SOLENT UNIVERSITY

BSc (Hons) [Year Three]

Academic Year 2022-2023

Nastaran Sharifi Sadr

Data Science (COM 618)

Predicting Brain Stroke

Report

The video of my video link:
https://ssu-my.sharepoint.com/:f:/g/personal/4sadrn08_solent_ac_uk/Eu1OgIgalh9EoxDxh1WgMasB5p4swKjntC2mOHX1jwxPjA?email=drishty.sobnath%40solent.ac.uk&e=FgagCK

Supervisor:Dr Drishty Sobnath January 2023

Contents

Introduction

Every Year 79500 people in the USA have a brain stroke and about 137000 of these people die after the brain stroke and about 185000 of the rest if they survive, they will have again another brain stroke in five years [Nichd,2022].
Based on these statistics, finding a treatment method, in order to quickly diagnose a stroke, seems necessary for the healthcare section in the world. Today with using artificial intelligence, doctors can detect the risk of stroke much faster than before, as the key formula in treating stroke is time.
The advantages of using AI in healthcare system are, ability to analyzing data faster than human and improving diagnosis, automating many routine tasks and spending less time as the result, monitoring healthcare information about the person using wearable health Tec etc. [Chg-meridian, n.d.]
Also, there some disadvantage of using AI in healthcare, such as training complications, and difficulty in change[Chg-meridian, n.d.].

Definition of brain stroke

Brain stroke happens when the blood flow of a part of the brain reduced or paused and then it is difficult for oxygen to reach the brain and as a result the brain cells die in few minutes [Mayoclinic, n.d] . There are some symptoms of brain stroke [Mayoclinic, n.d]:

1. Trouble speaking and understanding what other people are saying
2. Having Headache
3. Problems seeing in one or both eyes
4. Trouble walking

There are two main causes of stroke [Mayoclinic, n.d], In the first case, one of the arteries in the brain is blocked, which is called an ischemic stroke. In the second case, one of the arteries of the brain is ruptured or blood leaks from it, which is called hemorrhagic. Also, some patients may experience a temporary disruption of blood supply to the brain, which is known as a transient ischemic attack. This condition will not cause long-term symptoms [Mayoclinic, n.d].

Aim and objective

The aim of this research is to find correlation between a person's lifestyle and having brain stroke. For this aim, the data will be analyzed, and different algorithms will be applied on the data and all the result will be compared to choose an algorithm that can predict the brain stroke with the highest accuracy.

Description of data source

The dataset of this research is from Kaggle [Izzet, 2022]. It is contains several characteristics that are results of a research. These characteristics are used to predict whether a person is likely to get a brain stroke depends on gender, age, hypertension, heart disease, work type etc.

The details of these characteristic shown in the table below:

Variable	Description	Variable Type	categories	Data Type
Gender	It is describes male/female or others	Independent	3 categories	Categorical
Age	It describes the age of patient	Independent	It doesn't have any category.	Numerical
Hypertension	If the patient has hypertension=1, and if the patient doesn't have hypertension=0	Independent	2 categories	Categorical
Heart Disease	If the patient has heart disease=1, and if the patient doesn't have heart disease=0	Independent	2 categories	Categorical
Ever Married	If the patient ever married=1, And if not=0	Independent	2 categories	Categorical
Work Type	It shows the type of work: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"	Independent	5 categories	Categorical
Residence Type	If the patient's residence type is "Rural" or "Urban"	Independent	2 categories	Categorical
Avg Glucose Level	It shows average glucose level in blood.	Independent	It doesn't have any category.	Numerical
Bmi	body mass index	Independent	It doesn't have any category.	Numerical
Smoking Status	It shows how often the patient smokes: "formerly smoked", "never smoked", "smokes" or "Unknown"*	Independent	4 categories	Categorical

Stroke	If the patient has stroke=1, and the patient doesn't have stroke=0	Dependent (As the aim is to predict stroke depends on other variables)	2 categories	Categorical

Exploratory data analysis (EDA)

It means that we collect and prepare the available data in a summary and use variety of different methods to analyze them.[IBM, 2020]. It is include Bivariate and univariate analyses, dropping missing data, finding duplicates rows, investigating data types etc. For this reason, is needed to first load the dataset.

Loading The Dataset

For loading and exploring the dataset, Pandas library has been used as panadas is the fastest tool to analyzing data [Pandas, 2023]. Read_CSV from pandas has been used to loading the data. The data has 4981 rows and 11 columns.

```
from pandas import read_csv
#reading the CSV file
my_data=pd.read_csv(r'C:\university\second semeaster\Data Science\Brain Stroke prediction project\Full_data.csv')
print("The data shape is:")
print(my_data.shape)
```

```
The data shape is:
(4981, 11)
```

The image below shoes the first five rows in the dataset with index 0 to 4.

```
print("Five first rows of data")
print(my_data.head(5))
```

```
Five first rows of data
   gender   age  hypertension  heart_disease ever_married      work_type Residence_type  avg_glucose_level   bmi   smoking_s
tatus  stroke
0    Male  67.0             0              1          Yes        Private          Urban             228.69  36.6  formerly s
moked       1
1    Male  80.0             0              1          Yes        Private          Rural             105.92  32.5     never s
moked       1
2  Female  49.0             0              0          Yes        Private          Urban             171.23  34.4           s
mokes       1
3  Female  79.0             1              0          Yes  Self-employed          Rural             174.12  24.0     never s
moked       1
4    Male  81.0             0              0          Yes        Private          Urban             186.21  29.0  formerly s
moked       1
```

The image below shows the last five rows in the dataset. It is obvious that the last index is 4980 as we have 4981 rows and the first index started from zero.

```
print("five last rows of the data")
print(my_data.tail(5))
```

```
five last rows of the data
      gender   age  hypertension  heart_disease ever_married work_type Residence_type  avg_glucose_level   bmi   smoking_sta
tus  stroke
4976    Male  41.0             0              0           No   Private          Rural              70.15  29.8  formerly smo
ked       0
4977    Male  40.0             0              0          Yes   Private          Urban             191.15  31.1           smo
kes       0
4978  Female  45.0             1              0          Yes  Govt_job          Rural              95.02  31.8           smo
kes       0
4979    Male  40.0             0              0          Yes   Private          Rural              83.94  30.0           smo
kes       0
4980  Female  80.0             1              0          Yes   Private          Urban              83.75  29.1     never smo
ked       0
```

With using info() function from pandas help to get information about the dataset. The result shows that gender, ever_maried, work_type, residence_type and smiking_status are variables with object types. We may need to change them to numeric types as the algorithms work with numeric types.

```
print(my_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4981 entries, 0 to 4980
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             4981 non-null   object
 1   age                4981 non-null   float64
 2   hypertension       4981 non-null   int64
 3   heart_disease      4981 non-null   int64
 4   ever_married       4981 non-null   object
 5   work_type          4981 non-null   object
 6   Residence_type     4981 non-null   object
 7   avg_glucose_level  4981 non-null   float64
 8   bmi                4981 non-null   float64
 9   smoking_status     4981 non-null   object
 10  stroke             4981 non-null   int64
dtypes: float64(3), int64(3), object(5)
memory usage: 428.2+ KB
None
```

