STROKE PREDICTION

Group members:

SOUMYADEEP MAJI, AEC, 10800318031

ARUP MAJI, AEC, 10800318113

ARKYA PATWA, AEC, 10800318115

AMIT KUMAR HAZRA, AEC, 10800319129

Table of Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty (Prof. Arnab Chakraborty / Mentor / Faculty Name) for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him/her time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Soumyadeep Maji

Arup Maji

Arkya Patwa

Amit Kumar Hazra

Project Objective

Description of the Problem:

Stroke is a common ailment in the current society. In 2013, approximately 6.9 million people had an ischemic stroke and 3.4 million people had a hemorrhagic stroke. And in 2015 the stroke rate increased up to 40.2 million a year. It has been evident that the rate of stroke has increased by 10% in developed countries and by 10.6% in the developing countries between 2010 to 2020 .According to WHO about 3.0 million deaths resulted from ischemic stroke while 3.3 million deaths resulted from hemorrhagic stroke. In the WHO standards around 5% of the patients who died of stroke were infants less than one year of age, and about 55% patients belonged to age class above 65 years.

Objective:

The objective is to study the given dataset of stroke patients and apply different machine learning models to identify who are most likely to suffer from stroke and also evaluate the precision and accuracy of the data.

The problem revolves around a labelled data set in which various models have to be introduced so as to get the best predictions out of it or it can be said as analyzing of data set.

The Project Objective is to find out the best predictions, comparing training and test data and analyzing the labelled data by graphs using the different models like the Logistic Regression, K Nearest Neighbor, Decision Tree.

Plan:

The given dataset has mixed values of string, numbers and null. We first need to process the data to be applicable for undergoing a machine learning model. At first the null values are to be replaced by non-null numbers and the strings are replaced and removed by numbers. Then by undergoing proper machine learning models they will give respective outputs.

Project Scope

Project Aim: To study the stroke dataset using different Machine Learning Model.

Project Constraints:
Schedule: Less Time Quality : To the point Others(Policies,Regulations,Management requirements): Add some other models also

Project Team Leads:

Name	Title/Department	Role	Responsibilities
Arkya Patwa	ECE	Leader, coder	KNN model, Documentation
Arup Maji	ECE	Code-Analyser, Coder	Data Pre-processing, Decision Tree model, Data integration, Comparison of model

Soumyadeep Maji	ECE	Coder	Data Collection, Pre-processing, Logistic Regression
Amit Kumar Hazra	ECE	Coder	Documentation

Project Prioritization:

Potential Processes	Priority Ranking(L,M,H)	Estimated Completion dates	Notes
Pre-processing of data	High	24/04/21	It took most of the time for preparing the data. The data null values was resolved and the categorical column was replaced by numerical data.
Logistic regression	Medium	25/04/21	It is a simple model with accuracy 76%.
Decision Tree	High	25/04/21	We used gini algorithm for its working with max depth = 5.
K-Nearest Neighbours	High	26/04/21	For this we took the value of K as 5.

Data Description

- Id – It is passenger’s identification number
  - Integer value
- Gender – It gives the gender of the person
  - Two Classes
    - Male
    - Female
- Age – Age of the person
  - Integer value
  - Min value = 7 months
  - Max value = 82 Years
- Hypertension – If the person has hypertension or not
  - Two classes
    - 0 – Don’t have hypertension
    - 1 – Have hypertension
- Heart-Disease –

    - Two classes
        - 0 - Don’t have heart-disease
        - 1 – Have heart-disease
- Ever-married –
    - Two classes
        - Yes - They are married
        - No – They are unmarried
- Work-type –
    - Four Classes
        - Private
        - Self-employed
        - Govt-job
        - Children
        - Never-worked
- Residence_type –
    - Two classes
        - Urban
        - Rural
- Avg_glucose_level –
    - Gives the average glucose level of the person

    - Continuous value
    - Min value – 55.39
    - Max value – 271.74
- Bmi –
    - Body mass index of the person
    - Continuous values
    - Have some null values
    - Min value – 13.8
    - Max value – 64.8
- Smoking_status –
    - Four classes
        - Never smoked
        - Unknown
        - Formerly smoke
        - smokes
- Stroke –
    - Two classes
        - 0 – Will not have stroke
        - 1 – Will have stroke
    - Target Variable

Model Building

- Models used –
  - Logistics Regression
  - Decision Tree
  - K Nearest Neighbor

- <u>Logistics Regression:</u>

  Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable. A binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model. It is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

    - Precision :
      - Class 0 : 0.80
      - Class 1 : 0.58

- Recall :
    - Class 0 : 0.89
    - Class 1 : 0.41
- F score :
    - Class 0 : 0.84
    - Class 1 : 0.48
- Support :
    - Class 0 : 218
    - Class 1 : 82
- Accuracy : **0.76**

- Decision Tree:

  A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

- Precision :
  - Class 0 : 0.89
  - Class 1 : 0.52
- Recall :
  - Class 0 : 0.86
  - Class 1 : 0.59
- F score :
  - Class 0 : 0.87
  - Class 1 : 0.55
- Support :
  - Class 0 : 239
  - Class 1 : 61
- Accuracy : **0.81**

- K Nearest Neighbor :

  K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

- Precision :
  - Class 0 : 0.85
  - Class 1 : 0.43
- Recall :
  - Class 0 : 0.84
  - Class 1 : 0.45
- F score :
  - Class 0 : 0.85
  - Class 1 : 0.44
- Support :
  - Class 0 : 239
  - Class 1 : 61
- Accuracy : **0.77**

Observing the accuracy of the above three models, we can conclude that Decision Tree having **0.81** accuracy is the most accepted model.

Code

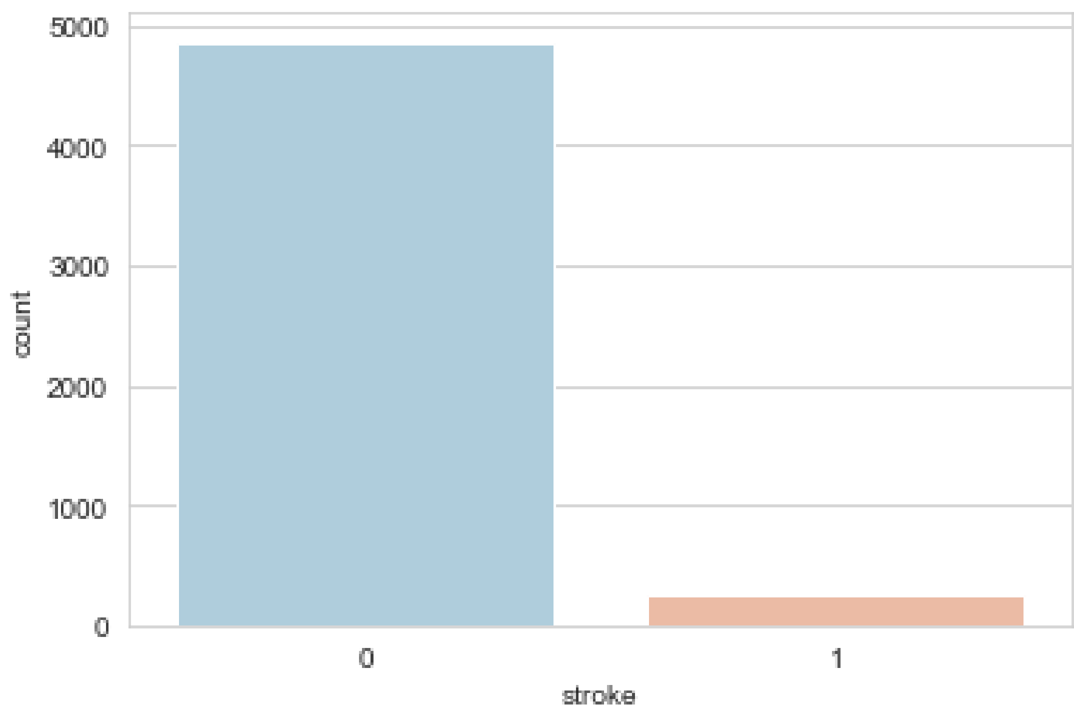
We can have used one type of data file -

1. CSV (Comma separated values)

```
# importing required modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns

# Reading the csv file
df_stroke = pd.read_csv('stroke.csv')
```

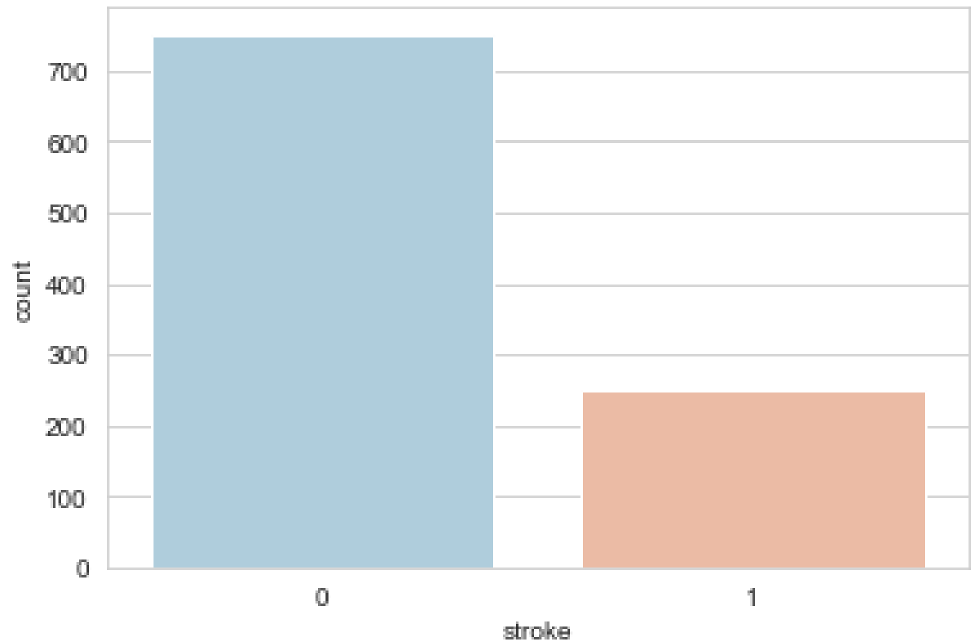
```
# gathering some information about the output variable
sns.set_style('whitegrid')
sns.countplot(x='stroke',data=df_stroke,palette='RdBu_r')
df_stroke.stroke.value_counts()
0    4861
1     249Name: stroke, dtype: int64
```



```
# since the ratio of class of output variable is too large (4861:249)
# we should slice the data so that the ratio decreases
# lets take first 1000 data and check if the ratio is acceptable
```

