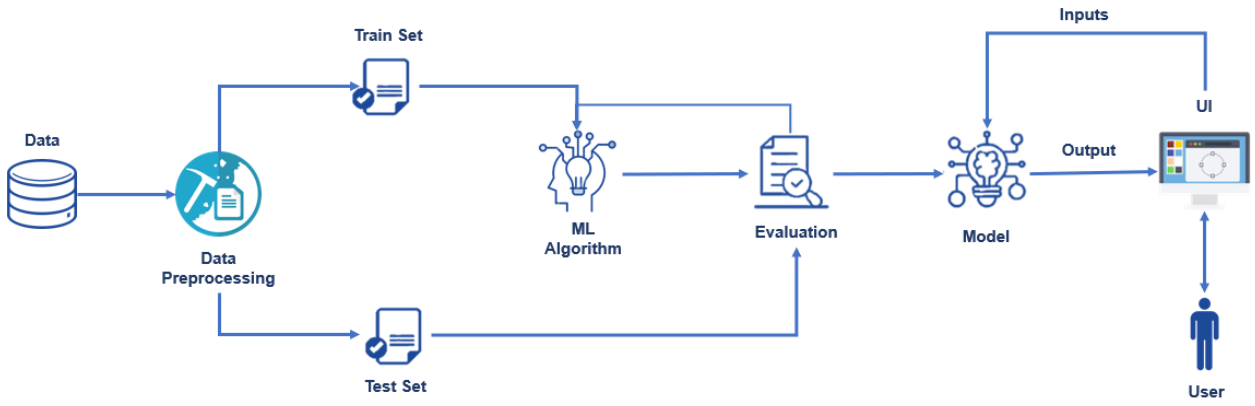
Heart Stroke Prediction Using IBM Watson Machine Learning

Project Description:

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. In recent times, Heart Stroke prediction is one of the most complicated tasks in the medical field. In the modern era, approximately one person dies per minute due to heart Stroke. Data science plays a crucial role in processing huge amount of data in the field of healthcare. As heart stroke prediction is a complex task, there is a need to automate the prediction process to avoid risks associated with it and alert the patient well in advance.

The project aims to predict the chances of Heart Stroke and classifies the patient's risk level by implementing different Machine Learning techniques such as KNN, Decision Tree and Random Forest. From these models the Best performing model is selected and saved. Here we will be building a flask application that uses a machine learning model to get the prediction of heart stroke. We will also train our model on IBM Cloud and deployment on IBM Cloud

Technical Architecture:



Prerequisites:

To complete this project, you must require following software's concepts and packages

- **Anaconda navigator:**
  - Refer to the video below to know more about how download anaconda navigator

- **Python packages:**

  Open anaconda prompt as administrator

- Type **"pip install numpy"** and click enter.
- Type **"pip install pandas"** and click enter.
- Type **"pip install scikit-learn"** and click enter.
- Type" **pip install matplotlib"** and click enter.
- Type "**pip install pickle-mixin"** and click enter.
- Type **"pip install seaborn"** and click enter.
- Type **"pip install imblearn"** and click enter.
- Type **"pip install pandas-profiling"** and click enter.
- Type **"pip install Flask"** and click enter.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **ML Concepts**
  - **Supervised learning**
  - **Unsupervised learning**
  - **Regression**
  - **Linear Regression**
  - **Randomforest**
  - **SVR**
  - **Stochastic Gradient Descent Classifier**
  - **Xgboost**
  - **Evaluation metrics**

- **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

  Link: **https://www.youtube.com/watch?v=lj4I_CvBnt0**

Project Objectives:

By the end of this project you'll understand:

- You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
- You will be able to know how to pre-process/clean the data using different data pre-processing techniques.
- How to perform oversampling when the dataset is imbalanced.
- Applying different algorithms according to the dataset
- You will be able to know how to find the accuracy of the model.
- You will be able to build web applications using the Flask framework.

