**CREDIT CARD FRAUD DETECTION USING PREDICTIVE MODELLING.**

**Lakshmi Sruti Veda Pidatala-801053371**

**Sreenivasa Ganapati Deety-801019748**

**Akash Phani Raghava Yelisetty-801044039**

**Tushara Vepakomma-801044941**

**ABSTRACT**

The billing solution to most individuals in the given day tends to be payments made through credit cards. With the rise in using these plastic payments as opposed to that of cash, there is a rise in fraudulent transactions. As easy as these transactions are, the need for security on these mechanisms is a must. Any fraudulent transactions can be detected with the help of advanced machine learning techniques. However, designing these fraud detection algorithms is a task and is highly challenging. This is especially challenging because of highly unbalanced class distribution. Also, public data is not readily available because of confidentiality issues.

In this thesis, we aim to deal with unbalanced and evolving data streams which are non-stationary using logistic regression machine learning algorithms. Predictive modelling is applied on the considered dataset and the dataset is split into test and train datasets to perform further regression techniques. Confusion matrix is further developed to have a better understanding and assess the feedbacks provided by investigators on the fraud alerts.

**INTRODUCTION**

**1.1 INTRODUCTION AND HISTORY OF PYTHON:**

Python is a high-level and object-oriented scripting language. It can be both interpreted and interactive. One of the assets of using python is that it is highly readable. It makes use of

fewer syntactical constructions than many other languages. It also includes a rich set of supporting libraries. Python has few keywords, simple structure, and a clearly defined syntax. Python supports multiple programming paradigms including the object oriented, functional, procedural, and has a standard library. It also has support for an interactive mode which allows interactive testing and debugging of snippets of code. Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. It is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

**1.2 PROJECT DESCRIPTION**

In the virtual mode of payment, i.e. credit-based transactions, for an attack to take place the attackers need only little information for doing fraudulent transaction (secure code, card number, expiration date etc.). Most of the time, the genuine cardholder is not aware

that someone else has seen or stolen his card information. The only way to detect this kind of fraud is to analyse the spending patterns on every card and to figure out any inconsistency with respect to the “usual” spending patterns. Fraud detection based on the analysis of existing purchase data of cardholder is a promising way to reduce the rate of successful credit card frauds. Since humans tend to exhibit specific behaviouristic profiles, every cardholder can be represented by a set of patterns containing information about the typical purchase category, the time since the last purchase, the amount of money spent etc.

Our project makes use of a dataset that has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

**PROBLEM STATEMENT**

Credit card fraud stands as major problem for word wide financial institutions observed from many financial reports such as (Bhattacharyya et al., 2011) ,fraud in United Kingdom alone was estimated to be £535 million in 2007 and now costing around 13.9 billion a year (Mahdi et al., 2010)and has been on the surge ever since. Perpetrators are also evolving their methods and practices to avoid detection. Thus an effective and innovative methods need to be develop which will evolve accordingly to the need. The initial implementation was done by k-means algorithm and clustering. To train such a problem did not prove to be fruitful, because they produced similar results for both the hyper-parameters(fraud and non-fraud). Even a change in dimensionality of data, did not bring up any results. Clustering in-fact was less comparable to k-means algorithm and regression models in general, because it could not bring any less accuracy scores for such a scenario.

**PROPOSED SYSTEM**

Our goal is to implement machine learning model in order to classify, to the highest possible degree of accuracy, credit card fraud from a dataset. After initial data exploration, implementing a logistic regression model turned out to be most suitable for best accuracy reports. Logistic regression, is also a good candidate for binary classification.

Python sklearn library is proposed to be used to implement the project, using pandas to data frame for class ==0 for no fraud and class==1 for fraud, matplotlib for plotting the fraud and non fraud data, train\_test\_split for data extraction (Split arrays or matrices into random train and test subsets) and used Logistic Regression machine learning algorithm for fraud detection and print predicting score according to the algorithm. Finally Confusion matrix is going to be plotted on true and predicted.

**WHY PYTHON?**

Software development companies generally prefer Python because of its versatile features such as providing large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Data pre-processing is easier on Python.

Here, as we considered a dataset which is in a csv format, implementation of this input data is helped by using data frames which aides in lesser lines of codes. We’ve made use of some library functions which made it easier to develop this project.

**Numpy**: 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. The number of axes is rank. It offers Matlab-ish capabilities within Python. It also performs fast array operations. It is most suitable for 2D arrays, multi-D arrays, linear algebra etc. implementation.

**Matplotlib**: It supports a very wide variety of graphs and plots like histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

**Pandas:** Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

**ELEMENTS OF PYTHON**

* **Reserved words:** Keywords are the reserved words in Python. They are used to define the syntax and structure of the Python language. There are 33 keywords in Python. Some of them are import, assert, break, raise etc. Reserved words are case sensitive and are lower case except for keywords like True, False, None.
* **Data types:** Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.Python has five standard data types.

1. **Numbers**: Number data types store numeric values. Number objects are created when you assign a value to them
2. **String**: Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
3. **List**: Lists are the most versatile of Python’s compound data types. A list contains items separated by commas and enclosed within square brackets [ ]. To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type. The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator.
4. **Tuple**: A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses. The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as read-only lists.
5. **Dictionary**: Python’s dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

**SAMPLE PROGRAM IMPLEMENTING THE DATA TYPES**:

#List

buyinglist = ['marker', 'glue', 'umbrella', 'pen']

print('I am having', len(buyinglist), 'items to be purchased in my basket.')

print('The items in my basket are', end=' ')

for item in buyinglist:

print(item, end=' ')

print('\nI need to buy a bottle too')

buyinglist.append('bottle')

print('My items in basket are', buyinglist)

print('I have to sort the list now based on items in basket')

buyinglist.sort()

print('Sorted items in the list are', buyinglist)

print('The first item i will purchase is', buyinglist[0])

firstitem = buyinglist[0]

del buyinglist[0]

print('I already purchased the', firstitem)

print('My items in basket now', buyinglist)

#Tuple

basket1= ('pen', 'pencil', 'eraser')

print('Number of items in the basket1 are', len(basket1))

basket2 = 'laptop', 'mobilephone', basket1

print('Number of items in basket2 are', len(basket2))

print('All items in basket2 are', basket2)

print('Items present in basket1 are', basket2[2])

print('Last item in the basket1 is', basket2[2][2])

print('Number ofitems in the basket2 are',

len(basket2)-1+len(basket2[2]))

# Dictionary

bh = {

'Tushara': 'tushara.vepa@gmail.com',

'Sruti': 'srutiveda@gmail.com',

'Tanvi': 'tanvi@gmail.com',

'Apurwa': 'ap@gmail.com'

}

print("Tushara's address is", bh['Tushara'])

print('\nThere are {} names in the bh\n'.format(len(bh)))

# Deleting key-value from dictionary

del bh['Apurwa']

print('\nThere are {} names in bh\n'.format(len(bh)))

for name, address in bh.items():

print('Try to Contact {} at {}'.format(name, address))

# Adding a key-value pair in dictionary

bh['Laasya'] = 'laasya@gmail.com'

if 'Laasya' in bh:

print("\nLaasya's address is", bh['Laasya'])

#Sequence

buyinglist = ['pencil', 'eraser', 'umbrella', 'notebook']

name = 'Tushara'

#Indexing

print('Item 0 is', buyinglist[0])

print('Item 1 is', buyinglist[1])

print('Item 2 is', buyinglist[2])

print('Item 3 is', buyinglist[3])

print('Item -1 is', buyinglist[-1])

print('Item -2 is', buyinglist[-2])

print('Character 0 is', name[0])

# Slicing#

print('Item 1 to 3 is', buyinglist[1:3])

print('Item 2 to end is', buyinglist[2:])

print('Item 1 to -2 is', buyinglist[1:-1])

print('Item start to end is', buyinglist[:])

# Slicing on the string #

print('characters 1 to 3 is', name[1:3])

print('characters 2 to end is', name[2:])

print('characters 1 to -1 is', name[1:-1])

print('characters start to end is', name[:])

#Set

items = set(['pen', 'pencil', 'eraser'])

'pen' in items

'tushara' in items

items2 = items.copy()

items2.add('notebook')

items2.issuperset(items)

items.remove('pen')

#strings

a = "Hello,how are you"

print(a[1])

b = "Hello,how are you"

print(b[2:5])

a = " Hello,how are you"

print(a.strip())

a = "Hello,how are you"

print(len(a))

a = "Hello,how are you"

print(a.lower())

a = "Hello,how are you"

print(a.upper())

a = "Hello,how are you"

print(a.replace("H", "K"))

a = "Hello,how are you"

print(a.split(","))

#Frozenset

items = ('a', 'b', 'c', 'd', 'e')

fSet = frozenset(items)

print('The frozen set is:', fSet)