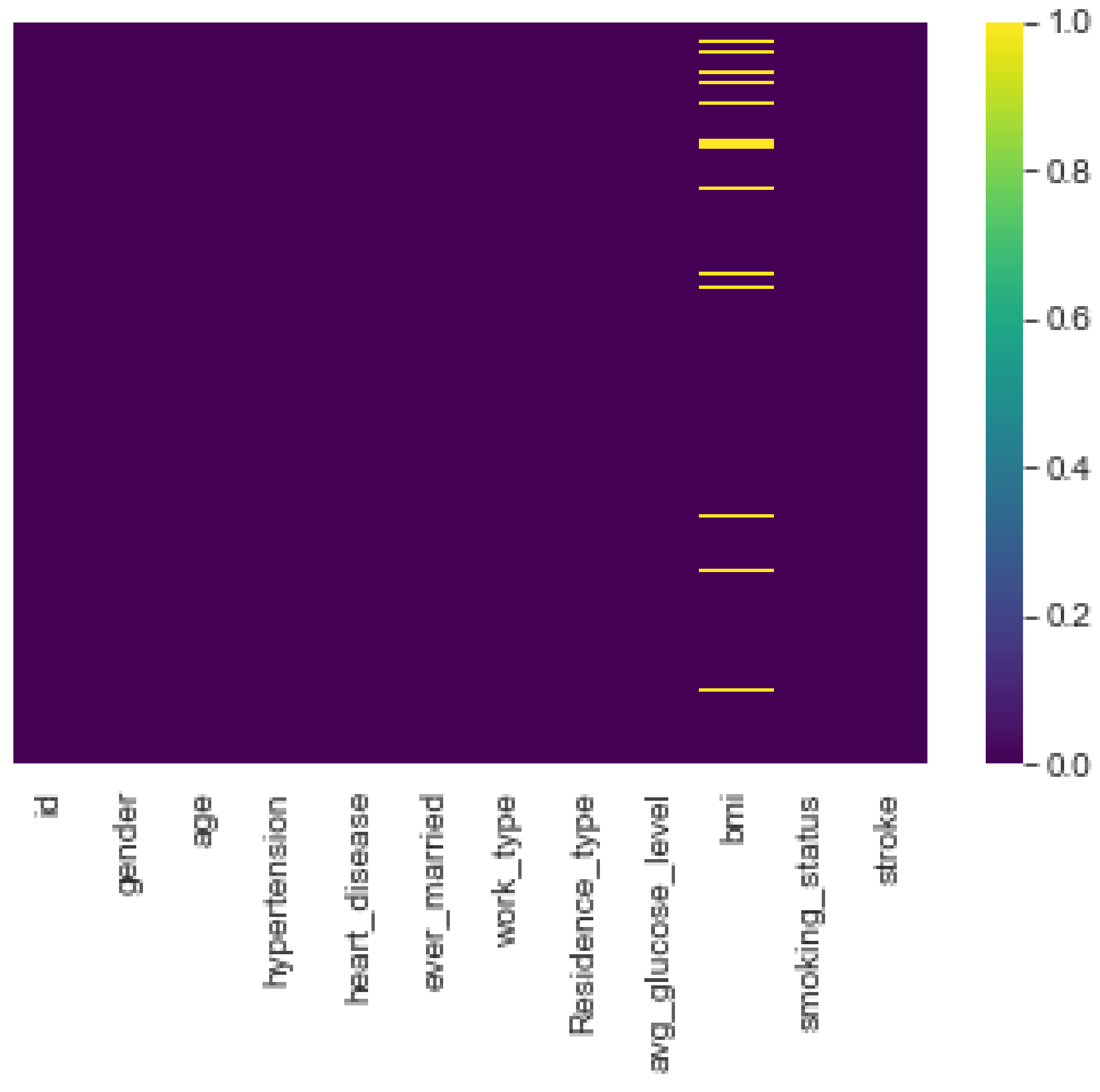
```
df_stroke = df_stroke[:1000]
sns.set_style('whitegrid')
sns.countplot(x='stroke',data=df_stroke,palette='RdBu_r')
df_stroke.stroke.value_counts()
```

```
0     751
1     249Name: stroke, dtype: int64
```




```
# gathering information about the null values in dataframe
sns.heatmap(df_stroke.isnull(),yticklabels=False,cbar=True,cmap='viridis')
```

```
<AxesSubplot:>
```



1.0
0.8
0.6
0.4
0.2
0.0
id
gender
age
hypertension
heart_disease
ever_married
work_type
Residence_type
avg_glucose_level
bmi
smoking_status
stroke

knowing more about data using info() method

print(df_stroke.info())

finding which features really matter in predicting the target variable

print(df_stroke.corr())

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 1000 non-null   int64
 1   gender             1000 non-null   object
 2   age                1000 non-null   float64
 3   hypertension       1000 non-null   int64
 4   heart_disease      1000 non-null   int64
 5   ever_married       1000 non-null   object
 6   work_type          1000 non-null   object
 7   Residence_type     1000 non-null   object
 8   avg_glucose_level  1000 non-null   float64
 9   bmi                942 non-null    float64
 10  smoking_status     1000 non-null   object
 11  stroke             1000 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 74.3+ KB
None
```

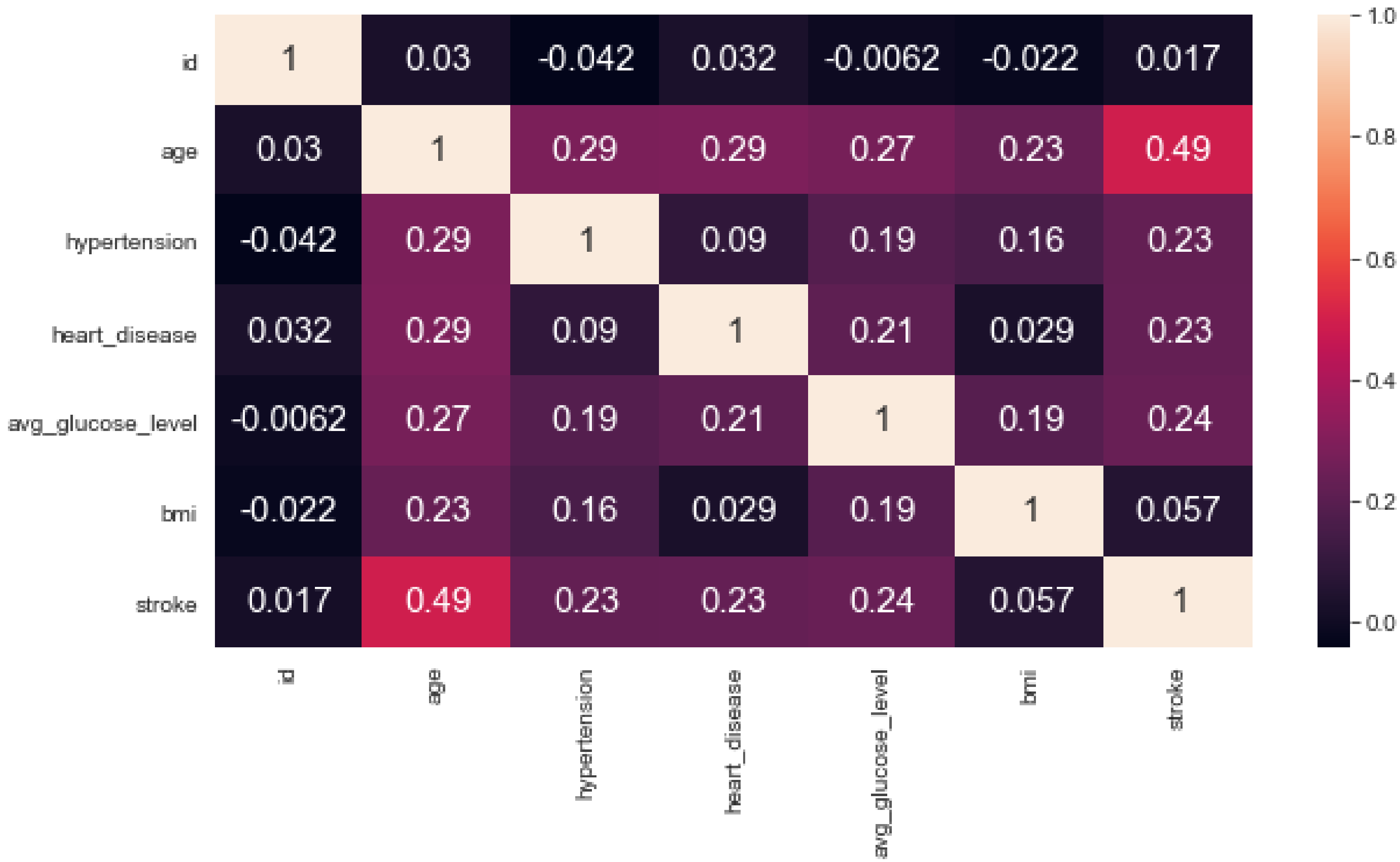
```
                         id       age  hypertension  heart_disease  \
id                 1.000000  0.030230     -0.041954       0.031712
age                0.030230  1.000000      0.286122       0.291412
hypertension      -0.041954  0.286122      1.000000       0.089960
heart_disease      0.031712  0.291412      0.089960       1.000000
avg_glucose_level -0.006230  0.271538      0.188011       0.212584
bmi               -0.021561  0.234416      0.160267       0.029295
stroke             0.016786  0.494177      0.226350       0.227414

                   avg_glucose_level       bmi    stroke
id                         -0.006230 -0.021561  0.016786
age                         0.271538  0.234416  0.494177
hypertension                0.188011  0.160267  0.226350
heart_disease               0.212584  0.029295  0.227414
avg_glucose_level           1.000000  0.190602  0.236228
bmi                         0.190602  1.000000  0.056830
stroke                      0.236228  0.056830  1.000000
```

```
# visualizing the corelation so that we can get to know about the relation between
# the features at a glance
plt.figure(figsize=(10,5))
```

