



**PROJECT REPORT:
END-TO-END CREDIT CARD FRAUD
DETECTION SYSTEM**

**COURSE NAME: AMOD 5410H-BIG DATA
COURSE INSTRUCTOR: PROF. BRIAN SRIVASTAVA**

Submitted By:

**RAJAT DAXESH DESAI (0698360)
JIMISH PRAMOD KHOLAPURE (0692165)**

Abstract

Credit Card has become an integral part of every individual. People use credit-cards in their day-to-day life for various transactions, be it buying groceries in a store, paying different kind of bills, buying video games and all other sorts of entertainments, to pay for the subscription of various services, and the list goes on.

Credit Cards are also used by the respective banks to lend money to their customers and expect them to return it with the interests after a particular time-period. These banks keep track of the individuals daily credit history in order to generate a credit score, which helps them to determine a customer's buying capacity and who is eligible for loan of a certain amount and if they would be able to repay it to the bank and all other risk factors included in the steps of issuing a loan.

Furthermore, they must keep safe the credentials of the credit-card holder in order to prevent anyone from committing financial frauds with their customers. Thus, they must keep track of the cards using patterns of each and every with their respective banks. Thus, banks have to find a way in which they can identify the types of fraud that occur in a transaction and find a way to stop the fraud from happening on the first place by predicting what can be described as a fraudulent transaction. Thus, the aim of our project is to build an End-to-End Credit Card Fraud Detection System, which provides a solution to the aforementioned problem and develops a model which detects the fraudulent transaction from happening in the first place. This isn't the first time anyone is attempting to build this type of system. Banks have been using the same for quite some time now, but the main aim of our project is to learn about the working of various tools and machine learning algorithms that are going to be used by us to create the same.

Pre-Processing

The dataset is obtained from the Kaggle dataset repository (<https://www.kaggle.com/mlg-ulb/creditcardfraud>) is uploaded by Machine Learning Group-(ULB), who collected the data for research on big data mining and fraud detection.

The dataset contains the transactions made by credit cards in September 2013 by European cardholders. The dataset contains 285,299 rows distributed along 31 columns; each row is a transaction that has happened in two days. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues the original features and more background information about the data has not been depicted. Features V1,V2...., V8 are the principal components obtained with PCA, the only features that which have not been transformed with PCA are 'Time', 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.

The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise. Have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. No other pre-processing of the data is required.

Methods and Tools

The tool that we are going to use is Kafka. The data must be streamed continuously in a pipeline and must be analysed and stored in the model to work continuously. Also, the transactions should be stored automatically in real time. Thus, Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data. Kafka is open-source software which provides a framework for storing, reading and analysing streaming data.

Furthermore, we'll use Jupyter Notebook to code in python. We'll first determine which sampling technique should be used as the dataset is highly imbalanced as fraudulent transactions present are very less, which it should be and then determine the model estimator to be used in order to obtain the highest accuracy possible. That is, we'll try to implement four types of models namely Random Forest Classifier, SJDC (Stochastic Gradient Descent Classification), Logistic Regression, Support Vector Machine Classifier and then try to determine which model has the highest accuracy and deploy it for detecting the fraudulent transactions.

A Guide to Kafka

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

To download: <https://kafka.apache.org/>

Kafka Capabilities:

- High Throughput
- Scalable
- Permanent Storage
- High Availability
- Built-in Stream Processing

A Guide to Python

Python is a popular programming language used on a server to create web application.

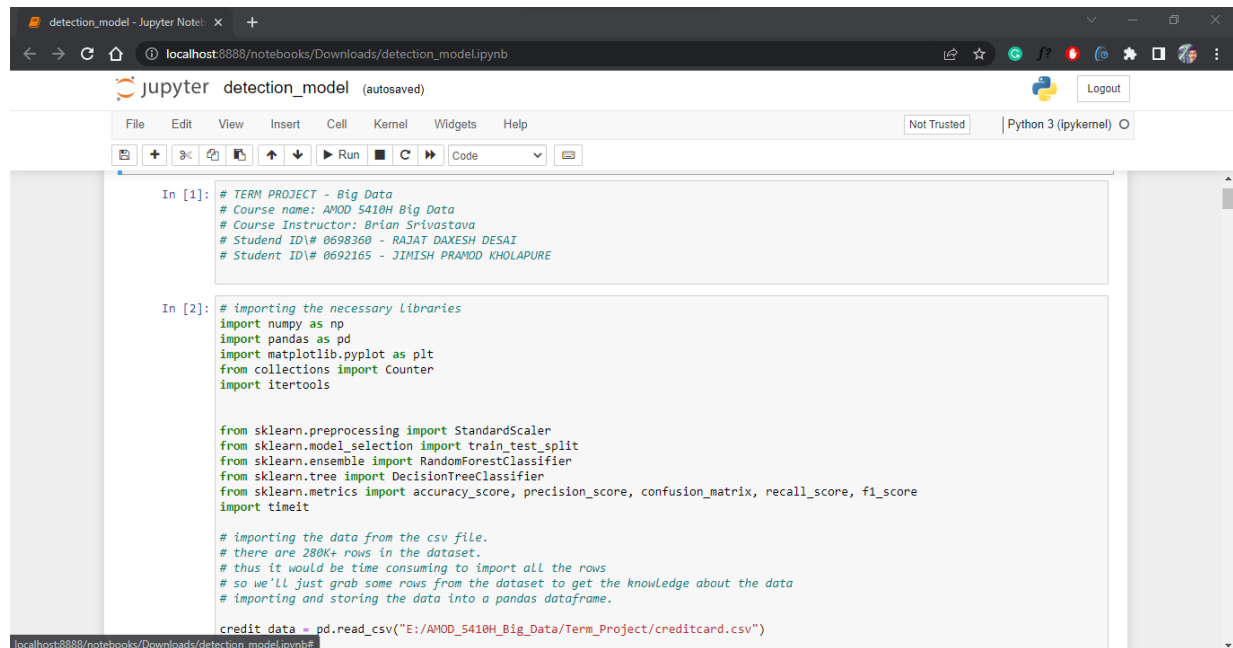
To download: <https://www.python.org/>

Jupyter Notebook is a web-based interactive development environment for the notebooks, code and data. Its flexible interface allows user to configure and arrange workflows in data science, scientific computing, computational journalism and machine learning. A modular design invites extensions to expand and enrich functionality.

To download: <https://jupyter.org/>

Implementation

1. Imported the necessary libraries for the code implementation along with the selected dataset. Here, we have used Jupyter notebook as the python IDE.



```
In [1]: # TERM PROJECT - Big Data
# Course name: AMOD 5410H Big Data
# Course Instructor: Brian Srivastava
# Student ID# 0698360 - RAJAT DAXESH DESAI
# Student ID# 0692165 - JIMISH PRAMOD KHOLAPURE

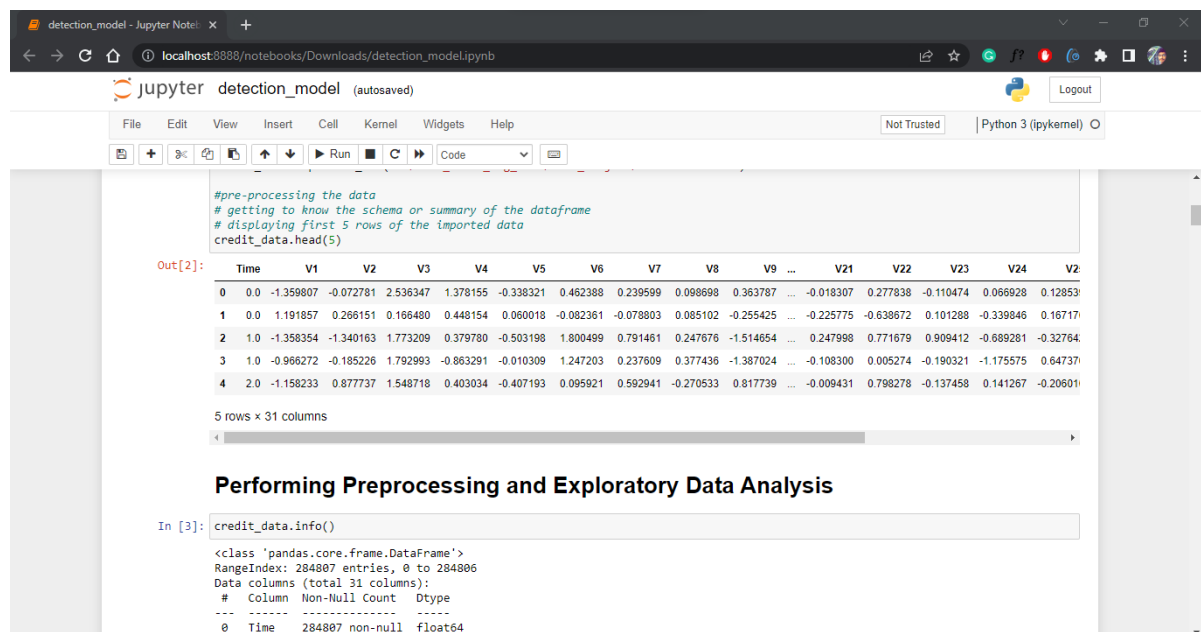
In [2]: # importing the necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import itertools

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix, recall_score, f1_score
import timeit

# importing the data from the csv file.
# there are 280K+ rows in the dataset.
# thus it would be time consuming to import all the rows
# so we'll just grab some rows from the dataset to get the knowledge about the data
# importing and storing the data into a pandas dataframe.

credit_data = pd.read_csv("E:/AMOD_5410H_Big_Data/Term_Project/creditcard.csv")
```

