IDS Project.
Credit Approval System.

Algorithms Used:

Logistic Regression

Naïve Bayes

Decision Trees

Random Forest

K Nearest Neighbour

Ensembled Algorithm

By Team

F4LS3 P0S1T1V3

Mohit Aggarwal(18UCS124)

Swarnim Sharma (18UCS126)

Ajitesh Saxena(18UCS137)

Siddhant Bhatnagar(18UCS138)

Project Objective:

This machine learning project focuses on credit approval i.e. the process an individual must go through to become eligible for a loan or to pay for goods and services over an extended period. The objective of this project is to reflect the impact of the various attributes on the credit approval process and to compare various ML classification algorithms to find the optimal algorithm.

Introduction to the Dataset:

To protect the confidentiality of the data the names of all the attributes and values in this dataset have been changed to meaningless symbols. This dataset consists of various attributes - continuous, nominal with larger numbers of values, and nominal with small numbers of values. Certain missing values are also present in this dataset.

The Importance of a Credit Approval System:

Credit Analysis determines the likelihood of a third-party to pay back the loan to the bank on time. Credit approval mostly depends on the financial and personal background of the applicant. There are various factors involved in credit approval like gender, age, status, employment, credit history and various other attributes all play an important role in the approval decision. Credit Analysis reduces the financial or other risks which are responsible for the loss in the transaction.

It is necessary to manage the credit risk otherwise it can create adverse effects on credit management. Therefore, before making any decision, analysis of credit approval is very important.

Source of this dataset:

https://archive.ics.uci.edu/ml/datasets/Credit+Approval

Dataset Description:

1. Data Set Characteristics: Multivariate
2. Number of Instances: 690
3. Area: Financial
4. Attribute Characteristics: Categorical, Integer, Real
5. Number of Attributes: 15
6. Associated Tasks: Classification
7. Missing values: Yes

Attribute Information

A1: b, a.
A2: continuous.
A3: continuous.
A4: u, y, l, t.
A5: g, p, gg.
A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff.
A7: v, h, bb, j, n, z, dd, ff, o.
A8: continuous.
A9: t, f.
A10: t, f.
A11: continuous.
A12: t, f.
A13: g, p, s.
A14: continuous.
A15: continuous.
A16: +,- (class attribute

Working Strategy

1. First, we import the required libraries. We have used NumPy for working with the array operations. We have used matplotlib.pyplot where matplotlib will be used for visualization and Pyplot is a module of Matplotlib which works like a MATLAB-like interface. Pandas have been used for data analysis.
2. We import the dataset using read_csv() method of pandas library, and we have converted the dataset into the DataFrame format.
3. We have replaced class values + and - with 1 and 0 respectively so it becomes easier to work with.
4. We have replaced missing values '?' with nan (not a number) as it is easily understandable by the machine.
5. We have extracted the information from the dataset to study the data.
6. We count the number of missing values in each attribute.
7. We extract the data types of each attribute.
8. If the data is numeric then we fill the missing values of the attribute with the mean of all the values present in that attribute.
9. If the data is categorical then we fill the missing values of the attribute with the mode of all values present in that attribute.
10. We encode categorical variables with help of get_dummies() method of pandas libraries.
11. We get our final pre-processed dataset.
12. We delete the duplicate columns from the pre-processed dataset which were created during the encoding of categorical variables.
13. We divide the dataset into independent(X) and dependent (y) variable.
14. We divide our data into training set (75%) and testing set (25%).
15. We apply different ML classification algorithms (logistic regression, KNN, naive bayes, decision tree, random forest and calculate the accuracy for each of them.
16. We select the algorithm whose accuracy is greater than or equal to 80% and combine them together.
17. We find the confusion matrix of the combined algorithm.

A Brief overview of concepts used:

Data pre-processing

It is transformation of data by raw format into a format for analyse is called data processing.

How to do data-pre-processing?

a. **Identify and handle missing value:**
   1) remove the data with missing value or drop it
   2) replace the data
   i) if numeric variable then replace with average of all col value or median
   ii) if categorical variable then replace the value with mode (most frequent value)
   iii) basis of function or data distribution (if have additional information of data)

b. **Data normalization (feature scaling):**
   -> uniform the feature value in different range large variation in value of different feature
   -> like age range (0,100) and income range (10000,100000) then income will influence more
   -> model need or require the feature influence equal (normalization)

   1) x = (x(non uniform) - x(min))/x(max) (simple featuring)
   2) x = (x(non uniform) - x(min))/(x(max) - x(min)) (min-max method)
   3) x = (x(preview) - mean)/(standard deviation) (z score)

c. **Turning categorical value in numerical value:** It can be done with help of pd.get_dummies(df[col])

Data visualization

It is the transformation of data into a graphical and a more easy to understand form.

Why create data visualization??

i) exploratory data analysis

ii) clearer communication of data

iii) More attractive and effective

Seaborn library:

It is an interface that draw a high level attractive statistical graphics like plots, and it is a library based on MATLAB plotting library. all the graphs in the project for counting frequency of categorical variable or analysing distribution of continuous (numerical variable) variable have been done using sea born library.

Classification Algorithms used:

1) Logistic Regression Algorithm:

i) number of examples m and number of attributes n

ii) given X(independent variable) (m*n) and dependent variable y(m*1)

iii) initialize weights matrix W (n*1) and a bias matrix b (m*1) with random values

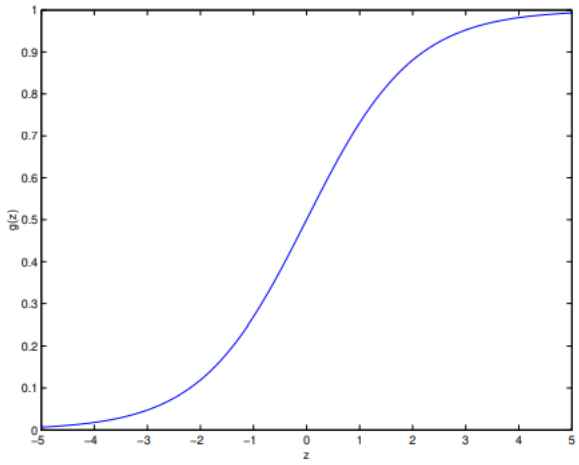
iv) apply y_t = X*W + b and get probability by sigmoid function = p = 1 / (1 + e^-y_t) p(m*1)

v) if probability > threshold value (p = 0.5) then y = 1 , elif y = 0 so p <- (1,0)

vi) compare p and y ; get error e = p*log(y) + (1-p)*(1 - log(y))

vii) update weight W = W - alpha*e

viii) iterate through the algorithm several times



To predict y for a given test example x; apply y = W*X + b

2) K Nearest Neighbour:

if greater than one independent variable predicting y = 0 | y = 1

i) they will make 2 data (one for y = 1 and elif for y = 0)

ii) if there is a new point (where will it go ?) the algo will count the number of neighbour of that point

if more number of neighbour belong to y=1 category then it will also classified to that category else vice versa

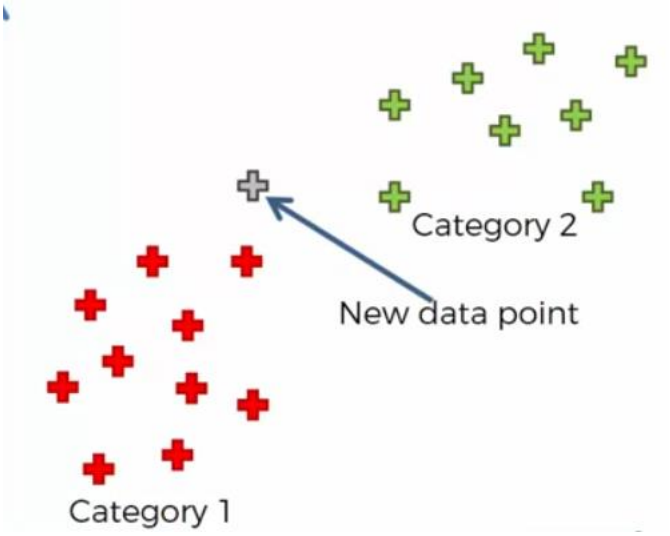
note : how KNN algorithm works ?

i) select k = 5 neighbour

ii) get k nearest neighbour by calculating Euclidean and Manhattan distance or can apply Minkowski distance

iii) among that k nearest count how many point belong to y = 1 and y = 0

iv) assign the new data that category that possess more neighbour



3) Naive Bayes Theorem:

The intuition of Bayes Theorem:

=> P(A|B) = P(B|A) * P(A) / P(B) (probability of a given that b)

0) P(A|B) (posterior)

1) P(A) (prior)

2) P(B) (marginal)

3) P(B|A) (likelihood)

Working of naive bias algorithm:

a) no. of independent variable y = A / total number of training observation

b) p(x) / p(y)

i) draw a circle of particular radius and around the new point

ii) count the number of data in that circle = p(x)

iii) count total number of observation = p(y)

c) p(x) / p(y)

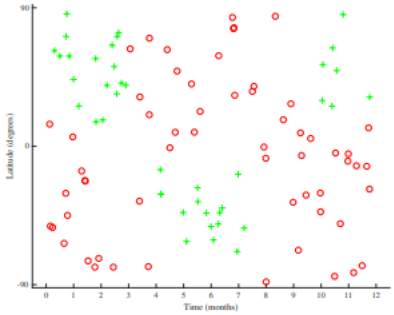
i) count the number of data y = b in that circle = p(x)

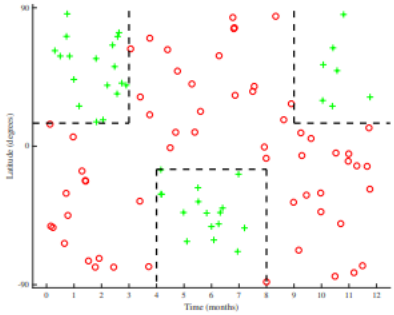
ii) count total number of training observation y = b = p(y)

Note : if independent variable interrelated to each other then it is correct

4) Decision Tree Classifier:

Decision trees are one of the first inherently non-linear machine learning techniques. it splits the graph in many division in the independent variable for y = 0 | 1. it will make a tree of condition and the leaf node will tell the y value. Some of the characteristics of Decision tree algorithm are that it is traditional and less powerful. it uses criteria like entropy, Gini index to select a attribute as a node and to split





5) Random Forest Classifier:

Characteristics of Random Forest classifier

i) it predicts wrong in very rare case

ii) it is very powerful and over-fitted algorithm

iii) it is combination of ensemble and machine learning

How to random forest classifier work?

i) get random k data point by training set

ii) build the decision tree with that k data point

iii) select no. of tree of that kind

iv) all trees will predict y value for new data

v) give y = value (new data) that possess more prediction

PHASE 1 : DATA PRE-PROCESSING

Importing NumPy library for computing mathematical calculations.

