DATA SCIENCE MAJOR PROECT

> Contents

** Question**

- **Problem statement**: Create a classification model to predict whether a person makes over $50k a year
- **Context**: This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry
- **Dataset** : https://drive.google.com/file/d/193ND4XKmMSnqdOlbDGP36b5V5s6HcQJb/view?usp=sharing

Importing all the necessary Libraries

[ ] ↳ *1 cell hidden*

Handling with the data, rearranging the features, handling with null values and outliers

[ ] ↳ *16 cells hidden*

HANDLING WITH OUTLIERS

↳ *24 cells hidden*

Handling with missing values

[ ] ↳ *5 cells hidden*

HANDLING WITH CATEGORICAL DATA

[ ] ↳ *8 cells hidden*

SPLITTING DATA AS TRAIN AND TEST

```
df_new.head()
```

	age	Fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	sex_ Male	income_ >50K	workclass_ Federal-gov	w
0	39	77516	13	2174	0	40	1	0	0	
1	50	83311	13	0	0	13	1	0	0	

```
df_new["income_ >50K"].value_counts()
```

```
0    24586
1     7812
Name: income_ >50K, dtype: int64
```

```
X=df_new.drop(columns=["income_ >50K"],axis=1)
X["education_num"]=X["education_num"].astype("int")
```

```
y=df_new["income_ >50K"]
```

```
X.head()
```

	age	Fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	sex_ Male	workclass_ Federal-gov	workclass_ Local-gov
4	28	338409	13	0	0	40	0	0	0

```
X_new=X.copy()
y_new=y.copy()
```

WITH-OUT BALANCING THE DATA

```
Counter(y)
```

```
Counter({0: 24586, 1: 7812})
```

```
X_train, X_test, y_train, y_test_no = train_test_split(X, y)
```

Double-click (or enter) to edit

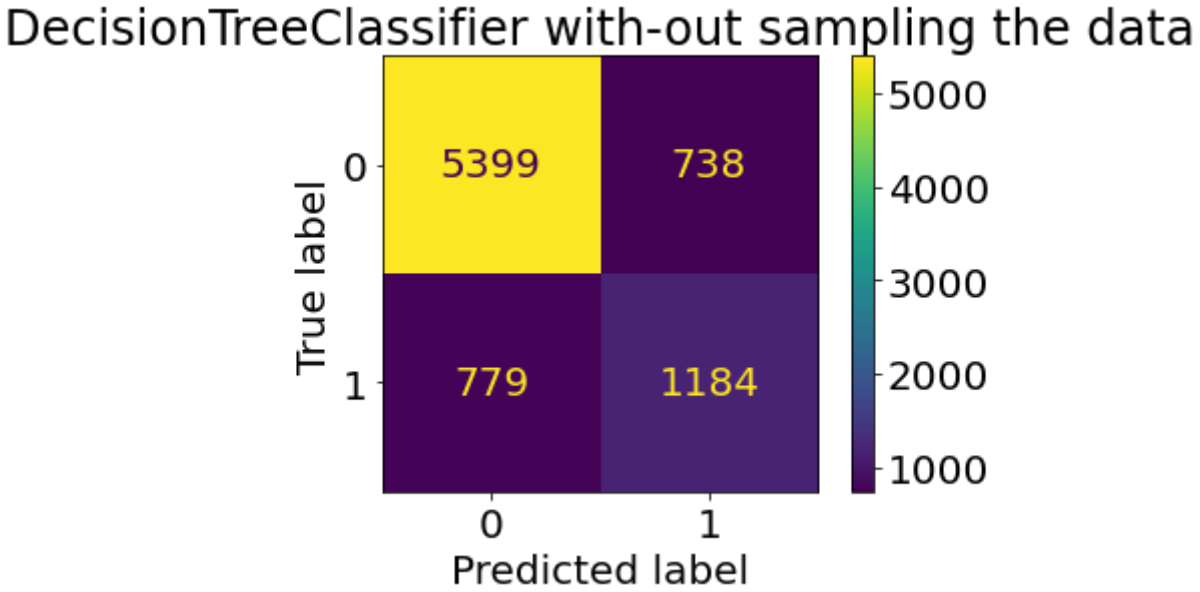
```
print("X_train row number :",X_train.shape[0])
print("X_test row number :",X_test.shape[0])
print("y_train row number :",y_train.shape[0])
print("y_test row number :",y_test_no.shape[0])
```

```
X_train row number : 24298
X_test row number : 8100
y_train row number : 24298
y_test row number : 8100
```