2. Pre-processing the dataset to clean the data from noise and unwanted data items that would cause error or misguide the results.



```
#pre-processing the data
# getting to know the schema or summary of the dataframe
# displaying first 5 rows of the imported data
credit_data.head(5)

Out[2]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows x 31 columns

Performing Preprocessing and Exploratory Data Analysis

```
In [3]: credit_data.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
```

3. Getting the summary of the dataset

```
detection_model - Jupyter Note: x +
localhost8888/notebooks/Downloads/detection_model.ipynb

jupyter detection_model (autosaved)
Not Trusted | Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

credit_data.Class.value_counts()

Out[4]: 0    284315
        1     492
        Name: Class, dtype: int64

In [5]: credit_data["Amount"].describe()

Out[5]: count    284807.000000
        mean     88.349619
        std      250.120109
        min       0.000000
        25%       5.000000
        50%      22.000000
        75%      77.165000
        max     25091.160000
        Name: Amount, dtype: float64

In [6]: valid_transaction = len(credit_data[credit_data.Class == 0])
        fraud = len(credit_data[credit_data.Class == 1])
        fraud_percent = (fraud / (fraud + valid_transaction)) * 100

        print("Number of Genuine transactions: ", valid_transaction)
        print("Number of Fraud transactions: ", fraud)
        print("Percentage of Fraud transactions: {:.4f}".format(fraud_percent))

        Number of Genuine transactions: 284315
        Number of Fraud transactions: 492
        Percentage of Fraud transactions: 0.1727

localhost8888/notebooks/Downloads/detection_model.ipynb#
```

4. Visualising the dataset in to fraud and genuine transactions in order to make it easier for the novice user to understand.

```
detection_model - Jupyter Note: x +
localhost8888/notebooks/Downloads/detection_model.ipynb

File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel)

In [7]: # Visualizing the dataset into Genuine and Fraud Labels

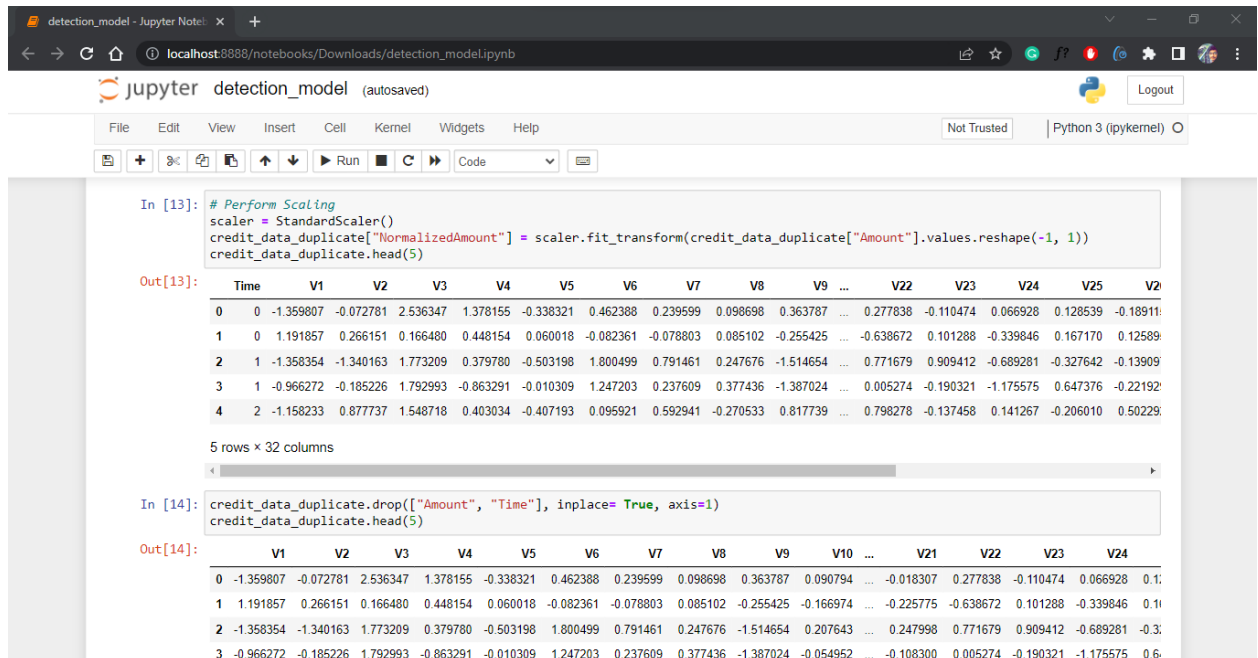
        labels = ["Genuine", "Fraud"]
        count_classes = credit_data.value_counts(credit_data['Class'], sort=True)
        count_classes.plot(kind="bar", rot=0)
        plt.title("Visualization of Labels")
        plt.ylabel("Count")
        plt.xticks(range(2), labels)
        plt.show()

        Visualization of Labels

        250000
        200000
        150000
        100000
        50000
        0
        Genuine      Fraud
        Class

In [8]: credit_data["Amount"].head(5)
```

5. Performing Scaling operation to get the multi ranged data into a specified range.



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, labeled 'In [13]:', performs scaling on the 'Amount' column of a dataset. It uses a `StandardScaler` to fit and transform the data, then displays the first five rows of the transformed data. The second cell, labeled 'In [14]:', drops the 'Amount' and 'Time' columns from the dataset and displays the first five rows of the resulting data.

```
In [13]: # Perform Scaling
scaler = StandardScaler()
credit_data_duplicate["NormalizedAmount"] = scaler.fit_transform(credit_data_duplicate["Amount"].values.reshape(-1, 1))
credit_data_duplicate.head(5)
```

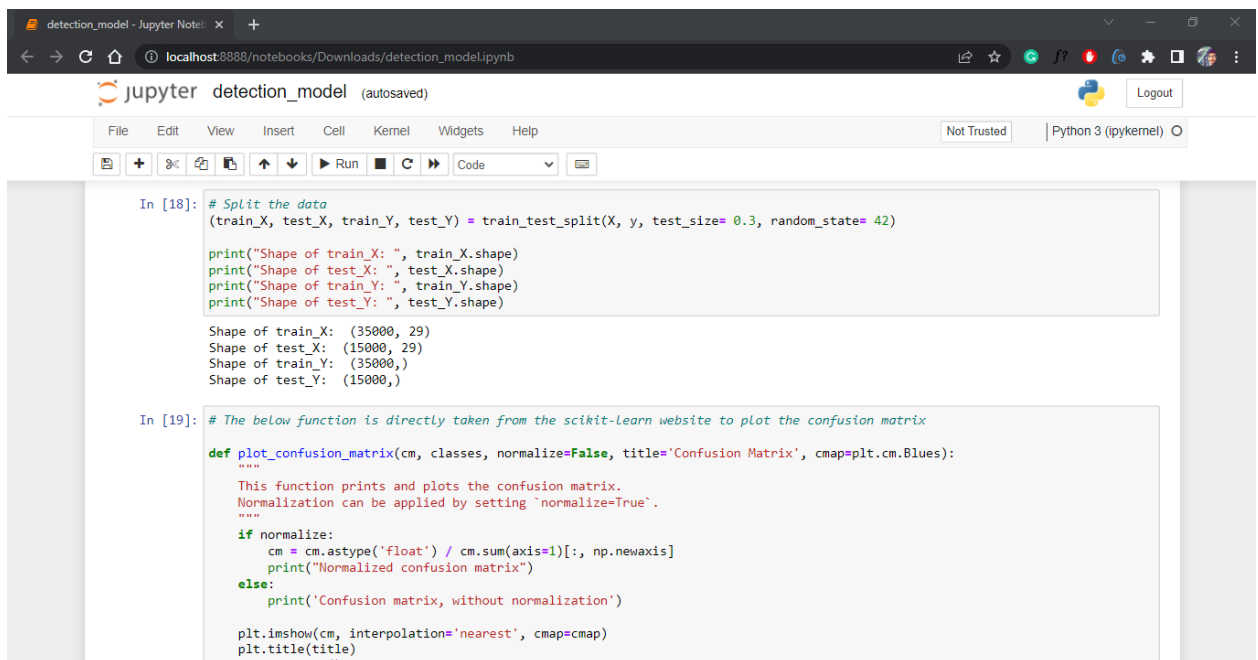
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V22	V23	V24	V25	V26
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.277838	-0.110474	0.066928	0.128539	-0.189111
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.638672	0.101288	-0.339846	0.167170	0.125899
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.771679	0.909412	-0.689281	-0.327642	-0.139099
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	0.005274	-0.190321	-1.175575	0.647376	-0.221929
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.798278	-0.137458	0.141267	-0.206010	0.502299

5 rows x 32 columns

```
In [14]: credit_data_duplicate.drop(["Amount", "Time"], inplace=True, axis=1)
credit_data_duplicate.head(5)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	V25
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.128539
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207843	...	0.247998	0.771679	0.909412	-0.689281	-0.327642
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376

