ANNADA COLLEGE HAZARIBAGH, JHARKHAND
V.B.U, HAZARIBAGH-825301

CREDIT CARD FRAUD DETECTION WITH ML

AN PROJECT SUBMITTED
IN PARTIAL – FULFILLMENT OF THE REQUIREMENT
FOR THE AWARD OF THE DEGREE
OF

BACHELOR OF COMPUTER APPLICATION

(For Degree in BCA)

BY

SAURAV KUMAR

(UNIVERSITY ROLL NO-210219016863)

Under the Guidance

Of

KANCHAN RAJU

DEPARTMENT OF BCA

ANNADA COLLEGE, V.B.U, HAZARIBAGH-825301

(2020-2023)

BONAFIDE CERTIFICATE

This is to certify that project report entitled "**Credit Card Fraud Detection With ML"** submitted by **SAURAV KUMAR (UNIVERSITY ROLLNO-210219016863),**In partial fulfillment of the requirements for the award of the degree **Bachelor of Computer Application**. In **Department of BACHELOR OF Computer APPLICATION (BCA). ANNADA COLLEGE, V.B.U, HAZARIBAGH** is a bonafide record of the work carried out under my guidance and supervision **Sai Coding Solution, Devrani Complex, Circular Road, Ranchi.**

SIGNATURE

KANCHAN RAJU

Senior Trainer

Sai Coding Solution Pvt. Ltd.

Devrani Complex, Circular Road.

Lalpur, Ranchi

Certificate of Department of Bachelor of Computer Application (BCA), ANNADA COLLEGE, V.B.U, HAZARIBAGH

This is to certify that project report entitled "**Credit Card Fraud Detection With ML"** submitted by **SAURAV KUMAR (UNIVERSITY ROLLNO-210219016863), In** partial fulfillment of the requirements for the award of the degree **Bachelor of Computer Application (BCA)**. In **Department of Bachelor of Computer Application (BCA). ANNADA COLLEGE, V.B.U, HAZARIBAGH.**

SAURAV KUMAR has worked under my supervision and guidance and no part of this report has been submitted for the award of any other degree. Diploma, fellowship or other similar titles or prizes and that the work has not been published in any journal or Magazine

Project In – charge **Course Co-Ordinator.**

Department Of Bachelor of Computer Application

ANNADA College, V.B.U, HAZARIBAGH

CERTIFICATE FOR PROJECT

This is to certify that this is a bona fide record of the project work entitled **“Credit Card Fraud Detection With ML”** done satisfactory at “ANNADA College, V.B.U, HAZARIBAGH**”** by **SAURAV KUMAR ,** in partial fulfillment of BCA (UG) Examination.

This report or similar report on the topic has not been submitted for any other examination and doesn’t form part of any other course undergone by the candidate.

INTERNAL GUIDE **EXTERNAL GUIDE**

UNDER THE GUIDANCE OF

EXAMINER'S CERTIFICATION

This is to certify that project report entitled "**Credit Card Fraud Detection With ML"** submitted by **SAURAV KUMAR (ROLLNO-210219016863),** In partial fulfillment of the requirements for the award of the degree Bachelor **of Computer Application (BCA)**. In **Department of Bachelor of Computer Application (BCA). ANNADA COLLEGE, V.B.U, HAZARIBAGH.**

This project Is Approved and Acceptable in Quality and Form.

SIGNATURE	**SIGNATURE**
NAME: -	**NAME: -**
(External Examiner)	**(External Examiner)**

MY MAJOR PROJECT CERTIFICATE

DECLARATION

By **SAURAV KUMAR (210219016863)** hereby declare that the work, which is being presented in the dissertation, entitled **“CREDIT CARD FRAUD DETECTION WITH ML**”, in partial fulfillment of requirement for the award of the degree of **“BACHELOR OF COMPUTER APPLICATION”** submitted in **AANADA COLLEGE, V.B.U, HAZARIBAGH** is an authentic record of our work carried out under the guidance of **KANCHAN RAJU.**

I have not submitted the matter embodied in this dissertation for the award of any other degree.

ACKNOWLEDGEMENT

This project work has been an intellectually invigorating experience for me. I am sure that the knowledge and experience gathered during the course of this work will make me stand in good stead in feature.

With immense pleasure and due respect, I express my sincere gratitude to In-charge, **ANNADA COLLEGE, V.B.U, HAZARIBAGH.** For all his support and co-operation in successfully completing this thesis work by providing excellent facilities.

I am highly grateful to the Department Co-Ordinator **SANJEEV CHATERJEE** his ever-helping attitude and encouraging us to excel in studies, besides, he has been a source of inspiration during my entire period of BCA.

I would like to take this opportunity to extend my sincere gratitude and thanks to my pioneer **KANCHAN RAJU,** firstly for coming up with such an innovative thesis idea. He has not only made us to work but guided us to orient toward research. It has been real pleasure working under his guidance and it is chiefly his encouragement and motivation that has made this thesis a reality.

Last, but not the least I am heartly thankful to my parent for showering their blessings forever during my entire life and also to my family members and friends for providing me a great

Table of Contents:

Abstract

It is vital that credit card companies are able to identify fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Such problems can be tackled with Data Science and its importance, along with Machine Learning, cannot be overstated. This project intends to illustrate the modelling of a data set using machine learning with Credit Card Fraud Detection. The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the data of the ones that turned out to be fraud. This model is then used to recognize whether a new transaction is fraudulent or not. Our objective here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications. Credit Card Fraud Detection is a typical sample of classification. In this process, we have focused on analysing and pre-processing data sets as well as the deployment of multiple anomaly detection algorithms such as Local Outlier Factor and Isolation Forest algorithm on the PCA transformed Credit Card Transaction data.

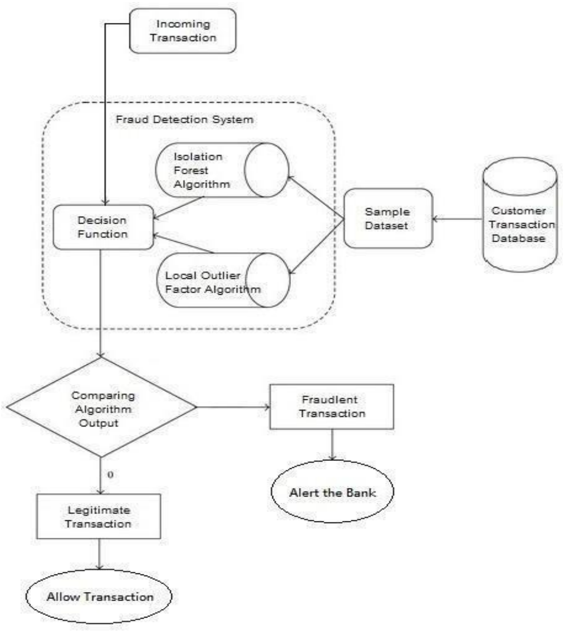
Keywords— Credit card fraud, applications of machine learning, data science, isolation forest algorithm, local outlier factor, automated fraud detection.

I. INTRODUCTION

'Fraud' in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account. Necessary prevention measures can be taken to stop this abuse and the behavior of such fraudulent practices can be studied to minimize it and protect against similar occurrences in the future. In other words, Credit Card Fraud can be defined as a case where a person uses someone else’s credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used.

Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behavior, which consist of fraud, intrusion, and defaulting.