Importing MATLAB plotting library for plotting the graphs and data visualization.

Importing pandas library to read and store dataset in DataFrame format.

```
In [1]: import numpy as np
   ...: import matplotlib.pyplot as plt
   ...: import pandas as pd
```

Importing and reading the dataset of credit card approval file in csv format and then storing it in DataFrame format with help of pandas library:

```
In [2]: dataset = pd.read_csv('credit_card_approval.csv')
   ...: df_dataset = pd.DataFrame(dataset)
```

Visualization of credit card approval dataset:

```
    A1     A2      A3 A4 A5  A6  A7  ...  A10 A11 A12  A13    A14    A15 A16
240  b  20.50  10.000  y  p   c   v  ...    f   0   f    s  00040      0   +
241  b  48.25  25.085  u  g   w   v  ...    t   3   f    g  00120     14   +
242  b  28.33   5.000  u  g   w   v  ...    f   0   t    g  00070      0   +
243  a  18.75   7.500  u  g   q   v  ...    t   5   f    g      ?  26726   +
244  b  18.50   2.000  u  g   i   v  ...    t   2   f    g  00120    300   +
245  b  33.17   3.040  y  p   c   h  ...    t   1   t    g  00180  18027   +
246  b  45.00   8.500  u  g  cc   h  ...    t   1   t    g  00088   2000   +
247  a  19.67   0.210  u  g   q   h  ...    t  11   f    g  00080     99   +
248  ?  24.50  12.750  u  g   c  bb  ...    t   2   f    g  00073    444   +
249  b  21.83  11.000  u  g   x   v  ...    t   6   f    g  00121      0   +
250  b  40.25  21.500  u  g   e   z  ...    t  11   f    g  00000   1200   +
251  b  41.42   5.000  u  g   q   h  ...    t   6   t    g  00470      0   +
252  a  17.83  11.000  u  g   x   h  ...    t  11   f    g  00000   3000   +
253  b  23.17  11.125  u  g   x   h  ...    t   1   f    g  00100      0   +
254  b      ?   0.625  u  g   k   v  ...    f   0   f    g  00380   2010   -
255  b  18.17  10.250  u  g   c   h  ...    f   0   f    g  00320     13   -
256  b  20.00  11.045  u  g   c   v  ...    f   0   t    g  00136      0   -
257  b  20.00   0.000  u  g   d   v  ...    f   0   f    g  00144      0   -
258  a  20.75   9.540  u  g   i   v  ...    f   0   f    g  00200   1000   -
259  a  24.50   1.750  y  p   c   v  ...    f   0   f    g  00132      0   -
260  b  32.75   2.335  u  g   d   h  ...    f   0   t    g  00292      0   -
261  a  52.17   0.000  y  p  ff  ff  ...    f   0   f    g  00000      0   -
262  a  48.17   1.335  u  g   i   o  ...    f   0   f    g  00000    120   -
263  a  20.42  10.500  y  p   x   h  ...    f   0   t    g  00154     32   -
264  b  50.75   0.585  u  g  ff  ff  ...    f   0   f    g  00145      0   -
265  b  17.08   0.085  y  p   c   v  ...    f   0   f    g  00140    722   -
266  b  18.33   1.210  y  p   e  dd  ...    f   0   f    g  00100      0   -
267  a  32.00   6.000  u  g   d   v  ...    f   0   f    g  00272      0   -
268  b  59.67   1.540  u  g   q   v  ...    f   0   t    g  00260      0   +
269  b  18.00   0.165  u  g   q   n  ...    f   0   f    g  00200     40   +
270  b  37.58   0.000  ?  ?   ?   ?  ...    f   0   f    p      ?      0   +
271  b  32.33   2.500  u  g   c   v  ...    f   0   t    g  00280      0   -
272  b  18.08   6.750  y  p   m   v  ...    f   0   f    g  00140      0   -
273  b  38.25  10.125  y  p   k   v  ...    f   0   f    g  00160      0   -
274  b  30.67   2.500  u  g  cc   h  ...    f   0   t    s  00340      0   -
275  b  18.58   5.710  u  g   d   v  ...    f   0   f    g  00120      0   -
```

Total Number of Columns: 16 (A1 – A16)

Independent variable (X) attributes: 15 (A1 – A15)

Types of Attributes:

Categorical variables : A1, A4, A5, A6, A7, A9, A10, A12, A13

Numerical variables : A2, A3, A8, A11, A14, A15

Dependent variable (y): 1 (A16)

'+' : credit card approval successful (It represents successful approval of credit card)

'-' : credit card approval denied (it represents denial of credit card)

Transforming the dependent variable (y) (A16) for better analysis of data:

'+' class maps to '1'

'-' class maps to '0'

```
In [4]: for i in range(0, len(df_dataset)):
   ...:     if df_dataset['A16'][i] is '+':
   ...:         df_dataset['A16'][i] = 1
   ...:     else:
   ...:         df_dataset['A16'][i] = 0
```

	A1	A2	A3	A4	A5	A6	A7	...	A10	A11	A12	A13	A14	A15	A16
240	b	20.50	10.000	y	p	c	v	...	f	0	f	s	00040	0	1
241	b	48.25	25.085	u	g	w	v	...	t	3	f	g	00120	14	1
242	b	28.33	5.000	u	g	w	v	...	f	0	t	g	00070	0	1
243	a	18.75	7.500	u	g	q	v	...	t	5	f	g	?	26726	1
244	b	18.50	2.000	u	g	i	v	...	t	2	f	g	00120	300	1
245	b	33.17	3.040	y	p	c	h	...	t	1	t	g	00180	18027	1
246	b	45.00	8.500	u	g	cc	h	...	t	1	t	g	00088	2000	1
247	a	19.67	0.210	u	g	q	h	...	t	11	f	g	00080	99	1
248	?	24.50	12.750	u	g	c	bb	...	t	2	f	g	00073	444	1
249	b	21.83	11.000	u	g	x	v	...	t	6	f	g	00121	0	1
250	b	40.25	21.500	u	g	e	z	...	t	11	f	g	00000	1200	1
251	b	41.42	5.000	u	g	q	h	...	t	6	t	g	00470	0	1
252	a	17.83	11.000	u	g	x	h	...	t	11	f	g	00000	3000	1
253	b	23.17	11.125	u	g	x	h	...	t	1	f	g	00100	0	1
254	b	?	0.625	u	g	k	v	...	f	0	f	g	00380	2010	0
255	b	18.17	10.250	u	g	c	h	...	f	0	f	g	00320	13	0
256	b	20.00	11.045	u	g	c	v	...	f	0	t	g	00136	0	0
257	b	20.00	0.000	u	g	d	v	...	f	0	f	g	00144	0	0
258	a	20.75	9.540	u	g	i	v	...	f	0	f	g	00200	1000	0
259	a	24.50	1.750	y	p	c	v	...	f	0	f	g	00132	0	0
260	b	32.75	2.335	u	g	d	h	...	f	0	t	g	00292	0	0
261	a	52.17	0.000	y	p	ff	ff	...	f	0	f	g	00000	0	0
262	a	48.17	1.335	u	g	i	o	...	f	0	f	g	00000	120	0
263	a	20.42	10.500	y	p	x	h	...	f	0	t	g	00154	32	0
264	b	50.75	0.585	u	g	ff	ff	...	f	0	f	g	00145	0	0
265	b	17.08	0.085	y	p	c	v	...	f	0	f	g	00140	722	0
266	b	18.33	1.210	y	p	e	dd	...	f	0	f	g	00100	0	0
267	a	32.00	6.000	u	g	d	v	...	f	0	f	g	00272	0	0
268	b	59.67	1.540	u	g	q	v	...	f	0	t	g	00260	0	1
269	b	18.00	0.165	u	g	q	n	...	f	0	f	g	00200	40	1
270	b	37.58	0.000	?	?	?	?	...	f	0	f	p	?	0	1
271	b	32.33	2.500	u	g	c	v	...	f	0	t	g	00280	0	0
272	b	18.08	6.750	y	p	m	v	...	f	0	f	g	00140	0	0
273	b	38.25	10.125	y	p	k	v	...	f	0	f	g	00160	0	0
274	b	30.67	2.500	u	g	cc	h	...	f	0	t	s	00340	0	0
275	b	18.58	5.710	u	g	d	v	...	f	0	f	g	00120	0	0

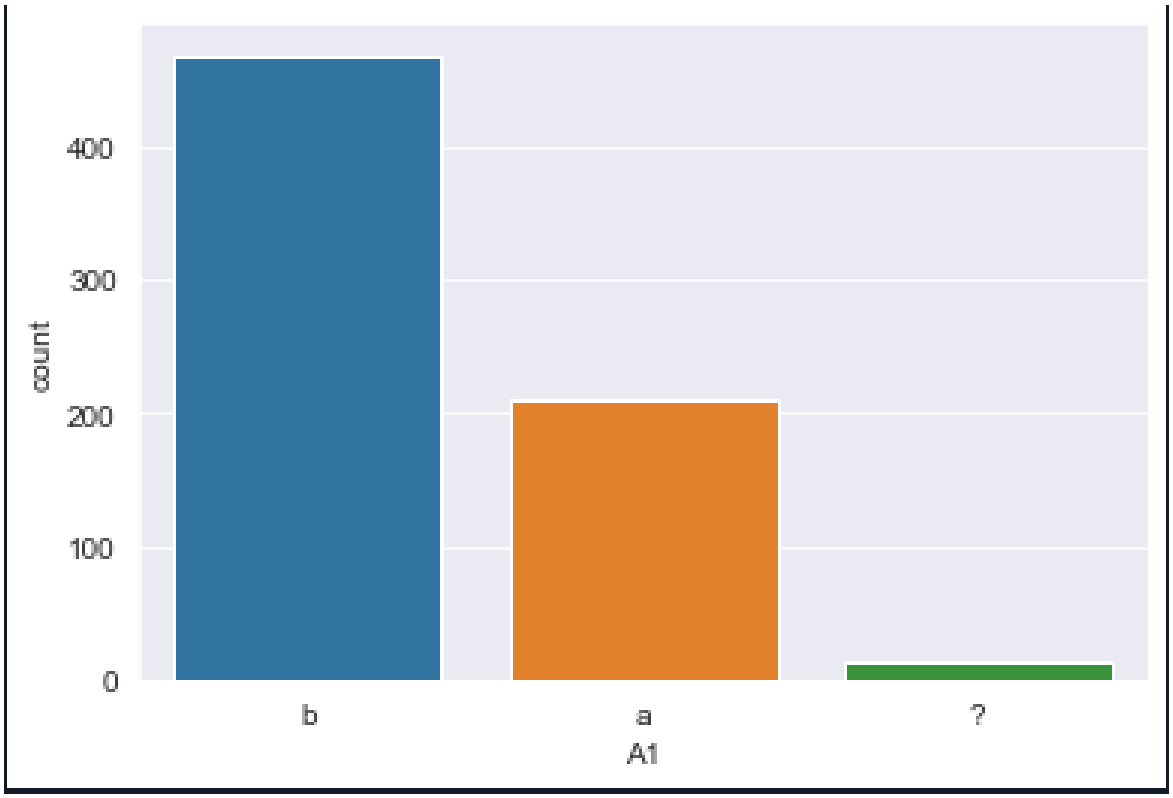
Now, In the dependent variable (A16) :