6. Splitting the dataset into training set and test set for testing various models.



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, labeled 'In [18]:', splits the dataset into training and testing sets using `train_test_split` and prints the shapes of the resulting arrays. The second cell, labeled 'In [19]:', defines a custom function `plot_confusion_matrix` that takes a confusion matrix, classes, and other parameters, and plots it using `plt.imshow`.

```
In [18]: # Split the data
(train_X, test_X, train_Y, test_Y) = train_test_split(X, y, test_size=0.3, random_state=42)

print("Shape of train_X: ", train_X.shape)
print("Shape of test_X: ", test_X.shape)
print("Shape of train_Y: ", train_Y.shape)
print("Shape of test_Y: ", test_Y.shape)

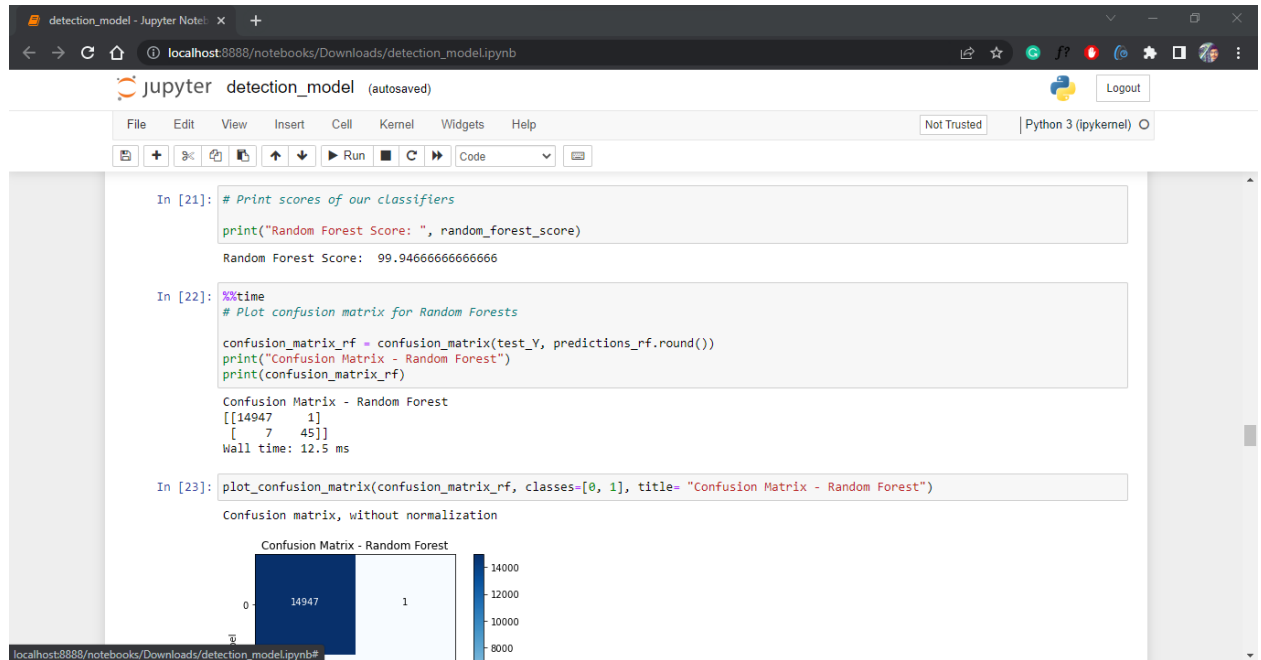
Shape of train_X: (35000, 29)
Shape of test_X: (15000, 29)
Shape of train_Y: (35000,)
Shape of test_Y: (15000,)
```

```
In [19]: # The below function is directly taken from the scikit-learn website to plot the confusion matrix

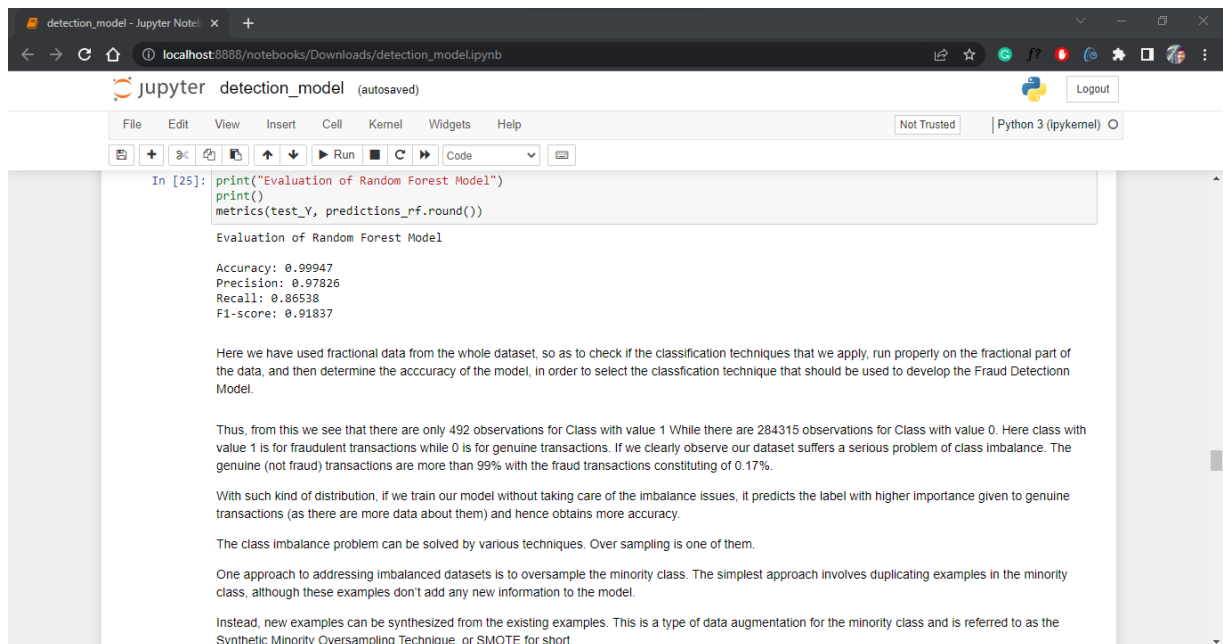
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion Matrix', cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting 'normalize=True'.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
```

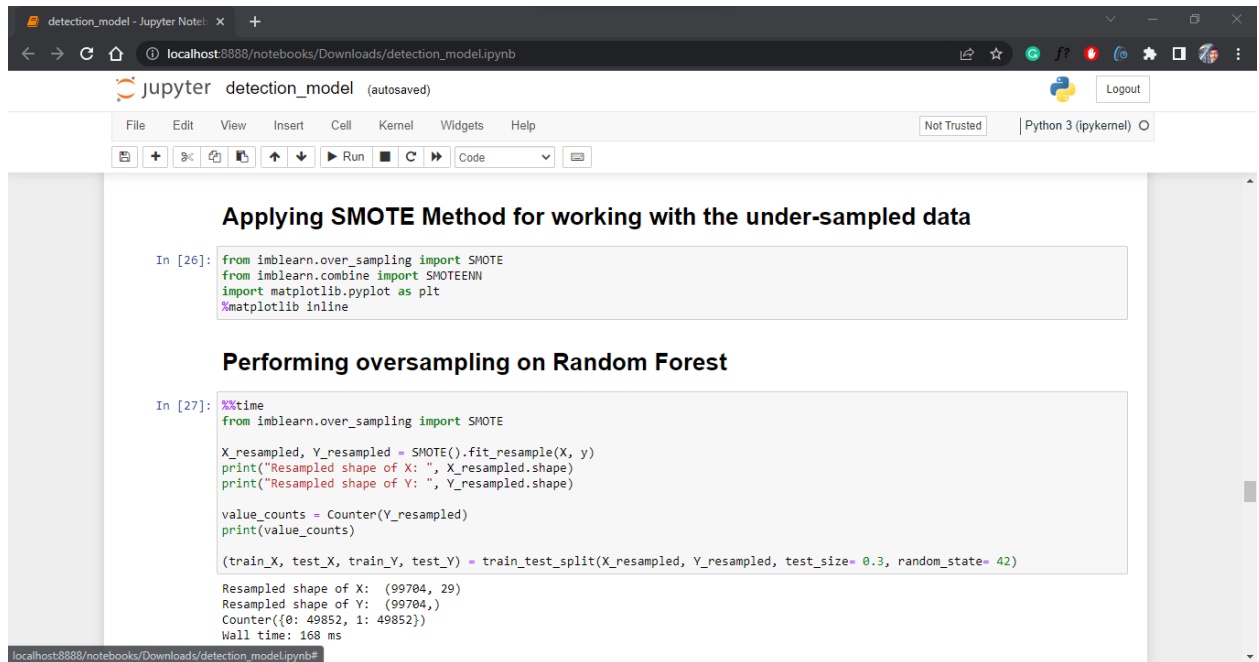

7. Checking the score for the random forest and plotting the confusion matrix for the same.



8. Evaluation of Random Forest Classifier and the writeup for the same



9. Applying SMOTE (Synthetic Minority Oversampling technique)for the under-sampled data to balance to dataset in proper manner