```
sns.heatmap(df_stroke.corr(),annot=True,annot_kws={"size":15})
```

```
<AxesSubplot:>
```

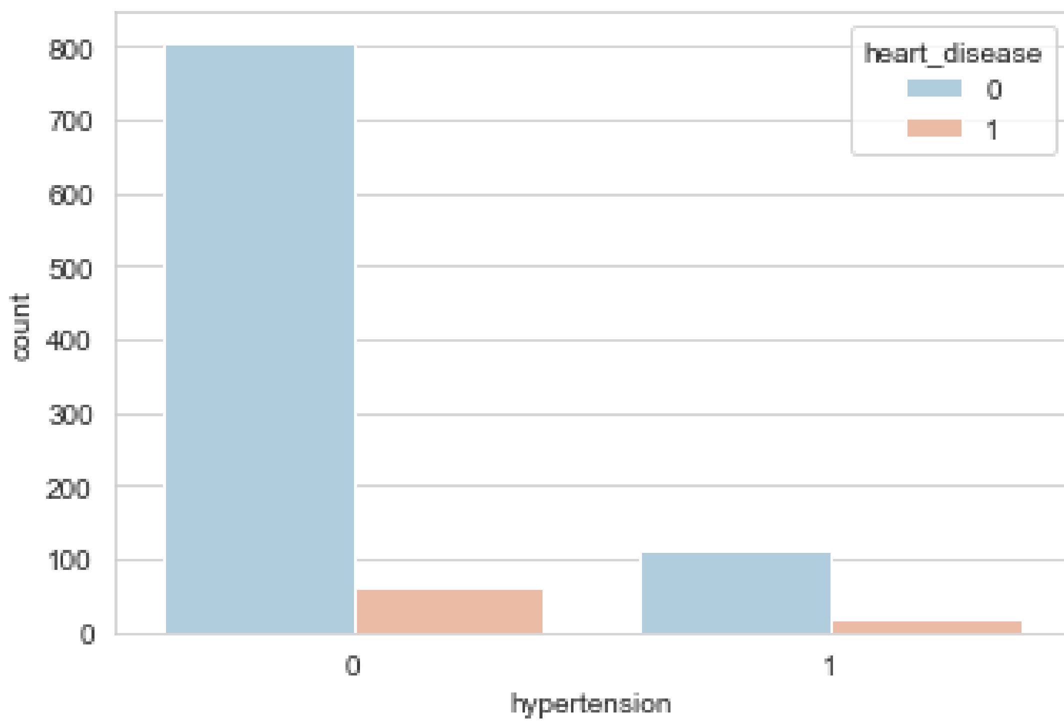


checking the dependence of hypertension and heart_disease

sns.set_style('whitegrid')

sns.countplot(x='hypertension',hue='heart_disease',data=df_stroke,palette='RdBu_r')

<AxesSubplot:xlabel='hypertension', ylabel='count'>



```
print(df_stroke[(df_stroke.hypertension == 1) & (df_stroke.heart_disease ==
1)].shape[0])
```

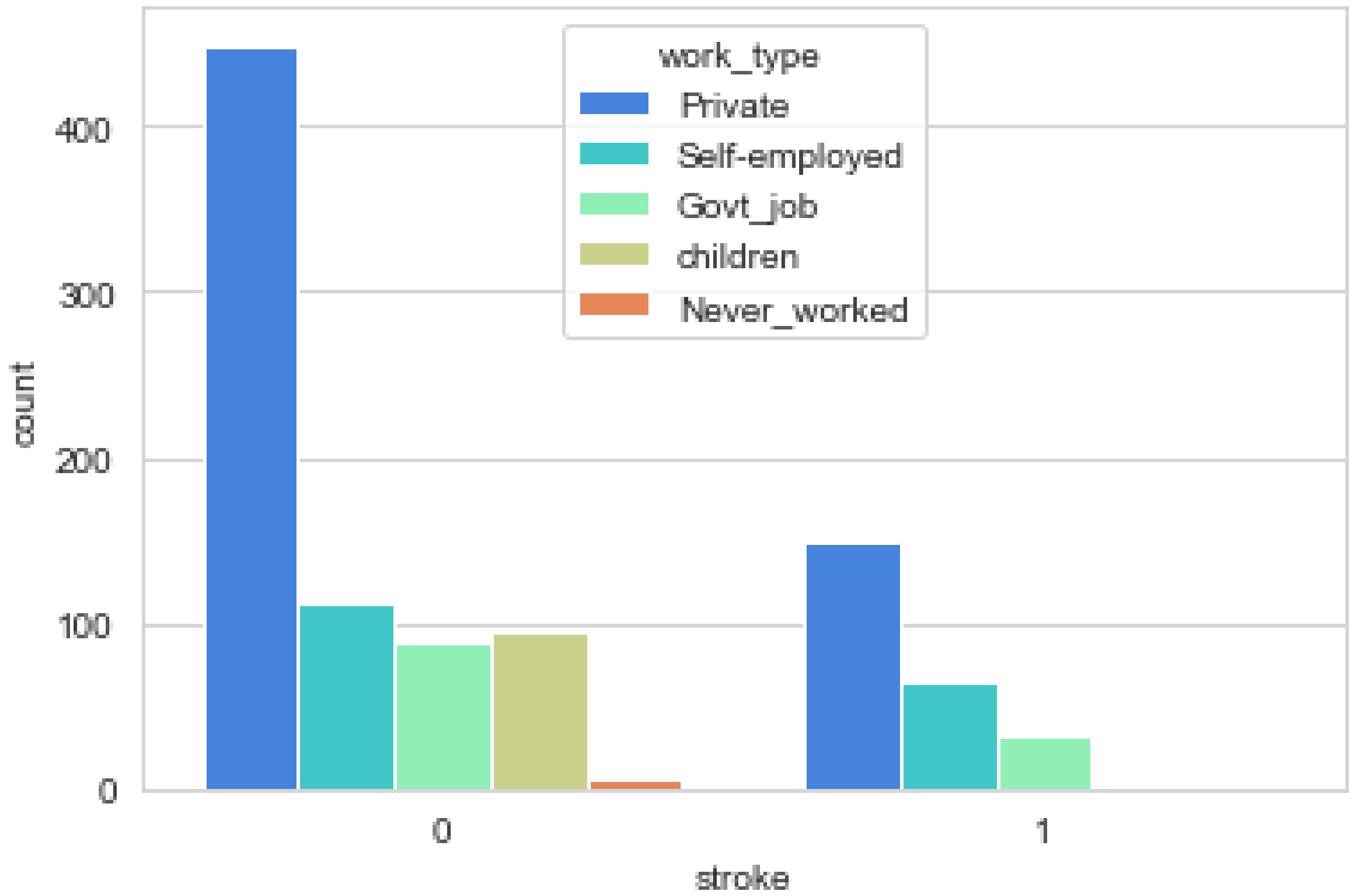
19

```
print ("0 - Male =", len(df_stroke[(df_stroke.hypertension == 0) &
(df_stroke.gender == 'Male')]), end= ", ")
print ("0 - Female =", len(df_stroke[(df_stroke.hypertension == 0) &
(df_stroke.gender == 'Female')]), end= ", ")
print ("1 - Male =", len(df_stroke[(df_stroke.hypertension == 1) &
(df_stroke.gender == 'Male')]), end= ", ")
print ("1 - Female =", len(df_stroke[(df_stroke.hypertension == 1) &
(df_stroke.gender == 'Female')]), end= ", ")
```

0 - Male = 356, 0 - Female = 512, 1 - Male = 52, 1 - Female = 80,

```
sns.set_style('whitegrid')
sns.countplot(x='stroke',hue='work_type',data=df_stroke,palette='rainbow')
```

<AxesSubplot:xlabel='stroke', ylabel='count'>



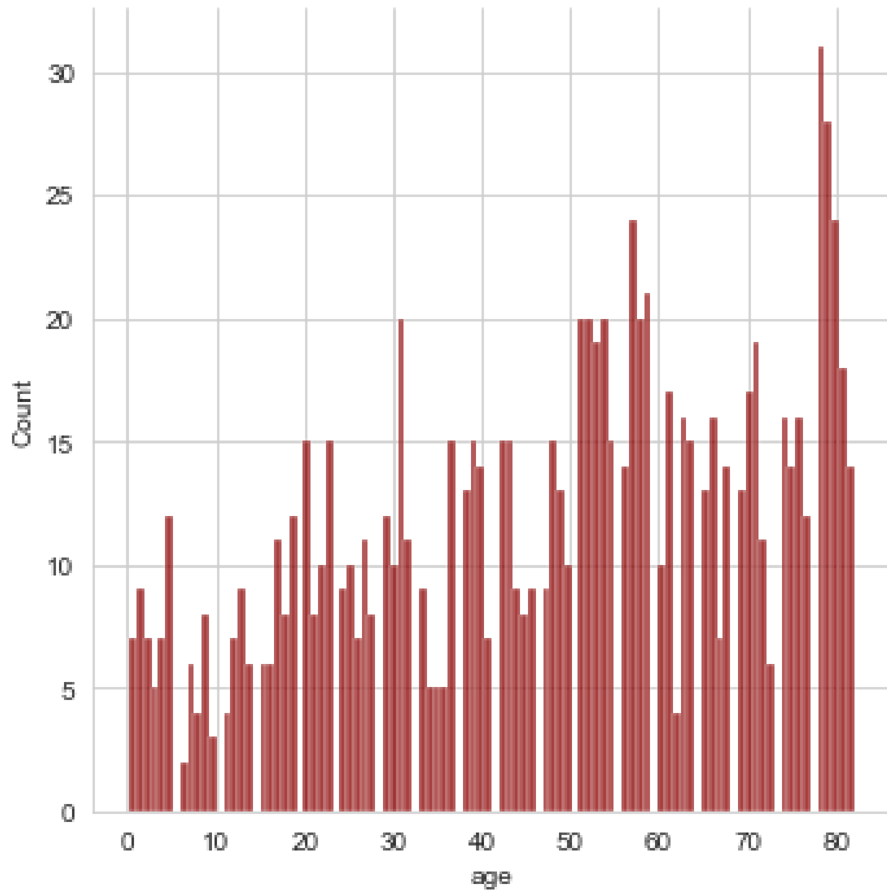
```
print ("0 - work_type-Private =", len(df_stroke[(df_stroke.stroke == 0) &
(df_stroke.work_type == 'Private')]), end= ", ")
print ("0 - work_type-Self-employed =", len(df_stroke[(df_stroke.stroke == 0) &
(df_stroke.work_type == 'Self-employed')]), end= ", ")
print ("0 - work_type-Govt_job =", len(df_stroke[(df_stroke.stroke == 0) &
(df_stroke.work_type == 'Govt_job')]), end= ", ")
print ("0 - work_type-children =", len(df_stroke[(df_stroke.stroke == 0) &
(df_stroke.work_type == 'children')]), end= ", ")
print ("1 - work_type-Private =", len(df_stroke[(df_stroke.stroke == 1) &
(df_stroke.work_type == 'Private')]), end= ", ")
print ("1 - work_type-Self-employed =", len(df_stroke[(df_stroke.stroke == 1) &
(df_stroke.work_type == 'Self-employed')]), end= ", ")
print ("1 - work_type-Govt_job =", len(df_stroke[(df_stroke.stroke == 1) &
(df_stroke.work_type == 'Govt_job')]), end= ", ")
```

