SP-IMLA

STROKE PREDICTION USING AN INTEGRATED MACHINE LEARNING APPROACH

- MRITUNJAY MEHTA
- ASHIMA GARG

INTRODUCTION

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths.

First what is a stroke?

- Stroke is a medical emergency. A stroke occurs when blood flow to a part of your brain is interrupted or reduced, preventing brain tissue from getting oxygen and nutrients. Brain cells begin to die within minutes.

Risk Factors

- Age: People aged 55 years and over
- Hypertension: if the systolic pressure is 140 mm Hg or more, or the diastolic pressure is 90 mm Hg or more
- Hypercholesterolemia: If the cholesterol level in the blood is 200 milligrams per decilitre
- Smoking
- Diabetes
- Obesity: if the body mass index (BMI) is 30 or more

MOTIVATION

Stroke is becoming an important cause of premature death and disability in low-income and middle-income countries like India, largely driven by demographic changes and enhanced by the increasing prevalence of the key modifiable risk factors. As a result, developing countries are exposed to a double burden of both communicable and noncommunicable diseases. The poor are increasingly affected by stroke, because of both the changing population exposures to risk factors and, most tragically, not being able to afford the high cost for stroke care. Majority of stroke survivors continue to live with disabilities, and the costs of on-going rehabilitation and long term-care are largely undertaken by family members, which impoverish their families. Being able to predict the possibility of a stroke is highly desirable for clinicians. This allows clinicians to set reasonable goals with patients and relatives, and to reach shared before-care decisions for recovery or rehabilitation.

Techniques/ Technologies used

- Python
- Skit-learn
- Pandas
- Matplotlib
- Logistic Regression
- K-Means Clustering
- One hot encoding
- Jupyter Notebook

Some other basic data science knowledge

Facilities Required

System Requirements for the proposed work

Ram: minimum 8-16 Gb

CPU: Intel Corei5 7 th Generation or more is preferred.

Storage: minimum 256gb or more (1TB preferred).

HIGH SPEED INTERNET CONNECTIVITY is preferred

Methodology / Planning of Work

01 Exploratory Data Analysis and Cleaning

02 Using logistic regression & Kmeans Clustering Algorithms to accurately predict stroke occurrence using dataset.

03 Confusion Matrix for each & every algorithms for better future analysis & comparison.

04 Collection, Representation, Proper documentation & Integration of Data & results found

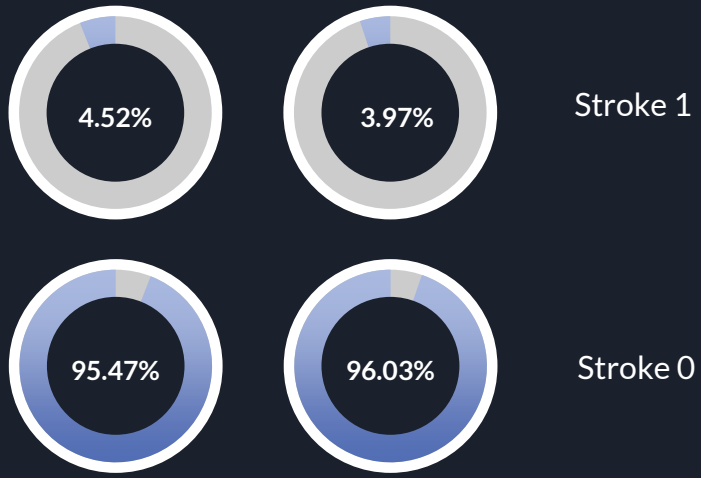
Exploratory Data Analysis

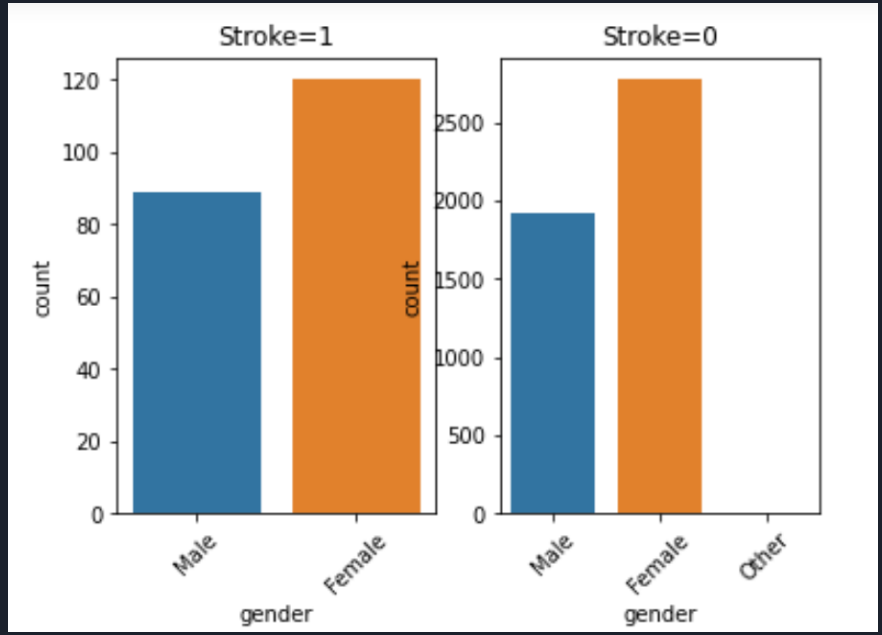
Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

It is a good practice to understand the data first and try to gather as many insights from it. EDA is all about making sense of data in hand,before using that data for analysis.

Observations based on….