Also, describe() method from pandas has been used to get extra information about the dataset. For example, with looking at the values of count row for each variable, we can guess that there is not any missing value as the total number of rows is 4981 too.

```
my_data.describe()
```

]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
count	4981.00	4981.00	4981.00	4981.00	4981.00	4981.00	4981.00	4981.00	4981.00	4981.00	4981.00
mean	0.58	43.42	0.10	0.06	0.34	0.83	0.49	105.94	28.50	1.58	0.05
std	0.49	22.66	0.29	0.23	0.47	1.10	0.50	45.08	6.79	1.09	0.22
min	0.00	0.08	0.00	0.00	0.00	0.00	0.00	55.12	14.00	0.00	0.00
25%	0.00	25.00	0.00	0.00	0.00	0.00	0.00	77.23	23.70	1.00	0.00
50%	1.00	45.00	0.00	0.00	0.00	0.00	0.00	91.85	28.10	1.00	0.00
75%	1.00	61.00	0.00	0.00	1.00	2.00	1.00	113.86	32.60	3.00	0.00
max	1.00	82.00	1.00	1.00	1.00	3.00	1.00	271.74	48.90	3.00	1.00

Checking the dataset for missing values

Missing data can affect the result and bias the data specially when the goal is building a biased machine learning model [Pritha B, 2022]. Also, missing data can affect on the accuracy of the model [Nasima T, 2022]. So, missing value first is needed to find and then depends on the conditions whether drop the rows with missing values or replacing them with central tendency.

At the first view, it doesn't show any missing values. But we know that we have five variables with type of object. So, it is better to change those variables types to numeric.

```
#printing rows containing empty variables
empty_data=my_data[my_data.isna().any(axis=1)]
print("#####Missing data########")
print(empty_data)
#It doesn't show any missing data

print(my_data.dtypes)
```

```
#####Missing data########
Empty DataFrame
Columns: [gender, age, hypertension, heart_d
_status, stroke]
Index: []
gender                object
age                  float64
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

Mapping categories to numbers

For mapping all category variables to numbers is needed to first get the categories information. with unique method from panadas we can find them.

```
print(my_data['gender'].unique())
print(my_data['ever_married'].unique())
print(my_data['work_type'].unique())
print(my_data['Residence_type'].unique())
print(my_data['smoking_status'].unique())
```

```
['Male' 'Female']
['Yes' 'No']
['Private' 'Self-employed' 'Govt_job' 'children']
['Urban' 'Rural']
['formerly smoked' 'never smoked' 'smokes' 'Unknown']
```

Now, we can map each category to a number. For example, instead of male, would be 0 and instead of female would be 1.

```
my_data['gender'][my_data['gender'] == 'Male'] = 0
my_data['gender'][my_data['gender'] == 'Female'] = 1

my_data['ever_married'][my_data['ever_married'] == 'Yes'] = 0
my_data['ever_married'][my_data['ever_married'] == 'No'] = 1

my_data['work_type'][my_data['work_type'] == 'Private'] = 0
my_data['work_type'][my_data['work_type'] == 'Self-employed'] = 1
my_data['work_type'][my_data['work_type'] == 'Govt_job'] = 2
my_data['work_type'][my_data['work_type'] == 'children'] = 3

my_data['Residence_type'][my_data['Residence_type'] == 'Urban'] = 0
my_data['Residence_type'][my_data['Residence_type'] == 'Rural'] = 1

my_data['smoking_status'][my_data['smoking_status'] == 'formerly smoked'] = 0
my_data['smoking_status'][my_data['smoking_status'] == 'never smoked'] = 1
my_data['smoking_status'][my_data['smoking_status'] == 'smokes'] = 2
my_data['smoking_status'][my_data['smoking_status'] == 'Unknown'] = 3


my_data.head()
```

Image below shows head of dataset after mapping categories to numbers.

]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	67.0	0	1	0	0	0	228.69	36.6	0	1
1	0	80.0	0	1	0	0	1	105.92	32.5	1	1
2	1	49.0	0	0	0	0	0	171.23	34.4	2	1
3	1	79.0	1	0	0	1	1	174.12	24.0	1	1
4	0	81.0	0	0	0	0	0	186.21	29.0	0	1

Now we again check the dataset with missing values. As, we can see even after changing the object variables to numbers, still there is not any missing values in the dataset.

```
57]:
my_data.isnull().sum()
my_data.isna().any()

#we use this code if it had missing values
#use the simpleImputer function to replace missing values
#imputer=SimpleImputer(strategy='mean')
#imputer.fit(my_data)
#new_data=imputer.transform(my_data)
#my_data=pd.DataFrame(new_data,columns=new_header_names)
```

```
t[157]: gender              False
age                 False
hypertension        False
heart_disease       False
ever_married        False
work_type           False
Residence_type      False
avg_glucose_level   False
bmi                 False
smoking_status      False
stroke              False
dtype: bool
```

Removing duplicate rows

Duplicate values mean that we have repeated values in the dataset. In this case, there is not any duplicate row in the dataset.

```
:
print(my_data.shape)
#Count duplicate in a DataFrame
print(my_data.duplicated().sum())

# Count the number of non-duplicates
#print(~my_data.duplicated().sum())


#my_data.drop_duplicates(inplace=True)
my_data.shape
#There is not any dublicate value
```

```
(4981, 11)
0
```

```
.58]: (4981, 11)
```

Changing data types from Object to Float

Before we mapped all category variables to numbers. But, even after that still data types of gender, ever_maried, work_type, residence_type and smoking status still are object. So, with to_numeric method from pandas library we can change those variables' types to numeric type.

```
print("Data types are: ")
print("dataset before changing objects to numeric")
print(my_data.dtypes)
print("-------------------------------------------------")
#print(my_data.dtypes)
my_data['gender']=pd.to_numeric(my_data['gender'], errors='coerce')
my_data['ever_married']=pd.to_numeric(my_data['ever_married'], errors='coerce')
my_data['work_type']=pd.to_numeric(my_data['work_type'], errors='coerce')
my_data['Residence_type']=pd.to_numeric(my_data['Residence_type'], errors='coerce')
my_data['smoking_status']=pd.to_numeric(my_data['smoking_status'], errors='coerce')


print("dataset after changing objects to numeric")
print(my_data.dtypes)
print("-------------------------------------------------")

my_data=my_data.applymap(np.float)

print("dataset after changing int to float")
print(my_data.dtypes)
print("-------------------------------------------------")
```

After changing all values to Numeric, we know that some variable's types are int and some of them are float. To increasing the performance of algorithms it is recommended to change all variables to one type, for example, changing int to float.

```
Data types are:
dataset before changing objects to numeric
gender               object
age                 float64
hypertension          int64
heart_disease         int64
ever_married         object
work_type            object
Residence_type       object
avg_glucose_level   float64
bmi                 float64
smoking_status       object
stroke                int64
dtype: object
-------------------------------------------------
dataset after changing objects to numeric
gender                int64
age                 float64
hypertension          int64
heart_disease         int64
ever_married          int64
work_type             int64
Residence_type        int64
avg_glucose_level   float64
bmi                 float64
smoking_status        int64
stroke                int64
dtype: object
```

So, now we have all variables with float type.

