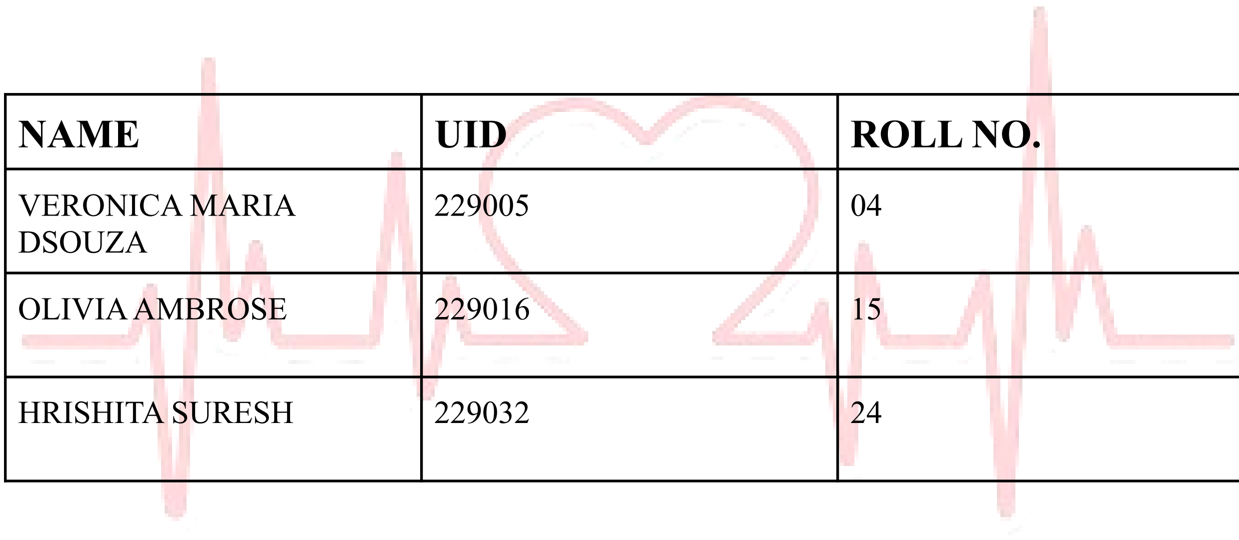
STROKE PREDICTION WEB-APP DEPLOYMENT VIA STREAMLIT

PROJECT BY:

NAME	**UID**	**ROLL NO.**
VERONICA MARIA DSOUZA	229005	04
OLIVIA AMBROSE	229016	15
HRISHITA SURESH	229032	24



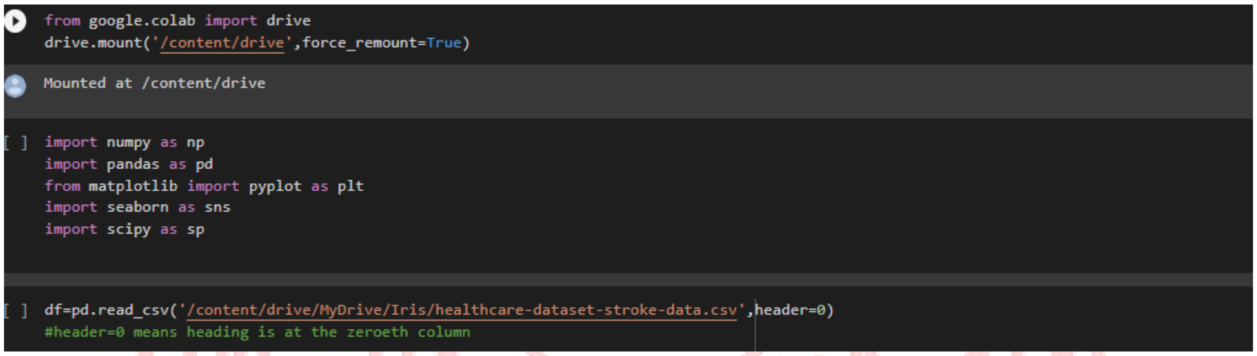
SYNOPSIS:

WHAT DOES OUR WEB APPLICATION EXACTLY DO?

→ Our web application accepts user inputs about various health related queries such as the age, smoking status, bmi, etc., of an individual and then with the help of our machine learning model built via PYTHON, it makes a prediction about how likely an individual is to have a stroke in the near future.

FIRSTLY WE TAKE A LOOK AT OUR ML MODEL BUILDING:

→ STEP 1: importing necessary inbuilt packages and loading the dataset:



```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)

Mounted at /content/drive

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import scipy as sp

df=pd.read_csv('/content/drive/MyDrive/Iris/healthcare-dataset-stroke-data.csv',header=0)
#header=0 means heading is at the zeroeth column
```

→ STEP 2: Let us learn about our dataset before we get into pre-processing:

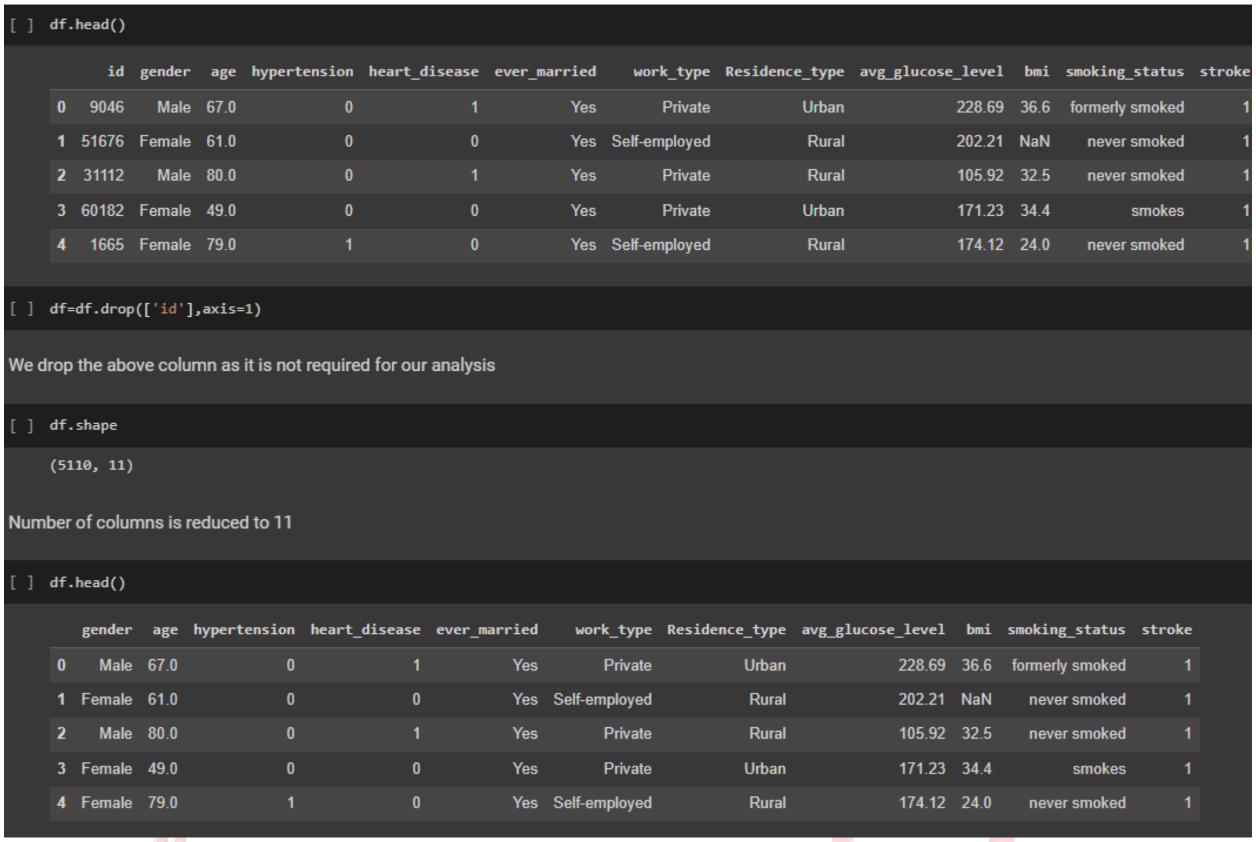
The data set contains 5110 rows and 12 columns i.e., 5110 observations and 10 columns that are used as features excluding id and 1 column used as the target i.e., stroke

Information about the columns:

1. gender: gives the gender of the individual
2. age: gives the age of the individual
3. hypertension: describes whether the individual has hypertension or not. 0 for no and 1 for yes.
4. heart_disease: describes whether the individual has heart disease or not. 0 for no and 1 for yes.
5. ever_married: describes whether the individual has ever been married or not.
6. work_type: describes the work environment of the individual
7. Residence_type: describes the area where the individual resides
8. avg_glucose_level: gives the average blood glucose level of the individual
9. bmi: gives the BMI of individual
10. smoking_status: describes whether a person has smoked or not

11. stroke: describes whether the individual has suffered a stroke or not. 0 for no and 1 for yes.

STEP 3: Snippets of preprocessing and feature engineering:



```
[ ] df.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

```
[ ] df=df.drop(['id'],axis=1)
```

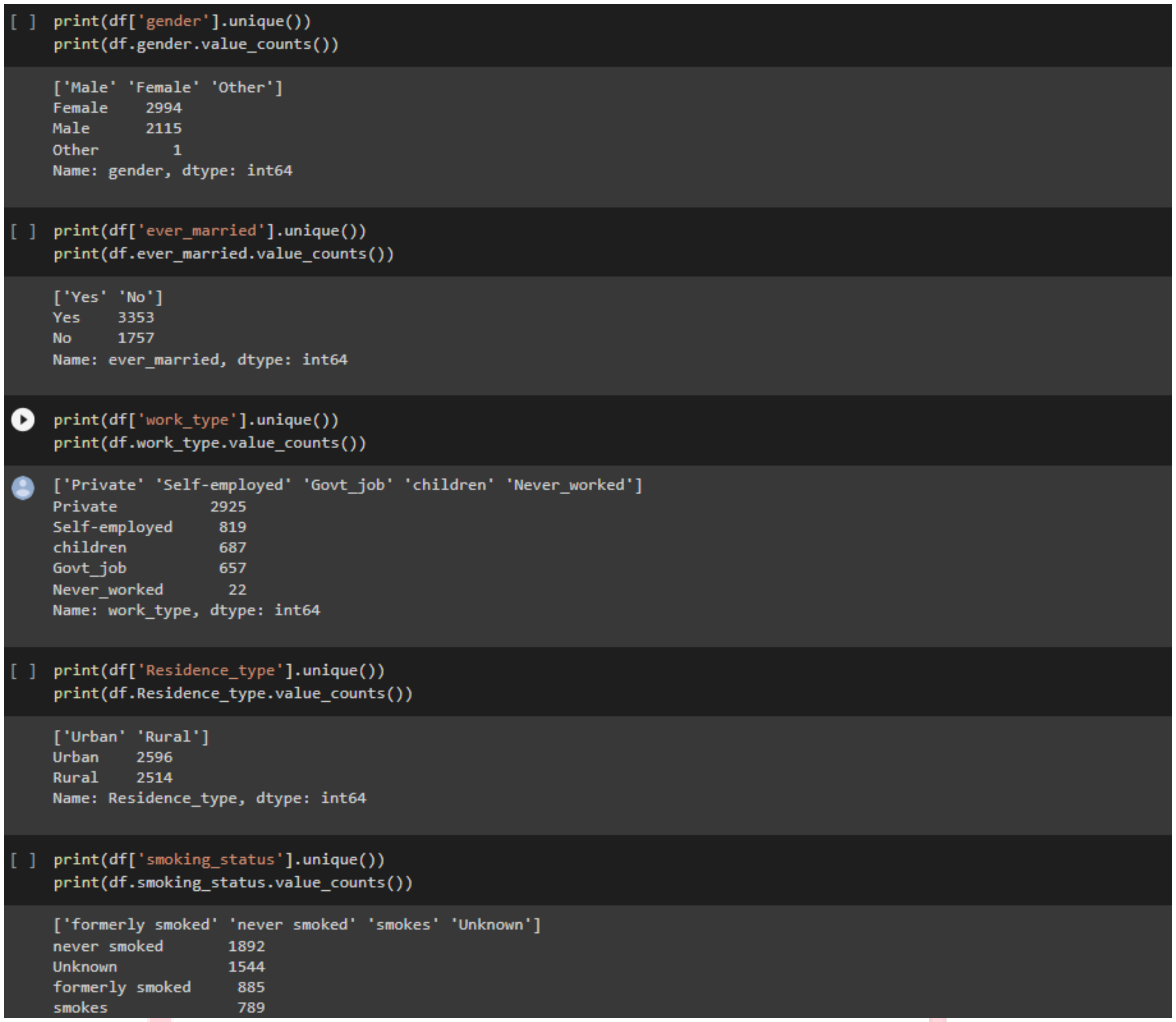
We drop the above column as it is not required for our analysis

