Random Forest Classification Model - Credit Card Fraud Detection dataset

Objective

- The aim of this dataset is to build a model that can accurately detect credit card fraudulent transactions to prevent fraudulent activity.

Random Forest

- A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.
- It establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome. It employs the bagging method to generate the required prediction.
- It eradicates the biggest limitation of decision tree - overfitting of dataset and increases precision.
- The main difference between the decision tree algorithm and the random forest algorithm is that establishing root nodes and segregating nodes is done randomly in the latter.

How does Random Forest algorithm work?

- Random forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled.
- The algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample.
- Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees.
- Depending on the type of problem, the determination of the prediction will vary. For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e. the most frequent categorical variable—will yield the predicted class.
- Finally, the oob sample is then used for cross-validation, finalizing that prediction.

Dataset source & brief

- The “Credit Card Fraud Detection” dataset on Kaggle is a highly imbalanced dataset that contains transactions made by credit cards in September 2013 by European cardholders.
- The dataset includes a total of 284,807 transactions, out of which only 492 are fraudulent, making the dataset highly imbalanced. The dataset includes 28 features, which are numerical values obtained by PCA transformation to maintain the confidentiality of sensitive information.
- The features include 'Time', 'Amount', and 'V1' through 'V28', as well as the 'Class' variable, which is the target variable indicating whether the transaction is fraudulent (1) or not (0).

Outline

- In this project, I will start with data processing and exploratory data analysis (EDA) to get a better understanding of the data. Next, will perform modeling, where I will build Bagging & Random Forest classification models to predict fraudulent transactions. I will also address the issue of imbalanced classes by using undersampling. Finally, will evaluate the performance of the models and choose the best one based on various evaluation metrics such as precision, recall, F1-score, and accuracy.

Import required libraries

In [106]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Load the dataset

In [107]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Dataset\creditcard.csv")
df.head()
```

Out[107]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [108]:

```
df.shape      # Check shape
```

Out[108]:

```
(284807, 31)
```

In [109]:

```
df.info()    # Check info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [110]:

```
df.isnull().sum()     # Check null values
```

Out[110]:

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [111]:

```
df.describe().T.style.background_gradient(cmap='Blues')  # statistical summary
```

