A Major Project Report on

CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING AND DATA SCIENCE

Submitted In partial fulfillment of the Requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

BY

EKKEY SRAVANTHI	**(17UJ1A0506)**
PERALA SHRAVYA	**(17UJ1A0518)**
KARANKOT PRADEEP	**(17UJ1A0536)**
THIMMANI SUKUMAR	**(17UJ1A0523)**
AERUKONDA SAI CHARAN	**(18UJ5A0501)**

Under the Esteemed Guidance of

Mr. G. CHAKRAPANI, M.Tech

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MALLA REDDY COLLEGE OF ENGINEERING AND MANAGEMENT SCIENCE
(Approved by AICTE New Delhi & Affiliated to JNTU Hyderabad)Kistapur, Medchal, Medchal Dist- 501401.

2017-2021

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD, KUKATPALLY, HYDERABAD-85.

MALLA REDDY ENGINEERING COLLEGE AND MANAGEMENT SCIENCES
(Approved by AICTE New Delhi & Affiliated to JNTU Hyderabad)
Kistapur, Medchal, Medchal Dist- 501401.

CERTIFICATE

This is to certify that the major project report entitled " Credit card fraud detection using machine learning and data science" being submitted by

EKKEY SRAVANTHI	**(17UJ1A0506)**
PERALA SHRAVYA	**(17UJ1A0518)**
KARANKOT PRADEEP	**(17UJ1A0536)**
THIMMANI SUKUMAR	**(17UJ1A0523)**
AERUKONDA SAI CHARAN	**(18UJ5A0501)**

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Jawaharlal Nehru Technological University , Hyderabad, during the academic year 2017-2021is a record of bonafied work carried out under my guidance and supervision.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Internal Guide

Mr. G. CHAKRAPANI M.Tech.

Assistant Professor
Department of CSE

Head of The Department

Dr. D. MAHAMMAD RAFI, M.Tech,Ph.D

Professor & Head of CSE
Department of CSE

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First and foremost, I express my deep debt of gratitude to the Chairperson and Managing Trustee **Mr. V. Malla Reddy** for their immense contribution in making this organization grow and providing me the state of the art facilities to do this project work.

My heartfelt gratefulness to Director of MREM **Mr. L. Venugopal Reddy** for their deep commitment and dedication, to bring this institution to the peak in terms of discipline and values.

we owe my full satisfaction in my project work to the extensive hands of cooperation of Principal Academics **Dr. CNV. SRIDHAR, M.Tech,Ph.D.**

we would like to express my deep appreciation and sincere gratitude to my project internal guide , **Mr. G.Chakrapani**, Assistant Professor, Department of Computer Science & Engineering, Malla Reddy Engineering College and Management Sciences, for his guidance, support, encouragement, understanding and patience. I have been honored to work under his supervision and learn from her advice and useful insights throughout my project work.

we extend my gratitude to **Dr. D.Mahammad Rafi**, Professor and Head, Department of Computer Science and Engineering for his constant support to complete the project work.

we express my sincere thanks to all the teaching and non-teaching staffs of Malla Reddy Engineering College and Management Sciences who cooperated with me, which helped me to realize my dream. We would like to thank our internal project mates and department faculties for their full-fledged guidance and giving courage to carry out the project. I am very much thankful to one and all that helped me for the successful completion of my project.

EKKEY SRAVANTHI **(17UJ1A0506)**
PERALA SHRAVYA **(17UJ1A0518)**
KARANKOT PRADEEP **(17UJ1A0536)**
THIMMANI SUKUMAR **(17UJ1A0523)**
AERUKONDA SAI CHARAN **(18UJ5A0501)**

ABSTRACT

In the last years, credit and debit cards usage has significantly increased. However, a non-negligible part of the credit card transactions is fraudulent and billions of euros are stolen every year throughout the world. In Belgium alone, the volume of credit card transactions reached 16 billion euros in 2017 with 140 million euros of illegitimate transactions.

Credit card fraud detection present several characteristics that makes it a challenging task. First, the feature set describing a credit card transaction usuallyignores detailed sequential information which was proven to be relevant for the detection of credit card fraudulent transactions. Second, purchase behaviours and fraudster strategies may change over time, making a learnt fraud detection decision function irrelevant if not updated. This phenomenon named dataset shift (change in the distribution p(x, y)) may hinder fraud detection systems to obtain good performances. We conducted an exploratory analysis in order to quantify the day by day dataset shift and identified calendar related time periods that show different properties. Third, credit card transactions data suffer from a strong imbalance regarding the class labels which needs to be considered either from the classifier perspective or from the data perspective (less than 1% of the transactions are fraudulent transactions).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

PCA	Principal Component Analysis
LOF	Local Outlier Factor
KNN	K-nearest Neighbor
SAS	Statistical Software Suite
BAE	British Multinational Arms, Security, And Aerospace Company
IBM	International Business Machines
LRD	Local Reachability Density
MIT	Massachusetts Institute of Technology
CNRI	Corporation of National Research Initiative
LLNL	Lawrence Livermore National Laboratory
MATLAB	Matrix Laboratory
LAPACK	Linear Algebra Package
BLAS	Basic Linear Algebra Subprograms
BSD	Berkeley Software Distribution
GUI	Graphical User Interface
API	Application Programming Interface
CCFLAM	Credit Card Fraud Intelligent Agent Model
HCL	Hardware Compatibility List
DAST	Dynamic Application Security Testing
CCFD	Credit Card Fraud Detection
AI	Artificial Intelligence

CHAPTER - I

INTRODUCTION

In 2011, 700 million payment cards have been issued in the EU. The volume of non-cash transaction for that year exceeded 3000 billion euros. The implementation of EMV1 (chip-embedded cards) for face-to-face transactions and strong identification of customers via 3D secure2 for e-commerce transactions increased significantly the security of European payments.

However, even after the implementation of EMV and 3D-SECURE security, a non-negligible number of credit card transactions remains illegitimate: the amount of european credit card fraud reaches 1.5 billion euros yearly. In order to complement the decrease in credit card fraud obtained with the inclusion of authentication methods, experts agree that data driven fraud detection systems are the future of credit card fraud detection [Ali et al., 2019].

According to [Europol, 2012]3, international organised crime groups dominate the criminal market of credit card fraud and affect non-cash payments on a global level. Doing fraudulent card payment is not risky and provide high profit to these organised crime groups. These incomes are afterwards invested in order to develop further fraudulent strategies, to finance other criminal activities or to start legal businesses (money laundering).

Experts reported that the international and highly organized nature of criminal networks creates the need for international police cooperation. However, the 1EMV (Europay, MasterCard, Visa) : a standard for payment cards based on chip technology.
2 3D secure: double identification of the card-holder via a PIN sent by SMS 3EUROPOL: European agency for criminality repression and collaboration between police force.

CHAPTER - 2

LITERATURE SURVEY

2.LITERATURE SURVEY

Kuldeep Randhawa et al. proposed a technique using machine learning to detect credit card fraud detection. Initially, standard models were used after that hybrid models came into picture which made use of AdaBoost and majority voting methods. Publicly available data set had been used to evaluate the model efficiency and another data set used from the financial institution and analyzed the fraud. Then the noise was added to the data sample through which the robustness of the algorithms could be measured. The experiments were conducted on the basis of the theoretical results which show that the majority of voting methods achieve good accuracy rates in order to detect the fraud in the credit cards. For further evaluation of the hybrid models noise of about 10% and 30% has been added to the sample data. Several voting methods have achieved a good score of 0.942 for 30% added noise. Thus, it was concluded that the voting method showed much stable performance in the presence of noise.

Abhimanyu Roy et al. proposed deep learning topologies for the detection of fraud in online money transaction. This approach is derived from the artificial neural network with in-built time and memory components like long-term short-term memory and several other parameters. According to the efficiency of these components in fraud detection, almost 80 million online transactions through credit card have been pre-labeled as fraudulent and legal. They have used high performance distributed cloud computing environment. The study proposed by the researchers provides an effective guide to the sensitivity analysis of the proposed parameters as per the performance of the fraud detection. The researchers also proposed a framework for the parameter tuning of Deep Learning topologies for the detection of fraud. This enables the financial institution to decrease the losses by avoiding fraudulent activities.

Shiyang Xuan et al. used two types of random forests which train the behavior features of normal and abnormal transactions. The researcher compares these two random forests which are differentiated based on their classifiers, performance on the detection of credit card fraud.

S.No	Title of Paper	Authors	Year	TechnologyUsed	Advantages	Disadvantages
	Credit card fraud detection using AdaBoost and majority voting	Kuldeep Randhawa, Chu Kiong Loo, Manjeevan Seera, Chee Peng Lim and Asoke K. Nandi	2018	Machine learning to detect credit card fraud detection.	Majority of voting methods achieve good accuracy rates in order to detect the fraud in the credit cards.	The precision value achieved is less as compared to other algorithms.
	Deep learning detecting fraud in credit card transactions	A. Roy and J. Sun and R. Mahoney and L. Alonzi and S. Adams and P. Beling	2018	Deep learning topologies for the detection of fraud in online money transaction.	Proposed model outperformed and prevented the frauds in any online transaction through credit cards	There is a need to improve the accuracy of the proposed algorithm
	Random Forest for Credit Card Fraud Detection	Guanjun Liu, Zhenchuan Li, Lutao Zheng, Shuo Wang and Changjun Jiang Shiyang Xua	2018	The B2C dataset for the identification and detection of fraud from the credit cards	Proposed random forests provide good results on the small dataset	Problems like imbalanced data make it less effective than any other dataset.

1.1 EXISTING SYSTEM

For quite a while, there has been a solid enthusiasm for the morals of banking (Molyneux, 2007; George, 1992), just as the ethical intricacy of fake conduct (Clarke, 1994). Extortion implies getting administrations/products and additionally cash by deceptive methods and is a developing issue everywhere throughout the world these days. Extortion manages cases including criminal purposes that, generally, are hard to recognize. Charge cards are a standout amongst the most well-known focuses of extortion however by all account not the only one; misrepresentation can happen with an acknowledge items, for example, individual advances, home credits, and retail. Besides, the essence of extortion has changed significantly amid the most recent couple of decades as advancements have changed and created. A basic assignment to support organizations and money related establishments including banks is to find a way to anticipateextortion and to manage it productively and adequately when it happens (Anderson, 2007).
Anderson (2007) has identified and explained the different types of fraud, which are as many and varied as the financial institution's products and technologies.

1.1.1 Disadvantages

Fraud detection and prevention software that analyzes patterns of normal and unusual behavior as well as individual transactions in order to flag likely fraud. Profiles include such information as IP address. Technologies have existed since the early 1990s to detect potential fraud. One early market entrant was Falcon; other leading software solutions for card fraud include Actimize, SAS, BAE Systems Detica, and IBM. All these does not utilize the modern machine learning techniques which is far more efficient and more.