```
[ ] df.shape

(5110, 11)
```

Number of columns is reduced to 11

```
[ ] df.head()
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1



```
print(df['gender'].unique())
print(df.gender.value_counts())
```

```
['Male' 'Female' 'Other']
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

```
print(df['ever_married'].unique())
print(df.ever_married.value_counts())
```

```
['Yes' 'No']
Yes    3353
No     1757
Name: ever_married, dtype: int64
```

```
print(df['work_type'].unique())
print(df.work_type.value_counts())
```

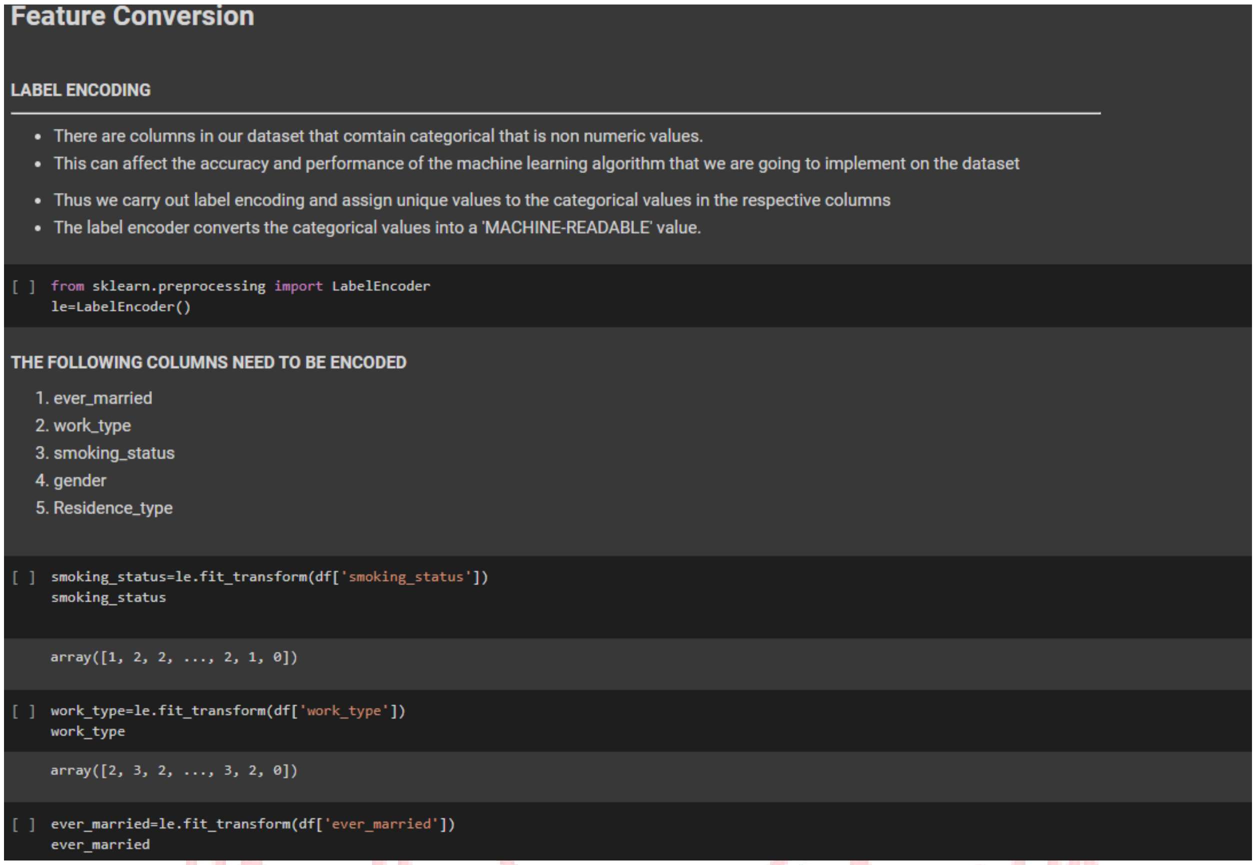
```
['Private' 'Self-employed' 'Govt_job' 'children' 'Never_worked']
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

```
print(df['Residence_type'].unique())
print(df.Residence_type.value_counts())
```

```
['Urban' 'Rural']
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64
```

```
print(df['smoking_status'].unique())
print(df.smoking_status.value_counts())
```

```
['formerly smoked' 'never smoked' 'smokes' 'Unknown']
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
```



Feature Conversion

LABEL ENCODING

- There are columns in our dataset that comtain categorical that is non numeric values.
- This can affect the accuracy and performance of the machine learning algorithm that we are going to implement on the dataset
- Thus we carry out label encoding and assign unique values to the categorical values in the respective columns
- The label encoder converts the categorical values into a 'MACHINE-READABLE' value.

```
[ ] from sklearn.preprocessing import LabelEncoder
    le=LabelEncoder()
```

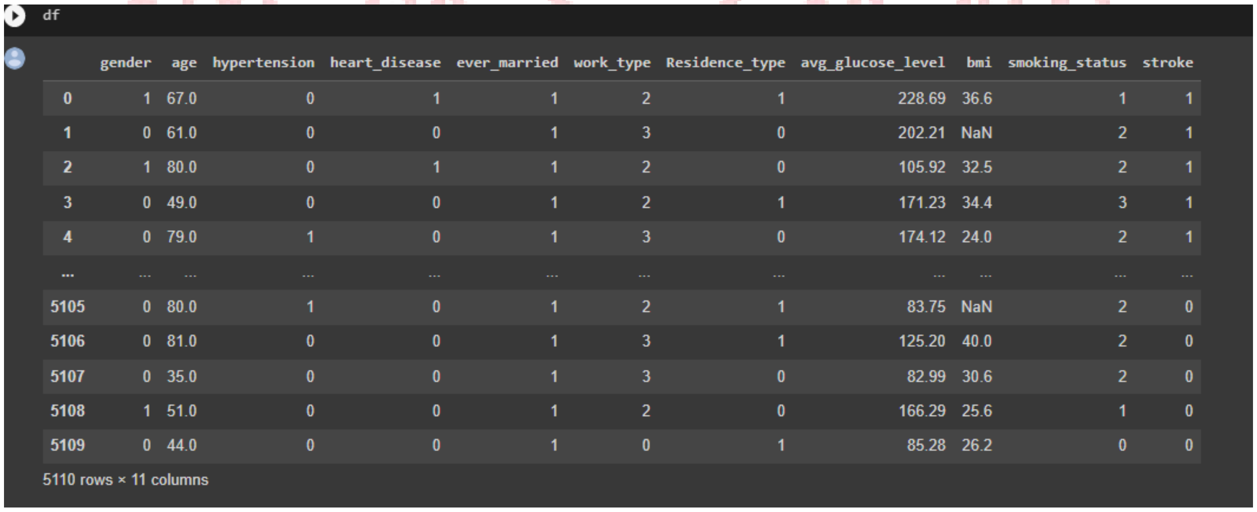
THE FOLLOWING COLUMNS NEED TO BE ENCODED

1. ever_married
2. work_type
3. smoking_status
4. gender
5. Residence_type

```
[ ] smoking_status=le.fit_transform(df['smoking_status'])
    smoking_status

    array([1, 2, 2, ..., 2, 1, 0])
```

```
[ ] work_type=le.fit_transform(df['work_type'])
    work_type

    array([2, 3, 2, ..., 3, 2, 0])
```

