

CHAPTER 1

1. INTRODUCTION

1.1 Introduction :-

Credit card fraud detection is a relevant problem that draws the attention of machine-learning and computational intelligence communities, where a large number of automatic solutions have been proposed. In fact, this problem appears to be particularly challenging from a learning perspective, since it is characterized at the same time by class imbalance, namely, genuine transactions far outnumber frauds, and concept drift, namely, transactions might change their statistical properties over time. These, however, are not the only challenges characterizing learning problems in a real-world fraud-detection system (FDS).

In a real-world FDS, the massive stream of payment requests is quickly scanned by automatic tools that determine which transactions to authorize. Classifiers are typically employed to analyze all the authorized transactions and alert the most suspicious ones. Alerts are then inspected by professional investigators that contact the cardholders to determine the true nature (either genuine or fraudulent) of each alerted transaction. By doing this, investigators provide a feedback to the system in the form of labeled transactions, which can be used to train or update the classifier, in order to preserve (or eventually improve) the fraud-detection performance over time. The vast majority of transactions cannot be verified by investigators for obvious time and cost constraints. These transactions remain unlabeled until customers discover and report frauds, or until a sufficient amount of time has elapsed such that non-disputed transactions are considered genuine. Thus, in practice, most of supervised samples are provided with a substantial delay, a problem known as verification latency. The only recent supervised information made available to update the classifier is provided through the alert–feedback interaction. Most papers in the literature ignore the verification latency as well as the alert–feedback interaction, and unrealistically assume that the label of each transaction is regularly made available to the FDS, e.g., on a daily basis. However, these aspects have to be considered when designing a real-world FDS, since verification latency is harmful when concept drift occurs, and the alert–feedback interaction is

responsible of a sort of sample selection bias (SSB) that injects further differences between the distribution of training and test data.

1.2 Existing System

- A. Data-Driven Approaches in Credit Card Fraud Detection:** Both supervised and unsupervised methods have been proposed for credit card fraud-detection purposes. Unsupervised methods consist in outlier/ anomaly detection techniques that consider as a fraud any transaction that does not conform with the majority. Remarkably, an unsupervised DDM in an FDS can be directly configured from unlabeled transactions. A well-known method is peer group analysis, which clusters customers according to their profile and identifies frauds as transactions departing from the typical cardholder's behavior. The typical cardholder's behavior has also been modeled by means of self-organizing maps. Supervised methods are by far the most popular in fraud detection, and exploit labeled transactions for training a classifier. Frauds are detected by classifying feature vectors of the authorized transactions or possibly by analyzing the posterior of the classifier.
- B. Performance Measure for Fraud Detection:** The typical performance measure for fraud-detection problems is the AUC. AUC can be estimated by means of the Mann–Whitney statistic and its value can be interpreted as the probability that a classifier ranks frauds higher than genuine transactions. Another ranking measure frequently used in fraud detection is average precision, which corresponds to the area under the precision– recall curve. While these measures are widely used in detection problems, cost-based measures have been specifically designed for fraud-detection purposes. Cost-based measures quantify the monetary loss of a fraud by means of a cost matrix that associates a cost with each entry of the confusion matrix.
- C. Major Challenges to be Addressed in a Real-World FDS:**
 - 1) Class Imbalance:** Class distribution is extremely unbalanced in credit card transactions, since frauds are typically less than 1% of the overall transactions in our analysis. Learning under class imbalance has lately received a lot of attention, since traditional learning methods yield classifiers that are poorly performing on the minority class, which is

definitively the class of interest in detection problems. Several techniques have been proposed to deal with class imbalance, and for a comprehensive overview, we refer the reader to. The two main approaches for dealing with class imbalance are: a) sampling methods and b) cost-based methods.

- 2) Concept Drift: There are two main factors introducing changes/evolutions in the stream of credit card transactions, which in the literature are typically referred to as concept drift. At first, genuine transactions evolve because cardholders typically change their spending behaviors over time (e.g., during holidays, they purchase more and differently from the rest of the year). Second, frauds change over time, since new fraudulent activities are perpetrated. In our experiments.
- 3) Alert–Feedback Interaction and Sample Selection Bias: The majority of classifiers used for credit card fraud detection in the literature are tested in experiments where transaction labels are supposed to be available the very next day since the transaction is authorized. In a real-world FDS, the only recent supervised information is the feedbacks F_t , provided by investigators, while the vast majority of transactions authorized everyday do not receive a label in a short time ($|F_t| \ll |T_t|$).

The interaction between the FDS (raising alerts) and the investigators (providing true labels) recalls the active learning scenario, where it is possible to select few—very informative—samples and query their labels to an oracle which in the FDS would be the investigators. However, this is not feasible in a real-world FDS, since investigators have to focus on the most suspicious transactions to detect the largest number of frauds. Requests to check (possibly genuine) transactions for obtaining informative samples would be ignored. Considering the limited number of transactions investigators can check, addressing these questions would necessarily imply that some high-risk transaction is not being controlled, with the consequent loss in detection performance.

1.3 Proposed System

Detecting frauds in credit card transactions is perhaps one of the best test-beds for computational intelligence algorithms. In fact, this problem involves a number of relevant challenges, namely: concept drift (customers' habits evolve and fraudsters change their strategies over time), class imbalance (genuine transactions far outnumber frauds), and verification latency (only a small set of transactions are timely checked by investigators). However, the vast majority of learning algorithms that have been proposed for fraud detection rely on assumptions that hardly hold in a real-world fraud-detection system (FDS). This lack of realism concerns two main aspects: 1) the way and timing with which supervised information is provided and 2) the measures used to assess fraud-detection performance. This paper has three major contributions. First, we propose, with the help of our industrial partner, a formalization of the fraud-detection problem that realistically describes the operating conditions of FDSs that everyday analyze massive streams of credit card transactions.

1.4 Advantages of proposed system

With this level of Detecting frauds control system, fraudsters don't have the chance to make multiple transactions on a stolen or counterfeit card before the cardholder is aware of the fraudulent activity. This alone can save a significant amount of money that would traditionally be lost to fraud.

CHAPTER 2

1. REQUIREMENT ANALYSIS

2.1 Requirement Analysis

The project involved analyzing the design of few applications so as to make the application more users friendly. To do so, it was really important to keep the navigations from one screen to the other well ordered and at the same time reducing the amount of typing the user needs to do. In order to make the application more accessible, the browser version had to be chosen so that it is compatible with most of the Browsers.

2.2 Requirement Specification

2.2.1 Functional requirement

In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Functional software requirements help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform.

- Jupyter Notebook
- Creditcard.csv (dataset file)

2.2.2 Non-Functional requirement

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?

A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

2.3 Computational resource requirements

2.3.1 Hardware requirements :-

✓ Processor	:	Pentium IV or higher
✓ RAM	:	512 MB
✓ Hard Disk	:	40 GB
✓ Monitor	:	15' Colour Monitor.
✓ Mouse	:	Optical Mouse.

2.3.2 Software requirements :-

✓ Operating System	:	Windows 7 or Above
✓ Compiler / Interpreter	:	Python 3.7
✓ Application	:	Jupyter Notebook
✓ Libraries	:	Pandas, numpy.