Gender

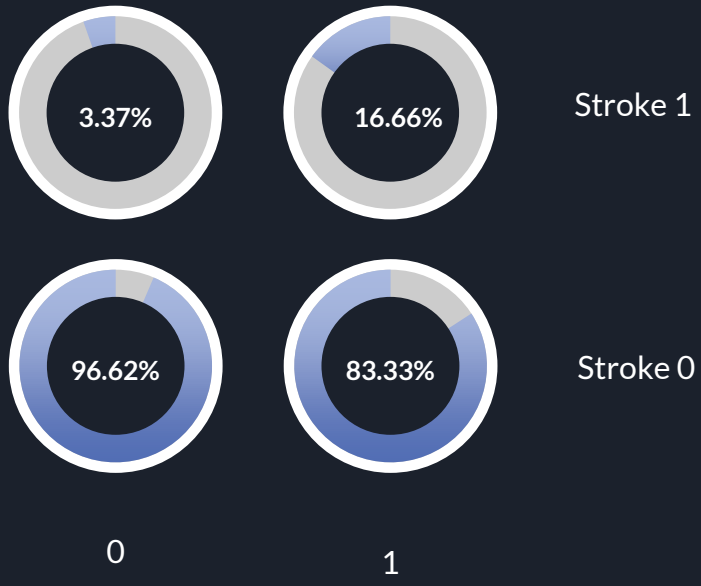


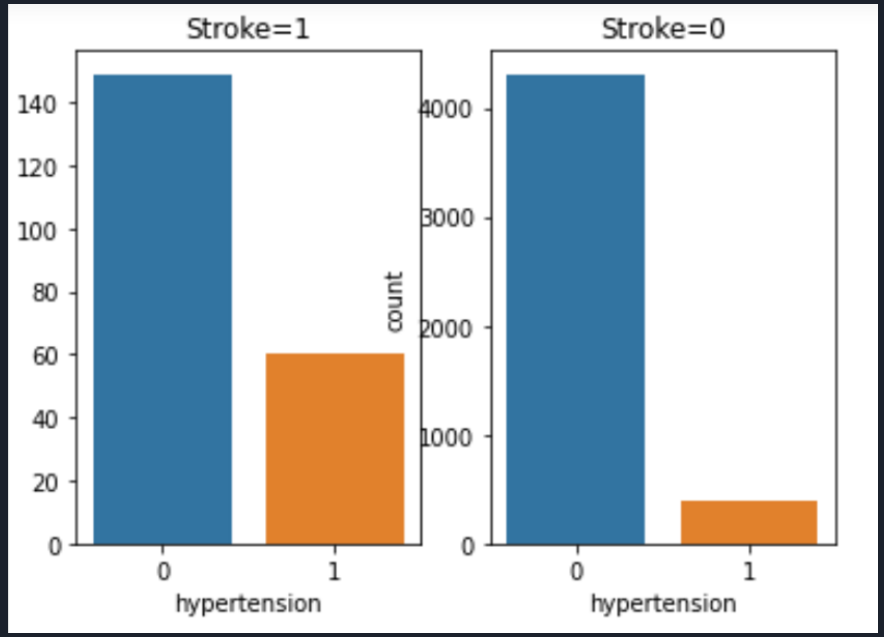


- Men are more prone to strokes than women

Observations based on….

Hypertension

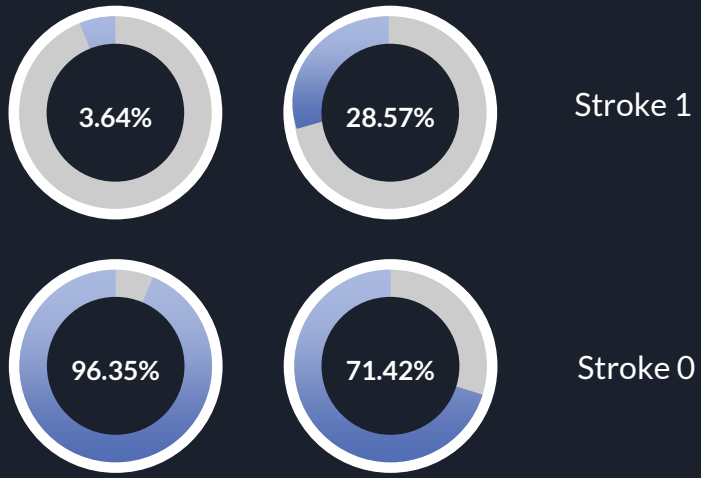


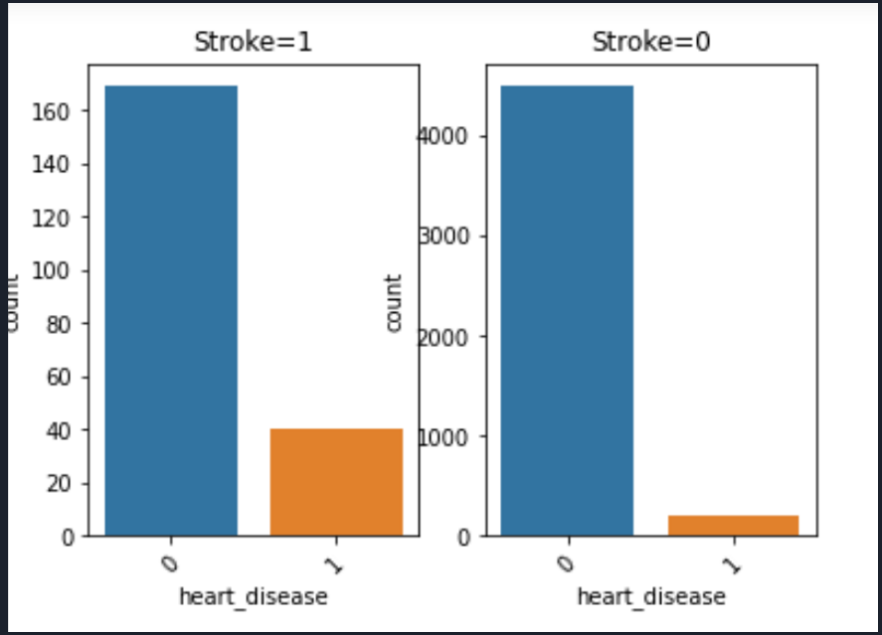


- People having hypertension are more prone to strokes.

Observations based on….

Heart Disease

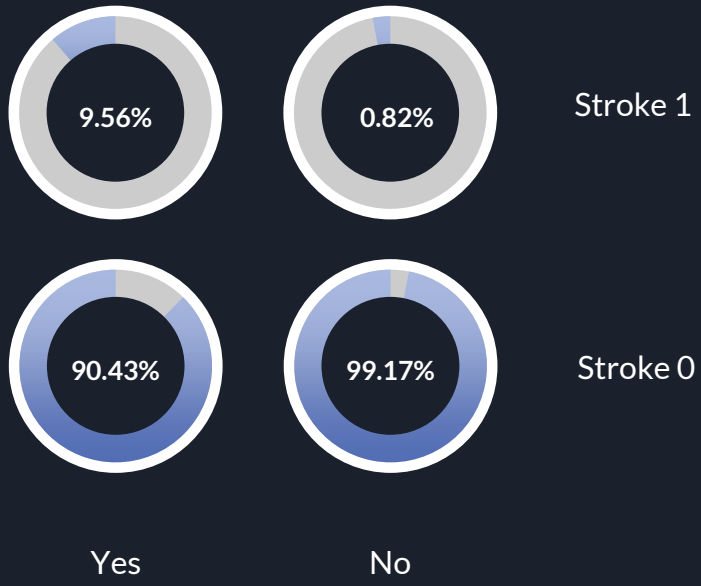


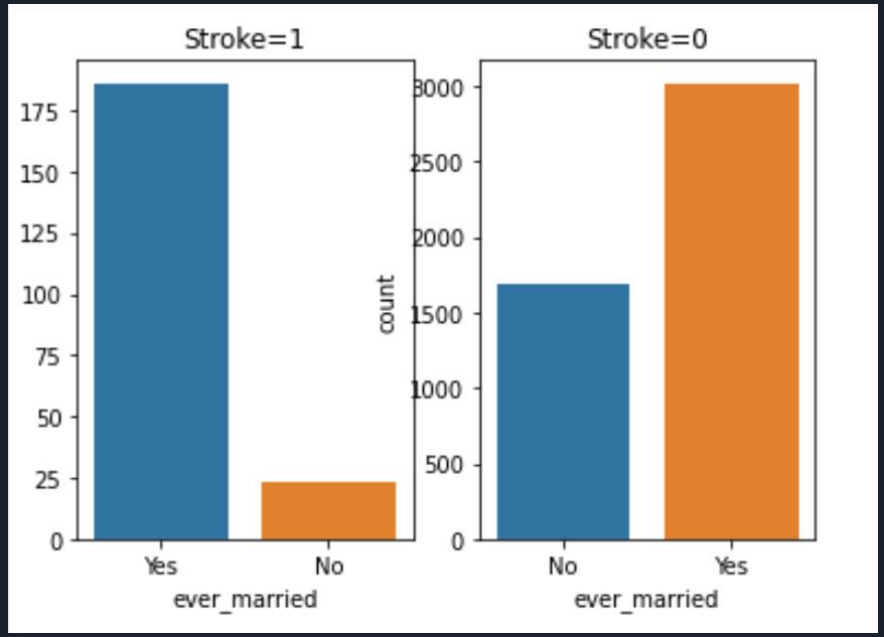


- People having heart disease are are very much likely to develop stroke attacks.