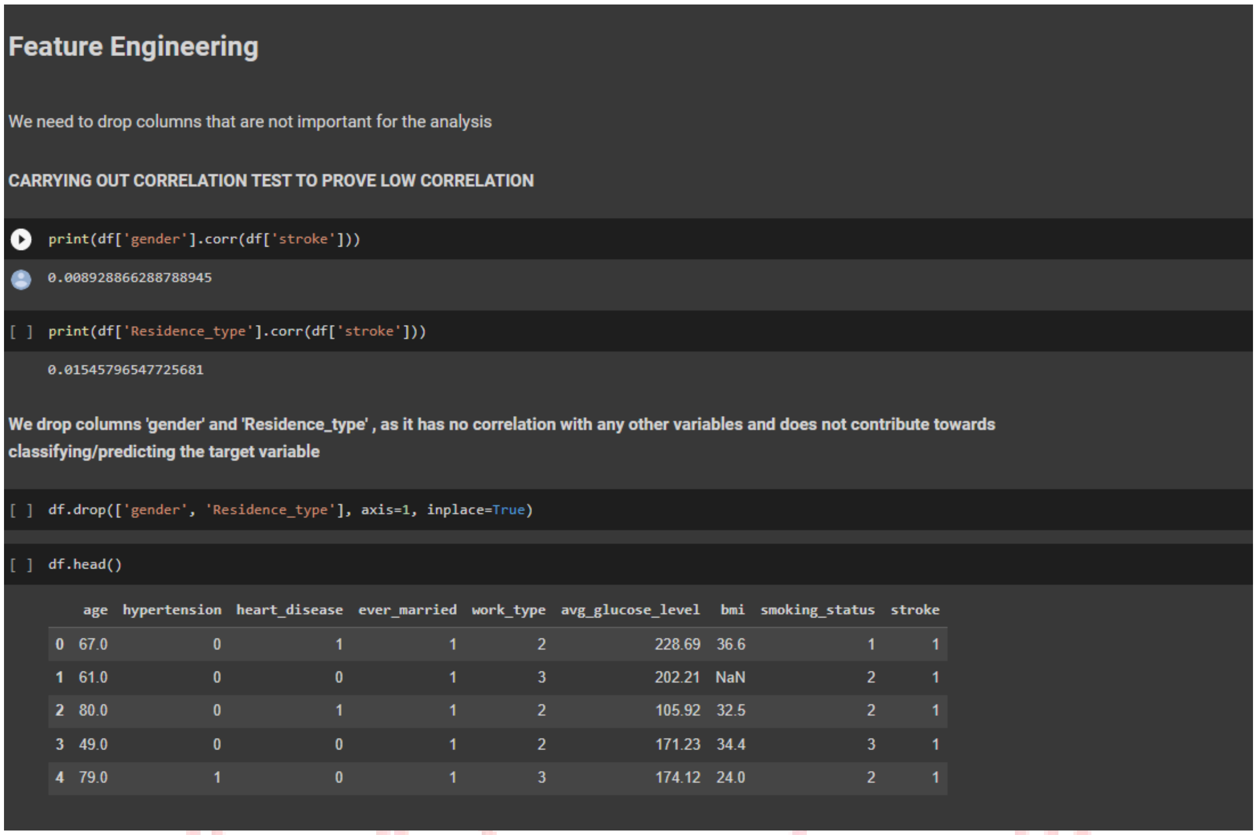
```
[ ] ever_married=le.fit_transform(df['ever_married'])
    ever_married
```



```
df
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.0	0	1	1	2	1	228.69	36.6	1	1
1	0	61.0	0	0	1	3	0	202.21	NaN	2	1
2	1	80.0	0	1	1	2	0	105.92	32.5	2	1
3	0	49.0	0	0	1	2	1	171.23	34.4	3	1
4	0	79.0	1	0	1	3	0	174.12	24.0	2	1
...	...	...	...	...	...	...	...	...	...	...	...
5105	0	80.0	1	0	1	2	1	83.75	NaN	2	0
5106	0	81.0	0	0	1	3	1	125.20	40.0	2	0
5107	0	35.0	0	0	1	3	0	82.99	30.6	2	0
5108	1	51.0	0	0	1	2	0	166.29	25.6	1	0
5109	0	44.0	0	0	1	0	1	85.28	26.2	0	0

5110 rows × 11 columns



Feature Engineering

We need to drop columns that are not important for the analysis

CARRYING OUT CORRELATION TEST TO PROVE LOW CORRELATION

```
print(df['gender'].corr(df['stroke']))
```

```
0.008928866288788945
```

```
print(df['Residence_type'].corr(df['stroke']))
```

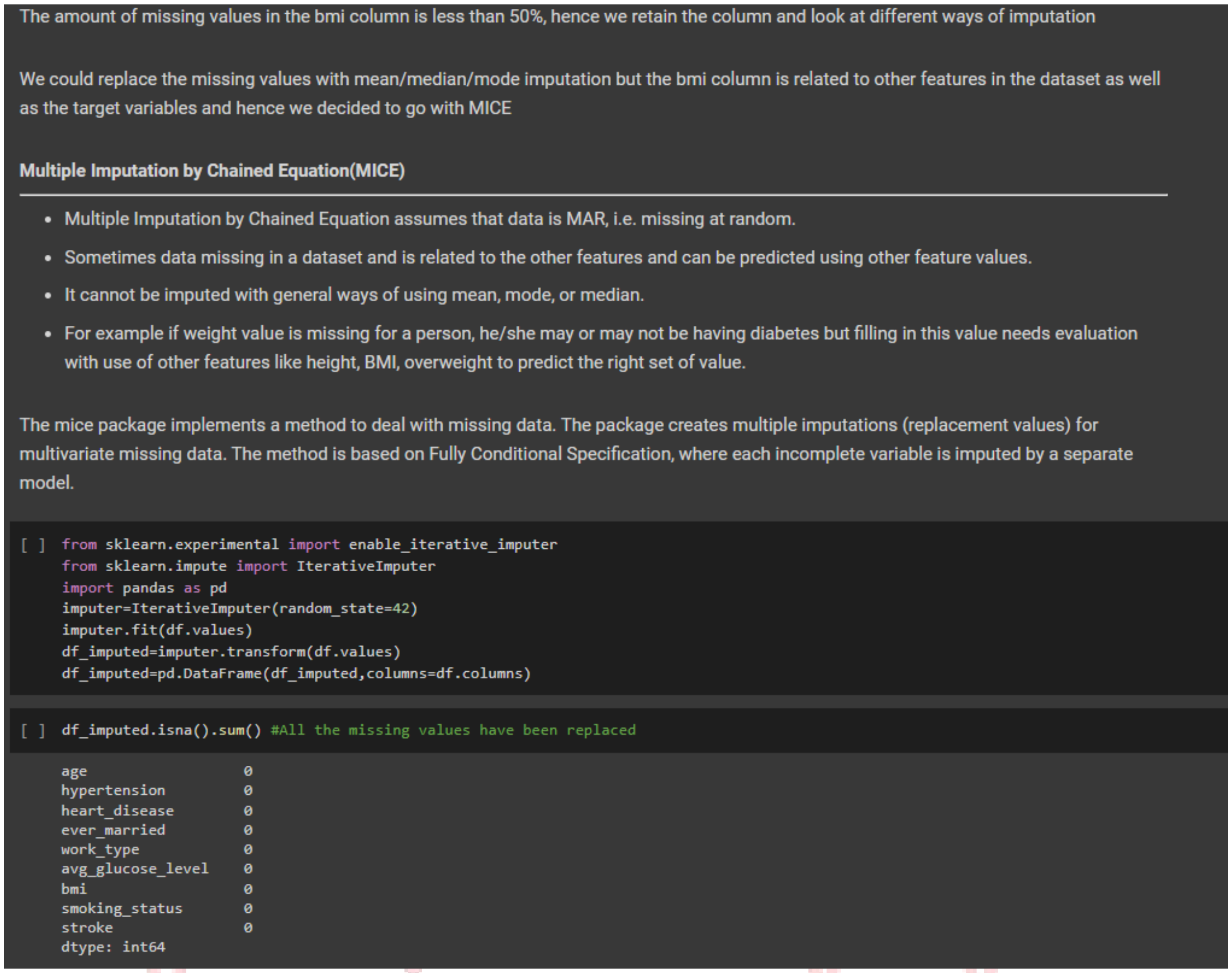
```
0.01545796547725681
```

We drop columns 'gender' and 'Residence_type' , as it has no correlation with any other variables and does not contribute towards classifying/predicting the target variable

```
df.drop(['gender', 'Residence_type'], axis=1, inplace=True)
```

```
df.head()
```

	age	hypertension	heart_disease	ever_married	work_type	avg_glucose_level	bmi	smoking_status	stroke
0	67.0	0	1	1	2	228.69	36.6	1	1
1	61.0	0	0	1	3	202.21	NaN	2	1
2	80.0	0	1	1	2	105.92	32.5	2	1
3	49.0	0	0	1	2	171.23	34.4	3	1
4	79.0	1	0	1	3	174.12	24.0	2	1


The amount of missing values in the bmi column is less than 50%, hence we retain the column and look at different ways of imputation

We could replace the missing values with mean/median/mode imputation but the bmi column is related to other features in the dataset as well as the target variables and hence we decided to go with MICE

Multiple Imputation by Chained Equation(MICE)

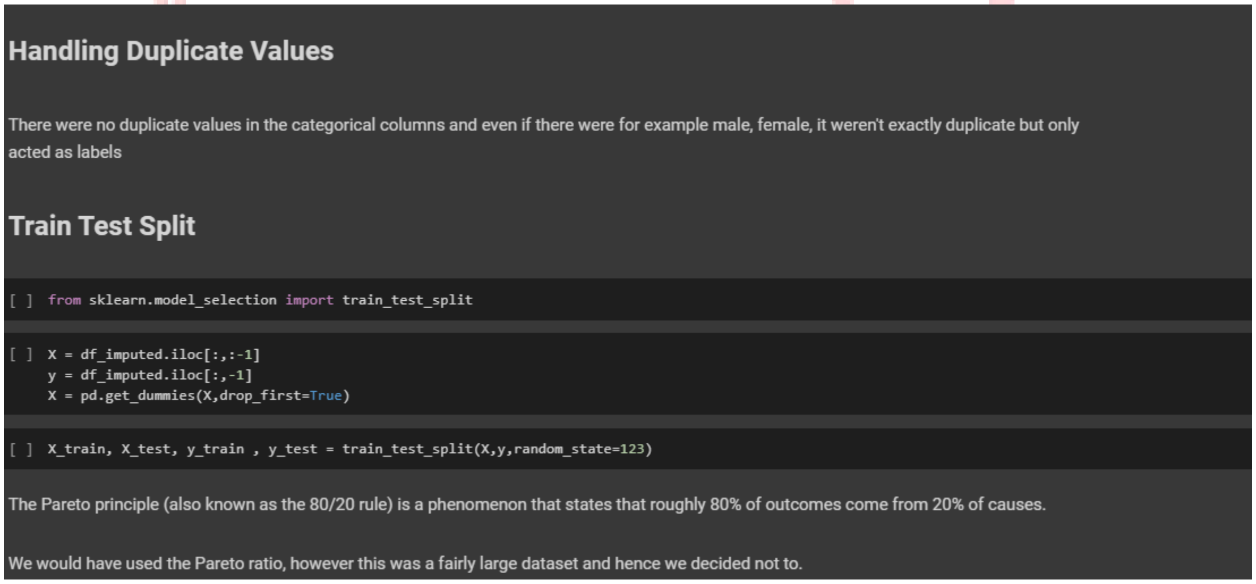
- Multiple Imputation by Chained Equation assumes that data is MAR, i.e. missing at random.
- Sometimes data missing in a dataset and is related to the other features and can be predicted using other feature values.
- It cannot be imputed with general ways of using mean, mode, or median.
- For example if weight value is missing for a person, he/she may or may not be having diabetes but filling in this value needs evaluation with use of other features like height, BMI, overweight to predict the right set of value.

The mice package implements a method to deal with missing data. The package creates multiple imputations (replacement values) for multivariate missing data. The method is based on Fully Conditional Specification, where each incomplete variable is imputed by a separate model.