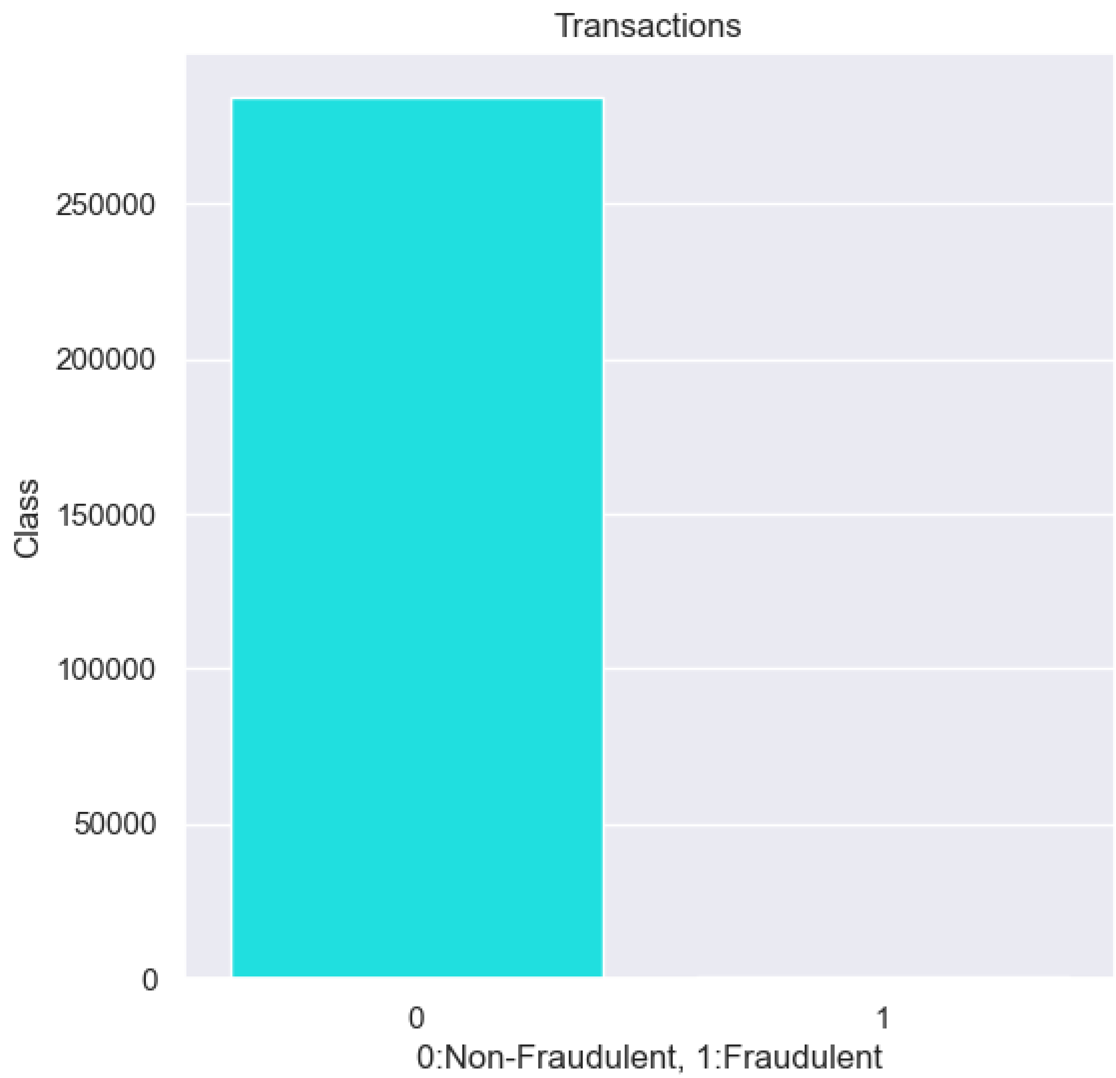
Out[111]:

	count	mean	std	min	25%	50%
Time	284807.000000	94813.859575	47488.145955	0.000000	54201.500000	84692.00000
V1	284807.000000	0.000000	1.958696	-56.407510	-0.920373	0.01810
V2	284807.000000	0.000000	1.651309	-72.715728	-0.598550	0.06548
V3	284807.000000	-0.000000	1.516255	-48.325589	-0.890365	0.17984
V4	284807.000000	0.000000	1.415869	-5.683171	-0.848640	-0.01984
V5	284807.000000	0.000000	1.380247	-113.743307	-0.691597	-0.05433
V6	284807.000000	0.000000	1.332271	-26.160506	-0.768296	-0.27418
V7	284807.000000	-0.000000	1.237094	-43.557242	-0.554076	0.04010
V8	284807.000000	0.000000	1.194353	-73.216718	-0.208630	0.02235
V9	284807.000000	-0.000000	1.098632	-13.434066	-0.643098	-0.05142
V10	284807.000000	0.000000	1.088850	-24.588262	-0.535426	-0.09291
V11	284807.000000	0.000000	1.020713	-4.797473	-0.762494	-0.03275
V12	284807.000000	-0.000000	0.999201	-18.683715	-0.405571	0.14003
V13	284807.000000	0.000000	0.995274	-5.791881	-0.648539	-0.01356
V14	284807.000000	0.000000	0.958596	-19.214325	-0.425574	0.05060
V15	284807.000000	0.000000	0.915316	-4.498945	-0.582884	0.04807
V16	284807.000000	0.000000	0.876253	-14.129855	-0.468037	0.06641
V17	284807.000000	-0.000000	0.849337	-25.162799	-0.483748	-0.06567
V18	284807.000000	0.000000	0.838176	-9.498746	-0.498850	-0.00363
V19	284807.000000	0.000000	0.814041	-7.213527	-0.456299	0.00373
V20	284807.000000	0.000000	0.770925	-54.497720	-0.211721	-0.06248
V21	284807.000000	0.000000	0.734524	-34.830382	-0.228395	-0.02945
V22	284807.000000	-0.000000	0.725702	-10.933144	-0.542350	0.00678
V23	284807.000000	0.000000	0.624460	-44.807735	-0.161846	-0.01119
V24	284807.000000	0.000000	0.605647	-2.836627	-0.354586	0.04097
V25	284807.000000	0.000000	0.521278	-10.295397	-0.317145	0.01659
V26	284807.000000	0.000000	0.482227	-2.604551	-0.326984	-0.05213
V27	284807.000000	-0.000000	0.403632	-22.565679	-0.070840	0.00134
V28	284807.000000	-0.000000	0.330083	-15.430084	-0.052960	0.01124
Amount	284807.000000	88.349619	250.120109	0.000000	5.600000	22.00000
Class	284807.000000	0.001727	0.041527	0.000000	0.000000	0.00000