Project Flow:

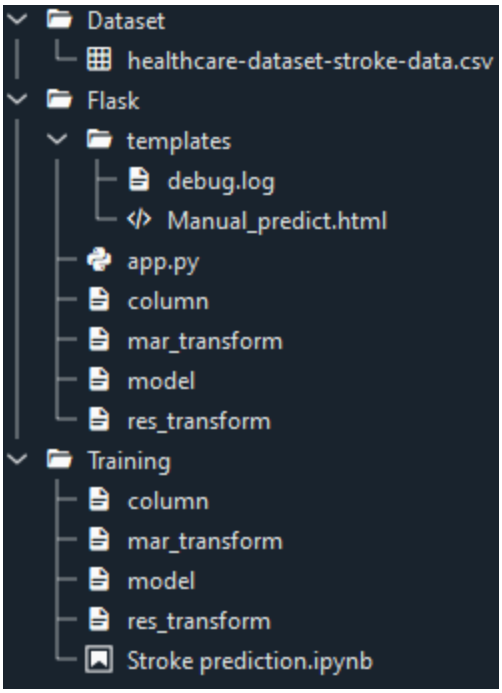
- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
  - Collect the dataset or Create the dataset
- Visualizing and analyzing data
  - Univariate analysis
  - Bivariate analysis
  - Multivariate analysis
  - Descriptive analysis
- Data Pre-processing.
  - Drop unwanted features
  - Checking for null values
  - Handling categorical data
  - Splitting data into train and test
  - Feature Scaling
- Model Building
  - Import the model building Libraries
  - Initializing the model
  - Training and testing the model
  - Evaluation of Model
  - Save the Model
- Application Building
  - Create an HTML file
  - Build a Python Code

Project Structure:

Create a Project folder which contains files as shown below



- Training folder contains:
  - Stroke Prediction.ipynb is the jupyter notebook file where the model is built.
  - model is the model file that generates when the notebook file is executed.
  - Column,mar_transform,res_transform are the transformation and encoding files that were generated when u run the main program.
- Flask folder is the application folder where the web application and server-side program are present.
- Dataset folder contains the dataset file healthcare-dataset-stroke-data.csv

Milestone 1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

You can download the dataset used in this project using the below link.

Dataset: - https://www.kaggle.com/datasets/ujjwalchowdhury/heart-stroke prediction

Milestone 2: Visualizing And Analyzing The Data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods.

Note: There are n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

ACTIVITY :1

Importing The Libraries

Import the necessary libraries as shown in the image.

To know about the packages, refer the link given in pre requisites.

```
In [1]: import warnings
        warnings.filterwarnings("always")
        warnings.filterwarnings("ignore")
```

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from matplotlib import style
        import seaborn as sns
        import pandas_profiling as pdp
```

```
In [3]: %matplotlib inline
        style.use('fivethirtyeight')
        sns.set(style="whitegrid",color_codes=True)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC,SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier
from sklearn.tree import  DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb
import lightgbm as lgb
import catboost as cb
from sklearn.model_selection import train_test_split,cross_validate
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OrdinalEncoder
```

ACTIVITY:2

Read The Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.

```
df=pd.read_csv("healthcare-dataset-stroke-data.csv")
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

