```
# import the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, f1_score, precision_score, recall_score,
roc_curve, accuracy_score,auc
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import resample
import warnings
warnings.filterwarnings("ignore")
 #to find the headers
df = pd.read_csv("carclaims.csv")
df.head()
print("Number of Features Available:",df.shape[1])
print("Number of Samples Available :",df.shape[0])
df.isnull().sum()
#to plot the number of frauds found
plt.figure(figsize=(10,8))
bars = plt.bar(df.FraudFound.value_counts().index, df.FraudFound.value_counts().values)
plt.title("Fraud Type")
plt.xlabel("Type")
plt.ylabel("Count")
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.05, yval, ha='center', va='bottom')
plt.show()
```

```
# Replace the Labels to 0 and 1
df.loc[df['FraudFound'] == 'No','FraudFound'] = 0
df.loc[df['FraudFound'] == 'Yes','FraudFound'] = 1
df['FraudFound'] = df['FraudFound'].astype(int)

#Plotting the graph separately showing frauds found in each Car Makers
make = df.groupby('Make')['FraudFound'].sum().sort_values(ascending=False)
plt.figure(figsize=(20,8))
plt.title("Car Make Vs Frauds")
cmap = plt.get_cmap('Blues')
colors = [cmap(i) for i in np.linspace(0, 1, len(make.index))]
ax = sns.barplot(y=make.index, x=make.values, orient='h', palette=colors[::-1])
for i, v in enumerate(make.values):
    ax.text(v + 0.2, i + .25, str(format(int(v), ',d')), color='black', fontweight='light')
plt.xlabel("Number of Fraud")
plt.ylabel("Car Make")
plt.show()
# Plotting the number of claims found in each Car Makers.
make_count = df['Make'].value_counts().sort_values(ascending=False)
plt.figure(figsize=(20,8))
plt.title("Car Make Count")
cmap = plt.get_cmap('Blues')
colors = [cmap(i) for i in np.linspace(0, 1, len(make_count.index))]
ax = sns.barplot(y=make_count.index, x=make_count.values, orient='h', palette=colors[::-1])
for i, v in enumerate(make_count.values):
    ax.text(v + 0.2, i + .25, str(format(int(v), ',d')), color='black', fontweight='light')
plt.ylabel("Car Make")
plt.xlabel("Count of Cars")
plt.show()
```

```
# Plotting the number of frauds found in each Policy Holder's Age
policyAge = df.groupby('AgeOfPolicyHolder')['FraudFound'].sum().sort_values(ascending=True)
plt.figure(figsize=(20,8))
plt.title("Policy Holder's Age Vs Frauds")
cmap = plt.get_cmap('Blues')
colors = [cmap(i) for i in np.linspace(0, 1, len(policyAge.index))]
ax = sns.barplot(x=policyAge.index, y=policyAge.values, palette=colors)
for p in ax.patches:
   ax.annotate(str(format(int(p.get_height()), ',d')), (p.get_x()+0.24, p.get_height()*1.01))
plt.xlabel("Policy Holder's Age")
plt.ylabel("Number of Fraud")
plt.show()

#Plotting the graph separately showing frauds found in each gender
gender = df.groupby('Sex')['FraudFound'].sum()
plt.figure(figsize=(10,8))
plt.title("Gender Vs Frauds")
ax = sns.barplot(x=gender.index, y=gender.values)
for p in ax.patches:
   ax.annotate(str(format(int(p.get_height()), ',d')), (p.get_x()+0.24, p.get_height()*1.01))
plt.xlabel("Gender")
plt.ylabel("Number of Fraud")
plt.show()

#Plotting the graph separately showing frauds found in each Area
accidentArea = df.groupby('AccidentArea')['FraudFound'].sum()
plt.figure(figsize=(10,8))
plt.title("AccidentArea Vs Frauds")
plt.pie(accidentArea.values,labels=accidentArea.index,  autopct='%.0f%%')
plt.show()
```

```
#Plotting the pie chart separately showing frauds found in Type of Faults
fault = df.groupby('Fault')['FraudFound'].sum()
plt.figure(figsize=(10,8))