Observations based on….

Ever Married

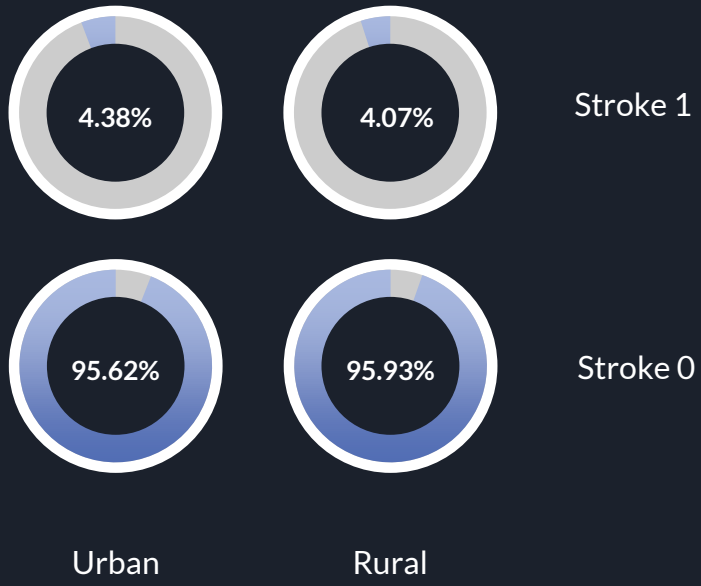


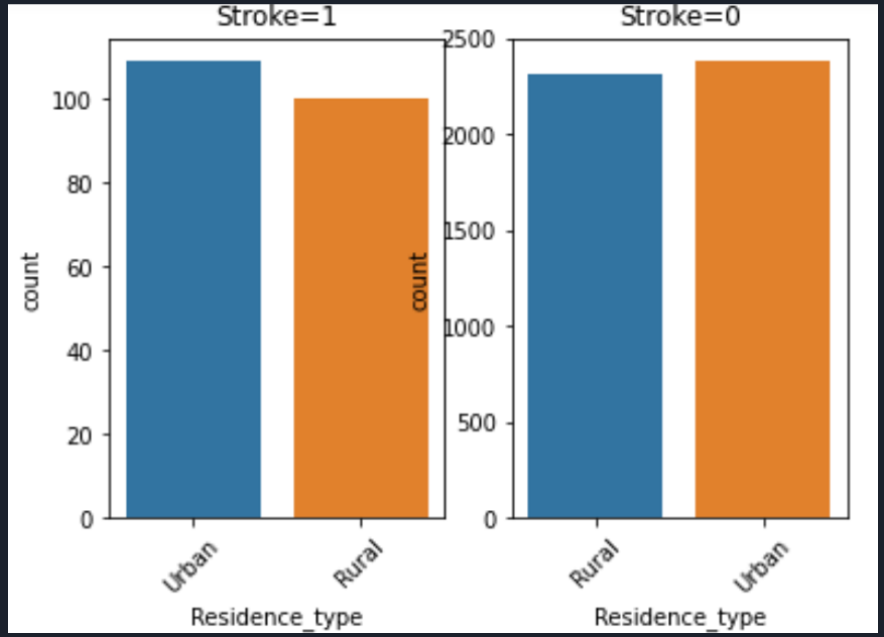


- Married people are more prone to have stroke attacks.

Observations based on….

Residence Type

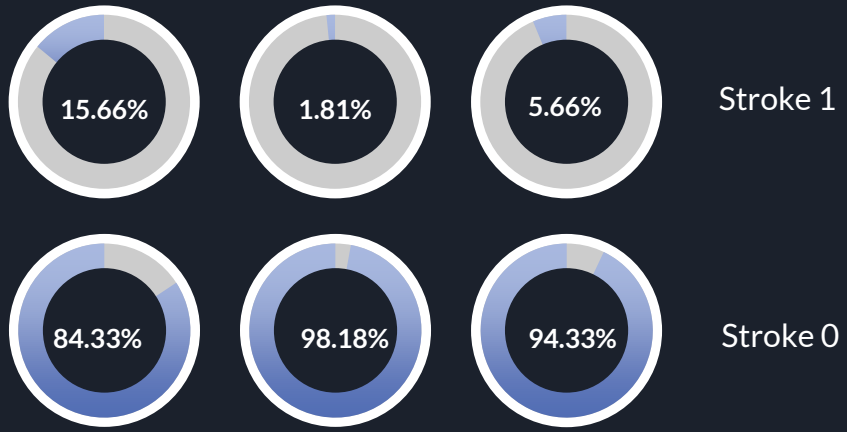


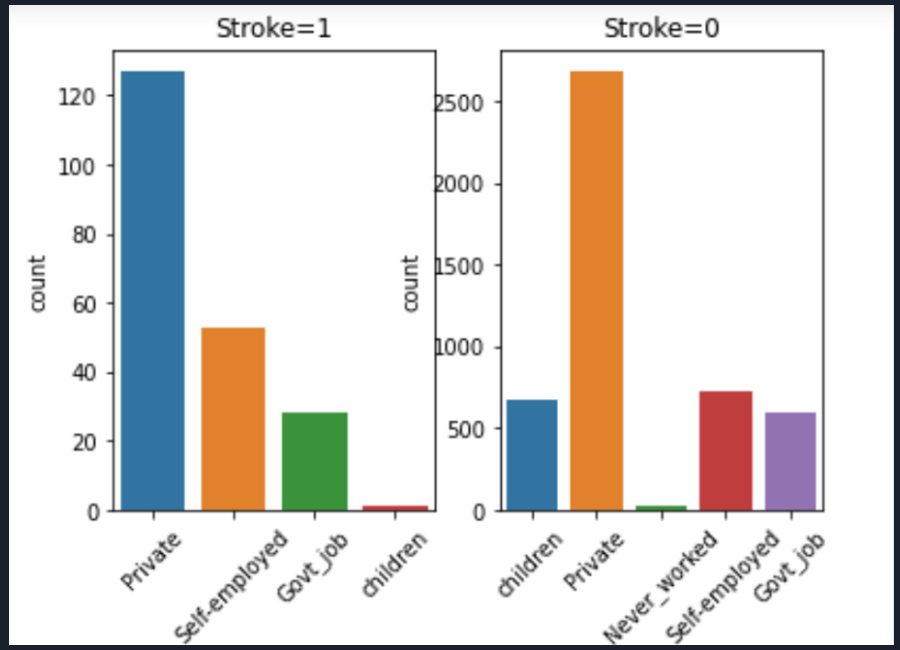


- Residence type doesn't show much variation in stroke attacks.
- Stroke attacks are slightly more likely to happen in urban areas than rural areas.