Let’s know more about our data

```
df.shape
```

```
(5110, 12)
```

```
df.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   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

- There are total of 5110 records in the dataset with a total of 12 features.
  Attribute Information
  1. id: a 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 the 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
- *Note: "Unknown" in smoking_status means that the information is unavailable for this patient

Activity 3:

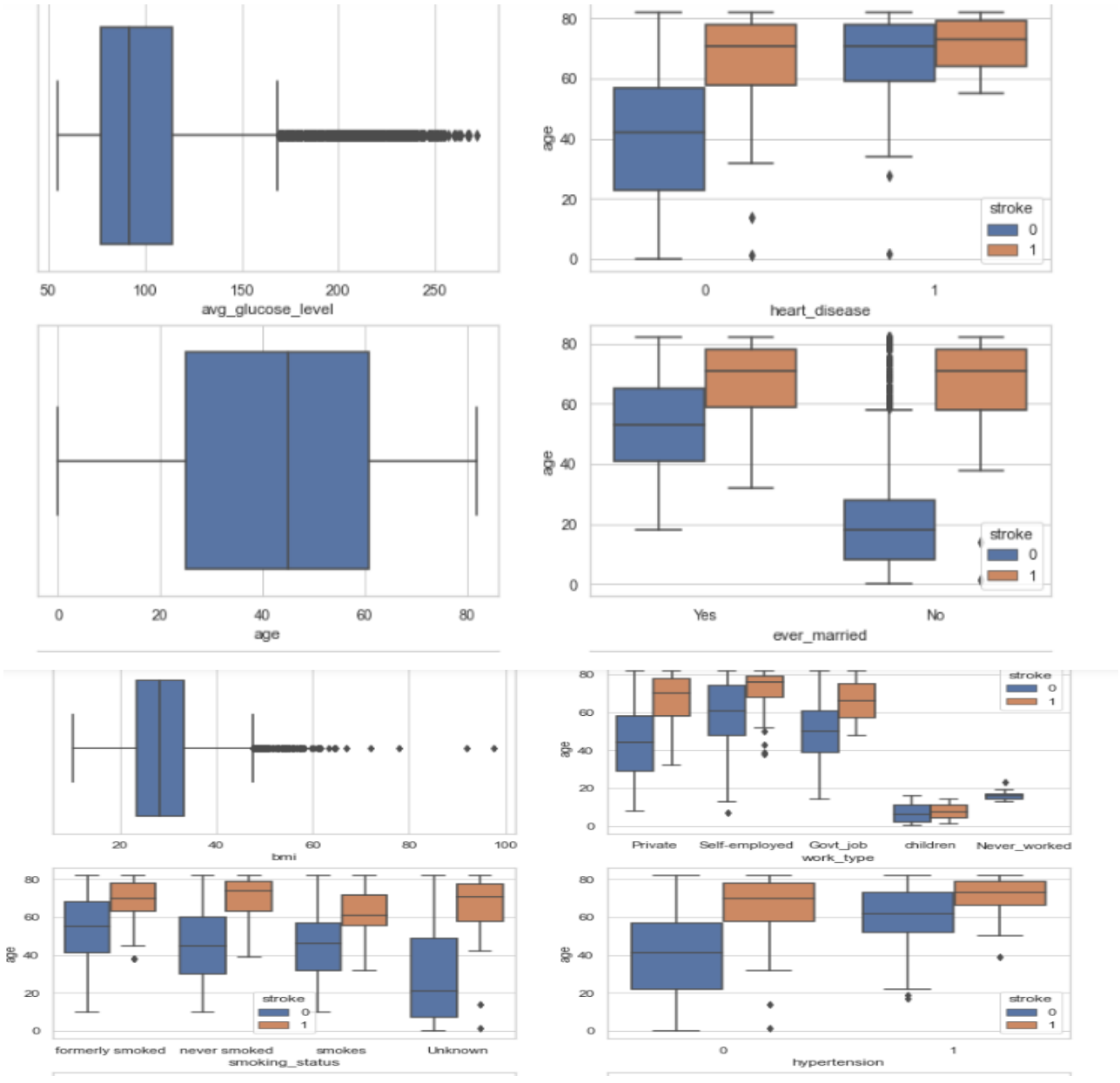
Univariate Analysis

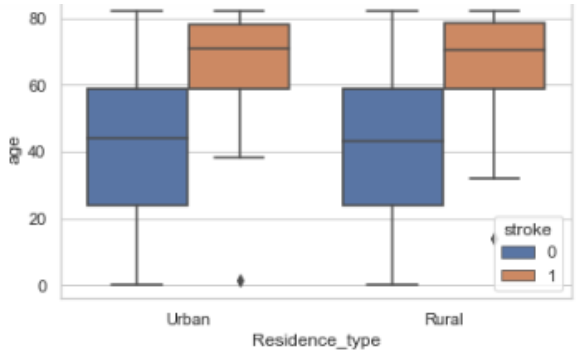
In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as pie plot, box plot and count plot.

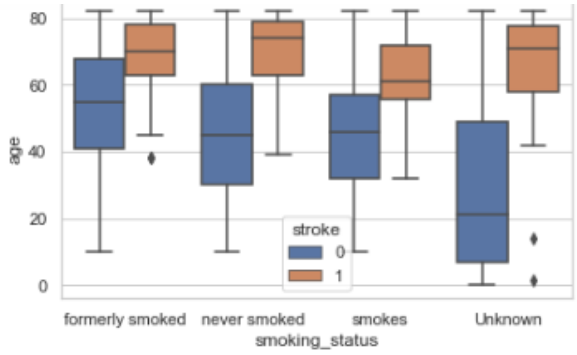
- Boxplot() is used to represent the variation in values of different features.
- This visualization helps us to identifying outliers using boxplot

