KING SAUD UNIVERSITY
COLLEGE OF COMPUTER AND INFORMATION SCIENCES
INFORMATION TECHNOLOGY DEPARTMENT

جامعة الملك سعود
King Saud University
كلية علوم الحاسب والمعلومات
قسم تقنية المعلومات

IT362
Data Science Principles
Mini project
Report

STROK PREDICTION

Group Members
Noura AlSultan
Nouf AlFulaij
Seba AlAhmadi
Hadeel Quayyid

Supervised by
Dr. Amani Alajlan

TABLE OF CONTENTS

Task1:

1. INTRODUCTION:

Do you know, that 80% of strokes are preventable? But at the same time according to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths.

Our dataset is used to predict whether a patient is likely to get a stroke based on the input parameters like a wide range of age groups, gender, health-related issues, and smoking status.

1.1 About Stroke

A stroke, sometimes called a brain attack, occurs when something blocks the blood supply to part of the brain or when a blood vessel in the brain bursts.

In either case, parts of the brain become damaged or die. A stroke can cause lasting brain damage, long-term disability, or even death.

A stroke is a medical emergency, and prompt treatment is crucial. Early action can reduce brain damage and other complications.

2.1 About the Dataset (**Stroke prediction**):

1) id: unique identifier
2) gender: "Male", "Female" or "Other"
3) age: age of the patient
4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6) ever_married: "No" or "Yes"
7) work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8) Residence_type: "Rural" or "Urban"
9) avg_glucose_level: average glucose level in blood
10) bmi: body mass index
11) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12) stroke: 1 if the patient had a stroke or 0 if not

2. OBJECTIVE

Knowing your stroke risk factors, following your health care provider's recommendations and adopting a healthy lifestyle are the best steps you can take to prevent a stroke. If you've had a stroke, these measures might help prevent another stroke. The follow-up care you receive in the hospital and afterward also may play a role which we use it in our database too to predict.

- **Controlling high blood pressure (hypertension).** This is one of the most important things you can do to reduce your stroke risk.
- **Lowering the amount of cholesterol and saturated fat in your diet**. Eating less cholesterol and fat.
- **Quitting tobacco use.** Smoking raises the risk of stroke for smokers and nonsmokers exposed to secondhand smoke
- **Maintaining a healthy weight.** Being overweight contributes to other stroke risk factors so try to keep your BMI in the normal range.

So, through this mini project, we will explore which factors are the leading drivers for stroke and build a descriptive model using Machine Learning to bring awareness to individuals and help healthcare providers and insurers estimate risk and cost.

TASK 2:

3. DATA PREPARATION, CLEANING & EDA

```
In [7]: HealthD.shape #to know the coulmns and rows number
Out[7]: (5110, 12)
```

Our dataset consists of 5110 rows and 10 columns

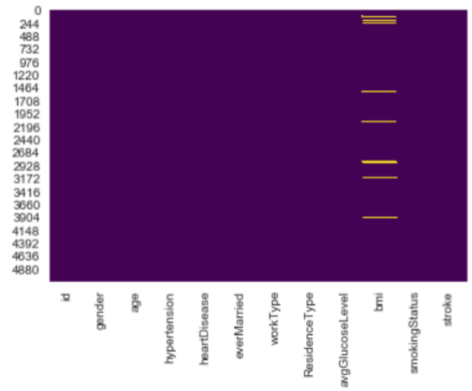
```
In [8]: HealthD.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                5110 non-null   int64
 1   gender            5110 non-null   object
 2   age               5110 non-null   float64
 3   hypertension      5110 non-null   int64
 4   heartDisease      5110 non-null   int64
 5   everMarried       5110 non-null   object
 6   workType          5110 non-null   object
 7   ResidenceType     5110 non-null   object
 8   avgGlucoseLevel   5110 non-null   float64
 9   bmi               4909 non-null   float64
 10  smokingStatus     5110 non-null   object
 11  stroke            5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

We got some information about the type of each column in the dataset (id, hypertension, heartDisease, stroke) are of type int. (gender, everMarried, workType, ResidemceType, smokingStatus) are of type object. (age, avgGlucoseLevel, bmi) are of type float.

Null values

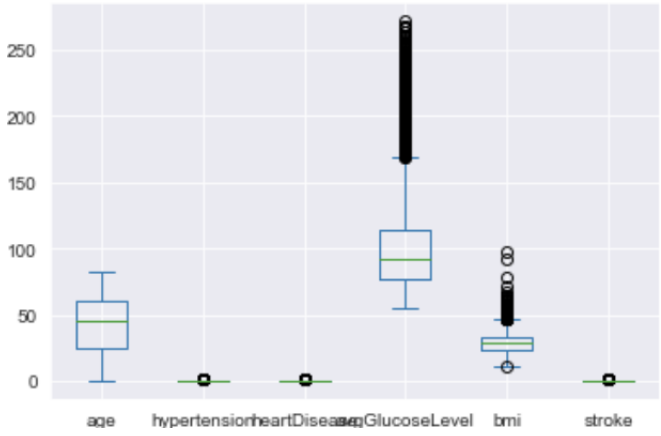
```
In [610]: #checking for missing values , BMI has null values
          HealthD.isnull().sum()

Out[610]: id                   0
          gender               0
          age                  0
          hypertension         0
          heartDisease         0
          everMarried          0
          workType             0
          ResidenceType        0
          avgGlucoseLevel      0
          bmi                201
          smokingStatus        0
          stroke               0
          dtype: int64
```

```
: #visualizing null values :
  sns.heatmap(HealthD.isnull(),cbar=False,cmap='viridis')

: <matplotlib.axes._subplots.AxesSubplot at 0x1a25a7a2e8>
```



The data was mostly clean and had no duplicated rows but there were missing values in the BMI. the code shows that BMI Colum contains missing values

Here is a heatmap of the null values which shows the null values in our dataset, so we handle the null value with the mean because it is a numeric column.



We checked if we have outliers by plotting a box plot and then it shows that we have a lot of outliers so removing them will be difficult and they can cause bias and/or influence estimates.

```
3]: # contenious of MEASURE OF SPREAD -range -> max-min , -IQR:dis between(3rd

rangeAGE= max(HealthD["age"]-min(HealthD["age"]))
rangeGL= max(HealthD["avgGlucoseLevel"]-min(HealthD["avgGlucoseLevel"]))
rangebmi= max(HealthD["bmi"]-min(HealthD["bmi"]))

print("the range of age =",rangeAGE)
print("the range of avgGlucoseLevel =",rangeGL)
print("the range of bmi =",rangebmi)

print("*******************************")
print("*******************************")

quaAGE= HealthD["age"].quantile(0.75)-HealthD["age"].quantile(0.25)
quaGAL= quaAGE= HealthD["avgGlucoseLevel"].quantile(0.75)-HealthD["avgGluc
quabmi= HealthD["bmi"].quantile(0.75)-HealthD["bmi"].quantile(0.25)

print("the IQR of age =",rangeAGE)
print("the IQR of avgGlucoseLevel =",quaGAL)
print("the IQR of bmi =",quabmi)

print("*******************************")
print("*******************************")

print("the variance of age =",HealthD["age"].var())
print("the variance of avgGlucoseLevel =",HealthD["avgGlucoseLevel"].var()
print("the variance of bmi =",HealthD["bmi"].var())
```

```
the range of age = 81.92
the range of avgGlucoseLevel = 216.62
the range of bmi = 87.3
*******************************
*******************************
the IQR of age = 81.92
the IQR of avgGlucoseLevel = 36.845
the IQR of bmi = 9.600000000000001
*******************************
*******************************
the variance of age = 511.33179182433673
the variance of avgGlucoseLevel = 2050.600819911381
the variance of bmi = 61.68636419426877
```

We calculate the range which is the difference between the largest and smallest values, IQR which is he middle half of the data, variance which is the average squared difference of the values from the mean, of the continues float columns.