```
-----------------------------------------------
dataset after changing int to float
gender                float64
age                   float64
hypertension          float64
heart_disease         float64
ever_married          float64
work_type             float64
Residence_type        float64
avg_glucose_level     float64
bmi                   float64
smoking_status        float64
stroke                float64
dtype: object
```

Mean, Median, Mode

Mean is average of each column; Mode is the most common number in a column and Median is the middle number in a set[Neil,2022].

```
print("------Median in csv fil:------")
print(my_data.median(numeric_only=True))

print("------Mean in csv fil:-------")
print(my_data.mean(numeric_only=True))

print("------Mode in csv fil:-------")
print(my_data.mode(numeric_only=True))
```

```
gender                 1.00
age                   45.00
hypertension           0.00
heart_disease          0.00
ever_married           0.00
work_type              0.00
Residence_type         0.00
avg_glucose_level     91.85
bmi                   28.10
smoking_status         1.00
stroke                 0.00
dtype: float64
------Mean in csv fil:-------
gender                 0.58
age                   43.42
hypertension           0.10
heart_disease          0.06
ever_married           0.34
work_type              0.83
Residence_type         0.49
avg_glucose_level    105.94
bmi                   28.50
smoking_status         1.58
stroke                 0.05
dtype: float64
------Mode in csv fil:-------
   gender   age  hypertension  heart_disease  ever_married  work_type  Residence_type  avg_glucose_level   bmi  smoking_stat
us  stroke
0     1.0  78.0           0.0            0.0           0.0        0.0             0.0              93.88  28.7
1.0     0.0
```

Skewness

Skewness indicates the degree of asymmetry of the probability distribution of data around the mean. Skewness value can be negative or positive. We can see that none of our data is normally distributed as skewness for none of them is zero. Or for example for age and gender we know that more weight is in the right as their skewness is negative. On the other hand, avg_glucose_level and smiking_status have positive skewness and it means more weight on the left of the distribution. [James C., 2022]

```
#checking for skewness in cryptos data
print(my_data.skew())
print("-----------------------------")
```

```
gender              -0.34
age                 -0.14
hypertension         2.74
heart_disease        3.90
ever_married         0.67
work_type            0.96
Residence_type       0.03
avg_glucose_level    1.59
bmi                  0.37
smoking_status       0.08
stroke               4.14
dtype: float64
-----------------------------
```

Kurtosis

Kurtosis is the height of distribution, and data that is normally distributed has kurtosis equal 3 [Will,2022].

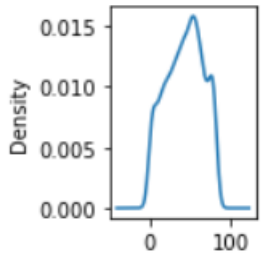
```
import scipy.stats as stats

from scipy.stats import kurtosis
kurtosis(my_data)
```

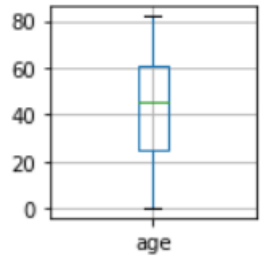
```
]: array([-1.88, -1.  ,  5.51, 13.17, -1.55, -0.6 , -2.  ,  1.75, -0.14,
       -1.35, 15.14])
```

Univariate analysis

Univariate analysis is type of analysis that we analyze only one variable[Statisticshowto , n.d].
With analyzing age variable, it is visible that age is normally distributed and there are not outliers for this variables.

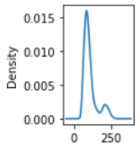
```
#visualising our data

#pd.set_option('display.width',1000)
#pd.set_option('display.precision',2)
my_data["age"].plot.kde()
plt.show()
#Age is normally distributed

my_data.boxplot(column=["age"])

plt.rcParams["figure.figsize"] = [5,5]
plt.rcParams["figure.autolayout"] = True

plt.show()
#age doesn't have any outliers
```

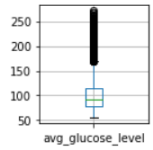




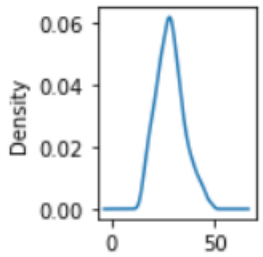
On the other hand, avg_gloucose_level has outliers and Bmi has outliers. Also, Bmi is normally distributed.

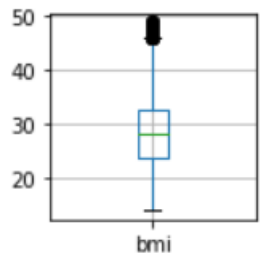
```
my_data["avg_glucose_level"].plot.kde()
plt.show()

my_data.boxplot(column=["avg_glucose_level"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```





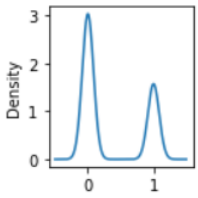

```
my_data["bmi"].plot.kde()
plt.show()

my_data.boxplot(column=["bmi"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```

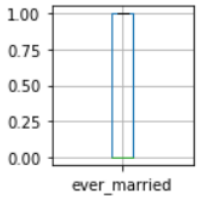




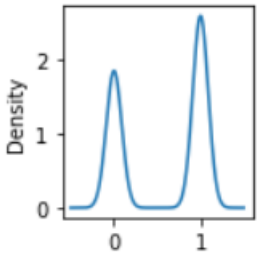
With looking at ever_married variable we know the number of people who married are more than those who haven't married. Also there is not outliers.

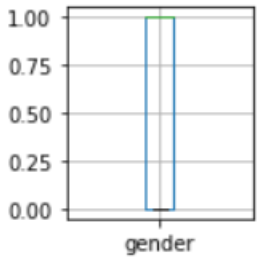
```
my_data["ever_married"].plot.kde()
plt.show()

my_data.boxplot(column=["ever_married"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



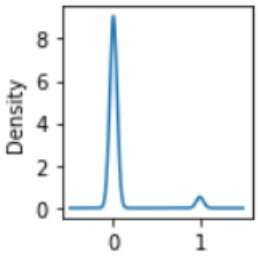


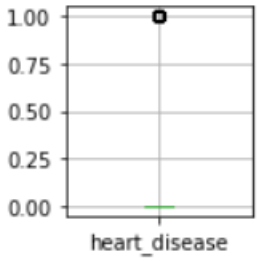

```
my_data["gender"].plot.kde()
plt.show()
my_data.boxplot(column=["gender"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



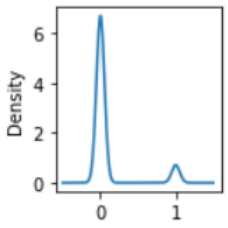


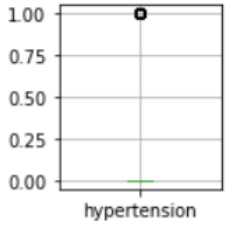
```
my_data["heart_disease"].plot.kde()
plt.show()

my_data.boxplot(column=["heart_disease"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



