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.

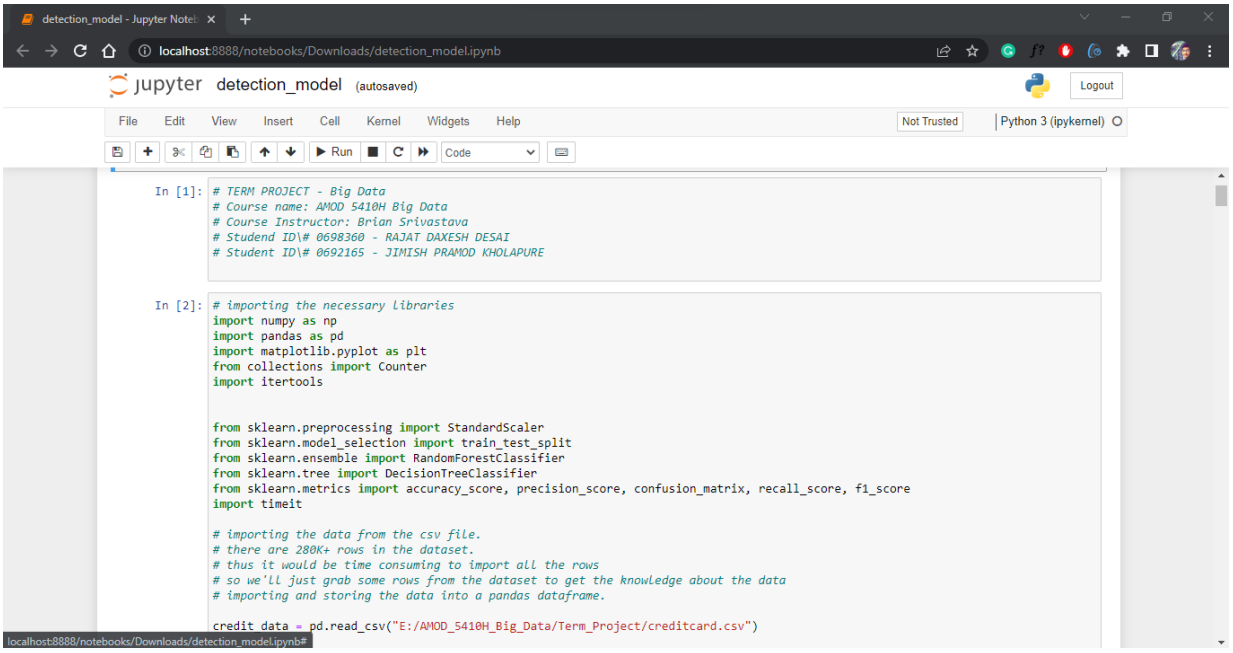
To download: https://www.python.org/

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.

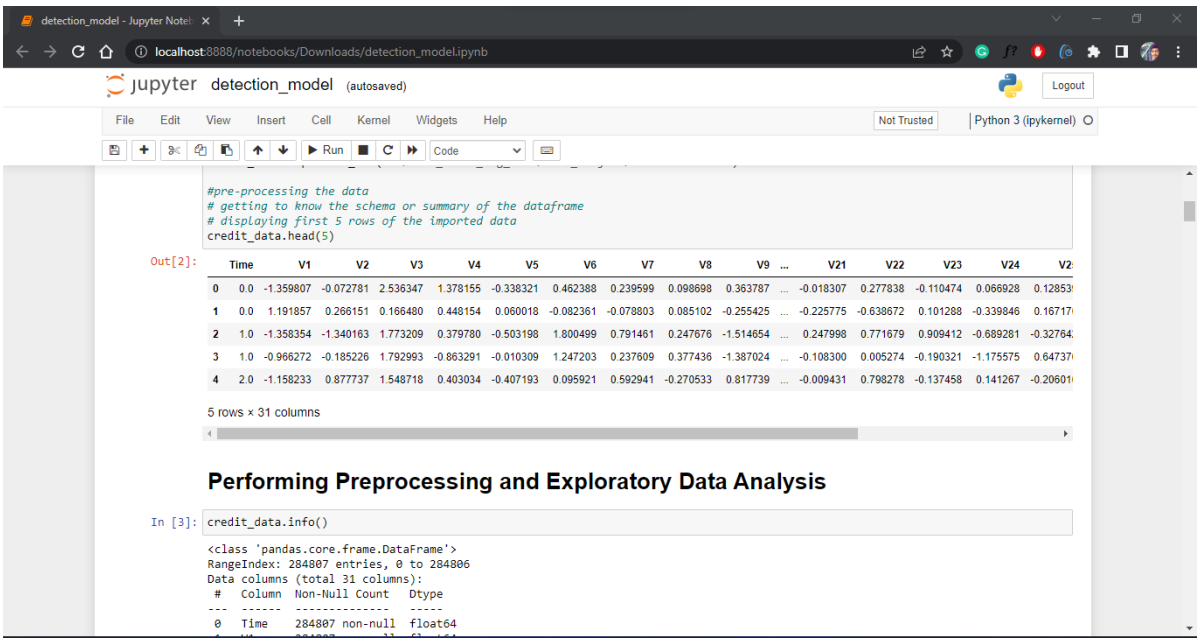
To download: https://jupyter.org/

Implementation

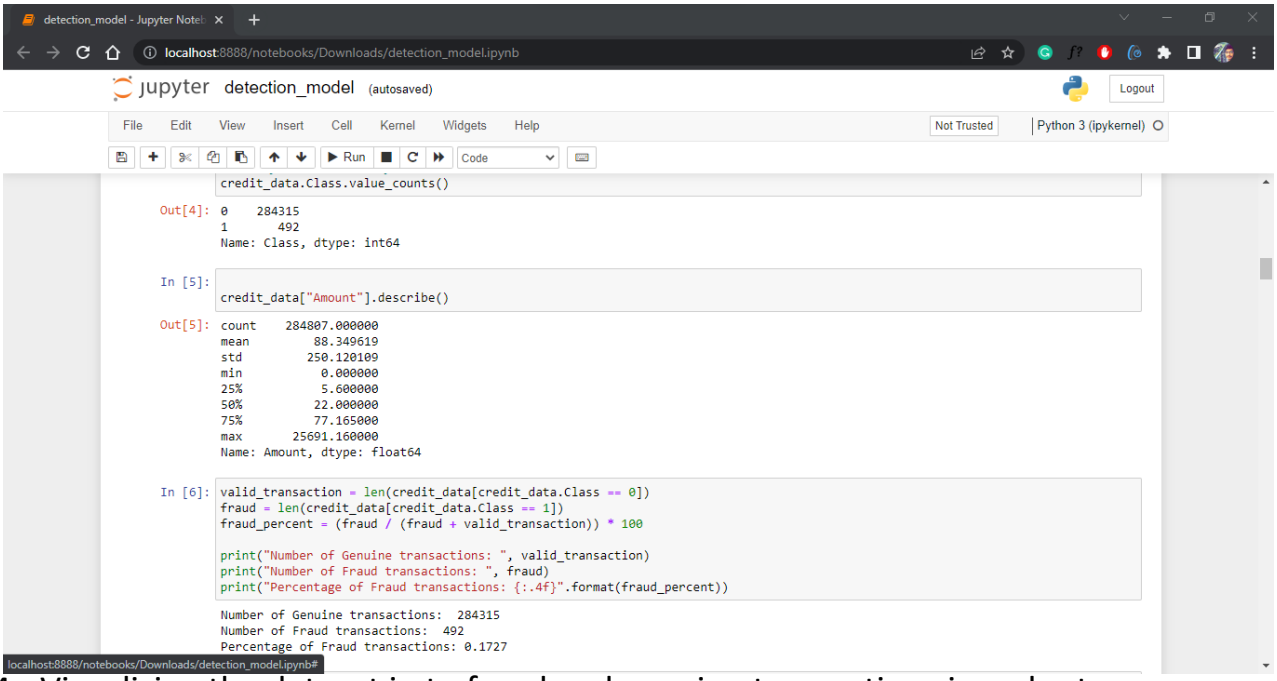
1. Imported the necessary libraries for the code implementation along with the selected dataset. Here, we have used Jupyter notebook as the python IDE.



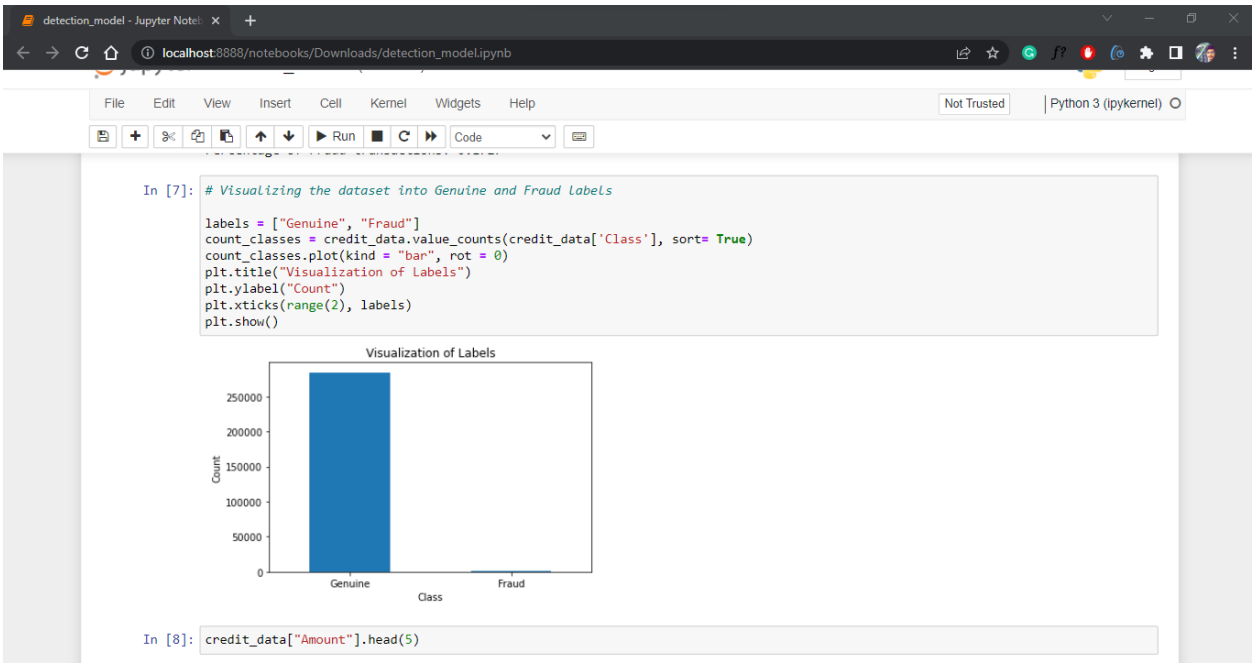
2. Pre-processing the dataset to clean the data from noise and unwanted data items that would cause error or misguide the results.



