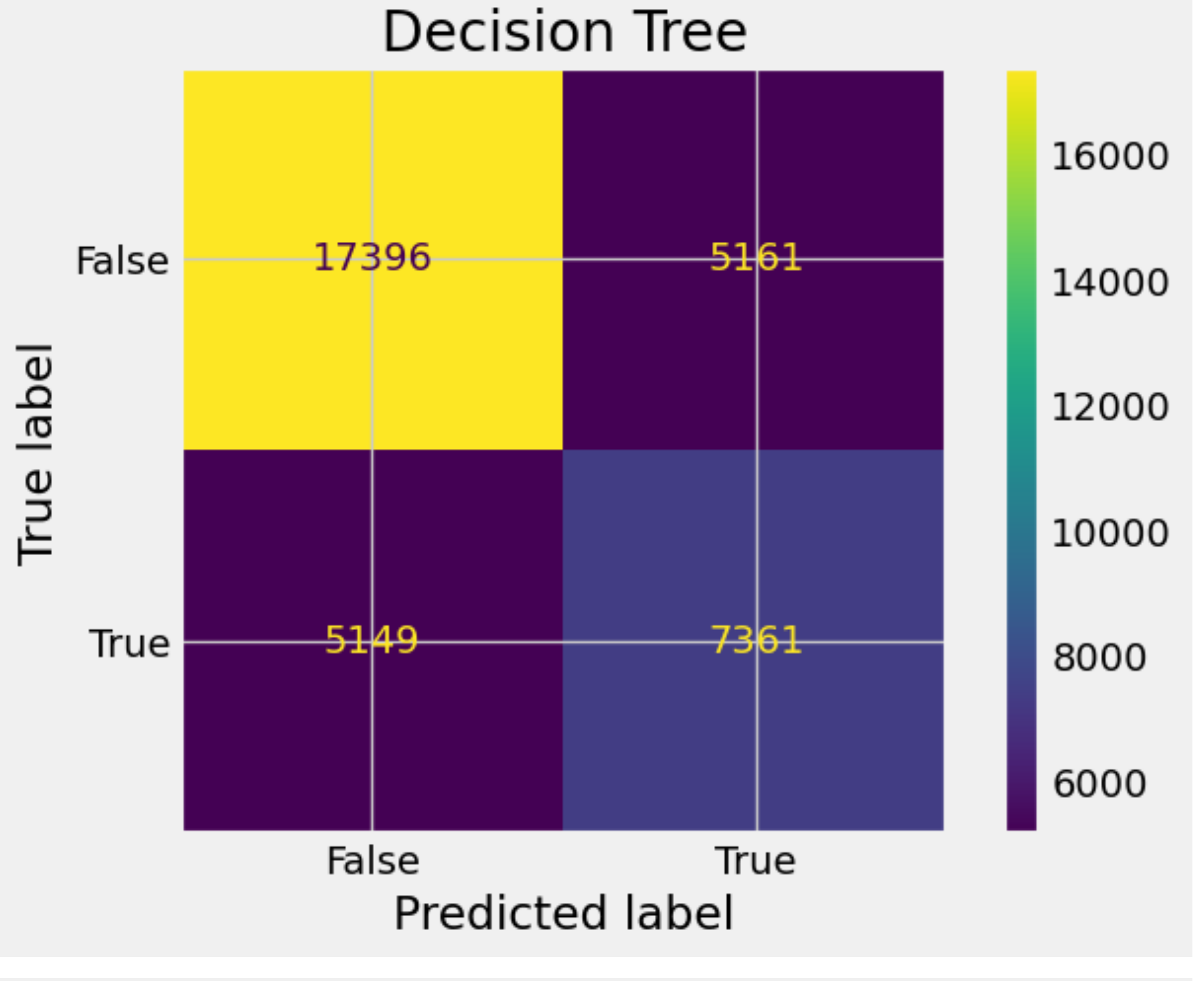
Recall

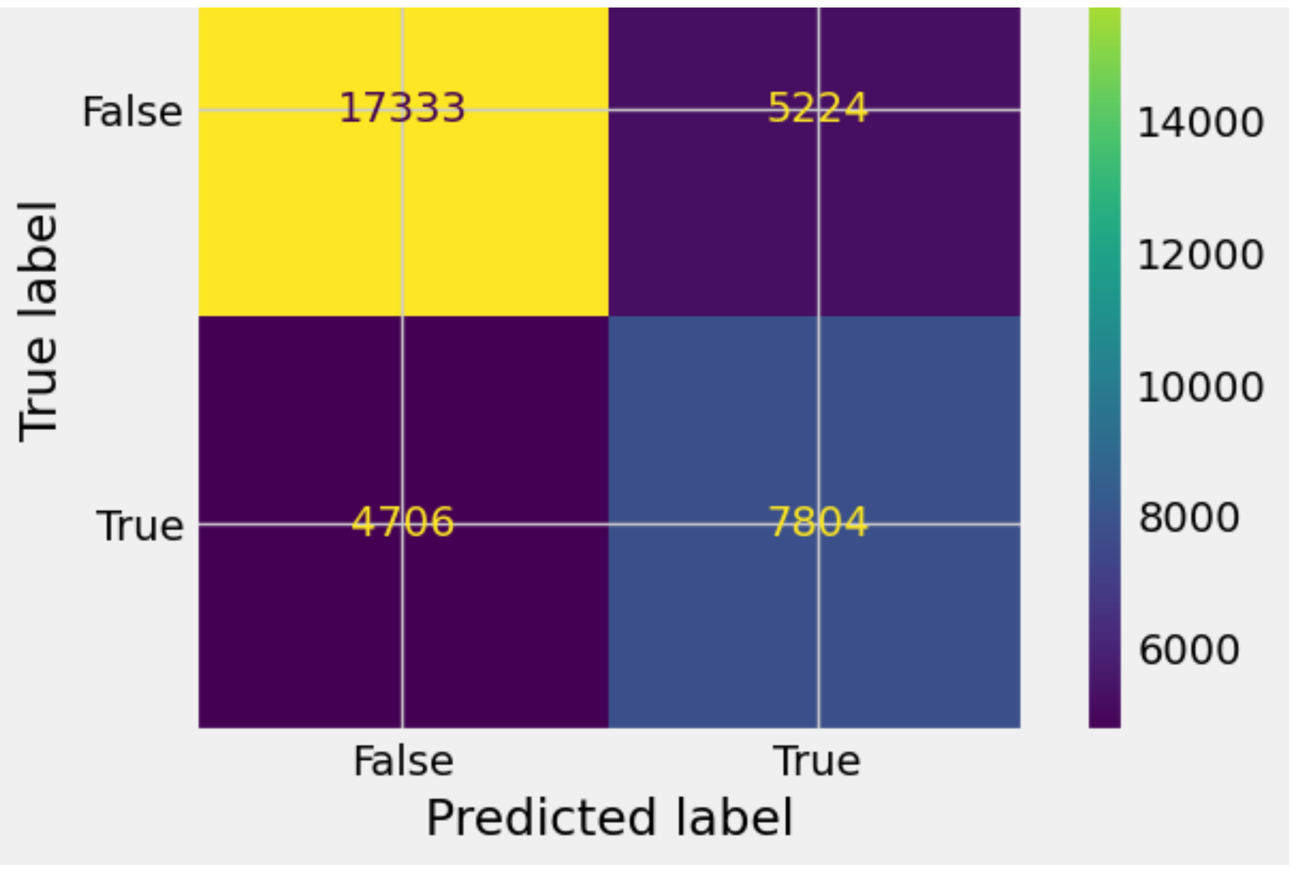
In [40]:
```
from sklearn.metrics import recall_score
def Recall(ytrue, ypred):
    return recall_score(ytrue, ypred)

# Printing the Recall scores for all the models
recall_table = []
recall_table.append(Recall(df_test_Y, clf_pred))
recall_table.append(Recall(df_test_Y, rfc_pred))
recall_table.append(Recall(df_test_Y, gbc_pred))
recall_table.append(Recall(df_test_Y, knc_pred))

print_scores(recall_table)
```
executed in 27ms, finished 23:19:22 2023-12-11