CHAPTER 3

3. DESIGN

3.1 Design

This chapter provides the design phase of the Application. To design the project, we use the UML diagrams. The Unified Modelling Language (UML) is a general-purpose, developmental, modelling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system

3.2 Architecture

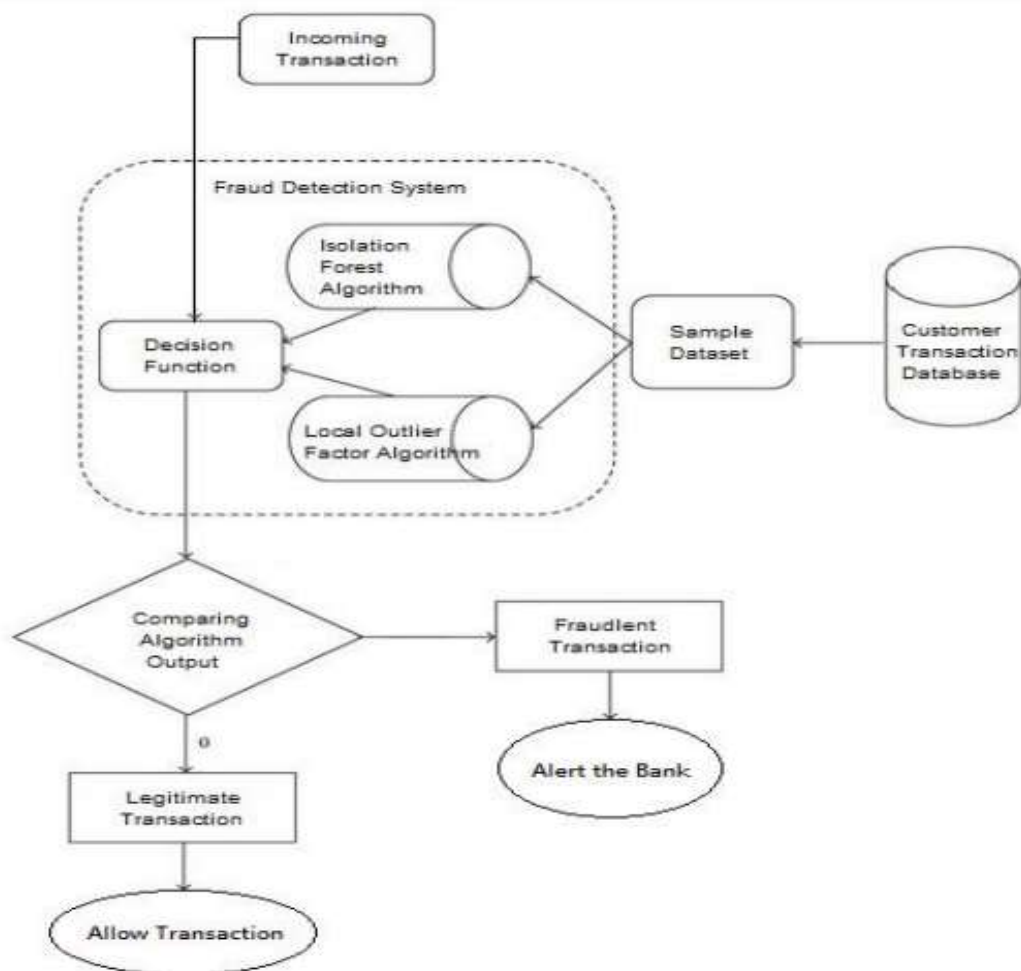


Figure: 3.1 Architecture Diagram

The above Architecture Diagram show about Credit Card Fraud Detection

3.3 Class Diagram

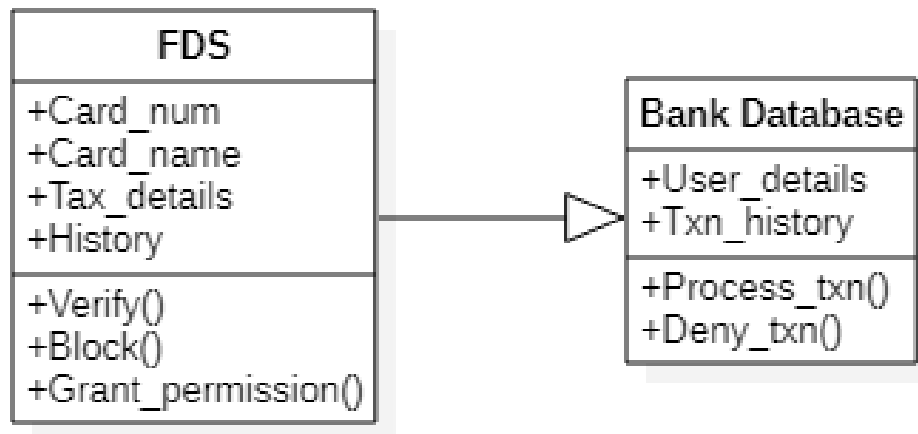


Figure: 3.2 Class Diagram

The above Class Diagram show about Credit Card Fraud Detection

3.4 Sequence Diagrams

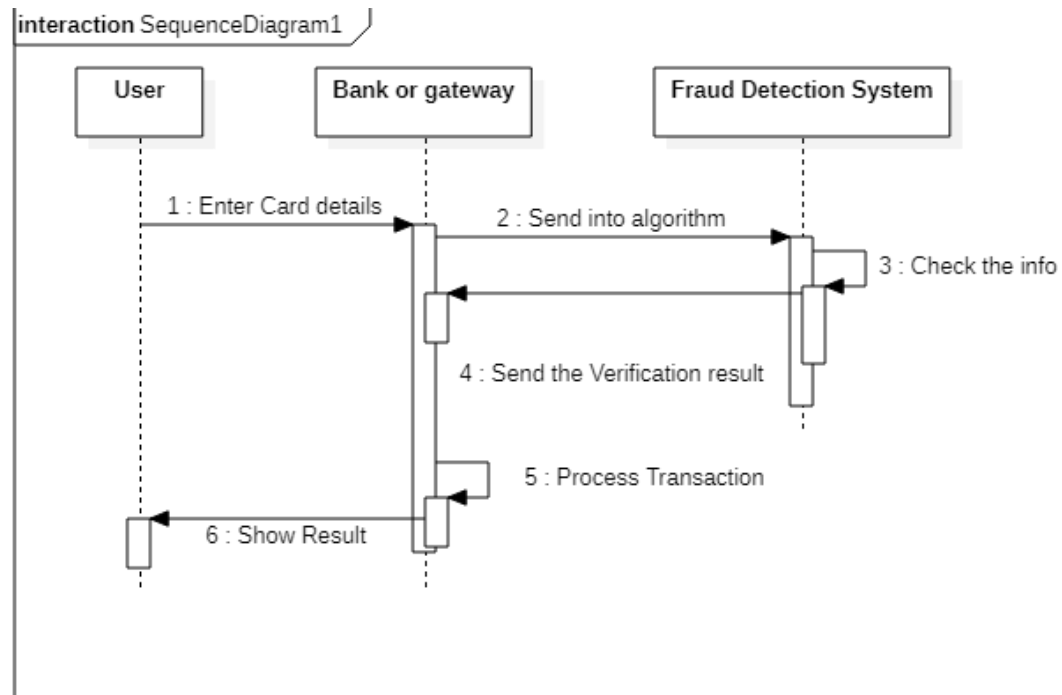


Figure: 3.3 Sequence Diagrams

The above Sequence Diagram show about Credit Card Fraud Detection

3.5 State Chart Diagram

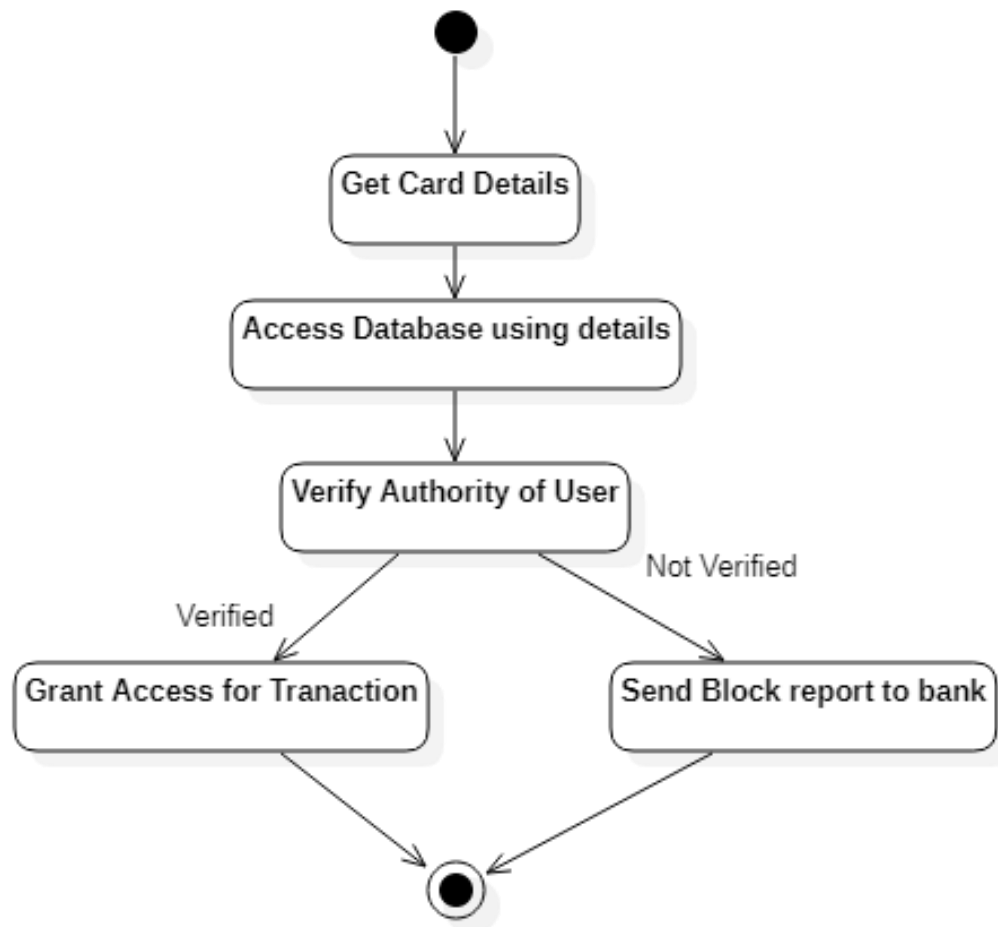


Figure: 3.4 State Chart Diagram

The above State Chart Diagram show about flow action of Credit Card Fraud Detection System

CHAPTER 4

4. Module

4.1 Module

A. Layers of Controls in an FDS:

- 1) Terminal: The terminal represents the first control layer in an FDS and performs conventional security checks on all the payment requests [63]. Security checks include controlling the PIN code (possible only in case of cards provided with chip), the number of attempts, the card status (either active or blocked), the balance available, and the expenditure limit. In case of online transactions, these operations have to be performed in real time (response has to be provided in a few milliseconds), during which the terminal queries a server of the card issuing company.
- 2) Transaction-Blocking Rules: Transaction-blocking rules are if-then (-else) statements meant to block transaction requests that are clearly perceived as frauds. These rules use the few information available when the payment is requested, without analyzing historical records or cardholder profile. An example of blocking rule could be “IF internet transactions AND unsecured Web site THEN deny the transaction.”
- 3) Scoring Rules: Scoring rules are also expert-driven models that are expressed as if-then (-else) statements. However, these operate on feature vectors and assign a score to each authorized transaction: the larger the score, the more likely the transaction to be a fraud. Scoring rules are manually designed by investigators, which arbitrarily define their associated scores. An example of scoring rule can be “IF previous transaction in a different continent AND less than 1 h from the previous transaction THEN fraud score = 0.95.”