This is a very relevant problem that demands the attention of communities such as machine learning and data science where the solution to this problem can be automated.
This problem is particularly challenging from the perspective of learning, as it is characterized by various factors such as class imbalance. The number of valid transactions far outnumber fraudulent ones. Also, the transaction patterns often change their statistical properties over the course of time.
These are not the only challenges in the implementation of a real-world fraud detection system, however. In real world examples, the massive stream of payment requests is quickly scanned by automatic tools that determine which transactions to authorize.
Machine learning algorithms are employed to analyse all the authorized transactions and report the suspicious ones. These reports are investigated by professionals who contact the cardholders to confirm if the transaction was genuine or fraudulent.
The investigators provide a feedback to the automated system which is used to train and update the algorithm to eventually improve the fraud-detection performance over time.



Fraud detection methods are continuously developed to defend criminals in adapting to their fraudulent strategies. These frauds are classified as:

- Credit Card Frauds: Online and Offline
- Card Theft
- Account Bankruptcy
- Device Intrusion
- Application Fraud
- Counterfeit Card
- Telecommunication Fraud

Some of the currently used approaches to detection of such fraud are:

- Artificial Neural Network
- Fuzzy Logic
- Genetic Algorithm
- Logistic Regression
- Decision tree
- Support Vector Machines
- Bayesian Networks
- Hidden Markov Model
- K-Nearest Neighbour

II. LITERATURE REVIEW

Fraud act as the unlawful or criminal deception intended to result in financial or personal benefit. It is a deliberate act that is against the law, rule or policy with an aim to attain unauthorized financial benefit.

Numerous literatures pertaining to anomaly or fraud detection in this domain have been published already and are available for public usage. A comprehensive survey conducted by Clifton Phua and his associates have revealed that techniques employed in this domain include data mining applications, automated fraud detection, adversarial detection. In another paper, Suman, Research Scholar, GJUS&T at Hisar HCE presented techniques like Supervised and Unsupervised Learning for credit card fraud detection. Even though these methods and algorithms fetched an unexpected success in some areas, they failed to provide a permanent and consistent solution to fraud detection.

A similar research domain was presented by Wen-Fang YU and Na Wang where they used Outlier mining, Outlier detection mining and Distance sum algorithms to accurately predict fraudulent transaction in an emulation experiment of credit card transaction data set of one certain commercial bank. Outlier mining is a field of data mining which is basically used in monetary and internet fields. It deals with detecting objects that are detached from the main system i.e. the transactions that aren’t genuine. They have taken attributes of customer’s behaviour and based on the value of those attributes they’ve calculated that distance between the observed value of that attribute and its predetermined value.

Unconventional techniques such as hybrid data mining/complex network classification algorithm is able to perceive illegal instances in an actual card transaction data set, based on network reconstruction algorithm that allows creating representations of the deviation of one instance from a reference group have proved efficient typically on medium sized online transaction.

There have also been efforts to progress from a completely new aspect. Attempts have been made to improve the alert- feedback interaction in case of fraudulent transaction.

In case of fraudulent transaction, the authorised system would be alerted and a feedback would be sent to deny the ongoing transaction.

Artificial Genetic Algorithm, one of the approaches that shed new light in this domain, countered fraud from a different direction.

It proved accurate in finding out the fraudulent transactions and minimizing the number of false alerts. Even though, it was accompanied by classification problem with variable misclassification cost.

☐ <u>**What is Python?**</u>

Python is an interpreted high-level programing language for general purpose programing.

<u>History of Python</u>

Python is by Guido van Rossum and first released in 1991. ABC a general-purpose programing language of that time, was influence behind Python.

<u>what about the name “Python”?</u>

According to **Guidovan Rossum.** “I chose Python as a working title for the project, being in a slightly irreverent mood (And a big fan of Monty Python's Flying Circus)."

☐ **Python philosophy**

☐ The language's core philosophy is summarized in the document *The* Zen of Python

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts
- There should be one—and preferably only one—obvious way to do it.

| Top 10 Programing Language

☐ Who uses Python?

☐ Features of Python

☐ Features of Python

☐ Features of Python

☐ Use of Python

☐ For Web Development

☐ In AI

☐ Game Development

☐ Software Testing

☐ In Big Data

☐ Object Automation

☐ In Data Science

☐ Installation of Python and IDE (Spyder-Anaconda)

☐ Installation

☐ Python 2.0 vs 3.0

☐ Install Anaconda

☐ What is anaconda

Anaconda is a free and open
source distribution of the Python and R programing
languages for data science and
machine learning related

applications (large-scale data processing, predictive

analytics, scientific computing), that aims to simplify package management and deployment. Package versions are managed by the package management system *conda.* The Anaconda distribution is used by over 6 million users, and it includes more than **250** popular data science packages suitable for Windows, Linux, and MacOS

☐ Hello world example

There are two methods to run python commands

1.Through Python Shell

☐>>> print ("Hello World!")

☐Hello World!

☐ >>>

2. Using Python Script

Print ("My first simple Python script!") and save it as **my_first_python.py**.

Running Python Script

☐ Using ide

☐ On command prompt

Important Things to Remember

☐To represent a statement in Python, newline (enter) is used. The use of **semicolon** at the end of the statement **is optional** in fact, it's **recommended to omit semicolon** at the end of the statement in Python.

☐ Indentation

☐Python programs get structured through indentation, i.e., code blocks are defined by their indentation.

☐Instead of curly braces {}, indentations are used to represent

☐ a block.

Generally, **four whitespaces** are used for indentation and is **preferred** over **tabs.**

- ◻ Variables
- ◻ As the name implies, a variable is something which can **change.** A variable is a way of referring to a **memory location** used by a computer program.
- ◻ A variable can be seen as a container to store certain values. While the program is running, variables are **accessed** and sometimes **changed**, i.e., a new value will **be assigned** to a variable.

There is no declaration of variables required in Python. If there is need of a variable, you think of a **name** and **start using** it as a variable.

```
i = 42

>>>i = i + 1

 >>>print(i)

 >>> 43
```

- ☐ Variables

- ◻ Another remarkable aspect of Python: Not only the **value of a variable** may change during program execution but the **type as well.**
- ◻ >>> x = 42
- ◻ >>> print(x) 42

- ◻ >>> x = "Now x references a string"

- ◻ >>> print(x) Now x references a string

<u>Valid Variable Names</u>

The naming of variables follows the more general concept of an **identifie**r. A Python **identifier** is a name used to identify a **variable, function, class, module or other object.**

A variable name and an **identifier** can consist of the uppercase letters **"A" through "Z"**, the lowercase **letters "a" through "z"**, the **underscore _** and, **excep**t for **the first character**, the **digits 0 through 9.**

--Python variables are case-sensitive--

☐ Key Words in Python

☐ No identifier can have the same name as one of the Python **keywords.**

☐ There are 33 keywords in Python 3.3.