plt.title("Fault Vs Frauds")
plt.pie(fault.values,labels=fault.index,  autopct='%.0f%%')
plt.show()
#Plotting the graph separately showing frauds found in NumberofCars involved
cars = df.groupby('NumberOfCars')['FraudFound'].sum()
plt.figure(figsize=(20,8))
plt.title("Cars Involved Vs Frauds")
ax = sns.barplot(x=cars.index,y=cars.values)
for p in ax.patches:
    ax.annotate(str(format(int(p.get_height()), ',d')), (p.get_x()+0.4, p.get_height()*1.01))
plt.xlabel("Cars Involved")
plt.ylabel("Number of Fraud");
plt.yticks([])
plt.show()
#Plotting the graph separately showing frauds found in MaritalStatus
fraud = df[df['FraudFound'] == 1]
plt.figure(figsize=(10,5))
plt.title("Marital Status Vs Frauds")
sns.countplot(x=fraud['MaritalStatus']);
plt.xlabel("Marital Status")
plt.ylabel("Number of Fraud");
#Preprocessing
le = LabelEncoder()
cols = df.select_dtypes('O').columns
df[cols]= df[cols].apply(le.fit_transform)
df['Year'] = le.fit_transform(df.Year)
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True,linewidth=0.5,fmt="0.2f")
```

```
df_new = df[['Make', 'AccidentArea','Sex',
        'MaritalStatus','Fault', 'VehicleCategory',\
        'VehiclePrice', 'Year',\
        'DriverRating', 'Days:Policy-Accident', 'Days:Policy-Claim',\
        'PastNumberOfClaims', 'AgeOfVehicle', 'AgeOfPolicyHolder',\
        'PoliceReportFiled', 'WitnessPresent', 'AgentType',\
        'NumberOfSuppliments', 'AddressChange-Claim', 'NumberOfCars',\
        'BasePolicy', 'FraudFound']]
plt.figure(figsize=(20,20))
sns.heatmap(df_new.corr(),annot=True,linewidth=0.5,fmt="0.2f")

#Split Data
X = df_new.drop(['FraudFound'], axis=1)
y = df_new['FraudFound']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
def conf_matrix(y_test,y_pred):
    con_matrix = confusion_matrix(y_test,y_pred)
    con_matrix = pd.DataFrame(con_matrix,range(2),range(2))
    plt.figure(figsize=(5,5))
    plt.title("Confusion Matrix")
    sns.heatmap(con_matrix,annot=True,cbar=False,fmt='g')
#Data Modeling
lr = LogisticRegression()
lr.fit(X_train,y_train)
lr_pred = lr.predict(X_test)
print('Accuracy of LogisticRegression model:',accuracy_score(y_test, lr_pred))
print('F1 Score for LogisticRegression model:', f1_score(y_test, lr_pred))
print('Precision for LogisticRegression model:', precision_score(y_test, lr_pred))
print('Recall for LogisticRegression model:', recall_score(y_test, lr_pred))



fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
```

```
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for LogisticRegression:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for LogisticRegression')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test,lr_pred)
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
rfc_pred = rfc.predict(X_test)
acc_rfc = accuracy_score(y_test, rfc_pred)
print('Accuracy of RandomForestClassifier model:',accuracy_score(y_test, rfc_pred))
print('F1 Score for RandomForestClassifier model:', f1_score(y_test, rfc_pred))
print('Precision for RandomForestClassifier model:', precision_score(y_test, rfc_pred))
print('Recall for RandomForestClassifier model:', recall_score(y_test, rfc_pred))
fpr, tpr, thresholds = roc_curve(y_test, rfc_pred)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for LogisticRegression:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for RandomForestClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test,rfc_pred)
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
knn_pred = rfc.predict(X_test)
acc_knn = accuracy_score(y_test, knn_pred)