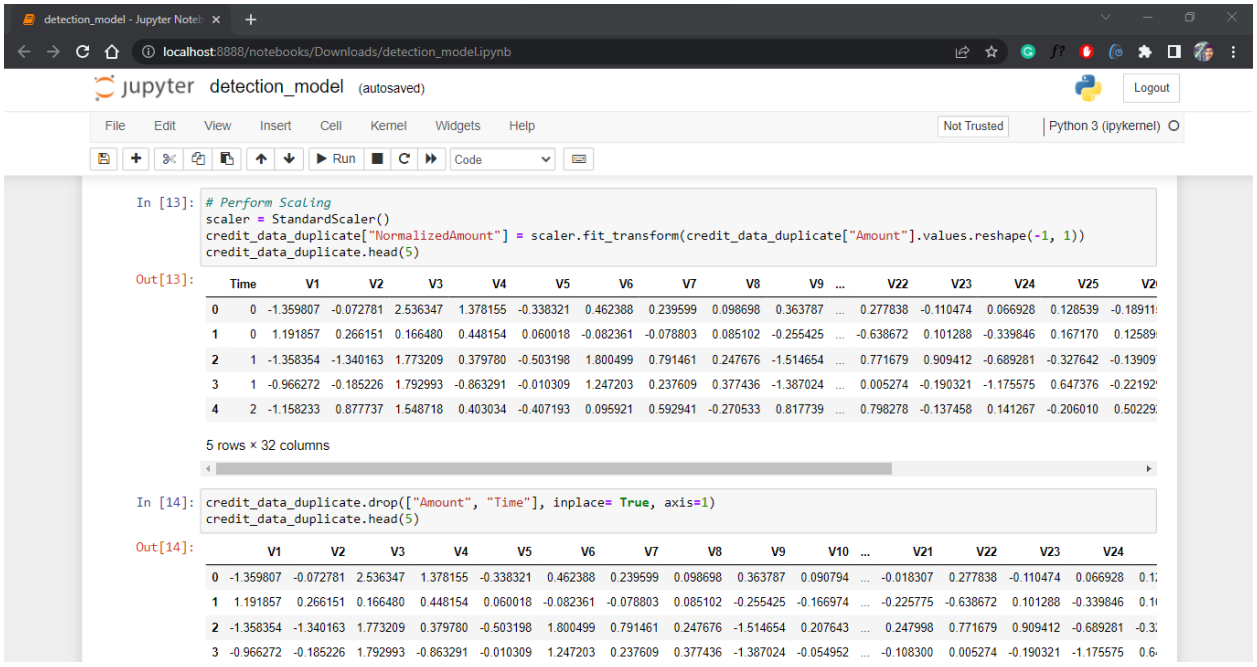
3. Getting the summary of the dataset



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



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



detection_model - Jupyter Notebook

localhost:8888/notebooks/Downloads/detection_model.ipynb

jupyter detection_model (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

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)
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V22	V23	V24	V25	V2
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	-0.1891
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	0.1258
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	-0.1390
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	-0.2219
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	0.5022

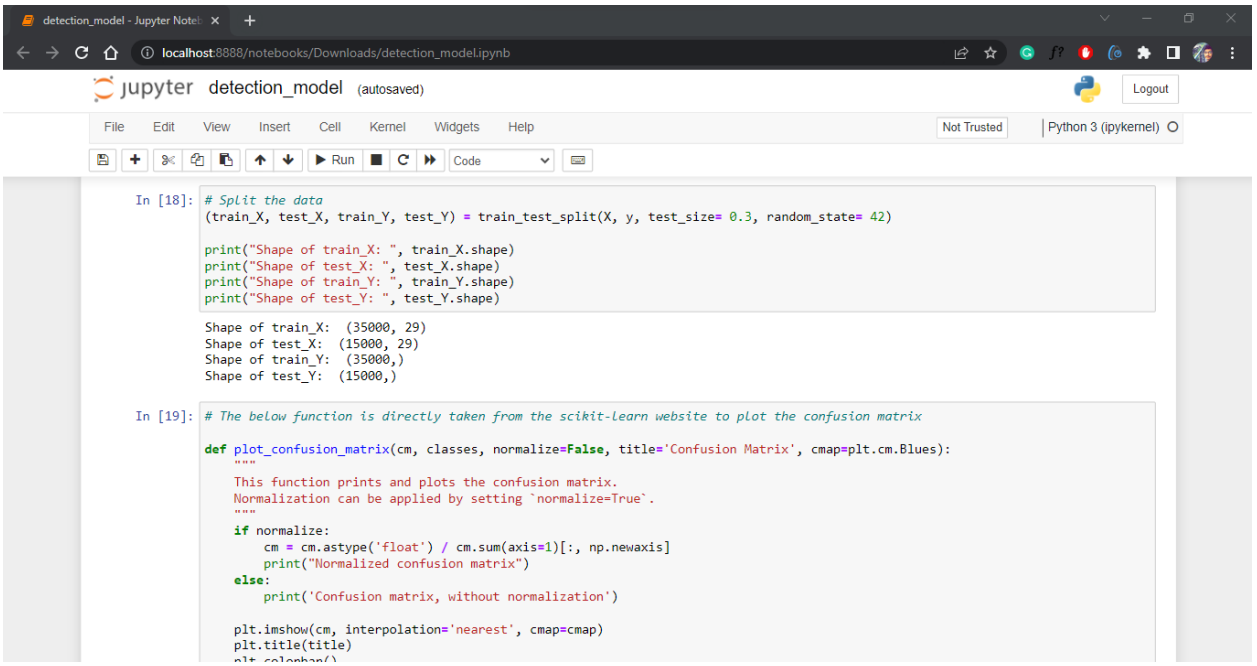
5 rows × 32 columns

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

Out[14]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	
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	0.1
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	0.1
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.3
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	0.6

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



detection_model - Jupyter Notebook

localhost:8888/notebooks/Downloads/detection_model.ipynb

jupyter detection_model (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

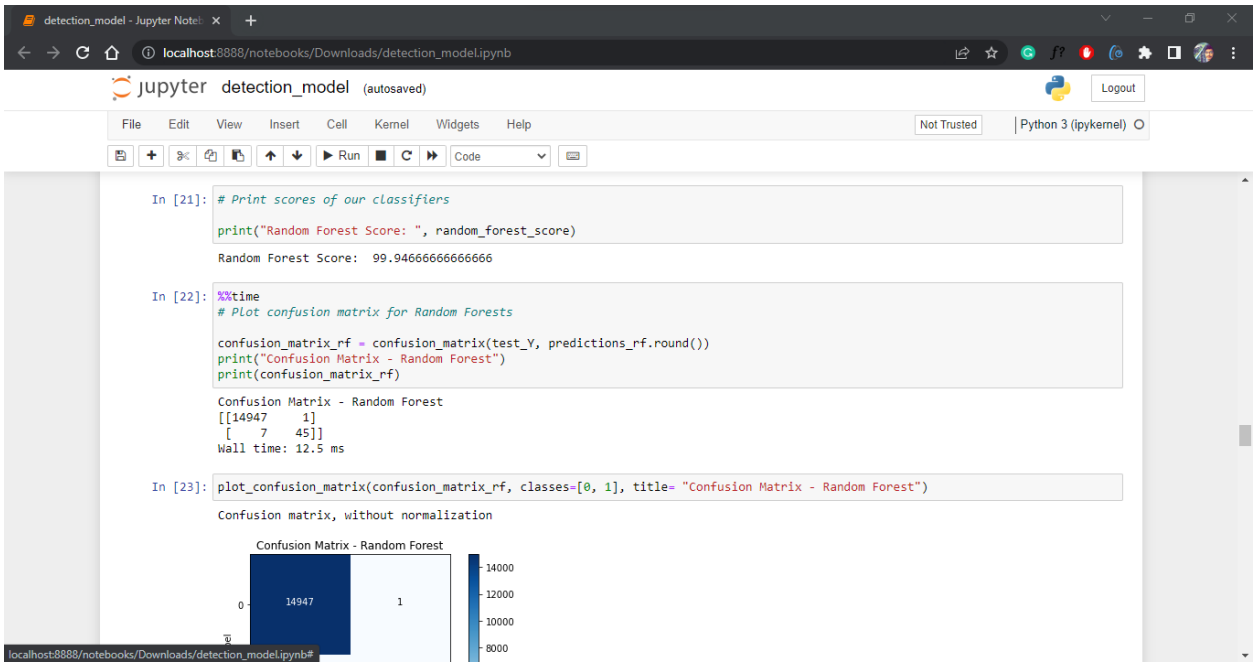
Run Code

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)
```

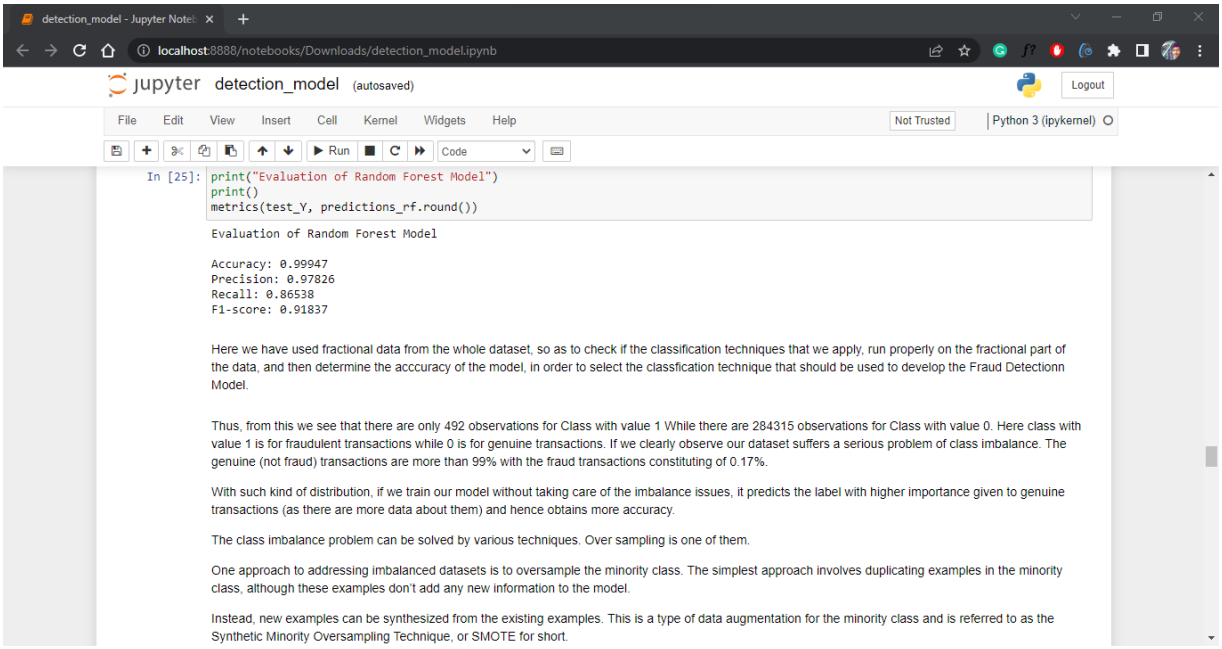
```
Shape of train_X:  (35000, 29)
Shape of test_X:  (15000, 29)
Shape of train_Y:  (35000,)
Shape of test_Y:  (15000,)
```

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()
```