1.2 SUMMARY OF EXISTING SYSTEM

For quite a while, there has been a solid enthusiasm for the morals of banking (Molyneux, 2007; George, 1992), just as the ethical intricacy of fake conduct (Clarke, 1994). Extortion implies getting administrations/products and additionally cash by deceptive methods and is a developing issue everywhere throughout the world these days.Extortion manages cases including criminal purposes that, generally, are hard to recognize. Charge cards are a standout

amongst the most well-known focuses of extortionhowever by all account not the only one; misrepresentation can happen with anacknowledge items, for example, individual advances, home credits, and retail. Besides, the essence of extortion has changed significantly amid the most recent couple of decadesas advancements have changed and created. A basic assignment to support organizationsand money related establishments including banks is to find a way to anticipateextortionand to manage it productively and adequately when it happens (Anderson, 2007).

Anderson (2007) has identified and explained the different types of fraud, which are as many and varied as the financial institution's products and technologies.

CHAPTER - 3

SYSTEM ANALYSIS

3.1 PROBLEM STATEMENT

Credit card frauds are increasing heavily because of fraud financial loss is increasing drastically. Every year due to fraud Billions of amounts lost. To analyze the fraud there is lack of research. Many machine learning algorithms are implemented to detect real world credit card fraud. ANN and hybrid algorithms are applied.

3.2 ALGORITHM

3.2.1. RANDOM FOREST:

Decision Trees

Let's quickly go over decision trees as they are the building blocks of the random forest model. Fortunately, they are pretty intuitive. I'd be willing to bet that most people have used a decision tree, knowingly or not, at some point in their lives.

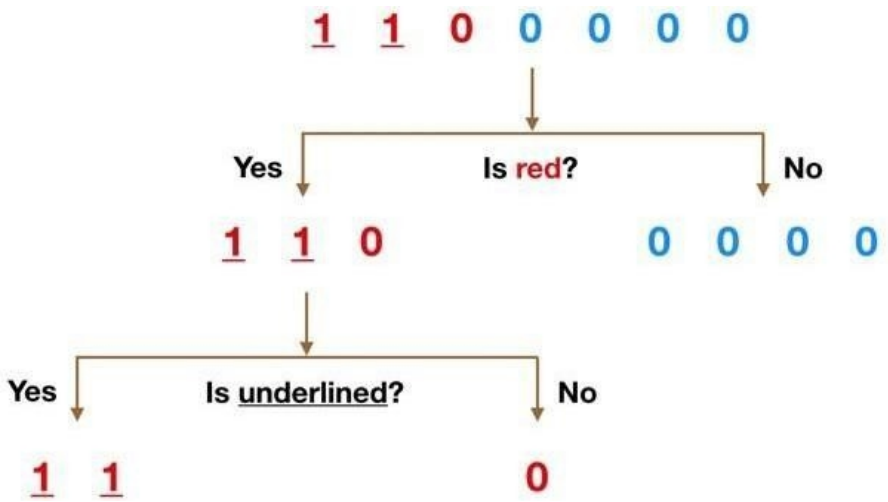


Figure 3.2.1 : Random Forest Model Decision Tree

Simple Decision Tree Example It's probably much easier to understand how a decision tree works through an example. Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their features.The features are color (red vs. blue) and whether the observation is underlined or not. So how can we do this?

Color seems like a pretty obvious feature to split by as all but one of the 0s are blue. So we can use the question, “Is it red?” to split our first node. You can think of a node in a tree as the point where the path splits into two — observations that meet the criteria go down the Yes branch and ones that don’t go down the No branch.

The No branch (the blues) is all 0s now so we are done there, but our Yes branch can still be split further. Now we can use the second feature and ask, “Is it underlined?” to make a second split. The two 1s that are underlined go down the Yes subbranch and the 0 that is not underlined goes down the right subbranch and we are all done. Our decisiontree was able to use the two features to split up the data perfectly. Victory! Obviously in real life our data will not be this clean but the logic that a decision tree employs remains the same. At each node, it will ask —

What feature will allow me to split the observations at hand in a way that the resulting groups are as different from each other as possible (and the members of each resulting subgroup are as similar to each other as possible)?

The Random Forest Classifier

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spitsout a class prediction and the class with the most votes becomes our model’s prediction (see figure below).

Visualization of a Random Forest Model Making a Prediction.The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that aremore accurate than any of the

individual predictions.

The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

- I use a uniformly distributed random number generator to produce a number.

more accurate than any of the individual predictions.

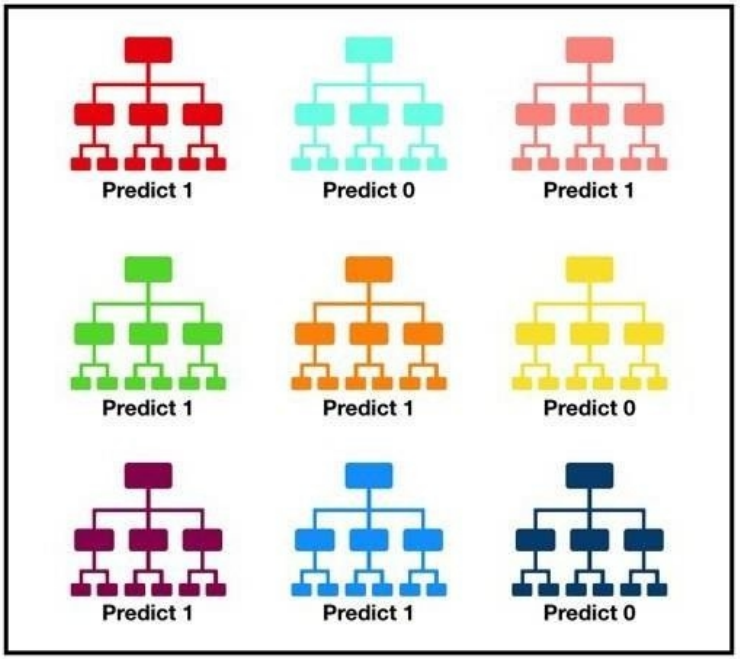


Figure 3.2.2 : Random Forest Prediction

An Example of Why Uncorrelated Outcomes are So Great

The wonderful effects of having many uncorrelated models is such a critical concept that I want to show you an example to help it really sink in. Imagine that we are playing the following game:

- I use a uniformly distributed random number generator to produce a number.
- If the number I generate is greater than or equal to 40, you win (so you have a 60% chance of victory) and I pay you some money. If it is below 40, I win and you pay me the same amount.
- Now I offer you the the following choices. We can either:

1. **Game 1** — play 100 times, betting $1 each time.

2. **Game 2**— play 10 times, betting $10 each time.

3. **Game 3**— play one time, betting $100.

Which would you pick? The expected value of each game is the same:
Expected Value Game 1 = (0.60*1 + 0.40*-1)*100 = 20
Expected Value Game 2= (0.60*10 + 0.40*-10)*10 = 20
Expected Value Game 3= 0.60*100 + 0.40*-100 = 20

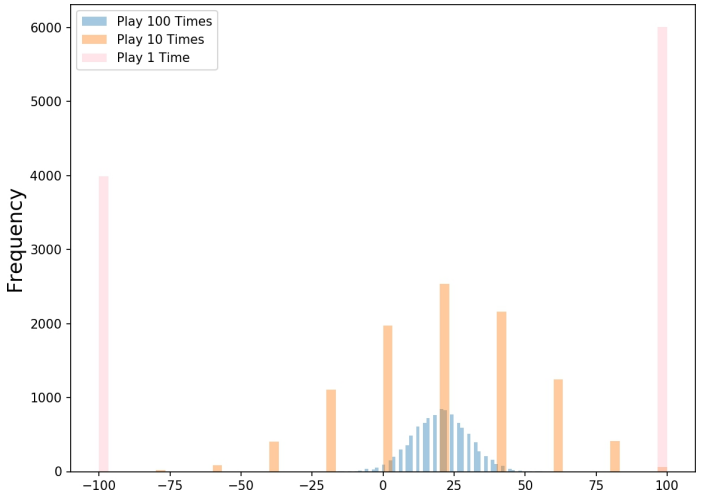


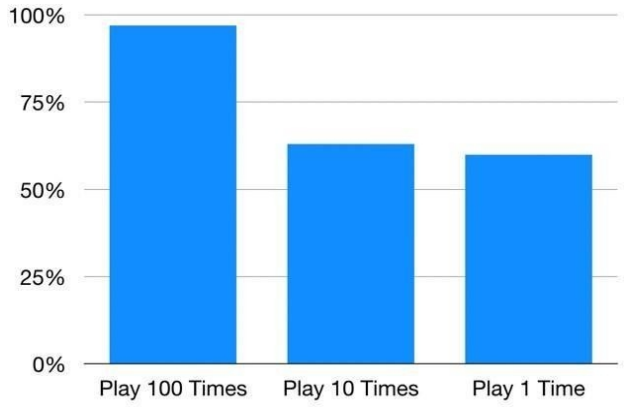
Figure 3..3: Money Won By Player vs Frequency

Outcome Distribution of 10,000 Simulations for each Game What about thedistributions? Let's visualize the results with a Monte Carlo simulation (we will run10,000 simulations of each game type; **for example, we will simulate 10,000 times the 100 plays of Game 1**). Take a look at the chart on the left — now which game would youpick? Even though the expected values are the same, **the outcome distributions are vastly different going from positive and narrow (blue) to binary (pink).**

Game 1 (where we play 100 times) offers up the best chance of making some money

— **out of the 10,000 simulations that I ran, you make money in 97% of them!** For Game 2 (where we play 10 times) you make money in 63% of the simulations, a drastic decline (and a drastic increase in your probability of losing money). And Game 3 that we only play once, you make money in 60% of the simulations, as expected.

Outcome Distribution of 10,000 Simulations for each Game What about the distributions? Let's visualize the results with a Monte Carlo simulation (we will run 10,000 simulations of each game type; **for example, we will simulate 10,000 times the 100 plays of Game 1**). Take a look at the chart on the left — now which game would you pick? Even though the expected values are the same, **the outcome distributions are vastly different going from positive and narrow (blue) to binary (pink).**



Game 1 (where we play 100 times) offers up the best chance of making some money — **out of the 10,000 simulations that I ran, you make money in 97% of them!** For Game 2 (where we play 10 times) you make money in 63% of the simulations, a drastic decline (and a drastic increase in your probability of losing money). And Game 3 that we only play once, you make money in 60% of the simulations, as expected.

Figure 3.4: Probability of Making Money Each Game

So even though the games share the same expected value, their outcome

distributions are completely different. The more we split up our $100 bet into different plays, the more confident we can be that we will make money. As mentionedpreviously, this works because each play is independent of the other ones.

Random forest is the same — each tree is like one play in our game earlier. We just saw how our chances of making money increased the more times we played. Similarly, with a random forest model, our chances of making correct predictions increase with the number of uncorrelated trees in our model. If you would like to run the code for simulating the game yourself you can find it on my GitHub here.

Ensuring that the Models Diversify Each Other. So how does random forest ensure that the behavior of each individual tree is not too correlated with the behavior of any of the other trees in the model? It uses the following two methods:

Bagging (Bootstrap Aggregation) — Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures.

Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging.

Notice that with bagging we are not sub setting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N (unless specified otherwise). But instead of the original training data, we take a random sample of size N with replacement. For example, if our training data was [1, 2, 3, 4, 5, 6] then we might give one of our trees the following list [1, 2, 2, 3, 6, 6]. Notice that both lists are of length six and that “2” and “6” are both repeated in the randomly selected training data we give to our tree (because we sample with replacement).

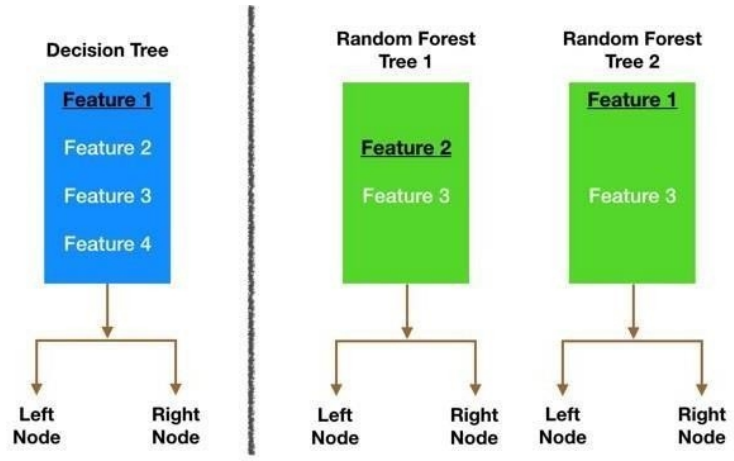


Figure 3.5 : Node Splitting A Random Forest Model

Feature Randomness — In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.

Let's go through a visual example — in the picture above, the traditional decision tree (in blue) can select from all four features when deciding how to split the node. It decides to go with Feature 1 (black and underlined) as it splits the data into groups that are as separated as possible.Now let's take a look at our random forest. We will just examine two of the forest's trees in this example. When we check out random forest Tree 1, we find that it it can only consider Features 2 and 3 (selected randomly) for its node splitting decision. We know from our traditional decision tree (in blue) that Feature 1 is the best feature for splitting, but Tree 1 cannot see Feature 1 so it is forced to go with Feature 2 (black and underlined). Tree 2, on the other hand, can only see Features 1 and 3 so it is able to pick Feature 1.

So in our random forest, we end up with trees that are not only trained on different sets of data (thanks to bagging) but also use different features to make decisions. And that, my dear reader, creates uncorrelated trees that buffer and protect each other from

their errors.

3.2.2 LOCAL OUTLIER FACTOR:

Local Outlier Factor (LOF) is a score that tells how likely a certain data point is an outlier/anomaly.

LOF $\approx 1 \Rightarrow$ no outlier

LOF $\gg 1 \Rightarrow$ outlier

First, I introduce a parameter k which is the number of neighbours the LOF calculation is considering. The LOF is a calculation that looks at the neighbours of a certain point to find out its density and compare this to the density of other points later on. Using a right number k isn't straight forward. While a small k has a more local focus, i.e. looks only at nearby points, it is more erroneous when having much noise in the data. A large k, however, can miss local outliers.

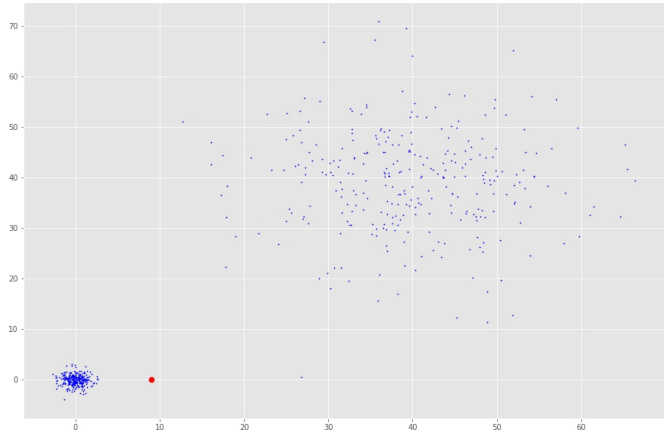


Figure 3.6 : Local Outlier Factor

The density of the red point to its nearest neighbors is not different from the density to the cloud in the upper right corner. However, it is probably an outlier compared to the nearest neighbors' density.

k-distance

With this *k* defined, we can introduce the *k-distance* which is the distance of a point to its *kth* neighbor. If *k* was 3, the *k-distance*would be the distance of a point to the third closest point .The red point's k-distance is illustrated by the red line if k=3.

Reachability distance

The *k-distance* is now used to calculate the reachability distance. This distance measure is simply the maximum of the distance of two points and the *k-distance* of the second point. reach-dist(a,b) = max{k-distance(b), dist(a,b)}

Basically, if point *a* is within the *k* neighbors of point *b*, the *reach-dist(a,b)* will be the *k-distance* of *b*. Otherwise

Local reachability density

The *reach-dist* is then used to calculate still another concept — the local reachability density (lrd). To get the lrd for a point *a*, we will first calculate the reachability distance of *a* to all its *k* nearest neighbors and take the average of that number. The lrd is then simply the inverse of that average.lrd(a) = 1/(sum(reach-dist(a,n))/k)

By intuition the local reachability density tells how far we have to travel from our

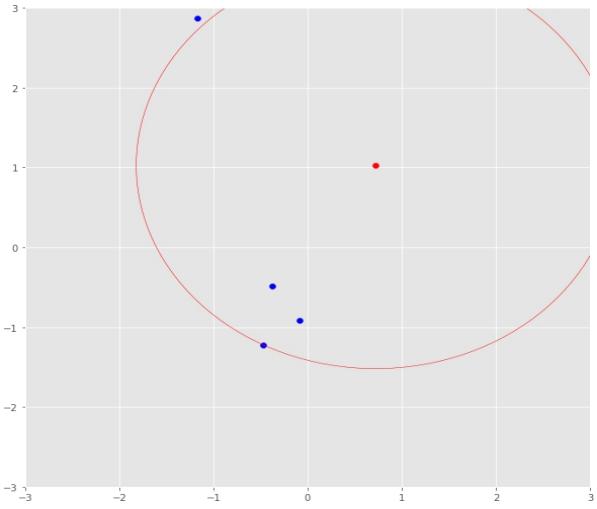


Figure 3.7 : Local Rechability Density

point to reach the next point or cluster of points. The lower it is, the less dense it is, the longer we have to travel.

v The lrd of the upper right point is the average reachability distance to its nearest neighbors which are points (-1, -1), (-1.5, -1.5) and (-1, -2). These neighbors, however, have other lrds as their nearest neighbors don't include the upper right point.

LOF (Local Outlier Factor)

The lrd of each point will then be compared to the lrd of their *k* neighbors. More specifically, *k* ratios of the lrd of each point to its neighboring points will be calculated and

averaged. The LOF is basically the average ratio of the lrds of the neighbors of *a* to the lrd of *a*. If the ratio is greater than *1*, the density of point *a* is on average smaller than the density of its neighbors and, thus, from point *a*, we have to travel longer distances to get to the next point or cluster of points than from *a*'s neighbors to their next neighbors. Keep in mind, the neighbors of a point *a* may don't consider *a* a neighbor as they have points in their reach which are way closer. In conclusion, the LOF of a point tells the density of this point compared to the density of its neighbors. If the density of a point is much smaller than the densities of its neighbors (LOF $\gg$1), the point is far from dense areas and, hence, an outlier.

3.3 SOFTWARE MODULES

3.3.1 NUMPY:

NumPy is a library for the Python programming language, adding support for large, multi- dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

The Python programming language was not initially designed for numerical computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier. An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become Numeric,[4] also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called Numarray was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.
There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then.

In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006.[7] This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0.[13] In 2011, PyPy started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with NumPy.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[16] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars.

In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Traits

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[16] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging

The ndarray data structure

The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data

structure. These arrays are strided views on memory.[7] In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory- mapped ndarrays.Limitations

Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy's np.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.[17]

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include scipy.weave , numexpr[18] and Numba.[19] Cython and Pythran are static-compiling alternatives to these.

The Basics

NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called *axes*.For example, the coordinates

of a point in 3D space [1, 2, 1] has one axis. That axis has 3 elements in it, so we say it has a length of 3. In the example pictured below, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

NumPy's array class is called ndarray. It is also known by the alias array. Note that numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an ndarray object are:

ndarray.ndim

the number of axes (dimensions) of the array.

ndarray.shape

the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n*rows and *m* columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

ndarray.size

the total number of elements of the array. This is equal to the product of the elements of shape.

ndarray.dtype

an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

ndarray.itemsize

the size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

ndarray.data

```
>>>
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0, 1, 2, 3,  4],
       [ 5, 6, 7, 8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
```

the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

```
>
>
>


a
.
d
t
y
p
e
.
```

Array Creation

There are several ways to create arrays.

For example, you can create an array from a regular Python list or tuple using the `array` function. The type of the resulting array is deduced from the type of the elements in the sequences.

```
>>>
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
```

```
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

A frequent error consists in calling `array` with multiple numeric arguments, rather than providing a single list of numbers as an argument.

```
>>>
>>> a = np.array(1,2,3,4)    # WRONG
>>> a = np.array([1,2,3,4]) # RIGHT
```

`array` transforms sequences of sequences into two-dimensional arrays, sequences of sequences of sequences into three-dimensional arrays, and so on.

```
>>>
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5, 2. , 3. ],
 [ 4. , 5. , 6. ]])
```

The type of the array can also be explicitly specified at creation time:

```
>>>
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content. These

```
>>>
>>>
np.zeros
( (3,4) )
                                    # dtype can also be
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],                  # uninitialized
array([[ 3.73603959e-262,
        6.02658058e-154,  6.55490914e-
```

minimize the necessity of growing arrays, an expensive operation.

The function zeros creates an array full of zeros, the function ones creates an array full of ones, and the function empty creates an array whose initial content is random and depends on the state of the memory. By default, the dtype of the created array isfloat64.eturns arrays instead of lists.

```
>>>
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 )                 # it accepts float arguments
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5,  1.8])
```

When arange is used with floating point arguments, it is generally not possible to predict the number of elements obtained, due to the finite floating point precision. For

this reason, it is usually better to use the function linspace that receives as an argument the number of elements that we want, instead of the step:

```
>>>
>>> from numpy import pi
>>> np.linspace( 0, 2, 9 )                 # 9 numbers from 0 to 2
array([ 0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2.  ])
>>> x = np.linspace( 0, 2*pi, 100 )        # useful to evaluate function at lots of points
>>> f = np.sin(x)
```

3.3.2 PANDAS

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. **pandas** is a Python package providing fast, flexible, and expressive data structures

designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, **Series** (1-dimensional) and **DataFrame** (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, **DataFrame** provides everything that R’s data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.
Here are just a few of the things that pandas does well:

- Easy handling of **missing data** (represented as NaN) in floating point as well as non-floating-point data
- Size mutability: columns can be **inserted and deleted** from DataFrame and higher dimensional objects
- Automatic and explicit **data alignment**: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let *Series*, DataFrame, etc. automatically align the data for you in computations
- Powerful, flexible **group by** functionality to perform split-apply-

combine operations on data sets, for both aggregating and transforming data

- Make it **easy to convert** ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets
- Intuitive **merging** and **joining** data sets
- Flexible **reshaping** and pivoting of data sets
- **Hierarchical** labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from **flat files** (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast **HDF5 format**
- **Time series**-specific functionality: date range generation and frequency conversion, moving window statistics, moving window lineaegressions, date shifting and lagging, etc.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.
Some other notes

- pandas is **fast**. Many of the low-level algorithmic bits have been extensively tweaked in Cythoncode. However, as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool.
- pandas is a dependency of statsmodels, making it an important part of the statistical computing ecosystem in Python.
- pandas has been used extensively in production in financial applications.