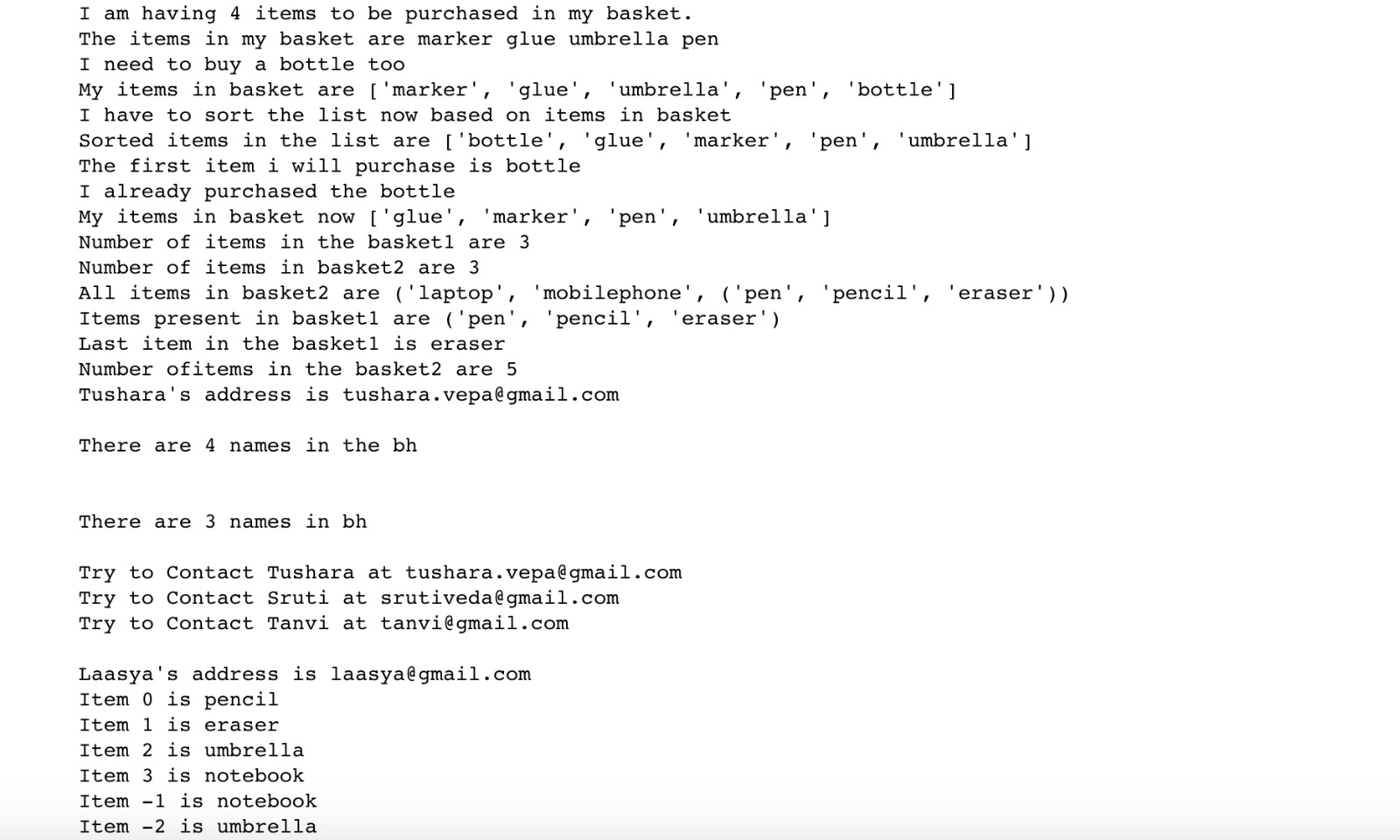
print('The empty frozen set is:', frozenset())

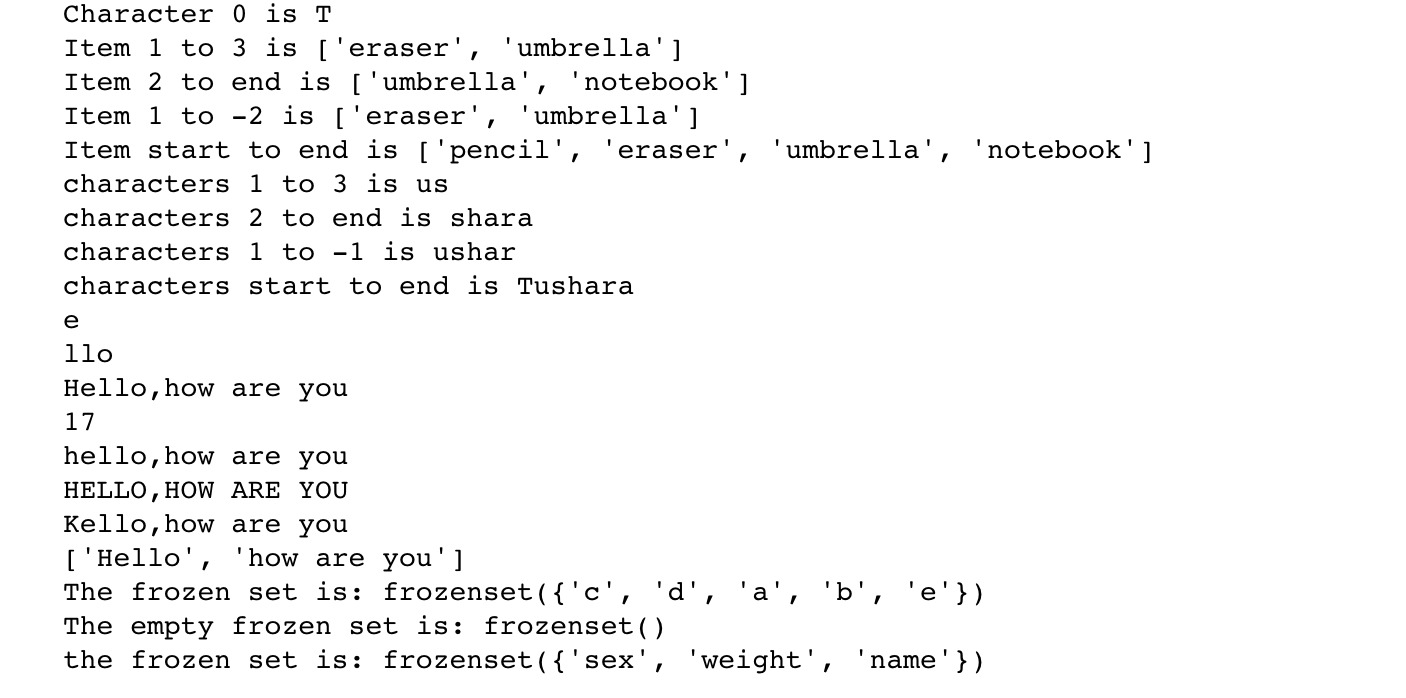
girl = {"name": "Tushara", "weight":54, "sex":"female"}

fset = frozenset(girl)

print("the frozen set is:", fset)

**OUTPUT:**

****



**SYNTAX EXPLAINATION OF PYTHON**

**Logical line**: It is created from one or more physical lines. It contains spaces, tabs and each logical line is terminated by a token NEWLINE. A physical line is a sequence of characters terminated by an end-of-line

sequence

**Comment**: A comment begins with a hash character(#) which is not a part of the string literal and ends at the end of the physical line.

**How to join physical lines?** When a physical line ends with a backslash characters(\) and is not a part of a string literal or comment then it can join another physical line.

**To write two statements in one line:** You can write two separate statements into a single line using a semicolon (;) character between two line.

**Use of whitespace**: Python uses whitespace to define program blocks whereas other languages like C, C++ use braces ({}) to indicate blocks of codes for class, functions or flow control. The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be the indented same amount.

**CONTROL ABSTRACTIONS OF PYTHON**

Python supports nested loops(if, elif, while), for loop, and control statements like continue, break and pass**.** The below code makes use of these control abstractions.