```
: for col in HealthD:
      print(col)
      print(HealthD[col].unique())
      print('\n')
      #to know if there is any inconsist va
```

```
id
[ 9046 51676 31112 ... 19723 37544 44679]


gender
['Male' 'Female' 'Other']
```

We printed the unique and values of each column to know if there is any inconsistent value in any column & to know what the values in each column are, we didn't find any inconsistent value.

Get Dataset description

categorical columns

```
HealthD[HealthD.stroke == 1].describe(include=['O'])
```

	gender	everMarried	workType	ResidenceType	smokingStatus
count	249	249	249	249	249
unique	2	2	4	2	4
top	Female	Yes	Private	Urban	never smoked
freq	141	220	149	135	90

checking the description of the people with stoke (our target value) for categorical columns ONLY
we can see the count, unique, top, and frequent values for each categorical column.
-gender has 2 unique values and top values is female.
-everMarried has 2 unique values and top values is yes.
-workType has 4 unique values and top values is private.
ResidenceType has 2 unique values and top values is Urban.
-smokingStatus has 4 unique values and top values is never smoked.

Get Dataset description

numeric columns

```
HealthD[HealthD.stroke == 1].describe() #for numircal coulmns ONLY
```

Out[19]:

	id	age	hypertension	heartDisease	avgGlucoseLevel	bmi	stroke
count	249.000000	249.000000	249.000000	249.000000	249.000000	209.000000	249.0
mean	37115.068273	67.728193	0.265060	0.188755	132.544739	30.471292	1.0
std	21993.344872	12.727419	0.442254	0.392102	61.921056	6.329452	0.0
min	210.000000	1.320000	0.000000	0.000000	56.110000	16.900000	1.0
25%	17013.000000	59.000000	0.000000	0.000000	79.790000	26.400000	1.0
50%	36706.000000	71.000000	0.000000	0.000000	105.220000	29.700000	1.0
75%	56669.000000	78.000000	1.000000	0.000000	196.710000	33.700000	1.0
max	72918.000000	82.000000	1.000000	1.000000	271.740000	56.600000	1.0

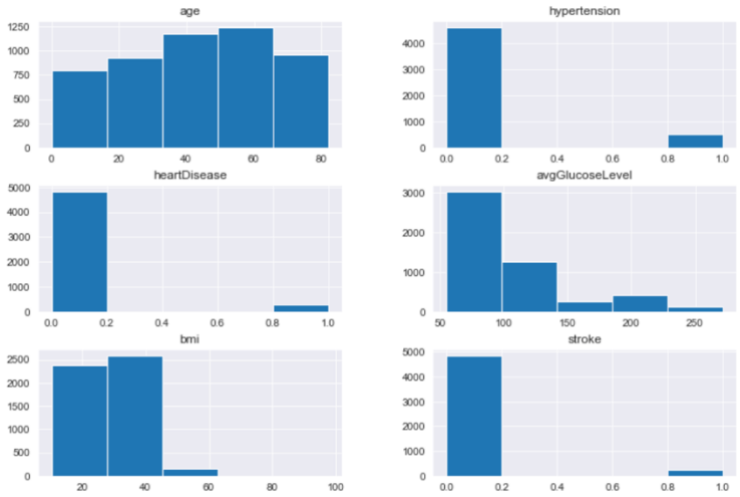
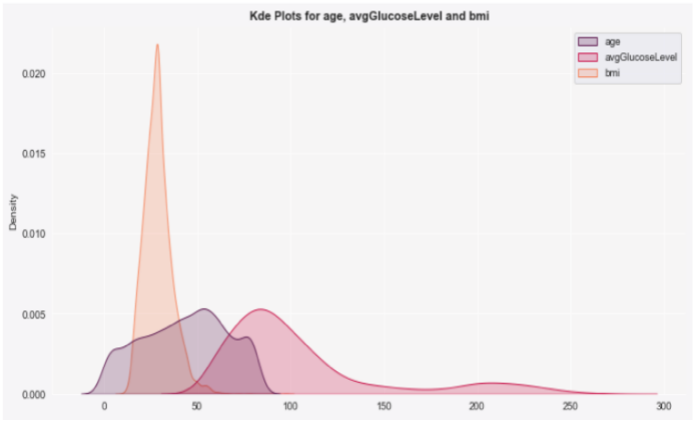
checking the description of the people with stoke (our target value) for numeric columns ONLY
we can see the count, mean std, min, max, 25%, 50%, 75% for each numerical column.
-average age of people who have stroke is 67 years old
-min age of people who have stroke is 1.32 years old
-max age of people who have stroke is 82 years old
-average avgGlucoseLevel of people who have stroke is 132.54
-min avgGlucoseLevel of people who have stroke is 56.1
-max avgGlucoseLevel of people who have stroke is 271.74
-average bmi of people who have stroke is 30.47
-min bmi of people who have stroke is 16.9
-max bmi of people who have stroke is 56.60

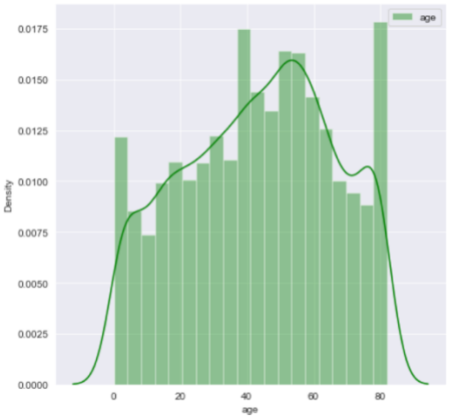
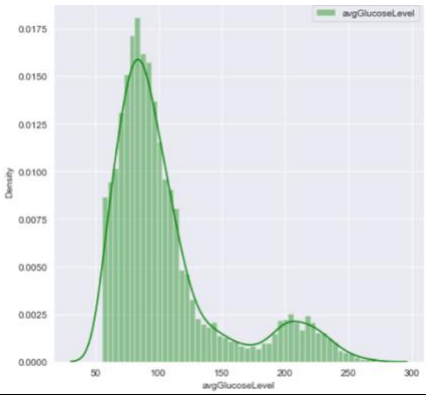
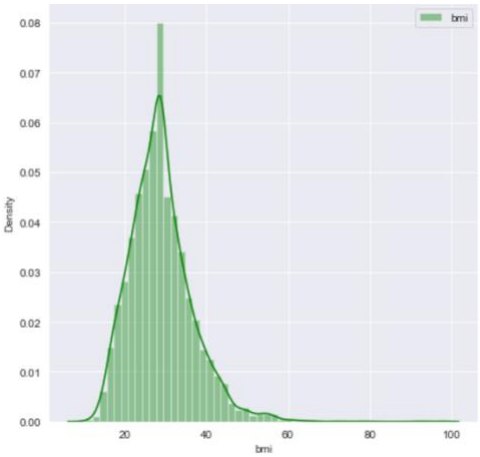
```
cols_to_drop=['id']
HealthD=HealthD.drop(cols_to_drop,axis=1)
HealthD.head()
```

	gender	age	hypertension	heartDisease	everMarried	workType	Resider
0	Male	67.0	0	1	Yes	Private	
1	Female	61.0	0	0	Yes	Self-employed	
2	Male	80.0	0	1	Yes	Private	
3	Female	49.0	0	0	Yes	Private	
4	Female	79.0	1	0	Yes	Self-employed	

after we observed all columns, we decide to drop the id column because it will not help us, and it is not relevant.

Findings on numeric columns:

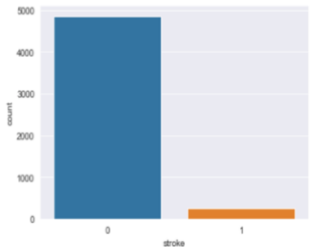
Univariate	
	
Most people age in our dataset is from the range 40-60, the average glucose level is from 50-100, the people with stroke, heart disease and hypertension are much less than the people that don't have these diseases.	From this figure we can see that the avgGlucoseLevel the distribution is between 50-150 also, the bmi is between 0 and 50 for most of the people and the age is distributed between 0 and 100 almost evenly.

	age: in the age column we have data from infants to old people and from the statical calculation it is clear that the adults are the median of the group.
	avgGlucoseLevel: average glucose levels shows that most of the people have controlled glucose level.
	bmi: the average bmi of the people is 28 which is quite high.

-Distribution of Target variable in the data set

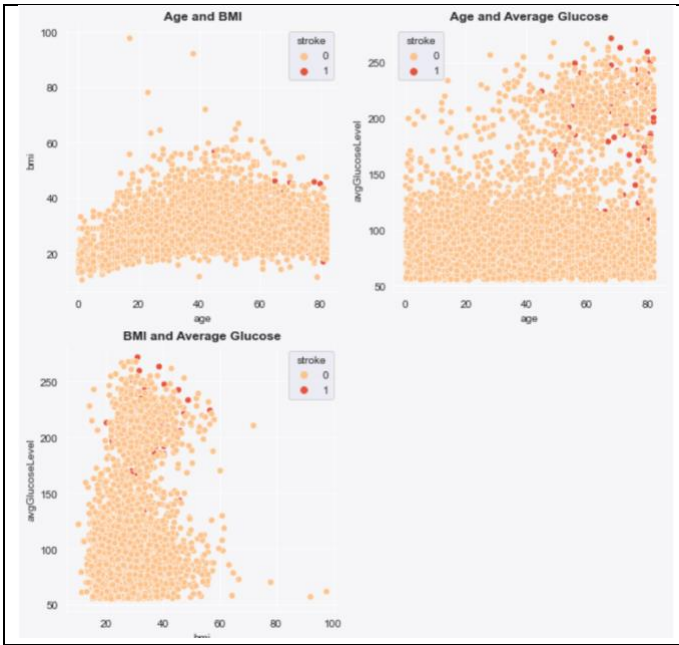
```
In [627]: #Non-graphical-univariate-numircal
          #Percentage for  having stroke
          d = HealthD["stroke"].value_counts()
          print(d)