☐ Python Statements

☐ In Python, **end of a statement** is marked by a **newline character.**

☐ **<u>Multi-line statement</u>**

statement extend over **multiple lines** with the line continuation character **(\\)**. For example:

```
□a = 1 + 2 + 3 + \

     4 + 5 + 6 + \

    7 + 8 + 9
```

This is explicit line continuation. In Python**, line continuation** is implied inside parentheses (), brackets [] and braces {}**.** For instance, we can implement the above multi-line statement as

```
a = (1 + 2 + 3 +
    4 + 5 + 6 +

     7 + 8 + 9)
```

☐ Python Comments

- In Python, we use the **hash (#) symbol** to start writing a comment.
- #This is a comment
- #Print out Hello
- print('Hello')

<u>Multi-line comments</u>

If we have comments that extend **multiple lines** use triple quotes, **either ''' or ""“**

"""This is also a perfect
Example of multi-line
Comments"""

- **Python Numbers**
- Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined **as into, float and complex** class in Python.
- We can use the **type ()** function to know which class a variable or a value belongs to and thi**s instance ()** function to check if an object belongs to a particular class.
- a = 5
- print (a, "is of type", **type(a)**)
- a = 2.0
- print (a, "is of type", **type(a**))
- **a = 1+2j**
- print (a, "is complex number?", **type(1+2j)**)
- Type Conversion

- ◻ Using Type Conversion, we convert the **data type** of an object to **required data type.** We use the predefined functions like **int (), float (), str (),** etc. to perform type conversion.

num_int = 123

num_str = "456“

print ("Data type of num_int:”, type(num_int))

print ("Data type of num_str before Type Casting:", **type(num_str))**
num_str = int(num_str)
print ("Data type of num_str after Type Casting:", type(num_str))
num_sum = num_int + num_str
print ("Sum of num_int and num_str:", num_sum)
print ("Data type of the sum:", type(num_sum))

☐ Python Operator

☐ Python input output

☐ Python Output Using print () function

☐ We use the print () function **to output data** to the standard output device (screen).
☐ We can also output data to a file, but this will be discussed later. An example use is given below.
☐ The actual syntax of the print () function is

Print (***objects**, **sep=' ', end='\n', file=sys.stdout**, flush=False)

☐ Here, objects are the value(s) to be printed.

☐ The sep separator is used between the values. It defaults into a space character.
☐ After all values are printed, **end** is printed. It defaults into a new line.
☐ The **file** is the object where the values are printed and its default value is sys.stdout (screen). Here is an example to illustrate this.

☐ Python Output formatting

☐ Sometimes we would like to format our output to make it look attractive. This can be done by using the **str.format()** method. This method is visible to any string object.

- >>> x = 5;
- y = 10
- >>> print('The value of x is {} and y is {}'.format(x,y))
- #The value of x is 5 and y is 10
- Here the curly braces {} are used as placeholders.
- Python Input
- In Python, we have the input() function to take the input from the user. The syntax for input() is
- >>> num = input('Enter a number: ')
- Enter a number: 10
- >>> num '10‘

the **entered** value 10 **is a string**, not a number. To convert this into a number we can use **int()** or **float()** functions.

- Python Import
- A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension **.py**.
- Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the **import keyword** to do this.
- For example, we can import the **math module** by typing in import math.
- **import math**
- **print(math.pi)**
- Python if....else Statement
- **We use if....else for decision**

- **making**
- **<u>Syntax of if...else</u>**
- if test expression:
    - Body of if

- else:
    - Body of else

Python Nested if statements

- We can have an if...elif...else statement inside **another** if...elif...else statement. This is called nesting in computer programing.
- Python List
- **Python List**
- List is an **ordered sequence** of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.
- Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].
- >>> a = [1, 2.2, 'python']
- We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.
- a = [5,10,15,20,25,30,35,40]
- print("a[2] = ", a[2])
- # a[2] = 15
- # a[0:3] = [5, 10, 15]
- print("a[0:3] = ", a[0:3])
- # a[0:3] = [5, 10, 15]
- print("a[5:] = ", a[5:])

□# a[5:] = [30, 35, 40]

Lists are mutable, meaning, value of elements of a list can be **altered.**

☐ Python Tuple

- ☐Tuple is an ordered sequence of items same as list. The only difference is that tuples are **immutable.** Tuples once created cannot be modified.
- ☐Tuples are used to write -protect data and are usually faster than list as it cannot change dynamically.
- ☐It is defined within parentheses () where items are separated by commas.
- ☐>>> t = (5,'program', 1+3j)

We can use the slicing operator [] to extract items but we **cannot change its value.**

t = (5,'program', 1+3j)

# t[1] = 'program'

print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))

print("t[0:3] = ",

t[0:3]) # Generates

error

# Tuples are immutable

t[0] = 10

☐ Python for loop

- ☐The for loop in Python is used to **iterate over a sequence** (list, tuple, string) or other iterable objects. Iterating over a sequence is called **traversal.**

Syntax of for
Loop**for val in**
sequence:

Body of for

Here, val is the variable that takes the value of
the item inside the sequence on each iteration.
Loop continues until we reach the last item in

the sequence. The body of for loop is separated from the rest of the code using indentation.

- ☐ The range() function

- ☐ We can generate a **sequence of numbers** using **range()** function. range(10) will generate numbers from **0 to 9** (10 numbers).
- ☐ We can also define the **start, stop and step size** as range(start,stop,step size). step size defaults to 1 if not provided.
- ☐ It does not generate sequence of numbers.

To force this function to output all the items, we can use the function list().

- ☐ # Output: range(0, 10)
- ☐ print(range(10))
- ☐ # Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- ☐ print(list(range(10)))
- ☐ # Output: [2, 3, 4, 5, 6, 7]
- ☐ print(list(range(2, 8)))
- ☐ # Output: [2, 5, 8, 11, 14, 17]
- ☐ print(list(range(2, 20, 3)))

- ☐ For loop with else

- ☐ A for loop can have an optional **else block** as well. The else part is executed if the items in the sequence used in for loop **exhausts.**

- ☐ **break statement** can be used **to stop** a for loop. In such case, the **else** part is **ignored.**
- ☐ Hence, a for loop's **else part** runs if no break occurs.

☐ Python while loop

The while loop in Python is used to
iterate over a block of code as long as the
test expression (condition) is true.
We generally use this loop when we don't
know beforehand, the number of times to
iterate.

<u>Syntax of while Loop in Python</u>

```
while test_expression:
     Body of while
```

In while loop**, test expression** is checked first.
The body of the loop is entered only if the
test_expressionevaluates to **True.**
After one iteration, the test expression is **checked again.** This
process continues until the test_expression evaluates **to False.**

☐ While loop example

```
# Program to add natural

# To take input from the user,
 n = int(input("Enter n: "))
n = 10

# initialize sum and counter
sum = 0
i = 1

while i<= n:
  sum = sum + ii
  = i+1
 # update counter
# print the sum
print("The sum is", sum)
```

☐ While loop with else

- Same as that of for loop, we can have an optional **else block** with while loop as well.
- The else part is executed if the condition in the while loop evaluates **to False**.
- The while loop can be terminated with a break statement. In such case, the else part is ignored.

```
# the use of else statement
# with the while loop
counter = 0
while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:    print("Inside else")
```

- Python Break and continue
- The break statement **terminates** the
- loop containing it. Control of the
- program flows to the statement
- immediately after the body of the loop.
- If break statement is inside a nested
- loop (loop inside another loop), break
- will terminate the innermost loop.
- Python continue statement

- The continue statement is used **to skip**
- the rest of the code inside a loop for the
- **current iteration only.** Loop **does not**
- **terminate** but continues on with the
- **next iteration.**

☐ Python Function

☐ In Python, function is a group of **related statements** that perform a **specific task.**

☐ Functions help **break our program** into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

☐ Furthermore, it avoids repetition and makes code reusable.

Syntax of Function

```
def function_name(parameters):

 """docstring"""
statement(s)
```

☐ Keyword **def** marks the start of function header.

☐ Parameters (arguments) through which we pass values to a function. They are optional.

☐ A colon (:) to mark the end of function header.

☐ Optional **documentation string (docstring)** to describe what the function does.

☐ One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).