Observations based on....

Job Type





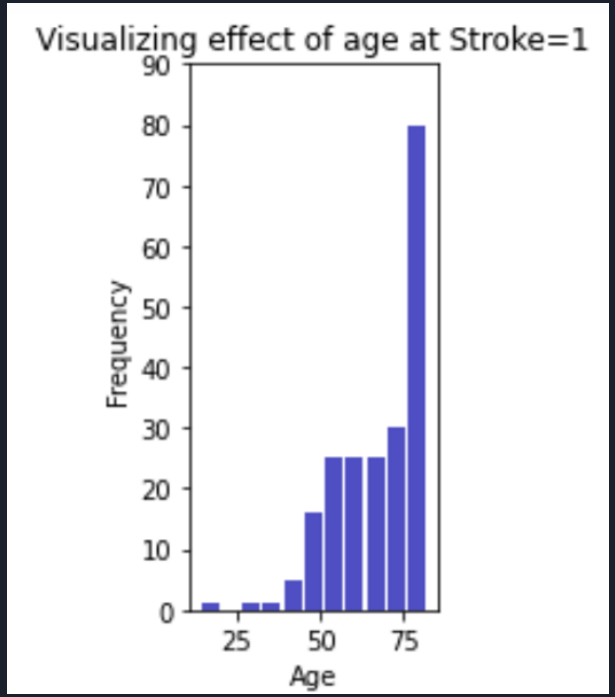
- People having private jobs are most prone to have stroke attacks followed by people having Govt-jobs.
- People who are self employed are least prone to have stroke attacks.

Observations based on….

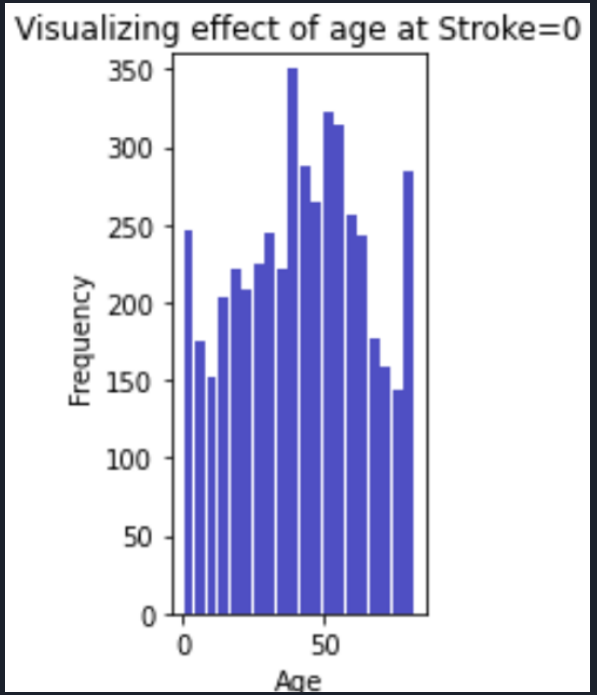
Age

Notice that…

- People below age 25 are least prone to strokes.
- People above age 75 are more prone to strokes.



Stroke 1



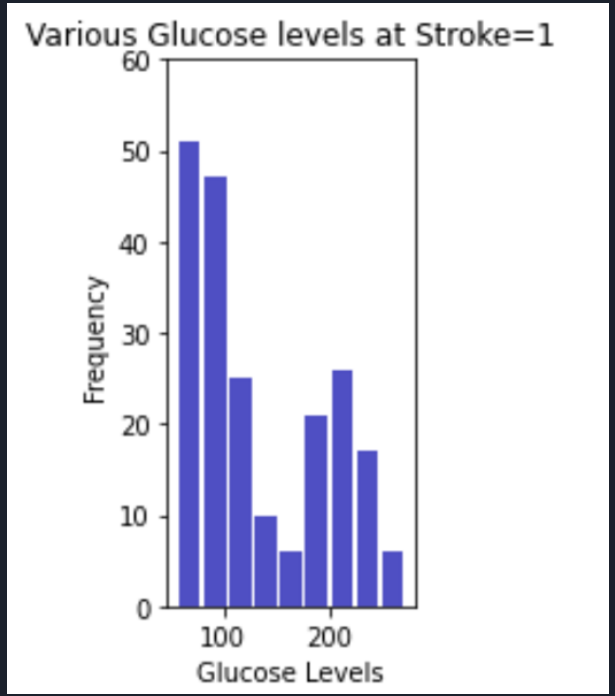
Stroke 0

Observations based on….

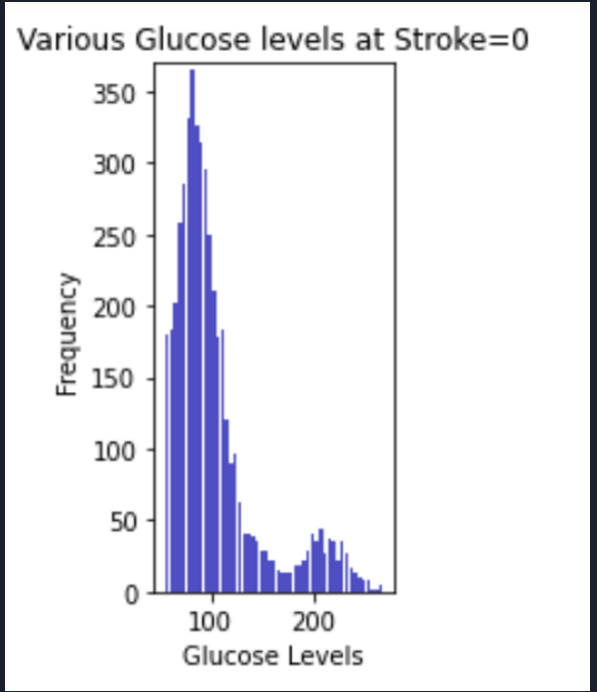
Glucose Levels

Notice that…

- People having lower glucose level are more prone to strokes.
- People having higher glucose levels are less prone to strokes.