**#Sample program in python using control structures**

var1 = "tushara is a masters student"

if (type(var1) == str):

print("The variable declared is a String variable")

elif (type(var1) == float):

print("The variable declared is a Float variable")

elif (type(var1) == int):

print("The variable declared is an integer variable")

elif (type(var1) == Bool):

print("The variable declared is a Boolean variable")

else:

print("The type of variable is unknown")

print('Done')

#While loop:

x = 42

proceed = True

while proceed:

random = int(input('Enter an integer : '))

if random == x:

print('The number entered randomly is same as the given x')

proceed = False

elif random < x:

print('The number entered is less than the actual number x')

else:

print('The number entered is higher than the actual number x')

else:

print('The while loop has been executed.')

print('The program is ending now')

print('Done')

#for loop

items = ["pen","pencil","eraser"]

for i in items:

print(i)

#break statement:

items = ["pen","pencil","eraser"]

for i in items:

print(i)

if i == "pencil":

break

#continue statement:

while True:

a = input('Enter a value through console ')

if a == 'exit':

break

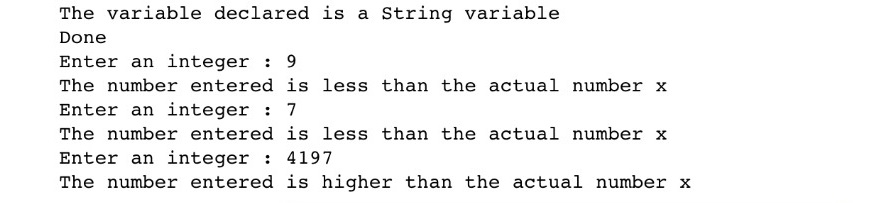
if len(a) < 5:

print('The value entered is very small')

continue

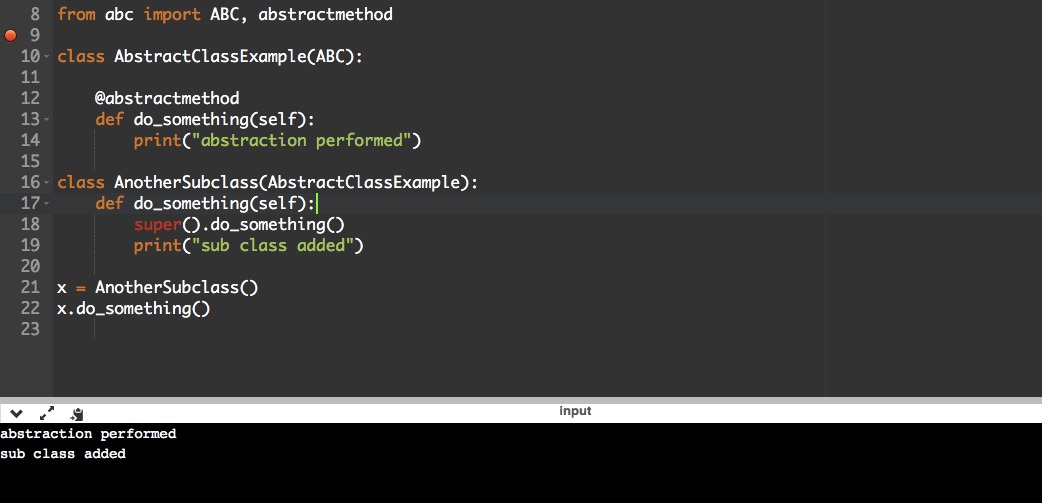
print('The input value entered is of right length')

**OUTPUT**



**HOW THE LANGUAGE HANDLES ABSTRACTION**

Abstraction means providing only essential information about the data to the outside world by hiding the details of implementation. Python handles abstraction as shown below using functions:



**IMPLEMENTATION OF THE PROJECT**

**STRATEGIES USED:**

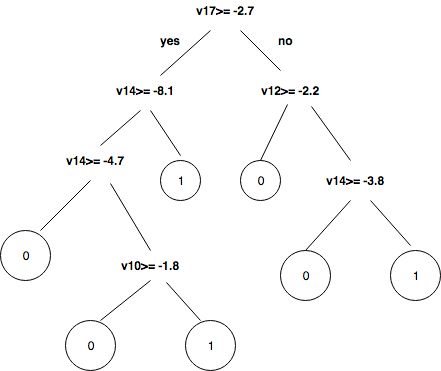
**Logistic regression:** Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable, i.e. the dependent variable is a binary variable that contains data coded as 1 (yes, success, true etc.) or 0 (no, failure, false etc.) against the other variables. There are some factors which are assumed for this case:

* The considered binary logistic regression requires the dependent variable to be binary.
* Inclusion of only meaningful variables.
* Model should have little or no multicollinearity.

**Data Mining:** Data mining consists of a series of procedures before which data can be termed as ‘Knowledge’. For the data to be relevant to the needs of the user, the following steps have to be performed.

* Data cleaning: Process to remove noisy and inconsistent data
* Data integration: Data from various sources are combined.
* Data selection : Data from analysis task is retrieved from database
* Data transformation: Data are transformed or consolidated into forms
* appropriate for mining by performing summary or aggregation operations.
* Data mining: Intelligent methods are applied in order to extract data patterns)
* Pattern evaluation: Identify interesting patterns representing knowledge
* Knowledge presentation: Visualization and knowledge representation techniques are used to present the mined knowledge to the user.

**Decision Tree:** Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too. The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior data(training data).



**Steps performed to achieve the result:**

* In prediction model to predict the continuous valued functions, credit card details of CSV file will is analysed.
* CRISP-DM process is followed.
* Data Understanding is performed by loading the dataset
* Once dataset is loaded in the pandas dataframe, data preparation phase is carried out by the help of data understanding.
* Here, first the variable descriptions and contents of dataset are viewed and the target variable i.e. fraud yes/no is converted to 1/0 variables.
* Predictive modeling splits the data into two partitions 70% of testing and 30%of training check output class distribution to predict the outcome.
* The decision tree to get the result as a tree with root node describes the best predictor in the data, the combination of two or more branches is denoted by decision node (non leaf nodes) and each branch represents a value for the attribute which is tested.
* The leaf node may be 1 in the case of fraud and 0 otherwise.

**PROJECT ENVIRONMENT**

**Hardware Requirements:**

* RAM: 4GB and Higher
* Processor: Intel i3 and above
* Hard Disk: 500GB: Minimum

**Software Requirements:**

* OS: Windows or Linux
* Python IDE : python 2.7.x and above
* Pycharm IDE Required
* Setup tools and pip to be installed for 3.6 and above
* Language : Python Scripting

**CODE:**

**Importing modules:**

import numpy as np

#import sklearn python machine learning module

import sklearn as sk

#import pandas dataframes

import pandas as pd

#import matplotlib for plotting

import matplotlib.pyplot as plt

#import datasets and linear\_model from sklearn module

from sklearn import datasets, linear\_model

#import Polynomial features from sklearn module

from sklearn.preprocessing import PolynomialFeatures

#import train\_test\_split data classification

from sklearn.model\_selection import train\_test\_split

#import ConfusionMatrix from pandas\_ml

from pandas\_ml import ConfusionMatrix

**Loading dataset:**

dataframe = pd.read\_csv(&#39;C:/Python27/creditcard.csv&#39;, low\_memory=False)

#dataframe.sample Returns a random sample of items from an axis of object.

#The frac keyword argument specifies the fraction of rows to return in the

random sample, so frac=1 means return all rows (in random order).

# If you wish to shuffle your dataframe in-place and reset the index

dataframe = dataframe.sample(frac=1).reset\_index(drop=True)

#dataframe.head(n) returns a DataFrame holding the first n rows of

dataframe.

dataframe.head()

print dataframe

**Checking target classes**

fraud\_class = dataframe.loc[dataframe[&#39;Class&#39;] == 1]

#here in dataframe class with 1 label is selected for non\_fraud\_class

non\_fraud\_class = dataframe.loc[dataframe[&#39;Class&#39;] == 0]

**Splitting data**

X = dataframe.iloc[:,:-1]

y = dataframe[&#39;Class&#39;]

#Finding the length of X and y

print(&quot;X and y sizes, respectively:&quot;, len(X), len(y))

#Splitting the training and Testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.35,

random\_state=500)

**Training data**

logistic = linear\_model.LogisticRegression(C=1e5)

#Fitting the Algorithm for X\_train and y\_train

logistic.fit(X\_train, y\_train)

**Testing data**

print(&quot;Score: &quot;, logistic.score(X\_test, y\_test))

print(&quot;Number of frauds on y\_test:&quot;, len(y\_test.loc[dataframe[&#39;Class&#39;] == 1]),

len(y\_test.loc[dataframe[&#39;Class&#39;] == 1]) / len(y\_test))

**Predicting data**

y\_predicted = np.array(logistic.predict(X\_test))

y\_right = np.array(y\_test)

#print y\_test

#The confusion matrix (or error matrix) is one way to summarize the

performance of a classifier

# for binary classification tasks. This square matrix

# consists of columns and rows that list the number of instances as absolute

or

# relative &quot;actual class&quot; vs. &quot;predicted class&quot; ratios.

#Plotting the Confusion matrix for y\_right and y\_predicted

**Confusion matrix**

confusion\_matrix = ConfusionMatrix(y\_right, y\_predicted)

print(&quot;Confusion matrix:&quot;,confusion\_matrix)

confusion\_matrix.plot(normalized=True)

plt.show()

#printing the stats of Confusion matrix

confusion\_matrix.print\_stats()

**MAIN RUNNING PROGRAM :**

#importing the modules

#import numpy n-dimensional array

import numpy as np

#import sklearn python machine learning modules

import sklearn as sk

#import pandas dataframes

import pandas as pd

import unittest

#import matplotlib for plotting

import matplotlib.pyplot as plt

#import datasets and linear\_model from sklearn module

from sklearn import datasets, linear\_model

#importing machine learning mopdules Decision Tree and KNN sklearn

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import NearestNeighbors

from sklearn.neighbors import KNeighborsClassifier

#import Polynomial features from sklearn module

from sklearn.preprocessing import PolynomialFeatures

#import train\_test\_split data classification

from sklearn.model\_selection import train\_test\_split

#import ConfusionMatrix from pandas\_ml

from pandas\_ml import ConfusionMatrix

#reading the csv file from C:/Python27

dataframe = pd.read\_csv('F:/Project Credit Card/code/creditcard1.csv', low\_memory=False)

#dataframe.sample Returns a random sample of items from an axis of object.