☐ An optional **return** statement to return a value from the function.

☐ Python function

How to call a function in python?

☐ Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the **function name** with appropriate parameters.

```
greet('Paul')
```

Hello,Paul. Good morning!

<u>Docstring</u>

It is used to explain in brief, what a function does.
This string is available to us as **doc_____**attribute of the function.

```
>>>print(greet.doc)
```

- ☐ Python Function

- ◻The return statement is used **to exit** a function and go back to the place from where it was called.

<u>Python Default Arguments</u>

Function arguments can have default values in Python.

We can provide a default value to an argument by using the assignment operator (=).

```
# default argument for second parameter
def add(num1, num2=1):
return num1 + num2
sum1 = add(100, 200)
sum2 = add(8)
# used default argument for second parameter
sum3 = add(100)
# used default argument for second parameter
print(sum1)
print(sum2)
print(sum3)
```

- ☐ Scope of Python Local Veriable
- ◻Parameters and variables defined inside a function is **not visible** from outside. Hence, they have a **local scope.**

◻The lifetime of variables inside a function is as long as the function executes.
◻They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

```
def f():

     s = "Only in spring, but London is great as well!“
      print(s)
s = "I am looking for a course in Paris!“
 print(s)
 f() print(s)
```

☐ Function Argument

Python Default Arguments

Function arguments can have default values in Python.

We can provide a default value to an argument by using the assignment operator (=). Here is an example.

```
def greet(name, msg = "Good morning!"):
def calArea(s1 ,s2=1):
```

Python Keyword Arguments

When we call a function with some values, these values get assigned to the arguments according to their position.

```
# 2 keyword arguments

 greet(name = "Bruce",msg = "How do you do?")
 # 2 keyword arguments (out of order)
 greet(msg = "How do you do?",name = "Bruce")
 # 1 positional, 1 keyword argument
 greet("Bruce",msg = "How do you do?")
```

☐ Function Argument

☐ **Python Arbitrary Arguments**

☐ Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

☐ In the function definition we use an asterisk (*) before the parameter name to denote this kind of argument. Here is an example.

☐ def greet(*names):

☐ """This function greets all   the person in the names tuple."""   # names is a tuple with arguments

☐   for name in names:

☐   print("Hello",name)

☐ greet("Monica","Luke","Steve","John")

☐ Function Classification

There are following types of the functions
based on their parameters and return values:

- Function with no argument and no return value
- Function with no argument but return value
- Function with argument and no return value
- Function with arguments and return value

1)Function with no argument and no return value

def sum():
 a = int(input("Enter A: "))

```
 b = int(input("Enter B: "))
 c=a+b
 print("Sum :", c)
```

 ☐ Function Classification

2) Function with no argument but return value

```
def sum():
 a = int(input("Enter A: "))

 b = int(input("Enter B: "))
 c=a+b
 return c
```

3) Function with argument and no return value

```
def sum(a,b):
c=a+bprint("Su
 m :", c)
```

4) Function with arguments and return value

```
def sum(a,b):
 c=a+br
 eturn c
```

☐ Global and Local

Veriables**Global Variables**

In Python, a variable declared **outside of the function** or in global scope is known as **global variable.** This means, global variable can be accessed **inside or outside** of the function.

Local Variables

A variable declared **inside the function's body** or in the local scope is known as **local variable.**

```
x = 5

def myfun():
    x = 10

    print("local x:", x)
myfun()
print("global x:", x)
```

- ☐ Non Local Variable
- ☐ **Nonlocal Variables**
- ☐ Nonlocal variable are used in nested function whose local scope is not defined. This means, the variable can be neither in the local nor the global scope.

```
☐def outer():

☐     x = "local“

☐         def inner():

☐         nonlocal x

☐         x = "nonlocal“

☐         print("inner:", x)

☐         inner()

☐     print("outer:", x)

☐outer()
```

- ☐ global keyword

We use global keyword to read and write a global variable inside a function.

```
☐c = 1 # global variable

☐  def add():
```

☐ c = c + 2 # increment c by 2

☐ print(c)

☐ add()

After using global keyword
c = 0 # global variabledef

```
 add():

    global c

   c = c + 2 # increment by 2
    print("Inside add():", c)
add()
```

☐ Lambda Function

In Python, anonymous function is a function that is defined

without a name.

While normal functions are defined using the def keyword, in Python anonymous functions are defined using
the lambda keyword.

Hence, anonymous functions are also called lambda functions.
Syntax of Lambda Function in python

lambda arguments: expression

Example

```
# Program to show the use of lambda functions

double = lambda x: x * 2
# Output: 10
print(double(5))
```

☐ Python Lambda Function
- ☐ **With the help of lambda function, we can create one line function definition.**
- ☐ **Note:** Function must have return type and parameter
- ☐ **xample:**
- ☐ Convert program from Celsius to Fahrenheit

□ **1) Approach 1: Using normal way**

□ # function definition to convert from c to F

□ **def** ctof(c):

□f=9/5*c+32

□**return** f

□**2) Approach 2: Using lambda**
□# lambda function
□ctof = **lambda** c:9/5*c+32

□# input c=int(input("Enter Temp(C):"))

□# function call f=ctof(c)

□# print **print**("Temp(F) :",f)

☐ Use of Lambda Expression
□**Example use with filter()**
□The filter() function in Python takes in a function and a list as arguments.
□The syntax of filter() method is:
□**filter(function, iterable)**
□**function** – function that tests if elements of an iterable returns true or false
□# Program to filter out only the even items from a list

□my_list = [1, 5, 4, 6, 8, 11, 3, 12]

□new_list= **list(filter(lambda x: (x%2 == 0) ,my_list))**

□# Output: [4, 6, 8, 12]

□print(new_list)

☐ Use of Lambda Function
□**Example use with map()**

- The map() function in Python takes in a function and a list.
- The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

- ☐ The syntax of map() is:
- ☐ map(function, iterable, ...)
- ☐ **function** - map() passes each item of the iterable to this function.
- ☐ **iterable**iterable which is to be mapped
- ☐ # Program to double each item in a list using map()
- ☐ my_list = [1, 5, 4, 6, 8, 11, 3, 12]
- ☐ new_list = list(map(lambda x: x * 2 , my_list))
- ☐ # Output: [2, 10, 8, 12, 16, 22, 6, 24]
- ☐ print(new_list)

- ☐ Home Work
- ☐ Check out these examples to learn more:
- ☐ Python Program to Find HCF or GCD
- ☐ Python Program to Find LCM
- ☐ Python Program to Make a Simple Calculator

- ☐ What is Object Oriented Programing?

Object-oriented Programing, or ***OOP*** **for** short, is

a programing style which provides a means of structuring programs so that properties and behaviors are bundled into individual *objects*.
Object-oriented programing is an approach for modeling

concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc. OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.

- ☐ Python Object Orinted Programing

□ **Introduction to OOPs in Python**An object has two characteristics:

- attributes
- Behavior

□ Class and Object

Class

A class is a blueprint for the object. **Object**

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined.

Therefore, no memory or storage is allocated.

Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

□ Define a class in Python

□ In Python, we define a class using the keyword **class.**

□ The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended.