```python
print ("0 - work_type-children =", len(df_stroke[(df_stroke.stroke == 1) & (df_stroke.work_type == 'children')]), end= ", ")
```

0 - work_type-Private = 449, 0 - work_type-Self-employed = 113, 0 - work_type-Govt_job = 89, 0 - work_type-children = 95, 1 - work_type-Private = 149, 1 - work_type-Self-employed = 65, 1 - work_type-Govt_job = 33, 0 - work_type-children = 2,

```python
sns.displot(df_stroke['age'].dropna(),kde=False,color='darkred',bins=100)
```

```
<seaborn.axisgrid.FacetGrid at 0x12cfa1d0>
```



calculating the median of bmi as it will be needed to replace the null values of bmi

```python
print ("Median of bmi for strokes-1 =", np.median(df_stroke[['bmi']].dropna()),
end= ", ")
```

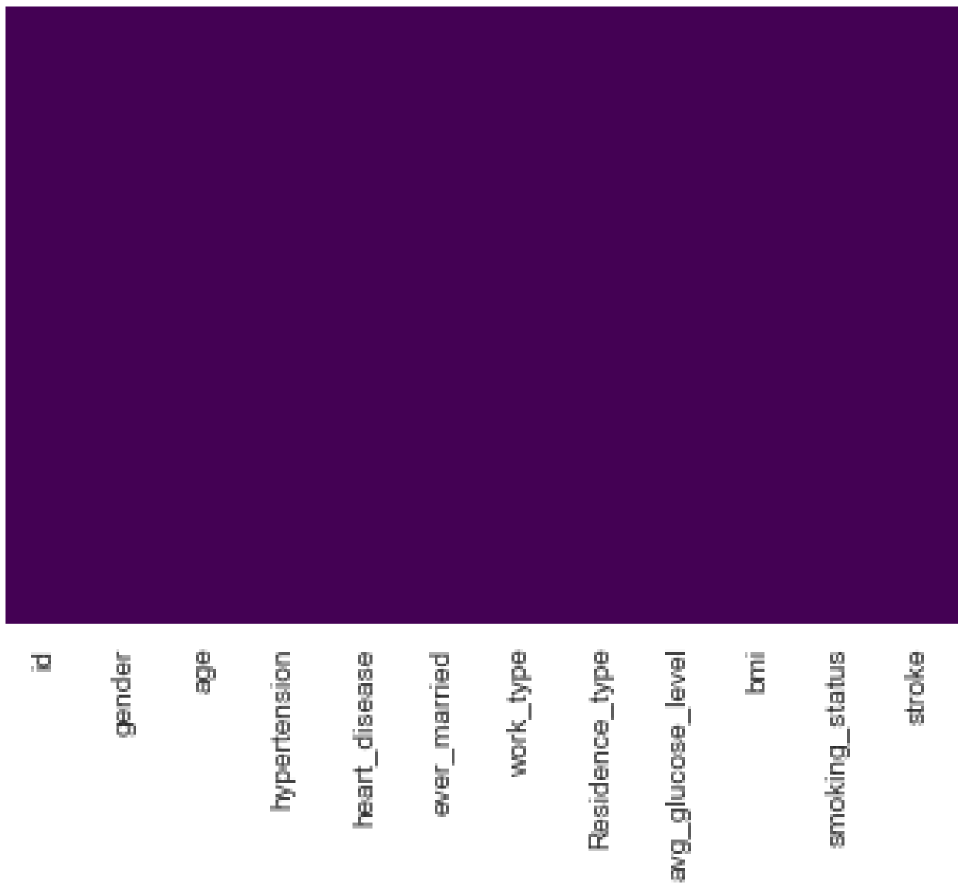
Median of bmi for strokes-1 = 28.45,

```python
# function that will help us to replace the null values of bmi with its ,median
values
def impute_bmi(cols):
    bmi = cols[0]
    if pd.isnull(bmi):
        return 28.45
    else:
        return bmi

# replacing the null values of bmi with its median value (28.45)
df_stroke['bmi']=df_stroke[['bmi']].apply(impute_bmi,axis=1)
```

```
# cross-checking if the values are correctly replaced or not
sns.heatmap(df_stroke.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<AxesSubplot:>
```



```python
# cross-checking the data as well using info method
print(df_stroke.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 1000 non-null   int64  
 1   gender             1000 non-null   object 
 2   age                1000 non-null   float64
 3   hypertension       1000 non-null   int64  
 4   heart_disease      1000 non-null   int64  
 5   ever_married       1000 non-null   object 
 6   work_type          1000 non-null   object 
 7   Residence_type     1000 non-null   object 
 8   avg_glucose_level  1000 non-null   float64
 9   bmi                1000 non-null   float64
 10  smoking_status     1000 non-null   object 
 11  stroke             1000 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 74.3+ KB
None
```

LOGISTIS REGRESSION:

```
from sklearn.linear_model import LogisticRegression
# splitting the data for training and testing
# 70% of the data are kept for training and 30% of the data are kept for the testing
x_train_logistics,x_test_logistics,y_train_logistics,y_test_logistics=train_test_split(df_stroke.drop('stroke',axis=1),df_stroke['stroke'],test_size=0.30,random_state=101)


# validating if the rows are correctly divided
print("No. of Train rows -> ",len(y_train_logistics), 0.70 * df_stroke.shape[0])
print("No. of Test rows -> ",len(y_test_logistics), 0.30 * df_stroke.shape[0])
print(x_train_logistics.shape)
print(y_train_logistics.shape)
```

No. of Train rows -> 700 700.0

No. of Test rows -> 300 300.0

(700, 10)

(700,)

```
# printing if the train and test data are correctly seperated
print(x_train_logistics.head())
print(y_train_logistics.head())
print(x_test_logistics.head())
print(y_test_logistics.head())
```

```
     gender   age  hypertension  heart_disease  ever_married  work_type  \
290       1  13.0             0              0             0          3
167       1  79.0             1              0             1          0
486       1  20.0             0              0             0          0
683       0   2.0             0              0             0          3
876       0  37.0             0              0             1          0

     Residence_type  avg_glucose_level    bmi  smoking_status
290               1             114.84  18.30               1
167               0              75.02  28.45               0
486               0             104.48  21.70               0
683               0              79.89  31.60               1
876               1             106.35  29.70               0
290    0
167    1
486    0
683    0
876    0
Name: stroke, dtype: int64
     gender   age  hypertension  heart_disease  ever_married  work_type  \
545       1  18.0             0              0             0          0
298       1  69.0             0              0             1          1
109       0  53.0             0              0             1          2
837       0  39.0             0              0             0          2
194       0  72.0             0              0             1          0
```

```
     Residence_type  avg_glucose_level   bmi  smoking_status
545               0              70.34  24.2               1
298               0             203.04  33.6               0
109               1              64.17  41.5               0
837               1              79.44  22.7               0
194               0              97.92  26.9               3
545    0
298    0
109    1
837    0
194    1
Name: stroke, dtype: int64
```

```python
# creating an object of LogisticRegression() class which will help us in
# training and testing our model
logmodel=LogisticRegression()
# fitting the training data so that model learns from previous data
logmodel.fit(x_train_logistics,y_train_logistics)

# predicting the output of the model for the testing data
# data testing is very important as based on it we will decide if the model is
# correctly trained and accurate or not
prediction_logistics=logmodel.predict(x_test_logistics)
# printing if model successfully predicted or not
print(prediction_logistics,len(prediction_logistics))
```

[0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1

0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1] 300

```
# print(classification_report(y_test,predictions))

# checking the accuracy of the model

cr_logistics = classification_report(y_test_logistics, prediction_logistics)

print(cr_logistics)
```

```
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       218
           1       0.59      0.41      0.49        82

    accuracy                           0.76       300
   macro avg       0.69      0.65      0.66       300
weighted avg       0.74      0.76      0.75       300
```

```python
# checking the correct and incorrect predicted outputs w.r.t. actual data
cm_logistic = confusion_matrix(y_test_logistics, prediction_logistics)
print(cm_logistic)
confusion_df =
pd.DataFrame(confusion_matrix(y_test_logistics,prediction_logistics),
        columns=["Predicted Class " + str(class_name) for class_name in [0,1]],
        index = ["Class " + str(class_name) for class_name in [0,1]])
print(confusion_df)
```

```
[[194  24]
 [ 48  34]]
         Predicted Class 0  Predicted Class 1
Class 0                194                 24
Class 1                 48                 34
```

```
prfs_logistics = precision_recall_fscore_support(y_test_logistics, prediction_logistics)
prfs_logistics
```

```
(array([0.80165289, 0.5862069 ]),
array([0.88990826, 0.41463415]),
array([0.84347826, 0.48571429]),
array([218, 82], dtype=int32))
```

```
# printing the coefficients
print(logmodel.coef_)
```

```
[[ 0.06198782 0.08930118 0.3001234 -0.04158852 -0.62088273 -0.01044166
-0.18756951 0.00392328 0.00530991 0.13081173]]
```

printing the intercept