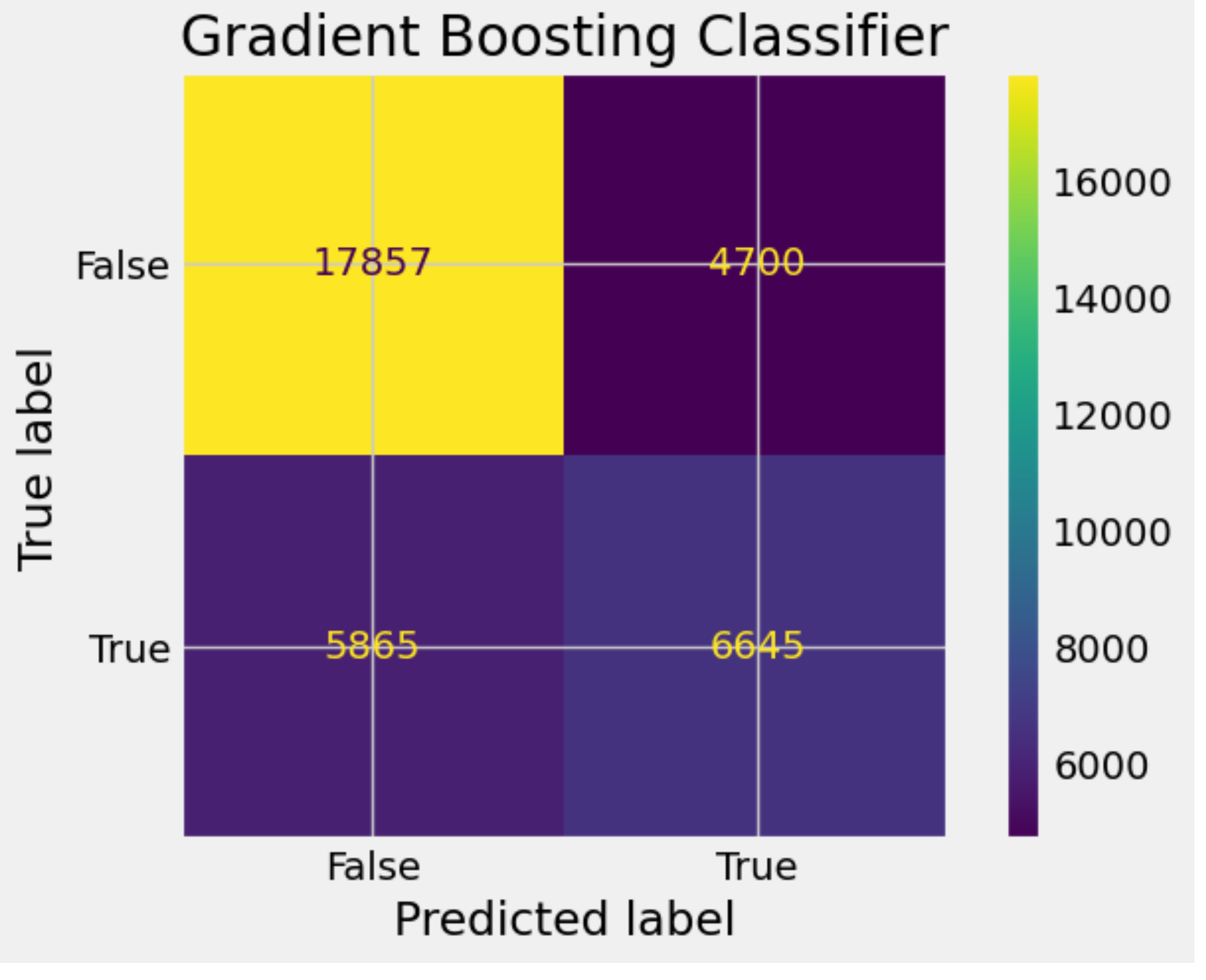
```
Decision Tree : 0.59
Random Forest Classifier : 0.62
Gradient Boosting Classifier : 0.53
KNN Classifier : 0.19
```