```
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(12,20))
sns.boxplot(x = 'avg_glucose_level', data=df, ax=axes[0][0])
sns.boxplot(x='age', data=df,ax=axes[1][0])
sns.boxplot(x='bmi', data=df, ax=axes[2][0])
sns.boxplot(x='smoking_status', y='age' ,hue='stroke', data=df, ax=axes[3][0])
sns.boxplot(x='hypertension', y='age' ,hue='stroke', data=df, ax=axes[3][1])
sns.boxplot(x='heart_disease', y='age' ,hue='stroke', data=df, ax=axes[0][1])
sns.boxplot(x='ever_married', y='age' ,hue='stroke', data=df, ax=axes[1][1])
sns.boxplot(x='work_type', y='age' ,hue='stroke', data=df, ax=axes[2][1])
sns.boxplot(x='Residence_type', y='age' ,hue='stroke', data=df, ax=axes[4][0])
sns.boxplot(x='smoking_status', y='age' ,hue='stroke', data=df, ax=axes[4][1])
```

```
<AxesSubplot:xlabel='smoking_status', ylabel='age'>
```







- The following visualizations represents that We have outliers in avg_glucose_level and BMI. These outliers should be removed

ACTIVITY:4

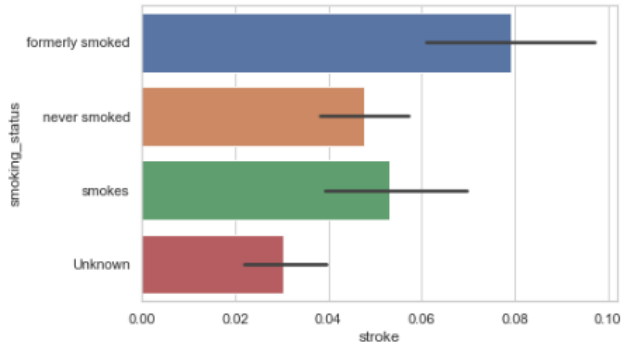
Bivariate Analysis

To find the relation between two features we use bivariate analysis.

- Boxplot: The following visualization represents the relationship between smoking status and the target variable

```
sns.barplot(x= 'stroke', y= 'smoking_status',data = df)
```

```
<AxesSubplot:xlabel='stroke', ylabel='smoking_status'>
```



- Unknown' and 'never smoker' has a low percentage of strokes in the sample. Let's combine them into one group: 'never smoker'.

```
replace_values = {'unknown' : 'never smoked' }
df=df.replace({'smoking_status ': replace_values})
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

ACTIVITY:5

Multivariate Analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used a heatmap from the seaborn package.