- 4) Data Driven Model (DDM): This layer is purely data driven and adopts a classifier or another statistical model to estimate the probability for each feature vector being a fraud.

5) Investigators: Investigators are professionals experienced in analyzing credit card transactions and are responsible of the expert-driven layers of the FDS. In particular, investigators design transaction-blocking and scoring rules. Investigators call cardholders and, after having verified, assign the label “genuine” or “fraudulent” to the alerted transaction, and return this information to the FDS. In the following, we refer to these labeled transactions as feedbacks and use the term alert–feedback interaction to describe this mechanism yielding supervised information in a real-world FDS.

B. Features Augmentation:

Any transaction request is described by few variables such as the merchant ID, cardholder ID, purchase amount, date, and time. All transaction requests passing the blocking rules are entered in a database containing all recent authorized transactions, where the feature-augmentation process starts. During feature augmentation, a specific set of aggregated features associated with each authorized transactions is computed, to provide additional information about the purchase and better discriminate frauds from genuine transactions.

C. Supervised Information:

Investigators’ feedbacks are the most recent supervised information made available to the FDS, but represent only a small fraction of the transactions processed every day. Additional labeled transactions are provided by cardholders that directly dispute unauthorized transactions. The timing of disputed transactions can vary substantially, since cardholders have different habits when checking the transcript of credit card sent by the bank. Moreover, checking disputed transactions entails some necessary administrative procedures that might introduce substantial delays.

D. System Update:

Customers’ spending behavior evolves and fraudsters continuously design new attacks, and thus their strategies also change over time. It is then necessary to constantly update the FDS to guarantee satisfactory performance. Expert-driven systems are regularly updated by investigators who add ad hoc (transaction-blocking or scoring) rules to counteract the onset of new fraudulent activities and remove those rules liable of too many false alerts.

CHAPTER 5

5. IMPLEMENTATION

5.1 Sample Code

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt #

import matplotlib.gridspec as gridspec

#loading the data

df_credit = pd.read_csv("creditcard.csv")