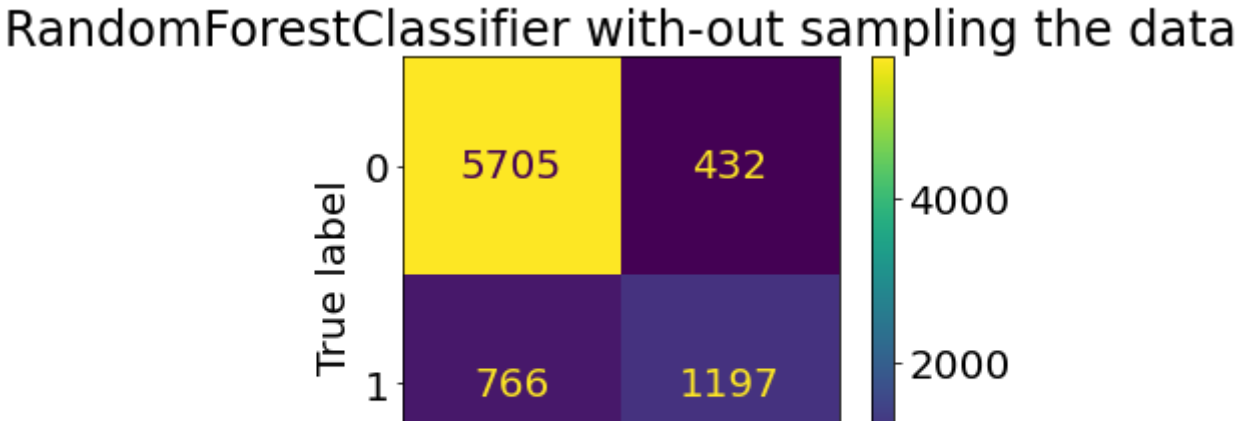
```
clf = DecisionTreeClassifier()
clf.fit(X_train,y_train)
```

```
y_pred_D_no=clf.predict(X_test)
cm=confusion_matrix(y_test_no,y_pred_D_no)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('DecisionTreeClassifier with-out sampling the data')
plt.show()

print(classification_report(y_test_no,y_pred_D_no))
print()
print()
clf=RandomForestClassifier()
clf.fit(X_train,y_train)
y_pred_R_no=clf.predict(X_test)
cm=confusion_matrix(y_test_no,y_pred_R_no)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('RandomForestClassifier with-out sampling the data')
plt.show()

print(classification_report(y_test_no,y_pred_R_no))
print()
print()
clf=LogisticRegression()
clf.fit(X_train,y_train)
y_pred_L_no=clf.predict(X_test)
cm=confusion_matrix(y_test_no,y_pred_L_no)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('LogisticRegression with-out sampling the data')
plt.show()

print(classification_report(y_test_no,y_pred_L_no))
print()
print()
clf=KNeighborsClassifier()
clf.fit(X_train,y_train)
y_pred_K_no=clf.predict(X_test)
```

```
cm=confusion_matrix(y_test_no,y_pred_K_no)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('KNeighborsClassifier with-out sampling the data')
plt.show()

print(classification_report(y_test_no,y_pred_K_no))
print()
print()
clf=SVC()
clf.fit(X_train,y_train)
y_pred_S_no=clf.predict(X_test)
cm=confusion_matrix(y_test_no,y_pred_S_no)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('SVC with-out sampling the data')
plt.show()

print(classification_report(y_test_no,y_pred_S_no))
```

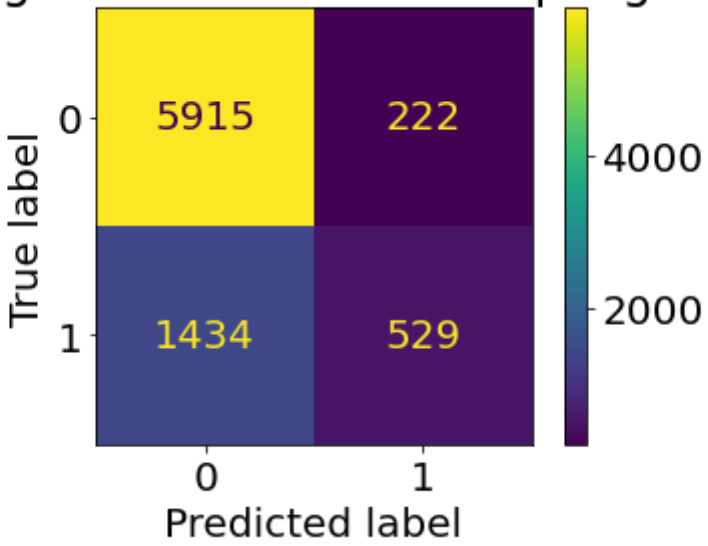


	precision	recall	f1-score	support
0	0.87	0.88	0.88	6137
1	0.62	0.60	0.61	1963
accuracy			0.81	8100
macro avg	0.74	0.74	0.74	8100
weighted avg	0.81	0.81	0.81	8100



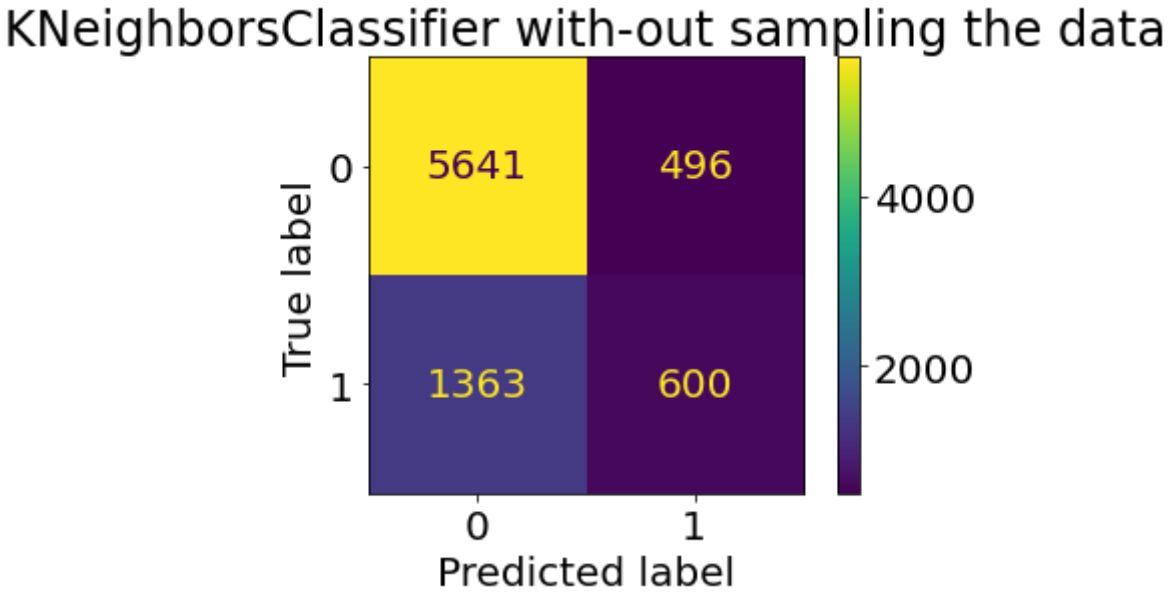
	precision	recall	f1-score	support
0	0.88	0.93	0.90	6137
1	0.73	0.61	0.67	1963
accuracy			0.85	8100
macro avg	0.81	0.77	0.79	8100
weighted avg	0.85	0.85	0.85	8100