The screenshot shows a Jupyter Notebook titled 'detection_model' with two code cells. The first cell, 'Applying SMOTE Method for working with the under-sampled data', imports SMOTE and SHOTENIN from imblearn. The second cell, 'Performing oversampling on Random Forest', uses SMOTE to resample the data, prints the new shapes, and splits the data into training and testing sets. The output shows the resampled shapes and the split parameters.

```
In [26]: from imblearn.over_sampling import SMOTE
from imblearn.combine import SHOTENIN
import matplotlib.pyplot as plt
%matplotlib inline

In [27]: %%time
from imblearn.over_sampling import SMOTE

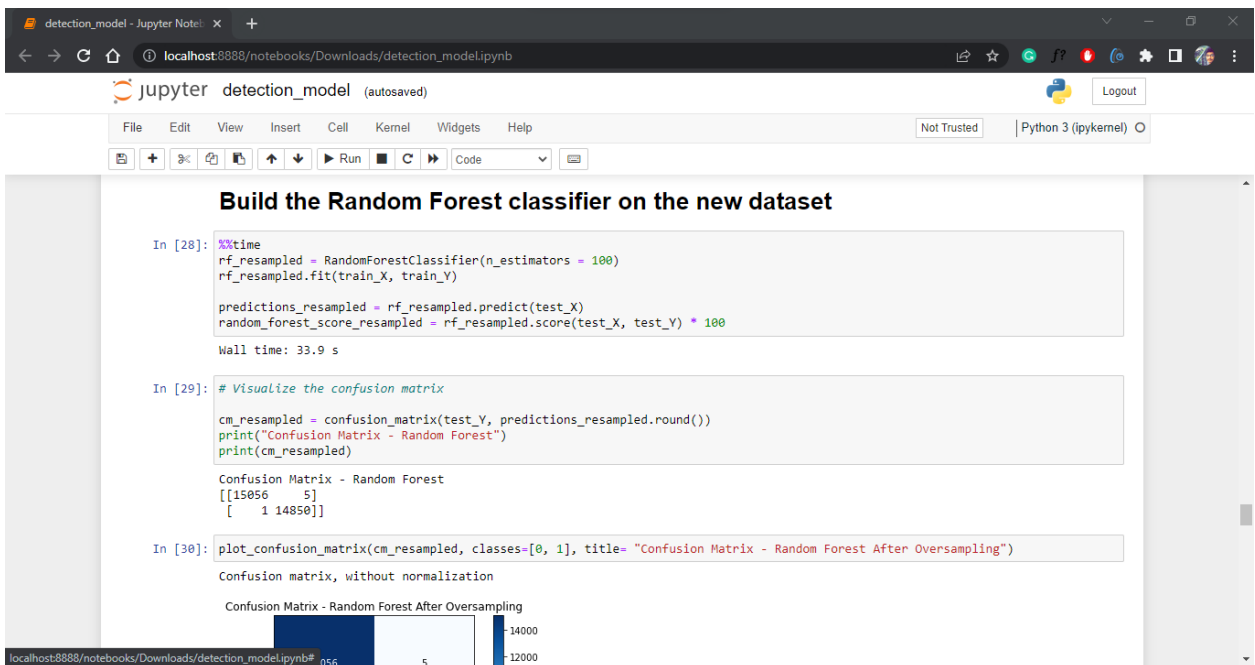
X_resampled, Y_resampled = SMOTE().fit_resample(X, y)
print("Resampled shape of X: ", X_resampled.shape)
print("Resampled shape of Y: ", Y_resampled.shape)

value_counts = Counter(Y_resampled)
print(value_counts)

(train_X, test_X, train_Y, test_Y) = train_test_split(X_resampled, Y_resampled, test_size= 0.3, random_state= 42)

Resampled shape of X: (99704, 29)
Resampled shape of Y: (99704,)
Counter({0: 49852, 1: 49852})
Wall time: 168 ms
```

10. Building Random Forest Classifier on the New Dataset, plotting the visualisation and confusion matrix.



The screenshot shows a Jupyter Notebook titled 'detection_model' with three code cells. The first cell, 'Build the Random Forest classifier on the new dataset', trains a Random Forest classifier and prints the predictions and scores. The second cell, '# Visualize the confusion matrix', prints the confusion matrix. The third cell, 'plot_confusion_matrix', plots the confusion matrix with a title.

```
In [28]: %%time
rf_resampled = RandomForestClassifier(n_estimators = 100)
rf_resampled.fit(train_X, train_Y)

predictions_resampled = rf_resampled.predict(test_X)
random_forest_score_resampled = rf_resampled.score(test_X, test_Y) * 100

Wall time: 33.9 s

In [29]: # Visualize the confusion matrix

cm_resampled = confusion_matrix(test_Y, predictions_resampled.round())
print("Confusion Matrix - Random Forest")
print(cm_resampled)

Confusion Matrix - Random Forest
[[15056  5]
 [ 1 14850]]

In [30]: plot_confusion_matrix(cm_resampled, classes=[0, 1], title= "Confusion Matrix - Random Forest After Oversampling")

Confusion matrix, without normalization

Confusion Matrix - Random Forest After Oversampling
```

11. Performing SVM (Support Vector Machine) on the dataset

```
detection_model - Jupyter Note: x +
localhost:8888/notebooks/Downloads/detection_model.ipynb
jupyter detection_model (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted Python 3 (ipykernel)

Performing SVM classification

In [32]: # importing libraries for SVM
from sklearn.svm import LinearSVC, LinearSVR, SVC, SVR

In [33]: %%time
svm = SVC()
svm.fit(train_X, train_Y)

prediction_svm = svm.predict(test_X)
svm_score = svm.score(test_X, test_Y)*100

Wall time: 1min

In [34]: # Visualize the confusion matrix
cm_resampled_svm = confusion_matrix(test_Y, prediction_svm.round())
print("Confusion Matrix - Random Forest")
print(cm_resampled_svm)

Confusion Matrix - Random Forest
[[14957  104]
 [ 205 14646]]

In [35]: plot_confusion_matrix(cm_resampled, classes=[0, 1], title= "Confusion Matrix - Support Vector Machine(SVM) After Oversampling")

Confusion matrix, without normalization
```

12. Evaluation of Support Vector Machine and confusion matrix.

```
detection_model - Jupyter Note: x +
localhost:8888/notebooks/Downloads/detection_model.ipynb
jupyter detection_model (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted Python 3 (ipykernel)

In [35]: plot_confusion_matrix(cm_resampled, classes=[0, 1], title= "Confusion Matrix - Support Vector Machine(SVM) After Oversampling")

Confusion matrix, without normalization

Confusion Matrix - Support Vector Machine(SVM) After Oversampling

True label
0
1
Predicted label
0 1
14000
12000
10000
8000
6000
4000
2000

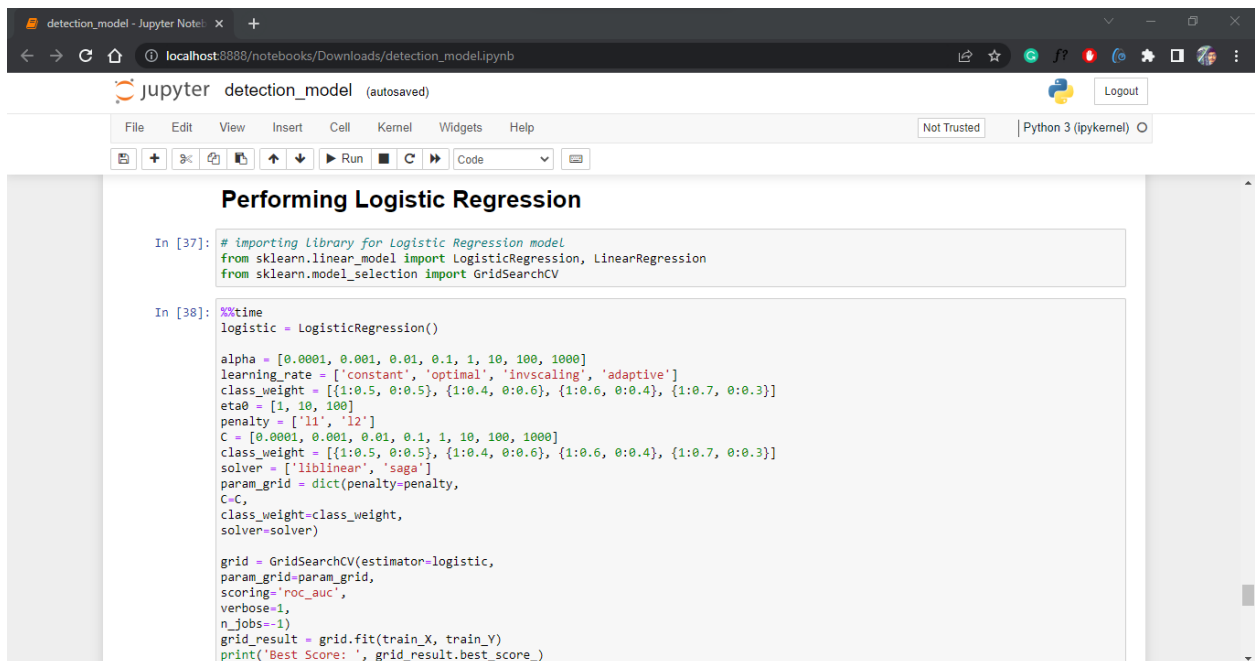
In [36]: print("Evaluation of SVM Model")
print()
metrics(test_Y, prediction_svm.round())

Evaluation of SVM Model

Accuracy: 0.98967
Precision: 0.99295
Recall: 0.98620
F1-score: 0.98956

localhost:8888/notebooks/Downloads/detection_model.ipynb*
```

13. Performing Logistic Regression on the dataset



The screenshot shows a Jupyter Notebook titled 'detection_model' with two code cells. The first cell imports the necessary libraries for logistic regression. The second cell defines the hyperparameters for the logistic regression model and performs a grid search to find the best parameters.