#The frac keyword argument specifies the fraction of rows to return in the random sample, so frac=1 means return all rows (in random order).

# If you wish to shuffle your dataframe in-place and reset the index

dataframe = dataframe.sample(frac=1).reset\_index(drop=True)

#dataframe.head(n) returns a DataFrame holding the first n rows of dataframe.

dataframe.head()

print dataframe

#The loc method gives direct access to the dataframe allowing for assignment to specific locations of the dataframe.

#here in dataframe class with 1 label is selected for fraud\_class

def fraud\_nonfraud():

fraud\_class = dataframe.loc[dataframe['Class'] == 1]

#here in dataframe class with 1 label is selected for non\_fraud\_class

non\_fraud\_class = dataframe.loc[dataframe['Class'] == 0]

#printing length of fraud\_class and non\_fraud class

print("Totally we have ", len(fraud\_class), "fraud data class point and", len(non\_fraud\_class), "nonfraudulent data class points.")

#plotting fraudplot for fraud\_class

ax = fraud\_class.plot.scatter(x='Amount', y='Class', color='Red', label='Fraud')

#plotting plot for non\_fraud\_class with fraud\_class

non\_fraud\_class.plot.scatter(x='Amount', y='Class', color='Green', label='Normal', ax=ax)

plt.show()

print("This Feature what is mentioned is based on the class Distribution.")

#Let us see the plot zooming only fraudplot

bx = fraud\_class.plot.scatter(x='Amount', y='Class', color='Orange', label='Fraud')

#Showing the plot

plt.show()

#Again plotting non\_fraud\_class plot with fraud\_class plot

ax = fraud\_class.plot.scatter(x='V15', y='Class', color='Orange', label='Fraud')

non\_fraud\_class.plot.scatter(x='V15', y='Class', color='Blue', label='Normal', ax=ax)

plt.show()

fraud\_nonfraud()

#Dataframes for X all the columns except class and y class columns

X = dataframe.iloc[:,:-1]

y = dataframe['Class']

#Finding the length of X and y

print("X and y sizes, respectively:", len(X), len(y))

#Splitting the training and Testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.35, random\_state=500)

#Calculating the data

print("Train and test sizes, respectively:", len(X\_train), len(y\_train), "|", len(X\_test), len(y\_test))

print("Total number of frauds:", len(y.loc[dataframe['Class'] == 1]), len(y.loc[dataframe['Class'] == 1])/len(y))

print("Number of frauds on y\_test:", len(y\_test.loc[dataframe['Class'] == 1]), len(y\_test.loc[dataframe['Class'] == 1]) / len(y\_test))

print("Number of frauds on y\_train:", len(y\_train.loc[dataframe['Class'] == 1]), len(y\_train.loc[dataframe['Class'] == 1])/len(y\_train))

#Applying Logistic Regression Machine Learning Algorithm

logistic = linear\_model.LogisticRegression(C=1e5)

#Fitting the Algorithm for X\_train and y\_train

logistic.fit(X\_train, y\_train)

dt = tree.DecisionTreeClassifier()

dt.fit(X\_train, y\_train)

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(X\_train, y\_train)

#Scoring

print("Using Logistivc Regression the Accuracy Score is: ", logistic.score(X\_test, y\_test))

print ("Using Decision tree the Accuracy score is ",dt.score(X\_test, y\_test))

print ("Using KNearestNeighbour the Accuracy score is ",classifier.score(X\_test, y\_test))

y\_predicted = np.array(logistic.predict(X\_test))

y\_right = np.array(y\_test)

#print y\_test

#The confusion matrix (or error matrix) is one way to summarize the performance of a classifier

# for binary classification tasks. This square matrix

# consists of columns and rows that list the number of instances as absolute or

# relative "actual class" vs. "predicted class" ratios.

#Plotting the Confusion matrix for y\_right and y\_predicted

confusion\_matrix = ConfusionMatrix(y\_right, y\_predicted)

print("Confusion matrix:",confusion\_matrix)

confusion\_matrix.plot(normalized=True)

plt.show()

#printing the stats of Confusion matrix

confusion\_matrix.print\_stats()

**PLOTTING FRAUD PLOTS:**

#importing the modules

#import numpy n-dimensional array

import numpy as np

#import sklearn python machine learning modules

import sklearn as sk

#import pandas dataframes

import pandas as pd

import unittest

#import matplotlib for plotting

import matplotlib.pyplot as plt

#import datasets and linear\_model from sklearn module

from sklearn import datasets, linear\_model

#importing machine learning mopdules Decision Tree and KNN sklearn

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import NearestNeighbors

from sklearn.neighbors import KNeighborsClassifier

#import Polynomial features from sklearn module

from sklearn.preprocessing import PolynomialFeatures

#import train\_test\_split data classification

from sklearn.model\_selection import train\_test\_split

#import ConfusionMatrix from pandas\_ml

from pandas\_ml import ConfusionMatrix

from django.shortcuts import render

# from models import trees

# Create your views here.

# def Credit(request):

# return render(request, "index.html")

#

# def ShowPage(request):

# f = Card()

# return render(request, "css.html", {'form': f})

# def ShowPage1(request):

# f = Card1()

# return render(request, "css.html", {'form1': f})

def index(request):

# posts = trees.objects.filter().order\_by('published\_date')

dataframe = pd.read\_csv('F:/Project Credit Card/code/creditcard/creditcard1.csv', low\_memory=False)

#dataframe.sample Returns a random sample of items from an axis of object.

#The frac keyword argument specifies the fraction of rows to return in the random sample, so frac=1 means return all rows (in random order).

# If you wish to shuffle your dataframe in-place and reset the index

#dataframe = dataframe.sample(frac=1).reset\_index(drop=True)

#dataframe.head(n) returns a DataFrame holding the first n rows of dataframe.

dataframe.head()

print (dataframe)

#The loc method gives direct access to the dataframe allowing for assignment to specific locations of the dataframe.

#here in dataframe class with 1 label is selected for fraud\_class

fraud\_class = dataframe.loc[dataframe['Class'] == 1]

#here in dataframe class with 1 label is selected for non\_fraud\_class

non\_fraud\_class = dataframe.loc[dataframe['Class'] == 0]

#printing length of fraud\_class and non\_fraud class

print("Totally we have ", len(fraud\_class), "fraud data class point and", len(non\_fraud\_class), "nonfraudulent data class points.")

#plotting fraudplot for fraud\_class

ax = fraud\_class.plot.scatter(x='Amount', y='Class', color='Red', label='Fraud')

#plotting plot for non\_fraud\_class with fraud\_class

non\_fraud\_class.plot.scatter(x='Amount', y='Class', color='Green', label='Normal', ax=ax)

plt.show()

print("This Feature what is mentioned is based on the class Distribution.")