print('Accuracy of KNeighborsClassifier model:',accuracy_score(y_test, knn_pred))
print('F1 Score for KNeighborsClassifier model:', f1_score(y_test, knn_pred))
print('Precision for KNeighborsClassifier model:', precision_score(y_test, knn_pred))
print('Recall for KNeighborsClassifier model:', recall_score(y_test, knn_pred))
fpr, tpr, thresholds = roc_curve(y_test, knn_pred)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for KNeighborsClassifier:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for KNeighborsClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test,knn_pred)

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train,y_train)
dt_pred = rfc.predict(X_test)
acc_dt = accuracy_score(y_test, dt_pred)
print('Accuracy of DecisionTreeClassifier model:',accuracy_score(y_test, dt_pred))
print('F1 Score for DecisionTreeClassifier model:', f1_score(y_test, dt_pred))
print('Precision for DecisionTreeClassifier model:', precision_score(y_test, dt_pred))
print('Recall for DecisionTreeClassifier model:', recall_score(y_test, dt_pred))
```

```
# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, dt_pred)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for DecisionTreeClassifier:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for DecisionTreeClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test,dt_pred)
#UpSampling
mn = df_new.FraudFound.value_counts()[0]
df_majority = df_new[df_new.FraudFound==0]
df_minority = df_new[df_new.FraudFound==1]
df_minority_upsampled = resample(df_minority,replace=True,n_samples = mn,random_state=42)
df_upsampled = pd.concat([df_majority,df_minority_upsampled])
df_upsampled.FraudFound.value_counts()
X_up = df_upsampled.drop('FraudFound',axis=1)
y_up = df_upsampled[['FraudFound']]
X_train_up,X_test_up,y_train_up,y_test_up = train_test_split(X_up,y_up,stratify=y_up)
X_train_up.shape, X_test_up.shape, y_train_up.shape, y_test_up.shape
# Upsampled Logistic Regression model
lr.fit(X_train_up,y_train_up)
lr_pred_up = lr.predict(X_test_up)
acc_lr_up = accuracy_score(y_test_up, lr_pred_up)
f1_lr_up = f1_score(y_test_up, lr_pred_up)
pre_lr_up = precision_score(y_test_up, lr_pred_up)
```

```
recall_lr_up = recall_score(y_test_up, lr_pred_up)
print("Accuracy of Upsampled LogisticRegression model:",acc_lr_up)
print(f'F1 Score for Upsampled LogisticRegression model:', f1_lr_up)
print(f'Precision for Upsampled LogisticRegression model:', pre_lr_up)
print(f'Recall for Upsampled LogisticRegression model:', recall_lr_up)
fpr, tpr, thresholds = roc_curve(y_test_up, lr_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled LogisticRegression:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled LogisticRegression')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_up,lr_pred_up)
# Upsampled RandomForestClassifier model
rfc.fit(X_train_up,y_train_up)
rfc_pred_up = rfc.predict(X_test_up)
acc_rfc_up = accuracy_score(y_test_up, rfc_pred_up)
f1_rfc_up = f1_score(y_test_up, rfc_pred_up)
pre_rfc_up = precision_score(y_test_up, rfc_pred_up)
recall_rfc_up = recall_score(y_test_up, rfc_pred_up)
print("Accuracy of Upsampled RandomForestClassifier model:",acc_rfc_up)
print(f'F1 Score for Upsampled RandomForestClassifier model:', f1_rfc_up)
print(f'Precision for Upsampled RandomForestClassifier model:', pre_rfc_up)
print(f'Recall for Upsampled RandomForestClassifier model:', recall_rfc_up)
fpr, tpr, thresholds = roc_curve(y_test_up, rfc_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled RandomForestClassifier:', roc_auc)
```

```