#looking the how data looks

df_credit.head()
```

5.2 Sample Dataset

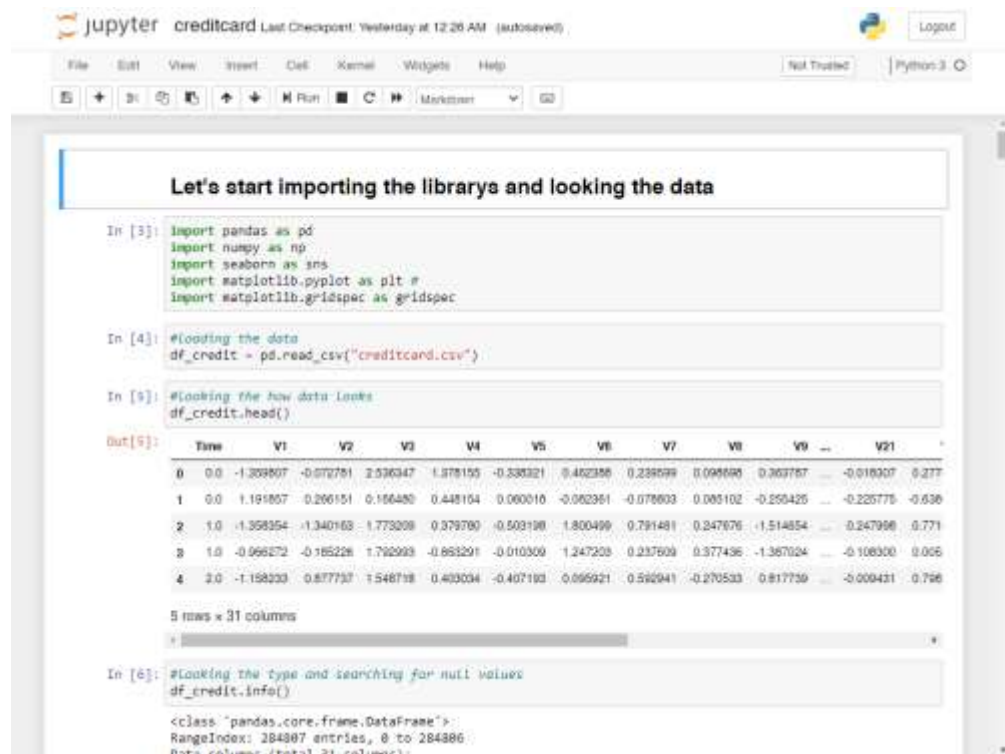
Time	V1	V2	V3	V4	V5
0	-1.35981	-0.07278	2.536347	1.378155	-0.33832
0	1.191857	0.266151	0.16648	0.448154	0.060018
1	-1.35835	-1.34016	1.773209	0.37978	-0.5032
1	-0.96627	-0.18523	1.792993	-0.86329	-0.01031
2	-1.15823	0.877737	1.548718	0.403034	-0.40719
2	-0.42597	0.960523	1.141109	-0.16825	0.420987
4	1.229658	0.141004	0.045371	1.202613	0.191881
7	-0.64427	1.417964	1.07438	-0.4922	0.948934
7	-0.89429	0.286157	-0.11319	-0.27153	2.669599
9	-0.33826	1.119593	1.044367	-0.22219	0.499361
10	1.449044	-1.17634	0.91386	-1.37567	-1.97138

CHAPTER 6

6. SCREENSHOTS

6.1 Screenshots

6.1.1 Importing libraries



The screenshot shows a Jupyter Notebook titled 'creditcard' with a 'Last Checkpoint: Yesterday at 12:26 AM (autosaved)' status. The interface includes a top menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and other functions. The notebook content is as follows:

```
Let's start importing the libraries and looking the data
```

```
In [3]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #
import matplotlib.gridspec as gridspec
```

```
In [4]: #loading the data
df_credit = pd.read_csv("creditcard.csv")
```

```
In [5]: #looking the how data looks
df_credit.head()
```

```
Out[5]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.309807	-0.072751	2.536347	1.578155	-0.338321	0.462358	0.238599	0.095660	0.263757	...	-0.018007	0.277
1	0.0	1.191857	0.206151	0.186480	0.448104	0.060010	-0.002951	-0.078803	0.005102	-0.250425	...	-0.225775	-0.636
2	1.0	-1.358354	-1.340163	-1.773208	0.379790	-0.503198	1.800499	0.791481	0.247676	-1.514854	...	0.247998	0.771
3	1.0	-0.066272	-0.185228	1.790093	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.367024	...	-0.108300	0.005
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407103	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798

5 rows x 31 columns

```
In [6]: #Looking the type and searching for null values
df_credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
```