```
[ ] from sklearn.experimental import enable_iterative_imputer
    from sklearn.impute import IterativeImputer
    import pandas as pd
    imputer=IterativeImputer(random_state=42)
    imputer.fit(df.values)
    df_imputed=imputer.transform(df.values)
    df_imputed=pd.DataFrame(df_imputed,columns=df.columns)
```

```
[ ] df_imputed.isna().sum() #All the missing values have been replaced
```

```
    age                  0
    hypertension         0
    heart_disease        0
    ever_married         0
    work_type            0
    avg_glucose_level    0
    bmi                  0
    smoking_status       0
    stroke               0
    dtype: int64
```


Handling Duplicate Values

There were no duplicate values in the categorical columns and even if there were for example male, female, it weren't exactly duplicate but only acted as labels

Train Test Split

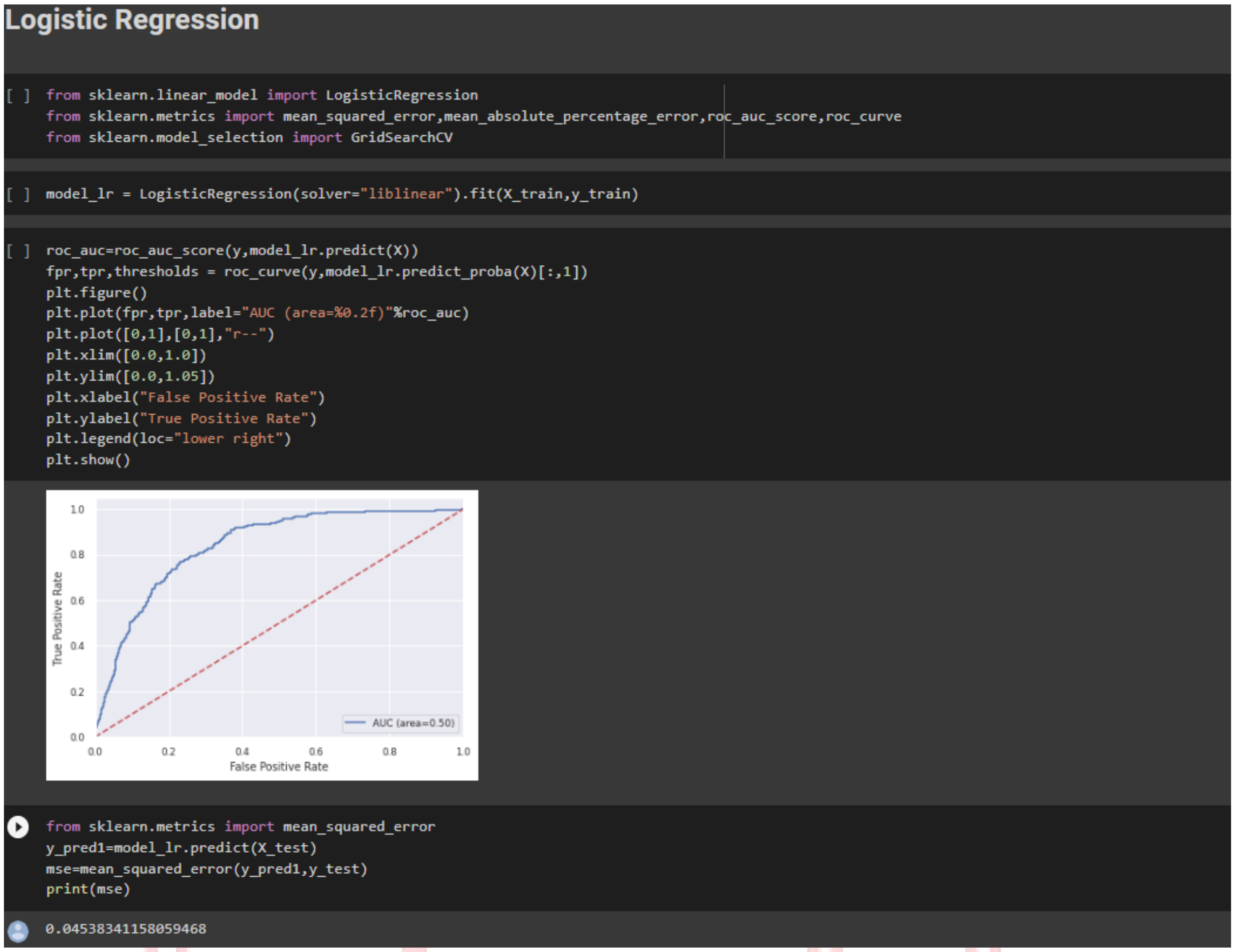
```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] X = df_imputed.iloc[:,:-1]
    y = df_imputed.iloc[:,-1]
    X = pd.get_dummies(X,drop_first=True)
```

```
[ ] X_train, X_test, y_train , y_test = train_test_split(X,y,random_state=123)
```

The Pareto principle (also known as the 80/20 rule) is a phenomenon that states that roughly 80% of outcomes come from 20% of causes.

We would have used the Pareto ratio, however this was a fairly large dataset and hence we decided not to.

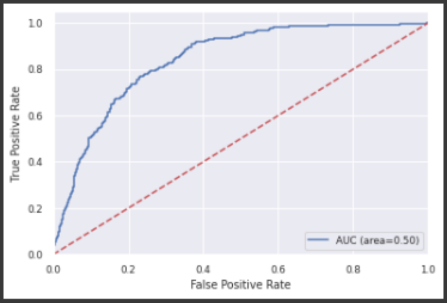


Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import mean_squared_error,mean_absolute_percentage_error,roc_auc_score,roc_curve
    from sklearn.model_selection import GridSearchCV
```

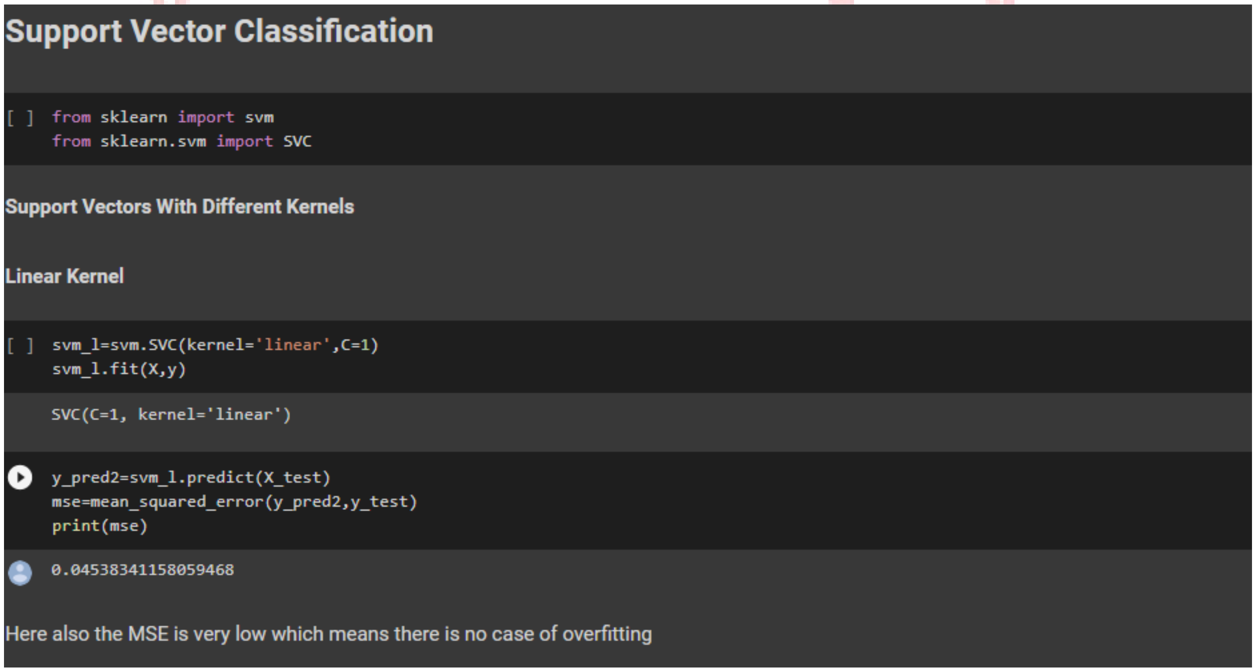
```
[ ] model_lr = LogisticRegression(solver="liblinear").fit(X_train,y_train)
```

```
[ ] roc_auc=roc_auc_score(y,model_lr.predict(X))
    fpr,tpr,thresholds = roc_curve(y,model_lr.predict_proba(X)[:,1])
    plt.figure()
    plt.plot(fpr,tpr,label="AUC (area=%0.2f)"%roc_auc)
    plt.plot([0,1],[0,1],"r--")
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend(loc="lower right")
    plt.show()
```



```
from sklearn.metrics import mean_squared_error
y_pred1=model_lr.predict(X_test)
mse=mean_squared_error(y_pred1,y_test)
print(mse)
```

```
0.04538341158059468
```



Support Vector Classification

```
[ ] from sklearn import svm
    from sklearn.svm import SVC
```

Support Vectors With Different Kernels

Linear Kernel

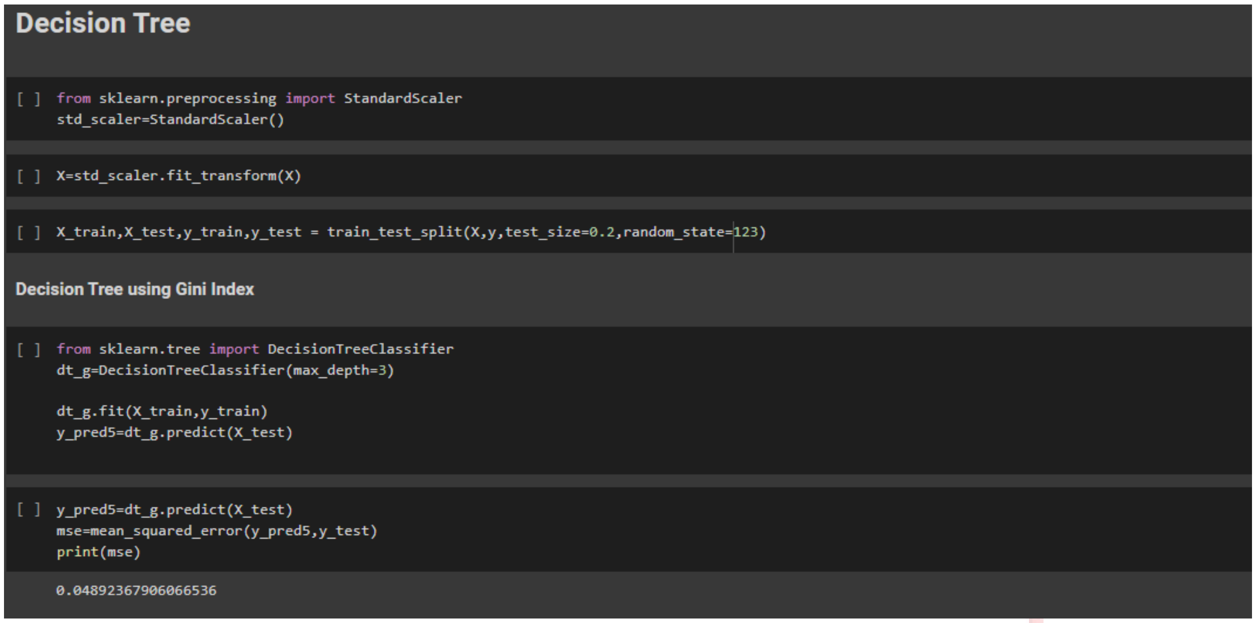
```
[ ] svm_l=svm.SVC(kernel='linear',C=1)
    svm_l.fit(X,y)
```

```
SVC(C=1, kernel='linear')
```

```
y_pred2=svm_l.predict(X_test)
mse=mean_squared_error(y_pred2,y_test)
print(mse)
```

```
0.04538341158059468
```