LogisticRegression with-out sampling the data

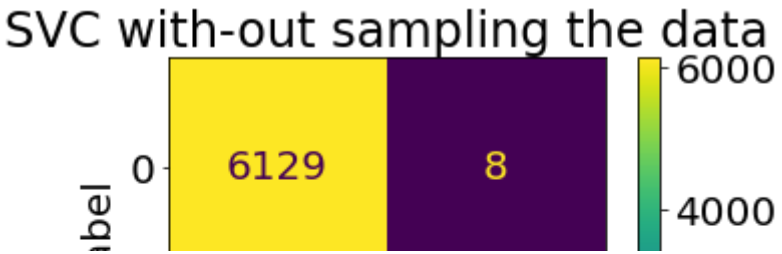


	precision	recall	f1-score	support
0	0.80	0.96	0.88	6137
1	0.70	0.27	0.39	1963
accuracy			0.80	8100
macro avg	0.75	0.62	0.63	8100

```
weighted avg       0.78      0.80      0.76      8100
```



	precision	recall	f1-score	support
0	0.81	0.92	0.86	6137
1	0.55	0.31	0.39	1963
accuracy			0.77	8100
macro avg	0.68	0.61	0.63	8100
weighted avg	0.74	0.77	0.75	8100



UNDERSAMPLING THE DATA

```
random_majority_indices=np.random.choice(df_new[df_new["income_ >50K"]==0].index,
                                         len(df_new[df_new["income_ >50K"]==1]),
                                         replace=False)


minority_class_indices=df_new[df_new["income_ >50K"]==1].index
print(minority_class_indices)
```

```
     Int64Index([    7,     8,     9,    10,    11,    14,    19,    20,    25,
                    27,
                 ...
                 32530, 32532, 32533, 32536, 32538, 32539, 32545, 32554, 32557,
                 32560],
                dtype='int64', length=7812)
```

```
under_sample_indices=np.concatenate([minority_class_indices,random_majority_indices])


under_sample=df_new.loc[under_sample_indices]


X=under_sample.drop(columns=["income_ >50K"],axis=1)
X["education_num"]=X["education_num"].astype("int")
y=under_sample["income_ >50K"]


sns.countplot(x="income_ >50K",data=under_sample)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa86deb6e10>
```


```
X_train, X_test, y_train, y_test_u = train_test_split(X, y)
Counter(y)
```

```
Counter({0: 7812, 1: 7812})
```

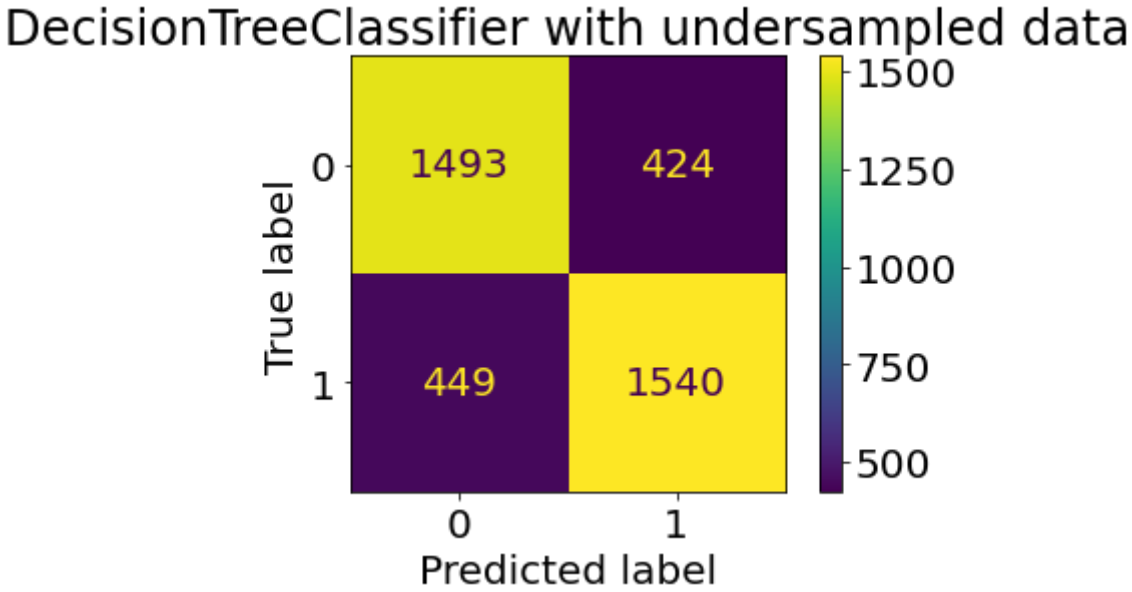
```
print("X_train row number :",X_train.shape[0])
print("X_test row number :",X_test.shape[0])
print("y_train row number :",y_train.shape[0])
print("y_test row number :",y_test_u.shape[0])
```

```
X_train row number : 11718
X_test row number : 3906
y_train row number : 11718
y_test row number : 3906
```