Here is a simple class definition.

```
class MyClass:
    "This is my first class"
```

```
 a = 10

def func(self):

 print('Hello')
# Output: 10
print(MyClass.a)
# Output: <function MyClass.func at
0x0000000003079BF8>print(MyClass.func)
# Output: 'This is my first class‘
print(MyClass.doc)
```

☐ Class Example

```
class Account(object):
   definit(self, holder, number, balance,min_bal=1500): self.Holder
    = holder
    self.Number = number

    self.Balance = balance
   self.min_bal  =  min_baldef
     deposit(self, amount):
    self.Balance =
   self.Balance+amountdef
   withdraw(self, amount):
    if(self.Balance - amount < -self.min_bal):
           print("insufficient fund")
           return False

    else:
```

s
e
l
f
.
B
a
l
a

```
n
c          r
e          e
           t
-          u
=          r
           n
a
m          T
o          r
u          u
n          e
t
    def balance(self):
     return self.Balance
```

☐ Creating a class

We start the class definition with the class keyword, followed by the class name and a colon.
Inside the class body, we define three functions – these are our object's methods. The first is called___init__, which is a special method. When we call the class object, a new instance of the class is created, and the initmethod on this new object is immediately executed with all the parameters that we passed to the class object. The purpose of this method is thus to set up a new object using data that we have provided.
You may have noticed that both of these method definitions have self as the first parameter, and we use this variable inside the method bodies – but we don't appear to pass this parameter in. This is because whenever we call a method on an object, the object itself is automatically passed in as the first parameter. This gives us a way to access the object's properties from inside the object's methods.

☐ Creating an Object in Python

The procedure to create an object is similar to a function call.

The first argument of the function in class must be the object itself.

This is conventionally called
***self*.** class MyClass:
"This is my second class“
a = 10
def func(self):
print('Hello')# create a new
MyClasso**b = MyClass()**
# Output: <function MyClass.func at 0x000000000335B0D0>

print(MyClass.func)

# Output: <bound method MyClass.func of <main.MyClass object at 0x000000000332DEF0>>
print(ob.func)

# Calling function func()
# Output:
Helloob.func()

- ☐ Constructor in Class

- ☐ All classes have a function called**init(),** which is always executed when the class is being initiated.
- ☐ This type of function is also called **constructors** in Object Oriented Programing (OOP). We normally use it **to initialize** all the variables.

- ☐ Python Inheritance

- ☐ Python Inheritance enables us to use the member attributes and methods of one class into another.

Python Inheritance Terminologies

Superclass**:** The class **from which** attributes and methods will be inherited.

Subclass: The class **which inherits** the members from superclass.

Method Overloading**: Redefining** the definitions of methods **in subclass** which was **already defined** in **superclass.**

- ☐ Python Multiple Inheritance
- ☐ One class extending **more than one class** is called multiple inheritance. This is one of the cool specialties of python which makes it more convenient than java in some cases (Java doesn't support multiple inheritance).

Multiple Inheritance vs Multi-level Inheritance

☐ Printing Student Result

☐ Python Module

Modules refer to **a file** containing Python statements and definitions. A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.
We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.
We can define our most used functions in a module and import it, instead of copying their definitions into different programs.
Let us create a module. Type the following and save it as example.py.

☐ Python Module

☐ Let us create a module. Type the following and save it as example.py

```
# Python Module example
 def add(a, b):
 """This program adds two numbers and return the result""" result =
    a + b
return result
```

How to import modules in Python?

We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

☐ >>> import example

Using the module name we can access the function using **dot (.)** operation. For example:

- □ >>>example.add(4,5.5)
- □ 9.5

□ Python File

What is a file?

File is a **named location** on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

When we want to read from or write to a file we need to **open it first.** When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

- Open a file
- Read or write (perform operation)
- Close the file

□ Python File Operation

□ How to open a file?

□ Python has a built-in function **open()** to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

>>> f = open("test.txt")

# open file in current directory

>>> f = open("C:/Python33/README.txt")

# specifying full path

We can specify the **mode** while opening a file. In mode, we specify whether we want to **read 'r', write 'w'** or **append 'a'** to the file.

The **default is reading in text mode.** In this mode, we get strings when reading from the file.

☐ Python File Operation

f = open("test.txt")
# equivalent to 'r' or
'rt' f =
open("test.txt",'w') #
write in text mode
f = open("img.bmp",'r+b')

# read and write in binary mode

When working with files in text mode, it is highly recommended to specify the **encoding type.**

f = open("test.txt",mode = 'r',encoding = 'utf-8')

<u>How to close a file Using Python?</u>

f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()

☐ Python File Operation

☐ <u>How to write to File Using Python?</u>

with open("test.txt",'w',encoding = 'utf-8') **as f:**
f.write("my first file\n")
f.write("This file\n\n")
f.write("contains three
lines\n")

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists.

We can use 'a' append mode to append a file

☐ Python File Operation

- >>> f = open("test.txt",'r',encoding = 'utf-8') >>>f.read(4) # read the first 4 data 'This' >>>f.read(4) # read the next 4 data ' is ' >>>f.read() # read in the rest till end of file
- >>>f.read() # further reading returns empty sting ''

>>>f.tell() # get the current file position

>>>f.seek(0) # bring file cursor to initial position 0

- Python File Operation

- We can read a file **line-by-line** using a **for loop.** This is both efficient and fast.

```
>>> for line in f:
        print(line, end = '')
```

Alternately, we can use **readline()** method to read individual lines of a file. This method reads a file till the newline, including the newline character.

>>>f.readline()

- Lamda Expression

- The lambda operator or lambda function is a way to create **small anonymous functions**, i.e. functions **without a name.** These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination with the functions filter(), map() and reduce(). The general syntax of a lambda function is quite simple:

- lambda **argument_list**: **expression**

- The argument list consists of a comma separated list of arguments and the expression is an arithmetic expression using these arguments.

- >>> f = lambda x, y : x + y

☐ >>> f(1,1)

☐ Lamda with reduce function

☐ The function **reduce(func, seq)** continually applies the function **func()** to the sequence seq. It returns a single value.

☐ >>> reduce(lambda x,y: x+y, [47,11,42,13])
☐ 113

☐ >>> f = lambda a,b: a if (a > b) else b

☐ >>> reduce(f, [47,11,42,102,13])

☐ >>> 102

☐ Exception Handling

☐ An **exception** is an **error** that happens during the execution of a program.

☐ Error handling is generally resolved by saving the state of execution at the moment the error occurred and interrupting the normal flow of the program to execute a special function or piece of code, which is known as the **exception handler.**

☐ >>> n = int(raw_input("Please enter a number: "))

☐ Please enter a number: **23.5**

☐ Traceback (most recent call last): File "<stdin>", line 1, in

<module>ValueError: invalid literal for int() with base 10: '23.5

☐ Exception Handling

☐ while True:

```
try:
    n = raw_input("Please enter an integer: ")
    n = int(n)
```

- □   break
- □   **except ValueError:**
- □   print("No valid integer! Please try again ...")

□ print "Great, you successfully entered an integer!"