Data Structures

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

Why more than one data structure?

The best way to think about the pandas data structures is as flexible containers for lower dimensional data. For example, DataFrame is a container for Series, and Series is a container for scalars. We would like to be able to insert and remove objects from these containers in a dictionary-like fashion.

Also, we would like sensible default behaviors for the common API functions which take into account the typical orientation of time series and cross-sectional data sets. When using ndarrays to store 2- and 3-dimensional data, a burden is placed on the user to consider the orientation of the data set when writing functions; axes are considered more or less equivalent (except when C- or Fortran-contiguousness matters for performance). In pandas, the axes are intended to lend more semantic meaning to the data; i.e., for a particular data set there is likely to be a "right" way to orient the data. The goal, then, is to reduce the amount of mental effort required to code up data transformations in downstream functions.

For example, with tabular data (DataFrame) it is more semantically helpful to think of the **index** (the rows) and the **columns** rather than axis 0 and axis 1. Iterating through the columns of the DataFrame thus results in more readable code:

```
for col in df.columns:
    series = df[col]
    # do something with series
```

Mutability and copying of data

All pandas data structures are value-mutable (the values they contain can be altered) but not always size-mutable. The length of a Series cannot be changed, but, for example, columns can be inserted into a DataFrame. However, the vast majority of methods produce new objects and leave the input data untouched. In general we like to **favor immutability** where sensible.

Intro to Data Structures

We'll start with a quick, non-comprehensive overview of the fundamental data structures in pandas to get you started. The fundamental behavior about data types, indexing, and axis labeling / alignment apply across all of the objects. To get started,

import NumPy and load pandas into your namespace:

```
In [1]: import numpy as np


In [2]: import pandas as pd
```

Here is a basic tenet to keep in mind: **data alignment is intrinsic**. The link between labels and data will not be broken unless done so explicitly by you. We'll give a brief intro to the data structures, then consider all of the broad categories of functionality and methods in separate sections.

Series

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are

collectively referred to as the **index**. The basic method to create a Series is to call:

```
>>> s = pd.Series(data, index=index)
```

Here, data can be many different things:

- a Python dict
- an ndarray
- a scalar value (like 5)

The passed **index** is a list of axis labels. Thus, this separates into a few cases depending on what **data is**:

From ndarray

If data is an ndarray, **index** must be the same length as **data**. If no index is passed, one will be created having values [0, ..., len(data) -

```
In [3]: s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd',
'e'])



In [4]: s

Out[4]:

a   0.469112

b  -0.282863

c  -1.509059

d  -1.135632

e   1.212112
].
```

```
4  -2.104569

dtype: float64
```

Note

pandas supports non-unique index values. If an operation that does not support duplicate index values is attempted, an exception will be raised at that time. The reason for being lazy is nearly all performance-based (there are many instances in computations, like parts of GroupBy, where the index is not used).

From dict

Series can be instantiated from dicts:

```
In [7]: d = {'b': 1, 'a': 0, 'c': 2}


In [8]: pd.Series(d)

Out[8]:

b    1

a    0

c    2

dtype: int64
```

Note

When the data is a dict, and an index is not passed, the Series index will be ordered by the dict's insertion order, if you're using Python version >= 3.6 and Pandas version >= 0.23.

If you're using Python < 3.6 or Pandas < 0.23, and an index is not passed, the Series index will be the lexically ordered list of dict keys.

In the example above, if you were on a Python version lower than 3.6 or a Pandas version lower than 0.23, the Series would be ordered by the lexical order of the dict keys (i.e. ['a', 'b', 'c'] rather than ['b', 'a', 'c']).

If an index is passed, the values in data corresponding to the labels in the index will be pulled out.

```
In [9]: d = {'a': 0., 'b': 1., 'c': 2.}



In [10]: pd.Series(d)

Out[10]:

a    0.0

b    1.0

c    2.0

dtype: float64



In [11]: pd.Series(d, index=['b', 'c', 'd', 'a'])

Out[11]:

b    1.0

c    2.0

d    NaN

a    0.0

dtype: float64
```

Note

NaN (not a number) is the standard missing data marker used in pandas.

From scalar value

If data is a scalar value, an index must be provided. The value will be repeated to match the length of **index**.

In [12]: pd.Series(5., index=['a', 'b', 'c', 'd', 'e'])

Out[12]:

a 5.0

b 5.0

c 5.0

d 5.0

e 5.0

dtype: float64

Series is ndarray-like

Series acts very similarly to a ndarray, and is a valid argument to most NumPy functions. However, operations such as slicing will also slice the index.

In [13]: s[0]

Out[13]: 0.46911229990718628

In [14]: s[:3]

Out[14]:

```
a    0.469112

b   -0.282863

c   -1.509059

dtype: float64
```

In [15]: s[s > s.median()]

Out[15]:

```
a    0.469112

e    1.212112

dtype: float64
```

In [16]: s[[4, 3, 1]]

Out[16]:

```
e    1.212112

d   -1.135632

b   -0.282863

dtype: float64
```

In [17]: np.exp(s)

Out[17]:

```
a    1.598575
b    0.753623
c    0.221118
```

```
d    0.321219

e    3.360575

dtype: float64
```

Note

We will address array-based indexing like s[[4, 3, 1]] in section.

Like a NumPy array, a pandas Series has a **dtype**.

```
In [19]: s.array

Out[19]:

<PandasArray>

[ 0.46911229990718628, -0.28286334432866328,  -1.5090585031735124,

 -1.1356323710171934,  1.2121120250208506]
```

Accessing the array can be useful when you need to do some operation without the index (to disable automatic alignment, for example).

Series.array will always be an **ExtensionArray**. Briefly, an ExtensionArray is a thin wrapper around one or more *concrete* arrays like a **numpy.ndarray**. Pandas knows

how to take an ExtensionArray and store it in a Series or a column of a DataFrame. See dtypes for more.

While Series is ndarray-like, if you need an *actual* ndarray, then use **Series.to_numpy()**.

```
In [20]: s.to_numpy()

Out[20]: array([ 0.4691, -0.2829, -1.5091, -1.1356, 1.2121])
```

Even if the Series is backed by a **ExtensionArray**, **Series.to_numpy()** will return a NumPy ndarray.

```
In [21]: s['a']

Out[21]: 0.46911229990718628


In [22]: s['e'] = 12.


In [23]: s

Out[23]:

a    0.469112

b   -0.282863

c   -1.509059
```

A Series is like a fixed-size dict in that you can get and set values by index label:

```
dtype: float64


In [24]: 'e' in s

Out[24]: True


In [25]: 'f' in s

Out[25]: False
```

If a label is not contained, an exception is raised:

```
>>> s['f']

KeyError: 'f'
```

Using the get method, a missing label will return None or specified default:

```
In [26]: s.get('f')

In [27]: s.get('f', np.nan)

Out[27]: nan
```

See also the section on attribute access.

Vectorized operations and label alignment with Series

When working with raw NumPy arrays, looping through value-by-value is usually not necessary. The same is true when working with Series in pandas. Series can also be passed into most NumPy methods expecting an ndarray.

In [28]: s + s

Out[28]:

a 0.938225

b -0.565727

c -3.018117

d -2.271265

e 24.000000

dtype: float64

In [29]: s * 2

Out[29]:

a 0.938225

b -0.565727

c -3.018117

d -2.271265

e 24.000000

dtype: float64

```
In [30]: np.exp(s)
Out[30]:
a         1.598575

b         0.753623

c         0.221118

d         0.321219

e    162754.791419

dtype: float64
```

A key difference between Series and ndarray is that operations between Series automatically align the data based on label. Thus, you can write computations without giving consideration to whether the Series involved have the same labels.

```
In [31]: s[1:] + s[:-1]

Out[31]:

a    NaN

b   -0.565727

c   -3.018117

d   -2.271265

e     NaN

dtype: float64
```

The result of an operation between unaligned Series will have the **union** of the indexes involved. If a label is not found in one Series or the other, the result will be marked as missing NaN. Being able to write code without doing any explicit data alignment grants immense freedom and flexibility in interactive data analysis and research. The integrated

data alignment features of the pandas data structures set pandas apart from the majority of related tools for working with labeled data.

Note

In general, we chose to make the default result of operations between differently indexed objects yield the**union** of the indexes in order to avoid loss of information. Having an index label, though the data is missing, is typically important information as part of a computation. You of course have the option of dropping labels with missing data via the **dropna** function.

Name attribute

Series can also have a name attribute:

```
In [32]: s = pd.Series(np.random.randn(5), name='something')

In [33]: s

Out[33]:

0   -0.494929

1    1.071804

2    0.721555

3   -0.706771

4   -1.039575

Name: something, dtype: float64

In [34]: s.name

Out[34]: 'something'
```

The Series name will be assigned automatically in many cases, in particular when taking 1D slices of DataFrame as you will see below.

New in version 0.18.0.

You can rename a Series with the **pandas.Series.rename()** method.

```
In [35]: s2 = s.rename("different")



In [36]: s2.name
```

```
Out[36]: 'different'
```

Note that s and s2 refer to different objects.

DataFrame

DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

Along with the data, you can optionally pass **index** (row labels) and **columns** (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

If axis labels are not passed, they will be constructed from the input data based on common sense rules.