Here also the MSE is very low which means there is no case of overfitting



Decision Tree

```
[ ] from sklearn.preprocessing import StandardScaler
    std_scaler=StandardScaler()
```

```
[ ] X=std_scaler.fit_transform(X)
```

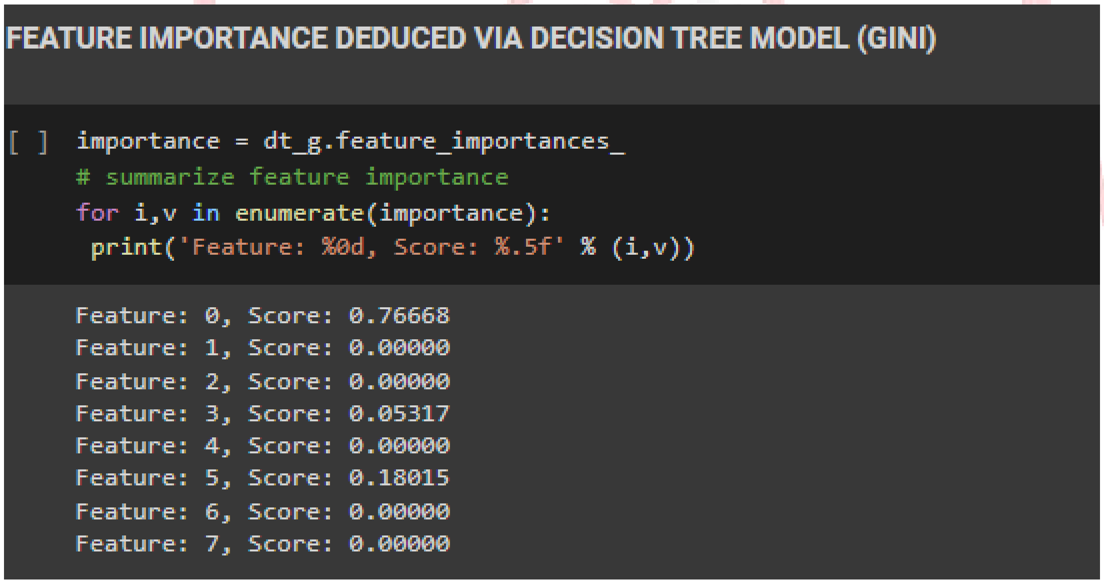
```
[ ] X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=123)
```

Decision Tree using Gini Index

```
[ ] from sklearn.tree import DecisionTreeClassifier
    dt_g=DecisionTreeClassifier(max_depth=3)

    dt_g.fit(X_train,y_train)
    y_pred5=dt_g.predict(X_test)
```

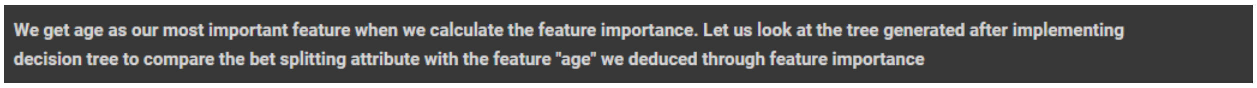
```
[ ] y_pred5=dt_g.predict(X_test)
    mse=mean_squared_error(y_pred5,y_test)
    print(mse)

    0.04892367906066536
```

FINDING FEATURE IMPORTANCE FROM THE DATASET:



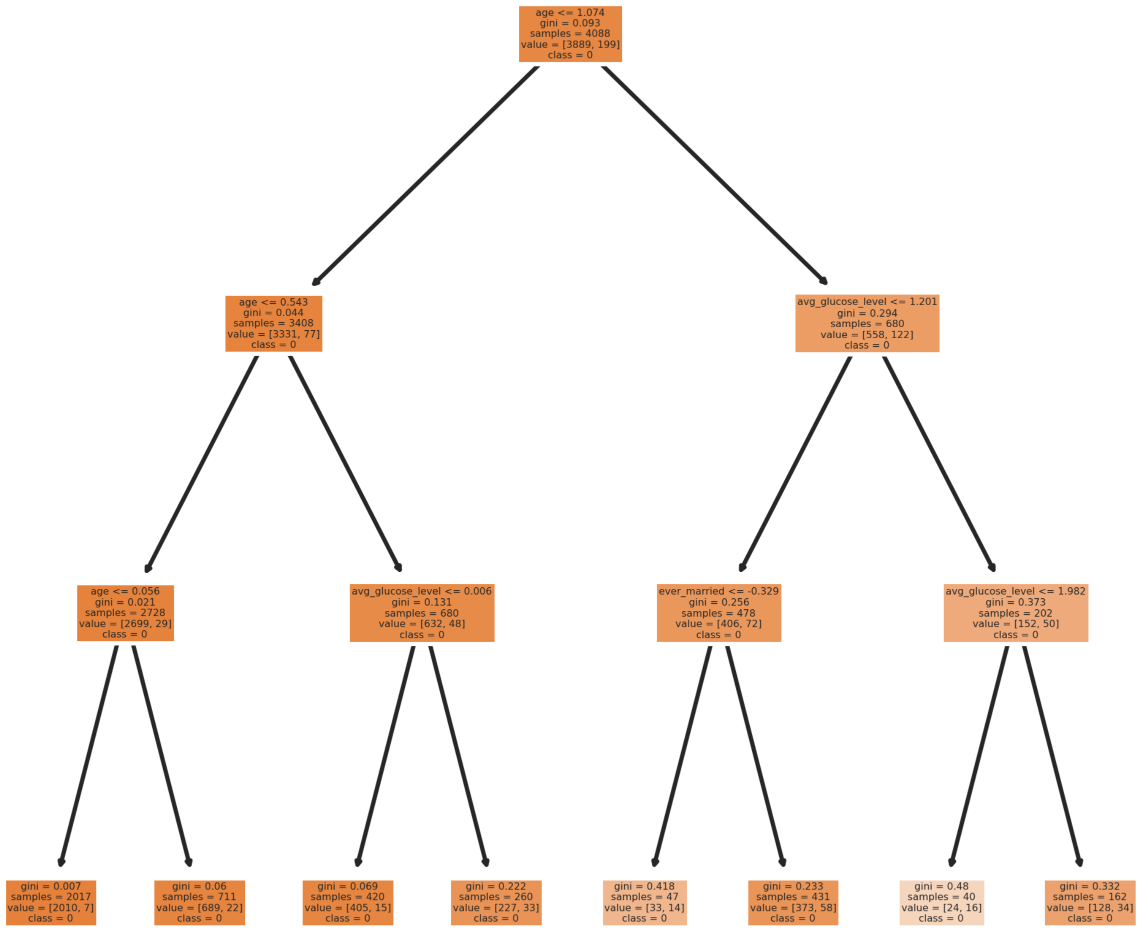
FEATURE IMPORTANCE DEDUCED VIA DECISION TREE MODEL (GINI)

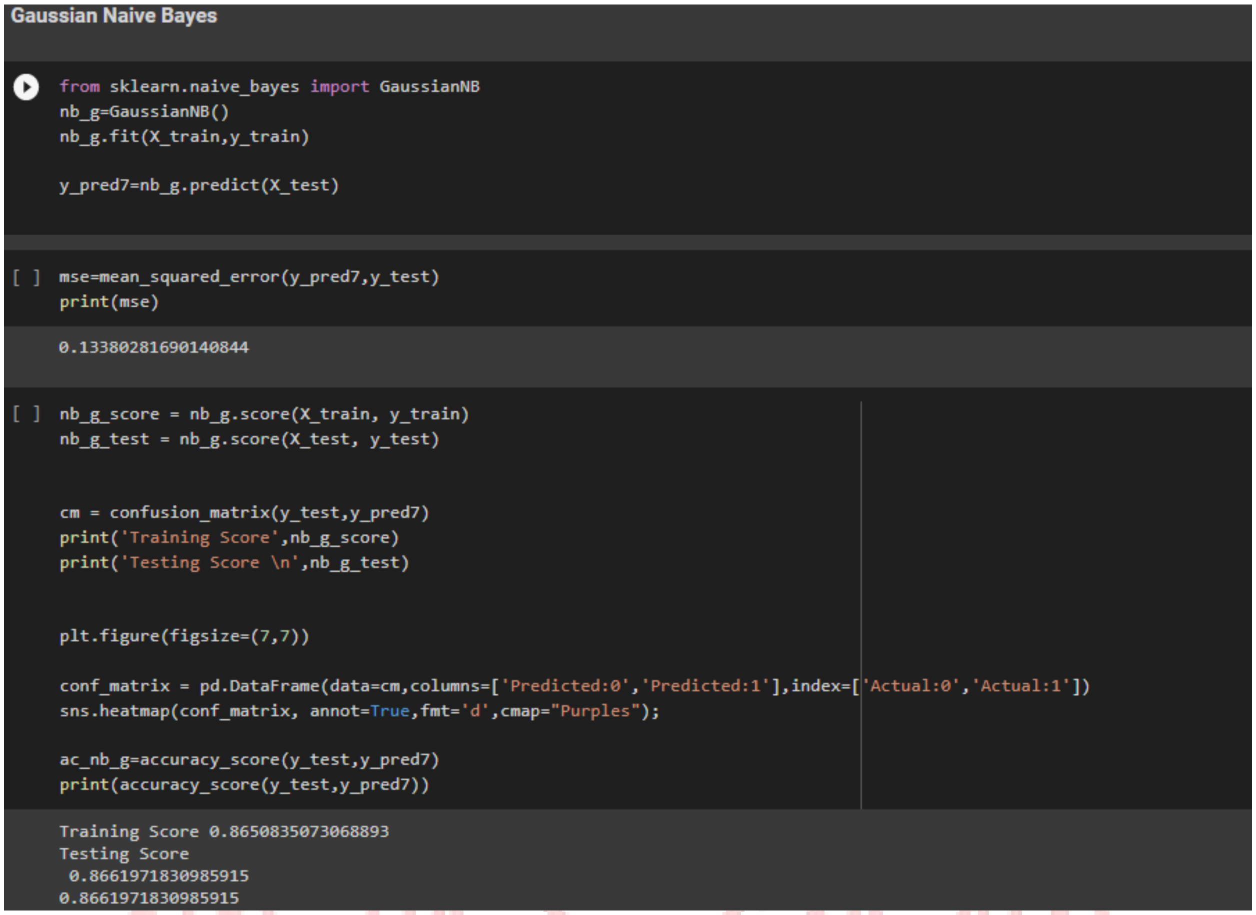
```
[ ] importance = dt_g.feature_importances_
    # summarize feature importance
    for i,v in enumerate(importance):
     print('Feature: %0d, Score: %.5f' % (i,v))

    Feature: 0, Score: 0.76668
    Feature: 1, Score: 0.00000
    Feature: 2, Score: 0.00000
    Feature: 3, Score: 0.05317
    Feature: 4, Score: 0.00000
    Feature: 5, Score: 0.18015
    Feature: 6, Score: 0.00000
    Feature: 7, Score: 0.00000
```



We get age as our most important feature when we calculate the feature importance. Let us look at the tree generated after implementing decision tree to compare the bet splitting attribute with the feature "age" we deduced through feature importance