Exploratory Data Analysis

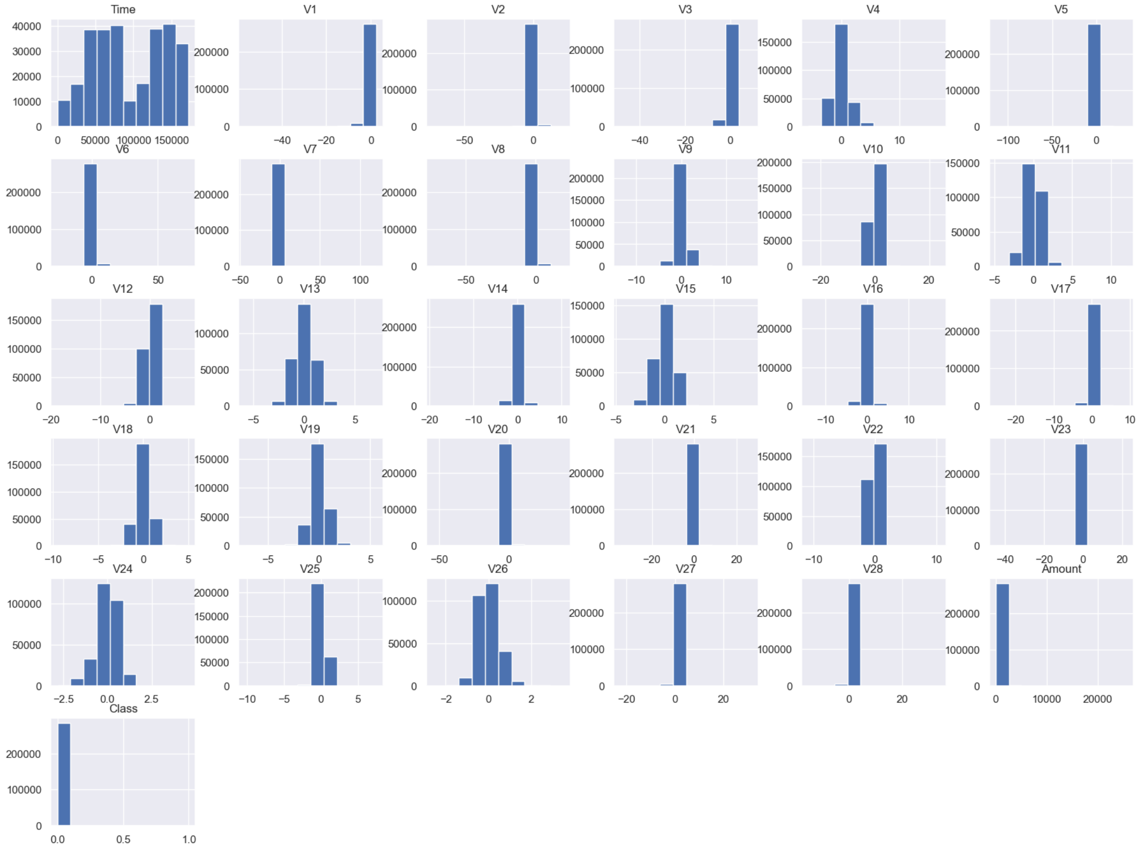
In [112]:

```
plt.figure(figsize=(6,6))
sns.barplot(x=df['Class'].value_counts().index, y=df['Class'].value_counts(), color='aqu
plt.title('Transactions')
plt.xlabel('0:Non-Fraudulent, 1:Fraudulent')
plt.show()
```