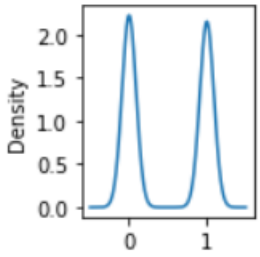


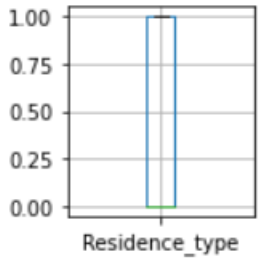

```
my_data["hypertension"].plot.kde()
plt.show()

my_data.boxplot(column=["hypertension"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



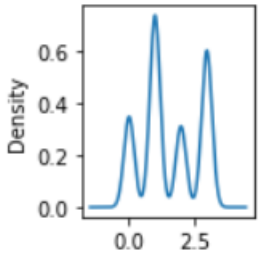


```
my_data["Residence_type"].plot.kde()
plt.show()


my_data.boxplot(column=["Residence_type"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```

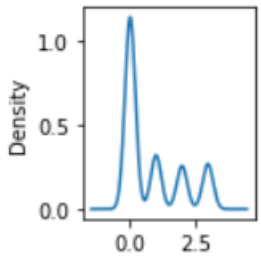


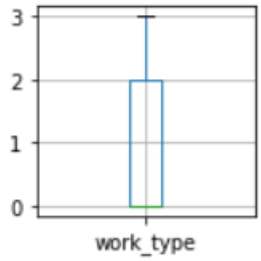



```
my_data["smoking_status"].plot.kde()
plt.show()

my_data.boxplot(column=["smoking_status"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



```
my_data["work_type"].plot.kde()
plt.show()

my_data.boxplot(column=["work_type"])
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```

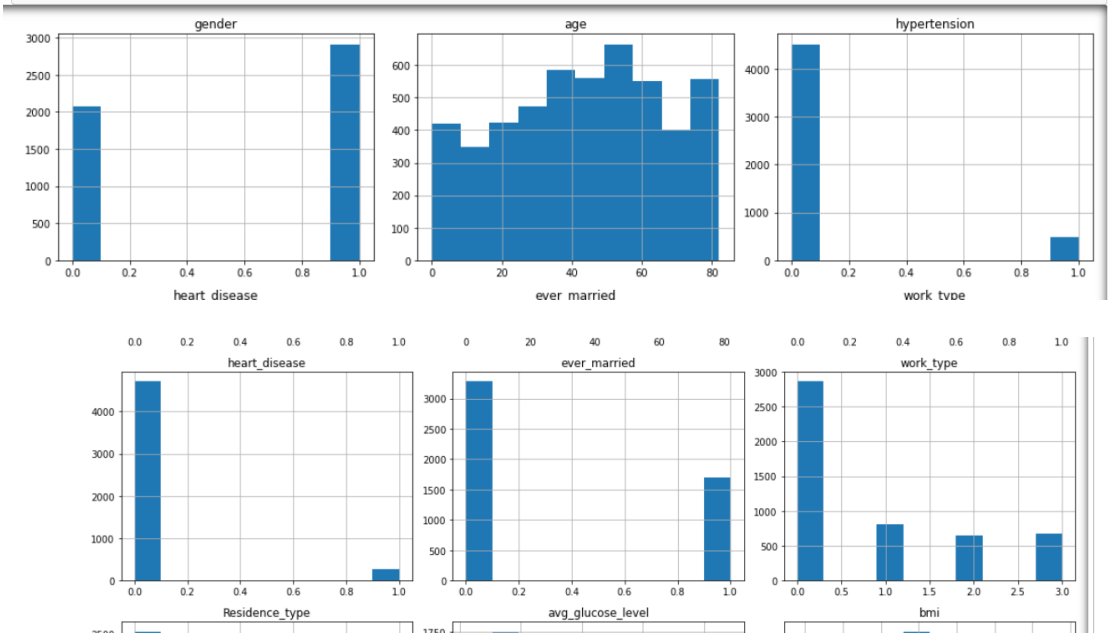


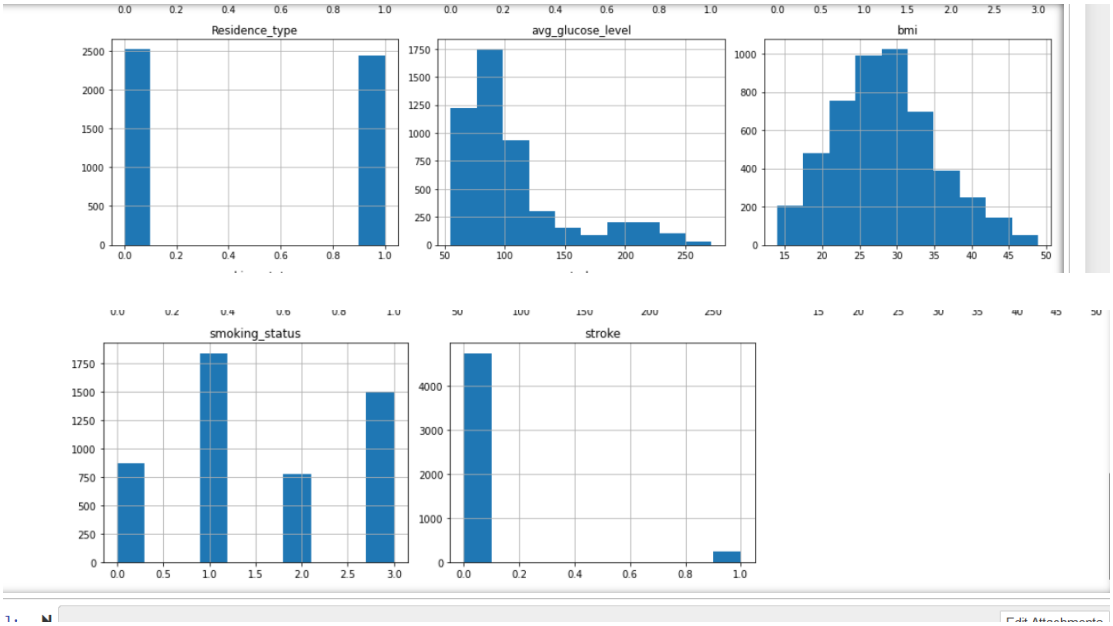


Another way to checking distribution of data is drawing histogram charts. It is visible that Bmi and age are normally distributed.