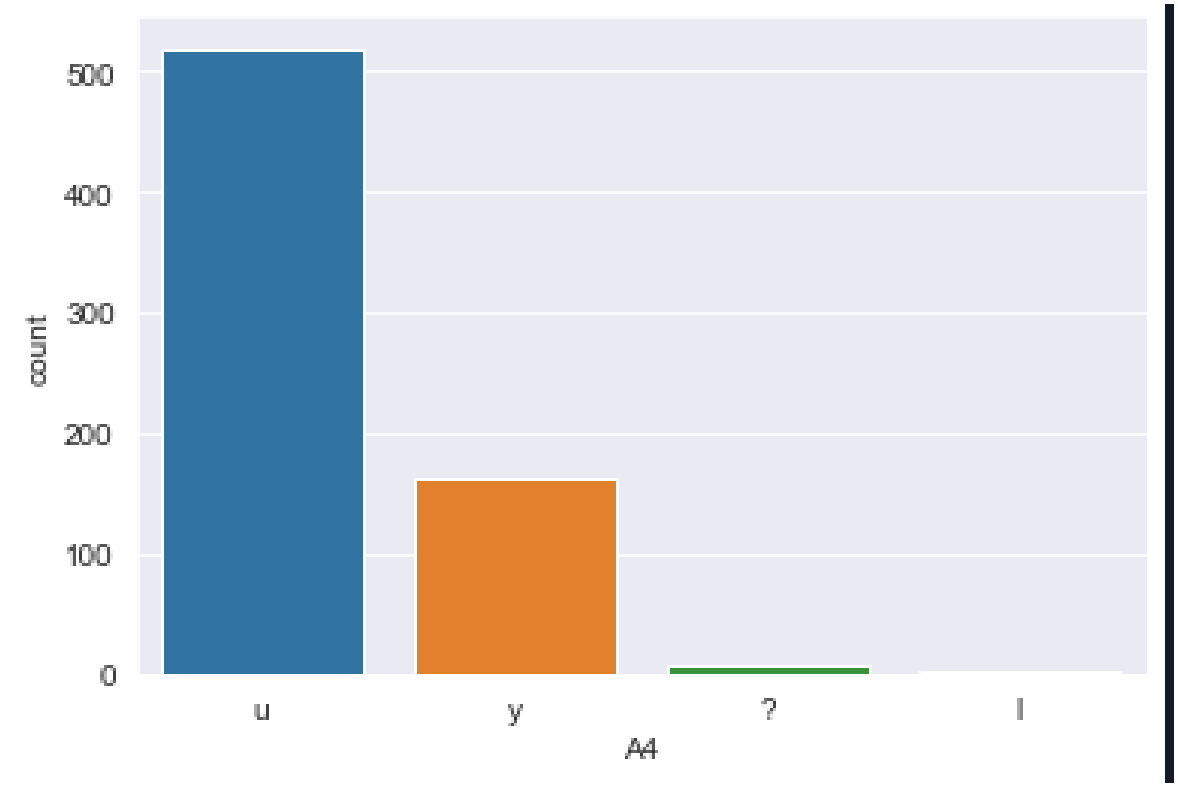
A set of binary number represents the denial and approval of credit card.

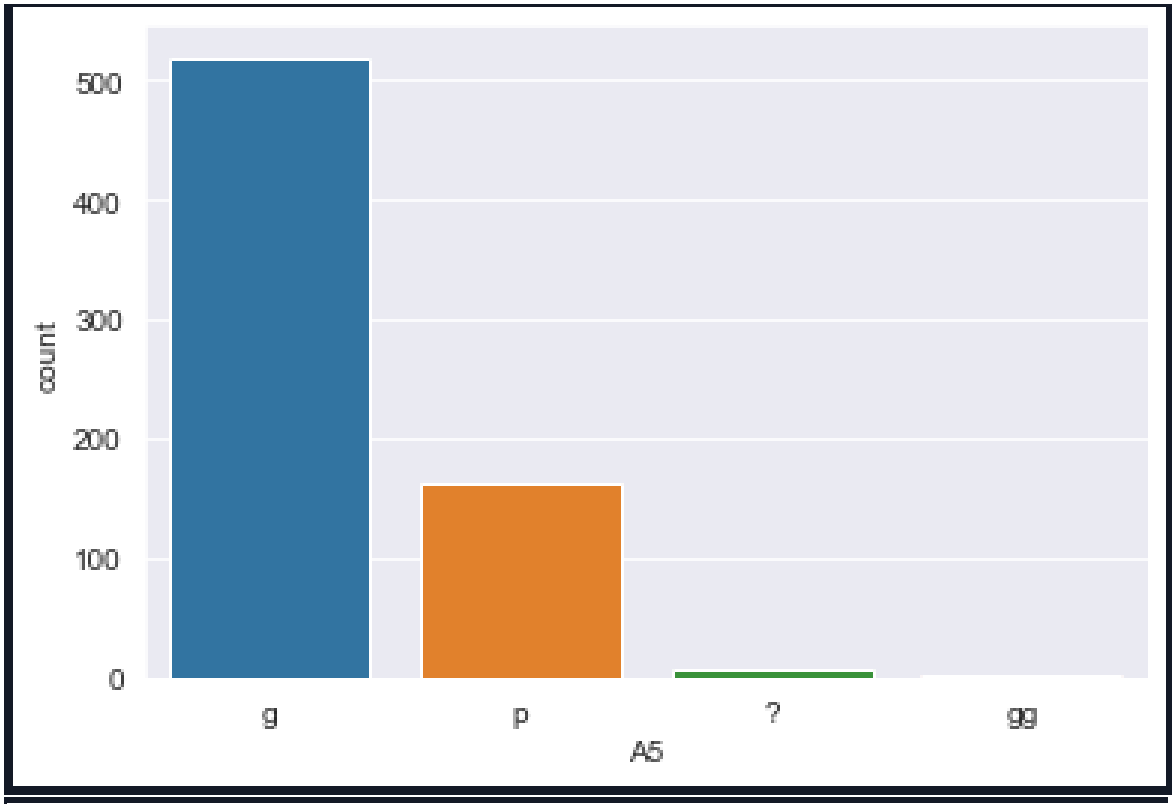
'1' : credit card approval successful (It represents successful approval of credit card)

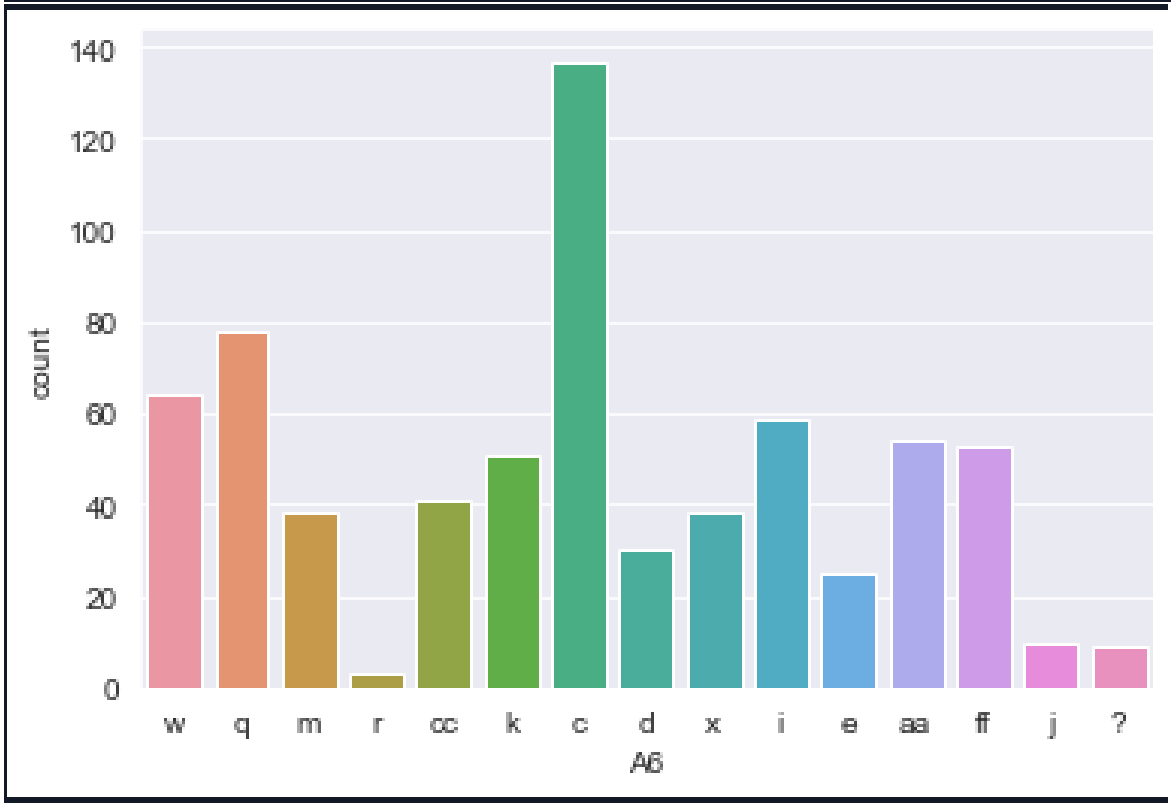
'0' : credit card approval denied (it represents denial of credit card)