          Percent_d = d[1]/ d.sum() * 100
          print("Percent who have stroke:",Percent_d )
          print("*********************************")
          print("*********************************")

          #Percentage for  not having covid
          dn = HealthD["stroke"].value_counts()
          print(dn)

          Percent_dn = dn[0]/ dn.sum() * 100
          print("Percent who haven't stroke:",Percent_dn )

0    4861
1     249
Name: stroke, dtype: int64
Percent who have stroke: 4.87279843444227
*********************************
*********************************
0    4861
1     249
Name: stroke, dtype: int64
Percent who haven't stroke: 95.12720156555773
```

```
In [640]: #3.plot of the class lable :
          sns.countplot(x='stroke', data=HealthD,order=HealthD['stroke'].value_counts().index)

Out[640]: <matplotlib.axes._subplots.AxesSubplot at 0x1a21e6df28>
```

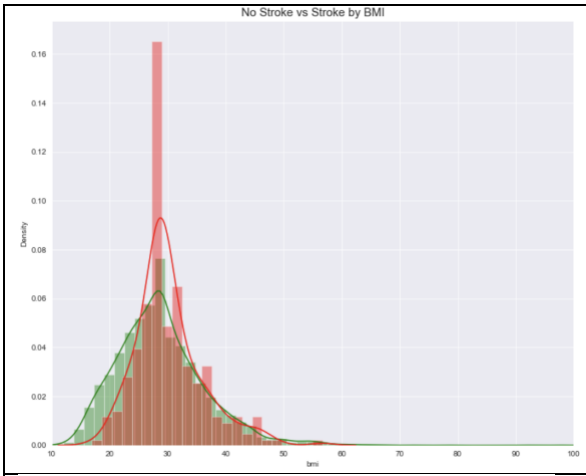
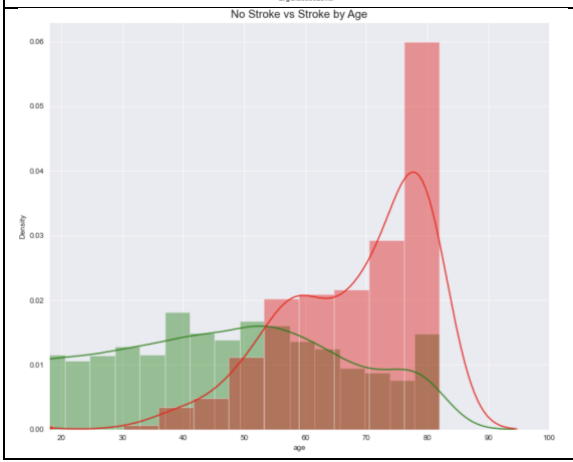


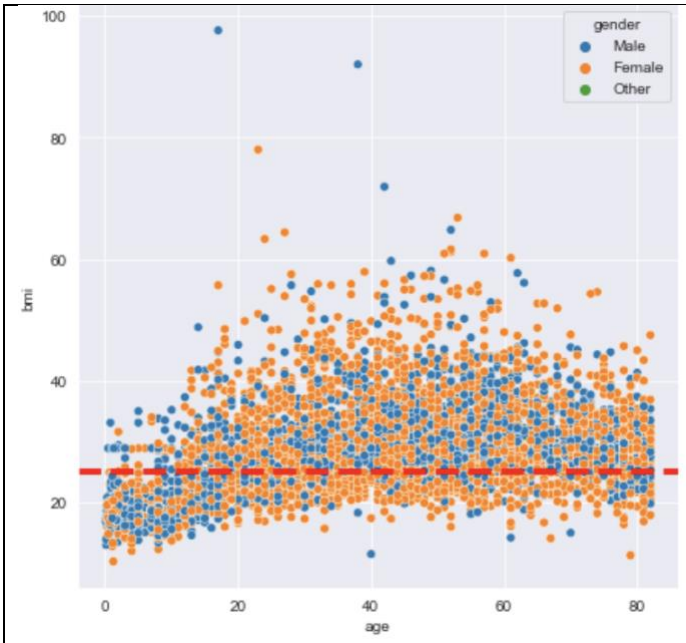
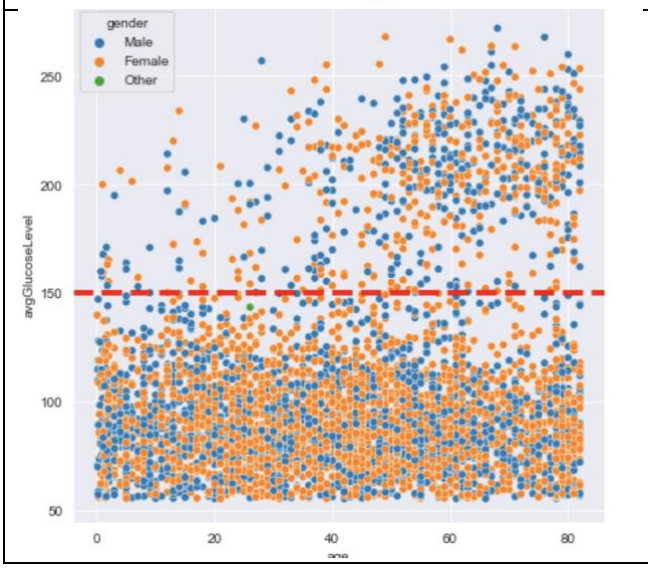
From the graph you can see that number people with stork much less than who haven't got stork. And when we calculated it, we found that the percentage of people with stork is 4.87% and who haven't is 95%.
From distribution it is clear that every 5 people out of 100 people are having strokes from our sampling data. this is a highly unbalanced data distribution, and null accuracy score of this distribution itself is 95%, which employs any dump model should randomly predictions of stroke could reach accuracy of 95%. So, while modeling and training data, either over sampling or under sampling must be done to obtain best results.

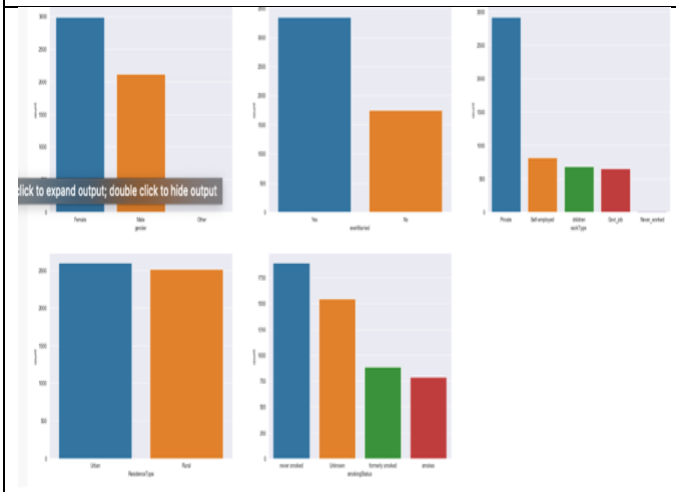
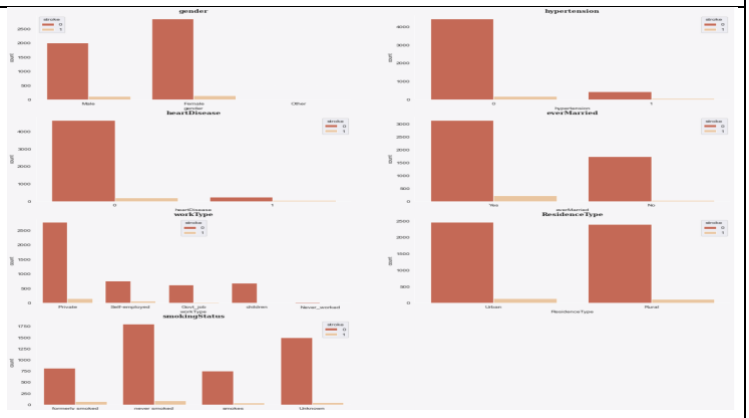
-Bivariate analysis of Numerical Variables:



After carefully analyzing these plots, we can get following insights, The data appears to be highly imbalanced (only few points where stroke = 1) also, Age and Avg glucose levels can be split into 2 clusters (one less than 150 and other more than that). Only few cases of people with glucose levels less than 150 experienced stroke and BMI and Glucose levels confirm that people with less than 150 glucose levels are less prone to strokes than people with glucose levels more than 150 level. BMI >40 have low avg glucose.

	From the graph, it shows that the density of overweight people who suffered a stroke is more.
	From graph, it shows that the density of people having glucose level less than 100 suffered stroke more.