Figure No. 6.1 Importing libraries

The Above Figure show the Interface of Jupyter

6.1.2 Normal Transaction

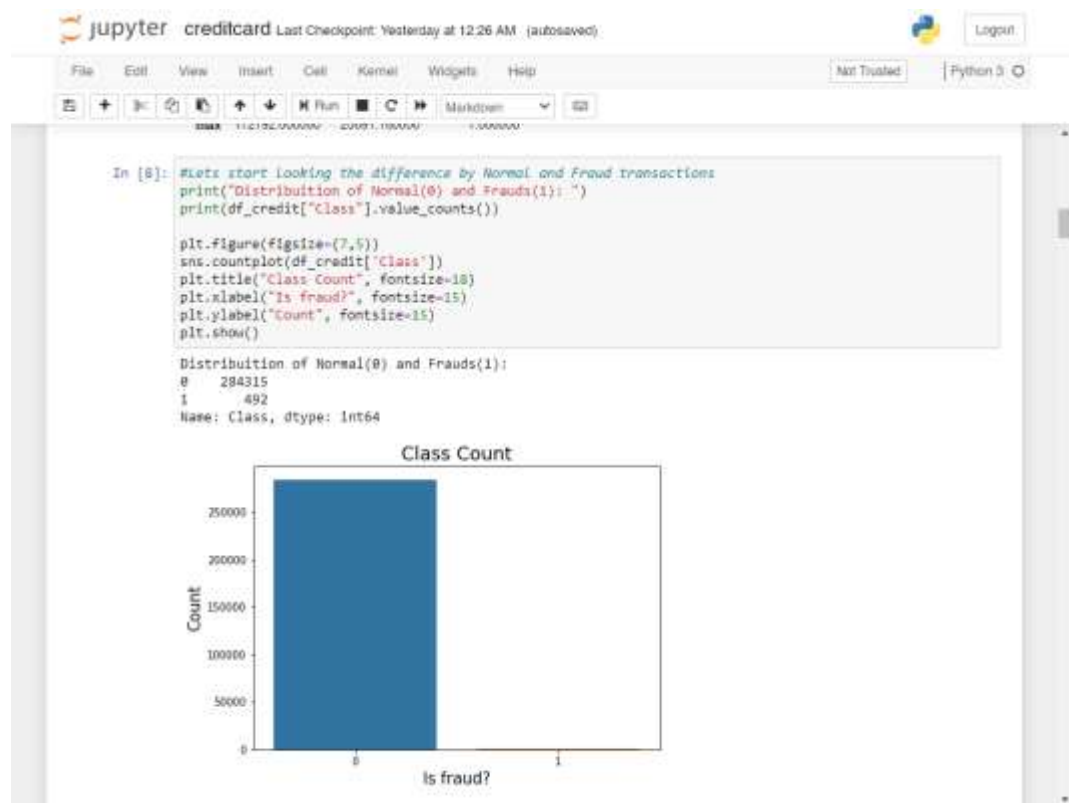


Figure No. 6.2 Normal Transaction

The Above Figure show the Interface of Jupyter

6.1.3 Fraud x Normal Transaction by hours

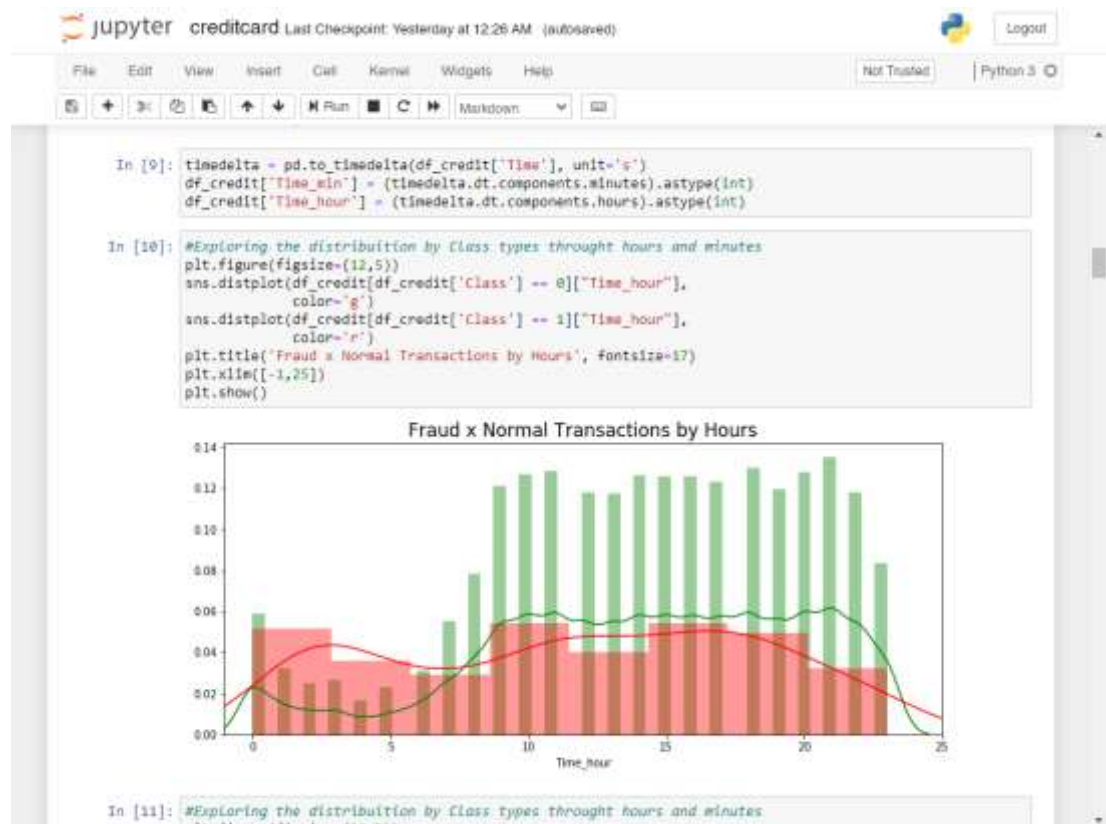


Figure No. 6.3 Fraud x Normal Transaction by hours

The Above Figure show the Interface of Jupyter

6.1.4 Fraud x Normal Transaction by minutes

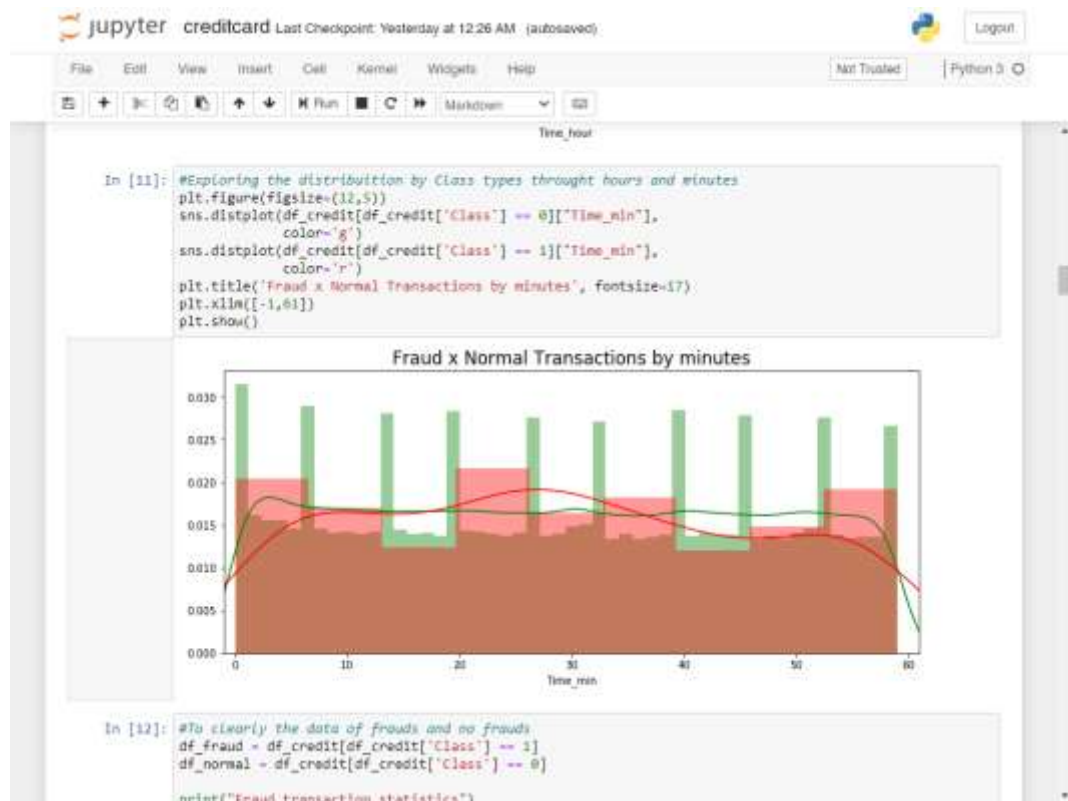


Figure No. 6.4 Fraud x Normal Transaction by minutes

The Above Figure show the Interface of Jupyter

6.1.5 Class x Amount

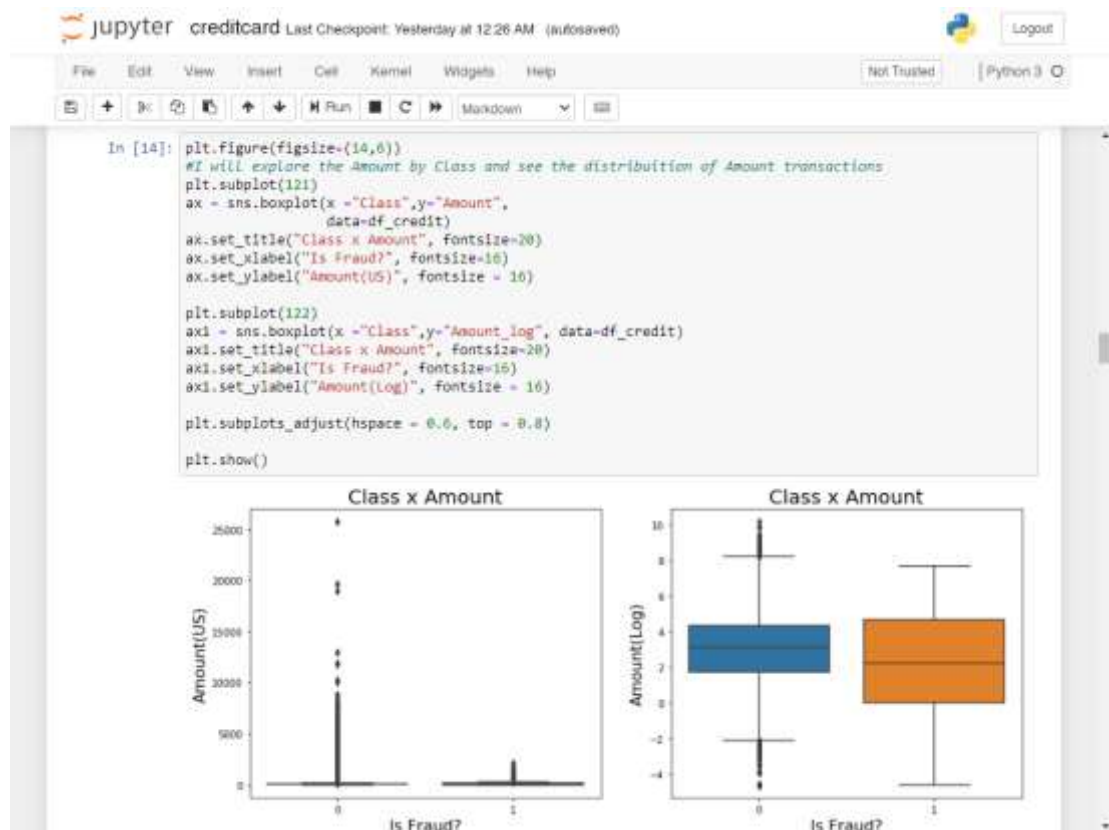


Figure No. 6.5 Class x Amount

The Above Figure show the Interface of Jupyter

6.1.6 Amount by minutes of Fraud and Normal Transactions

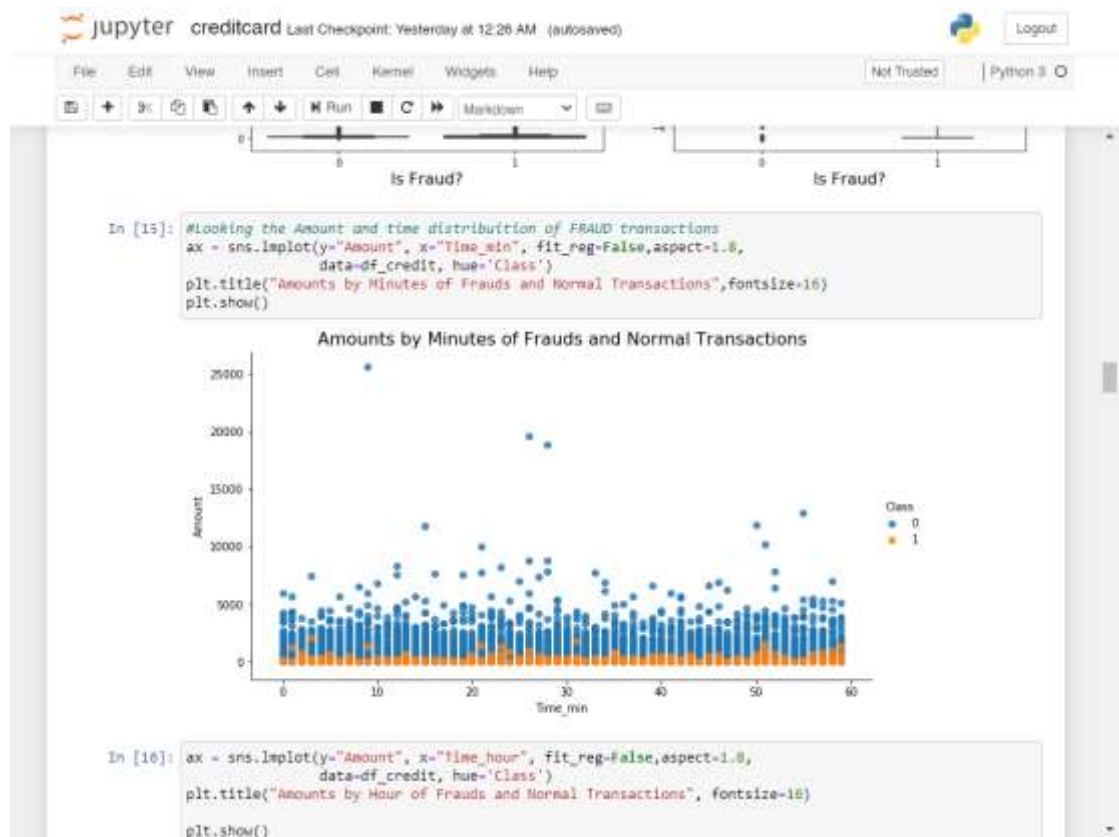


Figure No. 6.6 Amount by minutes of Fraud and Normal Transactions

The Above Figure show the Interface of Jupyter

6.1.7 Amount by Hours of Fraud and Normal Transactions

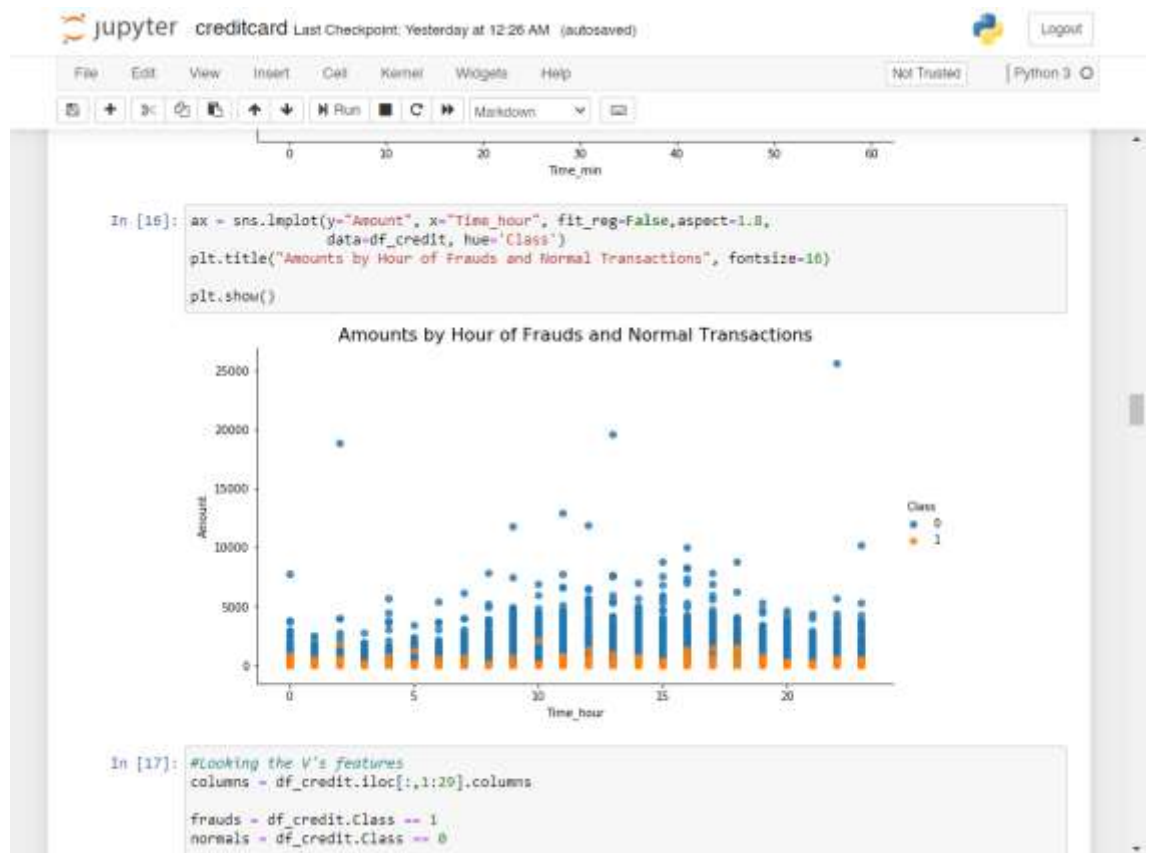


Figure No. 6.7 Amount by Hours of Fraud and Normal Transactions

The Above Figure show the Interface of Jupyter

6.1.8 V's Features

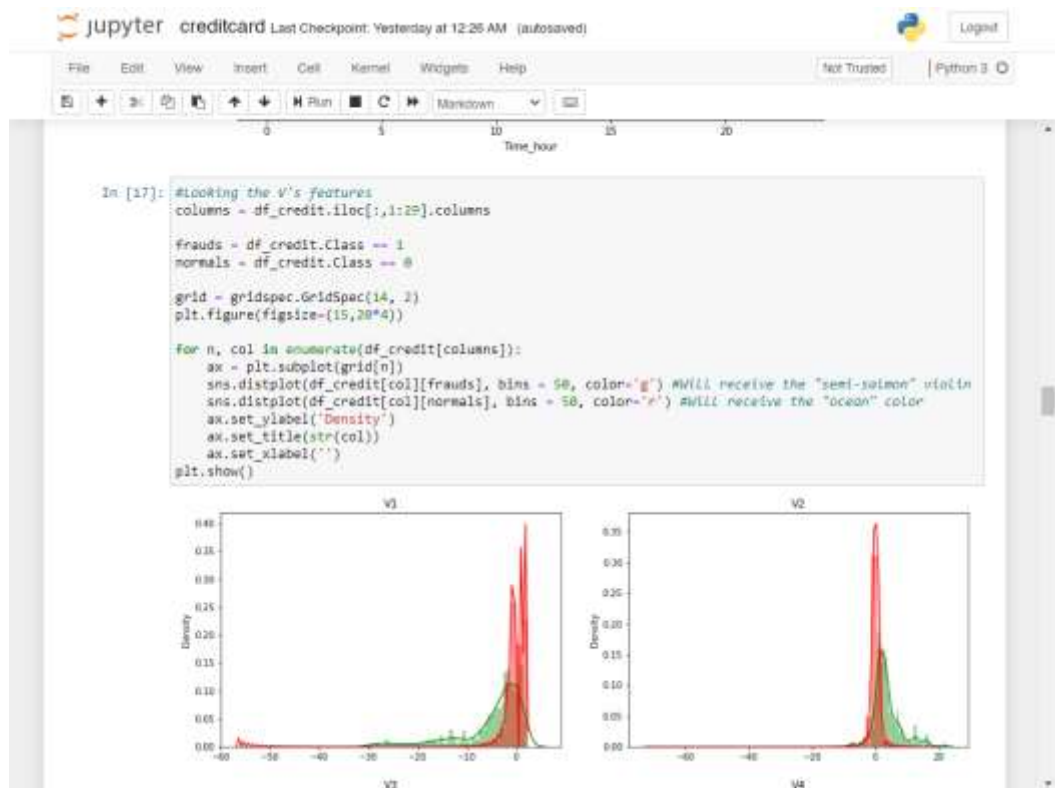


Figure No. 6.8 V's Features

The Above Figure show the Interface of Jupyter

6.1.9 Correlation Matrix

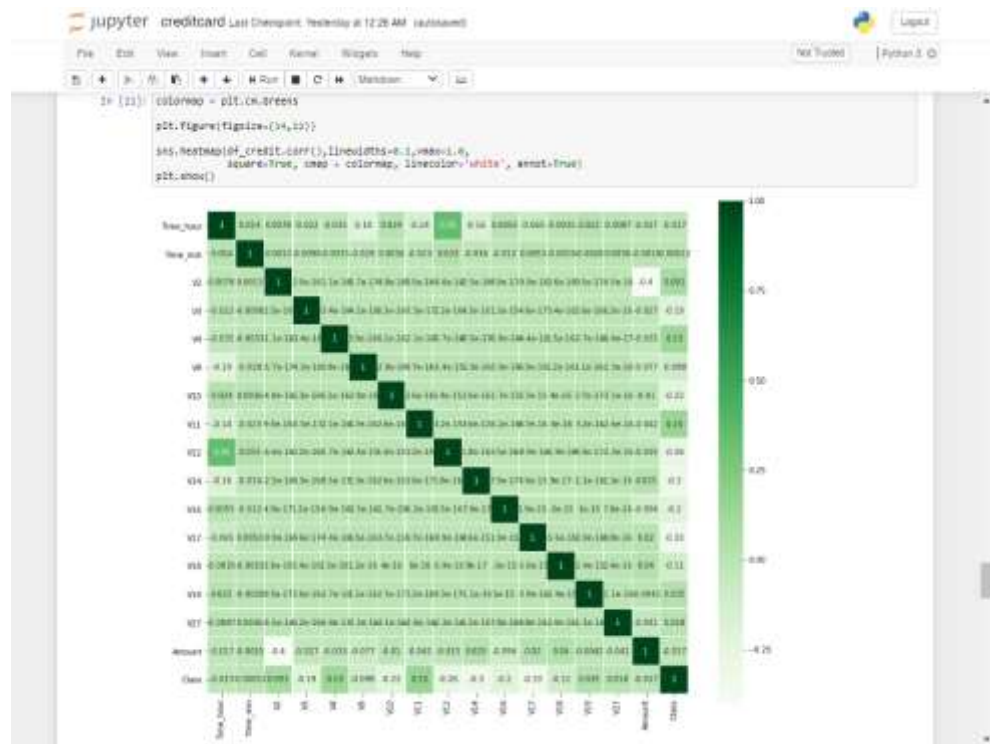


Figure No. 6.9 Correlation Matrix

The Above Figure show the Interface of Jupyter

6.1.10 Feature Importance Plot

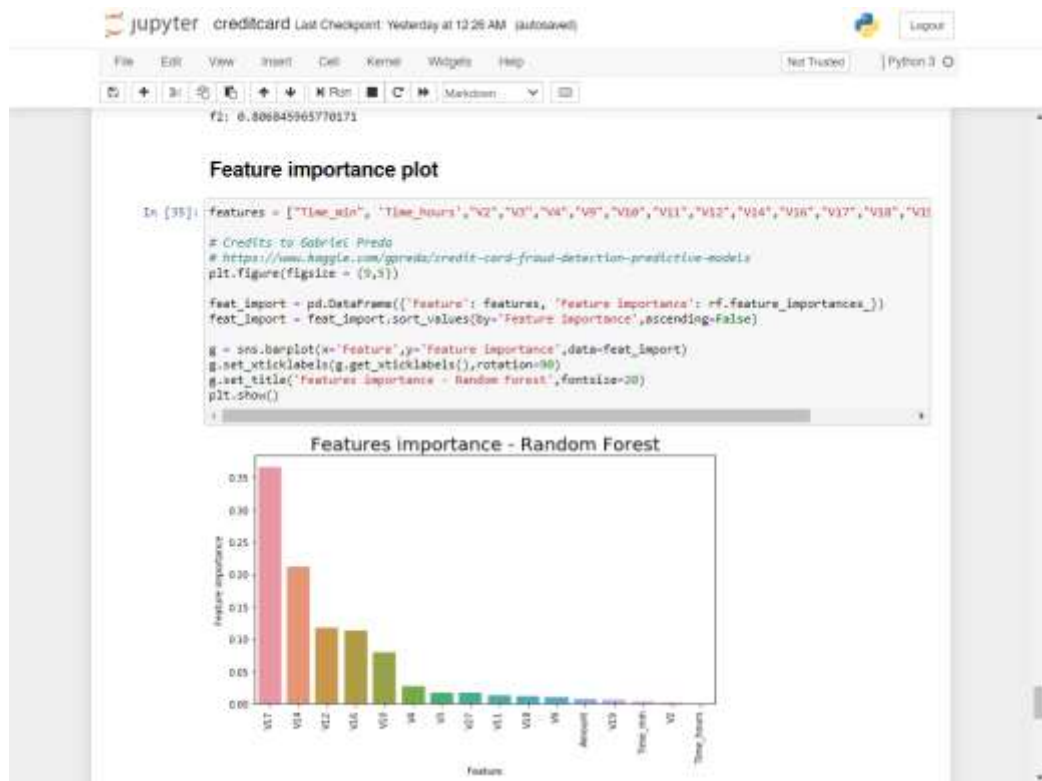


Figure No. 6.10 Feature Importance Plot

The Above Figure show the Interface of Jupyter

6.1.11 ROC Curve – Random Forest

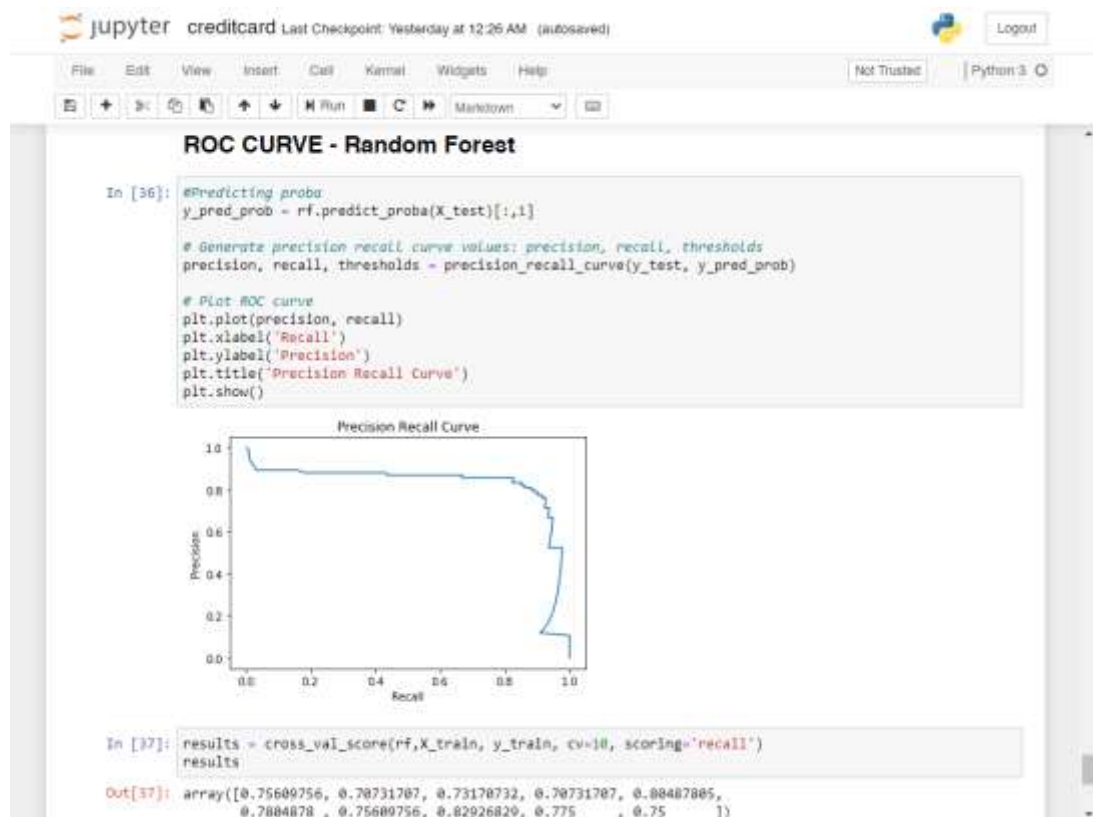


Figure No. 6.11 ROC Curve – Random Forest

The Above Figure show the Interface of Jupyter

CHAPTER 7

7. TESTING