Stroke 1



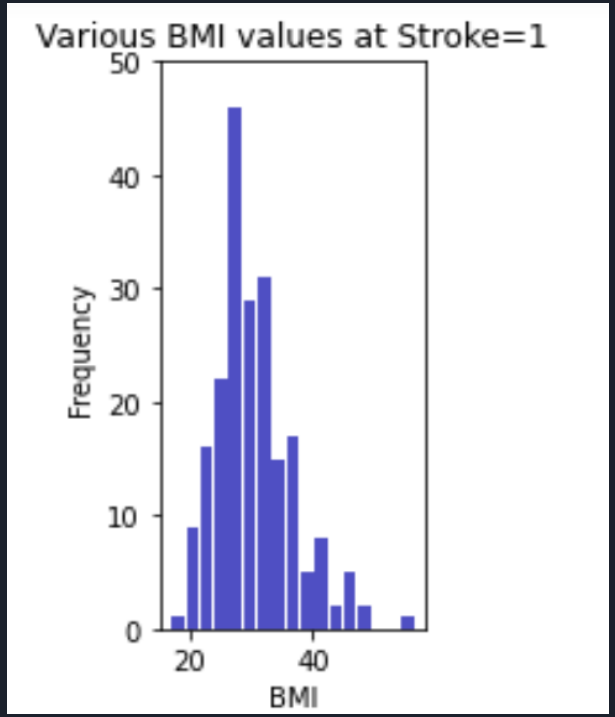
Stroke 0

Observations based on….

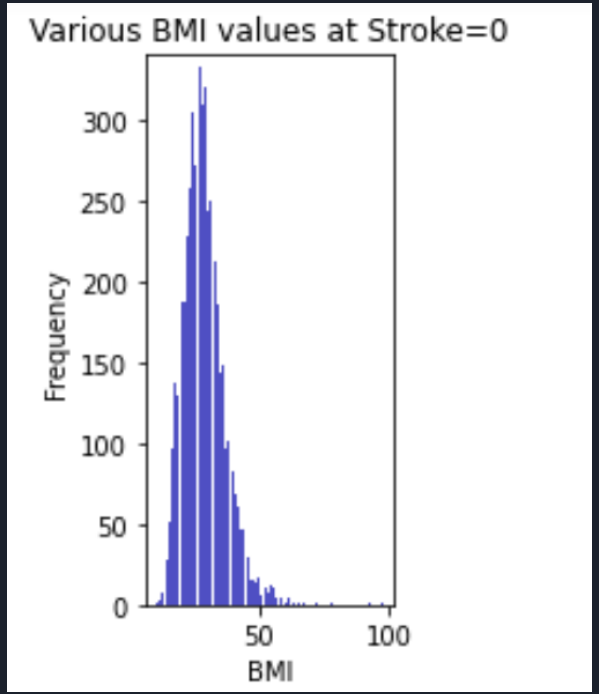
BMI

Notice that…

- People having BMI (body mass index) between 20-40 are most prone to strokes as comparatively to other people



Stroke 1



Stroke 0

Why One-Hot Encode Data in Machine Learning?

- ➔ Categorical data are variables that contain label values rather than numeric values.
- ➔ The number of possible values is often limited to a fixed set.
- ➔ Each value represents a different category.
- ➔ Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.
- ➔ In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.
- ➔ This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

One-hot Encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

It is the most widespread approach, and it works very well unless your categorical variable takes on a large number of values.

It creates new (binary) columns, indicating the presence of each possible value from the original data.

One-hot Encoding

```
gender = pd.get_dummies(df['gender'], prefix='gender')
smoking_status = pd.get_dummies(df['smoking_status'], prefix='smoking_status')
work_type = pd.get_dummies(df['work_type'], prefix='work_type')
Residence_type = pd.get_dummies(df['Residence_type'], prefix='Residence_type')
ever_married = pd.get_dummies(df['ever_married'], prefix='ever_married')
df_final = pd.merge(df_final, gender, left_index=True, right_index=True)
df_final = pd.merge(df_final, work_type, left_index=True, right_index=True)
df_final = pd.merge(df_final, smoking_status, left_index=True, right_index=True)
df_final = pd.merge(df_final, Residence_type, left_index=True, right_index=True)
df_final = pd.merge(df_final, ever_married, left_index=True, right_index=True)
df_final
```

One-hot Encoding

Out[17]:

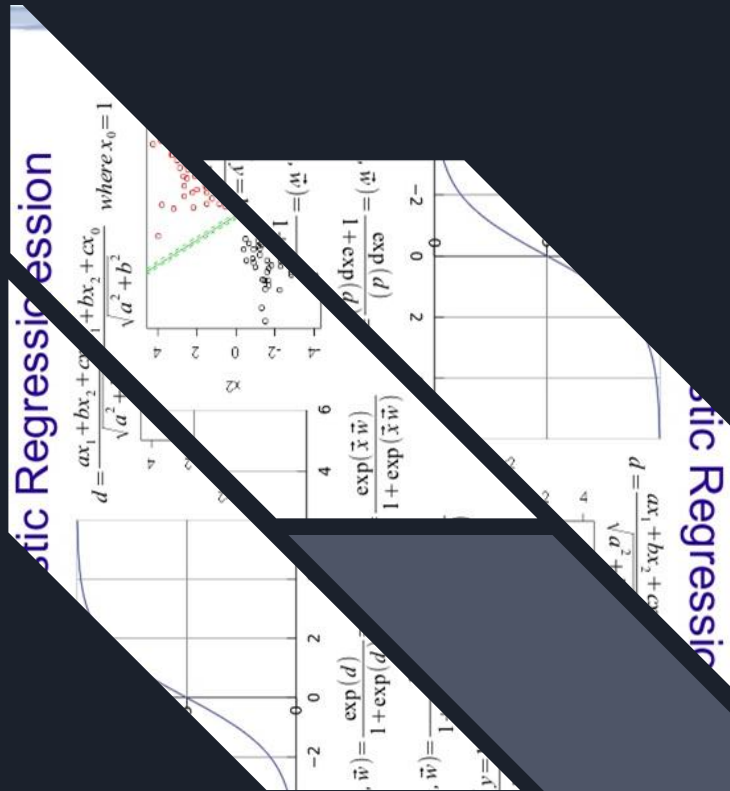
	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender_Female	gender_Male	gender_Other	work_type_Govt_job	...	work_type_empl
0	67.0	0	1	228.69	36.6	1	0	1	0	0	...	
2	80.0	0	1	105.92	32.5	1	0	1	0	0	...	
3	49.0	0	0	171.23	34.4	1	1	0	0	0	...	
4	79.0	1	0	174.12	24.0	1	1	0	0	0	...	
5	81.0	0	0	186.21	29.0	1	0	1	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
5104	13.0	0	0	103.08	18.6	0	1	0	0	0	...	
5106	81.0	0	0	125.20	40.0	0	1	0	0	0	...	
5107	35.0	0	0	82.99	30.6	0	1	0	0	0	...	
5108	51.0	0	0	166.29	25.6	0	0	1	0	0	...	
5109	44.0	0	0	85.28	26.2	0	1	0	0	1	...	

4909 rows × 22 columns

Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist.