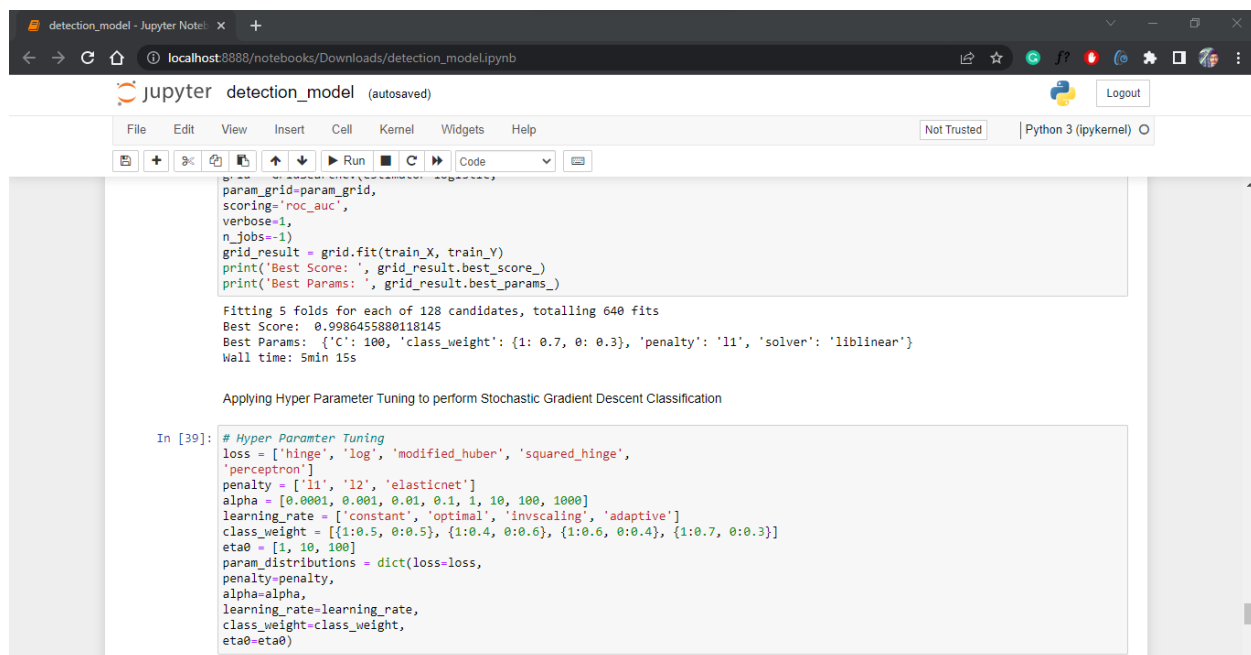
```
In [37]: # importing Library for Logistic Regression model
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import GridSearchCV

In [38]: %%time
logistic = LogisticRegression()

alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
learning_rate = ['constant', 'optimal', 'invscaling', 'adaptive']
class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
eta0 = [1, 10, 100]
penalty = ['l1', 'l2']
C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
solver = ['liblinear', 'saga']
param_grid = dict(penalty=penalty,
                  C=C,
                  class_weight=class_weight,
                  solver=solver)

grid = GridSearchCV(estimator=logistic,
                    param_grid=param_grid,
                    scoring='roc_auc',
                    verbose=1,
                    n_jobs=-1)
grid_result = grid.fit(train_X, train_Y)
print('Best Score: ', grid_result.best_score_)
```

14. Applying Hyper Parameter Tuning to perform Stochastic Gradient Descent Classification.



The screenshot shows a Jupyter Notebook titled 'detection_model' with two code cells. The first cell continues the grid search for the logistic regression model. The second cell defines the hyperparameters for the stochastic gradient descent classification model and performs a grid search to find the best parameters.

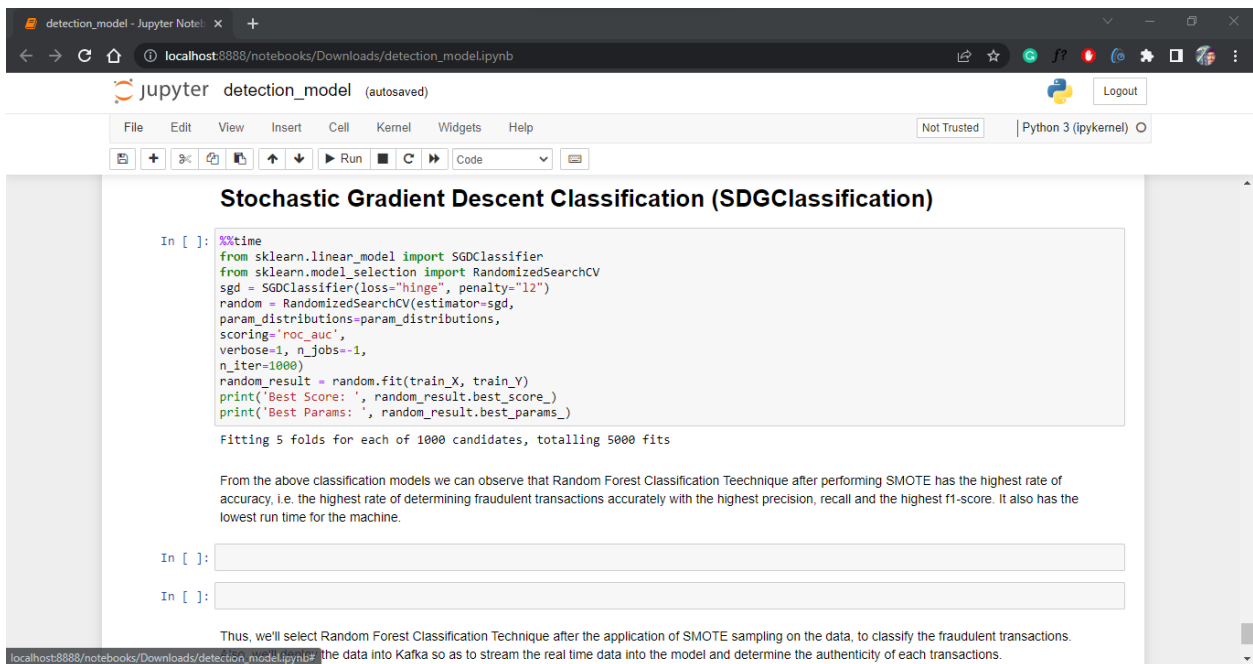
```
param_grid=param_grid,
scoring='roc_auc',
verbose=1,
n_jobs=-1)
grid_result = grid.fit(train_X, train_Y)
print('Best Score: ', grid_result.best_score_)
print('Best Params: ', grid_result.best_params_)

Fitting 5 folds for each of 128 candidates, totalling 640 fits
Best Score: 0.9986455880118145
Best Params: {'C': 100, 'class_weight': {1: 0.7, 0: 0.3}, 'penalty': 'l1', 'solver': 'liblinear'}
Wall time: 5min 15s

Applying Hyper Parameter Tuning to perform Stochastic Gradient Descent Classification

In [39]: # Hyper Parameter Tuning
loss = ['hinge', 'log', 'modified_huber', 'squared_hinge',
        'perceptron']
penalty = ['l1', 'l2', 'elasticnet']
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
learning_rate = ['constant', 'optimal', 'invscaling', 'adaptive']
class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
eta0 = [1, 10, 100]
param_distributions = dict(loss=loss,
                           penalty=penalty,
                           alpha=alpha,
                           learning_rate=learning_rate,
                           class_weight=class_weight,
                           eta0=eta0)
```

15. Performing Stochastic Gradient Descent Classification (SDGClassification)

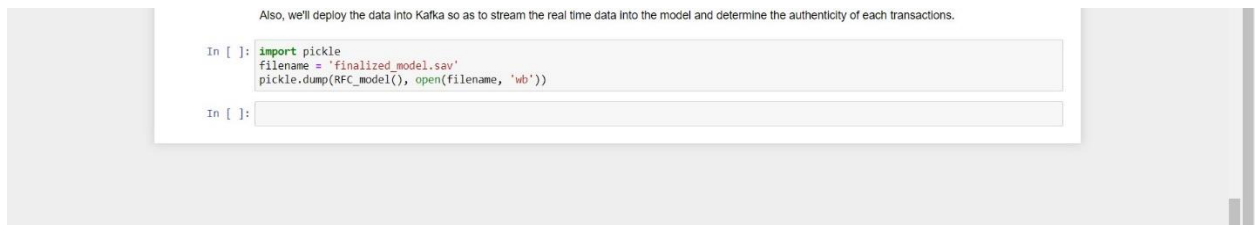


The screenshot shows a Jupyter Notebook interface with the title 'detection_model' and a status of '(autosaved)'. The browser address bar indicates the notebook is running on 'localhost:8888/notebooks/Downloads/detection_model.ipynb'. The notebook's menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar shows various icons for file operations and execution. The main content area is titled 'Stochastic Gradient Descent Classification (SDGClassification)'. It contains a code cell with the following Python code:

```
In [ ]: %%time
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import RandomizedSearchCV
sgd = SGDClassifier(loss="hinge", penalty="l2")
random = RandomizedSearchCV(estimator=sgd,
param_distributions=param_distributions,
scoring='roc_auc',
verbose=1, n_jobs=-1,
n_iter=1000)
random_result = random.fit(train_X, train_Y)
print('Best Score: ', random_result.best_score_)
print('Best Params: ', random_result.best_params_)
```

Below the code cell, the output shows 'Fitting 5 folds for each of 1000 candidates, totalling 5000 fits'. A text block explains that Random Forest Classification Technique after performing SMOTE has the highest rate of accuracy, i.e. the highest rate of determining fraudulent transactions accurately with the highest precision, recall and the highest f1-score. It also has the lowest run time for the machine. There are two empty input cells below the text. At the bottom, a text block states: 'Thus, we'll select Random Forest Classification Technique after the application of SMOTE sampling on the data, to classify the fraudulent transactions. the data into Kafka so as to stream the real time data into the model and determine the authenticity of each transactions.'