Edit Attachments

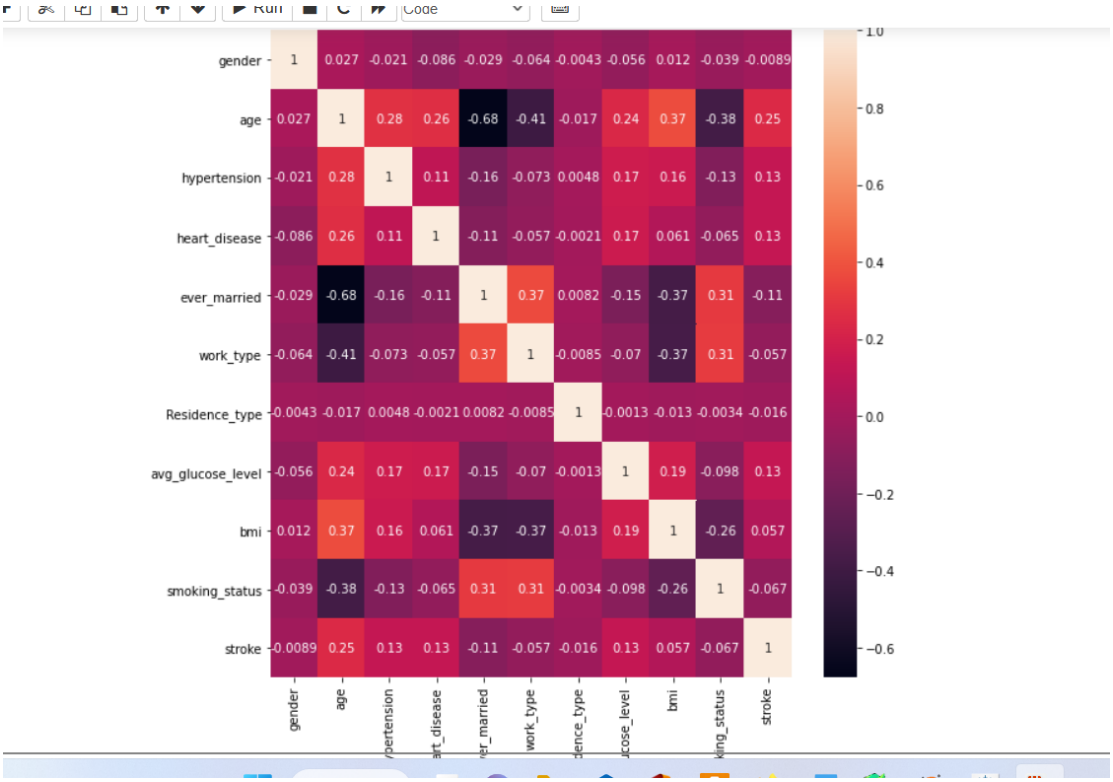
```
#visualising our data for ticker
pd.set_option('display.width',1000)
my_data.hist()
plt.rcParams["figure.figsize"] = [9,9]
plt.rcParams["figure.autolayout"] = True
plt.show()
```





Bivariate analysis

This analysis, work on more than one variables and their relationships. First let's see the correlation between stroke and other variables. Correlation shows which two variables are linearly related[Jmp, n.d]. correlation image shows that age,hypertension, heart disease and avarage glucose leves are correlated with stroke.



]:

```
#Create a Correlation Matrix using Pandas
corrMatrix_for_all=my_data.corr()
#print(corrMatrix_for_all)
#Get a Visual Representation of the Correlation Matrix using Seaborn and Matplotlib
import seaborn as sn
import matplotlib.pyplot as plt

sn.heatmap(corrMatrix_for_all, annot=True)
plt.rcParams["figure.figsize"] = [13,13]
plt.rcParams["figure.autolayout"] = True
plt.show()
```

397]:

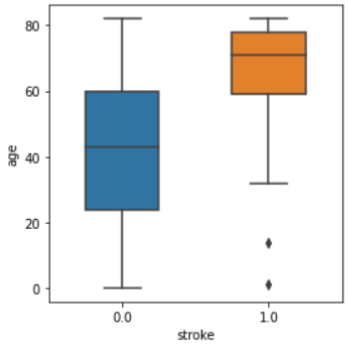
```
corrMatrix_for_stroke=my_data.corr()[['stroke']]
corrMatrix_for_stroke.style.background_gradient(cmap='YlOrRd').set_precision(2)
```

Out[397]:

	stroke
gender	-0.01
age	0.25
hypertension	0.13
heart_disease	0.13
ever_married	-0.11
work_type	-0.06
Residence_type	-0.02
avg_glucose_level	0.13
bmi	0.06
smoking_status	-0.07
stroke	1.00

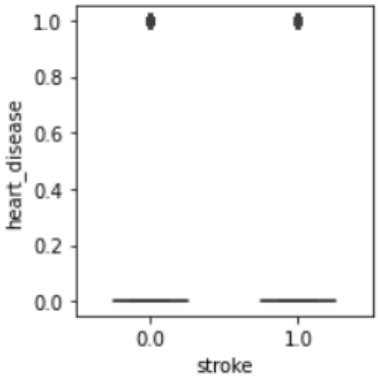
The image below shows relationship between age and stroke. It is visible there is more stroke after age 60.

```
#Bivariate age and stroke
import seaborn as sns
ax=sns.boxplot(x='stroke', y='age', data=my_data, width=0.5)
ax.set(xlabel='stroke', ylabel='age')
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```

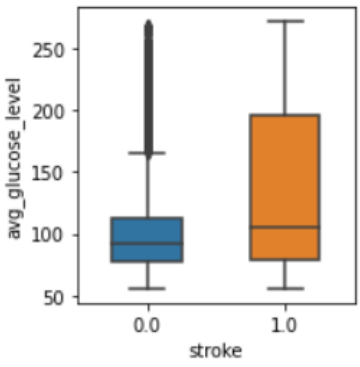


The image below shows relationship between heart disease and stroke.

```
#Bivariate heart disease and stroke
import seaborn as sns
ax=sns.boxplot(x='stroke',y='heart_disease', data=my_data, width=0.5)
ax.set(xlabel='stroke', ylabel='heart_disease')
plt.rcParams["figure.figsize"] = [10, 10]
plt.rcParams["figure.autolayout"] = True
plt.show()
```

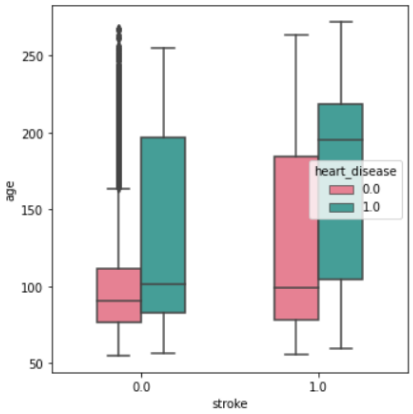



```
#Bivariate avg_glucose_level and stroke
import seaborn as sns
ax=sns.boxplot(x='stroke', y='avg_glucose_level', data=my_data, width=0.5)
ax.set(xlabel='stroke', ylabel='avg_glucose_level')
plt.rcParams["figure.figsize"] = [2,2]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



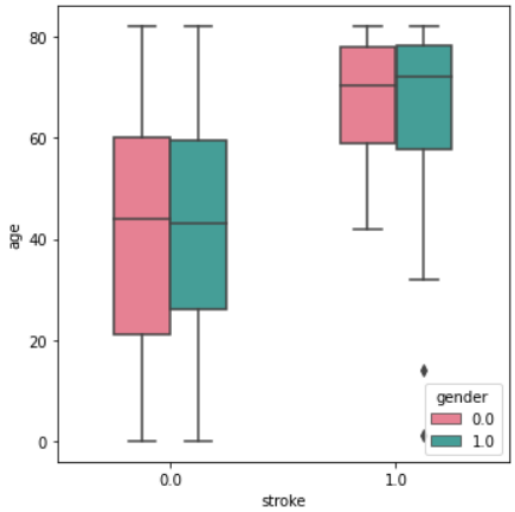
The image below shows almost affection of heart disease and average glucose level on having stroke is equal.

```
#Bivariate heart disease and stroke and avg-#glucose_level
import seaborn as sns
ax=sns.boxplot(x='stroke', y='avg_glucose_level' , hue='heart_disease', data=my_data, width=0.5, palette = 'husl')
ax.set(xlabel='stroke', ylabel='age')
plt.rcParams["figure.figsize"] = [8,8]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



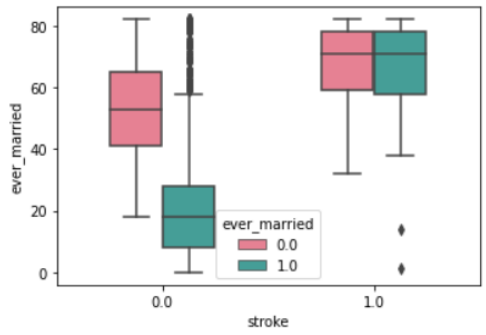
The image below shows woman more than 60 years old are most likely to have stroke.