7.1 Overview of Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.2 Types of Testing

7.2.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

7.2.2 Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate

that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

7.2.3 Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

7.3 Unit Testing :-

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.
- Features to be tested
- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

7.4 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

7.5 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

CHAPTER 8

8. CONCLUSION AND FUTURE WORK

8.1 Conclusion

The majority of works addressing the fraud-detection problem in credit card transactions unrealistically assume that the class of each transaction is immediately provided for training the classifier. Here we analyze in detail the real-world working conditions of FDS and provide a formal description of the articulated classification problem involved. In particular, we have described the alert–feedback interaction, which are the mechanism providing recent supervised samples to train/update the classifier. We also claim that, in contrast to traditional performance measures used in the literature, in a real-world FDS, the precision of the reported alerts is probably the most meaningful one, since investigators can check only few alerts. Our experiments on two vast data sets of real-world transactions show that, in order to get precise alerts, it is mandatory to assign larger importance to feedbacks during the learning problem. Not surprisingly, feedbacks play a central role in the proposed learning strategy, which consists in separately training a classifier on feedbacks and a classifier on delayed supervised samples, and then aggregating their posteriors to identify alerts. Our experiments also show that solutions that lower the influence of feedbacks in the learning process (e.g., classifiers that mix feedbacks and delayed supervised samples or that implement instance weighting schemes) are often returning less precise alerts. Future work concerns the study of adaptive and possibly nonlinear aggregation methods for the classifiers trained on feedbacks and delayed supervised samples. We also expect to further increase the alert precision by implementing a learning to rank approach that would be specifically designed to replace the linear aggregation of the posterior probabilities. Finally, a very promising research direction concerns semi-supervised learning methods for exploiting in the learning process also few recent unlabeled transactions.

8.2 Scope for Future work

Advances in technology give criminals increasingly powerful tools to commit fraud, especially using credit cards or internet bots. To combat the evolving face of fraud, researchers are developing increasingly sophisticated tools, with algorithms and data structures capable of handling large-scale complex data analysis and storage. The most popular area of current fraud detection research has been in credit card, but we see online bots and Ad click fraud as growing concerns for the future. With rapid reduction in the cost of computing power, publishers can exploit vulnerabilities by creating bots to click on Ads to generate more revenue. Banks typically implement a single fraud detection and prevention system that tries to capture fraudulent transactions based on a model generalized to all their customers. This network model incorporates general fraud trends from different products across the bank. However, this approach is ineffective in the long run as they are too broad to find ever more sophisticated