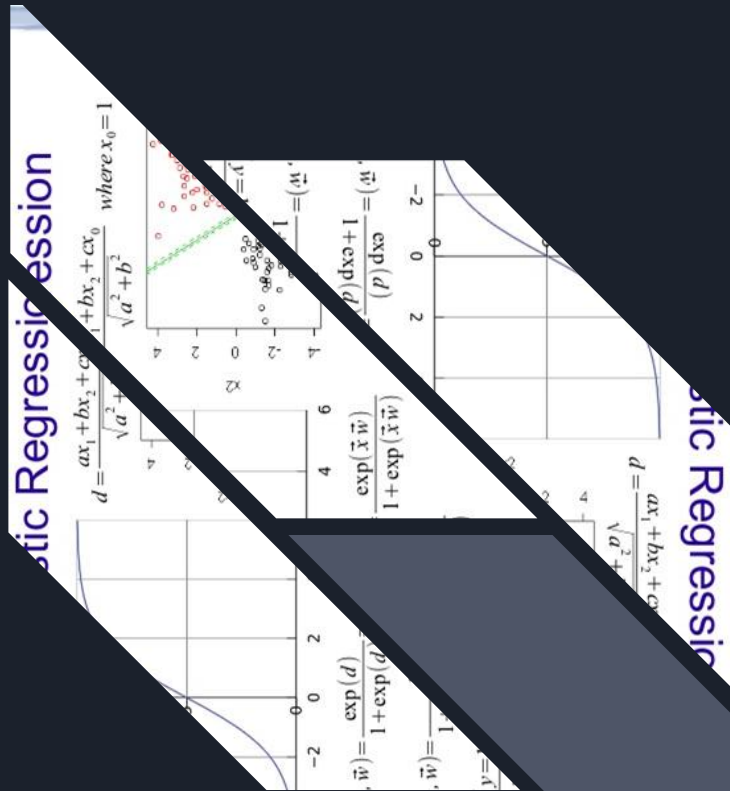
In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).



Application : Logistic Regression

Logistic regression is used in various fields, including machine learning, most medical fields, and social sciences.

Logistic regression may be used to predict the risk of developing a given disease (e.g. diabetes; coronary heart disease), based on observed characteristics of the patient (age, sex, body mass index, results of various blood tests, etc.).



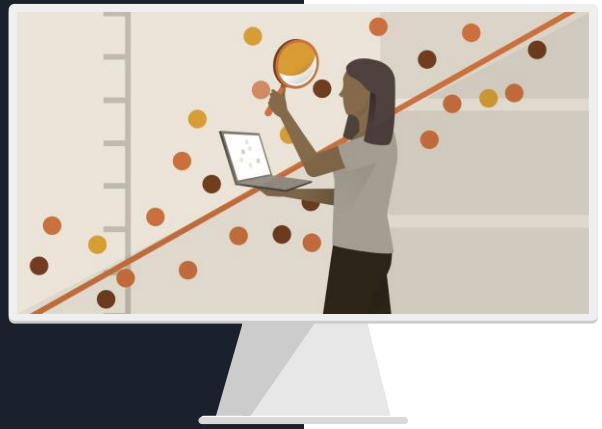
Logistic Regression

Pros

- Simple to implement.
- Effective
- Feature scaling not needed:

  Does not require input features to be scaled (can work with scaled features too, but doesn't require scaling)

- Tuning of hyperparameters not needed.



Cons

- Poor performance on non-linear data (image data for e.g.)
- **Poor performance with irrelevant and highly correlated features**
- Not very powerful
- **High reliance on proper presentation of data.**

  All the important variables / features should be identified for it to work well.

Logistic Regression

```
stroke1=list(df['stroke'])
count=0
for i in range(len(stroke1)):
    if count>211:
        if stroke1[i]==0:
            stroke1[i]=2
    if stroke1[i]==0:
        count+=1
df_final=df_final.drop(['stroke'],axis=1)
df_final['stroke']=stroke1
df_final= df_final[df_final['stroke'] != 2]
y_train=np.array(list(df_final['stroke']))
df_final=df_final.drop(['stroke'],axis=1)
```

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

Logistic Regression

In [20]:
```
df_final
```

Out[20]:

	age	hypertension	heart_disease	avg_glucose_level	bmi	gender_Female	gender_Male	gender_Other	work_type_Govt_job	work_type_Never_worke
0	67.0	0	1	228.69	36.6	0	1	0	0	
2	80.0	0	1	105.92	32.5	0	1	0	0	
3	49.0	0	0	171.23	34.4	1	0	0	0	
4	79.0	1	0	174.12	24.0	1	0	0	0	
5	81.0	0	0	186.21	29.0	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	
459	11.0	0	0	87.51	24.4	1	0	0	0	
460	7.0	0	0	72.35	17.0	1	0	0	0	
461	16.0	0	0	113.47	19.5	1	0	0	0	
462	44.0	0	0	103.78	49.8	1	0	0	0	
463	78.0	0	0	115.43	27.8	1	0	0	0	

421 rows × 21 columns

Logistic Regression

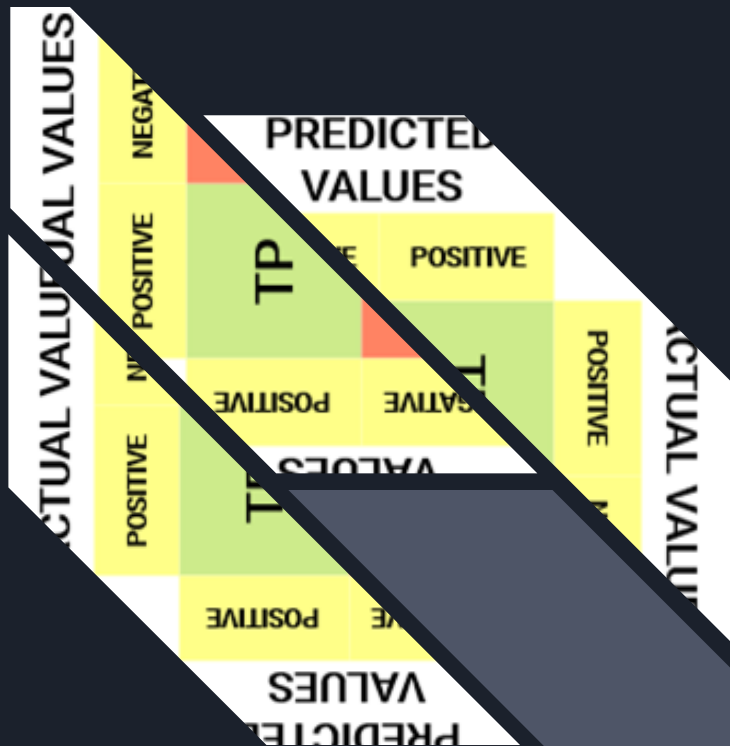
```
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
count=0
for i in range(len(y_test)):
    if y_pred[i]==y_test[i]:
        count+=1
print(count*100/len(y_pred))
```

```
94.60285132382892
```

*Accuracy came out to be **94.60%**.*

Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.