#Let us see the plot zooming only fraudplot

bx = fraud\_class.plot.scatter(x='Amount', y='Class', color='Orange', label='Fraud')

#Showing the plot

plt.show()

#Again plotting non\_fraud\_class plot with fraud\_class plot

ax = fraud\_class.plot.scatter(x='V15', y='Class', color='Orange', label='Fraud')

non\_fraud\_class.plot.scatter(x='V15', y='Class', color='Blue', label='Normal', ax=ax)

plt.show()

#Dataframes for X all the columns except class and y class columns

X = dataframe.iloc[: , :-1]

y = dataframe['Class']

#Finding the length of X and y

print("X and y sizes, respectively:", len(X), len(y))

#Splitting the training and Testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.35, random\_state=500)

#Calculating the data

print("Train and test sizes, respectively:", len(X\_train), len(y\_train), "|", len(X\_test), len(y\_test))

print("Total number of frauds:", len(y.loc[dataframe['Class'] == 1]), len(y.loc[dataframe['Class'] == 1])/len(y))

print("Number of frauds on y\_test:", len(y\_test.loc[dataframe['Class'] == 1]), len(y\_test.loc[dataframe['Class'] == 1]) / len(y\_test))

print("Number of frauds on y\_train:", len(y\_train.loc[dataframe['Class'] == 1]), len(y\_train.loc[dataframe['Class'] == 1])/len(y\_train))

#Applying Logistic Regression Machine Learning Algorithm

logistic = linear\_model.LogisticRegression(C=1e5)

#Fitting the Algorithm for X\_train and y\_train

logistic.fit(X\_train, y\_train)

dt = tree.DecisionTreeClassifier()

dt.fit(X\_train, y\_train)

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(X\_train, y\_train)

#Scoring

print("Using Logistic Regression the Accuracy Score is: ", logistic.score(X\_test, y\_test))

print ("Using Decision tree the Accuracy score is ",dt.score(X\_test, y\_test))

print ("Using KNearestNeighbour the Accuracy score is ",classifier.score(X\_test, y\_test))

y\_predicted = np.array(logistic.predict(X\_test))

y\_right = np.array(y\_test)

#print y\_test

#The confusion matrix (or error matrix) is one way to summarize the performance of a classifier

# for binary classification tasks. This square matrix

# consists of columns and rows that list the number of instances as absolute or

# relative "actual class" vs. "predicted class" ratios.

#Plotting the Confusion matrix for y\_right and y\_predicted

confusion\_matrix = ConfusionMatrix(y\_right, y\_predicted)

print("Confusion matrix:", confusion\_matrix)

confusion\_matrix.plot(normalized=True)

plt.show()

#printing the stats of Confusion matrix

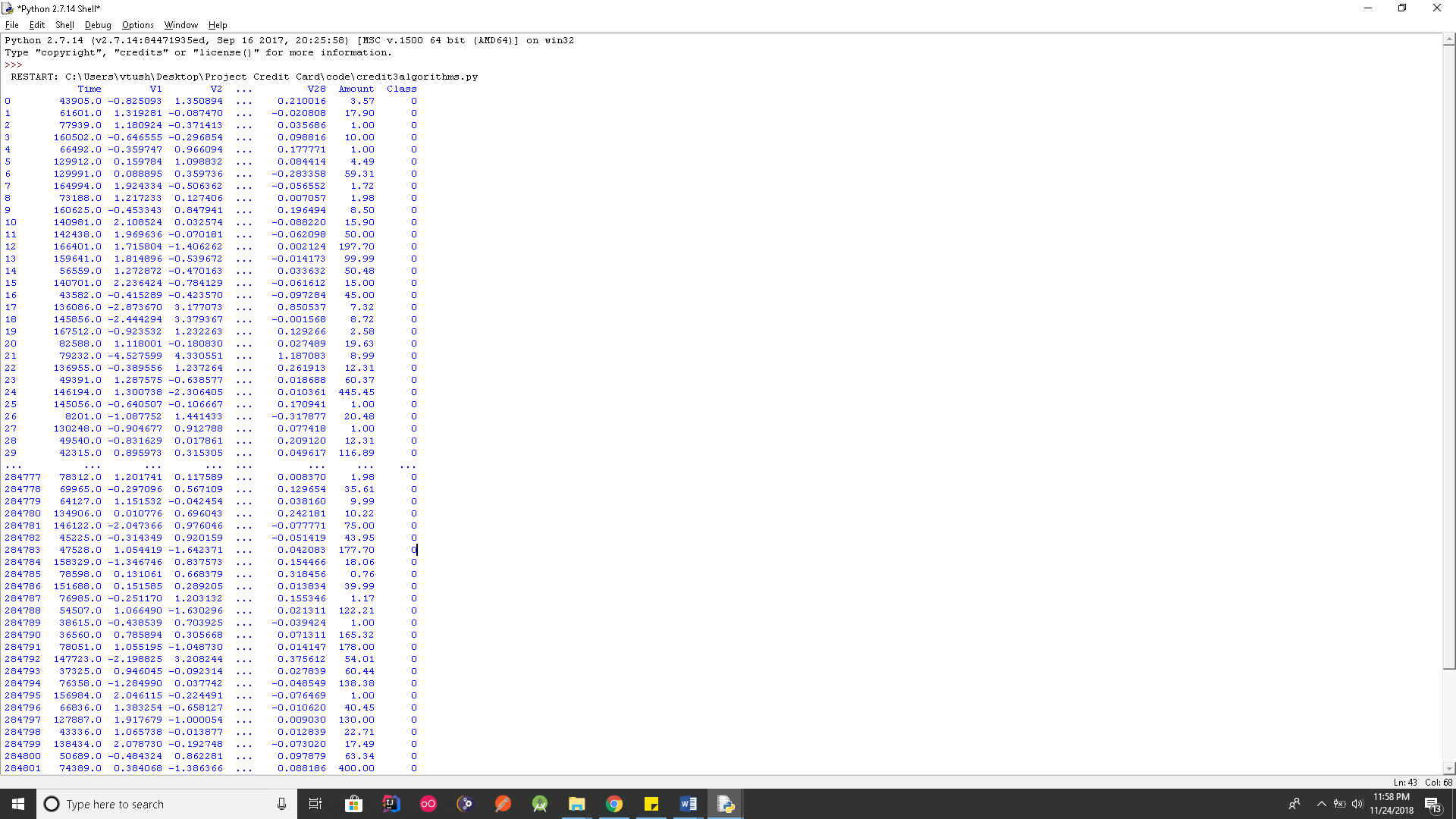
confusion\_matrix.print\_stats()

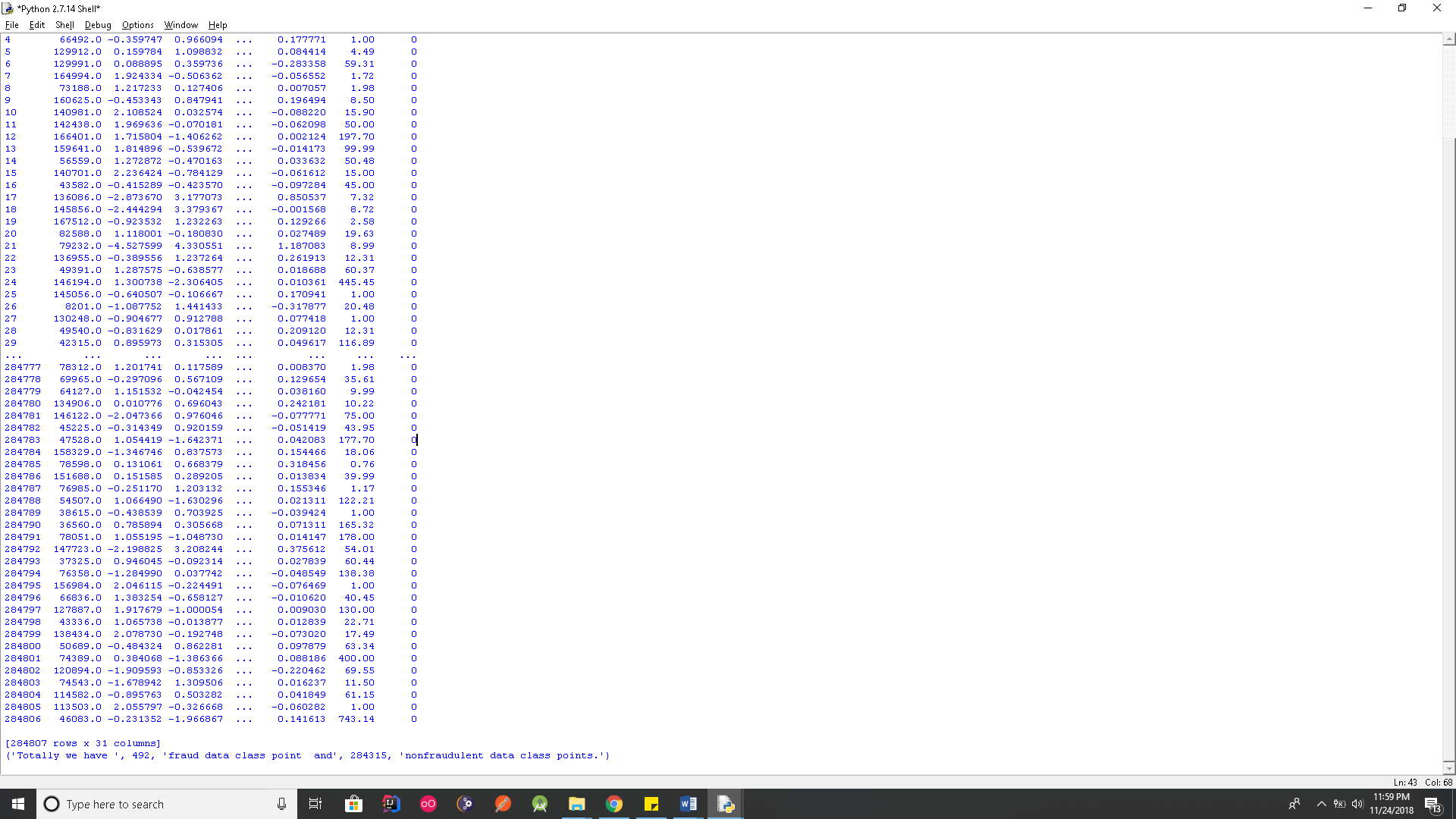
# posts = trees.objects.filter(Logistic\_Regression, Decison\_Tree, K\_nearestNeighbour)