	After plotting the age with the stroke Colum, based on the above plots, it seems clear that Age is a big factor in stroke patients Age is an important feature old age people are mostly having strokes (the density of people having age above 50 suffered stroke more) compared to younger ones - the older you get the more at risk you are.

	From the plot, we can see that there are lot of people having BMI above 25 are overweight and obese.
	From the plot, we can see that people having glucose level above 150 are relatively less as compare one below. So, we can say that people above 150 might be suffering from diabetes.

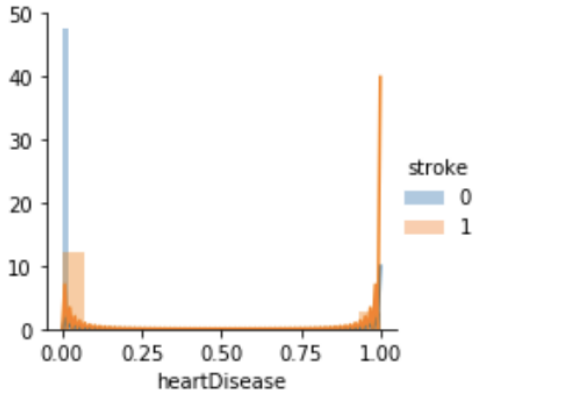
Categorical columns	
Univariate	**Bivariate**
	
For categorical columns we can see that Females are more than male in our data. The Other category in gender is not visible as it contains only one value, The number of people without hypertension are way less than people who has it, The number of people with heart disease is extremely low, The number of people who are married are way more than unmarried people(makes sense as the distribution is between 0 and 60), People seem to prefer working in private companies while the number of self-employed/ govt_job and children seems to be equal in number (children can be ignored). Unemployed people are extremely less, The Unknown category represents that we do not know if a person smoked or not. Non-smokers are way more than people who smoked/used to smoke which is a good thing, Number of people with strokes are less than 1000 in number.	Overview of the categorical features with the target value(stroke) shows that number of male and female who has stroke are equal in number, The number of people who do not have hypertension also shows signs of no stroke. And people with hypertension also do not show signs of more people with stroke (may be due to the fact that our data has so many negative(0) variables), The people with heart disease show signs of stroke too(as expected), The people who got married show signs of stroke way more than people who are unmarried, Private employees seems to experience stroke more than other work_types(may be due to work pressure). Self-employed people do show signs of stroke (may be due to reasons like heart disease, tension etc.), People who formerly smoked and who smoke (combined) are showing signs of stroke way more than people who never smoked.

Stroke with heart disease:

```
no = HealthD['heartDisease'].value_counts().values[0]
yes =  HealthD['heartDisease'].value_counts().values[1]
#Having or not having stroke according to the heartDisease attrbute
heartDisease_stroke = HealthD[(HealthD['heartDisease'] == 1) & (HealthD['stroke'] == 1)]
heartDisease_strokeN = int(len(heartDisease_stroke))
print ("he/she have heartDisease and stroke: ",heartDisease_strokeN)

heartDisease_NOstroke = HealthD[(HealthD['heartDisease'] == 1) & (HealthD['stroke'] == 0)]
heartDisease_NOstroks = int(len(heartDisease_NOstroke))
print ("he/she have heartDisease and NOT having stroke: ",heartDisease_NOstroks)

NOheartDisease_stroke = HealthD[(HealthD['heartDisease'] == 0) & (HealthD['stroke'] == 1)]
NOheartDisease_stroks = int(len(NOheartDisease_stroke))
print ("he/she haven't heartDisease and having stroke: ",NOheartDisease_stroks)

NOheartDisease_NOstroke = HealthD[(HealthD['heartDisease'] == 0) & (HealthD['stroke'] == 0)]
NOheartDisease_NOstrokeN = int(len(NOheartDisease_NOstroke))
print ("he/she haven't heartDisease and haven't stroke: ",NOheartDisease_NOstrokeN)

stroke_no = int(round (heartDisease_NOstroks/ no * 100, 0))
stroke_yes = int(round( heartDisease_strokeN/ yes *100, 0))
print("prcentage of pepole dont have heartDisease and dont have a stroke: ",stroke_no,"%")
print("prcentage of pepole have heartDisease and have a stroke: ",stroke_yes,"%")
```

```
he/she have heartDisease and stroke:  47
he/she have heartDisease and NOT having stroke:  229
he/she haven't heartDisease and having stroke:  202
he/she haven't heartDisease and haven't stroke:  4632
prcentage of pepole dont have heartDisease and dont have a stroke:  5 %
prcentage of pepole have heartDisease and have a stroke:  17 %
```

```
#plotting the histogram of age,stro
sns.FacetGrid(HealthD,hue='stroke')
```



As shown here and in the plot above risk of stroke for people with heart disease is higher than people who have not heart disease nearly 13% of people are having strokes when they have heart disease previously.

Task 3:

4. Feature engineering

We have done encoding on some categorical columns that contains types to make it easier to deal with and it will be much easier to perform the classification on them first, we encode the gender column to (female=0,male=1,other=2), Then the work type to (Private=1, Self-employed=2, children=3, Govt_job=4, Never_worked=5), And the Residence Type to. (Urban=1, Rural=2) and the smoking status to (never smoked=1, Unknown=2, formerly smoked=3, smokes=4) Finally the ever married column to (No=0, Yes=1)

We then print head to check conversion:

```
HealthD.head()
```

Out[53]:

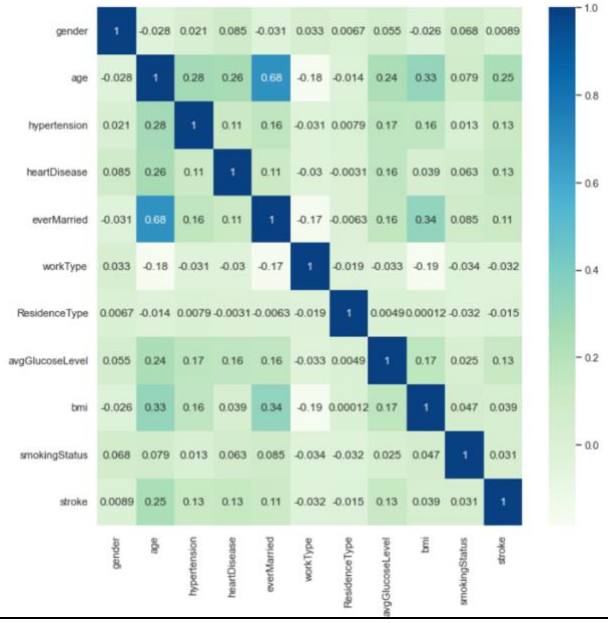
	gender	age	hypertension	heartDisease	everMarried	workType	ResidenceType	avgGlucoseLevel	bmi	smokingStatu
0	1	67.0	0	1	1	1	1	228.69	36.600000	
1	0	61.0	0	0	1	2	2	202.21	28.893237	
2	1	80.0	0	1	1	1	2	105.92	32.500000	
3	0	49.0	0	0	1	1	1	171.23	34.400000	
4	0	79.0	1	0	1	2	2	174.12	24.000000	

Task 4:

5. Explore the relationship and correlations between variables

```
In [54]: correlations=HealthD.corr()
         print(correlations["stroke"])
```

```
gender              0.008929
age                 0.245257
hypertension        0.127904
heartDisease        0.134914
everMarried         0.108340
workType           -0.032098
ResidenceType      -0.015458
avgGlucoseLevel     0.131945
bmi                 0.038947
smokingStatus       0.030682
stroke              1.000000
Name: stroke, dtype: float64
```



We calculated the correlation between all columns and our target value, and it shows that **age** have the strongest positive correlation between all the columns with stroke.
also, hypertension, heartDisease, evermarried, smokingStatusand, gender, bmi and avgGlucoseLevel while workType and ResidentType have negative correlation with stroke.

Task 5:

6. different classification models

First, we split the data into train and test in y we select the stroke column as the outcome (response) and in x we select all the columns as predictors except the stroke column which is the output as we say.

```
#splitting the dataset into train and test diffult 75%,25%
X = HealthD.drop('stroke',axis=1)
y = HealthD['stroke']
```

And then we use the train_test_split() function to split the sample set into an 80% training set, which we will use to train the model, and a 20% test set, to evaluate the model to avoid overfitting.

```
#split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=101)
print ("X_train: ", X_train.shape)
print ("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print ("y_test: ", y_test.shape)
```

```
X_train:  (4088, 10)
y_train:  (4088,)
X_test:  (1022, 10)
y_test:  (1022,)
```

We used 3 different methods to build the model:

- **First Method Logistic Regression:**

As we study Logistic Regression is a process of modeling the probability of a discrete outcome given an input variable.

we initiate the Logistic model by the LogisticRegression function and then we train the model by y_train , X_train using fit function.

Because the data set is highly an imbalance, in which the accuracy is not enough to estimate the performance of classification models because is just the accuracy of one class, so we need to consider the f1-score which is the single score that balances both the concerns of precision and recall in one number, we also used another way to solve it by SMOTE method to resampling as we will talking about it later.