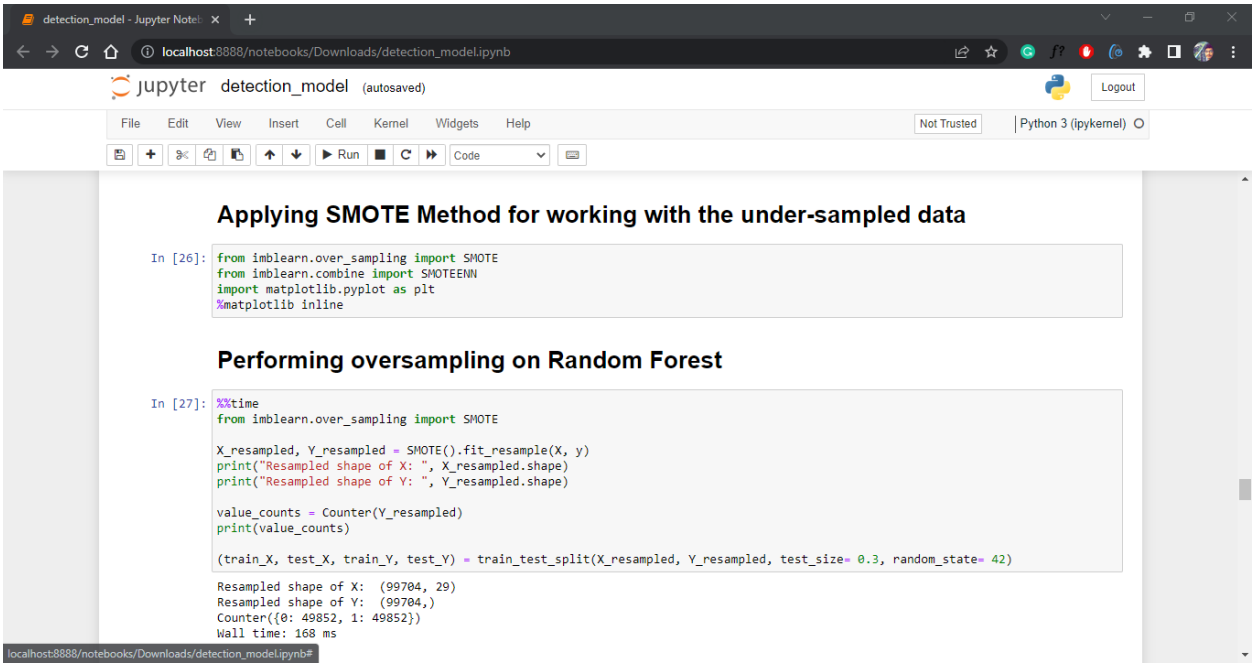
7. Checking the score for the random forest and ploting the confusion matrix for the same.



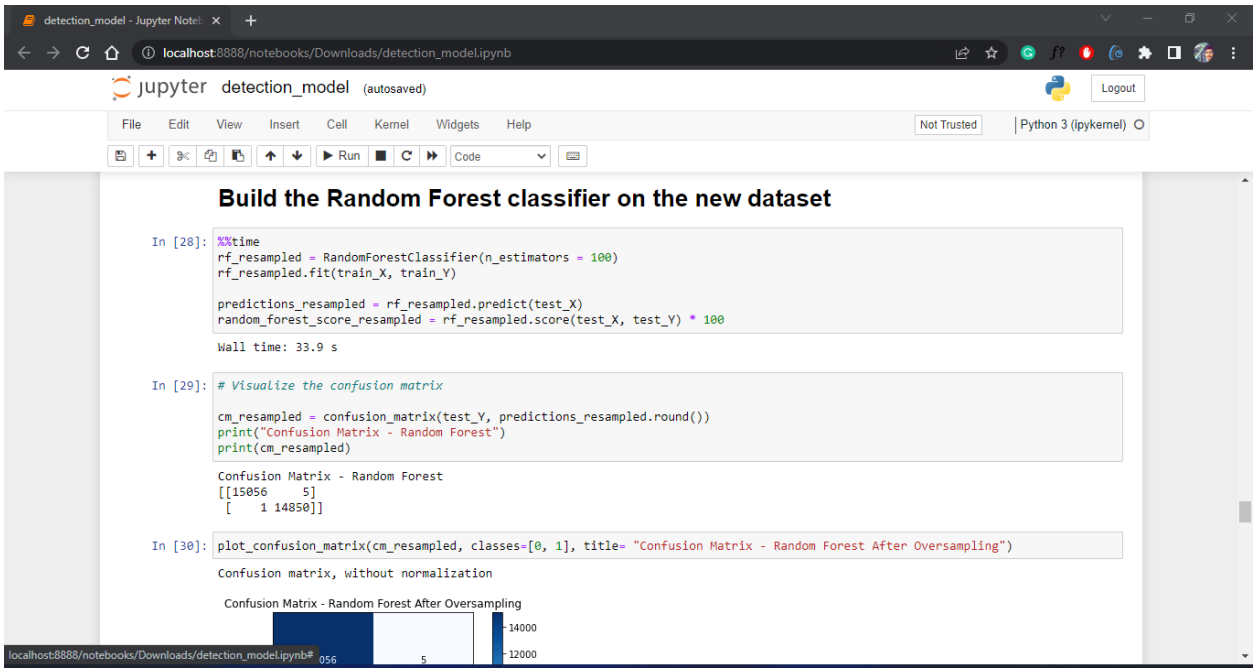
8. Evaluation of Random Forest Classifier and the writeup for the same



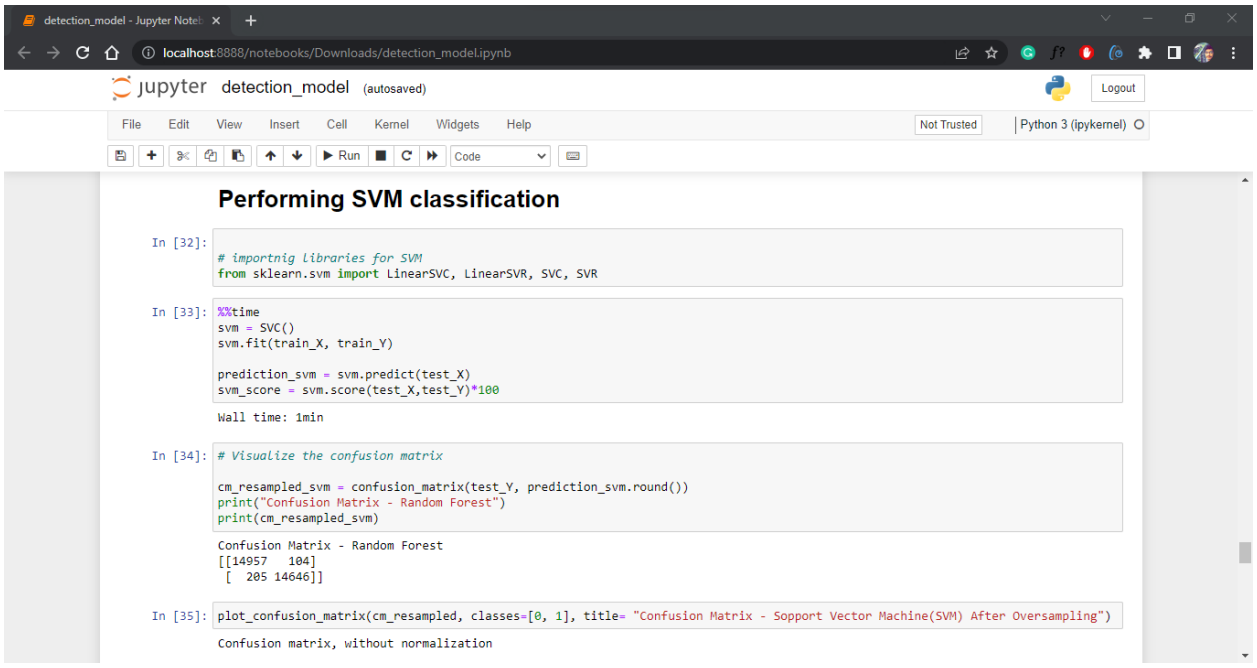
9. Appling SMOTE (Synthetic Minority Oversampling technique)for the under-sampled data to balance to dataset in proper manner



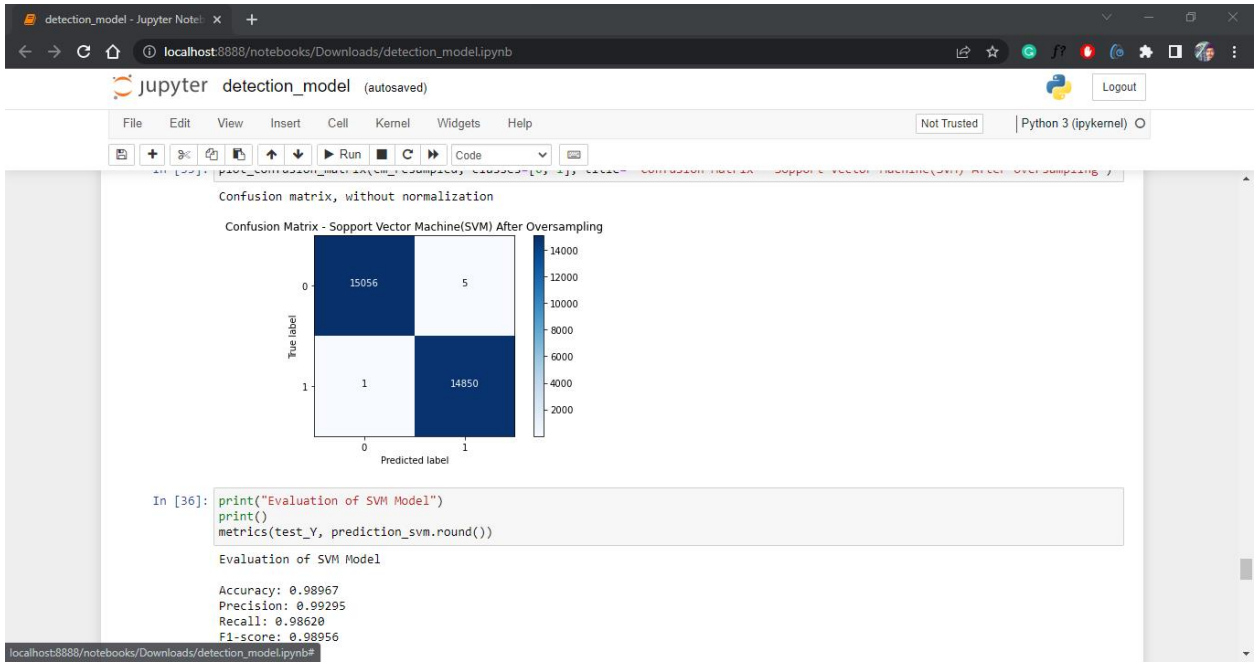
10. Building Random Forest Classifier on the New Dataset, plotting the visualisation and confusion matrix.