```
print(logmodel.intercept_)
```

[-6.56176575]

Decision Tree:

df_stroke

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.00	0	1	1	0	1	228.69	36.60	2	1
1	0	61.00	0	0	1	1	0	202.21	28.45	0	1
2	1	80.00	0	1	1	0	0	105.92	32.50	0	1
3	0	49.00	0	0	1	0	1	171.23	34.40	3	1
4	0	79.00	1	0	1	1	0	174.12	24.00	0	1
...	...	...	...	...	...	...	...	...	...	...	...
995	1	1.40	0	0	0	3	1	90.51	18.90	1	0
996	1	0.24	0	0	0	3	0	118.87	16.30	1	0
997	1	55.00	0	0	1	0	0	56.42	31.80	0	0
998	0	29.00	0	0	0	0	1	73.67	21.00	1	0
999	1	4.00	0	0	0	3	0	89.11	20.10	1	0

1000 rows × 11 columns

spliting the data for training and testing

x_train_tree,x_test_tree, y_train_tree, y_test_tree = train_test_split(df_stroke.drop("stroke", axis = 1), df_stroke["stroke"], test_size = 0.3, random_state = 176)

cheking if the data is successfully seperated

x_train_tree, y_train_tree

```
(      gender   age  hypertension  heart_disease  ever_married  work_type  \
 314        0  78.0             1              0             1          0
 207        1  78.0             0              0             0          1
 127        0  80.0             0              0             1          0
 145        1  66.0             0              0             1          0
 50         0  76.0             0              0             0          0
 ..       ...   ...           ...            ...           ...        ...
 920        0  21.0             0              0             0          2
 836        0  51.0             0              0             0          0
 757        1  19.0             0              0             0          0
 701        1  38.0             0              0             1          0
 832        0  33.0             0              0             1          0
```

```
     Residence_type  avg_glucose_level    bmi  smoking_status
314               1             218.46  34.30               0
207               1              90.19  26.90               0
127               1              73.54  24.00               1
145               1             151.16  27.50               2
50                1              89.96  28.45               1
..              ...                ...    ...             ...
920               1             111.61  36.90               3
836               1             110.76  24.70               2
757               0              84.31  31.80               0
701               1              88.97  30.20               0
832               1             121.04  31.40               1

[700 rows x 10 columns],
314    0
207    1
127    1
145    1
50     1
      ..
920    0
836    0
757    0
701    0
832    0
Name: stroke, Length: 700, dtype: int64)
```

x_test_tree, y_test_tree

```
(      gender   age  hypertension  heart_disease  ever_married  work_type  \
 752        0  78.0             0              0             0          0
 271        0  49.0             0              0             1          0
 919        1   8.0             0              0             0          3
 865        1  52.0             0              0             1          0
 42         1  82.0             0              1             1          0
 ..       ...   ...           ...            ...           ...        ...
 868        0  51.0             0              0             1          0
 765        0  56.0             0              0             1          0
 593        0  30.0             0              0             1          0
 135        0  71.0             0              0             1          2
 973        0  49.0             0              0             0          0

      Residence_type  avg_glucose_level    bmi  smoking_status
 752               1             103.86  30.60               1
 271               0              60.22  31.50               3
 919               1             111.02  22.40               1
 865               1             226.70  28.45               3
 42                1             144.90  26.40               3
 ..              ...                ...    ...             ...
 868               1             105.36  43.70               1
 765               0             114.21  21.30               0
 593               1              59.82  25.40               0
 135               1             263.32  38.70               0
 973               0              65.81  32.30               1

 [300 rows x 10 columns],
 752    0
 271    0
 919    0
 865    0
 42     1
       ..
 868    0
 765    0
 593    0
 135    1
 973    0
 Name: stroke, Length: 300, dtype: int64)
```

```python
from sklearn.tree import DecisionTreeClassifier
# creating an object of DecisionTreeClassifier for training and testing of data
dt_train_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,
                                       max_depth=5, min_samples_leaf=5)
dt_train_gini.fit(x_train_tree, y_train_tree)
```

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=5, random_state=100)
```

```python
prediction_tree = dt_train_gini.predict(x_test_tree)
```

Note: Precision is how many are correctly predicted with respect to actual number of data Recall is how many are correctly predicted with respect to total number of its own prediction e.g.- Precision of class A is the measure of how many data are correctly predicted as A to the actual number of data present in A Recall of class A is the measure of how many data are correctly predicted as A to total no of predicted class A data (correct + incorrect).

```
cr_tree = classification_report(y_test_tree,prediction_tree)
print(cr_tree)
```

```
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       239
           1       0.53      0.59      0.56        61

    accuracy                           0.81       300
   macro avg       0.71      0.73      0.72       300
weighted avg       0.82      0.81      0.81       300
```

```
cm_tree = confusion_matrix(y_test_tree, prediction_tree)
print(cm_tree)
confusion_df = pd.DataFrame(confusion_matrix(y_test_tree,prediction_tree),
             columns=["Predicted Class " + str(class_name) for class_name in
[0,1]],
             index = ["Class " + str(class_name) for class_name in [0,1]])
confusion_df
```

```
[[207  32]
 [ 25  36]]
```

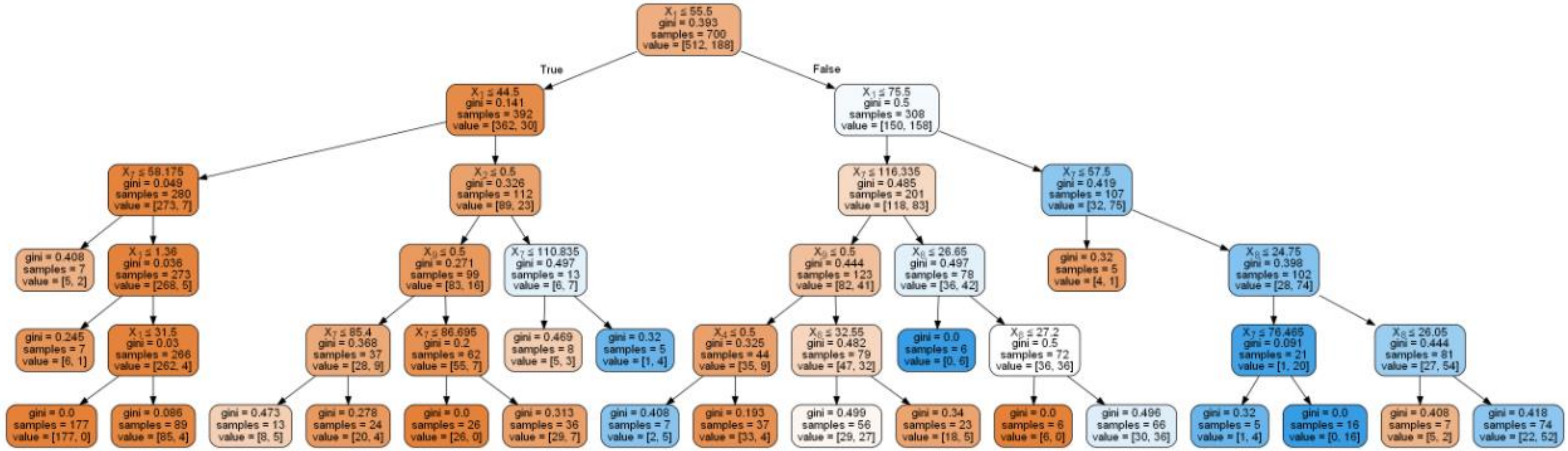
	Predicted Class 0	Predicted Class 1
Class 0	207	32
Class 1	25	36

```
prfs_tree = precision_recall_fscore_support(y_test_tree, prediction_tree)
prfs_tree
```

```
(array([0.89224138, 0.52941176]),
array([0.86610879, 0.59016393]),
array([0.87898089, 0.55813953]),
array([239, 61], dtype=int32))
```

```
# exporting the visualizing graph on a text file
from sklearn import tree
with open("dt_train_gini.txt", "w") as f:
    f = tree.export_graphviz(dt_train_gini, out_file=f)
```

```python
# visualizing the graph
from io import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(dt_train_gini, out_file=dot_data,filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



K-Nearest Neighbor -

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
df_stroke
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.00	0	1	1	0	1	228.69	36.60	2	1
1	0	61.00	0	0	1	1	0	202.21	28.45	0	1
2	1	80.00	0	1	1	0	0	105.92	32.50	0	1
3	0	49.00	0	0	1	0	1	171.23	34.40	3	1
4	0	79.00	1	0	1	1	0	174.12	24.00	0	1
...	...	...	...	...	...	...	...	...	...	...	...
995	1	1.40	0	0	0	3	1	90.51	18.90	1	0
996	1	0.24	0	0	0	3	0	118.87	16.30	1	0
997	1	55.00	0	0	1	0	0	56.42	31.80	0	0
998	0	29.00	0	0	0	0	1	73.67	21.00	1	0
999	1	4.00	0	0	0	3	0	89.11	20.10	1	0

1000 rows × 11 columns

```
# splitting the training and test data
x_train_knn, x_test_knn, y_train_knn, y_test_knn =
train_test_split(df_stroke.drop(["stroke"], axis = 1),
                                                   df_stroke["stroke"], test_size = 0.30,
random_state = 0)
knn_classifier =  KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
```

```python
knn_classifier.fit(x_train_knn, y_train_knn)

st_x = StandardScaler()

x_train_knn = st_x.fit_transform(x_train_knn)

x_test_knn = st_x.transform(x_test_knn)


# printing the data if they are split properly

print(x_train_knn[:5])

print(y_train_knn[:5])

print(x_test_knn[:5])

print(y_train_knn[:5])
```

```
[[ 1.18553713  0.40241273 -0.4010765  -0.29201253  0.6435382  -0.72590088
   0.94982836 -0.59482224 -0.14759054  1.70821431]
 [ 1.18553713  0.44639117 -0.4010765  -0.29201253  0.6435382  -0.72590088
   0.94982836 -0.51734352  0.05680762  0.8002796 ]
 [ 1.18553713  0.22649899  2.49328996 -0.29201253  0.6435382  -0.72590088
  -1.0528218   1.69977016 -0.14759054  1.70821431]
 [ 1.18553713  0.40241273  2.49328996 -0.29201253  0.6435382  -0.72590088
  -1.0528218   2.18613148  1.5733101   0.8002796 ]
 [ 1.18553713  0.66628334 -0.4010765  -0.29201253  0.6435382   1.2098348
   0.94982836 -0.35469735  0.20186438 -1.01558983]]
105    1
68     1
479    0
399    0
434    0
Name: stroke, dtype: int64
[[ 1.18553713  0.66628334 -0.4010765  -0.29201253  0.6435382  -0.72590088
  -1.0528218  -0.5559843  -0.6552892  -0.10765512]
 [ 1.18553713 -0.25726379 -0.4010765  -0.29201253  0.6435382  -0.72590088
   0.94982836  0.45892777  0.83483933  1.70821431]
 [ 1.18553713  0.88617551 -0.4010765  -0.29201253  0.6435382   0.24196696
  -1.0528218   1.78552905  0.53153883 -1.01558983]
 [-0.84349952  0.27047743 -0.4010765  -0.29201253  0.6435382   0.24196696
  -1.0528218  -0.71508183 -0.22011892 -1.01558983]
 [-0.84349952 -1.4007031  -0.4010765  -0.29201253 -1.55390931 -0.72590088
   0.94982836 -0.88048805  3.445861   -1.01558983]]
```