```
acc_log=accuracy_score(y_test,pred_log) # acurecy for test
print("Prediction Accuracy TEST for logistic regression: ",acc_log)
accRL = round(accuracy_score(y_test,pred_log) * 100, 2)
print("Prediction Accuracy TEST after round: ",accRL)
```

```
Prediction Accuracy TEST for logistic regression:  0.9393346379647749
Prediction Accuracy TEST after round:  93.93
```

```
from sklearn import metrics #confusion matrix for Logistic regression TESTING
print(metrics.classification_report(y_test,pred_log))
```

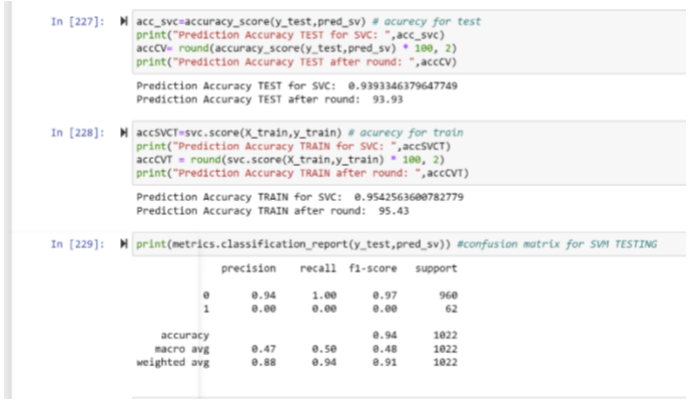
```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       960
           1       0.00      0.00      0.00        62

    accuracy                           0.94      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.88      0.94      0.91      1022
```

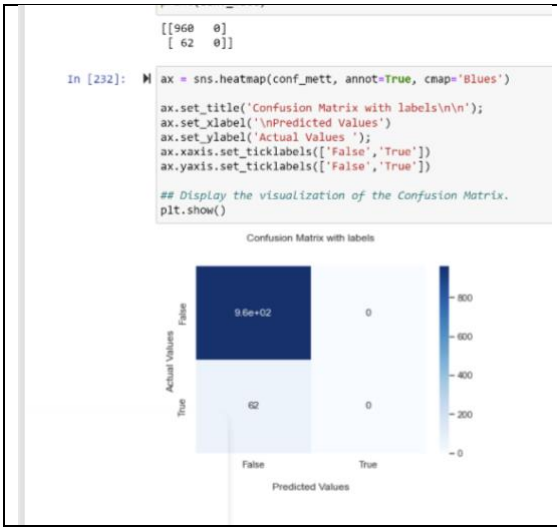
As we said earlier, even the accuracy is 0.94 because it correctly predicts the majority which is the class (0) that reason why accuracy is not the best measure for imbalanced data because it is just for one class when seeing the f1-score for class 1 (0.0) which means it is not predict class 1 correctly because of the imbalanced data set.

	These refer 960 of the negative tuples is predicted correctly. And 62 of the positive tuples were incorrectly labeled as negative!.

- **Second Method SVM:**

As we study SVM is to find optimal hyperplane for linearly separable patterns.
we initiate the SVM model by the SVC function then we do the same as we did in Logistic Regression: fit, make the machine predict y by using X_test, and then calculate the accuracy score by comparing predict y with actual y
but as we said accuracy is not the best so when seeing the f1-score for class 1 (0.0) we found is same as the Logistic Regression model which means is not predict class 1 correctly
because of the imbalanced data set.

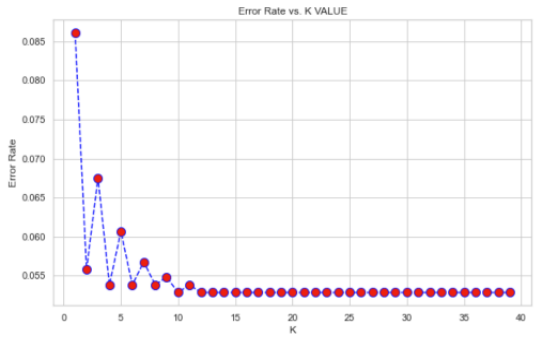


	These refer 960 of the negative tuples is predicted correctly. And 62 of the positive tuples were incorrectly labeled as negative!.

- **Third Method KNN:**

  KNN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest boundary line.
  But the biggest issue with K-NN is choosing the optimal number of neighbors to be considered while classifying the new data entry.

we used an error rate to choose an optimal number of neighbors: which have the lowest error rate.

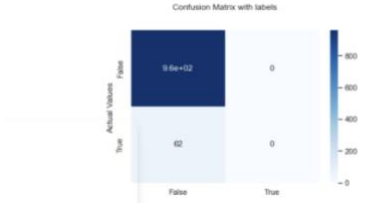


We initiate the KNN model by the KNeighborsClassifier(n_neighbors=15) function after choosing the optimal number of neighbors as we see above, and then we do the same as we did in Logistic Regression and SVM.

```
In [237]: acc_knn=accuracy_score(y_test,pred_knn) # acurecy for test
          print("Prediction Accuracy TEST for KNN: ",acc_knn)
          accKNR = round(accuracy_score(y_test,pred_knn) * 100, 2)
          print("Prediction Accuracy TEST after round: ",accKNR)

          Prediction Accuracy TEST for KNN:  0.9393346379647749
          Prediction Accuracy TEST after round:  93.93

In [238]: accKNRT=knn.score(X_train,y_train) # acurecy for train
          print("Prediction Accuracy TRAIN for KNN: ",accKNRT)
          accKNRT = round(knn.score(X_train,y_train) * 100, 2)
          print("Prediction Accuracy TRAIN after round: ",accKNRT)

          Prediction Accuracy TRAIN for KNN:  0.9542563600782779
          Prediction Accuracy TRAIN after round:  95.43

In [239]: print(metrics.classification_report(y_test,pred_knn)) #confusion matrix for knn TESTING
                        precision    recall  f1-score   support

                     0       0.94      1.00      0.97       960
                     1       0.00      0.00      0.00        62

              accuracy                           0.94      1022
             macro avg       0.47      0.50      0.48      1022
          weighted avg       0.88      0.94      0.91      1022
```

After we find the f1-score to decide the performance of the model since the dataset imbalance, we notice it is the same as the SVM model and Logistic Regression.