16. Making SMOTE applied Random Forest Classification as our detection model.



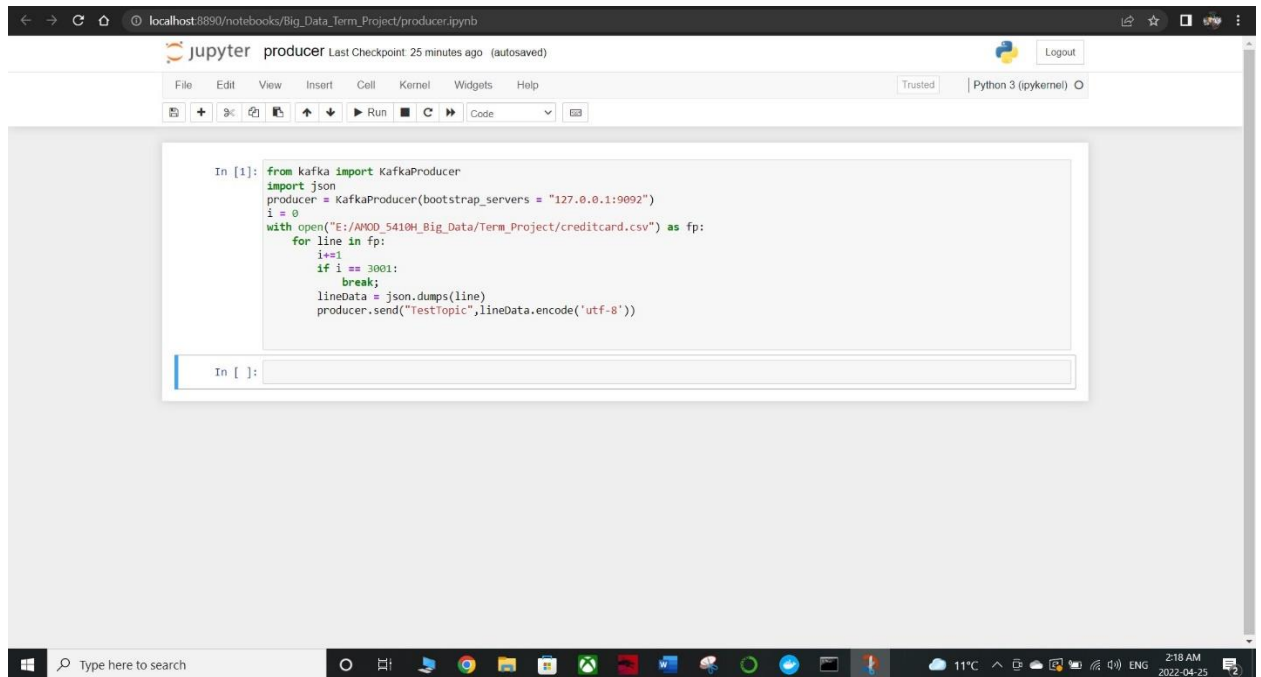
The screenshot shows a Jupyter Notebook interface with the title 'detection_model' and a status of '(autosaved)'. The browser address bar indicates the notebook is running on 'localhost:8888/notebooks/Downloads/detection_model.ipynb'. The notebook's menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar shows various icons for file operations and execution. The main content area is titled 'Stochastic Gradient Descent Classification (SDGClassification)'. It contains a code cell with the following Python code:

```
In [ ]: import pickle
filename = 'finalized_model.sav'
pickle.dump(RFC_model(), open(filename, 'wb'))
```

Below the code cell, there is an empty input cell. A text block at the bottom states: 'Also, we'll deploy the data into Kafka so as to stream the real time data into the model and determine the authenticity of each transactions.'

17. Opening Docker Desktop

19. Running Kafka Producer with the help of producer.ipynb

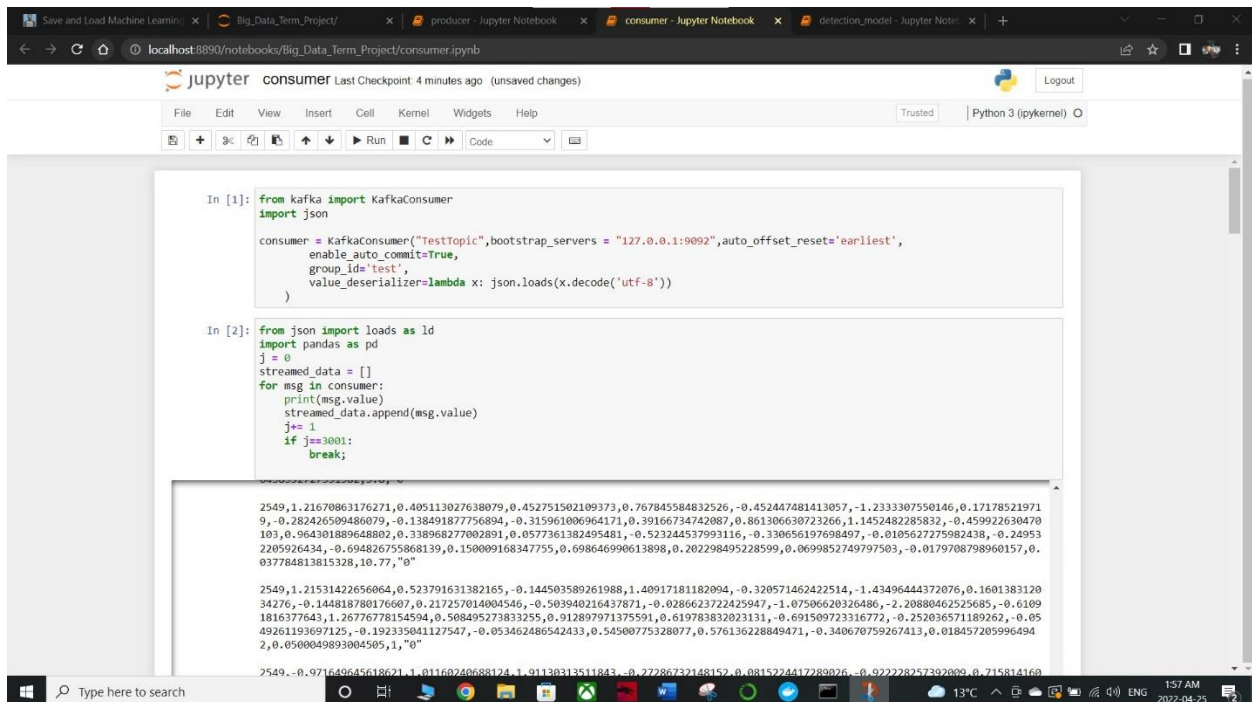


The screenshot shows a Jupyter Notebook interface with the title 'producer'. The code in the first cell is as follows:

```
In [1]: from kafka import KafkaProducer
import json
producer = KafkaProducer(bootstrap_servers = "127.0.0.1:9092")
i = 0
with open("E:/ANOD_5410H_Big_Data/Term_Project/creditcard.csv") as fp:
    for line in fp:
        i+=1
        if i == 3001:
            break;
        lineData = json.dumps(line)
        producer.send("TestTopic",lineData.encode('utf-8'))
```

The second cell is empty, showing 'In []:'.

20. Running kafka consumer with the help of consumer.ipynb in order to get the data as a stream from the kafka producer.



The screenshot shows a Jupyter Notebook interface with the title 'consumer'. The code in the first cell is as follows:

```
In [1]: from kafka import KafkaConsumer
import json

consumer = KafkaConsumer("TestTopic",bootstrap_servers = "127.0.0.1:9092",auto_offset_reset='earliest',
                           enable_auto_commit=True,
                           group_id='test',
                           value_deserializer=lambda x: json.loads(x.decode('utf-8'))
                           )
```

The second cell contains the following code:

```
In [2]: from json import loads as ld
import pandas as pd
j = 0
streamed_data = []
for msg in consumer:
    print(msg.value)
    streamed_data.append(msg.value)
    j+= 1
    if j==3001:
        break;
```

The output of the second cell shows a large stream of data, including a list of numbers and a string "0".