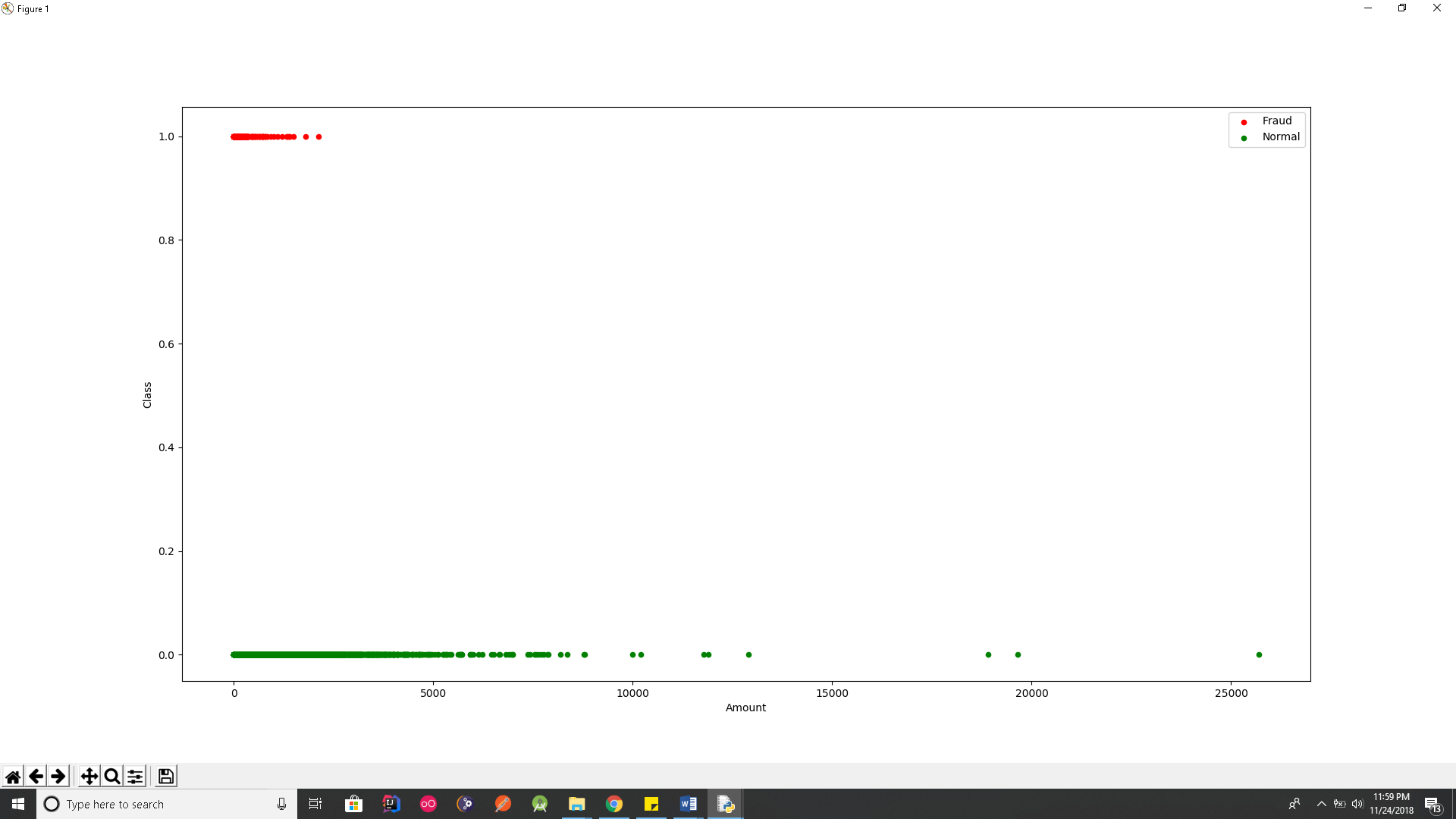
data={'x':len(X),'y':len(y),'fraud':len(fraud\_class),'lr':logistic.score(X\_test,y\_test),'dt':dt.score(X\_test,y\_test),'kn':classifier.score(X\_test,y\_test)}

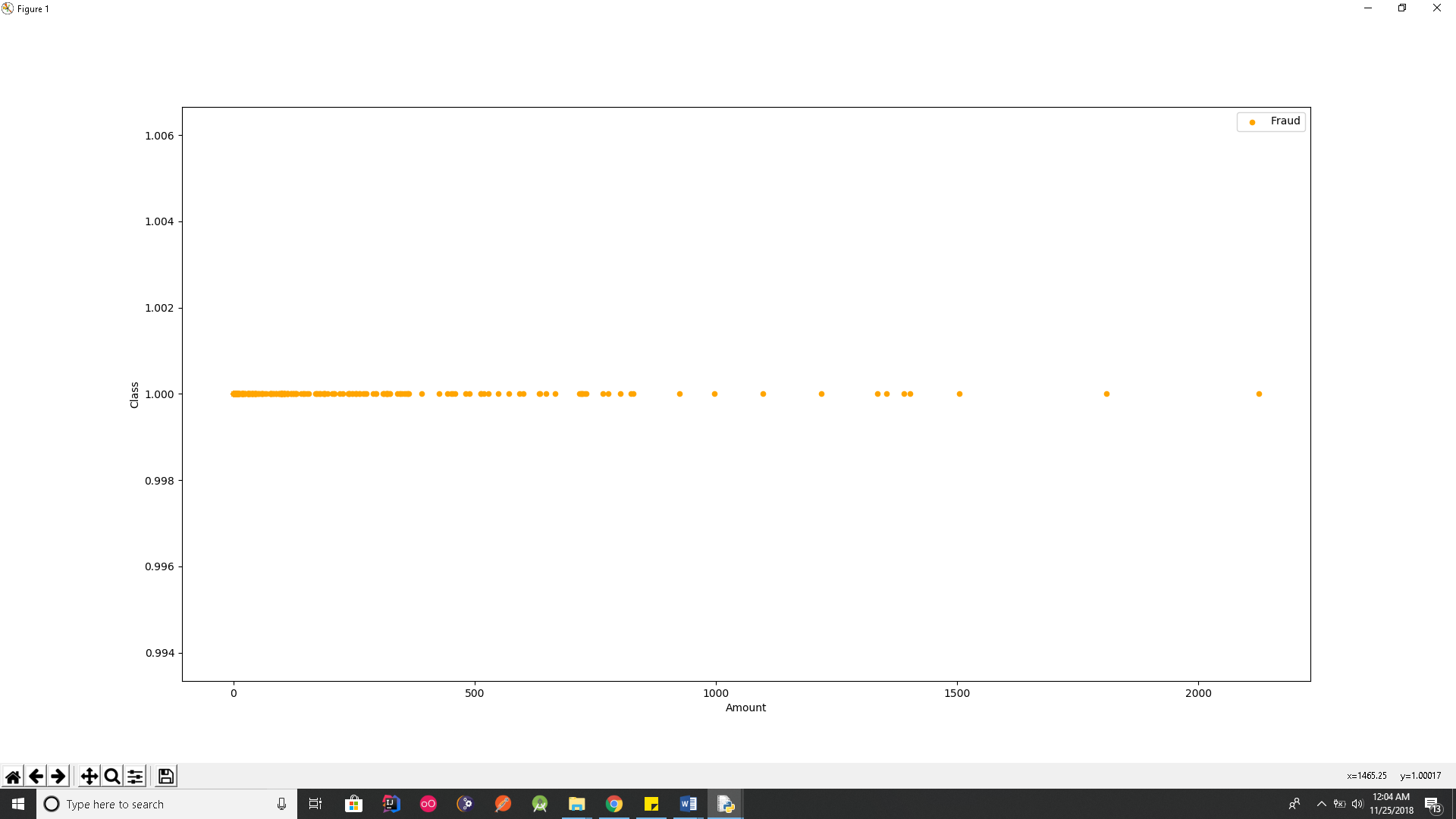
return render(request, 'index.html',data)