```
In [240]: conf_mettt=metrics.confusion_matrix(y_test,pred_knn)
          print(conf_mettt)

          [[960   0]
           [ 62   0]]

In [241]: ax = sns.heatmap(conf_mettt, annot=True, cmap='Blues')

          ax.set_title('Confusion Matrix with labels\n\n');
          ax.set_xlabel('\nPredicted Values')
          ax.set_ylabel('Actual Values ');
          ax.xaxis.set_ticklabels(['False','True'])
          ax.yaxis.set_ticklabels(['False','True'])

          ## Display the visualization of the Confusion Matrix.
          plt.show()
```



These refer 960 of the negative tuples is predicted correctly.
And 62 of the positive tuples were incorrectly labeled as negative!.

We have found that:

After we tried the models above and find that all their accuracies are high because our dataset is imbalanced the accuracy is not the best then we decided to see the f1 score because it is best to measure for the imbalanced datasets, but we found all method by seeing f1 score when saw the class1 (0.0) does not predict class 1 correctly!

- because of the imbalanced data set.
- to solve this, we can use another way to solve it, using SMOTE method for oversampling.

- **Another solution for imbalance dataset rather than f1-score:**

```
In [243]: print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
          print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))

          print('before OverSampling, the shape of X_train: {}'.format(X_train.shape))
          print('before OverSampling, the shape of y_train: {} \n'.format(y_train.shape))

          sm = SMOTE(random_state=2)
          X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

          print('After OverSampling, the shape of X_train: {}'.format(X_train_res.shape))
          print('After OverSampling, the shape of y_train: {} \n'.format(y_train_res.shape))

          print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1)))
          print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0)))

          Before OverSampling, counts of label '1': 187
          Before OverSampling, counts of label '0': 3901

          before OverSampling, the shape of X_train: (4088, 10)
          before OverSampling, the shape of y_train: (4088,)

          After OverSampling, the shape of X_train: (7802, 10)
          After OverSampling, the shape of y_train: (7802,)

          After OverSampling, counts of label '1': 3901
          After OverSampling, counts of label '0': 3901
```

We can solve the imbalance of data by oversampling the example of the minority class. This can be achieved by simply duplicating examples from the minority class in the training dataset that help to balance data but without adding additional information.
But we used SMOTE which is synthesize new examples from the minority class which is 1 class.
As we see above after SMOTE the distribution of the data becomes balanced.

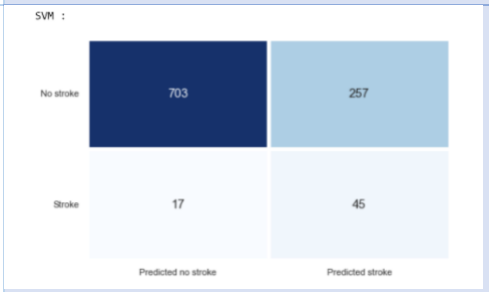
After that, we can use accuracy to compare performance of different models:

Task7:

Comparing method after resampling by SMOTE:

	Mean Accuracy	Precision	Recall	F1-score
1.SVM(1)	76.429%	0.149	0.725	0.247
2.LR	79.390%	0.156	0.580	0.246
3.KNN	88..746%	0.181	0.612	0.280

Confusion matrix after resampling by SMOTE:

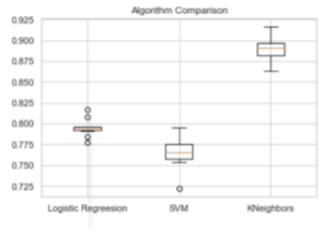
Classifier	Confusion matrix
1.LR	
2.SVM	
3.KNN	


```
df.sort_values(by= ['Accuracy', 'K-Fold Mean Accuracy'], inplace= True, ascending= False)
df
```

:7]:

	Model	Accuracy	K-Fold Mean Accuracy	Std. Deviation	ROC AUC	Precision	Recall	F1
2	KNeighbors	80.919765	88.746397	0.860525	0.717389	0.181818	0.612903	0.280443
0	Logistic Regreesion	78.473581	79.390656	2.399901	0.689281	0.156522	0.580645	0.246575
1	SVM	73.189824	76.429282	1.752258	0.729049	0.149007	0.725806	0.247253

```
#models' K-Fold Mean Accuracy
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison')
plt.show()
```

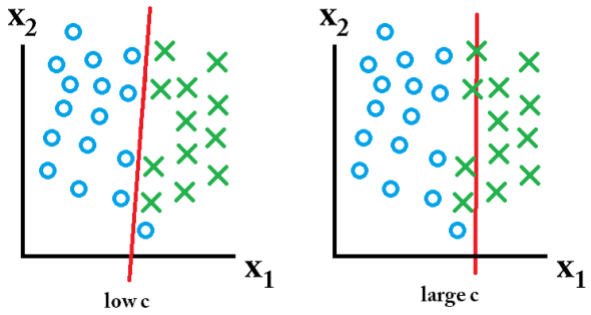


we found the best model: **KNN** and then Logistic Regression lastly SVM.

Task 6:

7. Experiment with different parameters in SVM:

As we studied C It is a hypermeter in SVM to control error It controls the tradeoff between smooth decision boundary and classifying the training points correctly.



This example on the internet to see if we C is large means larger error because does not classify correctly

rather than when we have a small C which is a small error as we see above the picture on the right classifies better.

And Gamma defined how far the influence of a single training example reaches, with low values meaning ‘far’

And high values meaning ‘close’.

1. 'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['linear']
Best parameters:
{'C': 10, 'gamma': 1, 'kernel': 'linear'}

Accuracy after round: 79.55

2. kernel: ['sigmoid'] C: [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001]

Best parameters:
{'C': 0.1, 'gamma': 1, 'kernel': 'sigmoid'}

Accuracy after round: 93.93

3. C: [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], kernel: ['rbf']
Best parameters:
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Accuracy after round: 88.65

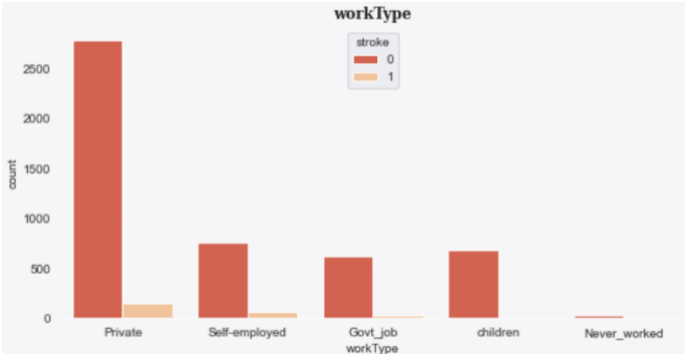
Best accuracy when kernel: ['sigmoid'] C: [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001]

Task8:

8. QUESTIONS ANSWERS:

1. Dose work-type affect if a person will have stroke or not?

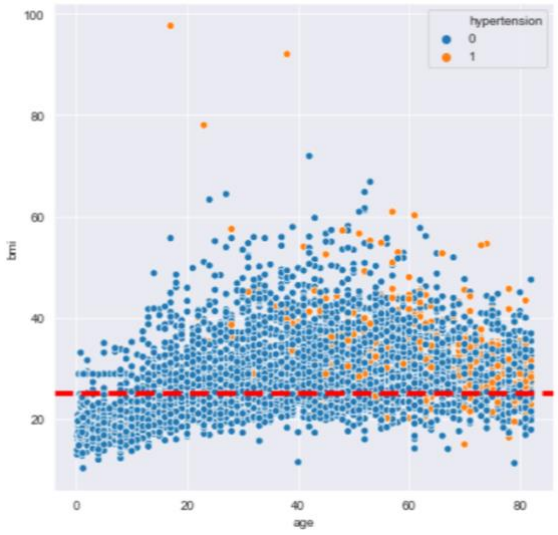
```
the prcentage of private workers to have a stroke:  5 %
the prcentage of self workers to have a stroke:  8 %
the prcentage of children to have a stroke:  0 %
the prcentage of gov workers to have a stroke:  5 %
the prcentage of pepole never worked to have a stroke:  0 %
```



Yes, from the graph below, we can see that 5% of the private workers have stroke and 8% of the self-employed have stroke However people from the government are more likely to not have a stroke compared to both first categories but children and people who never worked are not very likely to get a stroke their percentage of having stroke. Is 0% the reason could be that they don't face the same degree of pressure felt by workers

that shows us that work-type plays a good role in predicting stroke

2. Dose hypertension affected by BMI and age.?



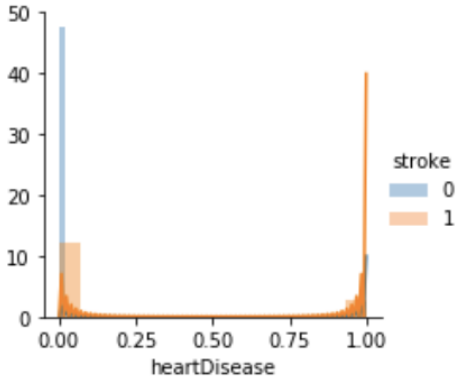
Yes, as we can see in the graph age and BMI have a big effect. if the person is hypertension patient or not, if the BMI is above 20 and the age above 60 the person is more likely to be hypertension patient

3. Does heart disease affect if a person will have stroke or not?