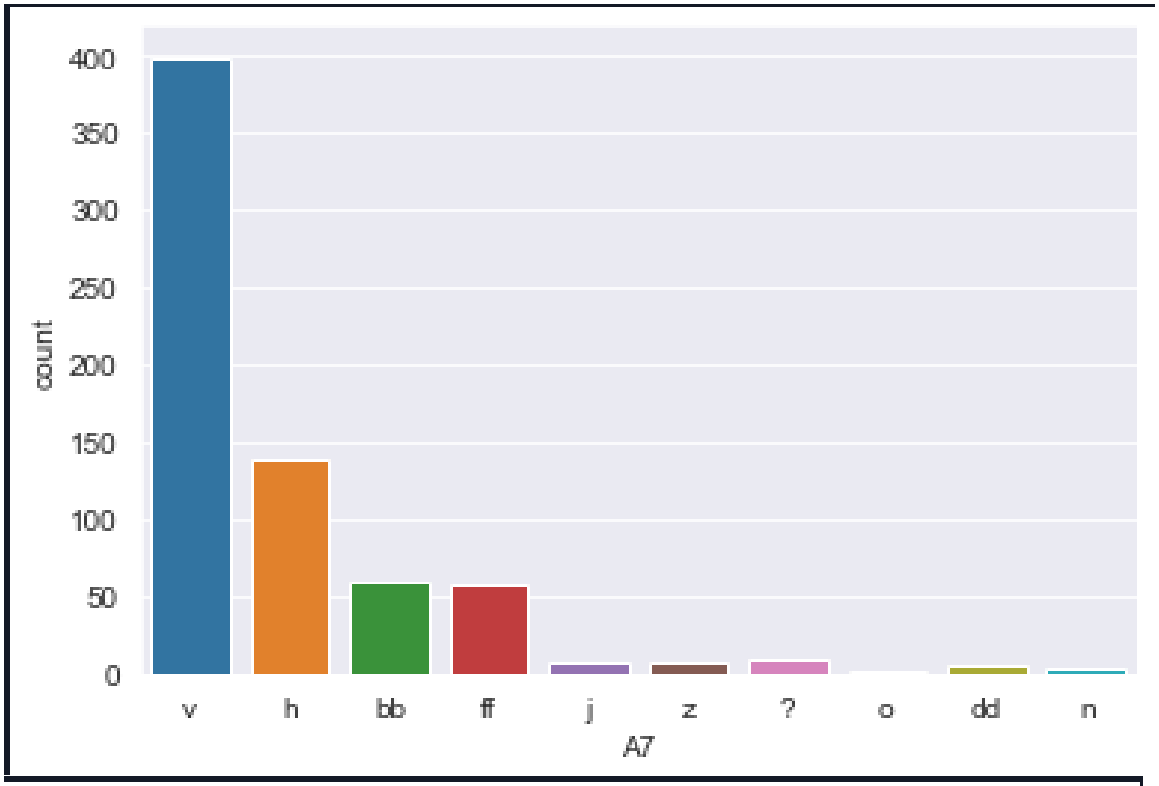
Visualization of Categorical variable data: A1, A4, A5, A6, A7, A9, A10, A12, A13:

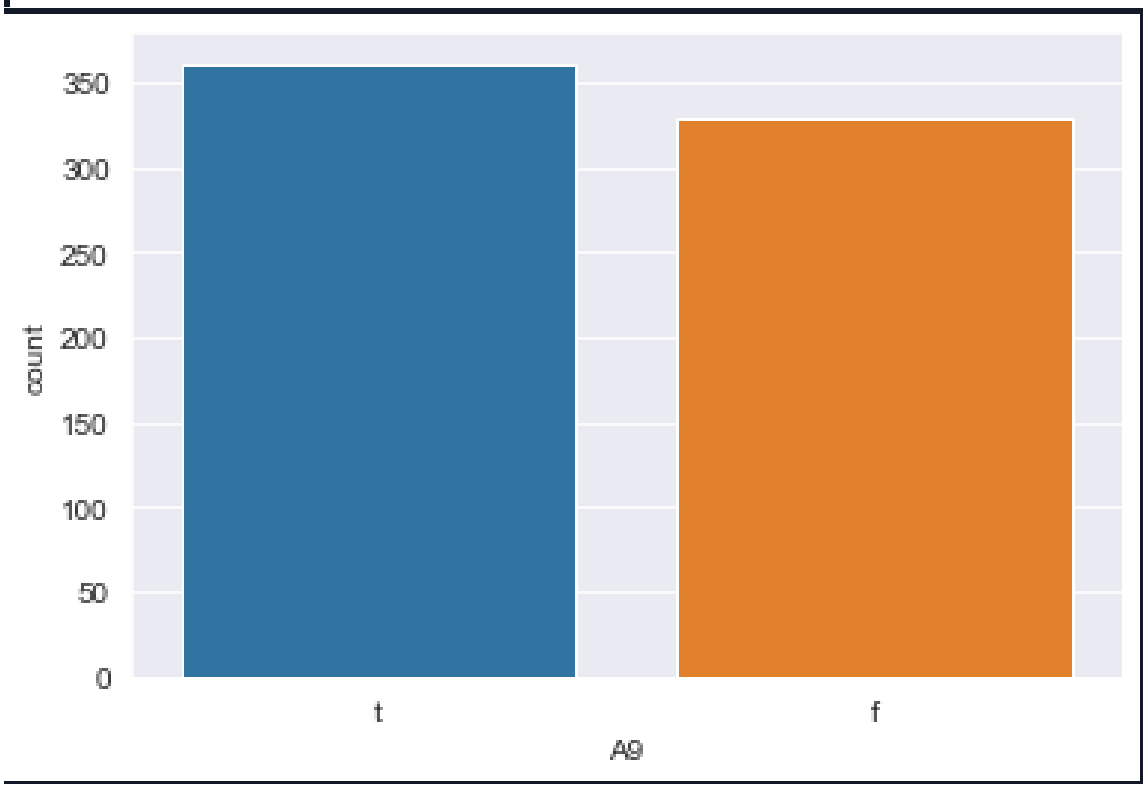


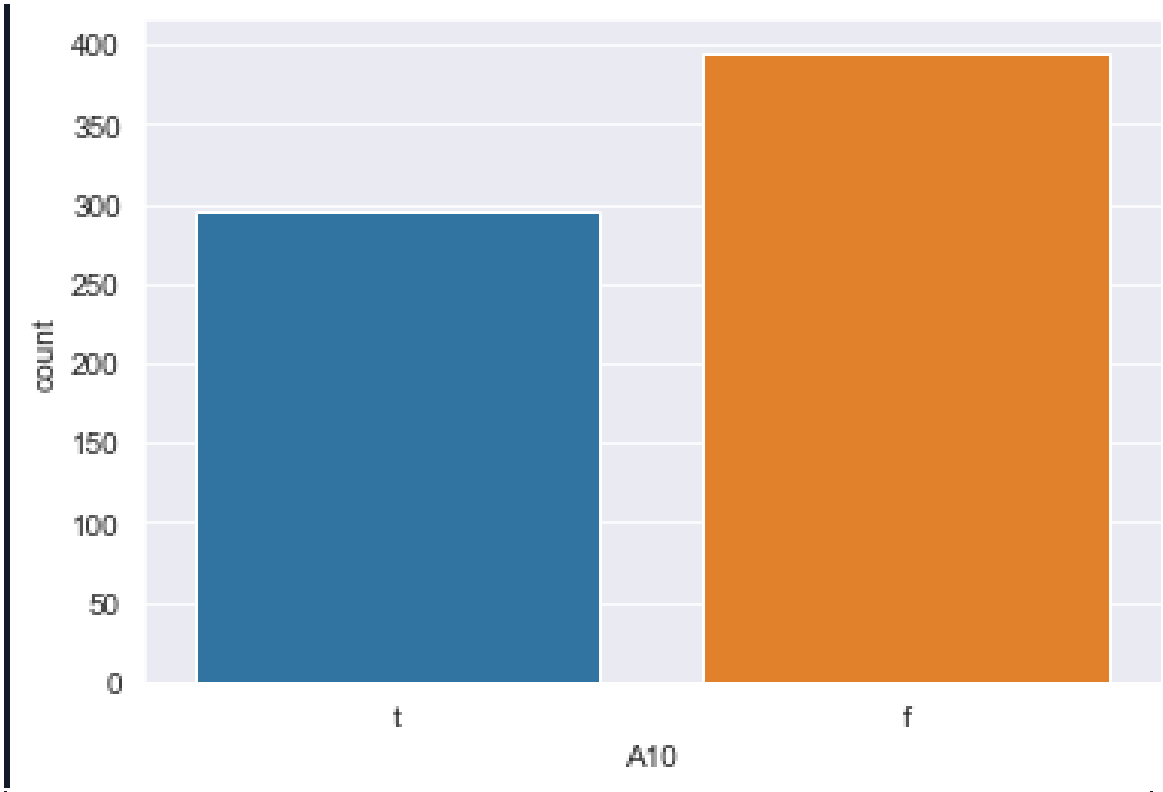


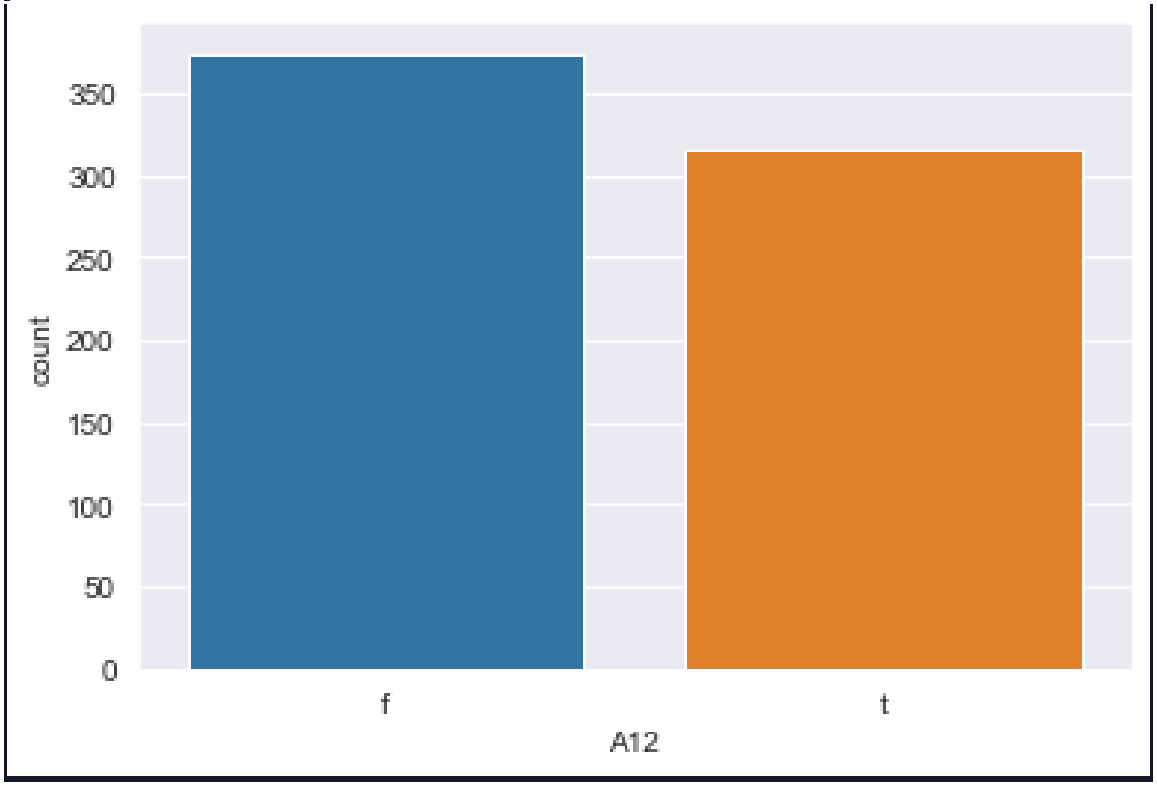

500
400
300
200
100
0
count
g
p
?
gg
A5

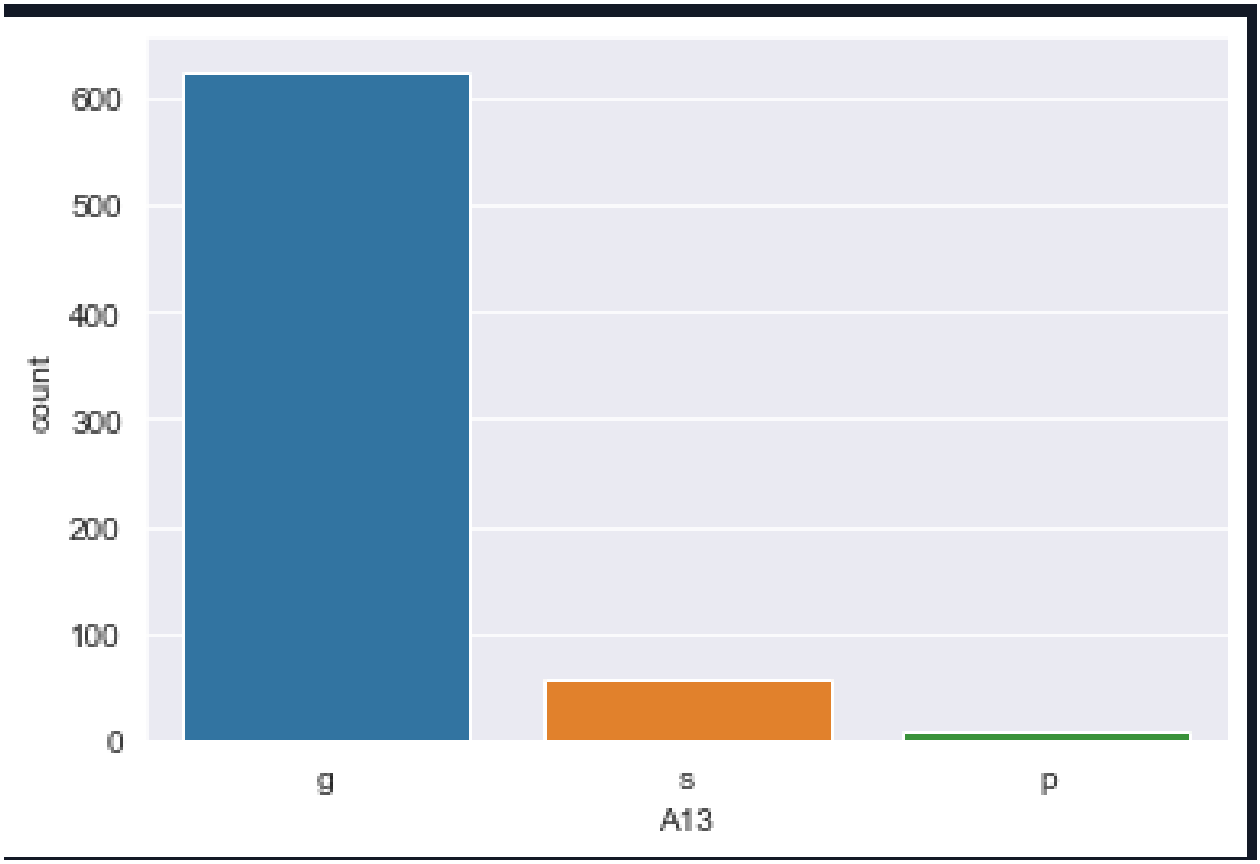

140
120
100
80
60
40
20
0
count
w
q
m
r
cc
k
c
d
x
i
e
aa
ff
j
?
A6


400
350
300
250
200
150
100
50
0
count
v
h
bb
ff
j
z
?
o
dd
n
A7


350
300
250
200
150
100
50
0
count
t
f
A9


400
350
300
250
200
150
100
50
0
count
t
f
A10


350
300
250
200
150
100
50
0
count
f
t
A12


600
500
400
300
200
100
0
count
g
s
p
A13

Transforming the representation of missing values of dataset for better analysis of data:

Replacing '**?**' to **NaN** (Not a Number) with help of **NumPy** library

```
In [5]: for fields in df_dataset.columns:
   ...:     for i in range(0, len(df_dataset)):
   ...:         if df_dataset[fields][i] is '?':
   ...:             df_dataset[fields][i] = np.nan
   ...:         else:
   ...:             continue
```

	A1	A2	A3	A4	A5	A6	A7	...	A10	A11	A12	A13	A14	A15	A16
240	b	20.50	10.000	y	p	c	v	...	f	0	f	s	00040	0	1
241	b	48.25	25.085	u	g	w	v	...	t	3	f	g	00120	14	1
242	b	28.33	5.000	u	g	w	v	...	f	0	t	g	00070	0	1
243	a	18.75	7.500	u	g	q	v	...	t	5	f	g	NaN	26726	1
244	b	18.50	2.000	u	g	i	v	...	t	2	f	g	00120	300	1
245	b	33.17	3.040	y	p	c	h	...	t	1	t	g	00180	18027	1
246	b	45.00	8.500	u	g	cc	h	...	t	1	t	g	00088	2000	1
247	a	19.67	0.210	u	g	q	h	...	t	11	f	g	00080	99	1
248	NaN	24.50	12.750	u	g	c	bb	...	t	2	f	g	00073	444	1
249	b	21.83	11.000	u	g	x	v	...	t	6	f	g	00121	0	1
250	b	40.25	21.500	u	g	e	z	...	t	11	f	g	00000	1200	1
251	b	41.42	5.000	u	g	q	h	...	t	6	t	g	00470	0	1
252	a	17.83	11.000	u	g	x	h	...	t	11	f	g	00000	3000	1
253	b	23.17	11.125	u	g	x	h	...	t	1	f	g	00100	0	1
254	b	NaN	0.625	u	g	k	v	...	f	0	f	g	00380	2010	0
255	b	18.17	10.250	u	g	c	h	...	f	0	f	g	00320	13	0
256	b	20.00	11.045	u	g	c	v	...	f	0	t	g	00136	0	0
257	b	20.00	0.000	u	g	d	v	...	f	0	f	g	00144	0	0
258	a	20.75	9.540	u	g	i	v	...	f	0	f	g	00200	1000	0
259	a	24.50	1.750	y	p	c	v	...	f	0	f	g	00132	0	0
260	b	32.75	2.335	u	g	d	h	...	f	0	t	g	00292	0	0
261	a	52.17	0.000	y	p	ff	ff	...	f	0	f	g	00000	0	0
262	a	48.17	1.335	u	g	i	o	...	f	0	f	g	00000	120	0
263	a	20.42	10.500	y	p	x	h	...	f	0	t	g	00154	32	0
264	b	50.75	0.585	u	g	ff	ff	...	f	0	f	g	00145	0	0
265	b	17.08	0.085	y	p	c	v	...	f	0	f	g	00140	722	0
266	b	18.33	1.210	y	p	e	dd	...	f	0	f	g	00100	0	0
267	a	32.00	6.000	u	g	d	v	...	f	0	f	g	00272	0	0
268	b	59.67	1.540	u	g	q	v	...	f	0	t	g	00260	0	1
269	b	18.00	0.165	u	g	q	n	...	f	0	f	g	00200	40	1
270	b	37.58	0.000	NaN	NaN	NaN	NaN	...	f	0	f	p	NaN	0	1
271	b	32.33	2.500	u	g	c	v	...	f	0	t	g	00280	0	0
272	b	18.08	6.750	y	p	m	v	...	f	0	f	g	00140	0	0
273	b	38.25	10.125	y	p	k	v	...	f	0	f	g	00160	0	0
274	b	30.67	2.500	u	g	cc	h	...	f	0	t	s	00340	0	0

Now, all the missing values in each attribute of dataset have been replaced by NaN (Not a Number). The pre-defined methods can easily count the number of missing values in each attribute and analyse the datatype of each attribute.

1) **dtypes** variable of DataFrame object give information of datatypes of all the attribute. If a numeric variable attribute has missing values, then it will return object datatype instead of integer or float datatype for that attribute irrespective of its actual numeric values.

2) **isnull().sum()** method extract and counts all the **NaN** values of each attribute of the dataset.

```
In [57]: df_dataset.dtypes
Out[57]:
A1       object
A2       object
A3      float64
A4       object
A5       object
A6       object
A7       object
A8      float64
A9       object
A10      object
A11       int64
A12      object
A13      object
A14      object
A15       int64
A16      object
dtype: object
```

```
In [58]: df_dataset.isnull().sum()
Out[58]:
A1     12
A2     12
A3      0
A4      6
A5      6
A6      9
A7      9
A8      0
A9      0
A10     0
A11     0
A12     0
A13     0
A14    13
A15     0
A16     0
dtype: int64
```

Number of missing values in attributes:

A1: 12 (categorical)

A2: 12 (numeric)

A4: 6 (categorical)

A5: 6 (categorical)

A6: 9 (categorical)

A7: 9 (categorical)

A14: 13 (numeric)

Filling the missing values with help of different strategies :

Fill the missing values with help of **fillna()** method applying the appropriate strategy. Then finally transform the datatype of variables with help of **astype()** method.