```
105    1
68     1
479    0
399    0
434    0
Name: stroke, dtype: int64
```

```
# training the model
knn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2 )
knn_classifier.fit(x_train_knn, y_train_knn)
KNeighborsClassifier()


prediction_knn = knn_classifier.predict(x_test_knn)
print(prediction_knn)
[0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0
```

```
0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0
0 0 0 0 0
0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0
0 0 1 0 0
0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1
0 0 1 0 0
0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0
 1 1 0 0 0 1]
```

```python
cr_knn = classification_report(y_test_knn,prediction_knn)
print(cr_knn)
```

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.85       239
           1       0.43      0.46      0.44        61

    accuracy                           0.77       300
   macro avg       0.65      0.65      0.65       300
weighted avg       0.77      0.77      0.77       300
```

```
cm = confusion_matrix(y_test_knn, prediction_knn)
print(cm)
print("Accuracy is: ", metrics.accuracy_score(y_test_knn, prediction_knn))
```

```
[[202 37]
 [ 33 28]]
```

Accuracy is: 0.7666666666666667

```
prfs_knn = precision_recall_fscore_support(y_test_knn, prediction_knn)
prfs_knn
```

```
(array([0.85957447, 0.43076923]),
array([0.84518828, 0.45901639]),
array([0.85232068, 0.44444444]),
array([239, 61], dtype=int32))
```

Comparing the models:

prfs_logistics

(array([0.80165289, 0.5862069]),

array([0.88990826, 0.41463415]),

array([0.84347826, 0.48571429]),

array([218, 82], dtype=int32))

prfs_tree

(array([0.89224138, 0.52941176]),

array([0.86610879, 0.59016393]),

array([0.87898089, 0.55813953]),

array([239, 61], dtype=int32))

prfs_knn

(array([0.85957447, 0.43076923]),

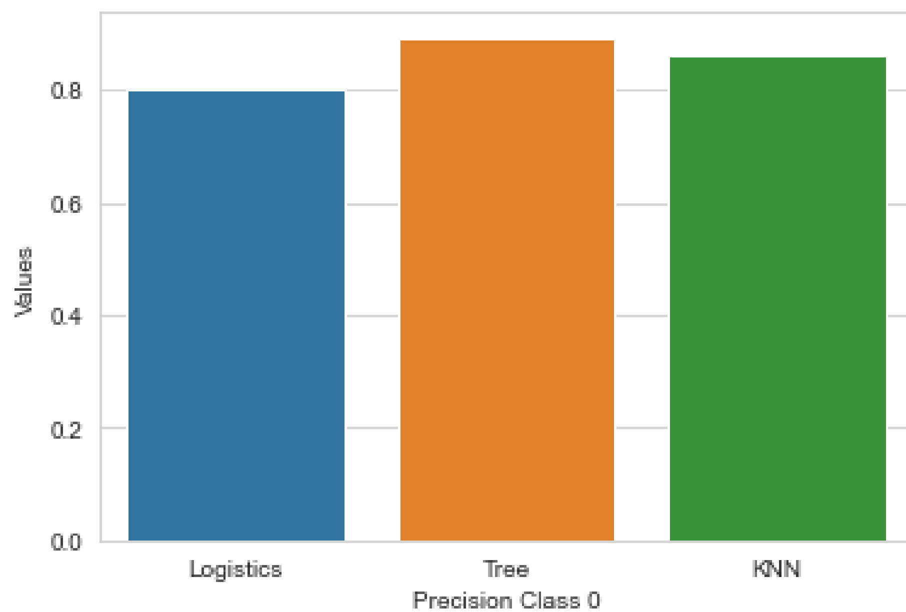
array([0.84518828, 0.45901639]),

array([0.85232068, 0.44444444]),

array([239, 61], dtype=int32))

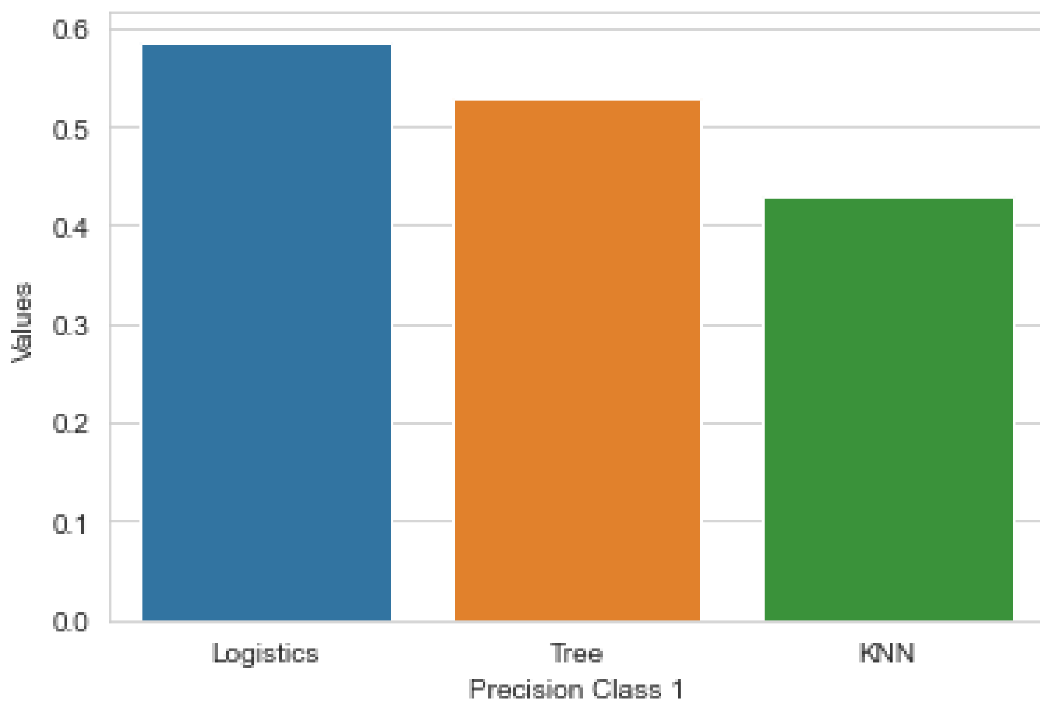
```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[0][0],
                                              prfs_tree[0][0], prfs_knn[0][0]],orient =
"vertical")
bar_plot.set(xlabel = "Precision Class 0", ylabel = "Values")
# print(prfs_logistics[0][0],prfs_tree[0][0], prfs_knn[0][0], sep = "\n")
```

```
[Text(0.5, 0, 'Precision Class 0'), Text(0, 0.5, 'Values')]
```

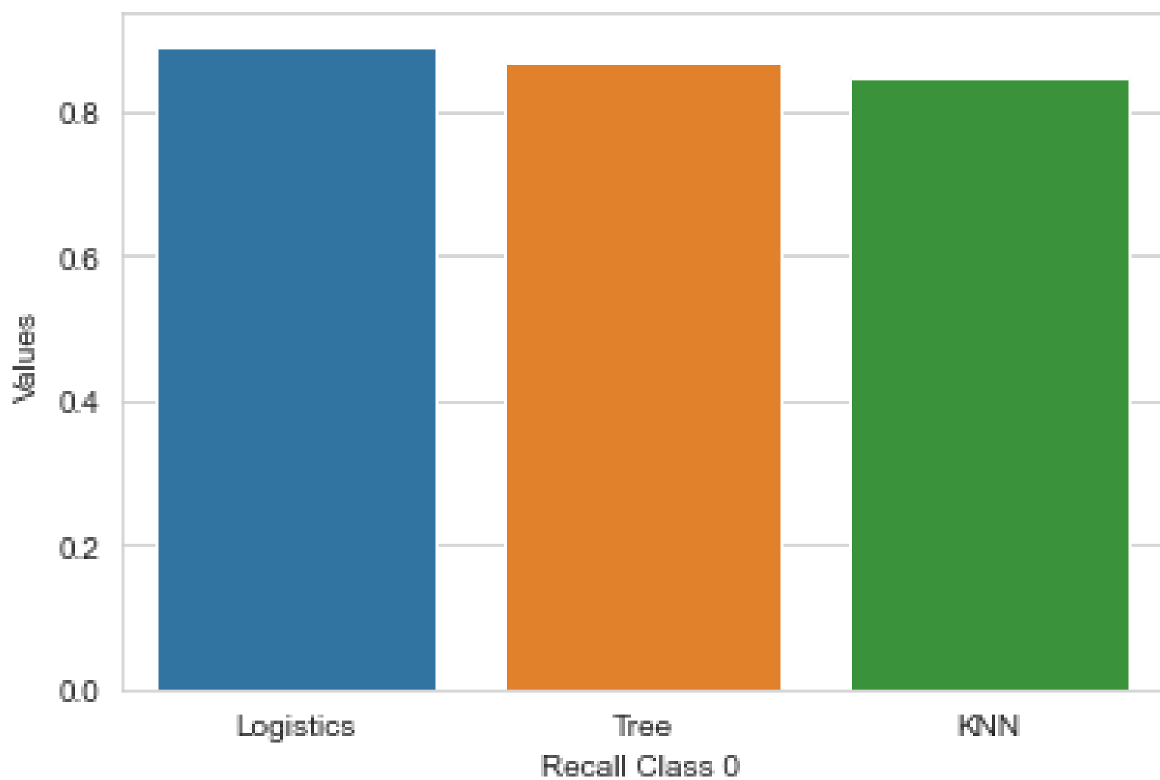


```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[0][1],
                                        prfs_tree[0][1], prfs_knn[0][1]],orient =
"vertical")
bar_plot.set(xlabel = "Precision Class 1", ylabel = "Values")
```

```
[Text(0.5, 0, 'Precision Class 1'), Text(0, 0.5, 'Values')]
```