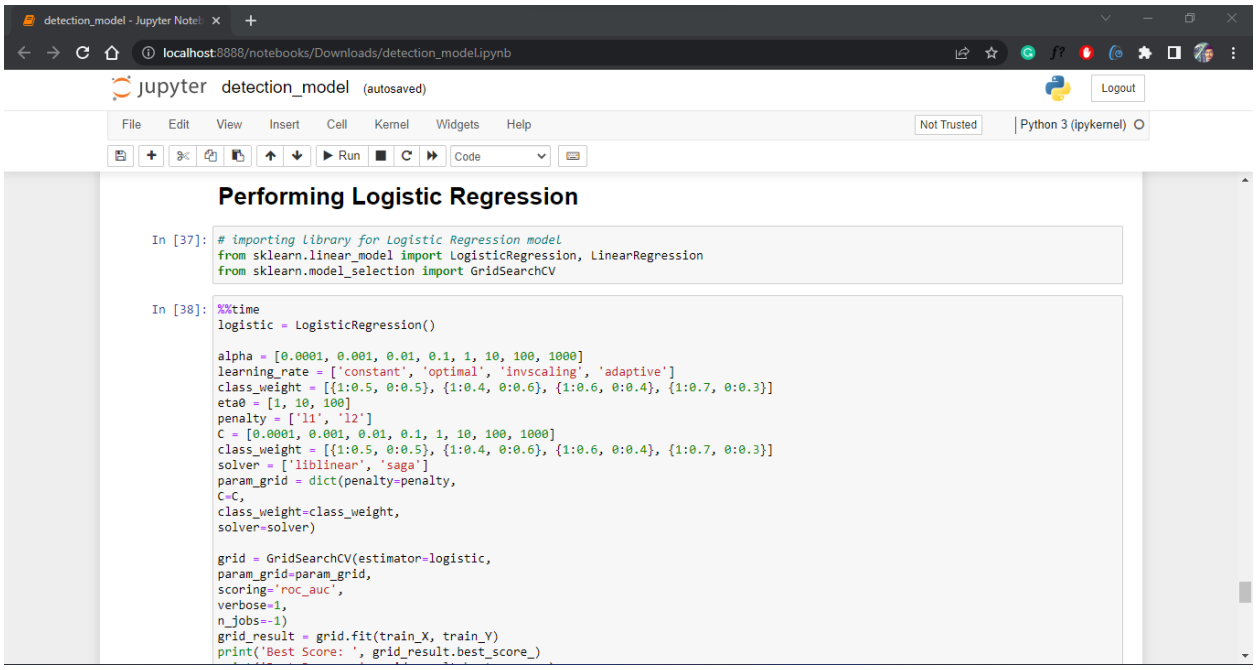
11.Performing SVM (Support Vector Machine) on the dataset



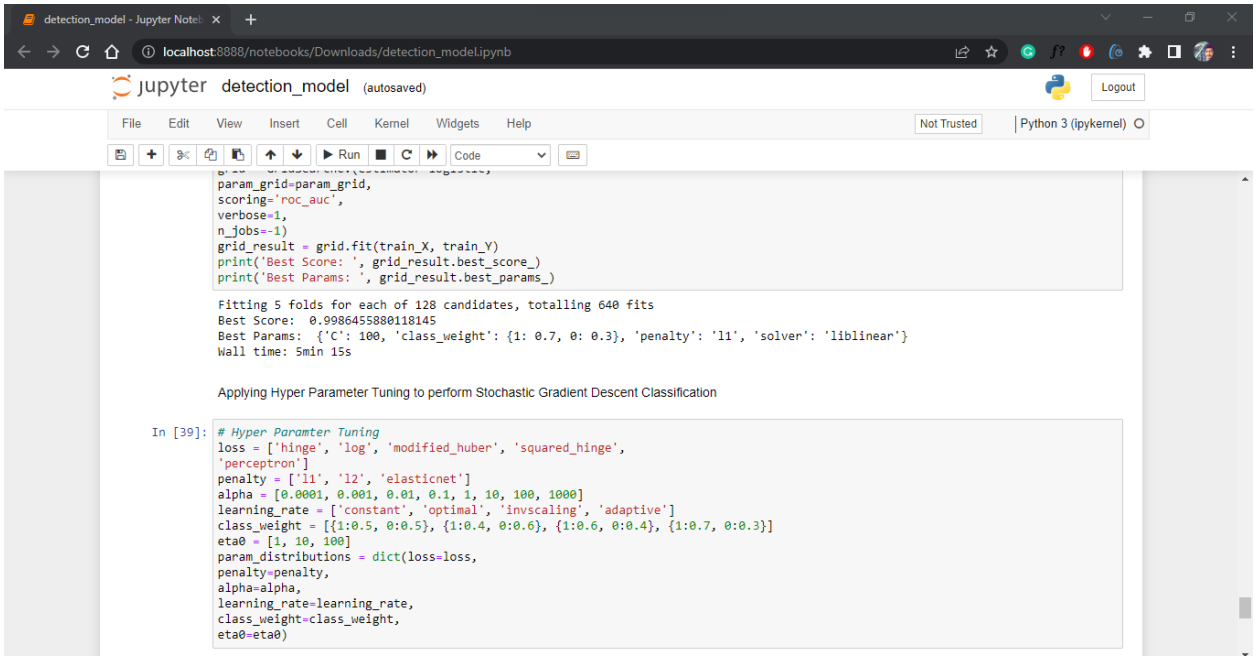
12.Evaluation of Support Vector Machine and confusion matrix.



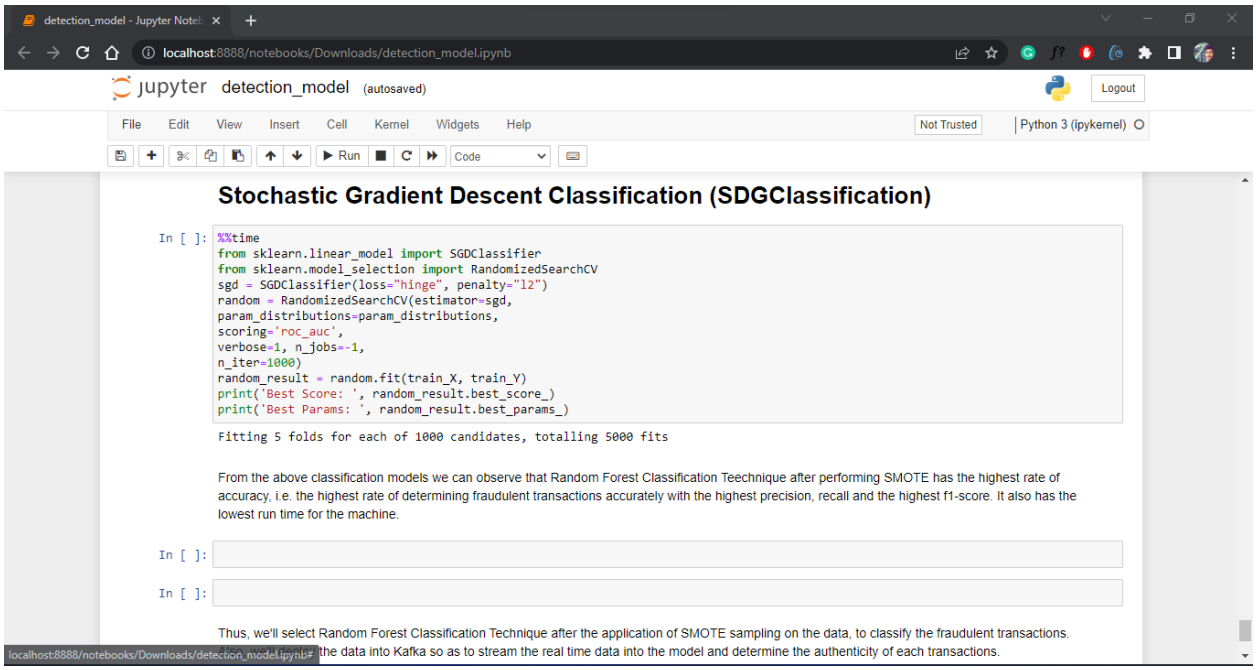
13.Performing Logistic Regression on the dataset



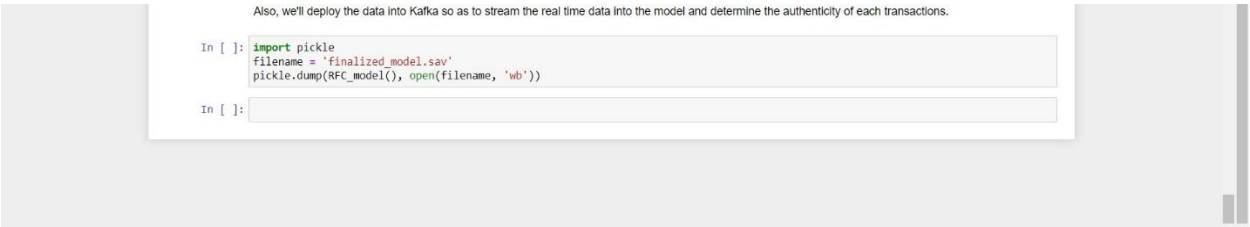
14. Applying Hyper Parameter Tuning to perform Stocastic Gradient Descent Classification.