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled RandomForestClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_up,rfc_pred_up)
# Upsampled KNeighborsClassifier model
knn.fit(X_train_up,y_train_up)
knn_pred_up = knn.predict(X_test_up)
acc_knn_up = accuracy_score(y_test_up, knn_pred_up)
f1_knn_up = f1_score(y_test_up, knn_pred_up)
pre_knn_up = precision_score(y_test_up, knn_pred_up)
recall_knn_up = recall_score(y_test_up, knn_pred_up)
print("Accuracy of Upsampled KNeighborsClassifier model:",acc_knn_up)
print(f'F1 Score for Upsampled KNeighborsClassifier model:', f1_knn_up)
print(f'Precision for Upsampled KNeighborsClassifier model:', pre_knn_up)
print(f'Recall for Upsampled KNeighborsClassifier model:', recall_knn_up)
fpr, tpr, thresholds = roc_curve(y_test_up, knn_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled KNeighborsClassifier:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled KNeighborsClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_up,knn_pred_up)
# Upsampled DecisionTreeClassifier model
dt.fit(X_train_up,y_train_up)
dt_pred_up = dt.predict(X_test_up)
acc_dt_up = accuracy_score(y_test_up, dt_pred_up)
f1_dt_up = f1_score(y_test_up, dt_pred_up)
pre_dt_up = precision_score(y_test_up, dt_pred_up)
recall_dt_up = recall_score(y_test_up, dt_pred_up)
print("Accuracy of Upsampled DecisionTreeClassifier model:", acc_dt_up)
print(f'F1 Score for Upsampled DecisionTreeClassifier model:', f1_dt_up)
print(f'Precision for Upsampled DecisionTreeClassifier model:', pre_dt_up )
print(f'Recall for Upsampled DecisionTreeClassifier model:', recall_dt_up)
fpr, tpr, thresholds = roc_curve(y_test_up, dt_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled DecisionTreeClassifier:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled DecisionTreeClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_up,dt_pred_up)
#DownSampling
mj = df_new.FraudFound.value_counts()[1]
df_majority = df_new[df_new.FraudFound==0]
df_minority = df_new[df_new.FraudFound==1]
df_majority_downsampled = resample(df_majority,replace=False,n_samples = mj,random_state=42)
```

```
df_downsampled = pd.concat([df_minority,df_majority_downsampled])
df_downsampled.FraudFound.value_counts()
X_dwn = df_downsampled.drop('FraudFound',axis=1)
y_dwn = df_downsampled[['FraudFound']]
X_train_dwn,X_test_dwn,y_train_dwn,y_test_dwn = train_test_split(X_dwn,y_dwn,stratify=y_dwn)
X_train_dwn.shape, X_test_dwn.shape, y_train_dwn.shape, y_test_dwn.shape

# DownSampled Logistic Regression model
lr.fit(X_train_dwn,y_train_dwn)
lr_pred_dwn = lr.predict(X_test_dwn)
acc_lr_dwn = accuracy_score(y_test_dwn, lr_pred_dwn)
f1_lr_dwn = f1_score(y_test_dwn, lr_pred_dwn)
pre_lr_dwn = precision_score(y_test_dwn, lr_pred_dwn)
recall_lr_dwn = recall_score(y_test_dwn, lr_pred_dwn)
print('Accuracy of Downsampled LogisticRegression model:', acc_lr_dwn)
print('F1 Score for Downsampled LogisticRegression model:', f1_lr_dwn)
print('Precision for Downsampled LogisticRegression model:', pre_lr_dwn)
print('Recall for Downsampled LogisticRegression model:', recall_lr_dwn)
# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test_dwn, lr_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled LogisticRegression:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled LogisticRegression')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_dwn,lr_pred_dwn)
```

```
# DownSampled RandomForestClassifier model
rfc.fit(X_train_dwn,y_train_dwn)
rfc_pred_dwn = rfc.predict(X_test_dwn)
acc_rfc_dwn = accuracy_score(y_test_dwn, rfc_pred_dwn)