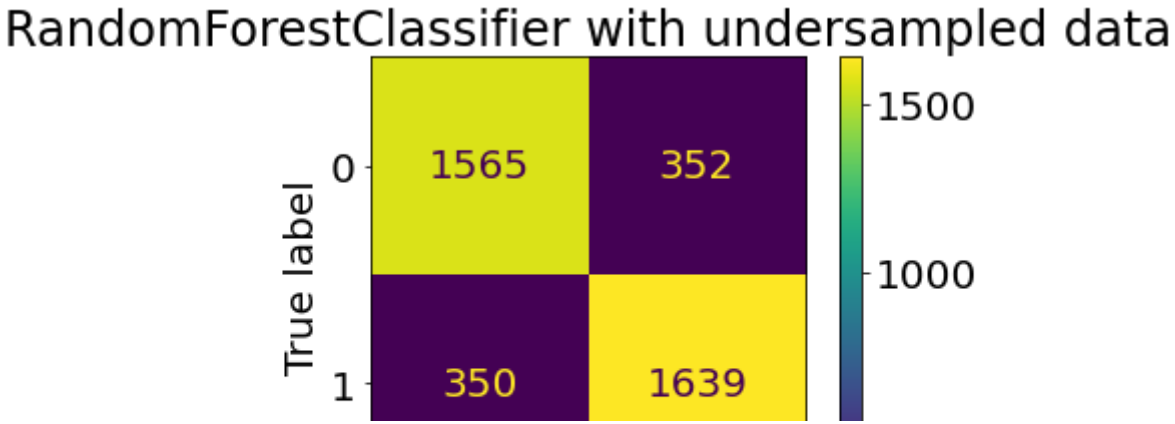
```
clf = DecisionTreeClassifier()
clf.fit(X_train,y_train)
y_pred_D_u=clf.predict(X_test)
cm=confusion_matrix(y_test_u,y_pred_D_u)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('DecisionTreeClassifier with undersampled data')
plt.show()
```

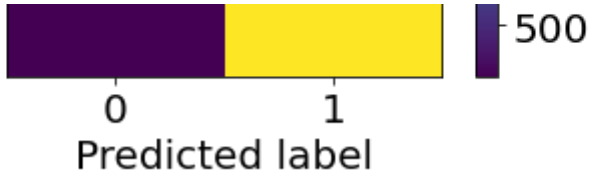
```
print(classification_report(y_test_u,y_pred_D_u))

print()
print()

clf=RandomForestClassifier()
clf.fit(X_train,y_train)
y_pred_R_u=clf.predict(X_test)
cm=confusion_matrix(y_test_u,y_pred_R_u)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('RandomForestClassifier with undersampled data')
plt.show()

print(classification_report(y_test_u,y_pred_R_u))

print()
print()

clf=LogisticRegression(max_iter=5000)
clf.fit(X_train,y_train)
y_pred_L_u=clf.predict(X_test)
cm=confusion_matrix(y_test_u,y_pred_L_u)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('LogisticRegression with undersampled data')
plt.show()

print(classification_report(y_test_u,y_pred_L_u))
print()
print()

clf=KNeighborsClassifier()
clf.fit(X_train,y_train)
y_pred_K_u=clf.predict(X_test)
cm=confusion_matrix(y_test_u,y_pred_K_u)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
```

```
disp.plot()
plt.title('KNeighborsClassifier with undersampled data')
plt.show()

print(classification_report(y_test_u,y_pred_K_u))
print()
print()

clf=SVC()
clf.fit(X_train,y_train)
y_pred_S_u=clf.predict(X_test)
cm=confusion_matrix(y_test_u,y_pred_S_u)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('SVC with undersampled data')
plt.show()

print(classification_report(y_test_u,y_pred_S_u))
```

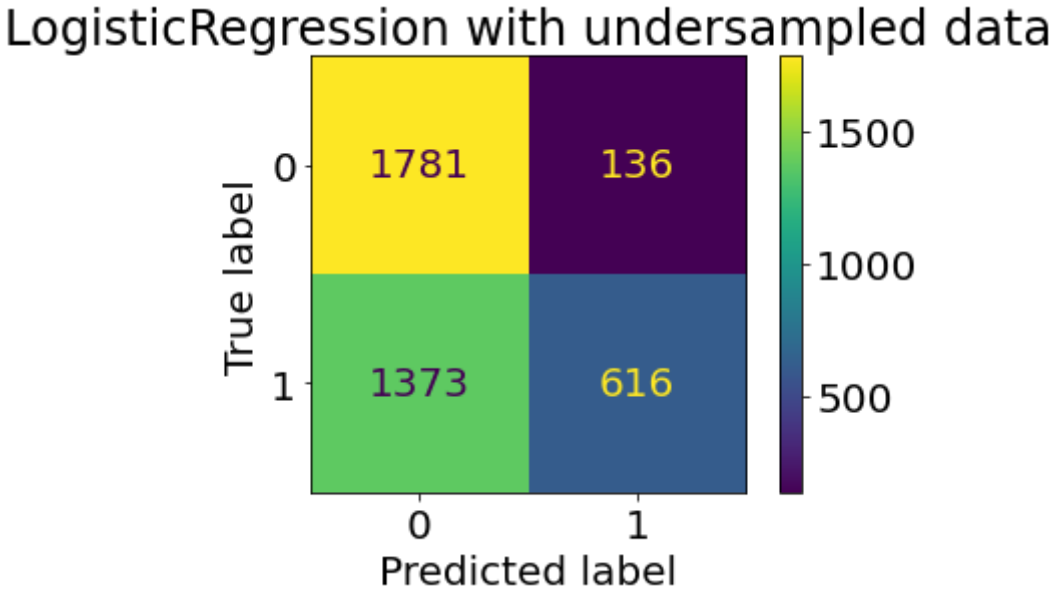


	precision	recall	f1-score	support
0	0.77	0.78	0.77	1917
1	0.78	0.77	0.78	1989
accuracy			0.78	3906
macro avg	0.78	0.78	0.78	3906
weighted avg	0.78	0.78	0.78	3906