Confusion Matrix

```
from sklearn.metrics import confusion_matrix
res=confusion_matrix(y_train, y_test)
res
```

```
array([[153,  59],
       [ 46, 163]])
```

Confusion Matrix

Out[55]:
```
array([[153,  59],
       [ 46, 163]])
```

In [56]:
```
TN=res[1][1]
TP=res[0][0]
FN=res[1][0]
FP=res[0][1]
print(TP,FP,FN,TN)
```

```
153 59 46 163
```

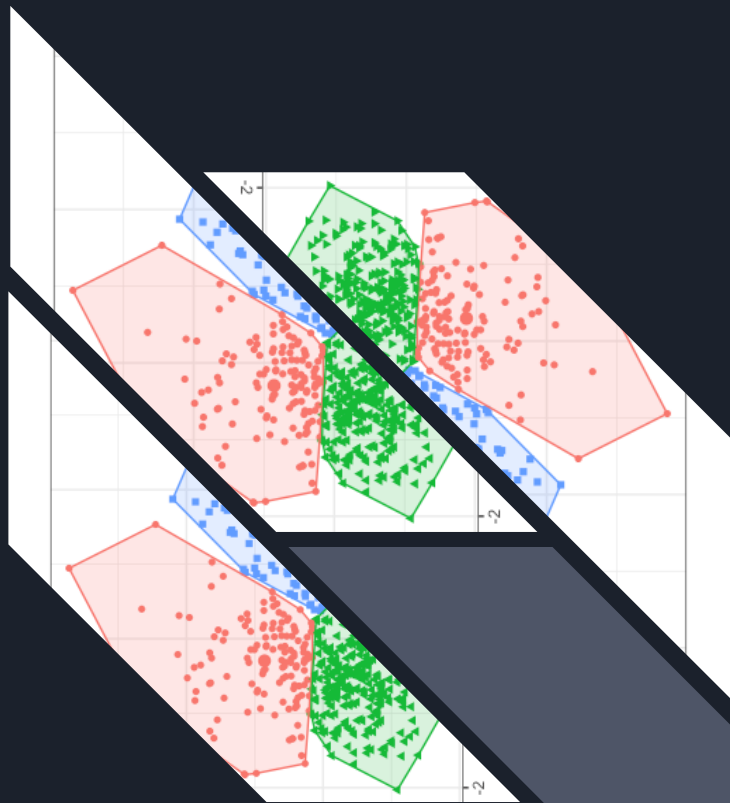
In [57]:
```
Sensitivity=(TP)/(TP+FN)
Specificity=(TN)/(TN+FP)
Precision=(TP)/(TP+FP)
Recall=(TP)/(TP+FN)
f1=2*(Precision*Recall)/(Precision+Recall)
```

In [58]:
```
print('Sensitivity: '+str(Sensitivity))
print('Specificity: '+str(Specificity))
print('Precision: '+str(Precision))
print('Recall: '+str(Recall))
print('f1: '+str(f1))
```

```
Sensitivity: 0.7688442211055276
Specificity: 0.7342342342342343
Precision: 0.7216981132075472
Recall: 0.7688442211055276
f1: 0.7445255474452555
```

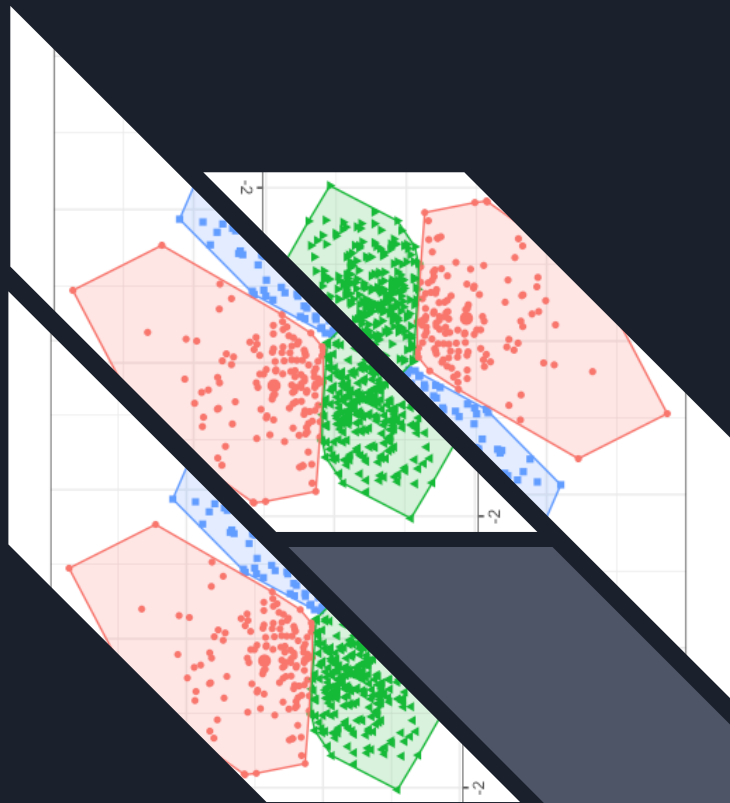
K-Means Clusturing

- *K means is one of the most popular Unsupervised Machine Learning Algorithms Used for Solving Classification Problems.*
- ***K Means segregates the unlabeled data into various groups, called clusters, based on having similar features, common patterns.***
- *A cluster refers to a collection of data points aggregated together because of certain similarities.*



How it works

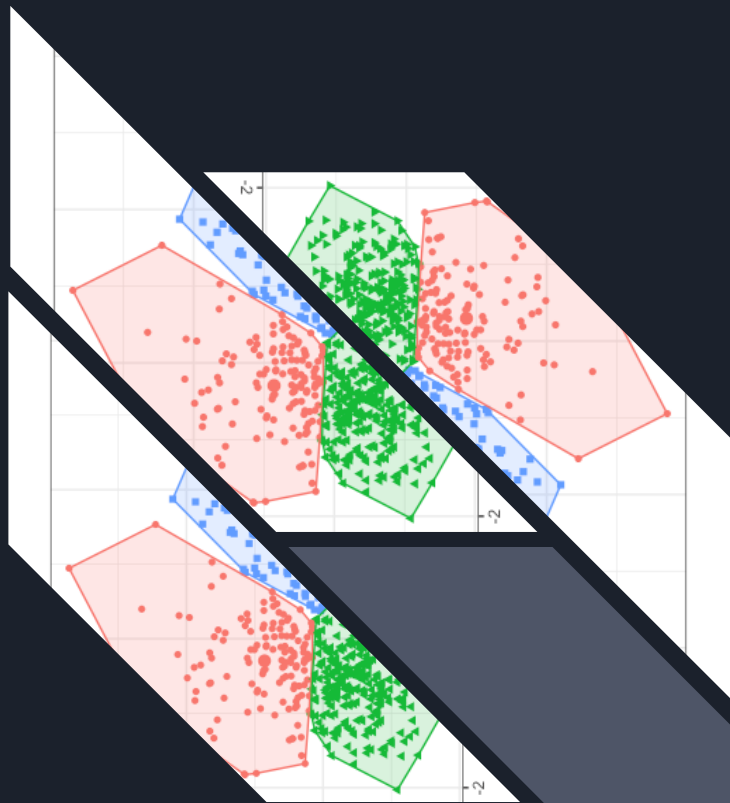
To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids



How it works

It halts creating and optimizing clusters when either:

- *The centroids have stabilized — there is no change in their values because the clustering has been successful.*
- *The defined number of iterations has been achieved.*



K-Means Clustering

Pros

Relatively simple to implement.