f1_rfc_dwn = f1_score(y_test_dwn, rfc_pred_dwn)
pre_rfc_dwn = precision_score(y_test_dwn, rfc_pred_dwn)
recall_rfc_dwn = recall_score(y_test_dwn, rfc_pred_dwn)
print('Accuracy of Downsampled RandomForestClassifier model:', acc_rfc_dwn)
print('F1 Score for Downsampled RandomForestClassifier model:', f1_rfc_dwn)
print('Precision for Downsampled RandomForestClassifier model:', pre_rfc_dwn)
print('Recall for Downsampled RandomForestClassifier model:', recall_rfc_dwn )
fpr, tpr, thresholds = roc_curve(y_test_dwn, rfc_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled RandomForestClassifier:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled RandomForestClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_dwn,rfc_pred_dwn)
```

```
# DownSampled KNeighborsClassifier model
knn.fit(X_train_dwn,y_train_dwn)
knn_pred_dwn = knn.predict(X_test_dwn)
acc_knn_dwn = accuracy_score(y_test_dwn, knn_pred_dwn)
f1_knn_dwn = f1_score(y_test_dwn, knn_pred_dwn)
pre_knn_dwn = precision_score(y_test_dwn, knn_pred_dwn)
```

```
recall_knn_dwn = recall_score(y_test_dwn, knn_pred_dwn)
print('Accuracy of Downsampled KNeighborsClassifier model:', acc_knn_dwn)
print('F1 Score for Downsampled KNeighborsClassifier model:', f1_knn_dwn)
print('Precision for Downsampled KNeighborsClassifier model:', pre_knn_dwn)
print('Recall for Downsampled KNeighborsClassifier model:', recall_knn_dwn)
fpr, tpr, thresholds = roc_curve(y_test_dwn, knn_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled KNeighborsClassifier:', roc_auc)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled KNeighborsClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_dwn,knn_pred_dwn)
# DownSampled DecisionTreeClassifier model
dt.fit(X_train_dwn,y_train_dwn)
dt_pred_dwn = dt.predict(X_test_dwn)
acc_dt_dwn = accuracy_score(y_test_dwn, dt_pred_dwn)
f1_dt_dwn =f1_score(y_test_dwn, dt_pred_dwn)
pre_dt_dwn = precision_score(y_test_dwn, dt_pred_dwn)
recall_dt_dwn = recall_score(y_test_dwn, dt_pred_dwn)
print('Accuracy of Downsampled DecisionTreeClassifier model:', acc_dt_dwn)
print('F1 Score for Downsampled DecisionTreeClassifier model:', f1_dt_dwn)
print('Precision for Downsampled DecisionTreeClassifier model:', pre_dt_dwn)
print('Recall for Downsampled DecisionTreeClassifier model:', recall_dt_dwn)
fpr, tpr, thresholds = roc_curve(y_test_dwn, dt_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled DecisionTreeClassifier:', roc_auc)
```

```
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled DecisionTreeClassifier')
plt.legend(loc="lower right")
plt.show()
conf_matrix(y_test_dwn,dt_pred_dwn)
#Evaluation Results
data = {'Model':['UpSampled Logistic Regression','DownSampled Logistic Regression',
        'UpSampled RandomForest Classifier','DownSampled RandomForest Classifier',
        'UpSampled KNeighborsClassifier', 'DownSampled KNeighborsClassifier',
        'UpSampled DecisionTreeClassifier','DownSampled DecisionTreeClassifier'],
    'Accuracy Score':[acc_lr_up, acc_lr_dwn, acc_rfc_up, acc_rfc_dwn, acc_knn_up, acc_knn_dwn,
            acc_dt_up, acc_dt_dwn],
    'F1_Score':[f1_lr_up, f1_lr_dwn, f1_rfc_up, f1_rfc_dwn, f1_knn_up, f1_knn_dwn,
            f1_dt_up, f1_dt_dwn],
    'Precision':[pre_lr_up, pre_lr_dwn, pre_rfc_up, pre_rfc_dwn, pre_knn_up, pre_knn_dwn,
            pre_dt_up, pre_dt_dwn],
    'Recall_Score':[recall_lr_up, recall_lr_dwn, recall_rfc_up, recall_rfc_dwn, recall_knn_up, recall_knn_dwn,
            recall_dt_up, recall_dt_dwn]}
comparision_table=pd.DataFrame(data)
comparision_table
```