In [113]:

```
df.hist(figsize = (20, 15))
plt.show()
```



Data Splitting

In [114]:

```
#Split data into dep and ind
x=df.drop(['Class'],axis=1)
y=df[['Class']]
```

Feature Scaling

In [117]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x1=sc.fit_transform(x)
pd.DataFrame(x1).head(2)
```

Out[117]:

	0	1	2	3	4	5	6	7	
0	-1.996583	-0.694242	-0.044075	1.672773	0.973366	-0.245117	0.347068	0.193679	0.082
1	-1.996583	0.608496	0.161176	0.109797	0.316523	0.043483	-0.061820	-0.063700	0.071

2 rows × 30 columns

In [118]:

```
y.value_counts()       # Check imbalance data
```

Out[118]:

```
Class
0        284315
1           492
dtype: int64
```

Handle imbalance data

In [119]:

```
import imblearn     # under sampling done
from imblearn.under_sampling import RandomUnderSampler
ros=RandomUnderSampler()
x_un,y_un=ros.fit_resample(x1,y)
print(x_un.shape,y_un.shape,y.shape)
```

```
(984, 30) (984, 1) (284807, 1)
```

In [120]:

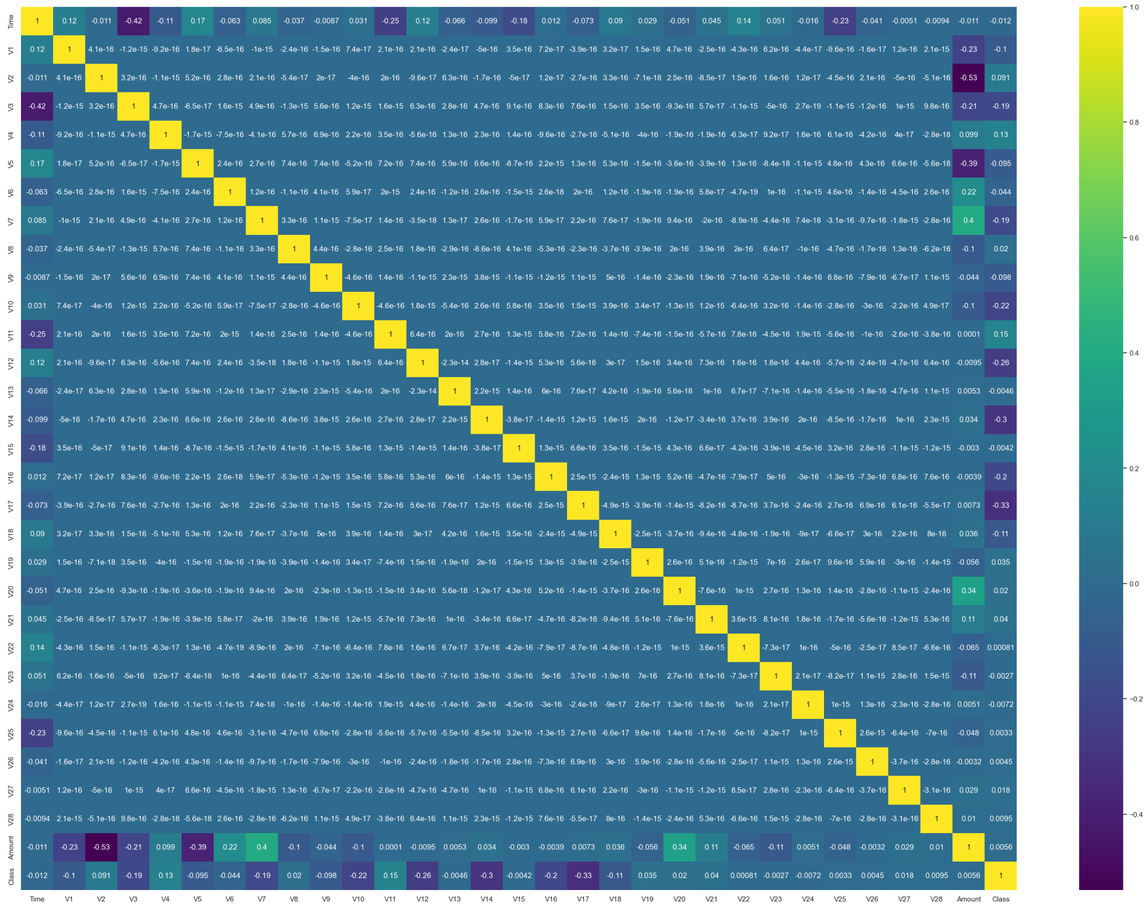
```
y_un.value_counts()
```

Out[120]:

```
Class
0        492
1        492
dtype: int64
```

In [121]:

```
plt.figure(figsize=(35,25))    # Correlation
sns.heatmap(df.corr(), annot=True, cmap='viridis')
plt.show()
```



In [122]:

```
# Split data into train and test
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_un,y_un,test_size=0.20,random_state=101
```

Building Bagging algorithm

- Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms. Bagging involves using different samples of data (training data) rather than just one sample. A training dataset comprises observations and features that are used for making predictions. The decision trees produce different outputs, depending on the training data fed to the random forest algorithm. These outputs will be ranked, and the highest will be selected as the final output.

In [123]:

```
from sklearn.ensemble import BaggingClassifier
bagging=BaggingClassifier()
bagging.fit(x_train,y_train)
```

Out[123]:

```
▾ BaggingClassifier
BaggingClassifier()
```

In [124]:

```
# Predict
y_pred_train_bag=bagging.predict(x_train)
y_pred_test_bag=bagging.predict(x_test)
```

In [125]:

```
# Evaluate bagging algorithm
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,prec
```

In [126]:

```
print(confusion_matrix(y_train,y_pred_train_bag))
print(confusion_matrix(y_test,y_pred_test_bag))
```

```
[[394   0]
 [  7 386]]
[[94  4]
 [ 8 91]]
```

In [127]:

```
print('Bagging algorithm train accuracy')
acc= accuracy_score(y_train,y_pred_train_bag)
print('Accuracy score is',acc)
prec= precision_score(y_train,y_pred_train_bag)
print('Precision score is',prec)
rec= recall_score(y_train,y_pred_train_bag)
print('Recall score is',rec)
f1= f1_score(y_train,y_pred_train_bag)
print('F1-Score is',f1)
```

```
Bagging algorithm train accuracy
Accuracy score is 0.9911054637865311
Precision score is 1.0
Recall score is 0.9821882951653944
F1-Score is 0.9910141206675225
```

In [128]:

```
print("Bagging algorithm test accuracy")
acc= accuracy_score(y_test,y_pred_test_bag)
print('Accuracy score is',acc)
prec= precision_score(y_test,y_pred_test_bag)
print('Precision score is',prec)
rec= recall_score(y_test,y_pred_test_bag)
print('Recall score is',rec)
f1= f1_score(y_test,y_pred_test_bag)
print('F1-Score is',f1)
```

```
Bagging algorithm test accuracy
Accuracy score is 0.9390862944162437
Precision score is 0.9578947368421052
Recall score is 0.9191919191919192
F1-Score is 0.9381443298969072
```

Random Forest Classifier model

In [129]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=200,oob_score=False)
rf.fit(x_train,y_train)
```

Out[129]:

```
▾ RandomForestClassifier
RandomForestClassifier(n_estimators=200)
```

In [130]:

```
#predict
y_pred_train_rf=rf.predict(x_train)
y_pred_test_rf=rf.predict(x_test)
```

In [131]:

```
print(confusion_matrix(y_train,y_pred_train_rf))
print(confusion_matrix(y_test,y_pred_test_rf))
```

```
[[394   0]
 [  0 393]]
[[96  2]
 [ 6 93]]
```

In [132]:

```
# Evaluate
print('Random Forest Train data accuracy')
acc= accuracy_score(y_train,y_pred_train_rf)
print('Accuracy score is',acc)
prec= precision_score(y_train,y_pred_train_rf)
print('Precision score is',prec)
rec= recall_score(y_train,y_pred_train_rf)
print('Recall score is',rec)
f1= f1_score(y_train,y_pred_train_rf)
print('F1-Score is',f1)
```

```
Random Forest Train data accuracy
Accuracy score is 1.0
Precision score is 1.0
Recall score is 1.0
F1-Score is 1.0
```