Note

From dict of Series or dicts

The resulting **index** will be the **union** of the indexes of the various Series. If there are any nested dicts, these will first be converted to Series. If no columns are passed, the

```
In [37]: d = {'one': pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
   ....:      'two': pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
   ....:


In [38]: df = pd.DataFrame(d)


In [39]: df
Out[39]:
   o
n
e
```

columns will be the ordered list of dict keys.

```
d NaN  4.0

b  2.0  2.0

a  1.0  1.0
```

```
In [41]: pd.DataFrame(d, index=['d', 'b', 'a'], columns=['two',
'three'])

Out[41]:

   t

w

o
```

The row and column labels can be accessed respectively by accessing the **index** and **columns** attributes:

When a particular set of columns is passed along with a dict of data, the passed columns override the keys in the dict.

```
In [42]: df.index

Out[42]: Index(['a', 'b', 'c', 'd'], dtype='object')



In [43]: df.columns

Out[43]: Index(['one', 'two'], dtype='object')
```

Note

From dict of ndarrays / lists

The ndarrays must all be the same length. If an index is passed, it must clearly also be the same length as the arrays. If no index is passed, the result will be range(n), where n is the array length.

```
In [44]: d = {'one': [1., 2.,
3., 4.],

   ....:      'two': [4., 3., 2.,
   1.]}



In [45]:
pd.DataFrame(d)
```

```
   one  two

 0 1.0  4.0

 1 2.0  3.0
                ataFrame(d, index=['a',
 2 3.0  2.0

 3 4.0  1.0

  o

n
```

```
c  3.0  2.0

d  4.0  1.0
```

From structured or record array

This case is handled identically to a dict of arrays.

```
In [47]: data = np.zeros((2, ), dtype=[('A', 'i4'), ('B',


In [48]: data[:] = [(1, 2., 'Hello'), (2,


In [49]:
pd.DataFrame(data)


  A          C

0  1 2.0
      b'Hello'
1


In [50]: pd.DataFrame(data,
index=['first', 'second'])


      A          C

first 1 2.0
b'Hello'
```

```
In [51]: pd.DataFrame(data, columns=['C', 'A', 'B'])

Out[51]:

        C  A    B

0  b'Hello' 1  2.0

1  b'World' 2  3.0
```

Note

DataFrame is not intended to work exactly like a 2-dimensional NumPy ndarray.

From a list of dicts

```
In [52]: data2 = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]


In [53]: pd.DataFrame(data2)

Out[53]:

   a   b     c

0  1   2   NaN

1 5 10 20.0


In [54]: pd.DataFrame(data2, index=['first', 'second'])

Out[54]:

       a   b     c
```

```
first 1 2  NaN

second 5 10  20.0



In [55]: pd.DataFrame(data2, columns=['a', 'b'])

Out[55]:

   a  b

0  1   2

1  5  10
```

From a dict of tuples

```
In [56]: pd.DataFrame({('a', 'b'):

...          ('a', 'a'): {('A', 'C'):

...          ('a', 'c'): {('A', 'B'):

...          ('b', 'a'): {('A', 'C'):

...          ('b', 'b'): {('A', 'D'): 9,

...

Out[

  a       b

  b   a   c
```

You can automatically create a MultiIndexed frame by passing a tuples dictionary.

```
A B  1.0  4.0 5.0  8.0  10.0

 C  2.0  3.0  6.0  7.0   NaN

 D NaN NaN NaN NaN  9.0
```

Assigning New Columns in Method Chains

Inspired by dplyr's mutate verb, DataFrame has an **assign()** method that allows you to easily create new columns that are potentially derived from existing columns.

In [74]: iris =

In [75]:

Out|7

	SepalLength	SepalWidth		Na
0	5.1	3.5	1.4	0.2 Iris-setosa
1	4.9	3.0	1.4	0.2 Iris-setosa
2	4.7	3.2	1.3	0.2 Iris-setosa

3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

....: .head())

....:

Out[76]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name	sepal_ratio
0	5.1	3.5	1.4	0.2	Iris-setosa	0.686275
1	4.9	3.0	1.4	0.2	Iris-setosa	0.612245
2	4.7	3.2	1.3	0.2	Iris-setosa	0.680851
3	4.6	3.1	1.5	0.2	Iris-setosa	0.673913
4	5.0	3.6	1.4	0.2	Iris-setosa	0.720000

In the example above, we inserted a precomputed value. We can also pass in a function of one argument to be evaluated on the DataFrame being assigned to.

In [77]: iris.assign(sepal_ratio=**lambda** x: (x['SepalWidth'] /

Out[7

	SepalLength	SepalWidth	PetalLength		Name	
0	5.1	3.5	1.4	0.2	Iris-setosa	0.686275
1	4.9	3.0	1.4	0.2	Iris-setosa	0.612245

```
2    4.7    3.2    1.3    0.2  Iris-setosa    0.680851
3    4.6    3.1    1.5    0.2  Iris-setosa    0.673913
4    5.0    3.6    1.4    0.2  Iris-setosa    0.720000
```

assign **always** returns a copy of the data, leaving the original DataFrame untouched.

Passing a callable, as opposed to an actual value to be inserted, is useful when you don't have a reference to the DataFrame at hand. This is common when using assign in a chain of operations. For example, we can limit the DataFrame to just those observations with a Sepal Length greater than 5, calculate the ratio, and plot:

```
In [78]: (iris.query('SepalLength > 5')
   ....:      .assign(SepalRatio=lambda x: x.SepalWidth / x.SepalLength,
   ....:              PetalRatio=lambda x: x.PetalWidth / x.PetalLength)
   ....:      .plot(kind='scatter', x='SepalRatio', y='PetalRatio'))
   ....:
Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2b527b1a58>
```

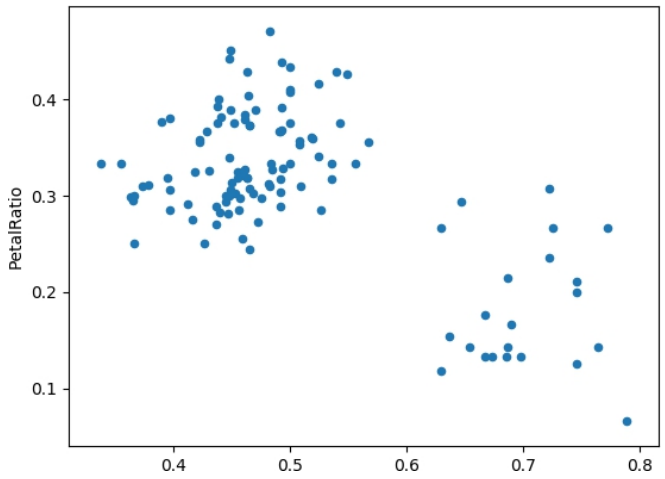


Figure 3.8 : Data Frame Sepal and Petal Ratio

Since a function is passed in, the function is computed on the DataFrame being assigned to. Importantly, this is the DataFrame that's been filtered to those rows with sepal length greater than 5. The filtering happens first, and then the ratio calculations. This is an example where we didn't have a reference to the *filtered* DataFrameavailable. The function signature for assign is simply **kwargs. The keys are the column names for the new fields, and the values are either a value to be inserted (for example, a Series or NumPy array), or a function of one argument to be called on the DataFrame. A *copy* of the original DataFrame is returned, with the new values inserted.
Changed in version 0.23.0.

Starting with Python 3.6 the order of **kwargs is preserved. This allows for *dependent* assignment, where an expression later in **kwargs can refer to a column created earlier in the same **assign()**.

```
In [79]: dfa =

   ....:              "B": [4, 5,

   ....:

In [80]: dfa.assign(C=lambda x:

   ....:        D=lambda x: x['A']

   ....:

Out[8

  A B C

0 1 4

1 2 5

2 3 6 9
```

In the second expression, x['C'] will refer to the newly created column, that's equal to dfa['A'] + dfa['B'].

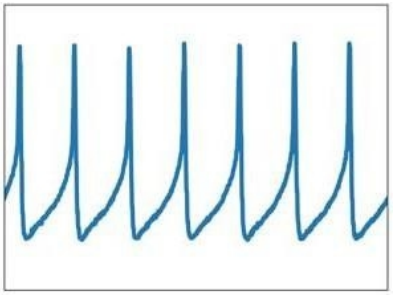
To write code compatible with all versions of Python, split the assignment in two.

```
In [81]: dependent = pd.DataFrame({"A": [1, 1, 1]})
```

```
In [82]: (dependent.assign(A=lambda x: x['A'] + 1)
   ....:          .assign(B=lambda x: x['A'] + 2))
   ....:
   A  B
0  2  4
1  2  4
2  2  4
```

3.4 Matplotlib:

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.



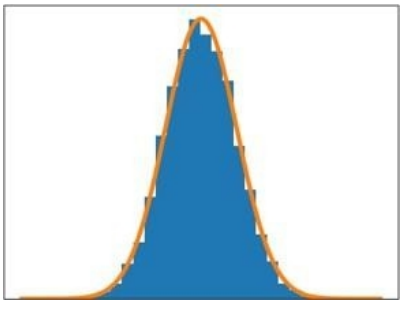


Figure 3.9 : Python Matplotlib 2d Plotting Library

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Installing an official release

Matplotlib and its dependencies are available as wheel packages for macOS, Windows and Linux distributions:

```
python -m pip install -U pip
python -m pip install -U  matplotlib
```

Although not required, we suggest also installing IPython for interactive use. To easily install a complete Scientific Python stack, see Scientific Python Distributions below.

macOS

To use the native OSX backend you will need a framework build build of Python.

Test data

The wheels (*.whl) on the PyPI download page do not contain test data or example code. If you want to try the many demos that come in the Matplotlib source distribution, download the *.tar.gz file and look in the examples subdirectory.

To run the test suite:

- extract the lib/matplotlib/tests or lib/mpl_toolkits/tests directories from the source distribution;
- install test dependencies: pytest, Pillow, MiKTeX, GhostScript, ffmpeg, avconv, ImageMagick, and Inkscape;
- run python -mpytest.

3.4.1 Third-party distributions of Matplotlib

3.4.2 Scientific Python Distributions

Anaconda and Canopy and ActiveState are excellent choices that "just work" out of the box for Windows, macOS and common Linux platforms. WinPython is an option for Windows users. All of these distributions include Matplotlib and *lots* of other useful (data) science tools.

Linux: using your package manager

If you are on Linux, you might prefer to use your package manager. Matplotlib is packaged for almost every major Linux distribution.

- Debian / Ubuntu: sudo apt-get install python3-matplotlib
- Fedora: sudo dnf install python3-matplotlib
- Red Hat: sudo yum install python3-matplotlib
- Arch: sudo pacman -S python-matplotlib

Installing from source

If you are interested in contributing to Matplotlib development, running thelatest

source code, or just like to build everything yourself, it is not difficult to build Matplotlib from source. Grab the latest *tar.gz* release file from the PyPI files page, or if you want to develop Matplotlib or just need the latest bugfixed version, grab the latest git version Install from source.The standard environment variables CC, CXX, PKG_CONFIG are respected. This means you can set them if your toolchain is prefixed. This may be used for cross compiling.

```
export CC=x86_64-pc-linux-gnu-gcc
export CXX=x86_64-pc-linux-gnu-g++
export PKG_CONFIG=x86_64-pc-linux-gnu-pkg-config
```

Once you have satisfied the requirements detailed below (mainly Python, NumPy, libpng and FreeType), you can build Matplotlib.

```
cd matplotlib
python -mpip install .
```

We provide a setup.cfg file which you can use to customize the build process. For example, which default backend to use, whether some of the optional libraries that Matplotlib ships with are installed, and so on. This file will be particularly useful to those packaging Matplotlib.If you have installed prerequisites to nonstandard places and need to inform Matplotlib where they are, edit setupext.py and add the base dirs to the basedir dictionary entry for your sys.platform; e.g., if the header of some required library is in /some/path/include/someheader.h, put /some/path in the basedir list for your platform.