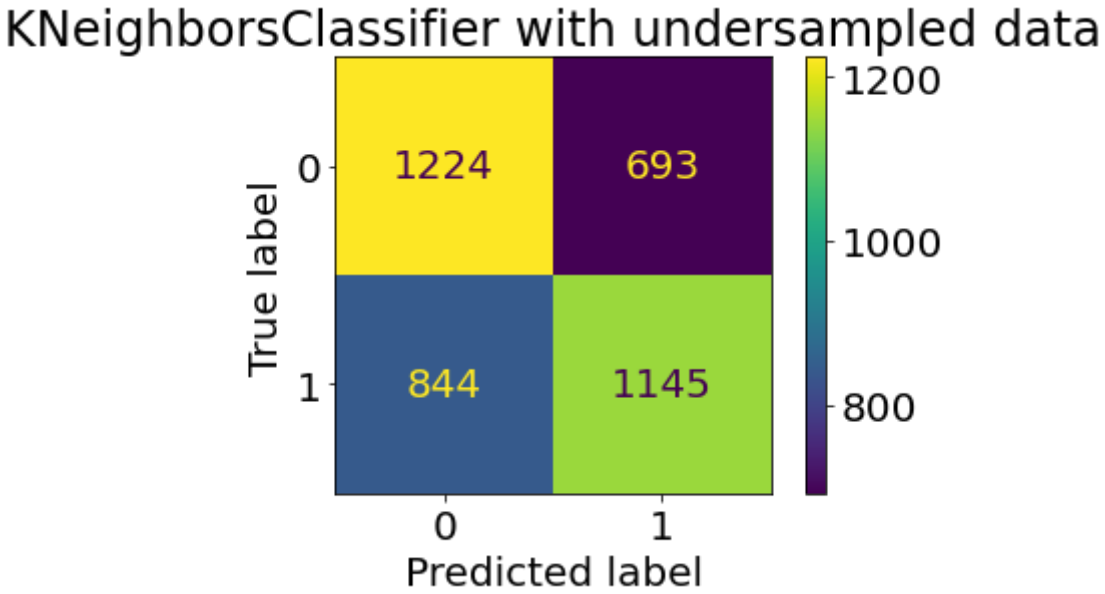


	precision	recall	f1-score	support
0	0.82	0.82	0.82	1917
1	0.82	0.82	0.82	1989
accuracy			0.82	3906
macro avg	0.82	0.82	0.82	3906
weighted avg	0.82	0.82	0.82	3906

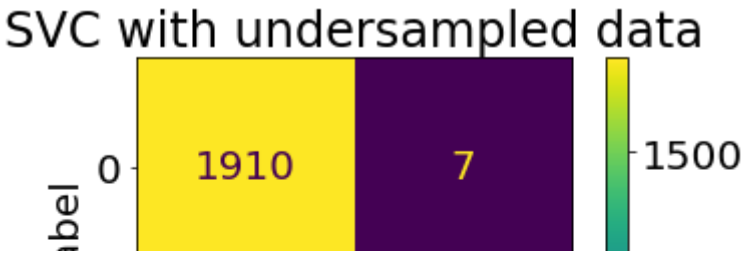


	precision	recall	f1-score	support
0	0.56	0.93	0.70	1917
1	0.82	0.31	0.45	1989
accuracy			0.61	3906
macro avg	0.69	0.62	0.58	3906

```
weighted avg       0.69      0.61      0.57      3906
```



	precision	recall	f1-score	support
0	0.59	0.64	0.61	1917
1	0.62	0.58	0.60	1989
accuracy			0.61	3906
macro avg	0.61	0.61	0.61	3906
weighted avg	0.61	0.61	0.61	3906



OVERSAMPLING THE DATA

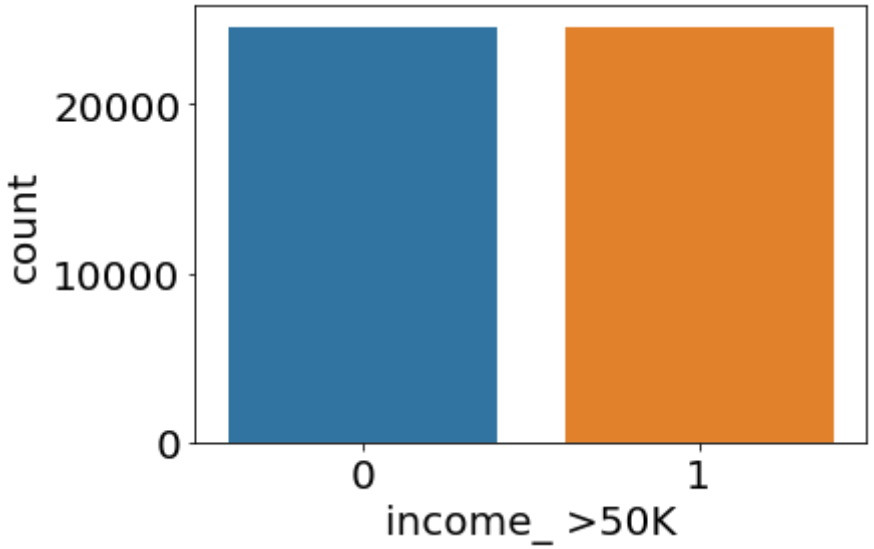
```
sm = SMOTE(random_state=42)
X_new, y_new = sm.fit_resample(X_new, y_new)
Counter(y_new)
```

```
Counter({0: 24586, 1: 24586})
```

```
1       0.98      0.16      0.28      1989
```

```
sns.countplot(y_new)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following var
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fa86db7e650>
```




```
X_train, X_test, y_train, y_test_o = train_test_split(X_new, y_new)
print("X_train row number :",X_train.shape[0])
```