forms of fraud. Credit card associations are combining network as well as custom models to develop a comprehensive system that detects fraud upon point of sale. For instance, MasterCard implements. In most of our entries we have been very interested in fraud detection in the financial industry. In this entry we also want to mention alternative emerging fraud behaviors; also how they harm some business and the strategies used in the industry to detect it. Online click fraud is the act of clicking on advertisements without a specific interest on the product. Such practice is usually performed by software in a systematic way, increasing the marketing expenses for the business offering the product. This also harms the credibility of the advertising companies and the online advertising industry as a whole. Click forensics estimate that fraud clicks correspond to a 19% of overall clicks through ads.

BIBLIOGRAPHY

- [1] A. C. Bahnsen, D. Aouada, and B. Ottersten, "Example-dependent cost-sensitive decision trees," *Expert Syst. Appl.*, vol. 42, no. 19, pp. 6609–6619, 2015.
- [2] C. Alippi, G. Boracchi, and M. Roveri, "A just-in-time adaptive classification system based on the intersection of confidence intervals rule," *Neural Netw.*, vol. 24, no. 8, pp. 791–800, 2011.
- [3] C. Alippi, G. Boracchi, and M. Roveri, "Hierarchical change-detection tests," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 2, pp. 246–258, Feb. 2016.
- [4] C. Alippi, G. Boracchi, and M. Roveri, "Just-in-time classifiers for recurrent concepts," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 4, pp. 620–634, Apr. 2013.
- [5] B. Baesens, V. Van Vlasselaer, and W. Verbeke, *Fraud Analytics Using Descriptive, Predictive, and Social Network Techniques: A Guide to Data Science for Fraud Detection*. Hoboken, NJ, USA: Wiley, 2015.