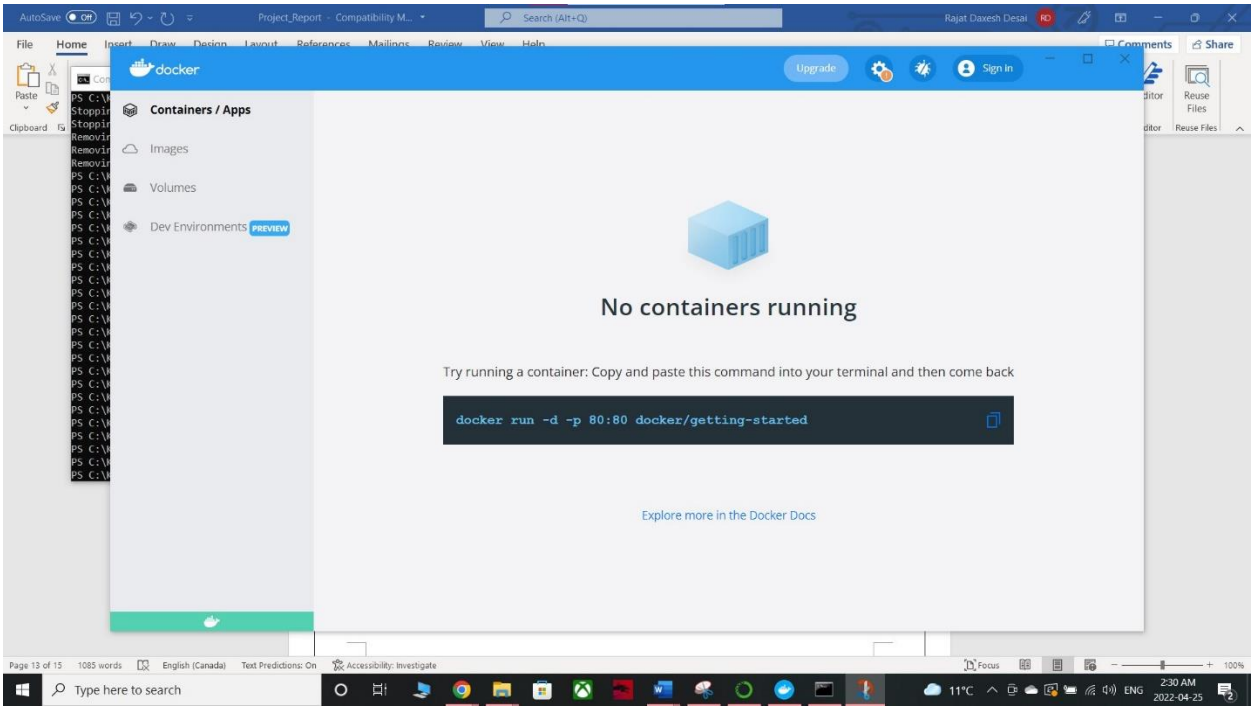
15.Performing Stocastic Gradient Descent Classification (SDGClassification)



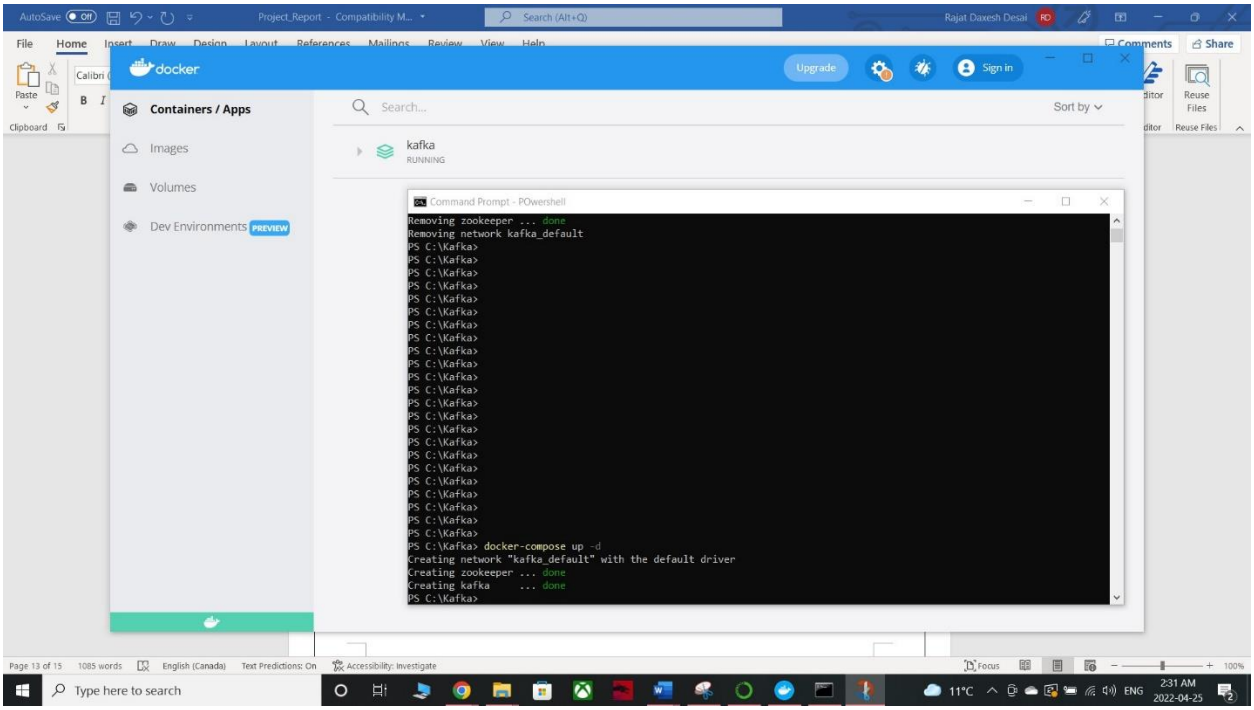
16. Making SMOTE applied Random Forest Classification as our detection model.



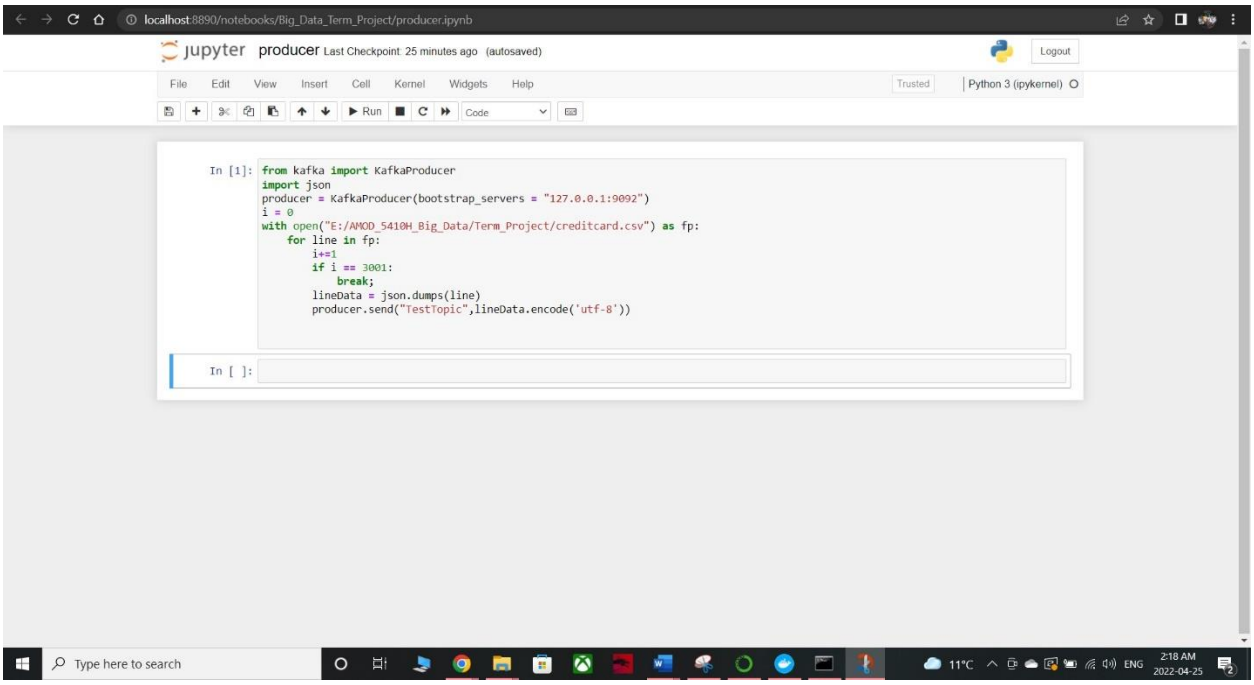
17. Opening Docker Desktop



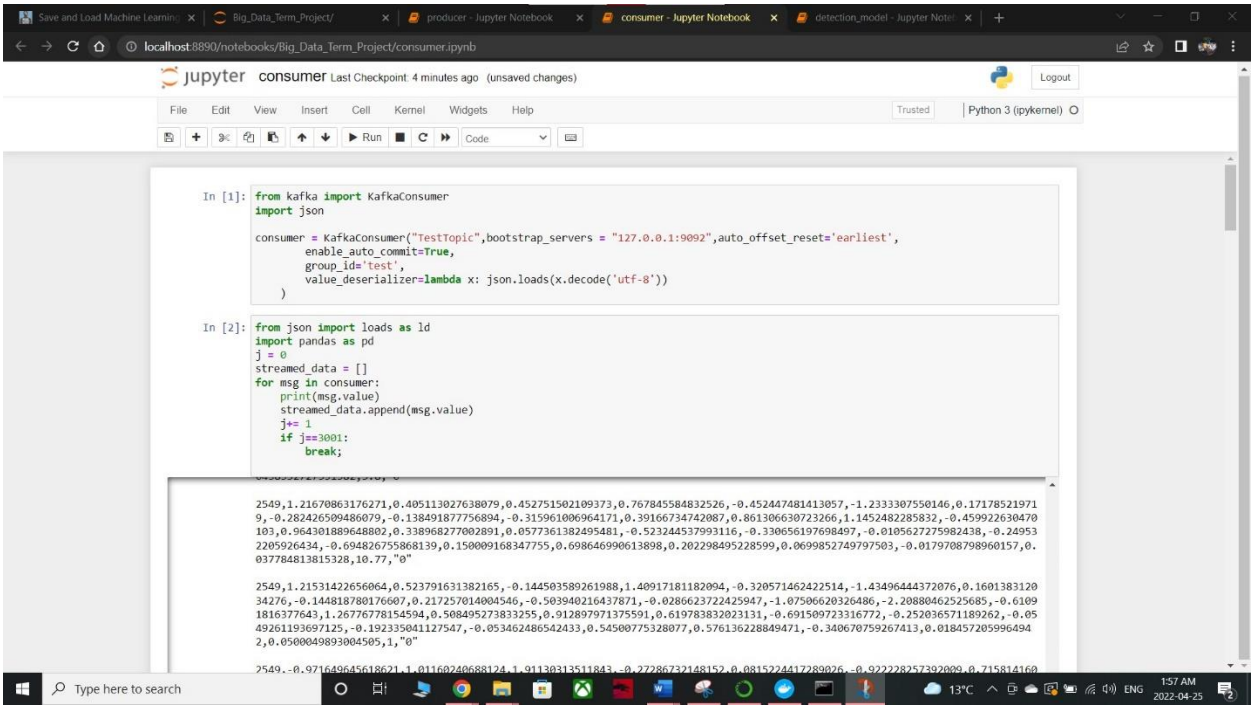
18. Running a docker-compose.yml file in the terminal to get the kafka running on the docker container.