Save and Load Machine Learning | Big_Data_Term_Project/ | producer - Jupyter Notebook | consumer - Jupyter Notebook | detection_model - Jupyter Notebook | +

localhost:8890/notebooks/Big_Data_Term_Project/consumer.ipynb

jupyter consumer Last Checkpoint: 6 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O

7, -0.17606997854666, -0.745817581998954, -0.15166061639604, 0.366912021976969, 0.126128708100379, 0.0493163952666057, -0.297565487472599, 0.104088411704776, 0, "0"

In [3]: streamed_data

Out[3]: [{"Time", "V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12", "V13", "V14", "V15", "V16", "V17", "V18", "V19", "V20", "V21", "V22", "V23", "V24", "V25", "V26", "V27", "V28", "Amount", "Class"}]

In [4]: streamed_data[0]

Out[4]: [{"Time", "V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12", "V13", "V14", "V15", "V16", "V17", "V18", "V19", "V20", "V21", "V22", "V23", "V24", "V25", "V26", "V27", "V28", "Amount", "Class"}]

In [5]: strm_data_list = [elt.replace("\n", "") for elt in streamed_data]

Save and Load Machine Learning | Big_Data_Term_Project/ | producer - Jupyter Notebook | consumer - Jupyter Notebook | detection_model - Jupyter Notebook | +

localhost:8890/notebooks/Big_Data_Term_Project/consumer.ipynb

jupyter consumer Last Checkpoint: 7 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O

In [4]: streamed_data[0]

Out[4]: [{"Time", "V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12", "V13", "V14", "V15", "V16", "V17", "V18", "V19", "V20", "V21", "V22", "V23", "V24", "V25", "V26", "V27", "V28", "Amount", "Class"}]

In [5]: strm_data_list = [elt.replace("\n", "") for elt in streamed_data]

In [6]: strm_data_list

Out[6]: [{"Time", "V1", "V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12", "V13", "V14", "V15", "V16", "V17", "V18", "V19", "V20", "V21", "V22", "V23", "V24", "V25", "V26", "V27", "V28", "Amount", "Class"}]

Save and Load Machine Learning | Big_Data_Term_Project/ | producer - Jupyter Notebook | consumer - Jupyter Notebook | detection_model - Jupyter Notebook | +

localhost:8890/notebooks/Big_Data_Term_Project/consumer.ipynb

jupyter consumer Last Checkpoint: a few seconds ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O

In [7]: `df = pd.DataFrame(strm_data_list)`

In [8]: `df.head()`

Out[8]:

	0
0	"Time","V1","V2","V3","V4","V5","V6","V7"
1	0 -1.3598071336738 -0.0727811733098497 2.53634673796914 1.37815522427443 -0.338320769942518 0.462387777762292 0.239598554061257 0.06
2	0 1.19185711131486 0.26615071205963 0.16648011335321 0.448154078460911 0.0600176492822243 -0.0823608088155687 -0.078802983323113 0.08
3	1 -1.35835406159823 -1.34016307473609 1.77320934263119 0.379779593034328 -0.503198133318193 1.80049938079263 0.791460956450422 0.2
4	1 -0.996271711572087 -0.185226008082898 1.79299333957872 -0.863291275036453 -0.0103088796030823 1.24720316752486 0.23760893977178 0.3

In [9]: `df1 = df[0].str.split(',', expand = True)`

In [10]: `df1.head()`

Out[10]:

	0	1	2	3	4	5	6	7
0	"Time"	"V1"	"V2"	"V3"	"V4"	"V5"	"V6"	"V7"
1	0	-1.3598071336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.338320769942518	0.462387777762292	0.239598554061257
2	0	1.19185711131486	0.26615071205963	0.16648011335321	0.448154078460911	0.0600176492822243	-0.0823608088155687	-0.078802983323113
3	1	-1.35835406159823	-1.34016307473609	1.77320934263119	0.379779593034328	-0.503198133318193	1.80049938079263	0.791460956450422
4	1	-0.996271711572087	-0.185226008082898	1.79299333957872	-0.863291275036453	-0.0103088796030823	1.24720316752486	0.23760893977178

5 rows x 31 columns

In [11]: `for i, col in enumerate(df1.columns):
df1.iloc[:, i] = df1.iloc[:, i].str.replace('\"', '')`

Save and Load Machine Learning | Big_Data_Term_Project/ | producer - Jupyter Notebook | consumer - Jupyter Notebook | detection_model - Jupyter Notebook | +

localhost:8890/notebooks/Big_Data_Term_Project/consumer.ipynb

jupyter consumer Last Checkpoint: a few seconds ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O

In [11]: `for i, col in enumerate(df1.columns):
df1.iloc[:, i] = df1.iloc[:, i].str.replace('\"', '')`

In [12]: `df1.head()`

Out[12]:

	0	1	2	3	4	5	6	7
0	Time	V1	V2	V3	V4	V5	V6	V7
1	0	-1.3598071336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.338320769942518	0.462387777762292	0.239598554061257
2	0	1.19185711131486	0.26615071205963	0.16648011335321	0.448154078460911	0.0600176492822243	-0.0823608088155687	-0.078802983323113
3	1	-1.35835406159823	-1.34016307473609	1.77320934263119	0.379779593034328	-0.503198133318193	1.80049938079263	0.791460956450422
4	1	-0.996271711572087	-0.185226008082898	1.79299333957872	-0.863291275036453	-0.0103088796030823	1.24720316752486	0.23760893977178

5 rows x 31 columns

In [13]: `df1.columns = df1.iloc[0]
df1 = df1[1:]
df1.head()`

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	V7
1	0	-1.3598071336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.338320769942518	0.462387777762292	0.239598554061257
2	0	1.19185711131486	0.26615071205963	0.16648011335321	0.448154078460911	0.0600176492822243	-0.0823608088155687	-0.078802983323113
3	1	-1.35835406159823	-1.34016307473609	1.77320934263119	0.379779593034328	-0.503198133318193	1.80049938079263	0.791460956450422
4	1	-0.996271711572087	-0.185226008082898	1.79299333957872	-0.863291275036453	-0.0103088796030823	1.24720316752486	0.23760893977178
5	2	-1.15823309349523	0.877736754848451	1.548717846511	0.403033933955121	-0.407193377311653	0.0959214624864256	0.592940745386545

5 rows x 31 columns

jupyter consumer Last Checkpoint: a minute ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [14]: df1.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 1 to 3000
Data columns (total 31 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Time        3000 non-null   object
1   V1          3000 non-null   object
2   V2          3000 non-null   object
3   V3          3000 non-null   object
4   V4          3000 non-null   object
5   V5          3000 non-null   object
6   V6          3000 non-null   object
7   V7          3000 non-null   object
8   V8          3000 non-null   object
9   V9          3000 non-null   object
10  V10         3000 non-null   object
11  V11         3000 non-null   object
12  V12         3000 non-null   object
13  V13         3000 non-null   object
14  V14         3000 non-null   object
15  V15         3000 non-null   object
16  V16         3000 non-null   object
17  V17         3000 non-null   object
18  V18         3000 non-null   object
19  V19         3000 non-null   object
20  V20         3000 non-null   object
21  V21         3000 non-null   object
22  V22         3000 non-null   object
23  V23         3000 non-null   object
24  V24         3000 non-null   object
25  V25         3000 non-null   object
26  V26         3000 non-null   object
27  V27         3000 non-null   object
28  V28         3000 non-null   object
29  Amount      3000 non-null   object
```

Type here to search 11°C 2:16 AM 2022-04-25

jupyter consumer Last Checkpoint: a minute ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [15]: df1.dtypes
Out[15]:
Time        object
V1          object
V2          object
V3          object
V4          object
V5          object
V6          object
V7          object
V8          object
V9          object
V10         object
V11         object
V12         object
V13         object
V14         object
V15         object
V16         object
V17         object
V18         object
V19         object
V20         object
V21         object
V22         object
V23         object
V24         object
V25         object
V26         object
V27         object
V28         object
Amount      object
Class       object
dtype: object

In [1]: v = df1['Class']
```

Type here to search 11°C 2:17 AM

localhost:8890/notebooks/Big_Data_Term_Project/consumer.ipynb

jupyter consumer Last Checkpoint: 2 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In []:

y = df1['Class']
X = df1.drop(['Class'], axis=1)

In []:

y.head()

In [33]:

X.head()

Out[33]:

	Time	V1	V2	V3	V4	V5	V6	V7	
1	0	-1.3598071336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.338320769942518	0.4623877777762292	0.239598554061257	0.098
2	0	1.19185711131486	0.26615071205963	0.16648011335321	0.448154078460811	0.0600176492822243	-0.0823608088155687	-0.0768029833323113	0.085
3	1	-1.35835406159823	-1.34016307473609	1.77320934293119	0.379779593034328	-0.503198133318193	1.80049939079263	0.791460959450422	0.24
4	1	-0.996271711572087	-0.185226008062898	1.79299333957872	-0.883291275039453	-0.0103088799030823	1.24720310752486	0.23760893977178	0.37
5	2	-1.15823309349523	0.877736754848451	1.548717846511	0.403033933955121	-0.407193377311653	0.0959214624684256	0.592940745385545	-0.27

5 rows × 30 columns

Now we'll deploy the selected classification technique i.e. Random Forest Classification after the application of SMOTE as the Fraud Detection Model for this data.