```
#age, stroke and gender
ax=sns.boxplot(x='stroke', y='age' , hue='gender', data=my_data, width=0.5, palette = 'husl')
ax.set(xlabel='stroke', ylabel='age')

plt.rcParams["figure.figsize"] = [3,3]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



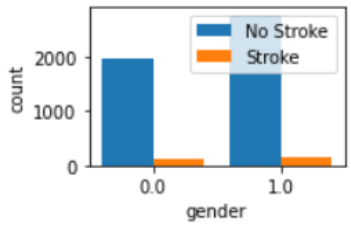
The Image below shows that no matter if the person married or not, but age 60 they are most likely to have stroke.

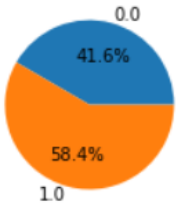
```
#age, stroke and ever_married
ax=sns.boxplot(x='stroke', y='age' , hue='ever_married', data=my_data, width=0.5, palette = 'husl')
ax.set(xlabel='stroke', ylabel='ever_married')
plt.rcParams["figure.figsize"] = [5, 3.50]
plt.rcParams["figure.autolayout"] = True
plt.show()
```



And the final image shows the percentage of woman who gets stoke is more than men.

```
#gender and stroke
ax=sns.countplot(x='gender', data=my_data,hue='stroke',saturation=1, dodge=True)

plt.legend(['No Stroke','Stroke'])

plt.rcParams["figure.figsize"] = [4,2]
plt.xlabel('gender')
plt.show()

my_data.groupby('gender').size().plot(kind='pie',autopct='%.1f%%')
plt.ylabel('')
plt.show()
```





Data Modeling

As the target is predicting brain stroke, so, decided to check Decision tree classifier on the data. Decision tree is on of supervised learning algorithms that predict categories and the base of that is on if-else statements [Scikit_learn, 2007].
First the dataset is separated to 2 parts, on part is only target (stroke) and the other part is the rest of independent variables.

```
###data modeling#########
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
print(my_data.shape)
y=my_data.pop('stroke')
X=my_data
print(y.shape)
print(X.shape)
```

```
(4981, 11)
(4981,)
(4981, 10)
```

Then the dataset bis separated to train and test data (%60 train and %40 test data). Then with calling DecisionTreeClassifier method from sklearn.tree library the model will be created. Now one Instance will be predict using model and the model predict 1 that means the person will get stoke which is correct.

```
#split 60-40
#print(X)
#print(y)
#splite teh data into train and test  60:40
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=1)

#check the number of train and test data
print('\n The total of training dataset',X_train.shape)
print('\n The total of test dataset', X_test.shape)

#now we can instantiate the model for our model
#in this case the decision tree model is used
my_model= DecisionTreeClassifier()
#train teh model to fit
my_model.fit(X_train,y_train)

#now let's predict the model
y_pred_train=my_model.predict(X_train)
#print(y_pred_train)
y_pred=my_model.predict(X_test)
#print(y_pred)

#But let's make this more meaningful. Let's predict the output for a specific instance.
print('-------------')
pred_one=my_model.predict(X=[[0.0,67.0,0.0,1.0,0.0,0.0,0.0,228.69,36.6,0.0 ]])
print(pred_one)

#As you are aware that the target labels are 2 for benign, 4 for malignant

#view the test data
#X_test.join(y_test)
```

```
 The total of training dataset (2988, 10)

 The total of test dataset (1993, 10)
-------------
[1.]
```

Now, let's calculate the accuracy of the model. It is visible the accuracy of the train set is %100 and the accuracy of set is %91. It seems that model predict well but having high accuracy in the train set might be a sign of overfitting. Also, confusion matrix shows that model has 1797 True positive and 17 Trye negative and 95 False positive and 84 False negative.

```
#compute train set accuracy
model_accuracy_train=accuracy_score(y_train,y_pred_train)
print("Model accuracy on Train data:{:.2f}".format(model_accuracy_train), '\n')
#compute test set accuracy
model_accuracy_test=accuracy_score(y_test,y_pred)
print("Model accuracy on Test data:{:.2f}".format(model_accuracy_test), '\n')
#constructing confusion matrix:
matrix_info=confusion_matrix(y_test,y_pred)
print("The Confusion Matrix: \n", matrix_info,'\n')
#Construct the classification report
class_report=classification_report(y_test,y_pred)
print("Report of classification:\n",class_report)
#view the test data
X_test.join(y_test)
```

```
Model accuracy on Train data:1.00

Model accuracy on Test data:0.91

The Confusion Matrix:
 [[1797   95]
 [  84   17]]

Report of classification:
              precision    recall  f1-score   support

         0.0       0.96      0.95      0.95      1892
         1.0       0.15      0.17      0.16       101

    accuracy                           0.91      1993
   macro avg       0.55      0.56      0.56      1993
weighted avg       0.91      0.91      0.91      1993
```

The accuracy of test increased to %92 and there is a big different in confusion matrix. But still the accuracy of the train set is %100. The model has learned the dataset differently.

```
#print(X)
#print(y)
#splite teh data into train and test  70:30
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
```

```
Model accuracy on Train data:1.00

Model accuracy on Test data:0.92

The Confusion Matrix:
 [[1362   56]
 [  69    8]]

Report of classification:
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.96      1418
         1.0       0.12      0.10      0.11        77

    accuracy                           0.92      1495
   macro avg       0.54      0.53      0.53      1495
weighted avg       0.91      0.92      0.91      1495
```

Now a seed variable is added to replace the random state and, max depth and min samples leaf added. This time the accuracy of the test data is %95. But this does not guaranty that the models fit or generalized well.

```
#Declare a SEED variable and set to 1 to ensure reproductivity.
SEED=1
my_model= DecisionTreeClassifier(max_depth=4, min_samples_leaf=8, random_state=SEED)
#train teh model to fit
my_model.fit(X_train,y_train)
```

```
 The total of training dataset (3486, 10)

 The total of test dataset (1495, 10)
Model accuracy on Train data:0.95

Model accuracy on Test data:0.95

The Confusion Matrix:
 [[1418    0]
 [  77    0]]

Report of classification:
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97      1418
         1.0       0.00      0.00      0.00        77

    accuracy                           0.95      1495
   macro avg       0.47      0.50      0.49      1495
weighted avg       0.90      0.95      0.92      1495
```