```
no = HealthD['heartDisease'].value_counts().values[0]
yes =  HealthD['heartDisease'].value_counts().values[1]
#Having or not having stroke according to the heartDisease attrbute
heartDisease_stroke = HealthD[(HealthD['heartDisease'] == 1) & (HealthD['stroke'] == 1)]
heartDisease_strokeN = int(len(heartDisease_stroke))
print ("he/she have heartDisease and stroke: ",heartDisease_strokeN)

heartDisease_NOstroke = HealthD[(HealthD['heartDisease'] == 1) & (HealthD['stroke'] == 0)]
heartDisease_NOstroks = int(len(heartDisease_NOstroke))
print ("he/she have heartDisease and NOT having stroke: ",heartDisease_NOstroks)

NOheartDisease_stroke = HealthD[(HealthD['heartDisease'] == 0) & (HealthD['stroke'] == 1)]
NOheartDisease_stroks = int(len(NOheartDisease_stroke))
print ("he/she haven't heartDisease and having stroke: ",NOheartDisease_stroks)

NOheartDisease_NOstroke = HealthD[(HealthD['heartDisease'] == 0) & (HealthD['stroke'] == 0)]
NOheartDisease_NOstrokeN = int(len(NOheartDisease_NOstroke))
print ("he/she haven't heartDisease and haven't stroke: ",NOheartDisease_NOstrokeN)

stroke_no = int(round (heartDisease_NOstroks/ no * 100, 0))
stroke_yes = int(round( heartDisease_strokeN/ yes *100, 0))
print("prcentage of pepole dont have heartDisease and dont have a stroke: ",stroke_no,"%")
print("prcentage of pepole have heartDisease and have a stroke: ",stroke_yes,"%")
```

```
he/she have heartDisease and stroke:  47
he/she have heartDisease and NOT having stroke:  229
he/she haven't heartDisease and having stroke:  202
he/she haven't heartDisease and haven't stroke:  4632
prcentage of pepole dont have heartDisease and dont have a stroke:  5 %
prcentage of pepole have heartDisease and have a stroke:  17 %
```

```
#plotting the histogram of age,stro
sns.FacetGrid(HealthD,hue='stroke')
```

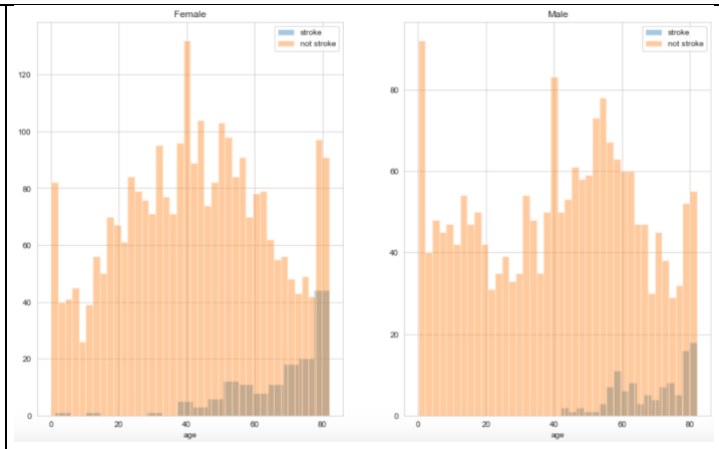


As shown in the code and in the plot above risk of stroke for people with heart disease is 17% while the chance of having stroke for people who do not have heart disease is 5% **that means that heart disease can be good indicator in predicting if a person is going to have a stroke or not if he\she have history in heart disease there is a chance that he\she will have stroke too.**

4. Does Gender affect if people have a stroke or not?

```
#Having or not having stroke according to the female attrbute -female is c
male_stroke = HealthD[(HealthD['gender']==1)&(HealthD['stroke']==1)]
males_stroke = int(len(male_stroke))
print ("male have a stroke: ",males_stroke)
male_per = int(round(males_stroke/(male) * 100, 0))

print("prcentage of male have a stroke: ",male_per,"%")


male_no_stroke = HealthD[(HealthD['gender']==1)&(HealthD['stroke']== 0)]
males_no_stroke =int(len(male_no_stroke))
print ("male that dont have a stroke: ",males_no_stroke)

female_stroke = HealthD[(HealthD['gender']==0)&(HealthD['stroke']==1)]
females_stroke = int(len(female_stroke))
print ("female have a stroke: ",females_stroke)
female_per = int(round(females_stroke/(female)* 100, 0))

print("prcentage of female have a stroke: ",female_per,"%")


female_no_stroke = HealthD[(HealthD['gender']==0)&(HealthD['stroke']== 0)]
females_no_stroke =int(len(female_no_stroke))
print ("female that dont have a stroke: ",females_no_stroke)
```

```
male have a stroke:  108
prcentage of male have a stroke:  5 %
male that dont have a stroke:  2007
female have a stroke:  141
prcentage of female have a stroke:  5 %
female that dont have a stroke:  2853
```



Total number of males 2007, 108(5%) of them have stroke and number of females 2853, 141(5%) of them have stork. Here from the graph, we can see that most females who have risk of having stork are between the ages of 40-81, and for the males most of them are between 43-81 and from the graph we can see that it almost both female and males almost have similar results.

We can see from the plots that the gender is not a feature that discriminate a person having a stroke or not.

9. Conclusion:

In this project we used Stroke prediction data set from Kaggle to predict stroke, since it is important to predict if people have a stroke or not, we did several steps.
First, we overview about data set using EDA to understand the relationship between columns.
Then we cleaned our data set and we do feature engineering to prepare the dataset for classification.
lastly, when we use a classification technique, we face serval difficulties in this project one of them was that our data set is highly unbalanced, so we needed to balance the data by using SMOTE to have better results in our models and decide which model is more accurate.
Finally, we conducted that the KNN model mostly has the best result in predicting stroke and we answer serval important questions using the data set.

10. Resources:

https://medium.com/@myselfaman12345/c-and-gamma-in-svm-e6cee48626b
https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-datas