In [34]:

import pickle
loaded_model = pickle.load(open('finalized_model.sav', 'rb'))

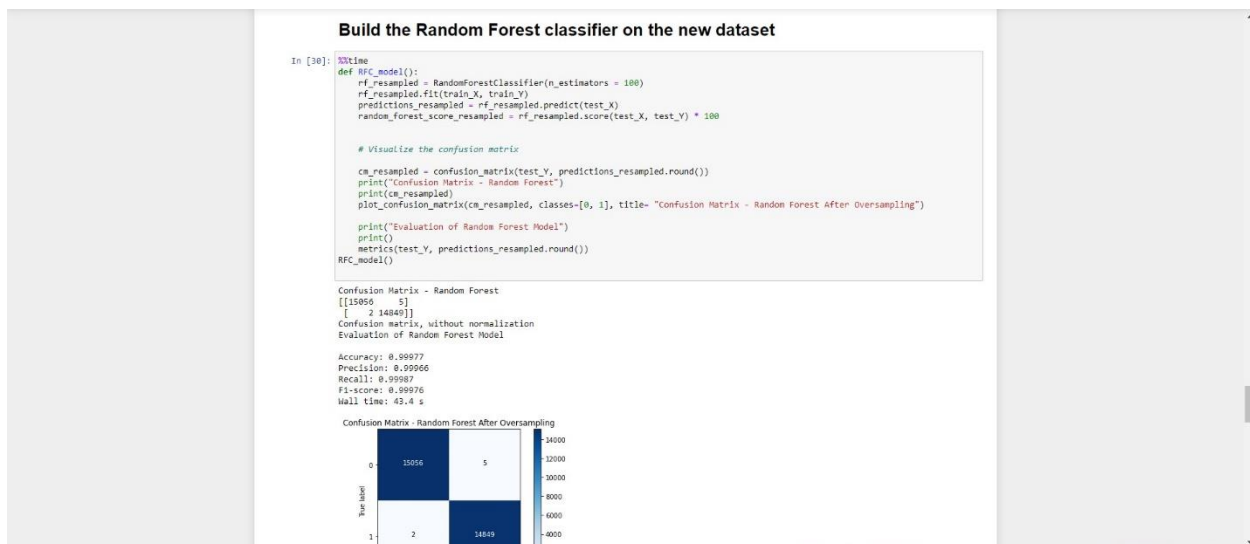
In [35]:

result = loaded_model.score(X, y)

Results

From the above classification models used and various types of tools used we can clearly observe that SMOTE applied Random Forest Classification technique has the highest accuracy of 99.99% and thus we'll use this method to build a model. Following are the accuracy and prediction scores of all the models implemented.

21. Random Forest Classifier Scores



22. Support Vector Machine Scores

Support Vector Machine (SVM) classification

```
In [31]: # importing libraries for SVM
from sklearn.svm import LinearSVC, LinearSVR, SVC, SVR
```

```
In [32]: %%time
svm = SVC()
svm.fit(train_X, train_Y)

prediction_svm = svm.predict(test_X)
svm_score = svm.score(test_X, test_Y)*100

Wall time: 1min 6s
```

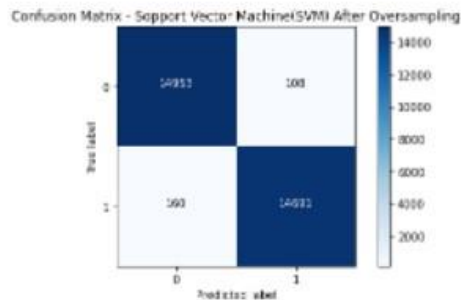
```
In [33]: # Visualize the confusion matrix

cm_resampled_svm = confusion_matrix(test_Y, prediction_svm.round())
print("Confusion Matrix - Random Forest")
print(cm_resampled_svm)

Confusion Matrix - Random Forest
[[14953  108]
 [ 160 14691]]
```

```
In [35]: plot_confusion_matrix(cm_resampled_svm, classes=[0, 1], title= "Confusion Matrix - Support Vector Machine(SVM) After Oversam
```

Confusion matrix, without normalization



```
In [36]: print("Evaluation of SVM Model")
print()
metrics(test_Y, prediction_svm.round())

Evaluation of SVM Model

Accuracy: 0.99104
Precision: 0.99270
Recall: 0.98923
F1-score: 0.99096
```

23. Logistic Regression Classification Result

F1-score: 0.99096

Logistic Regression Classification

```
In [38]: # importing library for Logistic Regression model
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import GridSearchCV
```

Applying Hyper Parameter Tuning to perform Logistic Regression Classification using Grid Search CV

```
In [39]: %%time
logistic = LogisticRegression()

alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
learning_rate = ['constant', 'optimal', 'invscaling', 'adaptive']
class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
eta0 = [1, 10, 100]
penalty = ['l1', 'l2']
C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
solver = ['liblinear', 'saga']
param_grid = dict(penalty=penalty,
                  C=C,
                  class_weight=class_weight,
                  solver=solver)

grid = GridSearchCV(estimator=logistic,
                    param_grid=param_grid,
                    scoring='roc_auc',
                    verbose=1,
                    n_jobs=-1)
grid_result = grid.fit(train_X, train_Y)
print('Best Score: ', grid_result.best_score_)
print('Best Params: ', grid_result.best_params_)

Fitting 5 folds for each of 128 candidates, totalling 640 fits
Best Score: 0.9986357946530464
Best Params: {'C': 100, 'class_weight': {1: 0.7, 0: 0.3}, 'penalty': 'l1', 'solver': 'liblinear'}
Wall time: 5min 5s
```

Applying Hyper Parameter Tuning to perform Stochastic Gradient Descent Classification using Random Search CV

24. Stochastic Gradient Descent Classification

```
class_weight=class_weight,  
eta0=eta0)
```

Stochastic Gradient Descent Classification (SGDClassification)

```
In [41]: %%time  
from sklearn.linear_model import SGDClassifier  
from sklearn.model_selection import RandomizedSearchCV  
sgd = SGDClassifier(loss="hinge", penalty="l2")  
random = RandomizedSearchCV(estimator=sgd,  
param_distributions=param_distributions,  
scoring='roc_auc',  
verbose=1, n_jobs=-1,  
n_iter=1000)  
random_result = random.fit(train_X, train_Y)  
print('Best Score: ', random_result.best_score_)  
print('Best Params: ', random_result.best_params_)  
  
Fitting 5 folds for each of 1000 candidates, totalling 5000 fits  
Best Score: 0.9986679431941579  
Best Params: {'penalty': 'l2', 'loss': 'modified_huber', 'learning_rate': 'adaptive', 'eta0': 10, 'class_weight': {1: 0.4, 0:  
0.6}, 'alpha': 0.0001}  
Wall time: 19min 37s
```

From the above classification models we can observe that Random Forest Classification Technique after performing SMOTE has the highest rate of accuracy, i.e. the highest rate of determining fraudulent transactions accurately with the highest precision, recall and the highest f1-score. It also has the lowest run time for the machine.

In []:

Thus, we'll select Random Forest Classification Technique after the application of SMOTE sampling on the data, to classify the fraudulent transactions. Also, we'll deploy the data into Kafka so as to stream the real time data into the model and determine the authenticity of each transactions.

```
In [42]: import pickle  
filename = 'finalized_model.sav'  
pickle.dump(RFC_model(), open(filename, 'wb'))
```

We first implemented Random Forest Classification without applying SMOTE technique, and then after applying SMOTE technique. After the comparison between the accuracies, we can clearly observe that the accuracy got increased after applying SMOTE technique. Here, as the dataset was under-sampled, and which it should be, we need to apply SMOTE technique for the same. We also applied different types of classification techniques like SVM, Logistic Regression and SGDClassification. We used docker to fire kafka. Kafka is a stream processing tool used for streaming real-time data and is useful to get the data from various types of databases and different types of datasources. Kafka uses a **pub-sub** model where publisher publishes a data as stream and subscriber uses that data as a streaming data to the kafka clusters and kafka consumers, where it can fitted onto the model and predicts the results which can then be further be fitted into the model so as to train them. Thus, kafka allows the continuous updation in the model through the means of streaming real time data into kafka clusters. Thus, kafka is useful with the ever evolving data with lots and lots of data coming into system through the means of different streams.

Future Works

In this project we used Random Forest Classifier as the classification model and used kafka as a streaming data tool, so as to check whether the model runs successfully with using kafka. We implemented this whole on the local device, rather than on a dynamic cluster present on any cloud services. Thus, for future implementation, we can deploy our model on any cloud platform (Microsoft Azure Blob) to be precise, and then would try to run the model by getting data from various data sources feeded into the model as streams through the means of kafka and spark. As the data is ever evolving, we need to update the model continuously by testing the model on the data and then feeding the same results into the model to train the model once again. We can also use another type of classification technique like Isolation Forest Regression to find out the fraudulent data out of the genuine data.

Conclusion

Thus, from our project we can conclude that for End-to-End Credit Card Fraud detection system, Random Forest Classification has the highest accuracy and can be used to implement a model. Kafka is very helpful in the system as it helps in **to build real-time streaming data pipelines and real-time streaming applications.**