Dependencies

Matplotlib requires the following dependencies:

- Python (>= 3.5)
- FreeType (>= 2.3)
- libpng (>= 1.2)
- NumPy (>= 1.10.0)
- setuptools

- cycler (>= 0.10.0)
- dateutil (>= 2.1)
- kiwisolver (>= 1.0.0)
- pyparsing

Optionally, you can also install a number of packages to enable better user interface toolkits. See What is a backend? for more details on the optional Matplotlib backends and the capabilities they provide.

- tk (>= 8.3, != 8.6.0 or 8.6.1): for the Tk-based backends;
- PyQt4 (>= 4.6) or PySide (>= 1.0.3): for the Qt4-based backends;
- PyQt5: for the Qt5-based backends;
- PyGObject or pgi (>= 0.0.11.2): for the GTK3-based backends;
- wxpython (>= 4): for the WX-based backends;
- cairocffi (>= 0.8) or pycairo: for the cairo-based backends;
- Tornado: for the WebAgg backend;

For better support of animation output format and image file formats, LaTeX, etc., you can install the following:

- ffmpeg/avconv: for saving movies;
- ImageMagick: for saving animated gifs;
- Pillow (>= 3.4): for a larger selection of image file formats: JPEG, BMP, and TIFF image files;
- LaTeX and GhostScript (>=9.0) : for rendering text with LaTeX.

Building on Linux

It is easiest to use your system package manager to install the dependencies.
If you are on Debian/Ubuntu, you can get all the dependencies required to build Matplotlib

with:

```
sudo apt-get build-dep python-matplotlib
```

If you are on Fedora, you can get all the dependencies required to build Matplotlib with:

```
sudo dnf builddep python-matplotlib
```

If you are on RedHat, you can get all the dependencies required to build Matplotlib by first installing yum-builddep and then running:

```
su -c "yum-builddep python-matplotlib"
```

These commands do not build Matplotlib, but instead get and install the build dependencies, which will make building from source easier.

Building on macOS

The build situation on macOS is complicated by the various places one can get the libpng and FreeType requirements (MacPorts, Fink, /usr/X11R6), the different architectures (e.g., x86, ppc, universal), and the different macOS versions (e.g., 10.4 and 10.5). We recommend that you build the way we do for the macOS release: get the source from the tarball or the git repository and install the required dependencies through a third-party package manager. Two widely used package managers are Homebrew, and MacPorts. The following example illustrates how to install libpng and FreeType using brew:

```
brew install libpng freetype pkg-config
```

If you are using MacPorts, execute the following instead:

```
port install libpng freetype pkgconfig
```

After installing the above requirements, install Matplotlib from source by executing:

```
python -mpip install .
```

Note that your environment is somewhat important. Some conda users have found that, to run the tests, their PYTHONPATH must include /path/to/anaconda/.../site-packages and their DYLD_FALLBACK_LIBRARY_PATH must include /path/to/anaconda/lib.

Building on Windows

The Python shipped from https://www.python.org is compiled with Visual Studio 2015 for 3.5+. Python extensions should be compiled with the same compiler, see e.g. https://packaging.python.org/guides/packaging-binary-extensions/#setting-up-a-build-environment-on-windows for how to set up a build environment.

Since there is no canonical Windows package manager, the methods for building FreeType, zlib, and libpng from source code are documented as a build script at matplotlib-winbuild.

There are a few possibilities to build Matplotlib on Windows:

- Wheels via matplotlib-winbuild
- Wheels by using conda packages (see below)
- Conda packages (see below)

Wheel builds using conda packages

This is a wheel build, but we use conda packages to get all the requirements. The binary requirements (png, FreeType,...) are statically linked and therefore not needed during the wheel install.Set up the conda environment. Note, if you want a qt backend, add pyqt to the list of conda packages.

```
conda create -n "matplotlib_build" python=3.7 numpy python-dateutil
pyparsing tornado cycler tk libpng zlib freetype msinttypes
conda activate matplotlib_build
```

For building, call the script build_alllocal.cmd in the root folder of the repository:

```
build_alllocal.cmd
```

General Concepts

matplotlib has an extensive codebase that can be daunting to many new users. However, most of matplotlib can be understood with a fairly simple conceptual framework and knowledge of a few important points.

Plotting requires action on a range of levels, from the most general (e.g., 'contour this 2-D array') to the most specific (e.g., 'color this screen pixel red'). The purpose of a plotting package is to assist you in visualizing your data as easily as possible, with all the necessary control -- that is, by using relatively high-level commands most of the time, and still have the ability to use the low-level commands when needed.

Therefore, everything in matplotlib is organized in a hierarchy. At the top of the hierarchy is the matplotlib "state-machine environment" which is provided by the matplotlib.pyplot module. At this level, simple functions are used to add plot elements (lines, images, text, etc.) to the current axes in the current figure.

Note

Pyplot's state-machine environment behaves similarly to MATLAB and should be most familiar to users with MATLAB experience.

The next level down in the hierarchy is the first level of the object-oriented interface, in which pyplot is used only for a few functions such as figure creation, and the user explicitly creates and keeps track of the figure and axes objects. At this level, the user uses pyplot to create figures, and through those figures, one or more axes objects can be created. These axes objects are then used for most plotting actions.

For even more control -- which is essential for things like embedding matplotlib plots in GUI applications -- the pyplot level may be dropped completely, leaving a purely object- oriented approach.

```
# sphinx_gallery_thumbnail_number = 3
import matplotlib.pyplot as plt
import numpy as np
```

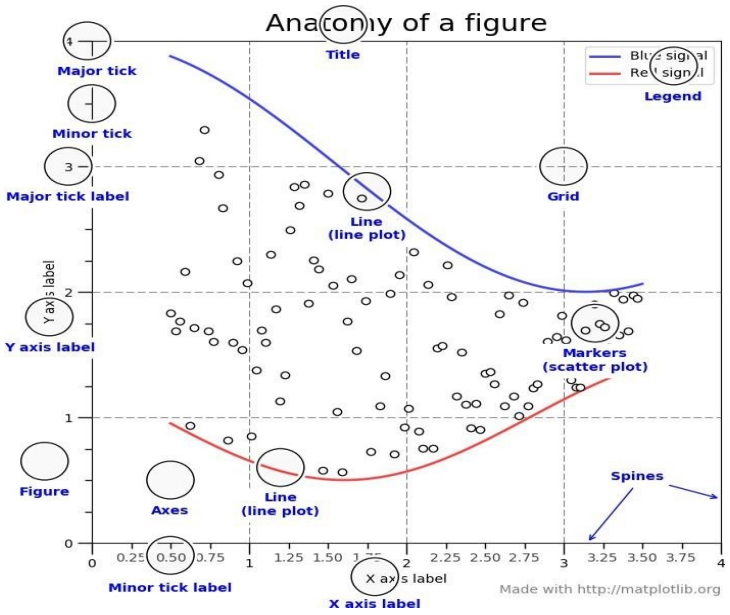


Figure 3.10 : All Child Axes, a smattering of 'special' artists and the canvas

The **whole** figure. The figure keeps track of all the child Axes, a smattering of 'special' artists (titles, figure legends, etc), and the **canvas**. (Don't worry too much about the canvas, it is crucial as it is the object that actually does the drawing to get you your plot, but as the user it is more-or-less invisible to you). A figure can have any number of Axes, but to be useful should have at least one.

3.5 SEABORN:

Seaborn is a Python data visualization library based on matplotlib. It provides a high- level interface for drawing attractive and informative statistical graphics.

3.5.1 An introduction to seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

Here is some of the functionality that seaborn offers:

- A dataset-oriented API for examining relationships between multiple variables
- Specialized support for using categorical variables to

show observations or aggregate statistics

- Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
- Automatic estimation and plotting of linear regression models for different kinds dependent variables
- Convenient views onto the overall structure of complex datasets
- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Here's an example of what this means:

```
import seaborn as sns

sns.set()

tips = sns.load_dataset("tips")

sns.relplot(x="total_bill", y="tip",

col="time",
```

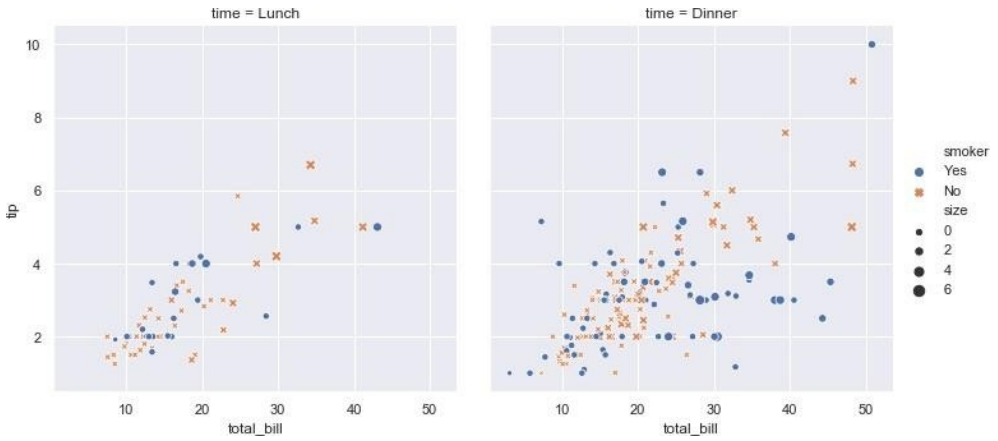


Figure 3.11 : Seaborn for time=lunch, time=dinner

1. We import seaborn, which is the only library necessary for this simple example.

```
import seaborn as sns
```

Behind the scenes, seaborn uses matplotlib to draw plots. Many tasks can be accomplished with only seaborn functions, but further customization might require using matplotlib directly. This is explained in more detail below. For interactive work, it's recommended to use a Jupyter/IPython interface in matplotlib mode, or else you'll have to call matplotlib.pyplot.show when you want to see the plot. We apply the default default seaborn theme, scaling, and color palette.

```
sns.set()
```

This uses the matplotlib rcParam system and will affect how all matplotlib plots look, even if you don't make them with seaborn. Beyond the default theme, there are several other options, and you can independently control the style and scaling of the plot to quickly translate your work between presentation contexts (e.g., making a plot that will have readable fonts when projected during a talk). If you like the matplotlib defaults or prefer a different theme, you can skip this step and still use the seaborn plotting functions.