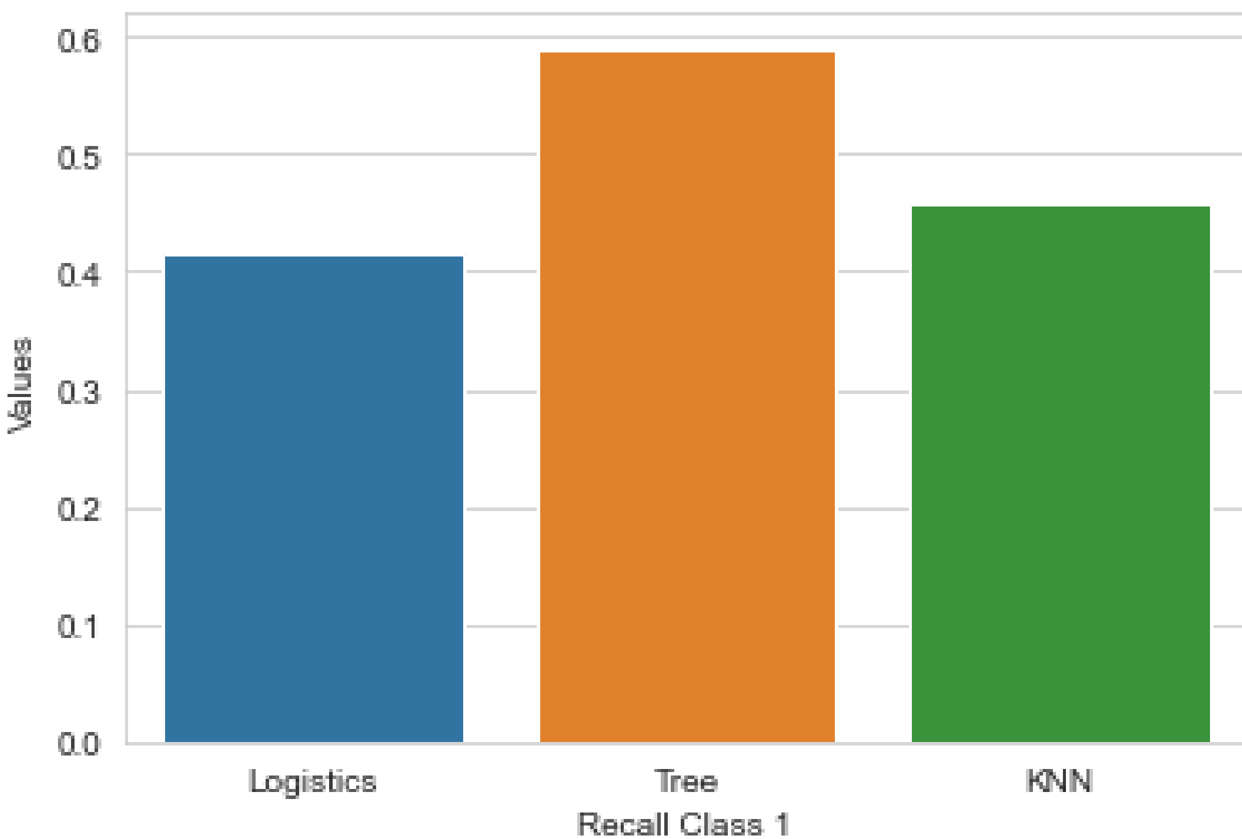


```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[1][0],
                                              prfs_tree[1][0], prfs_knn[1][0]],orient =
"vertical")
bar_plot.set(xlabel = "Recall Class 0", ylabel = "Values")
```



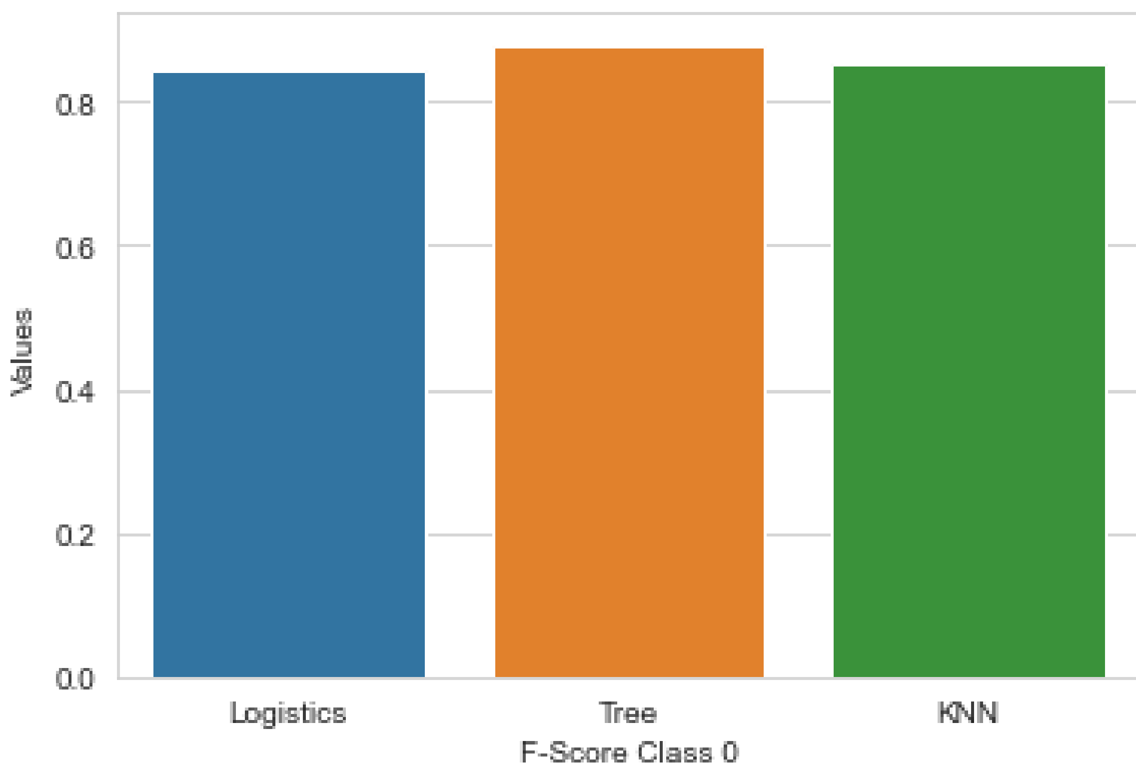
```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[1][1],
                                    prfs_tree[1][1], prfs_knn[1][1]],orient =
"vertical")
bar_plot.set(xlabel = "Recall Class 1", ylabel = "Values")
```

```
[Text(0.5, 0, 'Recall Class 1'), Text(0, 0.5, 'Values')]
```



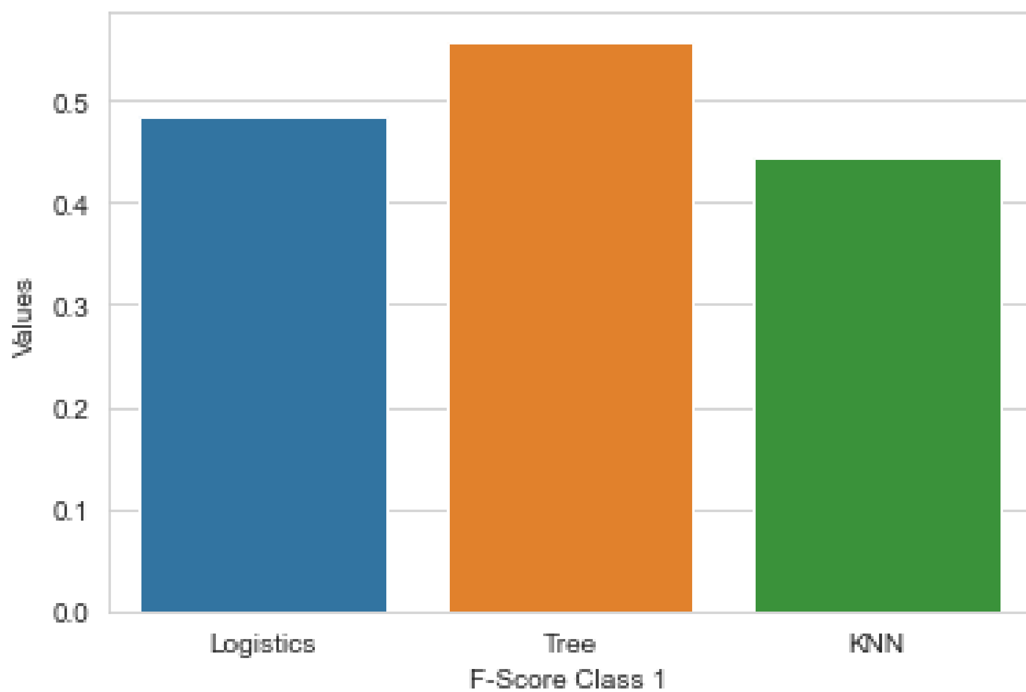
```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[2][0],
                                                prfs_tree[2][0], prfs_knn[2][0]],orient =
"vertical")
bar_plot.set(xlabel = "F-Score Class 0", ylabel = "Values")
```

```
[Text(0.5, 0, 'F-Score Class 0'), Text(0, 0.5, 'Values')]
```