□ Exception Handling

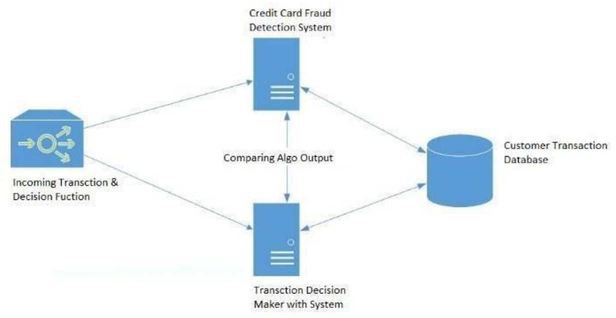
□ A try statement may have **more than one except clause** for different exceptions. But at most one except clause will be executed.

```
import sys
try:
   f = open('integers.txt')
   s = f.readline()
   i = int(s.strip())

except IOErroras (errno, strerror):

   print "I/O error({0}): {1}".format(errno, strerror)
except ValueError:
  print "No valid integer in line."
except:
  print "Unexpected error:", sys.exc_info()[0]
```

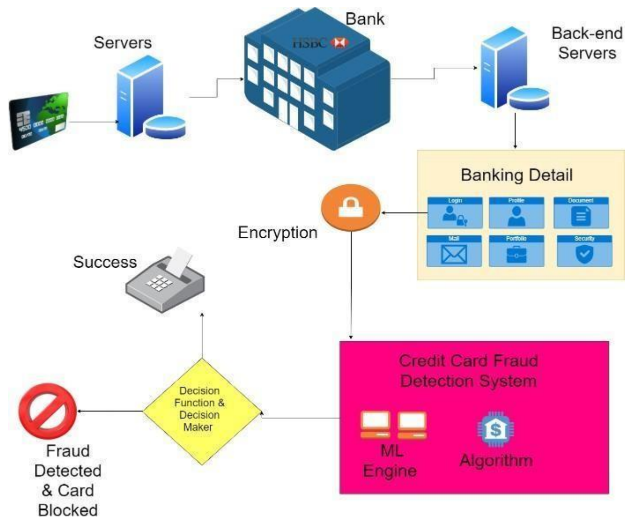
III. METHODOLOGY

The approach that this paper proposes, uses the latest machine learning algorithms to detect anomalous activities, called outliers.

The basic rough architecture diagram can be represented with the following figure:


Credit Card Fraud Detection System
Comparing Algo Output
Customer Transaction Database
Incoming Transction & Decision Fuction
Transction Decision Maker with System

When looked at in detail on a larger scale along with real life elements, the full architecture diagram can be represented as follows:

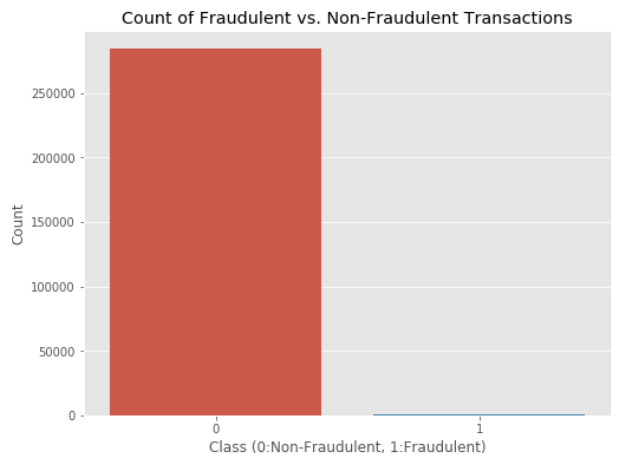

Servers
Bank
Back-end Servers
Banking Detail
Encryption
Success
Credit Card Fraud Detection System
ML Engine
Algorithm
Decision Function & Decision Maker
Fraud Detected & Card Blocked

First of all, we obtained our dataset from Kaggle, a data analysis website which provides datasets.

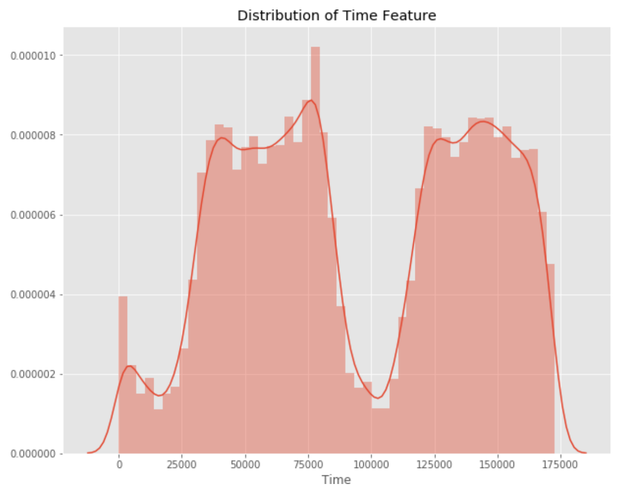
Inside this dataset, there are 31 columns out of which 28 are named as v1-v28 to protect sensitive data.

The other columns represent Time, Amount and Class. Time shows the time gap between the first transaction and the following one. Amount is the amount of money transacted. Class 0 represents a valid transaction and 1 represents a fraudulent one.

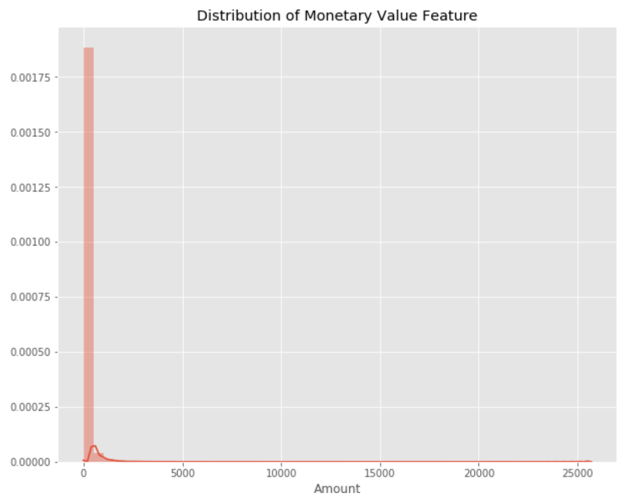
We plot different graphs to check for inconsistencies in the dataset and to visually comprehend it:



This graph shows that the number of fraudulent transactions is much lower than the legitimate ones.

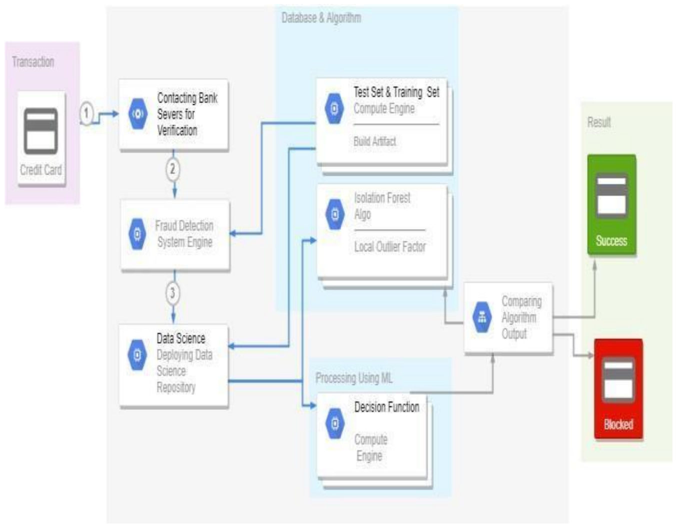


This graph shows the times at which transactions were done within two days. It can be seen that the least number of transactions were made during night time and highest during the days.