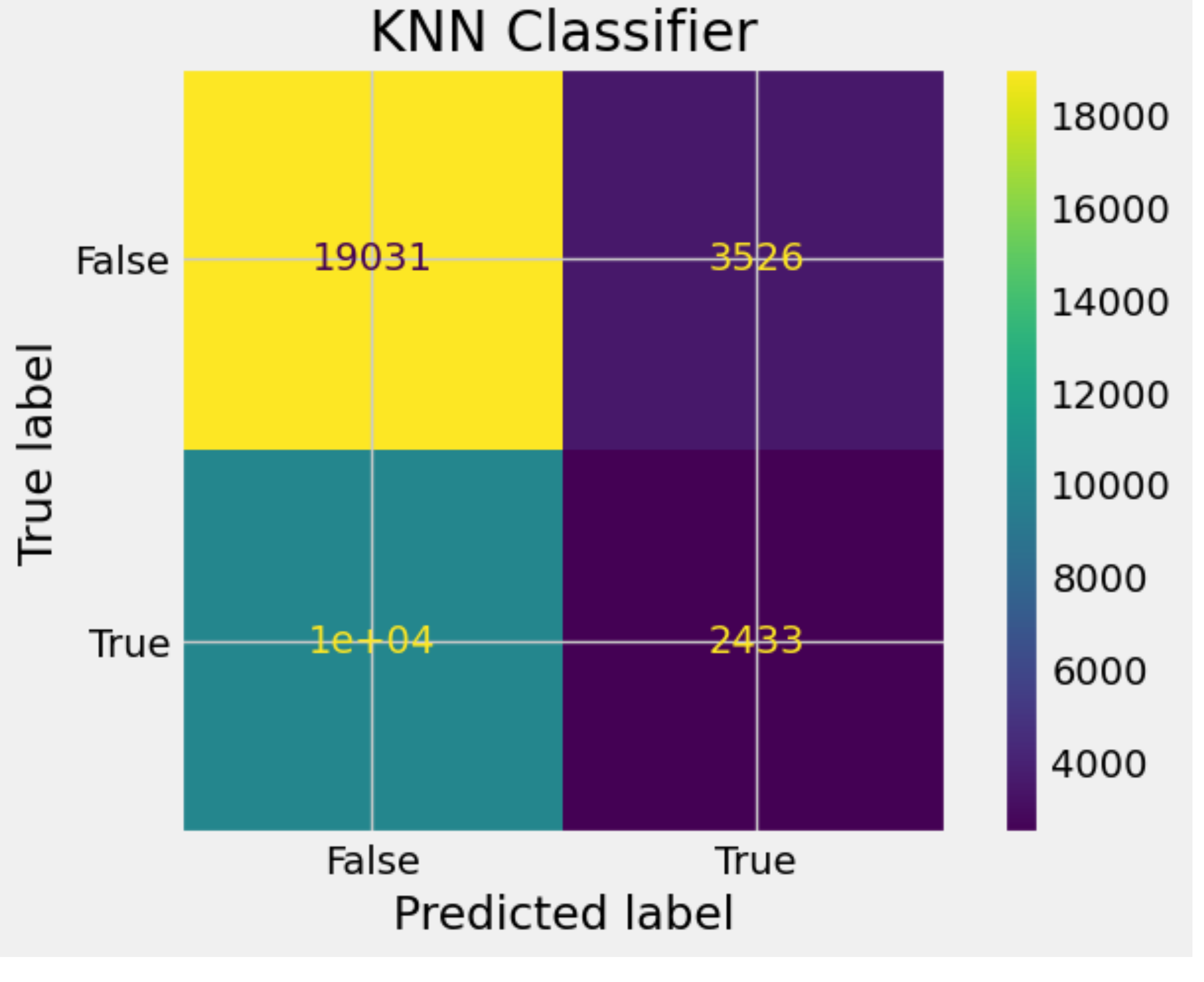
Confusion Matrix

In [41]:
```
from sklearn import metrics

preds = [clf_pred, rfc_pred, gbc_pred, knc_pred]

# Plotting the confusion matrix for all the models
# to understand the bump on the graph.
for i in range(len(preds)):
    confusion_matrix = metrics.confusion_matrix(df_test_Y, preds[i])

    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])

    cm_display.plot()
    plt.title(models[i])
    plt.show()
```
executed in 214ms, finished 23:19:22 2023-12-11




True label
False
True
17333
5224
4706
7804
False
True
Predicted label
14000
12000
10000
8000
6000


Gradient Boosting Classifier
True label
False
True
17857
4700
5865
6645
False
True
Predicted label
16000
14000
12000
10000
8000
6000


KNN Classifier
True label
False
True
19031
3526
1e+04
2433
False
True
Predicted label
18000
16000
14000
12000
10000
8000
6000
4000