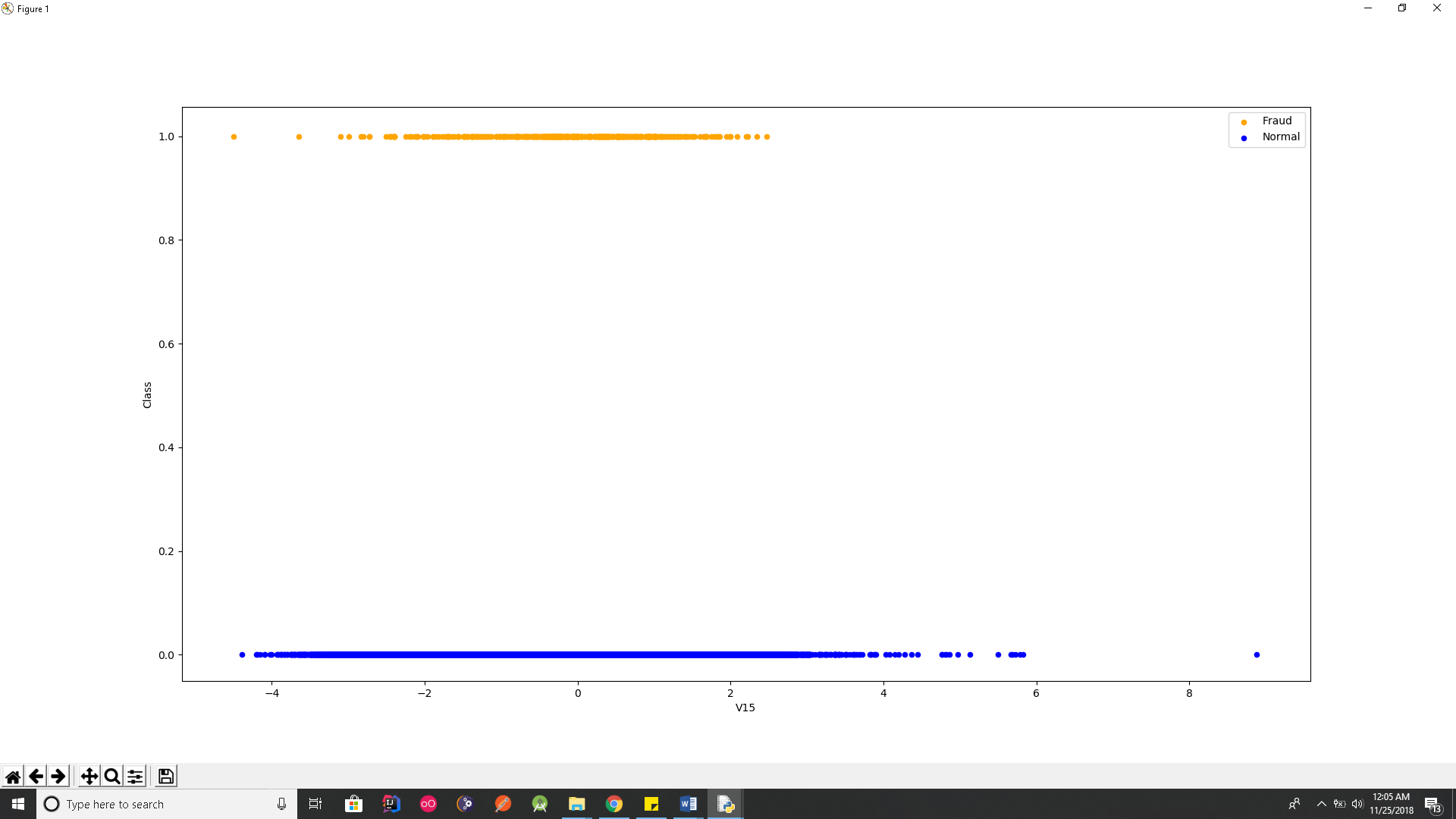
**RUNNING AND OUTPUT SCREENS**

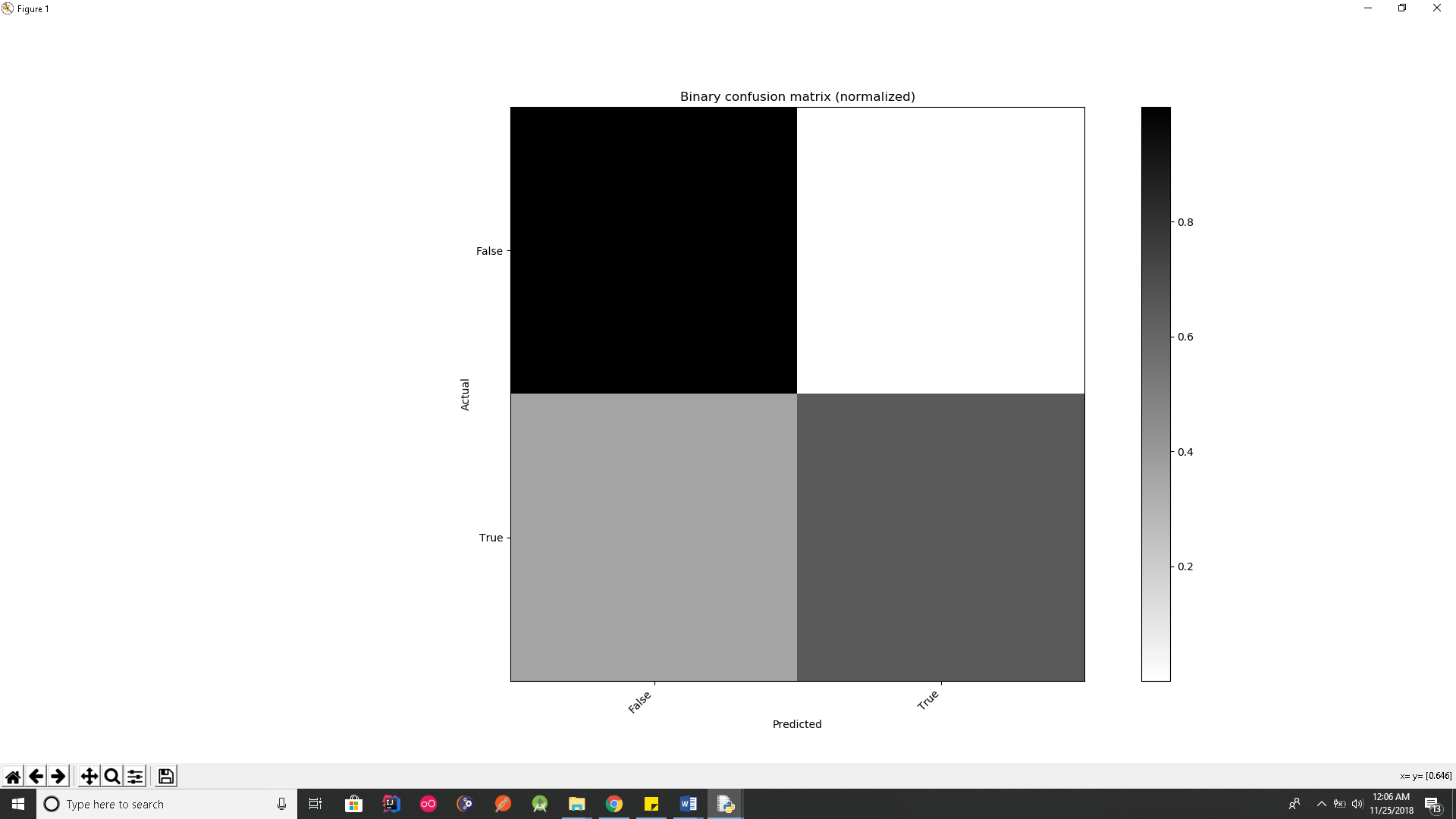


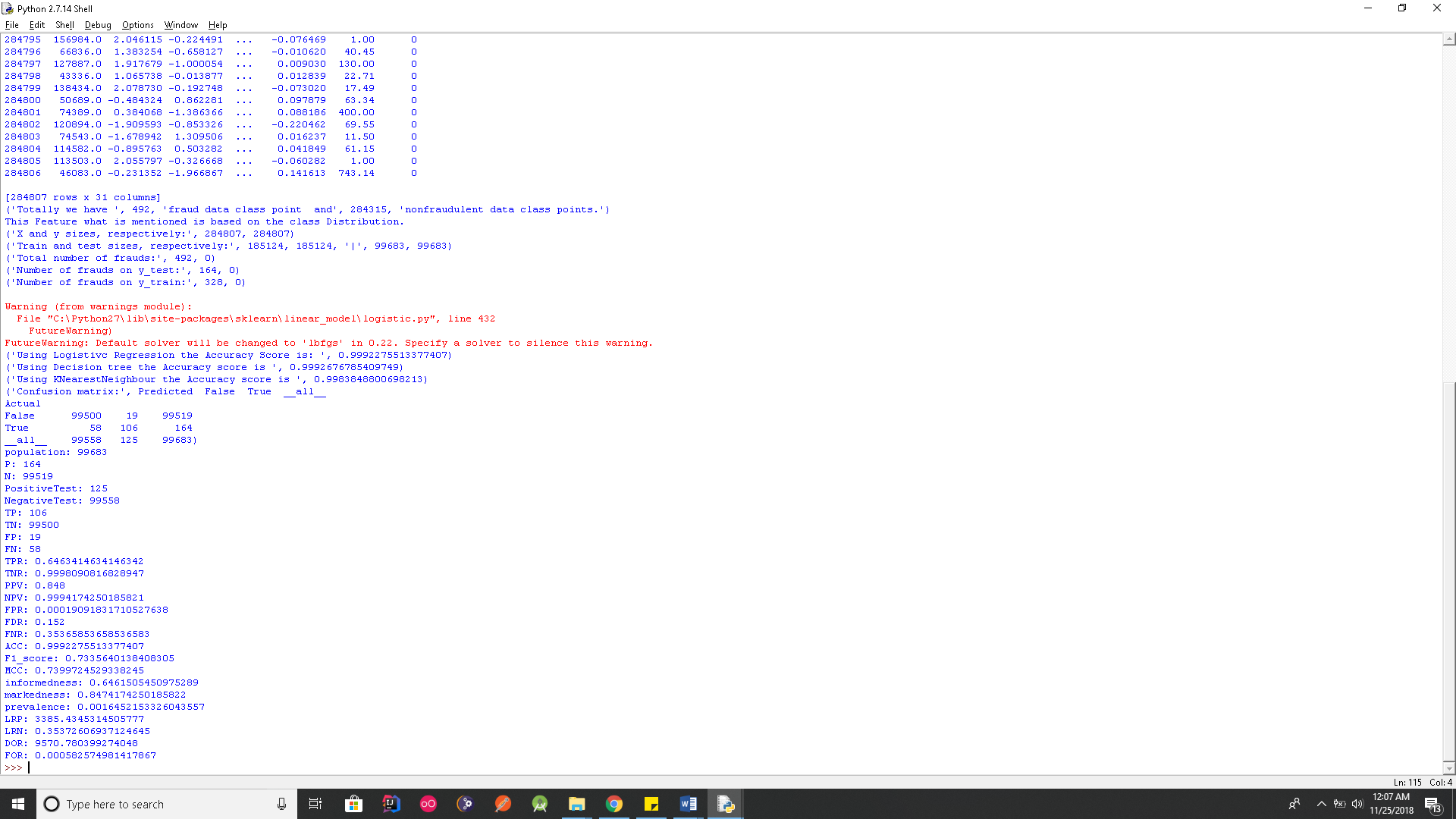


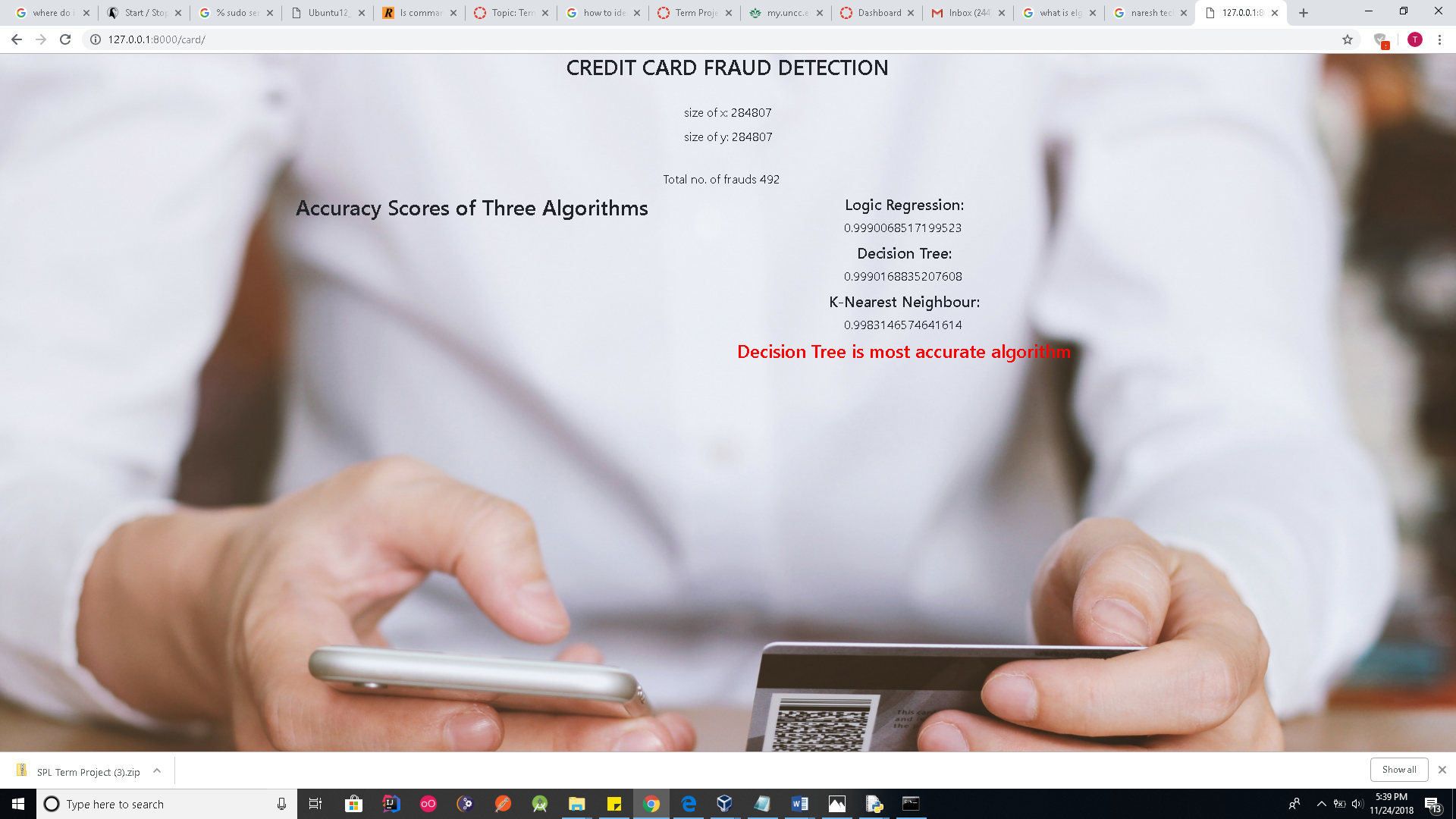












**EVALUATION OF PYTHON**

1. **Readability :**Python is a simple to use programming language.It has incredibly compact language syntax, in that functions and classes tend to take very little vertical space. This makes it much easier to understand functions at a glance. Unlike other programming languages that use brackets and semicolons to divide different parts of the code, python uses indentations to divide different parts of the code where each new block is written under a new indentation level.
2. **Writability:** Python is an easy writing language as it has an ability for the function   
   to return more than one parameter. Its automatic memory management allows you to write code without the worry of overflow or out or stack errors. It also has a very clear flow   
   from top to bottom that allows easy writing of the code.
3. **Reliability:** Python is obviously a highly reliable language.  It, being an easily read language, is better able to be debugged should an error creep into the program.  Python can be run in an interpreter as well as an executable and its use of standardized libraries allow more people to use the same set of code to insure that the program runs correctly. This also makes the language easier to learn.

**STRENGTHS OF PYTHON:**

1. Python is a strong scripting language and very powerful for data analysis
2. It supports multiple systems and platforms and has several frameworks which make the web programming very flexible.
3. Code written in python will be easy to read. Python also has built in list and dictionary data structures that helps in constructing fast run time data structures.
4. Python has a good object-oriented design which provides many processing capabilities and helps in increasing productivity.

**WEAKNESS OF PYTHON:**

1. Python is an interpreted language which is slow compared to Java.
2. Python isn’t good for memory intensive tasks.
3. Python is not a good language for mobile development. This is a weak language for mobile computation.
4. Python has limitations with accessing database. When compared with technologies like JDBC, python’s database access layer is very primitive.

**CONCLUSION**

For the given dataset, we can conclude that the approach we applied using predictive modelling forming linear regression and decision trees proved to be better and provided more accuracy than the previously adopted models of k-means algorithm. We reached an accuracy of almost 100% proving that customer behavioural patterns help in detecting frauds.