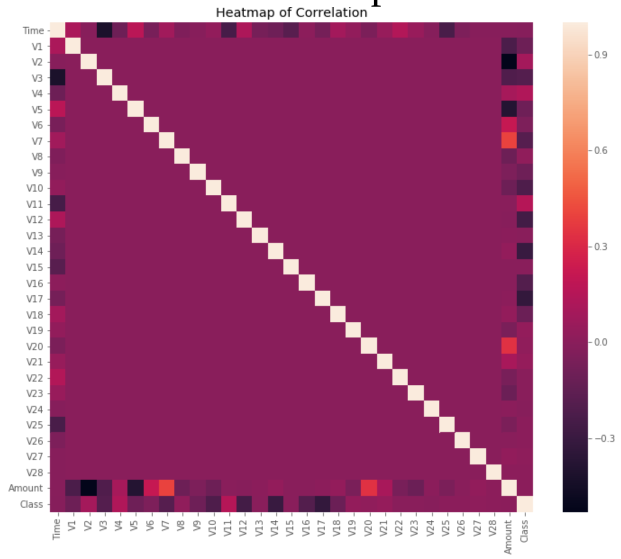


This graph represents the amount that was transacted. A majority of transactions are relatively small and only a handful of them come close to the maximum transacted amount.

After checking this dataset, we plot a histogram for every column. This is done to get a graphical representation of the dataset which can be used to verify that there are no missing any values in the dataset. This is done to ensure that we don't require any missing value imputation and the machine learning algorithms can process the dataset smoothly.



After this analysis, we plot a heatmap to get a coloured representation of the data and to study the correlation between out predicting variables and the class variable. This heatmap is shown below:



The dataset is now formatted and processed. The time and amount column are standardized and the Class column is removed to ensure fairness of evaluation. The data is processed by a set of algorithms from modules. The following module diagram explains how these algorithms work together: This data is fit into a model and the following outlier detection modules are applied on it:

- Local Outlier Factor
- Isolation Forest Algorithm

These algorithms are a part of sklearn. The ensemble module in the sklearn package includes ensemble-based methods and functions for the classification,

regression and outlier detection.
This free and open-source Python library is built using NumPy, SciPy and matplotlib modules which provides a lot of simple and efficient tools which can be used for data analysis

and machine learning. It features various classification, clustering and regression algorithms and is designed to interoperate with the numerical and scientific libraries.

We've used Jupyter Notebook platform to make a program in Python to demonstrate the approach that this paper suggests. This program can also be executed on the cloud using Google Collab platform which supports all python notebook files.

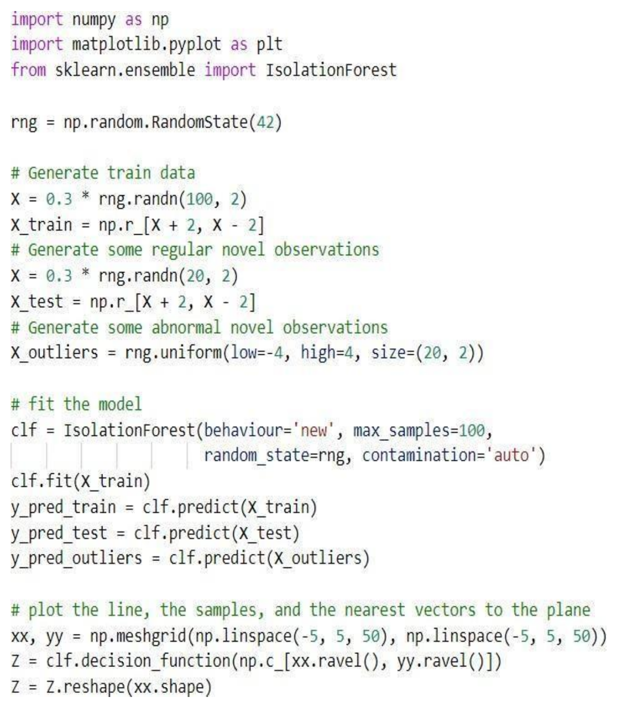
Detailed explanations about the modules with pseudocodes for their algorithms and output graphs are given as follows:

A. *Local Outlier Factor*

It is an Unsupervised Outlier Detection algorithm. 'Local Outlier Factor' refers to the anomaly score of each sample. It measures the local deviation of the sample data with respect to its neighbours.

More precisely, locality is given by k-nearest neighbours, whose distance is used to estimate the local data.

The pseudocode for this algorithm is written as:



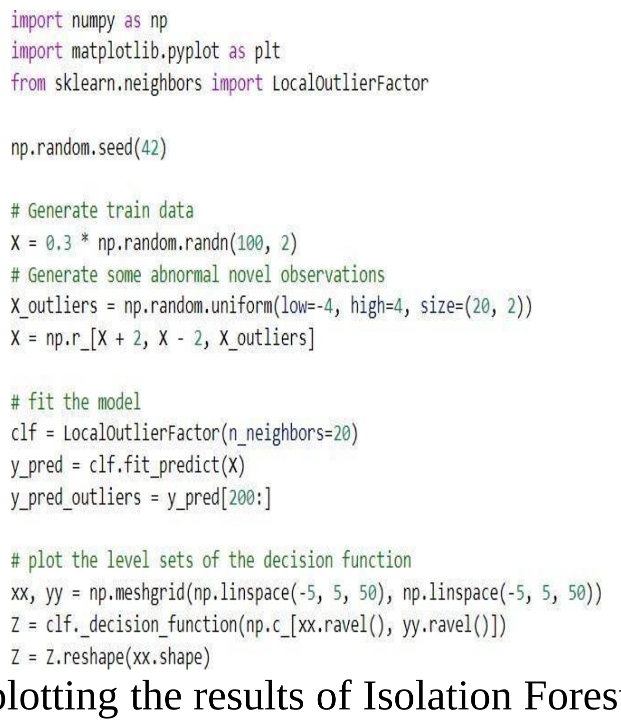
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(42)

# Generate train data
X = 0.3 * rng.randn(100, 2)
X_train = np.r_[X + 2, X - 2]
# Generate some regular novel observations
X = 0.3 * rng.randn(20, 2)
X_test = np.r_[X + 2, X - 2]
# Generate some abnormal novel observations
X_outliers = rng.uniform(low=-4, high=4, size=(20, 2))

# fit the model
clf = IsolationForest(behaviour='new', max_samples=100,
                      random_state=rng, contamination='auto')
clf.fit(X_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)

# plot the line, the samples, and the nearest vectors to the plane
xx, yy = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

On plotting the results of Local Outlier Factor algorithm, we get the following figure:

By comparing the local values of a sample to that of its neighbours, one can identify samples that are substantially lower than their neighbours. These values are quite amanous and they are considered as outliers.
As the dataset is very large, we used only a fraction of it in out tests to reduce processing times.
The final result with the complete dataset processed is also determined and is given in the results section of this paper.