So now let's try k-fold algorithm. k-fold algorithm is a technique that helps to know the model isn't overfit or underfit [Sajal,2020]. This technique split the data to k subsets and at the end there is k errors and at the end mean of CV(k-fold cross validation error) will be compared with the mean cv of training set[Sajal,2020]. The cross-validation mean is (90.9%) and the standard deviation is 0.013. It means the model overfits as train score is higher than cross-validation score. On the other hand, test score is almost equal to cross-validation score.

```
#in this case the decision tree model is used
my_model= DecisionTreeClassifier()
#train teh model to fit
my_model.fit(X_train,y_train)

#declare the num of folds
num_folds=KFold(n_splits=10, random_state=1,shuffle=True)
#test by changing n_splits to 3, 5 and 10
#compute the array containing the 10 folds and calculate the cros validation mean score
CV_scores=-cross_val_score(my_model,X_train, y_train, cv=num_folds)
print("\nCross Val mean: {:.3f} (std: {:.3f})".format(CV_scores.mean()*-1,CV_scores.std()),end="\n\n" )
#now let's predict the model
y_pred_train=my_model.predict(X_train)
#print(y_pred_train)
y_pred=my_model.predict(X_test)
#print(y_pred)


#compute train set accuracy
model_accuracy_train=accuracy_score(y_train,y_pred_train)
print("Model accuracy on Train data:{:.2f}".format(model_accuracy_train), '\n')

#compute test set accuracy
model_accuracy_test=accuracy_score(y_test,y_pred)
print("Model accuracy on Test data:{:.2f}".format(model_accuracy_test), '\n')

#constructing confusion matrix:
matrix_info=confusion_matrix(y_test,y_pred)
print("The Confusion Matrix: \n", matrix_info,'\n')

#Construct the classification report
class_report=classification_report(y_test,y_pred)
print("Report of classification:\n",class_report)
```

Let's add max_depth, min_samples and criterion to DecisionTreeClassifier function. This Time train score and test score and cv score are the same. This tells us the model is overfit.

```
#print(X_test.tail(5).join(y_test.tail(5)))

#now we can instantiate the model for our model
#in this case the decision tree model is used
my_model= DecisionTreeClassifier(max_depth=4, min_samples_leaf=0.05, random_state=1, criterion='gini')
#train teh model to fit
my_model.fit(X_train,y_train)

#declare the num of folds
```

```
 The total of training dataset (2988, 10)

 The total of test dataset (1993, 10)

Cross Val mean: 0.950 (std: 0.011)

Model accuracy on Train data:0.95

Model accuracy on Test data:0.95

The Confusion Matrix:
 [[1894    0]
 [  99    0]]

Report of classification:
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97      1894
         1.0       0.00      0.00      0.00        99

    accuracy                           0.95      1993
   macro avg       0.48      0.50      0.49      1993
weighted avg       0.90      0.95      0.93      1993


   The total of training dataset (2988, 10)

   The total of test dataset (1993, 10)

  Cross Val mean: 0.909 (std: 0.013)

  Model accuracy on Train data:1.00

  Model accuracy on Test data:0.90

  The Confusion Matrix:
   [[1789  105]
   [  88   11]]

  Report of classification:
                precision    recall  f1-score   support

          0.0       0.95      0.94      0.95      1894
          1.0       0.09      0.11      0.10        99

     accuracy                           0.90      1993
    macro avg       0.52      0.53      0.53      1993
 weighted avg       0.91      0.90      0.91      1993
```

Now let's have Grid search to find the best combination of hyperparameters for our model.

```
#create a cross validation split
kfolds_split=KFold(n_splits=10)

#declare a dictionary of hyperparameter and values
classifier_hypara=dict()
classifier_hypara['max_depth']=[2,3,4,6,8,10]
classifier_hypara['min_samples_split']=[2,4,6,8,9]
classifier_hypara['min_samples_leaf']=[0.05,0.1,0.5,1]
classifier_hypara['criterion']=['gini','entropy']

#perform a gridsearch and fit the grid
classifier_grid=GridSearchCV(my_model,classifier_hypara, scoring='accuracy', n_jobs=-1, cv=kfolds_split)
classifier_grid_fit=classifier_grid.fit(X,y)

#compute the array containing the 10 folds and calculate the cros validation mean score
CV_scores=-cross_val_score(classifier_grid_fit,X_train, y_train, cv=kfolds_split)
print("\nCross Val mean: {:.3f} (std: {:.3f})".format(CV_scores.mean()*-1,CV_scores.std()),end="\n\n" )

#we cab print teh hyperparameter tuning results
print('Best Hyperparameters: %s' %classifier_grid_fit.best_params_)
print('Best max_depth=', classifier_grid_fit.best_estimator_.get_params()['max_depth'])
print('Best min_samples_aplit =', classifier_grid_fit.best_estimator_.get_params()['min_samples_split'])
print('Best min_samples_leaf =', classifier_grid_fit.best_estimator_.get_params()['min_samples_leaf'])
print('Best criterion', classifier_grid_fit.best_estimator_.get_params()['criterion'])


#print best hyperparameteres
print('\n Suggested Best Hyperparameters: \n', classifier_grid_fit.best_estimator_.get_params())
print('best score: %s {:.3f}\n'.format(classifier_grid_fit.best_score_))
```

```
The total of training dataset (2988, 10)

The total of test dataset (1993, 10)

Hyperparameters of Default model
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': No
ne, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fra
ction_leaf': 0.0, 'random_state': 1, 'splitter': 'best'}
```

Now, the suggested hyperparameters will be apply to the model. It is visible that the accuracies haven't changed.

```
#You will note the results suggesting the best hyperparmeters
#combinations from the declared values in the hyperparameter dictionary
#(max_depth = 2, min_samples_split = 2, min_samples_leaf =0.05 and criterion =gini).
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.4, random_state=1, stratify=y)
#check the dimention of train and test data
print('\n The total of training dataset', X_train.shape)
print('\n The total of test dataset', X_test.shape)

#print some test data to view model predictions on instances
#print(X_test[1:5].join(y_test[1:5]))
#print(X_test.tail(5).join(y_test.tail(5)))

#now we can instantiate the model for our model
#in this case the decision tree model is used
my_model= DecisionTreeClassifier(max_depth=2, min_samples_leaf=0.05,min_samples_split = 2, random_state=1, criterion='gini')
#train teh model to fit
my_model.fit(X_train,y_train)

#declare the num of folds
num_folds=KFold(n_splits=10, random_state=1,shuffle=True)
#test by changing n_splits to 3, 5 and 10
#compute the array containing the 10 folds and calculate the cros validation mean score
CV_scores=-cross_val_score(my_model,X_train, y_train, cv=num_folds)
print("\nCross Val mean: {:.3f} (std: {:.3f})".format(CV_scores.mean()*-1,CV_scores.std()),end="\n\n" )



#now let's predict the model
y_pred_train=my_model.predict(X_train)
#print(y_pred_train)
y_pred=my_model.predict(X_test)
#print(y_pred)
```

```
 The total of training dataset (2988, 10)

 The total of test dataset (1993, 10)

Cross Val mean: 0.950 (std: 0.011)

Model accuracy on Train data:0.95

Model accuracy on Test data:0.95

The Confusion Matrix:
 [[1894    0]
 [  99    0]]

Report of classification:
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97      1894
         1.0       0.00      0.00      0.00        99

    accuracy                           0.95      1993
   macro avg       0.48      0.50      0.49      1993
weighted avg       0.90      0.95      0.93      1993
```