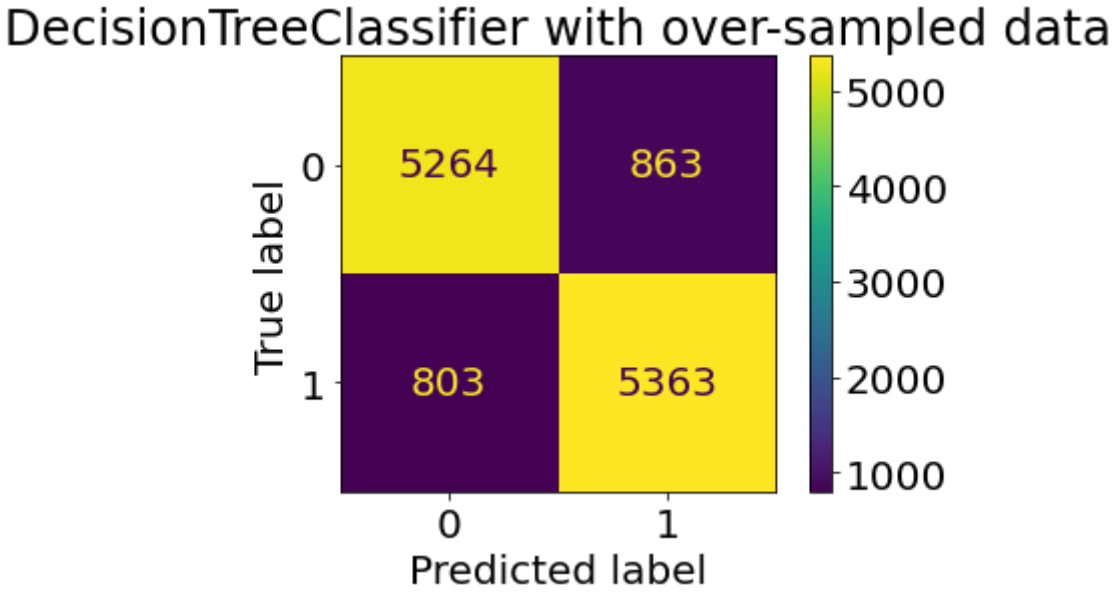
```
print("X_test row number :",X_test.shape[0])
print("y_train row number :",y_train.shape[0])
print("y_test row number :",y_test_o.shape[0])
```

```
X_train row number : 36879
X_test row number : 12293
y_train row number : 36879
y_test row number : 12293
```

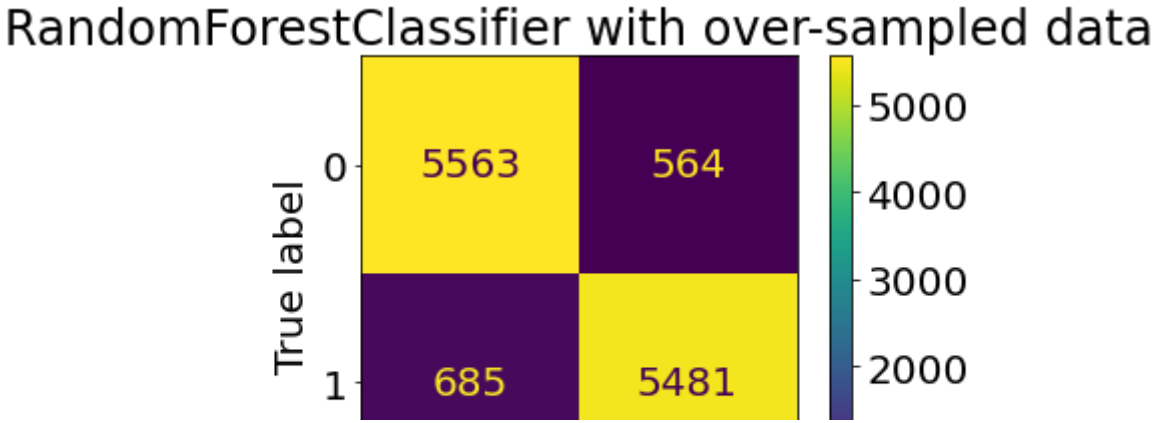
```
clf = DecisionTreeClassifier()
clf.fit(X_train,y_train)
y_pred_D_o=clf.predict(X_test)
cm=confusion_matrix(y_test_o,y_pred_D_o)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('DecisionTreeClassifier with over-sampled data')
plt.show()
print(classification_report(y_test_o,y_pred_D_o))
print()
print()
```

```
clf=RandomForestClassifier()
clf.fit(X_train,y_train)
y_pred_R_o=clf.predict(X_test)
cm=confusion_matrix(y_test_o,y_pred_R_o)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('RandomForestClassifier with over-sampled data')
plt.show()

print(classification_report(y_test_o,y_pred_R_o))

print()
print()
```