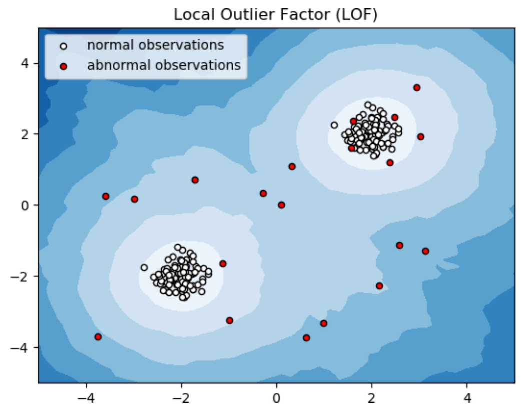
B. *Isolation Forest Algorithm*

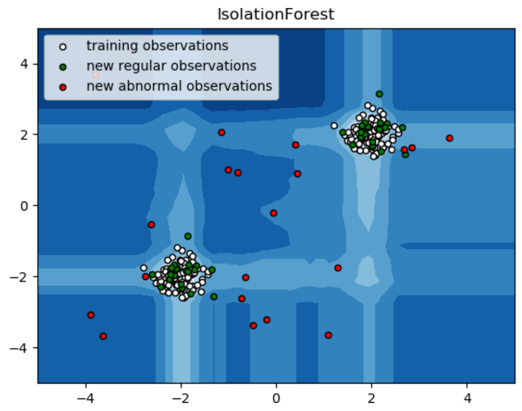
The Isolation Forest ‘isolates’ observations by arbitrarily selecting a feature and then randomly selecting a split value between the maximum and minimum values of the designated feature.
Recursive partitioning can be represented by a tree, the number of splits required to isolate a sample is equivalent to the path length root node to terminating node.
The average of this path length gives a measure of normality and the decision function which we use.
The pseudocode for this algorithm can be written as:



```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor

np.random.seed(42)

# Generate train data
X = 0.3 * np.random.randn(100, 2)
# Generate some abnormal novel observations
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))
X = np.r_[X + 2, X - 2, X_outliers]

# fit the model
clf = LocalOutlierFactor(n_neighbors=20)
y_pred = clf.fit_predict(X)
y_pred_outliers = y_pred[200:]

# plot the level sets of the decision function
xx, yy = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))
Z = clf._decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

On plotting the results of Isolation Forest algorithm, we get the following figure:


Local Outlier Factor (LOF)
normal observations
abnormal observations
4
2
0
−2
−4
−4
−2
0
2
4


IsolationForest
training observations
new regular observations
new abnormal observations
4
2
0
−2
−4
−4
−2
0
2
4

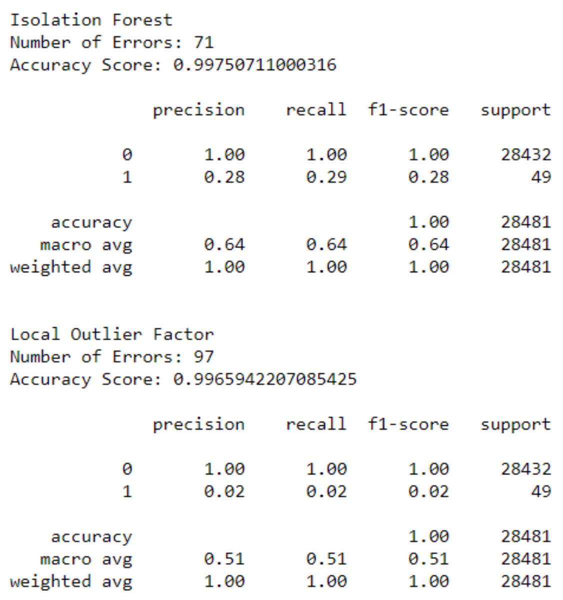
Partitioning them randomly produces shorter paths for anomalies. When a forest of random trees mutually produces shorter path lengths for specific samples, they are extremely likely to be anomalies.
Once the anomalies are detected, the system can be used to report them to the concerned authorities. For testing purposes, we are comparing the outputs of these algorithms to determine their accuracy and precision.

IV. IMPLEMENTATION

This idea is difficult to implement in real life because it requires the cooperation from banks, which aren't willing to share information due to their market competition, and also due to legal reasons and protection of data of their users.
Therefore, we looked up some reference papers which followed similar approaches and gathered results. As stated in one of these reference papers:
"This technique was applied to a full application data set supplied by a German bank in 2006. For banking confidentiality reasons, only a summary of the results obtained is presented below. After applying this technique, the level 1 list encompasses a few cases but with a high probability of being fraudsters.
All individuals mentioned in this list had their cards closed to avoid any risk due to their high-risk profile. The condition is more complex for the other list. The level 2 list is still restricted adequately to be checked on a case by case basis.
Credit and collection officers considered that half of the cases in this list could be considered as suspicious fraudulent behaviour. For the last list and the largest, the work is equitably heavy. Less than a third of them are suspicious.
In order to maximize the time efficiency and the overhead charges, a possibility is to include a new element in the query; this element can be the five first digits of the phone numbers, the email address, and the password, for instance, those new queries can be applied to the level 2 list and level 3 list.".

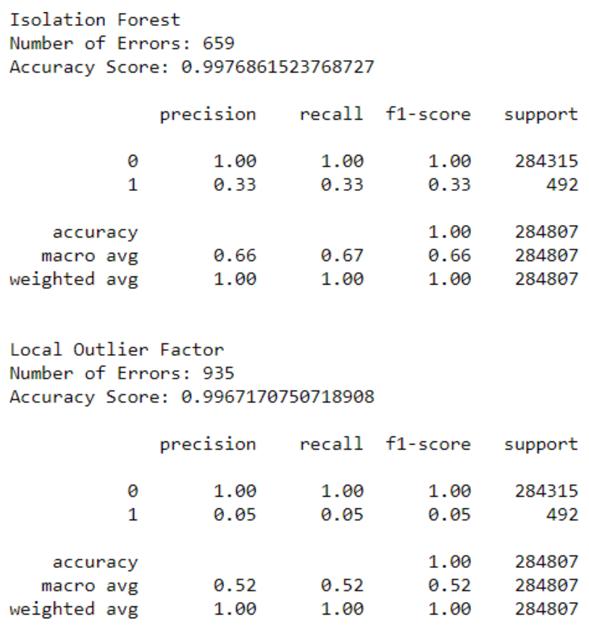
V. RESULTS

The code prints out the number of false positives it detected and compares it with the actual values. This is used to calculate the accuracy score and precision of the algorithms.
The fraction of data we used for faster testing is 10% of the entire dataset. The complete dataset is also used at the end and both the results are printed.
These results along with the classification report for each algorithm is given in the output as follows, where class 0 means the transaction was determined to be valid and 1 means it was determined as a fraud transaction.
This result matched against the class values to check for false positives.
Results when 10% of the dataset is used:



```
Isolation Forest
Number of Errors: 71
Accuracy Score: 0.99750711000316

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.28      0.29      0.28        49

    accuracy                           1.00     28481
   macro avg       0.64      0.64      0.64     28481
weighted avg       1.00      1.00      1.00     28481


Local Outlier Factor
Number of Errors: 97
Accuracy Score: 0.9965942207085425

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.02      0.02      0.02        49

    accuracy                           1.00     28481
   macro avg       0.51      0.51      0.51     28481
weighted avg       1.00      1.00      1.00     28481
```

Results with the complete dataset is used:



```
Isolation Forest
Number of Errors: 659
Accuracy Score: 0.9976861523768727

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.33      0.33      0.33       492

    accuracy                           1.00    284807
   macro avg       0.66      0.67      0.66    284807
weighted avg       1.00      1.00      1.00    284807


Local Outlier Factor
Number of Errors: 935
Accuracy Score: 0.9967170750718908

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.05      0.05      0.05       492

    accuracy                           1.00    284807
   macro avg       0.52      0.52      0.52    284807
weighted avg       1.00      1.00      1.00    284807
```

VI. 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 it.

VII. FUTURE ENHANCEMENTS

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.

REFERENCES

[1] "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

[2] 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, Australia

[3] "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

[4] "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

[5] "Credit Card Fraud Detection through Parenclitic Network Analysis- By Massimiliano Zanin, Miguel Romance, Regino Criado, and SantiagoMoral" published by Hindawi Complexity Volume 2018,
Article ID 5764370, 9 pages

[6] "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

[7] "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

[8] David J.Wetson,DavidJ.Hand,MAdams,Whitrow and Piotr Jusczak "Plastic Card Fraud Detection using Peer Group Analysis" Springer, Issue 2008.