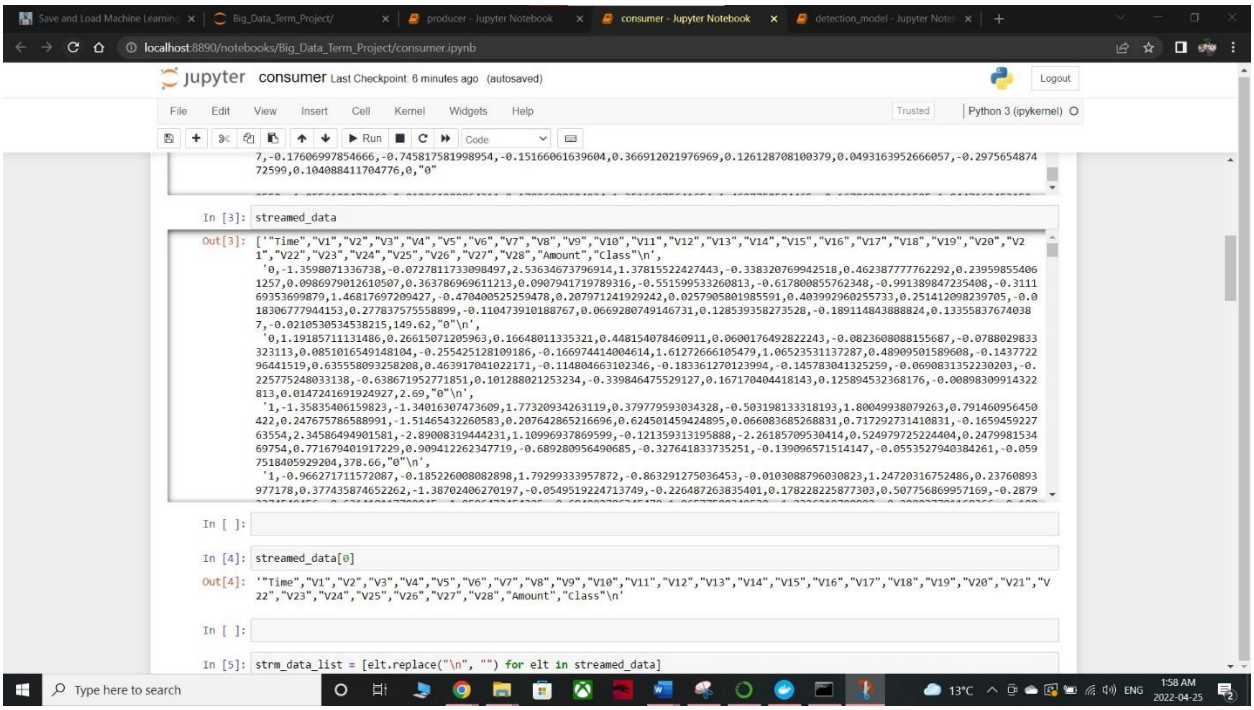


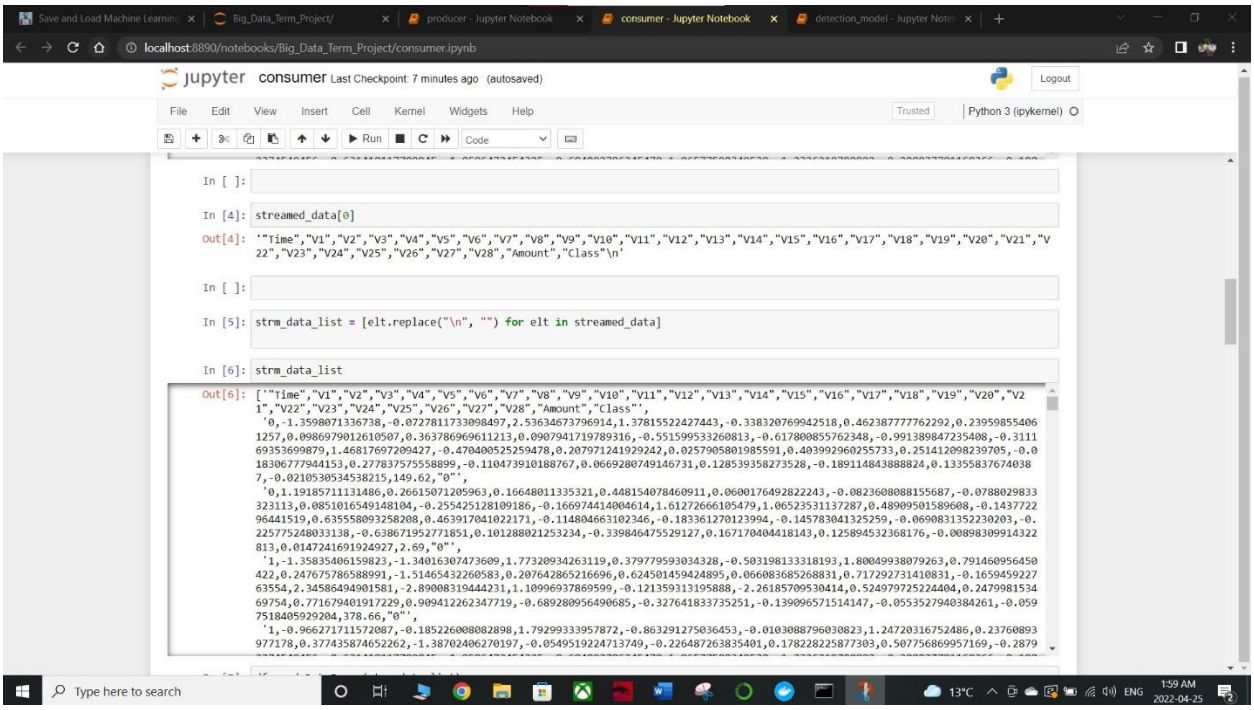
19. Running Kafka Producer with the help of producer.ipynb