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM DESIGN: PROPOSED SYSTEM FLOW CHART

A **flowchart** is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

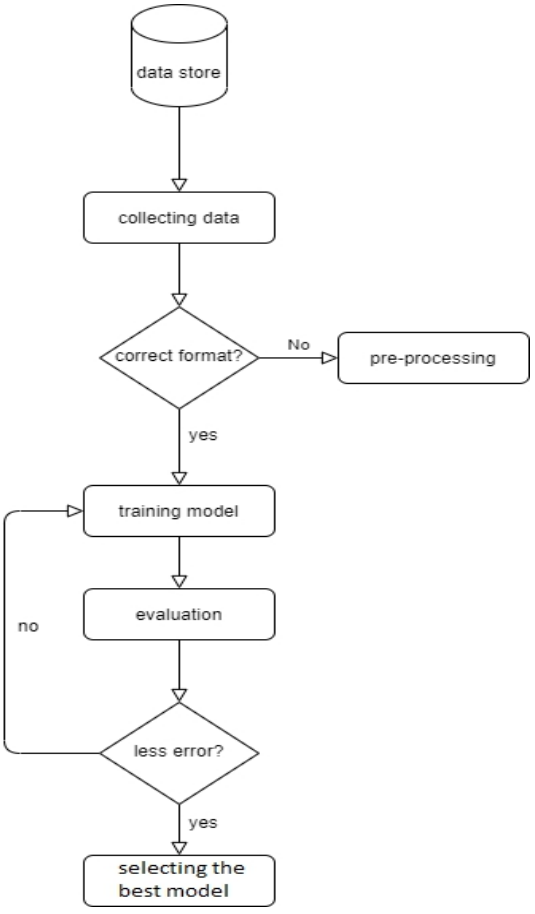


Figure 4.1: Proposed System Flow Chart

4.2 PROPOSED SYSTEM DATA FLOW DIAGRAM

► **DFD Symbols**

There are four basic symbols that are used to represent a data flow diagram.

► **Process**

This paper focused on credit card application which is used to detect the fraudulent credit card activities on credit transaction. In this peculiar type, the pattern of current fraudulent usage of the credit card has been analyzed with the previous transactions, by using the intelligent agent in data mining algorithm. Fig. 2 shows the data flow diagram of the new system model.

The system has three data mining engines: customer/bank database, credit card transaction database and fraud detection database. The customer/bank database has the following: opening of account operation, withdrawal and deposit transaction and credit card transaction. Fraud techniques database will give details of attack attempts on customer's credit card.

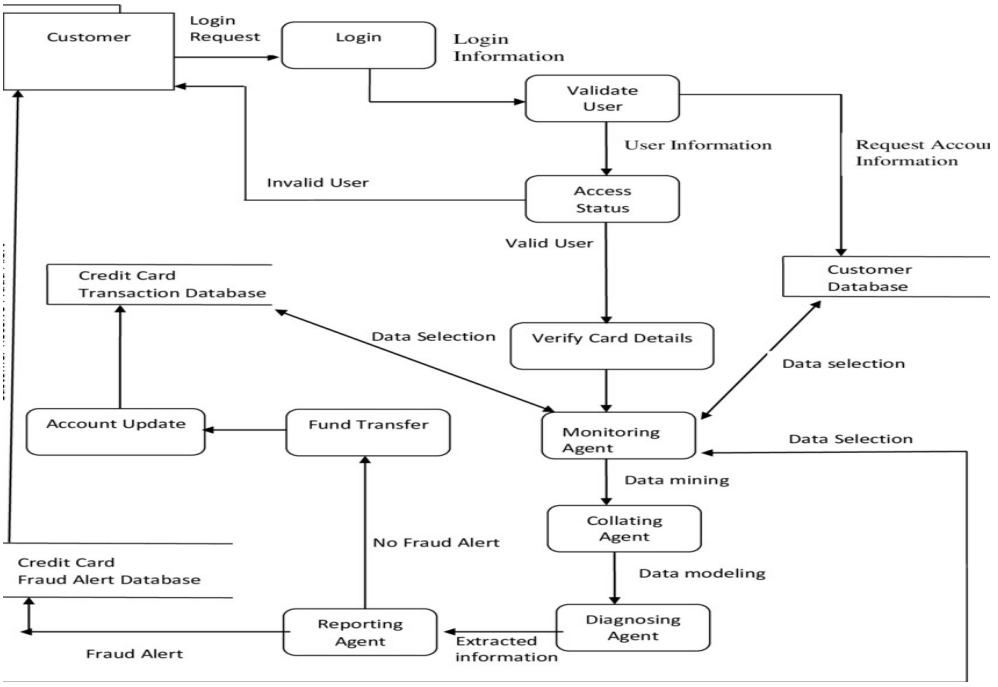


Figure 4.1 : Process diagram

► **Data Flow**

A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such the Customer ID or itcan represent a set of data element (or a data structure).

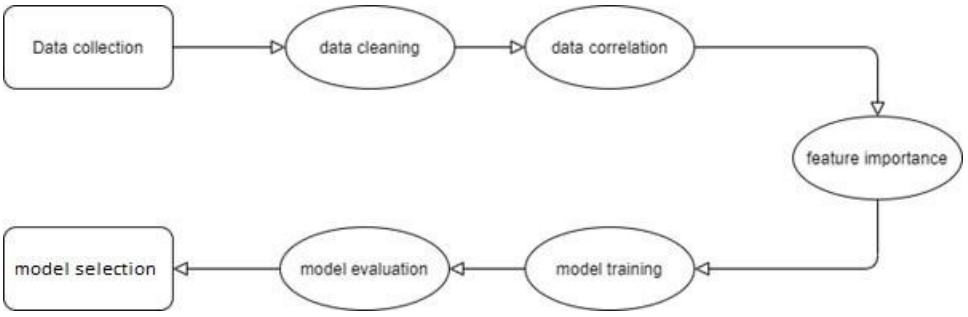


Figure 4.2 : proposed system data flow diagram

4.1.1 Proposed System Class Diagram

Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system.

Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system.

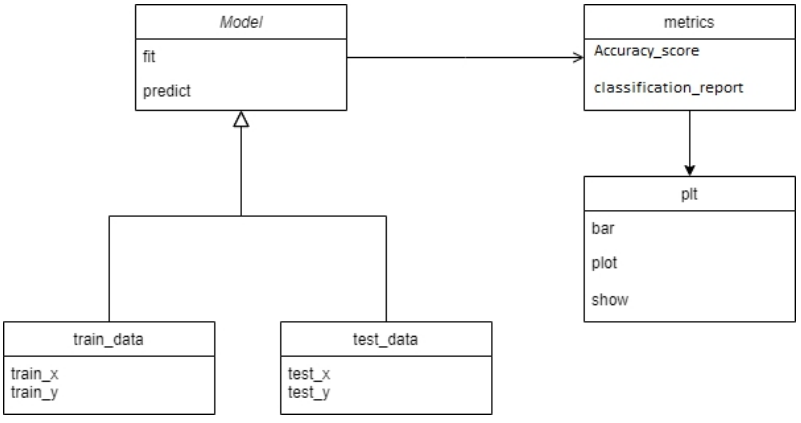


Figure 4.3 Proposed System Class Diagram

CHAPTER 5
SYSTEM REQUIREMENTS

5.1 SOFTWARE REQUIREMENTS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed. Inthis system study the following software requirements used.

- OS: Windows XP + /Linux/ MacOS
- Python 3
- Pandas

5.2 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. In this system study the following hardware requirements used.

Component	Minimum requirement
Processor	64-bit, i3 processor, 2.5 GHz minimum per core
Hard disk	120 GB

CHAPTER-6

CODING

```
# # importing libraries....


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
# # data_analysis.....
# In[2]:


data=pd.read_csv('creditcard.csv')
data


# # dataprepocessing.....


# In[3]:


data.isna().sum()
# In[4]:
data['Class'].value_counts()
# In[5]:
sns.countplot(x = "Class", data = data)
plt.title("Frequency")
plt.show()
# In[6]:
data.hist(figsize=(16,10))
plt.tight_layout()
plt.show()
```

```
# In[12]:


plt.figure(figsize = (60,20))
sns.heatmap(data.corr(),square=True, annot=True)
plt.show()

# # data_training....


# # Supervised learning
# ## classical machine learning
# In[8]:
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(data.drop(columns=['Class']),data['Class'],t
est_size=0.2, train_size=0.3)

# In[9]:


from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=10)
clf.fit(x_train, y_train)

# # evaluating.....


# In[15]:


from sklearn.metrics import accuracy_score
predict = clf.predict(x_test)
print('accuracy',round(accuracy_score(y_test,predict),4),'%,random_forest')
predict
# Adding Artificial neural Network
```

```
import tensorflow as tf
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(30, 1)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(2, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

# # conclusion.
```

RandomForestClassifier is the best classifier for this data..with accuracy 99.95% and

confusion matrix

```
 [[56868     1]
 [   26    67]]
 random_forest confusion matrix.
```

CHAPTER 7

TESTING

7.1 PROPOSED SYSTEM TESTING

7.1.1 BLACK BOX TESTING

Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input, and observes the output generated by the system under test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.

Black box testing is a powerful testing technique because it exercises a system end-to-end. Just like end-users “don’t care” how a system is coded or architected, and expect to receive an appropriate response to their requests, a tester can simulate user activity and see if the system delivers on its promises. Along the way, a black box test evaluates all relevant subsystems, including UI/UX, web server or application server, database, dependencies, and integrated systems.

An example of a security technology that performs black box testing is Dynamic Application Security Testing (DAST), which tests products in staging or production and provides feedback on compliance and security issues.

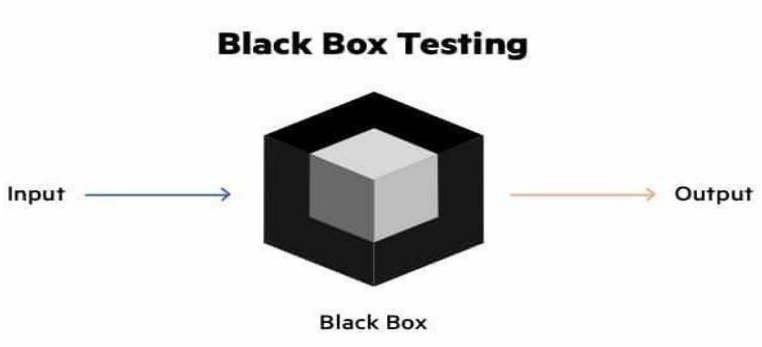


Figure 7.1 : Black Box Testing

Black Box Testing Pros and Cons

	Pros	Cons
1.	Testers do not require technical knowledge,	Difficult to automate
2.	Testers do not need to learn implementation details of the system	Requires prioritization, typically infeasible to test all user paths
3.	Tests can be executed by crowdsourced or outsourced testers	Difficult to calculate test coverage
4.	Low chance of false positives	If a test fails, it can be difficult to understand the root cause of the issue
5.	tests have lower complexity, since they simply model common user behavior	Tests may be conducted at low scale or on a non-production-like environment

7.1.2 Types Of Black Box Testing

Black box testing can be applied to three main types of tests: functional, non-functional, and regression testing

i) Functional Testing

Black box testing can test specific functions or features of the software under test. For example, checking that it is possible to log in using correct user credentials, and not possible to log in using wrong credentials

Functional testing can focus on the most critical aspects of the software (smoke testing/sanity testing), on integration between key components (integration testing), or on the system as a whole (system testing).

ii) Non-Functional Testing

Black box testing can check additional aspects of the software, beyond features and functionality. A non-functional test does not check "if" the software can perform a specific action but "how" it performs that action.