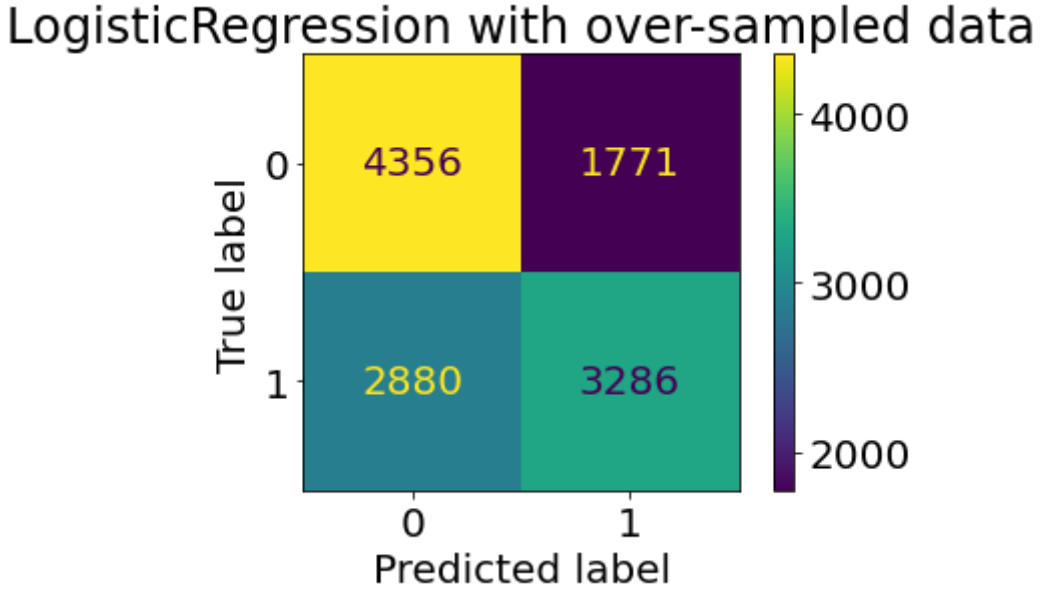
```
clf=LogisticRegression(max_iter=5000)
clf.fit(X_train,y_train)
y_pred_L_o=clf.predict(X_test)
cm=confusion_matrix(y_test_o,y_pred_L_o)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('LogisticRegression with over-sampled data')
plt.show()

print(classification_report(y_test_o,y_pred_L_o))

print()
print()

clf=KNeighborsClassifier()
clf.fit(X_train,y_train)
y_pred_K_o=clf.predict(X_test)
cm=confusion_matrix(y_test_o,y_pred_K_o)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('KNeighborsClassifier with over-sampled data')
plt.show()

print(classification_report(y_test_o,y_pred_K_o))

print()
print()
clf=SVC()
clf.fit(X_train,y_train)
y_pred_S_o=clf.predict(X_test)
cm=confusion_matrix(y_test_o,y_pred_S_o)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('SVC with over-sampled data')
plt.show()

print(classification_report(y_test_o,y_pred_S_o))
```



	precision	recall	f1-score	support
0	0.87	0.86	0.86	6127
1	0.86	0.87	0.87	6166
accuracy			0.86	12293
macro avg	0.86	0.86	0.86	12293
weighted avg	0.86	0.86	0.86	12293

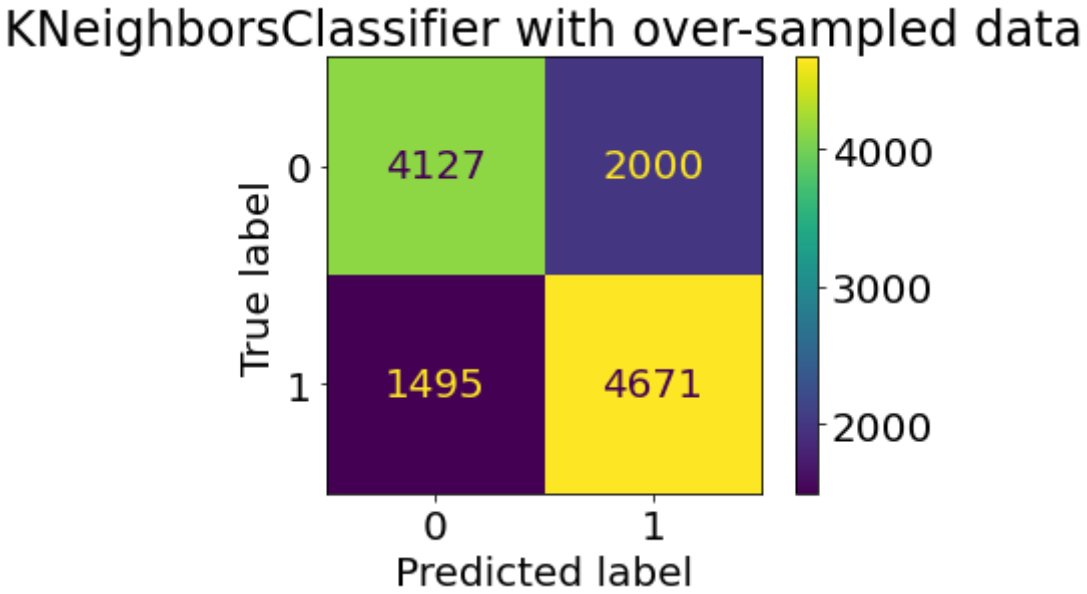


	precision	recall	f1-score	support
0	0.89	0.91	0.90	6127
1	0.91	0.89	0.90	6166
accuracy			0.90	12293
macro avg	0.90	0.90	0.90	12293
weighted avg	0.90	0.90	0.90	12293