- Correlation is a statistical measure that expresses the extent to which two variables are linearly related. It's a common tool for describing simple relationships without making a statement about cause and effect.
- To visualize the correlation heat map() function is used. From the below image we can easily find the highly correlated feature.

```
plt.figure(figsize = (18,9))
sns.heatmap(df.corr(),annot = True)
plt.show()
```

ACTIVITY:6

Descriptive Analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
df.describe()
```

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

Milestone 3: Data Pre-Processing

In this milestone, we will be preprocessing the dataset that is collected. Preprocessing includes:

- Drop unwanted features
- Handling the null values.
- Removing Outliers
- Handling the categorical values if any.
- Oversampling to balance the data.
- Identify the dependent and independent variables.
- Split the dataset into train and test sets.

ACTIVITY:1

Drop Unwanted Features

- drop() is used to drop specified labels from rows or columns.

  Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names.

- Here, our target variable is not dependent on the id feature so, we can remove the id column.

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

```
(5110, 11)
```

ACTIVITY:2

Handling For Null Values

Here we check the presence of Null values in the dataset and dropping the null values.

- For checking the null values, df.isnull() function is used.
- To sum those null values we use .sum() function to it. From the below image we found that education column and previous year rating column has null values.

```
df.isnull().sum()
```

```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
dtype: int64
```

- As we can see there are 201 null values in BMI column so we can remove this null values.
- Dropna() is used to drop the null values from the dataframe.

```
df.dropna(inplace=True)
df.isnull().sum()
```

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

ACTIVITY:3

Handling Outliers

With the help of boxplot, outliers are visualized (refer activity 3 univariate analysis). And here we are going to find upper bound and lower bound of Na_to_K feature with some mathematical formula.

- A function remove_outliers is used find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.
- If outliers are removed, we lose more data. It will impact model performance.
- Here removing outliers is impossible. So, the capping technique is used on outliers.
- Capping: Replacing the outliers with upper bound values.

```
def  remove_outliers(data):
       arr=[]
       q1=np.percentile(data,25)
       q3=np.percentile(data,75)
       iqr=q3-q1
       mi=q1-(1.5*iqr)
       ma=q3+(1.5*iqr)
       for i in list(data):
           if i<mi:
               i=mi
               arr.append(i)
           elif i>ma:
               i=ma
               arr.append(i)
           else:
               arr.append(i)
       return arr
```

- Calling the remove_outliers() for bmi and and avg_glucose_level

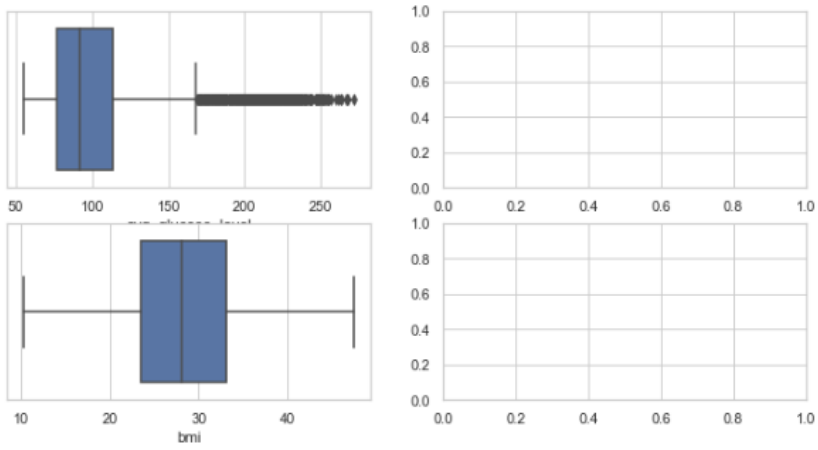
```
df['bmi'] = remove_outliers(df['bmi'])
df['average_glucose_level'] = remove_outliers(df['avg_glucose_level'])
print('Outliers  successfully removed')
```

```
Outliers  successfully removed
```

- Checking whether the outliers are removed or not and dropping ID column from the dataset

```
fig, axes= plt.subplots(nrows = 2, ncols = 2, figsize = (10, 5))
sns.boxplot(x = 'bmi', data = df, ax=axes[1][0])
sns.boxplot(x = 'avg_glucose_level', data = df, ax=axes[0][0])
```

```
<AxesSubplot:xlabel='avg_glucose_level'>
```



ACTIVITY:4

Handling Categorical Values

In machine learning, we usually deal with datasets that contain multiple labels in one or more than one column. These labels can be in the form of words or numbers. To make the data understandable or in human-readable form, the training data is often labelled in words.

- Let’s see the number of unique categories in categorical columns

```
for i in ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']:
    print(df[i].unique())
```

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

- Label Encoding on Categorical Variables

Label Encoding refers to converting the labels into the numeric form so as to convert them into the machine-readable form.