A) Filling the Numeric Type Variable:

Mean Strategy: Calculating the mean of the numeric attribute and then replacing the missing values of that attribute by mean value of all the values present in that attribute. It can also be replaced by median value by applying median strategy. The mean strategy will be applied to numeric attribute like A2, A14.

B) Filling the Categorical Type Variable:

Mode Strategy: Identifying and Counting all the values of attribute and then replacing the most occurring value of that attribute by missing value. It will be applied to categorical variable like A1, A4, A5, A6, A7.

```
In [6]: df_dataset['A1'] = df_dataset['A1'].fillna(df_dataset['A1'].mode()[0])
   ...:
   ...: df_dataset['A2'] = df_dataset['A2'].fillna(df_dataset['A2'].astype(float).mean())
   ...: df_dataset['A2'] = df_dataset['A2'].astype(float)
   ...:
   ...:
   ...: df_dataset['A4'] = df_dataset['A4'].fillna(df_dataset['A4'].mode()[0])
   ...:
   ...: df_dataset['A5'] = df_dataset['A5'].fillna(df_dataset['A5'].mode()[0])
   ...:
   ...: df_dataset['A6'] = df_dataset['A6'].fillna(df_dataset['A6'].mode()[0])
   ...:
   ...: df_dataset['A7'] = df_dataset['A7'].fillna(df_dataset['A7'].mode()[0])
   ...:
   ...:
   ...: df_dataset['A14'] = df_dataset['A14'].fillna(df_dataset['A14'].astype(float).mean())
   ...: df_dataset['A14'] = df_dataset['A14'].astype(float)
```

	A1	A2	A3	A4	A5	A6	A7	...	A10	A11	A12	A13	A14	A15	A16
240	b	20.500000	10.000	y	p	c	v	...	f	0	f	s	40.000000	0	1
241	b	48.250000	25.085	u	g	w	v	...	t	3	f	g	120.000000	14	1
242	b	28.330000	5.000	u	g	w	v	...	f	0	t	g	70.000000	0	1
243	a	18.750000	7.500	u	g	q	v	...	t	5	f	g	184.014771	26726	1
244	b	18.500000	2.000	u	g	i	v	...	t	2	f	g	120.000000	300	1
245	b	33.170000	3.040	y	p	c	h	...	t	1	t	g	180.000000	18027	1
246	b	45.000000	8.500	u	g	cc	h	...	t	1	t	g	88.000000	2000	1
247	a	19.670000	0.210	u	g	q	h	...	t	11	f	g	80.000000	99	1
248	b	24.500000	12.750	u	g	c	bb	...	t	2	f	g	73.000000	444	1
249	b	21.830000	11.000	u	g	x	v	...	t	6	f	g	121.000000	0	1
250	b	40.250000	21.500	u	g	e	z	...	t	11	f	g	0.000000	1200	1
251	b	41.420000	5.000	u	g	q	h	...	t	6	t	g	470.000000	0	1
252	a	17.830000	11.000	u	g	x	h	...	t	11	f	g	0.000000	3000	1
253	b	23.170000	11.125	u	g	x	h	...	t	1	f	g	100.000000	0	1
254	b	31.568171	0.625	u	g	k	v	...	f	0	f	g	380.000000	2010	0
255	b	18.170000	10.250	u	g	c	h	...	f	0	f	g	320.000000	13	0
256	b	20.000000	11.045	u	g	c	v	...	f	0	t	g	136.000000	0	0
257	b	20.000000	0.000	u	g	d	v	...	f	0	f	g	144.000000	0	0
258	a	20.750000	9.540	u	g	i	v	...	f	0	f	g	200.000000	1000	0
259	a	24.500000	1.750	y	p	c	v	...	f	0	f	g	132.000000	0	0
260	b	32.750000	2.335	u	g	d	h	...	f	0	t	g	292.000000	0	0
261	a	52.170000	0.000	y	p	ff	ff	...	f	0	f	g	0.000000	0	0
262	a	48.170000	1.335	u	g	i	o	...	f	0	f	g	0.000000	120	0
263	a	20.420000	10.500	y	p	x	h	...	f	0	t	g	154.000000	32	0
264	b	50.750000	0.585	u	g	ff	ff	...	f	0	f	g	145.000000	0	0
265	b	17.080000	0.085	y	p	c	v	...	f	0	f	g	140.000000	722	0
266	b	18.330000	1.210	y	p	e	dd	...	f	0	f	g	100.000000	0	0
267	a	32.000000	6.000	u	g	d	v	...	f	0	f	g	272.000000	0	0
268	b	59.670000	1.540	u	g	q	v	...	f	0	t	g	260.000000	0	1
269	b	18.000000	0.165	u	g	q	n	...	f	0	f	g	200.000000	40	1
270	b	37.580000	0.000	u	g	c	v	...	f	0	f	p	184.014771	0	1
271	b	32.330000	2.500	u	g	c	v	...	f	0	t	g	280.000000	0	0
272	b	18.080000	6.750	y	p	m	v	...	f	0	f	g	140.000000	0	0
273	b	38.250000	10.125	y	p	k	v	...	f	0	f	g	160.000000	0	0
274	b	30.670000	2.500	u	g	cc	h	...	f	0	t	s	340.000000	0	0
275	b	18.580000	5.710	u	g	d	v	...	f	0	f	g	120.000000	0	0

Now, all the missing values have been filled with appropriate strategies. All missing values of a categorical variable have been replaced by mode value of that attribute. All the missing values of a numeric variable have been replaced by calculated mean of that attribute.

```
In [65]: df_dataset.dtypes
Out[65]:
A1       object
A2      float64
A3      float64
A4       object
A5       object
A6       object
A7       object
A8      float64
A9       object
A10      object
A11       int64
A12      object
A13      object
A14     float64
A15       int64
A16      object
dtype: object
```

```
In [63]: df_dataset.isnull().sum()
Out[63]:
A1     0
A2     0
A3     0
A4     0
A5     0
A6     0
A7     0
A8     0
A9     0
A10    0
A11    0
A12    0
A13    0
A14    0
A15    0
A16    0
dtype: int64
```

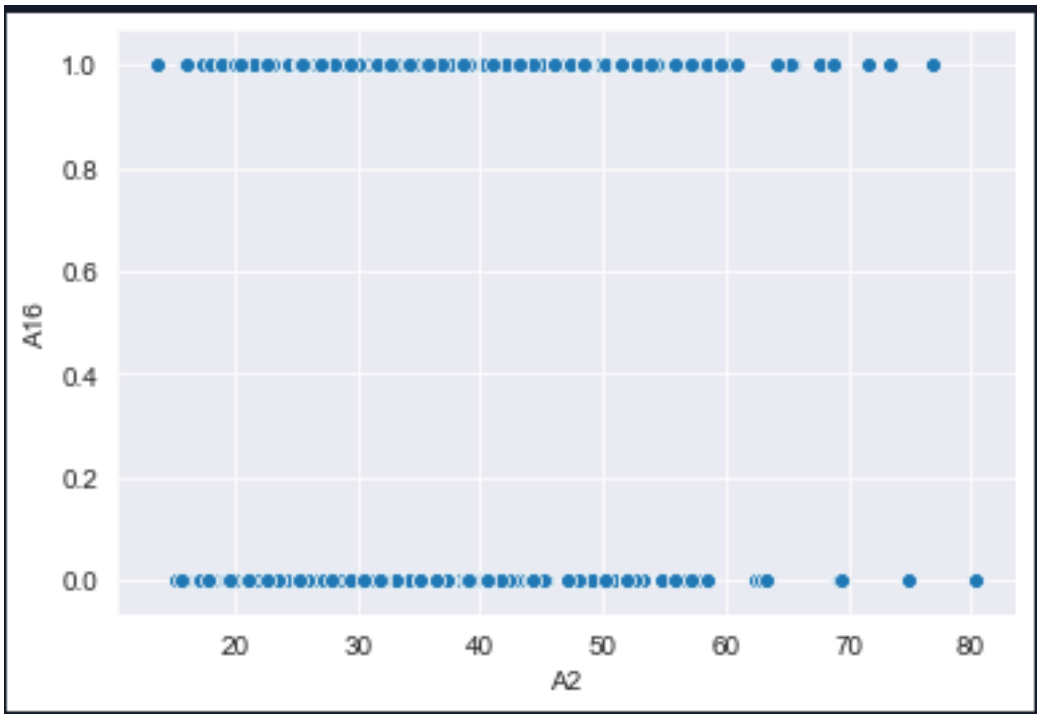
Number of missing values: 0

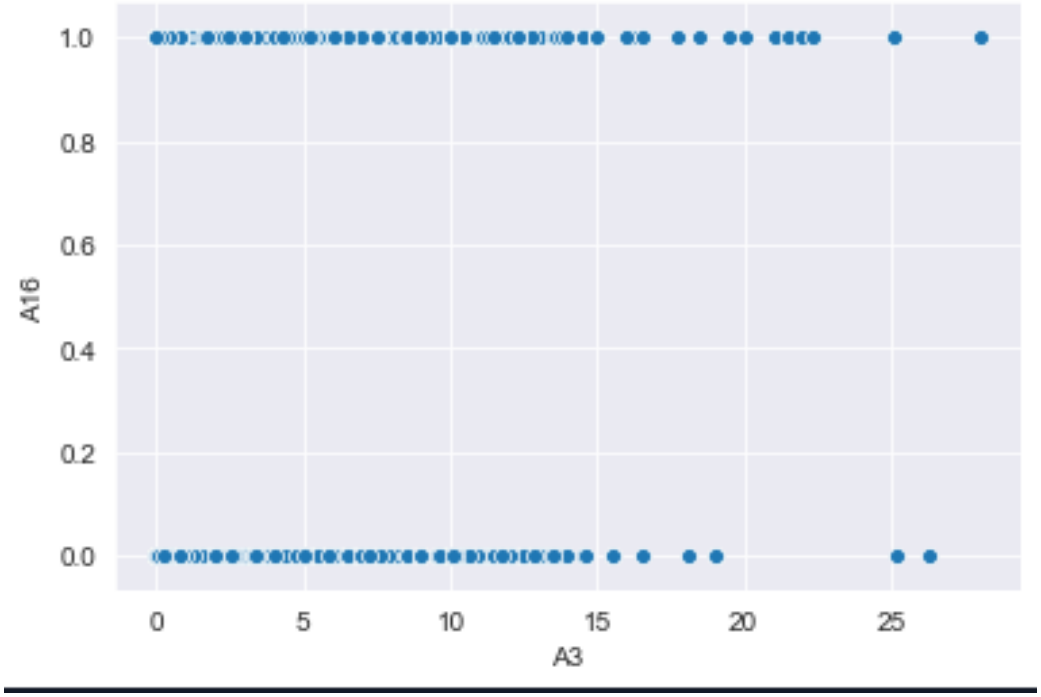
All Numeric variable datatype: integer or float