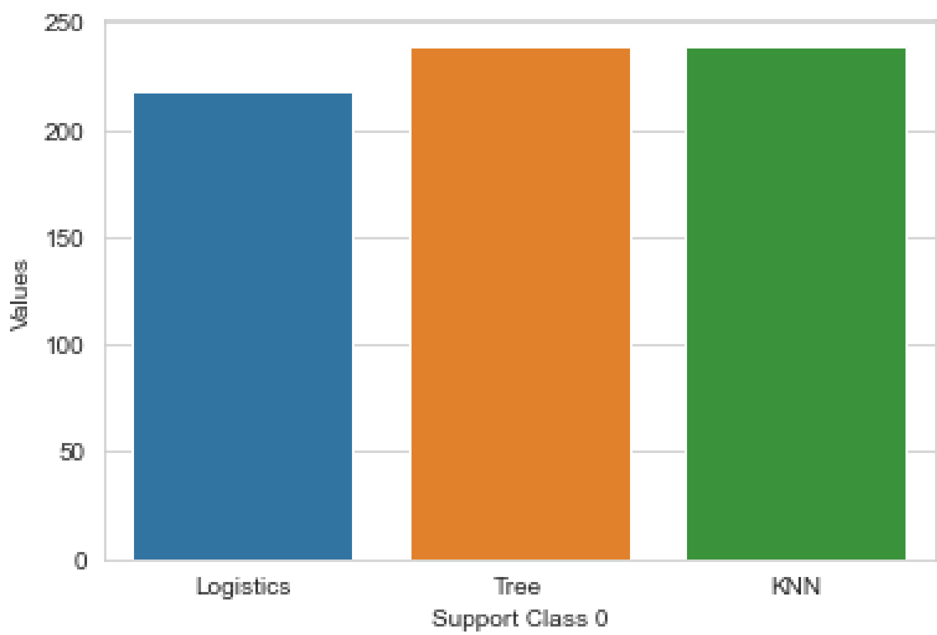
```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[2][1],
                                                  prfs_tree[2][1], prfs_knn[2][1]],orient =
"vertical")
bar_plot.set(xlabel = "F-Score Class 1", ylabel = "Values")
```

```
[Text(0.5, 0, 'F-Score Class 1'), Text(0, 0.5, 'Values')]
```



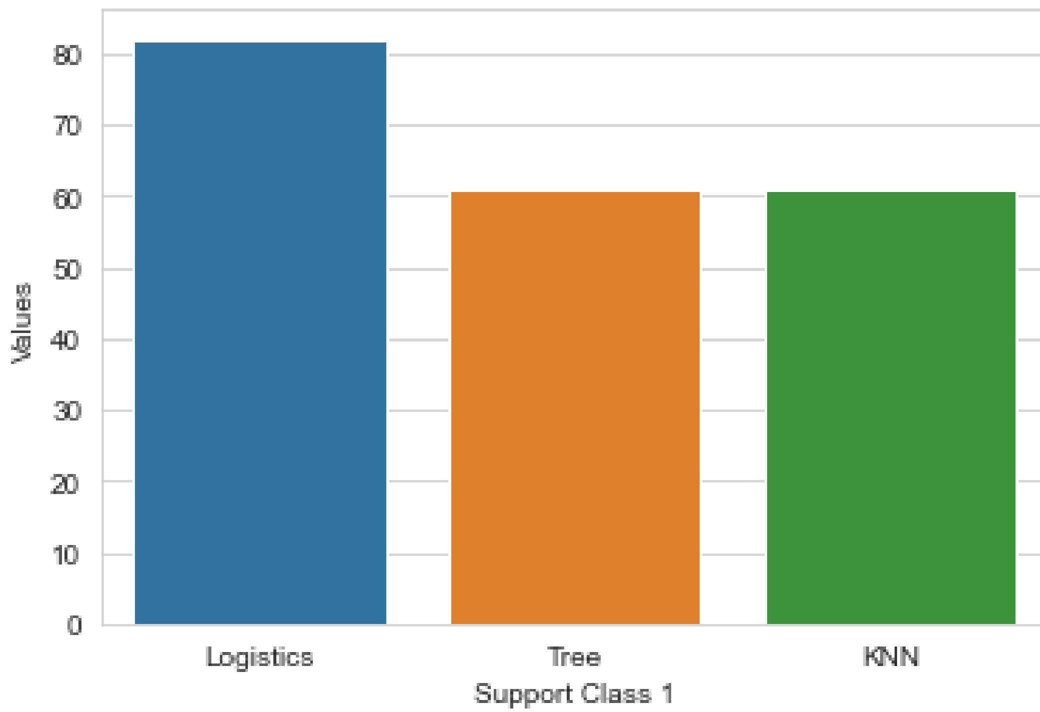
```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[3][0],
                                        prfs_tree[3][0], prfs_knn[3][0]],orient =
"vertical")
bar_plot.set(xlabel = "Support Class 0", ylabel = "Values")
```

```
[Text(0.5, 0, 'Support Class 0'), Text(0, 0.5, 'Values')]
```



```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y = [prfs_logistics[3][1],
                                        prfs_tree[3][1], prfs_knn[3][1]],orient =
"vertical")
bar_plot.set(xlabel = "Support Class 1", ylabel = "Values")
```

```
[Text(0.5, 0, 'Support Class 1'), Text(0, 0.5, 'Values')]
```

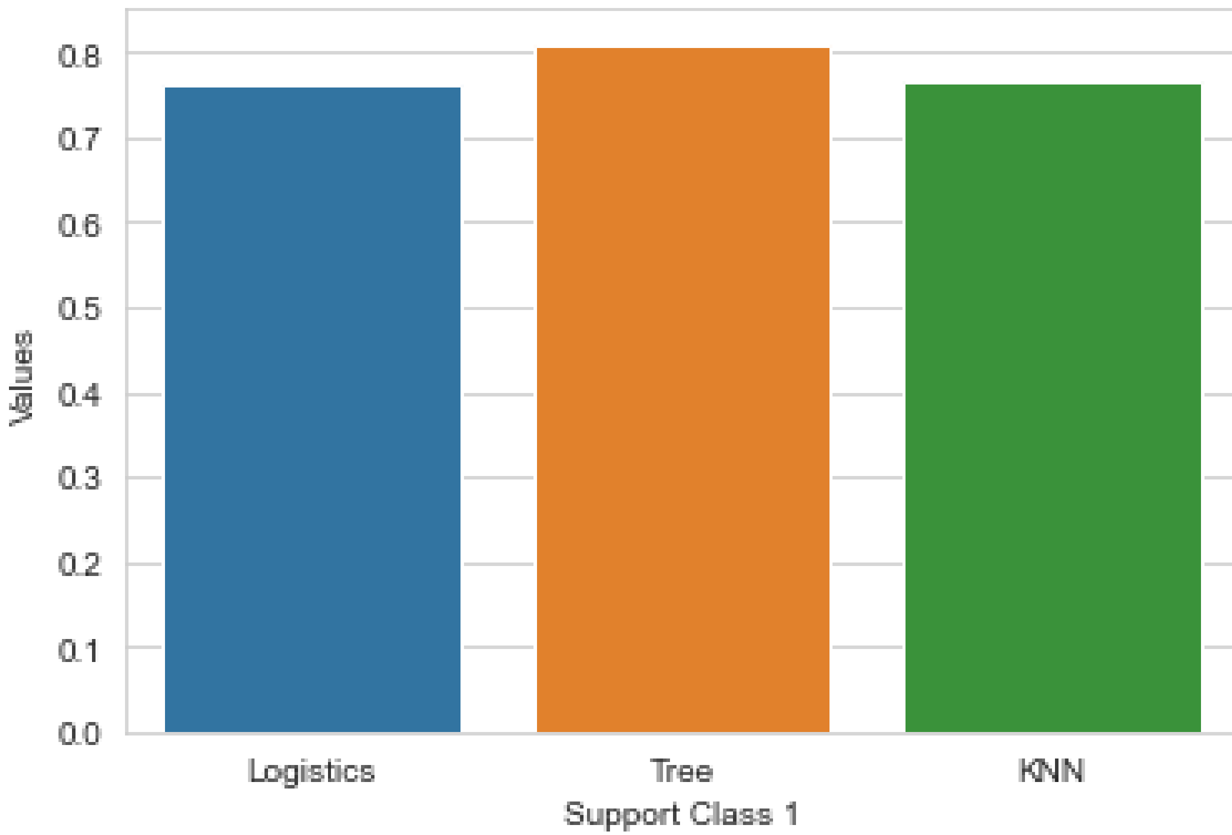


```
bar_plot = sns.barplot(x = ["Logistics","Tree", "KNN"],y =
[metrics.accuracy_score(y_test_logistics,prediction_logistics),

metrics.accuracy_score(y_test_tree,prediction_tree),

metrics.accuracy_score(y_test_knn,prediction_knn)],
                                        orient = "vertical")
bar_plot.set(xlabel = "Support Class 1", ylabel = "Values")
print("Accuracy of Logistic Regression model: ",
metrics.accuracy_score(y_test_logistics,prediction_logistics))
print("Accuracy of Decision Tree model: ",
metrics.accuracy_score(y_test_tree,prediction_tree))
print("Accuracy of K Nearest Neighbour: ",
round(metrics.accuracy_score(y_test_knn,prediction_knn), 2))
```

```
Accuracy of Logistic Regression model:  0.76
Accuracy of Decision Tree model:  0.81
Accuracy of K Nearest Neighbour:  0.77
```



So we will be accepting Decision Tree model as it is having maximum accuracy among other.

Future Scope of Improvements

More machine learning models including the Random Forest classifier will be added to this model. Accuracy should be compared again with other models to give a better understanding of the data.

More data processing is needed to increase its accuracy. We have to penalize some of the columns with some value because all columns don’t have same correlation.

We can also look for outliers in some columns which might be causing degrade in accuracy.

We will be using more different classification models in the more processed data and try to boost the accuracy to 90%.

Certificate

This is to certify that Mr/Ms **Soumyadeep Maji** of Asansol Engineering College, registration number: 181080110322, has successfully completed a project on Strokes Prediction using Machine Learning in Python under the guidance of Mr/Ms/Mrs Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Asansol Engineering College

Certificate

This is to certify that Mr/Ms **Arup Maji** of Asansol Engineering College, registration number: 181080110240, has successfully completed a project on Strokes Prediction using Machine Learning in Python under the guidance of Mr/Ms/Mrs Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Asansol Engineering College

Certificate

This is to certify that Mr/Ms **Arkya Patwa** of Asansol Engineering College, registration number: 181080110238, has successfully completed a project on Strokes Prediction using Machine Learning in Python under the guidance of Mr/Ms/Mrs Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Asansol Engineering College

Certificate

This is to certify that Mr/Ms **Amit Kumar Hazra** of Asansol Engineering College, registration number: 012214, has successfully completed a project on Strokes Prediction using Machine Learning in Python under the guidance of Mr/Ms/Mrs Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Asansol Engineering College