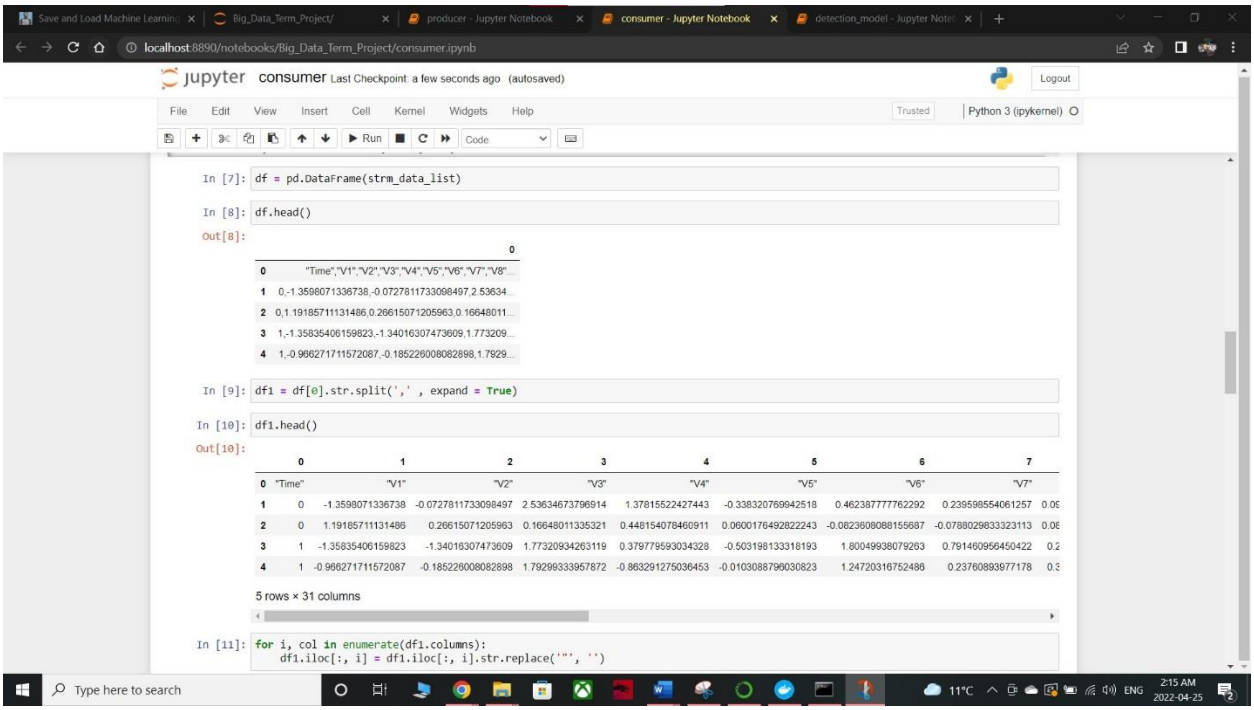


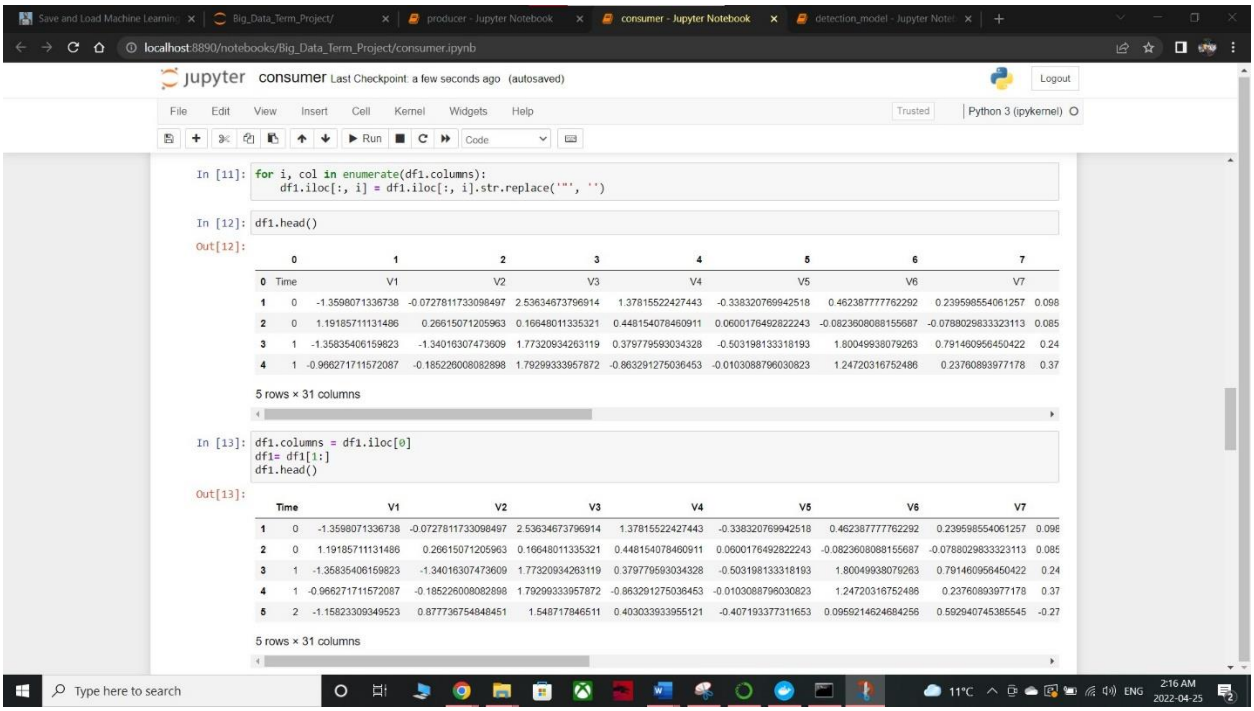
20. Running kafka consumer with the help of consumer.ipynb in order to get the data as a stream from the kafka producer.

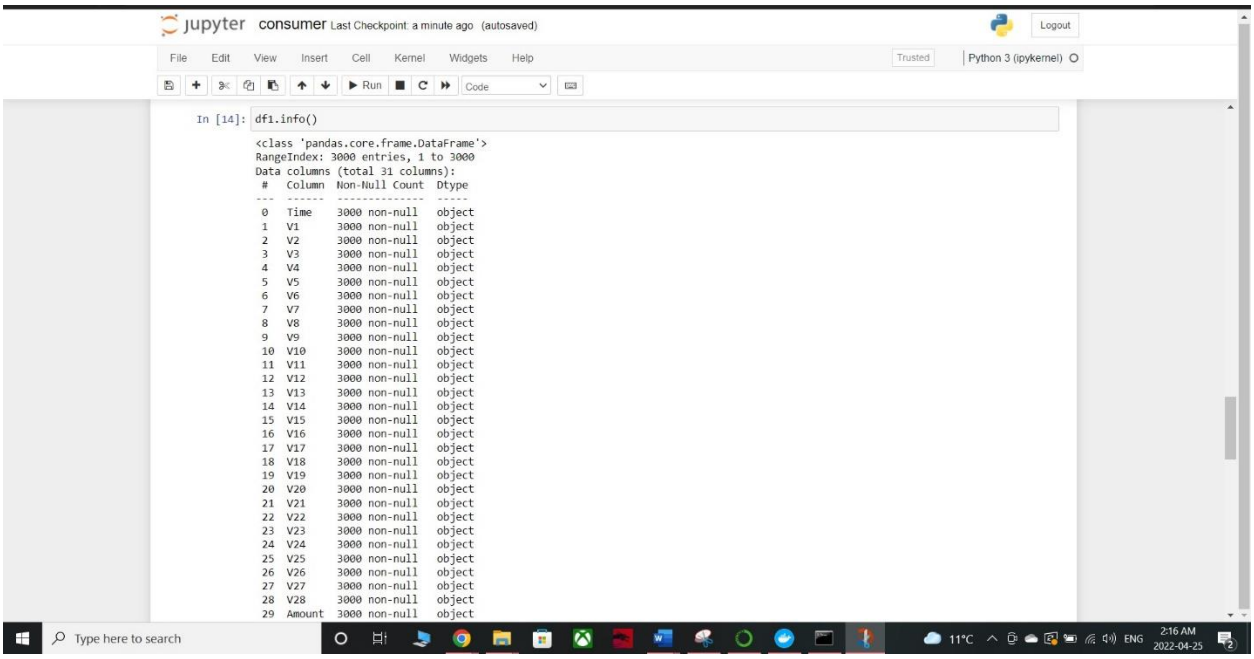


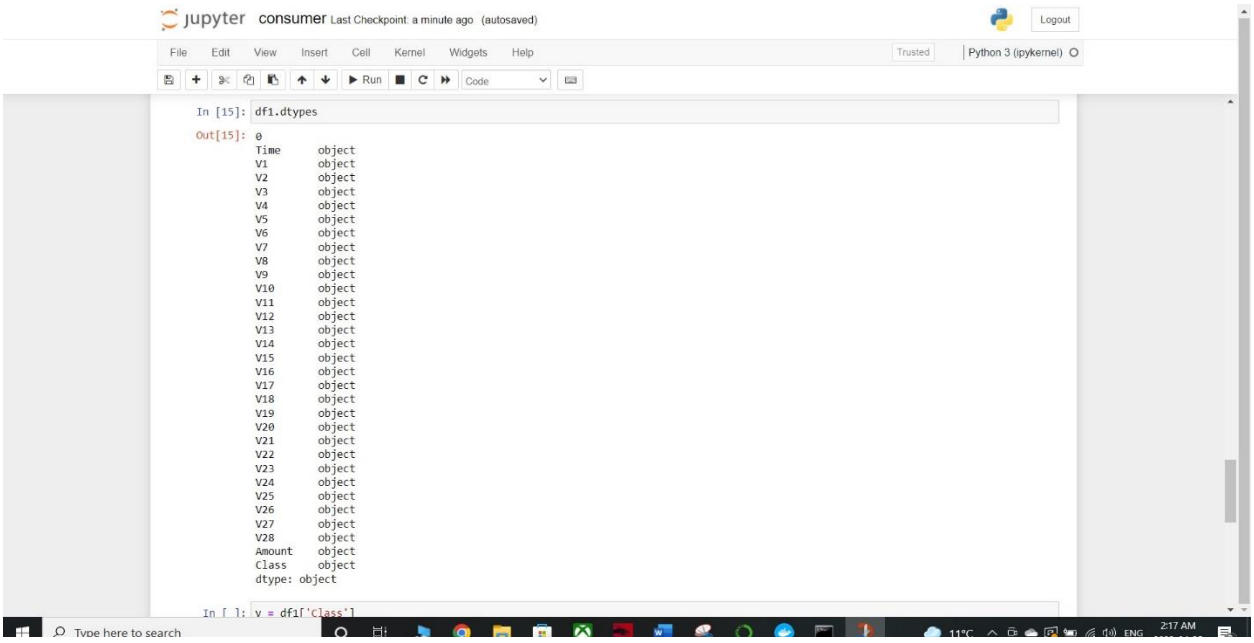

Save and Load Machine Learning
Big_Data_Term_Project/
producer - Jupyter Notebook
consumer - Jupyter Notebook
detection_model - Jupyter Notebook
localhost:8890/notebooks/Big_Data_Term_Project/consumer.ipynb
jupyter consumer Last Checkpoint: 6 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
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"\n',
'0,-1.3598071336738,-0.0727811733098497,2.53634673796914,1.37815522427443,-0.338320769942518,0.462387777762292,0.23959855406
1257,0.0986979012610507,0.363786969611213,0.0907941719789316,-0.551599533260813,-0.617800855762348,-0.991389847235408,-0.3111
69353699879,1.46817697209427,-0.470400525259478,0.207971241929242,0.0257905801985591,0.403992960255733,0.251412098239705,-0.0
1830677794153,0.277837575558899,-0.110473910188767,0.0669280749146731,0.128539358273528,-0.189114843888824,0.13355837674038
7,-0.0210530534538215,149.62,"0"\n',
'0,1.19185711131486,0.26615071205963,0.16648011335321,0.448154078460911,0.0600176492822243,-0.0823608088155687,-0.0788029833
323113,0.0851016549148104,-0.255425128109186,-0.166974414004614,1.61272666105479,1.06523531137287,0.489095015896608,-0.1437722
96441519,0.635558093258208,0.463917041022171,-0.114804663102346,-0.183361270123994,-0.145783041325259,-0.0690831352230203,-0.
2257752480033138,-0.638671952771851,0.101288021253234,-0.339846475529127,0.167170404418143,0.125894532368176,-0.00898309914322
813,0.0147241691924927,2.69,"0"\n',
'1,-1.35835406159823,-1.34016307473609,1.77320934263119,0.379779593034328,-0.503198133318193,1.80049938079263,0.791460956540
422,0.247675786588991,-1.51465432260583,0.207642865216696,0.624501459424895,0.066083685268831,0.717292731410831,-0.1659459227
63554,2.34586494901581,-2.89008319444231,1.10996937869599,-0.121359313195888,-2.26185709530414,0.524979725224404,0.2479981534
69754,0.771679401917229,0.909412262347719,-0.689280956490685,-0.327641833735251,-0.139096571514147,-0.0553527940384261,-0.059
7518405929204,378.66,"0"\n',
'1,-0.966271711572087,-0.185226008082898,1.79299333957872,-0.863291275036453,-0.0103088796030823,1.24720316752486,0.23760893
977178,0.377435874652262,-1.38702406270197,-0.0549519224713749,-0.226487263835401,0.178228225877303,0.507756869957169,-0.2879
In [ ]:
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","V
22","V23","V24","V25","V26","V27","V28","Amount","Class"\n'
In [ ]:
In [5]: strm_data_list = [elt.replace("\n", "") for elt in streamed_data]
Type here to search
13°C
ENG
1:58 AM
2022-04-25