In [133]:

```
print('Random Forest Test data accuracy')
acc= accuracy_score(y_test,y_pred_test_rf)
print('Accuracy score is',acc)
prec= precision_score(y_test,y_pred_test_rf)
print('Precision score is',prec)
rec= recall_score(y_test,y_pred_test_rf)
print('Recall score is',rec)
f1= f1_score(y_test,y_pred_test_rf)
print('F1-Score is',f1)
```

```
Random Forest Test data accuracy
Accuracy score is 0.9593908629441624
Precision score is 0.9789473684210527
Recall score is 0.9393939393939394
F1-Score is 0.9587628865979383
```

Confusion matrix

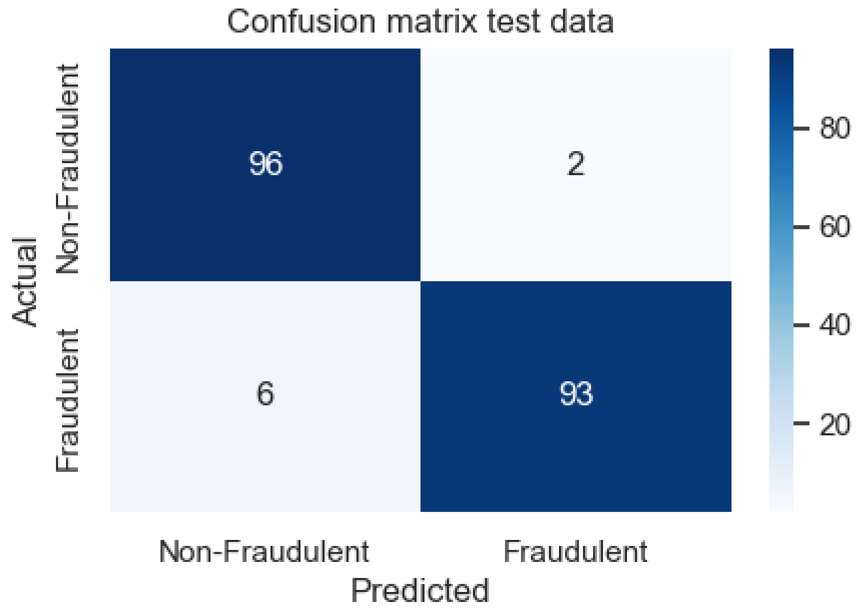
- A confusion matrix presents a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes. It plots a table of all the predicted and actual values of a classifier.

In [135]:

```
#printing the confusion matrix of train data
Labels = ['Non-Fraudulent', 'Fraudulent']
conf_matrix = confusion_matrix(y_train, y_pred_train_rf)
plt.figure(figsize=(5,3))
sns.heatmap(conf_matrix,xticklabels= Labels, yticklabels=Labels,cmap='Blues', annot=True
plt.title("Confusion matrix train data")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

In [136]:

```
#printing the confusion matrix of test data
Labels = ['Non-Fraudulent', 'Fraudulent']
conf_matrix = confusion_matrix(y_test, y_pred_test_rf)
plt.figure(figsize=(5,3))
sns.heatmap(conf_matrix,xticklabels= Labels, yticklabels=Labels,cmap='Blues', annot=True
plt.title("Confusion matrix test data")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [137]:

```
### Cross validation for Train data as it has overfitting problem
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(rf, x_train, y_train, cv=10)
print("Train Accuracy after Cross validation:", training_accuracy.mean())
```

```
Train Accuracy after Cross validation: 0.9301363193768257
```

Conclusion

- In this project I tried to build Random Forest Classifier Model to detect fraudulent credit card transactions.
- Both train and test data accuracy are coming above the commonly taken threshold value of 75%.
- There is less than 10% accuracy variation in both datasets.
- The main problem when dealing was the highly imbalanced dataset in which the majority of the transaction are non-fraud ones so used undersampling method to handle it.

In [ ]:

In [ ]:

```

```