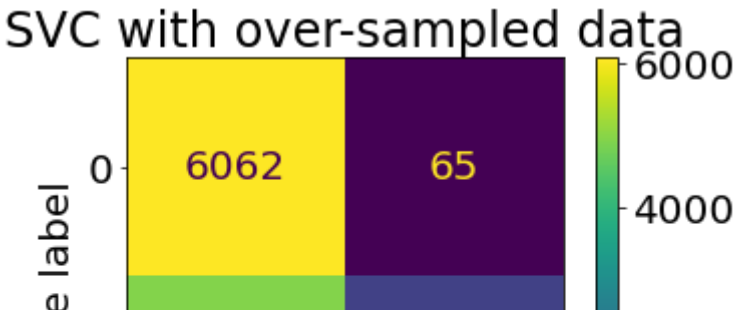


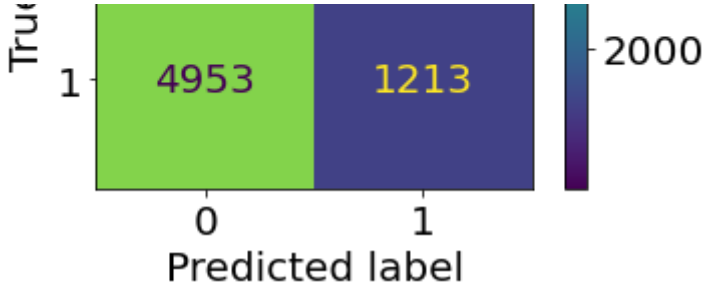
	precision	recall	f1-score	support
0	0.60	0.71	0.65	6127
1	0.65	0.53	0.59	6166
accuracy			0.62	12293
macro avg	0.63	0.62	0.62	12293

```
weighted avg       0.63      0.62      0.62      12293
```



	precision	recall	f1-score	support
0	0.73	0.67	0.70	6127
1	0.70	0.76	0.73	6166
accuracy			0.72	12293
macro avg	0.72	0.72	0.72	12293
weighted avg	0.72	0.72	0.72	12293





	precision	recall	f1-score	support
0	0.55	0.99	0.71	6127
1	0.95	0.20	0.33	6166
accuracy			0.59	12293
macro avg	0.75	0.59	0.52	12293
weighted avg	0.75	0.59	0.52	12293

```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score



print('**********************************Final Results after oversampling the data**********************************************')
print()
results={
    "Algorithm":["Decision Tree","Random Forest","Logistic Regression","KNN Classifier","SVC Classifier"],
    "Precision_Score":[precision_score(y_test_o,y_pred_D_o),precision_score(y_test_o,y_pred_R_o),precision_score(y_test_o,y_pred_L_o)
       "f1_Score":[f1_score(y_test_o,y_pred_D_o),f1_score(y_test_o,y_pred_R_o),f1_score(y_test_o,y_pred_L_o),f1_score(y_test_o,y_pred
     "Recall_Score":[recall_score(y_test_o,y_pred_D_o),recall_score(y_test_o,y_pred_R_o),recall_score(y_test_o,y_pred_L_o),recall_sco
         "Percentage of misclassification":[(1-accuracy_score(y_test_o,y_pred_D_o))*100,(1-accuracy_score(y_test_o,y_pred_R_o))*100,

      "Accuracy":[accuracy_score(y_test_o,y_pred_D_o),accuracy_score(y_test_o,y_pred_R_o),accuracy_score(y_test_o,y_pred_L_o),accurac

results_df=pd.DataFrame(results)
results_df=results_df.sort_values(ascending=False,by="Accuracy")
results_df.head()
```

**********Final Results after oversampling the data**********

	Algorithm	Precision_Score	f1_Score	Recall_Score	Percentage of misclassification	Accuracy
1	Random Forest	0.906700	0.897715	0.888907	10.160254	0.898397
0	Decision Tree	0.861388	0.865558	0.869770	13.552428	0.864476
3	KNN Classifier	0.700195	0.727740	0.757541	28.430814	0.715692
2	Logistic					

```
print('**********Final Results after undersampling the data**********')
print()
results1=pd.DataFrame({
    "Algorithm":["Decision Tree","Random Forest","Logistic Regression","KNN Classifier","SVC Classifier"],
    "Precision_Score":[precision_score(y_test_u,y_pred_D_u),precision_score(y_test_u,y_pred_R_u),precision_score(y_test_u,y_pred_L_u)
      "f1_Score":[f1_score(y_test_u,y_pred_D_u),f1_score(y_test_u,y_pred_R_u),f1_score(y_test_u,y_pred_L_u),f1_score(y_test_u,y_pred
     "Recall_Score":[recall_score(y_test_u,y_pred_D_u),recall_score(y_test_u,y_pred_R_u),recall_score(y_test_u,y_pred_L_u),recall_sco
     "Percentage of misclassification":[(1-accuracy_score(y_test_u,y_pred_D_u))*100,(1-accuracy_score(y_test_u,y_pred_R_u))*100,(1-ac
      "Accuracy":[accuracy_score(y_test_u,y_pred_D_u),accuracy_score(y_test_u,y_pred_R_u),accuracy_score(y_test_u,y_pred_L_u),accurac


results1.sort_values(ascending=False,by="Accuracy")
```

**********Final Results after undersampling the data**********

	Algorithm	Precision_Score	f1_Score	Recall_Score	Percentage of misclassification	Accuracy
1	Random Forest	0.823204	0.823618	0.824032	17.972350	0.820276
0	Decision Tree	0.784114	0.779155	0.774258	22.350230	0.776498
2	Logistic Regression	0.819149	0.449471	0.309703	38.632873	0.613671
3	KNN Classifier	0.622960	0.598380	0.575666	39.349718	0.606503

```
print('THE MODEL WITH THE BEST ACCURACY IS RANDOM FOREST CLASSIFIER WITH ACCURACY OF {}'.format(accuracy_score(y_test_o,y_pred_R_o)))
```

```
THE MODEL WITH THE BEST ACCURACY IS RANDOM FOREST CLASSIFIER WITH ACCURACY OF 0.898397461970227
```

✓ 0s completed at 4:30 PM