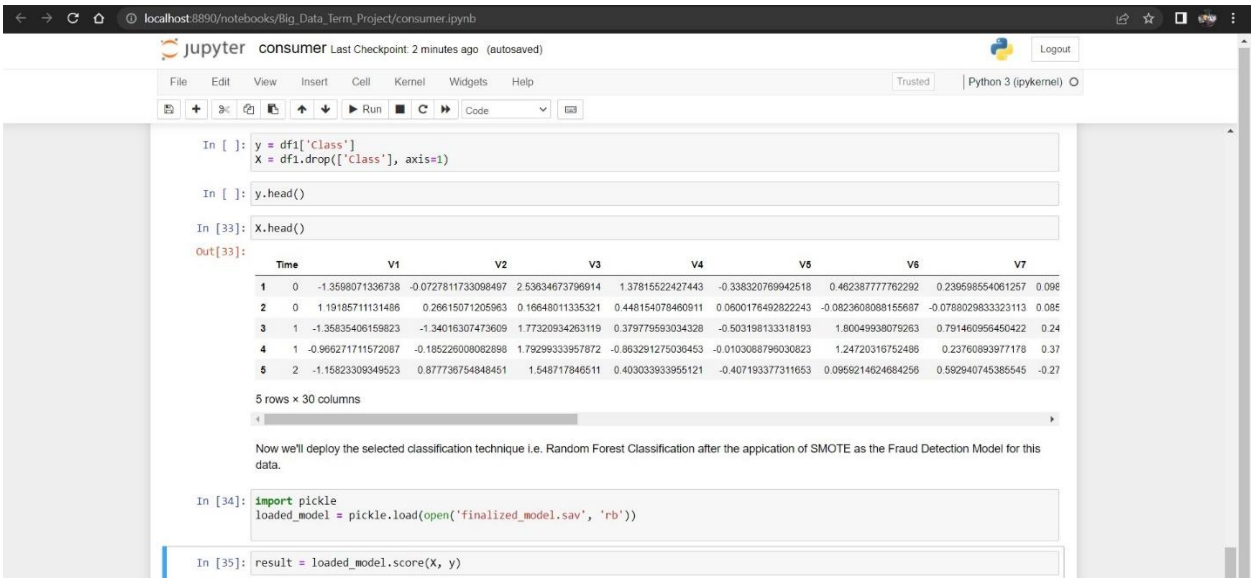

Save and Load Machine Learning
Big_Data_Term_Project/
producer - Jupyter Notebook
consumer - Jupyter Notebook
detection_model - Jupyter Notebook
localhost:8890/notebooks/Big_Data_Term_Project/consumer.ipynb
jupyter consumer Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [ ]:
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","V
22","V23","V24","V25","V26","V27","V28","Amount","Class"\n'
In [ ]:
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"',
'0,-1.3598071336738,-0.0727811733098497,2.53634673796914,1.37815522427443,-0.338320769942518,0.462387777762292,0.23959855406
1257,0.0986979012610507,0.363786969611213,0.0907941719789316,-0.551599533260813,-0.617800855762348,-0.991389847235408,-0.3111
69353699879,1.46817697209427,-0.470400525259478,0.207971241929242,0.0257905801985591,0.403992960255733,0.251412098239705,-0.0
1830677794153,0.277837575558899,-0.110473910188767,0.0669280749146731,0.128539358273528,-0.189114843888824,0.13355837674038
7,-0.0210530534538215,149.62,"0"',
'0,1.19185711131486,0.26615071205963,0.16648011335321,0.448154078460911,0.0600176492822243,-0.0823608088155687,-0.0788029833
323113,0.0851016549148104,-0.255425128109186,-0.166974414004614,1.61272666105479,1.06523531137287,0.489095015896608,-0.1437722
96441519,0.635558093258208,0.463917041022171,-0.114804663102346,-0.183361270123994,-0.145783041325259,-0.0690831352230203,-0.
2257752480033138,-0.638671952771851,0.101288021253234,-0.339846475529127,0.167170404418143,0.125894532368176,-0.00898309914322
813,0.0147241691924927,2.69,"0"',
'1,-1.35835406159823,-1.34016307473609,1.77320934263119,0.379779593034328,-0.503198133318193,1.80049938079263,0.791460956540
422,0.247675786588991,-1.51465432260583,0.207642865216696,0.624501459424895,0.066083685268831,0.717292731410831,-0.1659459227
63554,2.34586494901581,-2.89008319444231,1.10996937869599,-0.121359313195888,-2.26185709530414,0.524979725224404,0.2479981534
69754,0.771679401917229,0.909412262347719,-0.689280956490685,-0.327641833735251,-0.139096571514147,-0.0553527940384261,-0.059
7518405929204,378.66,"0"',
'1,-0.966271711572087,-0.185226008082898,1.79299333957872,-0.863291275036453,-0.0103088796030823,1.24720316752486,0.23760893
977178,0.377435874652262,-1.38702406270197,-0.0549519224713749,-0.226487263835401,0.178228225877303,0.507756869957169,-0.2879
Type here to search
13°C
ENG
1:59 AM
2022-04-25


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)
Run
Code
In [7]: df = pd.DataFrame(strm_data_list)
In [8]: df.head()
Out[8]:
0
0 "Time","V1","V2","V3","V4","V5","V6","V7","V8"...
1 0,-1.3598071336738,-0.0727811733098497,2.53634...
2 0,1.19185711131486,0.26615071205963,0.16648011...
3 1,-1.35835406159823,-1.34016307473609,1.773209...
4 1,-0.966271711572087,-0.185226008082898,1.7929...
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 0.09
2 0 1.19185711131486 0.26615071205963 0.16648011335321 0.448154078460911 0.0600176492822243 -0.0823608088155687 -0.0788029833323113 0.08
3 1 -1.35835406159823 -1.34016307473609 1.77320934263119 0.379779593034328 -0.503198133318193 1.80049938079263 0.791460956450422 0.2
4 1 -0.966271711572087 -0.185226008082898 1.79299333957872 -0.863291275036453 -0.0103088796030823 1.24720316752486 0.23760893977178 0.3
5 rows × 31 columns
In [11]: for i, col in enumerate(df1.columns):
df1.iloc[:, i] = df1.iloc[:, i].str.replace('"', '')
Type here to search
11°C
ENG
2:15 AM
2022-04-25


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)
Run
Code
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 0.098
2 0 1.19185711131486 0.26615071205963 0.16648011335321 0.448154078460911 0.0600176492822243 -0.0823608088155687 -0.0788029833323113 0.085
3 1 -1.35835406159823 -1.34016307473609 1.77320934263119 0.379779593034328 -0.503198133318193 1.80049938079263 0.791460956450422 0.24
4 1 -0.966271711572087 -0.185226008082898 1.79299333957872 -0.863291275036453 -0.0103088796030823 1.24720316752486 0.23760893977178 0.37
5 rows × 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 0.098
2 0 1.19185711131486 0.26615071205963 0.16648011335321 0.448154078460911 0.0600176492822243 -0.0823608088155687 -0.0788029833323113 0.085
3 1 -1.35835406159823 -1.34016307473609 1.77320934263119 0.379779593034328 -0.503198133318193 1.80049938079263 0.791460956450422 0.24
4 1 -0.966271711572087 -0.185226008082898 1.79299333957872 -0.863291275036453 -0.0103088796030823 1.24720316752486 0.23760893977178 0.37
5 2 -1.15823309349523 0.877736754848451 1.548717846511 0.403033933955121 -0.407193377311653 0.0959214624684256 0.592940745385545 -0.27
5 rows × 31 columns
Type here to search
11°C
ENG
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)
Run
Code
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 ENG 2:16 AM 2022-04-25

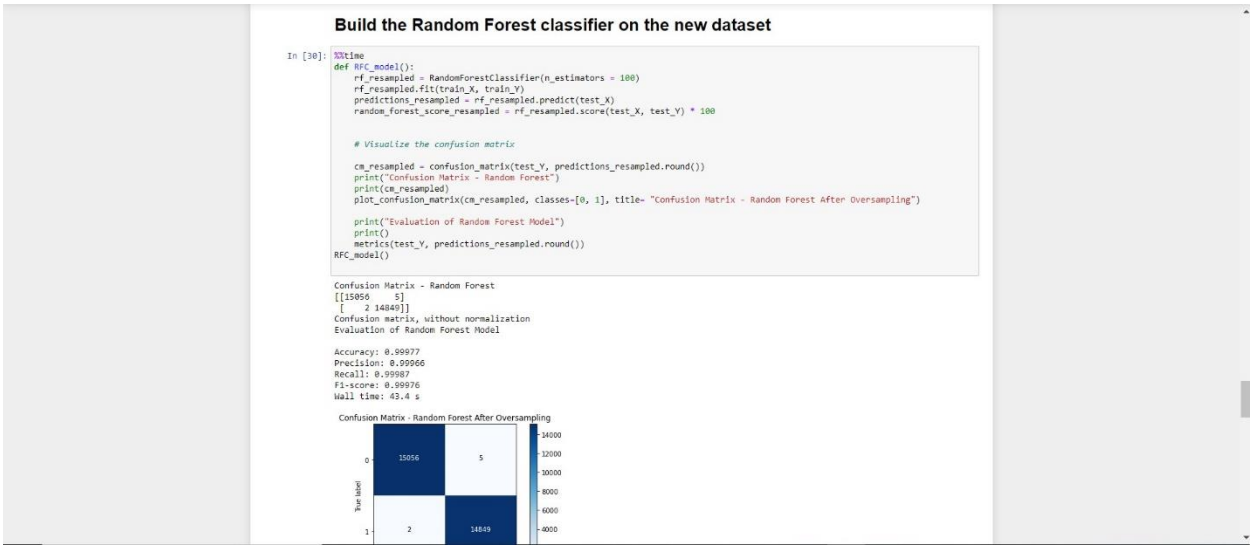

jupyter consumer Last Checkpoint: a minute ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [15]: df1.dtypes
Out[15]: 0
Time object
V1 object
V2 object
V3 object
V4 object
V5 object
V6 object
V7 object
V8 object
V9 object
V10 object
V11 object
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 [ ]: y = df1['Class']
Type here to search
11°C ENG 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)
Run
Code
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 0.462387777762292 0.239598554061257 0.098
2 0 1.19185711131486 0.26615071205963 0.16648011335321 0.448154078460911 0.0600176492822243 -0.0823608088155687 -0.0788029833323113 0.085
3 1 -1.35835406159823 -1.34016307473609 1.77320934263119 0.379779593034328 -0.503198133318193 1.80049938079263 0.791460956450422 0.24
4 1 -0.966271711572087 -0.185226008082898 1.79299333957872 -0.863291275036453 -0.0103088796030823 1.24720316752486 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 appication 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]:
```
# importnig 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
```

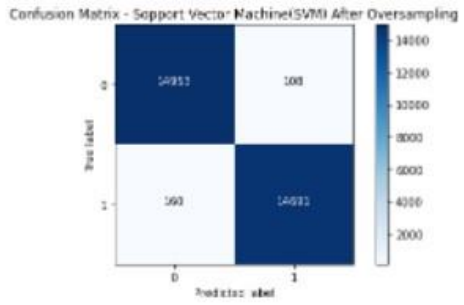
```
Wall time: 1min 6s
```

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 - Sopport 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 (SDGClassification)

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 Teechnique 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 fited 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**.