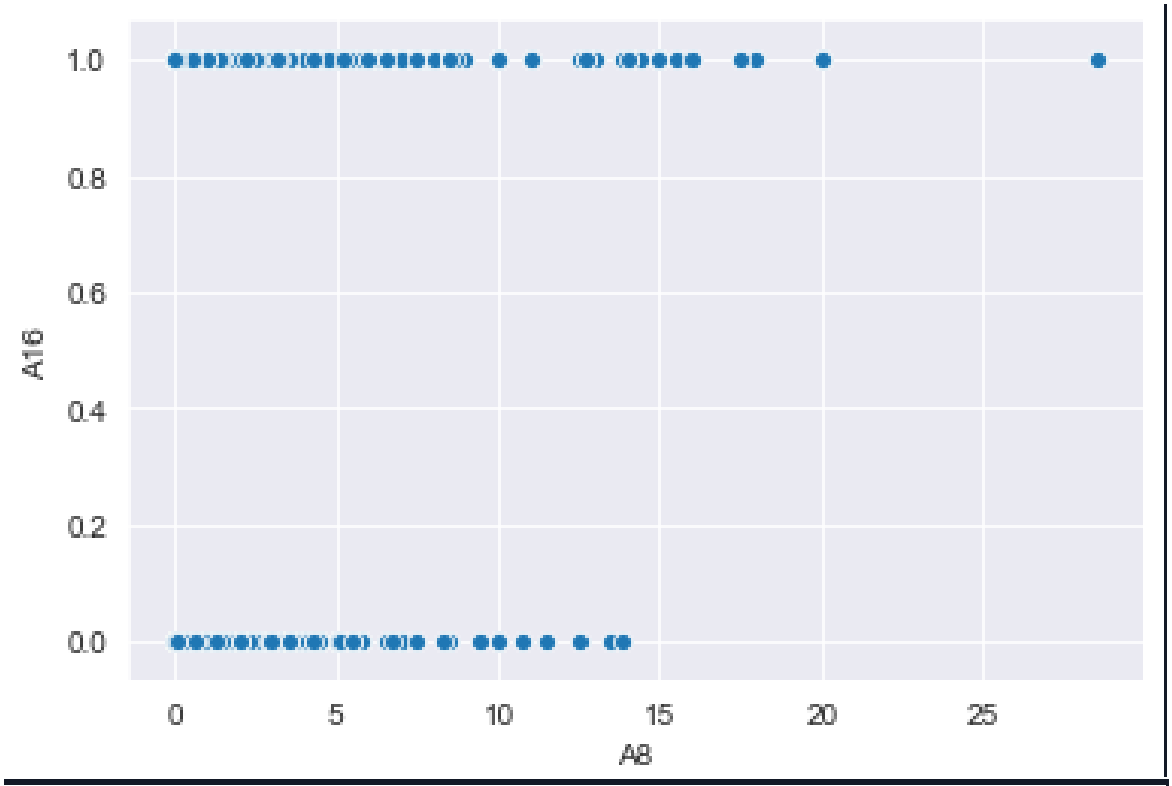
All categorical variable datatype: object

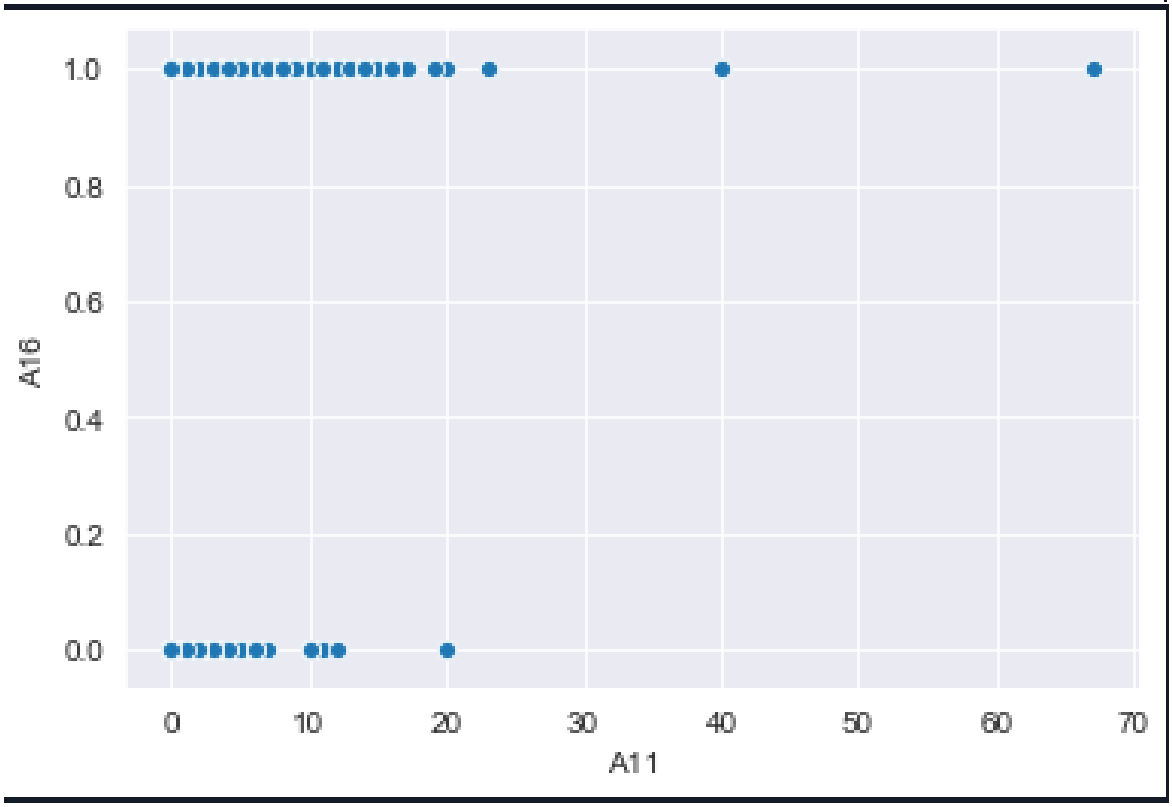
Visualizing the Numerical Variables: A2, A3, A8, A11

It is visualization graph of distribution of continuous numerical variable with respect to dependent variable.






1.0
0.8
0.6
0.4
0.2
0.0
A16
0
5
10
15
20
25
A8


1.0
0.8
0.6
0.4
0.2
0.0
A16
0
10
20
30
40
50
60
70
A11

Encoding of the categorical variable :

The algorithms cannot understand the categorical variable in object or string format. So, categorical variables must be encoded in mathematical format so that algorithm can understand the binary numbers (1,0) and learn to predict the dependent variable.

If there are a,b,c values of a categorical variable. By encoding the categorical variable, it will split in three columns or attribute a,b,c of values 0,1 where '001' represents 'a' , '010' represents 'b' and '001' represents 'c'.

Categorical variables of each attribute will be encoded with the help of **get_dummies()** method of **Pandas** library. A loop will iterate and check and identify if attribute type object then it is categorical variable and encode it.

```
In [7]: def encoding():
   ...:     df_final = df_dataset
   ...:     i=0
   ...:     for fields in df_dataset.columns:
   ...:         print(fields)
   ...:         if df_dataset[fields].dtype == 'O':
   ...:             df_1 = pd.get_dummies(df_dataset[fields] , drop_first = True)
   ...:         else:
   ...:             df_1 = df_dataset[fields].copy()
   ...:         df_dataset.drop([fields] , axis=1 , inplace=True)
   ...:         if i==0:
   ...:             df_final = df_1.copy()
   ...:         else:
   ...:             df_final = pd.concat([df_final,df_1] , axis=1)
   ...:         i = i+1
   ...:     df_final = pd.concat([df_dataset,df_final],axis=1)
   ...:     return df_final
```

```
In [8]: df_data  = encoding()
```

	b	A2	A3	u	y	gg	p	...	A11	t	p	s	A14	A15	1
240	1	20.500000	10.000	0	1	0	1	...	0	0	0	1	40.000000	0	1
241	1	48.250000	25.085	1	0	0	0	...	3	0	0	0	120.000000	14	1
242	1	28.330000	5.000	1	0	0	0	...	0	1	0	0	70.000000	0	1
243	0	18.750000	7.500	1	0	0	0	...	5	0	0	0	184.014771	26726	1
244	1	18.500000	2.000	1	0	0	0	...	2	0	0	0	120.000000	300	1
245	1	33.170000	3.040	0	1	0	1	...	1	1	0	0	180.000000	18027	1
246	1	45.000000	8.500	1	0	0	0	...	1	1	0	0	88.000000	2000	1
247	0	19.670000	0.210	1	0	0	0	...	11	0	0	0	80.000000	99	1
248	1	24.500000	12.750	1	0	0	0	...	2	0	0	0	73.000000	444	1
249	1	21.830000	11.000	1	0	0	0	...	6	0	0	0	121.000000	0	1
250	1	40.250000	21.500	1	0	0	0	...	11	0	0	0	0.000000	1200	1
251	1	41.420000	5.000	1	0	0	0	...	6	1	0	0	470.000000	0	1
252	0	17.830000	11.000	1	0	0	0	...	11	0	0	0	0.000000	3000	1
253	1	23.170000	11.125	1	0	0	0	...	1	0	0	0	100.000000	0	1
254	1	31.568171	0.625	1	0	0	0	...	0	0	0	0	380.000000	2010	0
255	1	18.170000	10.250	1	0	0	0	...	0	0	0	0	320.000000	13	0
256	1	20.000000	11.045	1	0	0	0	...	0	1	0	0	136.000000	0	0
257	1	20.000000	0.000	1	0	0	0	...	0	0	0	0	144.000000	0	0
258	0	20.750000	9.540	1	0	0	0	...	0	0	0	0	200.000000	1000	0
259	0	24.500000	1.750	0	1	0	1	...	0	0	0	0	132.000000	0	0
260	1	32.750000	2.335	1	0	0	0	...	0	1	0	0	292.000000	0	0
261	0	52.170000	0.000	0	1	0	1	...	0	0	0	0	0.000000	0	0
262	0	48.170000	1.335	1	0	0	0	...	0	0	0	0	0.000000	120	0
263	0	20.420000	10.500	0	1	0	1	...	0	1	0	0	154.000000	32	0
264	1	50.750000	0.585	1	0	0	0	...	0	0	0	0	145.000000	0	0
265	1	17.080000	0.085	0	1	0	1	...	0	0	0	0	140.000000	722	0
266	1	18.330000	1.210	0	1	0	1	...	0	0	0	0	100.000000	0	0
267	0	32.000000	6.000	1	0	0	0	...	0	0	0	0	272.000000	0	0
268	1	59.670000	1.540	1	0	0	0	...	0	1	0	0	260.000000	0	1
269	1	18.000000	0.165	1	0	0	0	...	0	0	0	0	200.000000	40	1
270	1	37.580000	0.000	1	0	0	0	...	0	0	1	0	184.014771	0	1
271	1	32.330000	2.500	1	0	0	0	...	0	1	0	0	280.000000	0	0
272	1	18.080000	6.750	0	1	0	1	...	0	0	0	0	140.000000	0	0
273	1	38.250000	10.125	0	1	0	1	...	0	0	0	0	160.000000	0	0
274	1	30.670000	2.500	1	0	0	0	...	0	1	0	1	340.000000	0	0
275	1	18.580000	5.710	1	0	0	0	...	0	0	0	0	120.000000	0	0

All categorical variables have been encoded in binary values (0,1) for computation and training of algorithm. So, here the data pre-processing phase gets completed with analysing the dataset and datatypes of attributes, removing and replacing the missing values of categorical and numerical variables and finally encoding of categorical variable.