Black box tests can uncover if software is:

Usable and easy to understand for its users

- Performant under expected or peak loads
- Compatible with relevant devices, screen sizes, browsers or operating systems

7.1.3 Black Box Testing Techniques

i) Equivalence Partitioning:

Testers can divide possible inputs into groups or "partitions", and test only one example input from each group. For example, if a system requires a user's birth date and provides the same response for all users under the age of 18, and a different response for

users over 18, it is sufficient for testers to check one birth date in the “under 18” group and one date in the “over 18” group.

Boundary Value Analysis :

Testers can identify that a system has a special response around a specific boundary value. For example, a specific field may accept only values between 0 and 99. Testers can focus on the boundary values (-1, 0, 99 and 100), to see if the system is accepting and rejecting inputs correctly.

ii) Decision Table Testing

Many systems provide outputs based on a set of conditions. Testers can then identify “rules” which are a combination of conditions, identify the outcome of each rule, and design a test case for each rule.

For example, a health insurance company may provide different premium based on the age of the insured person (under 40 or over 40) and whether they are a smoker or not. This generates a decision table with four rules and up to four outcomes—below is an example with three possible outcomes.

7.2 WHITE BOX TESTING

White-box testing's basic procedures require the tester to have an in-depth knowledge of the source code being tested. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analyzed for test cases to be created. The following are the three basic steps that white- box testing takes in order to create test cases:

1. Input involves different types of requirements, functional specifications, detailed designing of documents, proper source code and security specifications This is the preparation stage of white-box testing to lay out all of the basic information.
2. Processing involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
3. Output involves preparing final report that encompasses all of the above preparations and result.

CHAPTER 8

OUTPUT SCREEN

8.1 Proposed System Output Screens

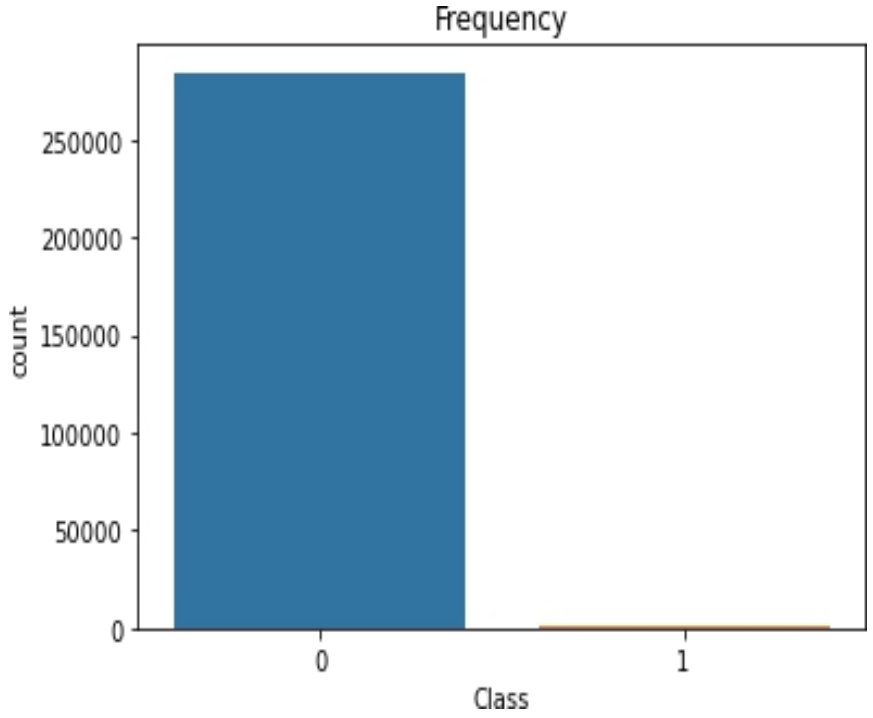


Figure 8.1 Count of Fraudulent VS Non-Fraudulent Transactions

intermediate results

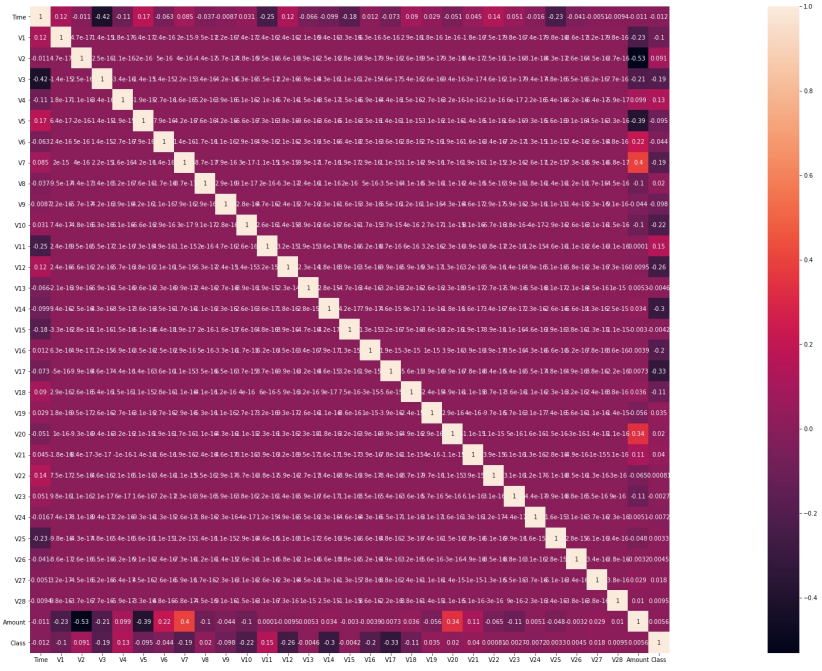


Figure 8.2 correlation Histogram

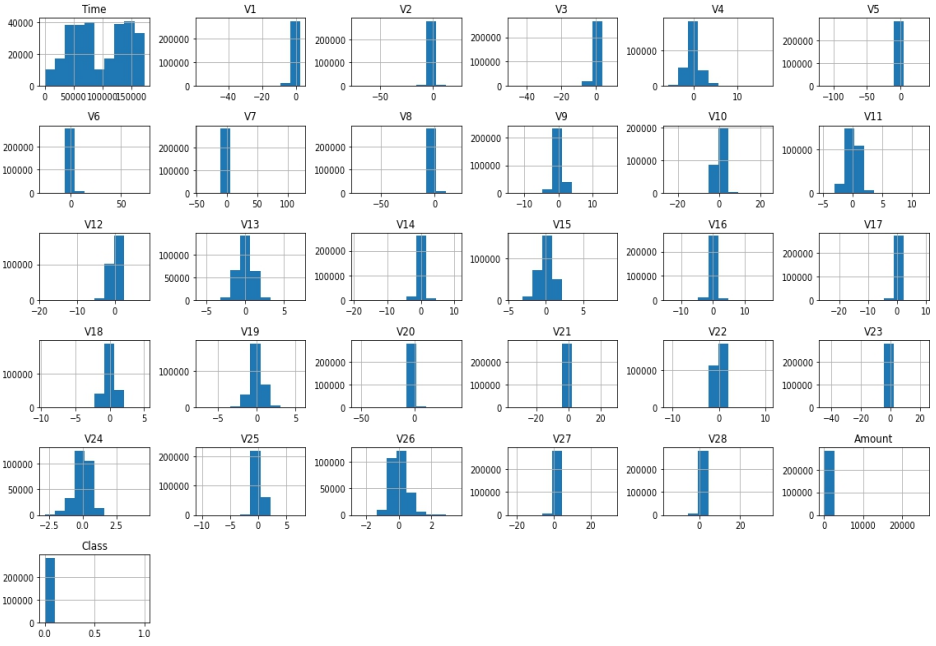


Figure-8.3 Histograms of Data Columns

CHAPTER 9
CONCLUSION

Credit card fraud is without a doubt an act of criminal dishonesty. This article has listed out the most common methods of fraud along with their detection methods and reviewed recent findings in this field. This paper has also explained in detail, how machine learning can be applied to get better results in fraud detection along with the algorithm, pseudocode, explanation its implementation and experimentation results.
While the algorithm does reach over 99.6% accuracy, its precision remains only at 28% when a tenth of the data set is taken into consideration. However, when the entire dataset is fed into the algorithm, the precision rises to 33%. This high percentage of accuracy is to be expected due to the huge imbalance between the number of valid and number of genuine transactions.

Since the entire dataset consists of only two days' transaction records, its only a fraction of data that can be made available if this project were to be used on a commercial scale. Being based on machine learning algorithms, the program will only increase its efficiency over time as more data is put into

CHAPTER 10

FURTHER ENCHANTMENTS

While we couldn't reach out goal of 100% accuracy in fraud detection, we did end up creating a system that can, with enough time and data, get very close to that goal. As with any such project, there is some room for improvement here.

The very nature of this project allows for multiple algorithms to be integrated together as modules and their results can be combined to increase the accuracy of the final result.

This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others. Once that condition is satisfied, the modules are easy to add as done in the code. This provides a great degree of modularity and versatility to the project.

More room for improvement can be found in the dataset. As demonstrated before, the precision of the algorithms increases when the size of dataset is increased. Hence, more data will surely make the model more accurate in detecting frauds and reduce the number of false positives. However, this requires official support from the banks themselves.

CHAPTER 11

REFERENCES

- "Credit Card Fraud Detection Based on Transaction Behaviour -by John Richard D. Kho, Larry A. Vea" published by Proc. of the 2017 IEEE Region 10 Conference (TENCON), Malaysia, November 5-8, 2017

- CLIFTON PHUA1, VINCENT LEE1, KATE SMITH1 & ROSS GAYLER2 " A Comprehensive Survey of Data Mining-based Fraud Detection Research" published by School of Business Systems, Faculty of Information Technology, Monash University,
- Wellington Road, Clayton, Victoria 3800, AustraiIa

- "Survey Paper on Credit Card Fraud Detection by Suman" , Research Scholar, GJUS&T Hisar HCE, Sonepat published by International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 3, March 2014

- "Research on Credit Card Fraud Detection Model Based on Distance Sum – by Wen-Fang YU and Na Wang" published by 2009 International Joint Conference on Artificial Intelligence

- "Credit Card Fraud Detection through Parenclitic Network AnalysisBy MassimilianoZanin, Miguel Romance, Regino Criado, and SantiagoMoral" published by Hindawi Complexity Volume 2018, Article ID 5764370, 9 pages
- "Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy" published by IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 29, NO. 8, AUGUST 2018

- "Credit Card Fraud Detection-by Ishu Trivedi, Monika, Mrigya, Mridushi" published by International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 1, January 2016

- David J.Wetson,David J.Hand,M Adams,Whitrow and Piotr Jusczak "Plastic Card Fraud Detection using Peer Group Analysis" Springer, Issue 2008.