Now let's try ensemble learning technique. This technique aggregates a few models and with using voting classifier picks the votes with highest numbers and at the end makes a meta model [Jason B,2021]. Three models "Logostic Regression", "KNeighborsClassifier" and DecesionTreeClassifier" will be used for Ensemble learning.

```
#instantiate teh models
lr=LogisticRegression(random_state=SEED)
knc=KNeighborsClassifier()
dtc=DecisionTreeClassifier(random_state=SEED)
```

```
#INSTANTIATE A CLASSIFIER LIST
classifier_list=[('Logostic Regression:',lr),('K Nearest Neighbours:', knc),('Decisio Tree:',dtc)]

#declare a for loop to iterate through the models
for clsf_name, clsf in classifier_list:
    clsf.fit(X_train,y_train)
    # Compute the array containing the 10-folds CV MESEs
    CV_scores_clsf=-cross_val_score(clsf, X_train, y_train, cv=kfolds_split)
    print("\nCross Val mean: {:.3f} (std: {:.3f})".format(CV_scores_clsf.mean()*-1,CV_scores.std()),end="\n\n" )

    #predict and calculate the azccuracy on test data for each model
    y_predict_test_clsf=clsf.predict(X_test)
    print('\n {:s} Test : {:.3f}'.format(clsf_name,accuracy_score(y_test,y_predict_test_clsf)),'\n')

     #predict and calculate the azccuracy on train data for each model
    y_predict_train_clsf=clsf.predict(X_train)
    print('\n {:s} Train : {:.3f}'.format(clsf_name,accuracy_score(y_train,y_predict_train_clsf)),'\n')
    print("---------------")

#instantiate the voting classifier
vc=VotingClassifier(estimators=classifier_list)

#fit vc to the traing set and lables
vc.fit(X_train,y_train)

# compute the array containg the 10-folds cv mses
CV_scores_vc=-cross_val_score(vc,X_train, y_train, cv=10)
print("\nCross Val mean: {:.3f} (std: {:.3f})".format(CV_scores_vc.mean()*-1,CV_scores.std()),end="\n\n" )

#now let's predict the lable for traingn set
y_pred_train_vc=vc.predict(X_train)
print('\n voting classifier Train{:.3f}'.format(accuracy_score(y_train,y_pred_train_vc)),'\n')
```

It is visible that that Decision tree still overfits. Ensemble learning predicts with score of %94.6 and Higher consistency.

```
Cross Val mean: 0.949 (std: 0.012)

 Logostic Regression: Test : 0.951

 Logostic Regression: Train : 0.949

---------------

Cross Val mean: 0.945 (std: 0.012)

 K Nearest Neighbours: Test : 0.942

 K Nearest Neighbours: Train : 0.953

---------------

Cross Val mean: 0.908 (std: 0.012)

 Decisio Tree: Test : 0.904

 Decisio Tree: Train : 1.000

---------------
```

```
Cross Val mean: 0.947 (std: 0.012)

 voting classifier Test0.946

 voting classifier Train0.956
```

Conclusion

A few methods have been used in this research to find a model with high score and high consistency.

1. Decision Tree Classifier with (60-40 train/test data)
2. Decision Tree Classifier with (70-30 train/test data)
3. Decision Tree Classifier with (random state and max depth and min samples leaf added)
4. Decision Tree Classifier with (K-Fold)
5. Decision Tree Classifier with Grid search Technique
6. Ensemble learning technique ("Logostic Regression", "KNeighborsClassifier" and DecesionTreeClassifier")

Decision Tree with all applied method and techniques on it, overfitted the model and didn't have consistency. At the end Ensemble learning had accuracy of %94.6 and High consistency.

Reference List

1. Anna, N., 2021. *MACHINE LEARNING TECHNOLOGY OVERVIEW IN TERMS OF DIGITAL MARKETING AND PERSONALIZATION.* Available from: https://www.scs-europe.net/dlib/2021/ecms2021acceptedpapers/0125_ocms_ecms2021_0052.pdf

2. Nichd. 2022. *"How many people are affected by/at risk for stroke?"*. Available from: https://www.nichd.nih.gov/health/topics/stroke/conditioninfo/risk

3. Chg-meridian, n.d. *"Advantages and disadvantages of AI in healthcare"*. Available from: https://www.chg-meridian.co.uk/resource-centre/blog/advantages-and-disadvantages-of-artificial-intelligence-in-healthcare.html

4. Mayoclinic, n.d. *"What is a stroke?"*. Available from: https://www.mayoclinic.org/diseases-conditions/stroke/symptoms-causes/syc-20350113

5. Izzet, 2022 *"Brain stroke prediction"*. Available from: https://www.kaggle.com/datasets/zzettrkalpakbal/full-filled-brain-stroke-dataset/code

6. Pandas, 2023 *"Pandas"*. Available from: https://pandas.pydata.org/

7. Pritha B, 2022 *"Missing Data| Types, Explanation, & Imputation"*. Available from: https://www.scribbr.co.uk/stats/missing-values/

8. Nasima T, 2022 *"All you need to know about different types of missing data values and how to handle it"*. Available from: https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/

9. Nasima T, 2022 *"All you need to know about different types of missing data values and how to handle it"*. Available from: https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/

10. IBM, 2020 *"Exploratory Data Analysis"*. Available from: https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/

11. IBM, 2020 *"Exploratory Data Analysis"*. Available from: https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/

12. Neil, 2022 *"What Are Mean Median Mode Range? Definition, Examples & Practice Question For Primary School"*. Available from: https://thirdspacelearning.com/blog/what-is-mean-median-mode/

13. James C., 2022 *"Skewness: Positively and Negatively Skewed Defined with Formula"*. Available from: https://www.investopedia.com/terms/s/skewness.asp

14. Statisticshowto ., n.d *Univariate Analysis: Definition, Examples"*. Available from: https://www.statisticshowto.com/univariate/

15. Will k., 2022 *'What is Kurtosis?"*. Available from: https://www.investopedia.com/terms/k/kurtosis.asp

Jmp, n.d,*'Correlation"*. Available from: https://www.jmp.com/en_gb/statistics-knowledge-portal/what-is-correlation.html

16. Scikit_learn, 2007, *'Decision trees"*. Available from https://scikit-learn.org/stable/modules/tree.html

17. Sajal , 2022, "*How to Apply K-Fold Averaging on Deep Learning Classifier"*. Available from https://www.analyticsvidhya.com/blog/2021/09/how-to-apply-k-fold-averaging-on-deep-learning-classifier/

18. Jason B. , 2021, "*A Gentle Introduction to Ensemble Learning Algorithms"*. Available from https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/

19.

20.