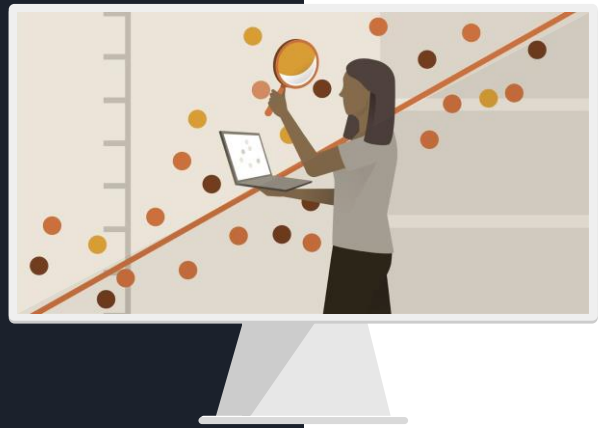
Scales to large data sets.

Guarantees convergence.

Can warm-start the positions of centroids.

Easily adapts to new examples.

Generalizes to clusters of different shapes and sizes, such as elliptical clusters.



Cons

Choosing manually.

Being dependent on initial values.

Clustering data of varying sizes and density.

Clustering outliers.

Scaling with number of dimensions.

Finding “k”

```
# K-means Clustering
```

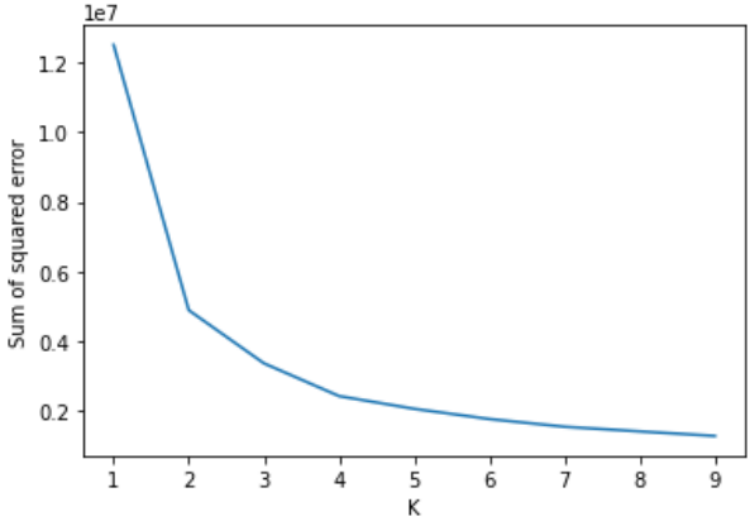
```
import sklearn
from sklearn.cluster import KMeans
```

```
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df_final[columns],y_train)
    sse.append(km.inertia_)
```

Elbow Plot (Finding “k”)

```
#elbow plot
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

```
[<matplotlib.lines.Line2D at 0x1a86a9a1ac0>]
```



Training the Model

```
KMeans_Clustering = KMeans(n_clusters =2, random_state=42)
KMeans_Clustering.fit(X_train)
```

```
KMeans(n_clusters=2, random_state=42)
```

```
print(KMeans_Clustering.cluster_centers_)
```

```
[[ 4.02612589e+01  6.73990499e-02  3.35510689e-02  8.94827821e+01
   2.82233670e+01  6.01247031e-01  3.98456057e-01  2.96912114e-04
   1.27375297e-01  5.34441805e-03  5.72743468e-01  1.41627078e-01
   1.52909739e-01  3.22149644e-01  1.55878860e-01  3.72624703e-01
   1.49346793e-01  4.94952494e-01  5.05047506e-01  3.80938242e-01
   6.19061758e-01]
 [ 5.78120215e+01  2.45080501e-01  1.34168157e-01  2.00527406e+02
   3.24576029e+01  5.40250447e-01  4.59749553e-01 -3.79470760e-19
   1.52057245e-01  1.78890877e-03  5.84973166e-01  2.30769231e-01
   3.04114490e-02  1.80679785e-01  2.45080501e-01  4.27549195e-01
   1.46690519e-01  4.97316637e-01  5.02683363e-01  1.37745975e-01
   8.62254025e-01]]
```

Classification Report

```
#prediction using kmeans and accuracy
kpred = KMeans_Clustering.predict(X_test)
print('Classification report:\n\n', sklearn.metrics.classification_report(y_test,kpred))
```

```
Classification report:

              precision    recall  f1-score   support

           0       0.96      0.87      0.91       929
           1       0.12      0.30      0.17        53

    accuracy                           0.84       982
   macro avg       0.54      0.58      0.54       982
weighted avg       0.91      0.84      0.87       982
```

Confusion Matrix

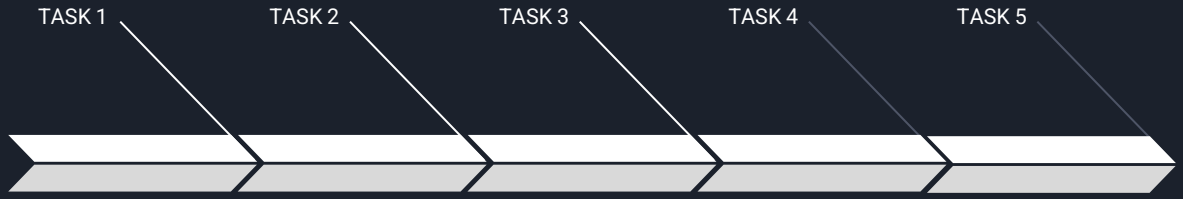
*Accuracy came out to be **83.70%**.*

```
Sensitivity=(TP)/(TP+FN)
Specificity=(TN)/(TN+FP)
Precision=(TP)/(TP+FP)
Recall=(TP)/(TP+FN)
Accuracy=(TP+TN)/(TP+TN+FP+FN)
f1=2*(Precision*Recall)/(Precision+Recall)

print('Sensitivity: '+str(Sensitivity))
print('Specificity: '+str(Specificity))
print('Precision: '+str(Precision))
print('Recall: '+str(Recall))
print('f1: '+str(f1))
print('Accuracy: '+str(Accuracy))
```

```
Sensitivity: 0.9561091340450771
Specificity: 0.11510791366906475
Precision: 0.8675995694294941
Recall: 0.9561091340450771
f1: 0.909706546275395
Accuracy: 0.8370672097759674
```

Project timeline



TASK 1	TASK 2	TASK 3	TASK 4	TASK 5
Research about the project	Exploratory Data Analysis and Cleaning	Using various methods to accurately predict stroke occurrence using dataset	Collection & Representation of Data found	Proper Documentation & Integration

Gantt Chart :

Task	Task 1	Task 2	Task 3	Task 4	Task 5
Start					
Week 1					
Week 2					
Week 3					
Week 4					
Week 5					
Week 6					
Week 7					
Week 8					
Week 9					
Week 10					
Week 11					
Week 12					
End					