PHASE 2 : TRAINING ALGORITHM

1) Division of independent and dependent variable:

a) **X:** Independent variable will make a matrix having 15 attributes

b) **y:** Dependent variable will make a vector having one attribute of classes

```
In [10]: train_data_percentage = 0.8
    ...: test_data_percentage = 1 - train_data_percentage
    ...: n = len(df_dataset)
    ...:
    ...: # independent and dependent variable
    ...: X = df_data.iloc[:, 0:-1].values
    ...: y = df_data.iloc[:, -1:].values
```

2) Splitting the training and testing data : importing the train_test_split() of sklearn.model_selection library and split the data i.e. independent matrix (X) and dependent vector (y) in two-halves for training and testing. 75% of data will be used for training the algorithm and 25% of data will be used for testing of algorithm predictions.

X : X_train, X_test y: y_train, y_test

```
In [11]: from sklearn.model_selection import train_test_split
    ...: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

3) Finally applying different ML classification algorithm and training the algorithm with help of training data and testing the algorithm prediction on testing data

i. Logistic Regression:

a) Importing logistic regression class from sklearn.linear_model library.

b) Initializing an object classifier and fitting the training data to train the classifier

c) Predicting the dependent variable with help of independent testing data

d) Analysing the accuracy of algorithm with help of predicted and actual dependent variable

```
In [12]: from sklearn.linear_model import LogisticRegression
    ...: classifier = LogisticRegression()
    ...: classifier.fit(X_train, y_train)
    ...: # accuracy = 0.84 (83 + 63)/173
    ...:
    ...: # y_pred vector (logistic regression)
    ...: y_pred_logistic_regression = classifier.predict(X_test)
    ...:
    ...: # accuracy matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm_logistic_regression = confusion_matrix(y_test, y_pred_logistic_regression)
```

Accuracy matrix (Confusion matrix) of logistic regression:

	0	1
0	83	11
1	16	63

ii. K Nearest Neighbour (KNN):

a) Importing k neighbors classifier class from sklearn.neighbors library.

b) Initializing an object classifier with passing parameters like number of neighbors and type of distance and fitting the training data to train the classifier.

c) Predicting the dependent variable with help of independent testing data.

d) Analyzing the accuracy of algorithm with help of predicted and actual dependent variable.

```
In [13]: from sklearn.neighbors import KNeighborsClassifier
    ...: classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
    ...: classifier.fit(X_train, y_train)
    ...: # accuracy = 0.68 (76 + 42)/173
    ...:
    ...: # y_pred vector (KNN)
    ...: y_pred_KNN = classifier.predict(X_test)
    ...:
    ...: # accuracy matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm_KNN = confusion_matrix(y_test, y_pred_KNN)
```

Accuracy matrix (Confusion matrix) of KNN

	0	1
0	76	18
1	37	42

iii. Naïve Bayes Algorithm:

a) Importing gaussian naïve bayes classifier from sklearn.naive_bayes library.

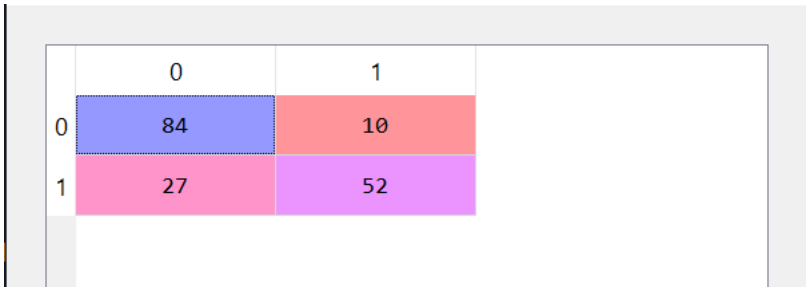
b) Initializing an object classifier and fitting the training data to train the classifier.

c) Predicting the dependent variable with help of independent testing data.

d) Analysing the accuracy of algorithm with help of predicted and actual dependent variable

```
In [14]: from sklearn.naive_bayes import GaussianNB
    ...: classifier = GaussianNB()
    ...: classifier.fit(X_train, y_train)
    ...: # accuracy = 0.78 (84 + 52)/173
    ...:
    ...: # y_pred vector (naive bayes)
    ...: y_pred_naive_bayes = classifier.predict(X_test)
    ...:
    ...: # accuracy matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm_naive_bayes = confusion_matrix(y_test, y_pred_naive_bayes)
```

Accuracy Matrix of Naïve Bayes Algorithm:



	0	1
0	84	10
1	27	52

iv. Decision Tree Algorithm:

a) Importing decision tree classifier from sklearn.tree library.

b) Initializing an object classifier with passing the parameter like gain criteria of decision tree (Entropy or Gini index) and fitting the training data to train the classifier.

c) Predicting the dependent variable with help of independent testing data.

d) Analysing the accuracy of algorithm with help of predicted and actual dependent variable

```
In [15]: from sklearn.tree import DecisionTreeClassifier
    ...: classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    ...: classifier.fit(X_train, y_train)
    ...: # accuracy = 0.84 (85 + 61)/173
    ...:
    ...: # y_pred vector (decision tree)
    ...: y_pred_decision_tree = classifier.predict(X_test)
    ...:
    ...: # accuracy matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm_decision_tree = confusion_matrix(y_test, y_pred_decision_tree)
```

Accuracy Matrix of Decision Tree Algorithm:

	0	1
0	85	9
1	18	61

v. Random Forest Algorithm:

a) Importing random forest classifier from sklearn.ensemble library.

b) Initializing an object classifier with passing the parameter like number of tree estimators and criteria of gain (Entropy or Gini index) and fitting the training data to train the classifier.

c) Predicting the dependent variable with help of independent testing data.

d) Analysing the accuracy of algorithm with help of predicted and actual dependent variable.

```
In [16]: from sklearn.ensemble import RandomForestClassifier
    ...: classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
    ...: classifier.fit(X_train, y_train)
    ...: # accuracy = 0.83 (87 + 57)/173
    ...:
    ...: # y_pred vector (random forest)
    ...: y_pred_random_forest = classifier.predict(X_test)
    ...:
    ...: # accuracy matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm_random_forest = confusion_matrix(y_test, y_pred_random_forest)
```

Accuracy Matrix of Random Forest Algorithm:

	0	1
0	87	7
1	22	57

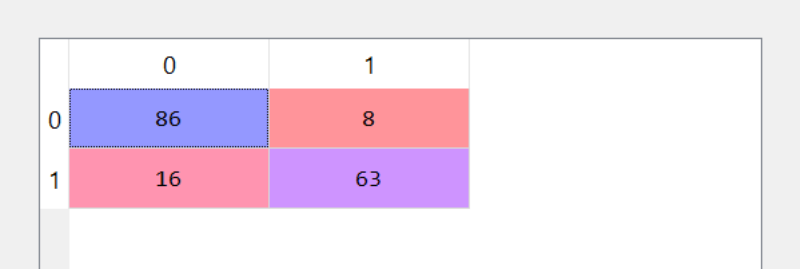
vi. Ensembled Algorithm:

```
In [17]: y_pred = y_pred_logistic_regression + y_pred_decision_tree + y_pred_random_forest
    ...: # accuracy = 0.86% (86 + 63)/173

In [18]: for i in range(0, len(y_test)):
    ...:     if y_pred[i] >= 2:
    ...:         y_pred[i] = 1
    ...:     elif y_pred[i] < 2:
    ...:         y_pred[i] = 0

In [19]: from sklearn.metrics import confusion_matrix
    ...: cm = confusion_matrix(y_test, y_pred)
```

Accuracy Matrix of Ensembled Algorithm:



	0	1
0	86	8
1	16	63

Conclusion

For classification of credit approval success or denial; first we applied data pre-processing in which we analysed the data, filled the missing value, encoded the categorical variable. Then we analysed the data by visualizing them with help of graphs and then finally applied various classification algorithm like logistic regression, naive Bayes algorithm, k nearest neighbour algorithm , decision tree and random forest. then we analysed the accuracy of all algorithm.

ALGORITHM	ACCURACY
LOGISTIC RIGRESSION	84%
K NEAREST NEIGHBOUR	68%
NAÏVE BAYES ALGORITHM	78%
DECISION TREE ALGORITHM	84%
RANDOM FOREST ALGORITHM	83%

We have applied ensembled method for more accurate predictions.

Naïve Bayes and k nearest neighbour algorithm has very low accuracy (< 80%).So, we removed them from our ensembled algorithm. We ensemble logistic regression and tree algorithms to predict out dependent vector.

Accuracy of ensembled algorithm: 86%

How to Improve Accuracy:

1) By collecting more data: The dataset has 690 records. If it gets increased to around 5,000 to 10,000. Then the algorithm can perform better
2) By applying dimensionality reduction: There are various kinds of attribute in our dataset. By applying Principal Component Analysis (PCA) removing the non-important attribute and giving more weightage to important attributes.

Bibliography:

https://archive.ics.uci.edu/ml/datasets/Credit+Approval

https://data-flair.training/blogs/machine-learning-classification-algorithms/

https://www.youtube.com/watch?v=pXdum128xww

https://medium.com/towards-artificial-intelligence/machine-learning-algorithms-for-beginners-with-python-code-examples-ml-19c6afd60daa

https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/