DECISION TREE PLOTTED:




Gaussian Naive Bayes

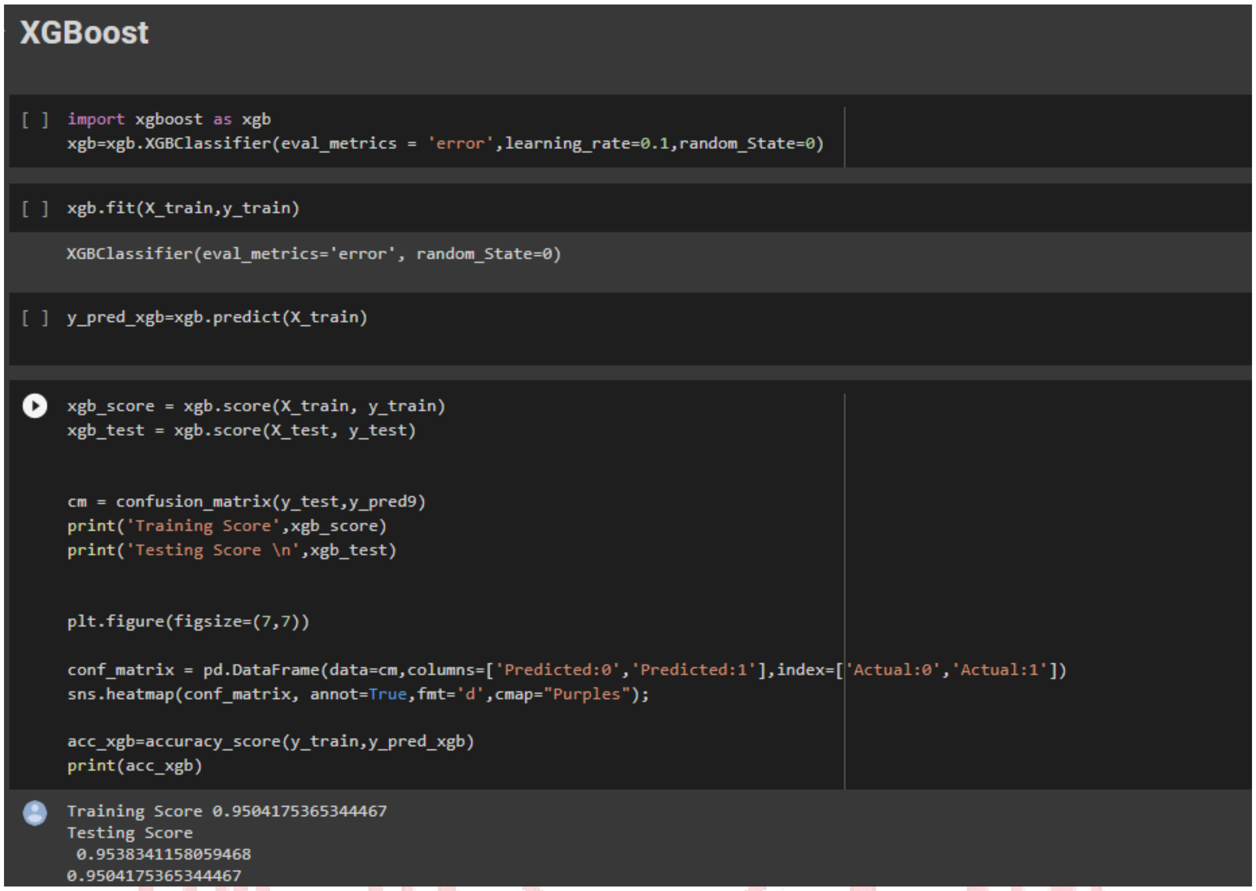
```
from sklearn.naive_bayes import GaussianNB
nb_g=GaussianNB()
nb_g.fit(X_train,y_train)

y_pred7=nb_g.predict(X_test)
```

```
[ ] mse=mean_squared_error(y_pred7,y_test)
    print(mse)
```

```
0.13380281690140844
```

```
[ ] nb_g_score = nb_g.score(X_train, y_train)
    nb_g_test = nb_g.score(X_test, y_test)


    cm = confusion_matrix(y_test,y_pred7)
    print('Training Score',nb_g_score)
    print('Testing Score \n',nb_g_test)


    plt.figure(figsize=(7,7))

    conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
    sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="Purples");

    ac_nb_g=accuracy_score(y_test,y_pred7)
    print(accuracy_score(y_test,y_pred7))
```

```
Training Score 0.8650835073068893
Testing Score 
 0.8661971830985915
0.8661971830985915
```



XGBoost

```
[ ] import xgboost as xgb
    xgb=xgb.XGBClassifier(eval_metrics = 'error',learning_rate=0.1,random_State=0)
```

```
[ ] xgb.fit(X_train,y_train)
```

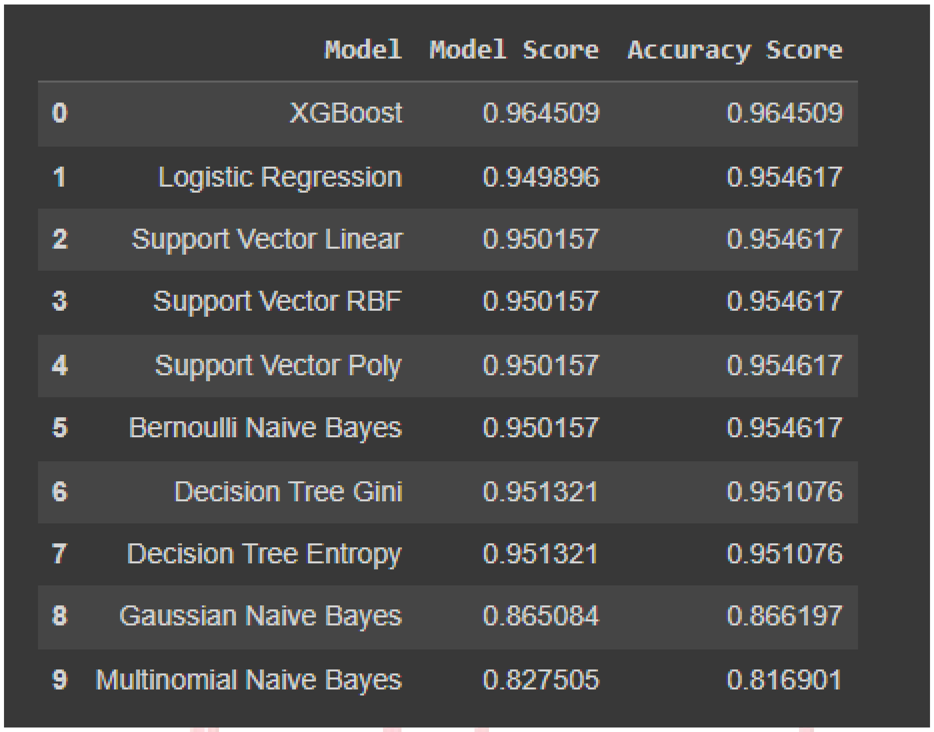
```
    XGBClassifier(eval_metrics='error', random_State=0)
```

```
[ ] y_pred_xgb=xgb.predict(X_train)
```

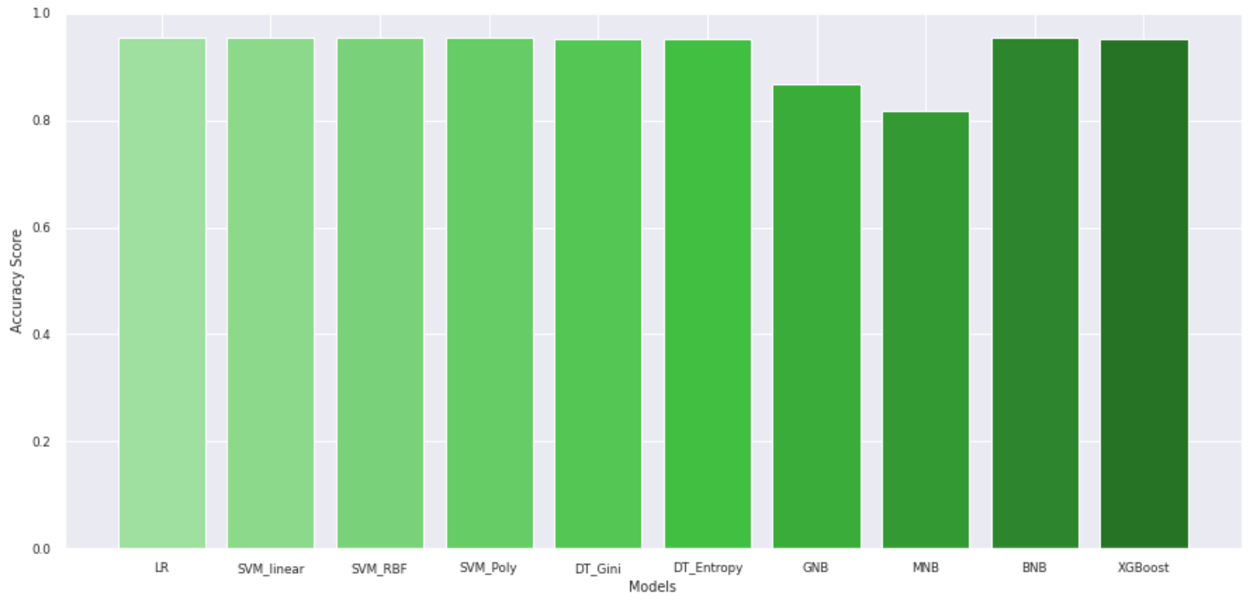
```
    xgb_score = xgb.score(X_train, y_train)
    xgb_test = xgb.score(X_test, y_test)


    cm = confusion_matrix(y_test,y_pred9)
    print('Training Score',xgb_score)
    print('Testing Score \n',xgb_test)


    plt.figure(figsize=(7,7))

    conf_matrix = pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
    sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="Purples");

    acc_xgb=accuracy_score(y_train,y_pred_xgb)
    print(acc_xgb)
```

```
Training Score 0.9504175365344467
Testing Score
 0.9538341158059468
0.9504175365344467
```

ACCURACY SCORE COMPARISON GRAPH:



	Model	Model Score	Accuracy Score
0	XGBoost	0.964509	0.964509
1	Logistic Regression	0.949896	0.954617
2	Support Vector Linear	0.950157	0.954617
3	Support Vector RBF	0.950157	0.954617
4	Support Vector Poly	0.950157	0.954617
5	Bernoulli Naive Bayes	0.950157	0.954617
6	Decision Tree Gini	0.951321	0.951076
7	Decision Tree Entropy	0.951321	0.951076
8	Gaussian Naive Bayes	0.865084	0.866197
9	Multinomial Naive Bayes	0.827505	0.816901



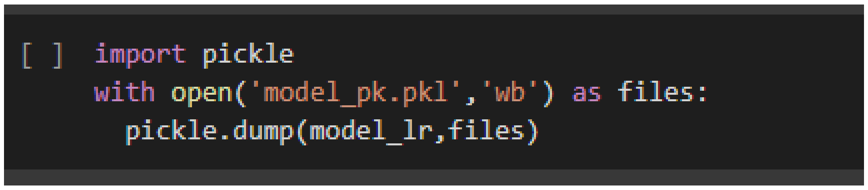
CONCLUSION:

We gain the best accuracy score from the LOGISTIC REGRESSION model. .

Thus we use it to build our prediction model.

1. First we pickle our model:

   What is a pickle?
   Pickle is a generic object serialization module that can be used for serializing and deserializing objects. While it’s most commonly associated with saving and reloading trained machine learning models, it can actually be used on any kind of object. Here’s how you can use Pickle to save a trained model to a file and reload it to obtain predictions.



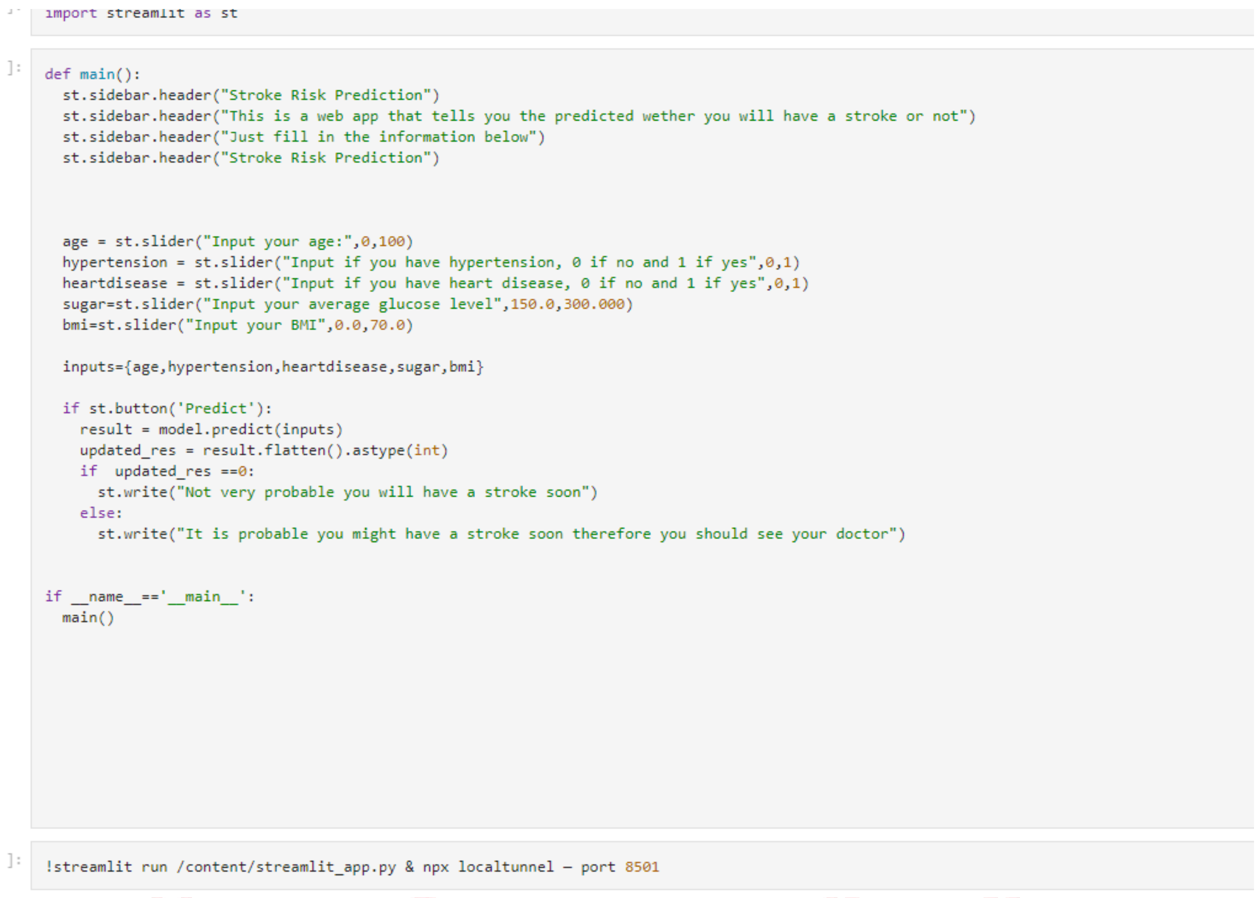
```
[ ]  import pickle
     with open('model_pk.pkl','wb') as files:
       pickle.dump(model_lr,files)
```

   Here, model_lr is our “logistic regression” model.

The raw pickle file:

```
€•      sklearn.linear_model._logistic”LogisticRegression”“”)”}”( penalty”l2”dual”‰tol”G?6âëC-C”G?ð      
fit_intercept”ˆintercept_scaling”K
class_weight”N
random_state”Nsolver”      liblinear”max_iter”Kd
multi_class”auto” verbose”K 
warm_start”‰n_jobs”Nl1_ratio”Nfeature_names_in_”numpy.core.multiarray”
_reconstruct”“”numpy” ndarray”“”K …”Cb”‡”R”(KK…”hdtype”“”O8”‰ˆ‡”R”(K|”NNNJÿÿÿÿJÿÿÿÿK?t”b‰]”(age”
hypertension”
heart_disease”
ever_married”    work_type”avg_glucose_level”bmi”smoking_status”et”bn_features_in_”Kclasses_”hhK …”h‡”R”(KK…”h$f8”‰ˆ‡”R”(K<”NNNJÿÿÿÿJÿÿÿÿK t”b‰C
v_Ù?-ÑW' tÓ?ØdÓ{ƒÈx¿ƒo*/ÅbÀ¿3ÞK85r?
`ÔJ^¿eŸü‹¥œ¿”t”b
intercept_”hhK …”h‡”R”(KK…”h<‰C=ö‡

™À”t”b n_iter_”hhK …”h‡”R”(KK…”h$i4”‰ˆ‡”R”(Kh=NNNJÿÿÿÿJÿÿÿÿK t”b‰C   ”t”b_sklearn_version”1.0.2”ub.
```

2. We use streamlit to create a UI for the model we built:

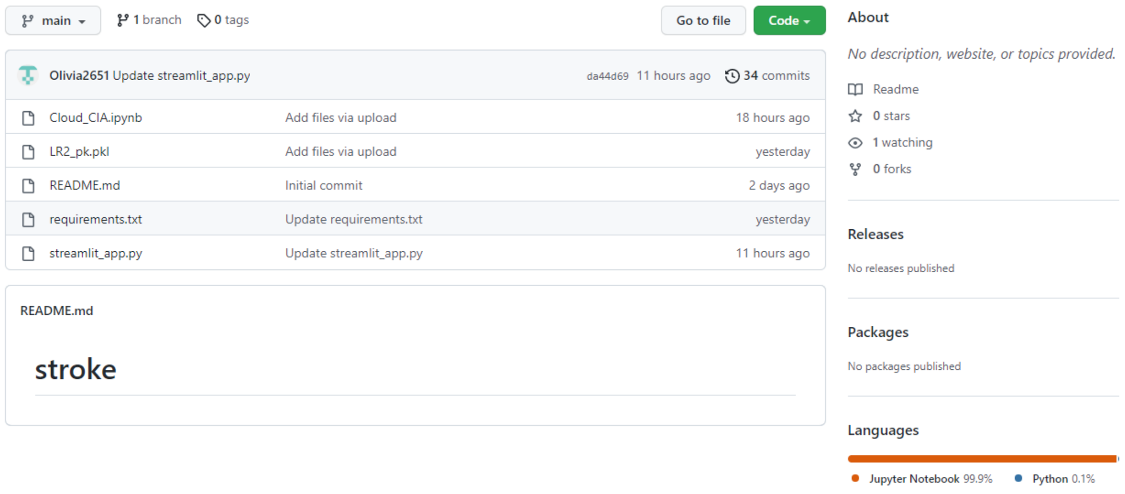


```
import streamlit as st
```

```
def main():
  st.sidebar.header("Stroke Risk Prediction")
  st.sidebar.header("This is a web app that tells you the predicted wether you will have a stroke or not")
  st.sidebar.header("Just fill in the information below")
  st.sidebar.header("Stroke Risk Prediction")



  age = st.slider("Input your age:",0,100)
  hypertension = st.slider("Input if you have hypertension, 0 if no and 1 if yes",0,1)
  heartdisease = st.slider("Input if you have heart disease, 0 if no and 1 if yes",0,1)
  sugar=st.slider("Input your average glucose level",150.0,300.000)
  bmi=st.slider("Input your BMI",0.0,70.0)

  inputs={age,hypertension,heartdisease,sugar,bmi}

  if st.button('Predict'):
    result = model.predict(inputs)
    updated_res = result.flatten().astype(int)
    if  updated_res ==0:
      st.write("Not very probable you will have a stroke soon")
    else:
      st.write("It is probable you might have a stroke soon therefore you should see your doctor")


if __name__=='__main__':
  main()
```

```
!streamlit run /content/streamlit_app.py & npx localtunnel — port 8501
```

3. We created a Git repository with all the necessary files:



main 1 branch 0 tags Go to file Code

Olivia2651 Update streamlit_app.py da44d69 11 hours ago 34 commits

Cloud_CIA.ipynb	Add files via upload	18 hours ago
LR2_pk.pkl	Add files via upload	yesterday
README.md	Initial commit	2 days ago
requirements.txt	Update requirements.txt	yesterday
streamlit_app.py	Update streamlit_app.py	11 hours ago

README.md

stroke

About

No description, website, or topics provided.

Readme
0 stars
1 watching
0 forks

Releases

No releases published

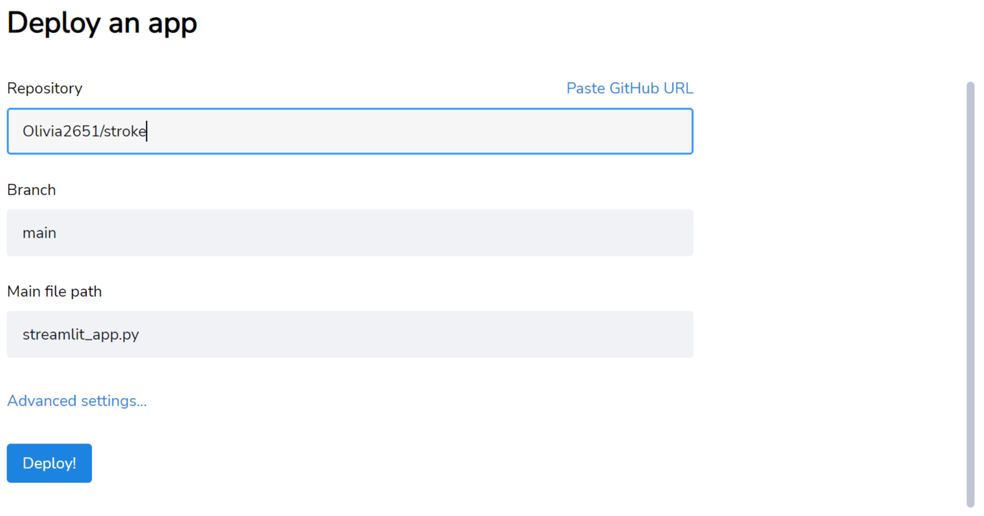
Packages

No packages published

Languages

Jupyter Notebook 99.9% Python 0.1%

4. We then deployed our app on Streamlit using the repository:



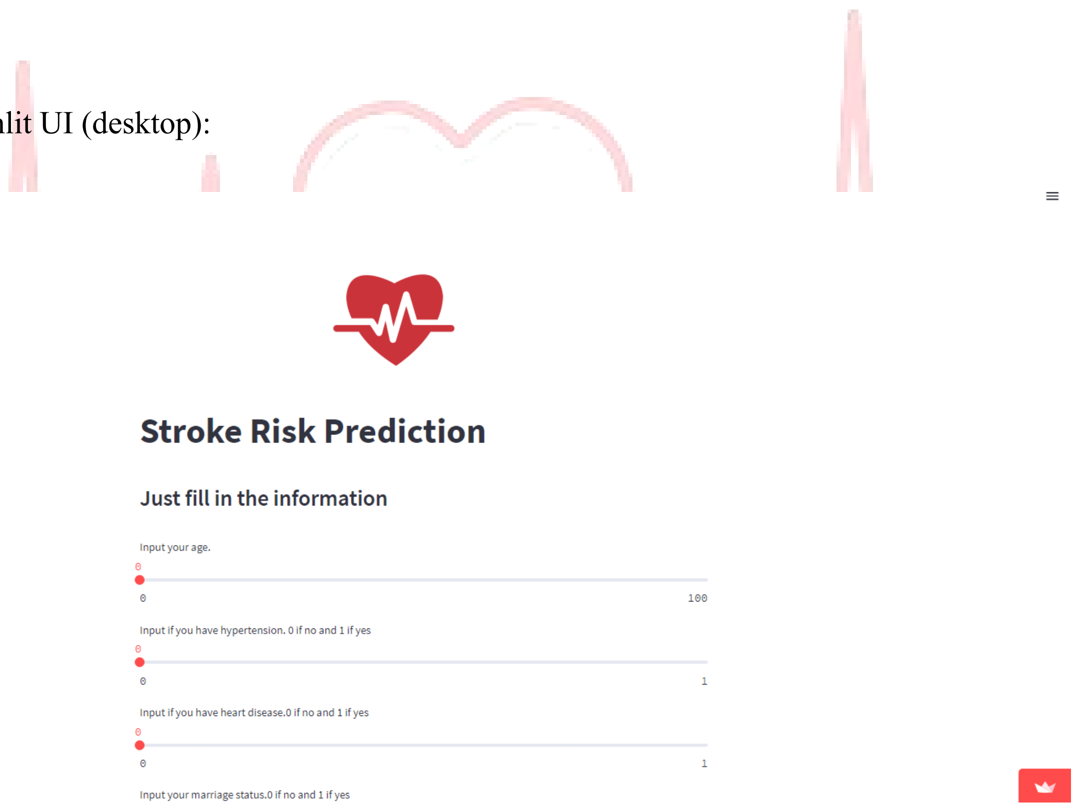
Deploy an app

Repository: Olivia2651/stroke

Branch: main

Main file path: streamlit_app.py

5. Streamlit UI (desktop):



Stroke Risk Prediction

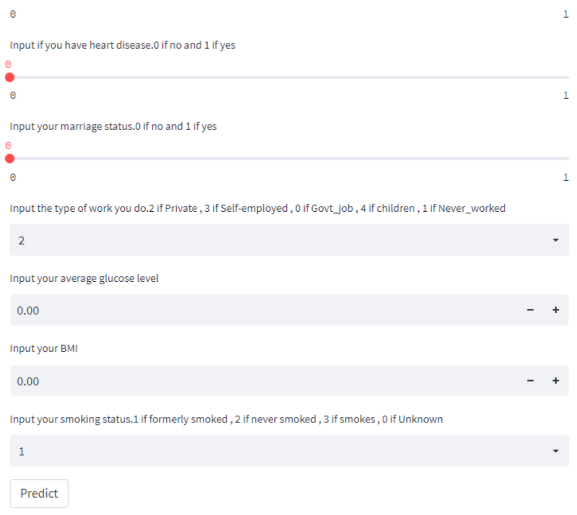
Just fill in the information

Input your age.
0
0 100

Input if you have hypertension. 0 if no and 1 if yes
0
0 1

Input if you have heart disease.0 if no and 1 if yes
0
0 1

Input your marriage status.0 if no and 1 if yes



0 1

Input if you have heart disease.0 if no and 1 if yes

0

0 1

Input your marriage status.0 if no and 1 if yes

0

0 1

Input the type of work you do.2 if Private , 3 if Self-employed , 0 if Govt_job , 4 if children , 1 if Never_worked

2

Input your average glucose level

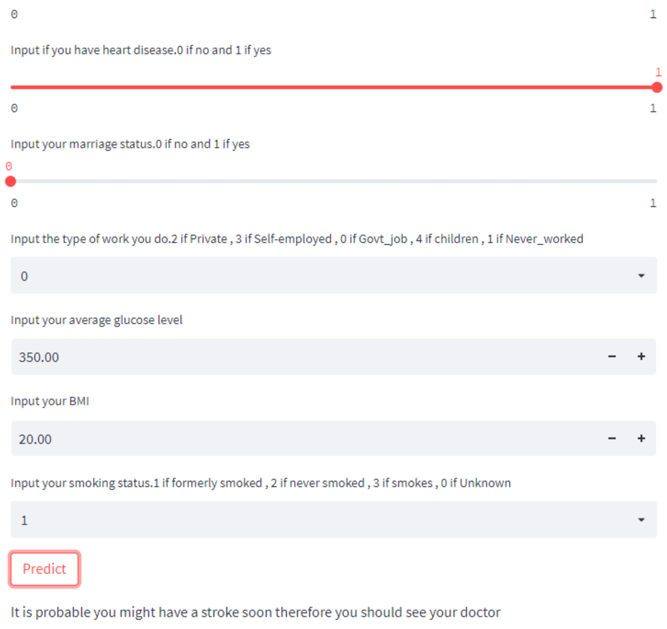
0.00

Input your BMI

0.00

Input your smoking status.1 if formerly smoked , 2 if never smoked , 3 if smokes , 0 if Unknown

1



0 1

Input if you have heart disease.0 if no and 1 if yes

1

0 1

Input your marriage status.0 if no and 1 if yes

0

0 1

Input the type of work you do.2 if Private , 3 if Self-employed , 0 if Govt_job , 4 if children , 1 if Never_worked

0

Input your average glucose level

350.00

Input your BMI

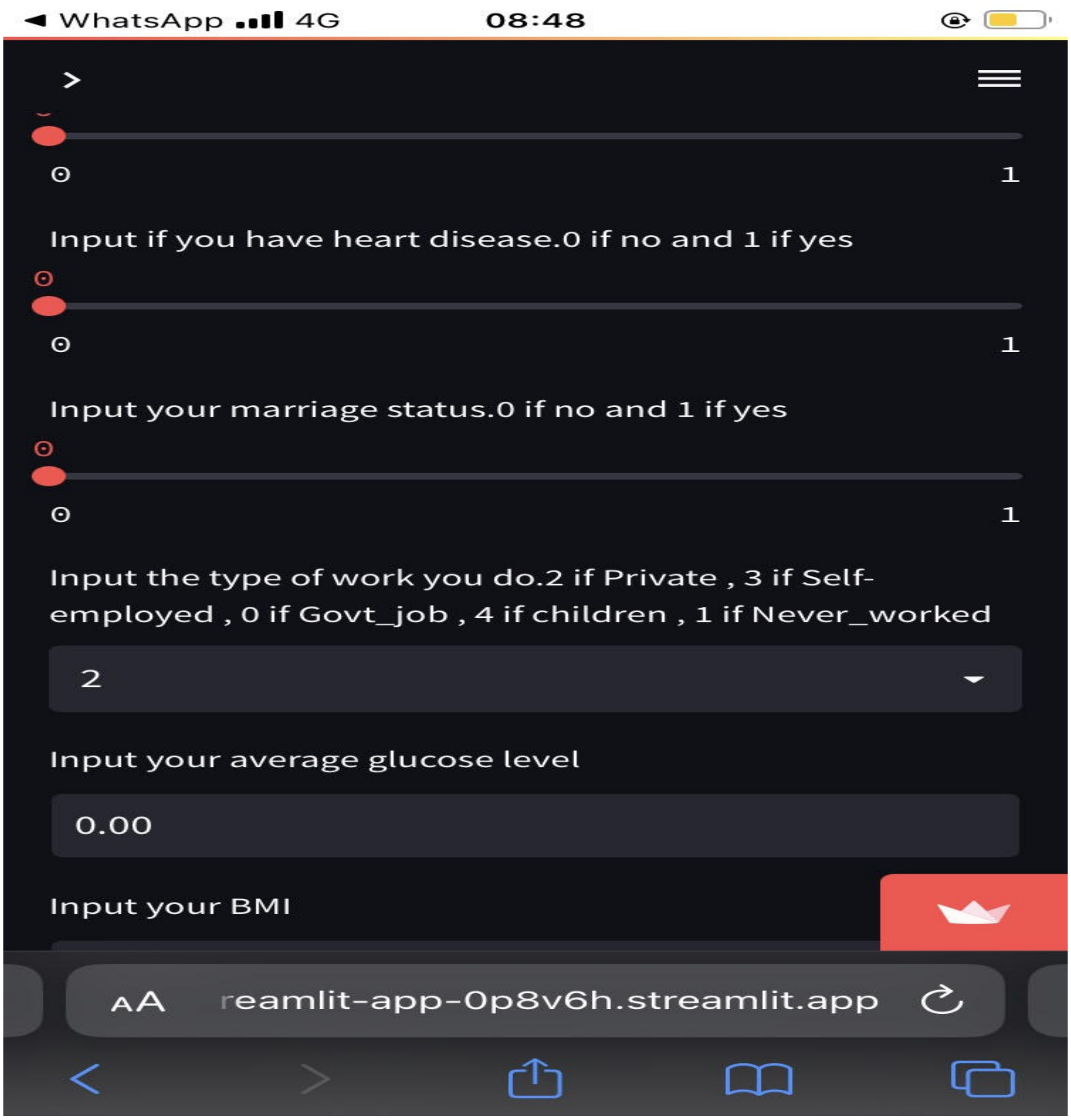
20.00

Input your smoking status.1 if formerly smoked , 2 if never smoked , 3 if smokes , 0 if Unknown

1

It is probable you might have a stroke soon therefore you should see your doctor

6. Streamlit UI (phone):


WhatsApp 4G 08:48

0 1

Input if you have heart disease.0 if no and 1 if yes

0

0 1

Input your marriage status.0 if no and 1 if yes

0

0 1

Input the type of work you do.2 if Private , 3 if Self-employed , 0 if Govt_job , 4 if children , 1 if Never_worked

2

Input your average glucose level

0.00

Input your BMI

AA reamlit-app-0p8v6h.streamlit.app