```
from sklearn.preprocessing import LabelEncoder
le1 = LabelEncoder()
le2 = LabelEncoder()
df['Residence_type'] = le1.fit_transform(df['Residence_type'])
df['ever_married'] = le2.fit_transform(df['ever_married'])
```

- Saving the encoding

Here joblib() is used to save the encoding

```
import joblib
joblib.dump(le1,"res_transform")
joblib.dump(le2,"mar_transform")
```

```
['mar_transform']
```

```
df.shape
```

```
(4909, 12)
```

```
df.iloc[0,:]
```

```
gender                          Male
age                             67.0
hypertension                       0
heart_disease                      1
ever_married                       1
work_type                    Private
Residence_type                     1
avg_glucose_level             228.69
bmi                             36.6
smoking_status          formerly smoked
stroke                             1
average_glucose_level         168.32
Name: 0, dtype: object
```

```
df.head()
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	average_glucose_level
0	Male	67.0	0	1	1	Private	1	228.69	36.6	formerly smoked	1	168.32
2	Male	80.0	0	1	1	Private	0	105.92	32.5	never smoked	1	105.92
3	Female	49.0	0	0	1	Private	1	171.23	34.4	smokes	1	168.32
4	Female	79.0	1	0	1	Self-employed	0	174.12	24.0	never smoked	1	168.32
5	Male	81.0	0	0	1	Private	1	186.21	29.0	formerly smoked	1	168.32

ACTIVITY:5

Handling Numerically Converted Categorical Values

Column Transform applies transformers to columns of an array or pandas DataFrame.Encode categorical features as a one-hot numeric array.

- The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array (depending on the sparse parameter)

```
x = df.iloc[:,0:10].values
y = df.iloc[:,10].values
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct= ColumnTransformer([("onehot",OneHotEncoder(drop='first'),[0,5,9])],remainder="passthrough")
X=ct.fit_transform(x)
```

Save the transform

```
joblib.dump(ct,"column")
```

```
['column']
```

```
X
```

```
array([[1.0, 0.0, 0.0, ..., 1, 228.69, 36.6],
       [1.0, 0.0, 0.0, ..., 0, 105.92, 32.5],
       [0.0, 0.0, 0.0, ..., 1, 171.23, 34.4],
       ...,
       [0.0, 0.0, 0.0, ..., 0, 82.99, 30.6],
       [1.0, 0.0, 0.0, ..., 0, 166.29, 25.6],
       [0.0, 0.0, 0.0, ..., 1, 85.28, 26.2]], dtype=object)
```

```
X[0]
```

```
array([1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 67.0, 0, 1, 1, 1,
       228.69, 36.6], dtype=object)
```

ACTIVITY:6

Oversampling

Imbalanced datasets are those where there is a severe skew in the class distribution, such as 1:100 or 1:1000 examples in the minority class to the majority class.

- This bias in the training dataset can influence many machine learning algorithms, leading some to ignore the minority class entirely. This is a problem as it is typically the minority class on which predictions are most important.
- One approach to addressing the problem of class imbalance is to randomly resample the training dataset. The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called undersampling, and to duplicate examples from the minority class, called oversampling.

```
X.shape
```

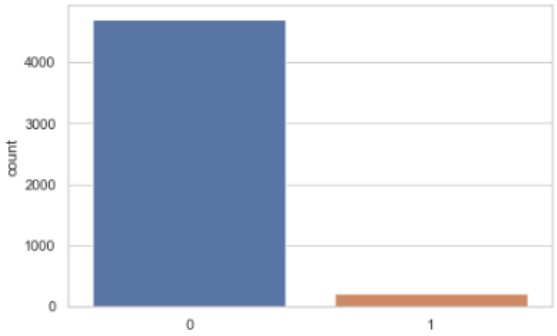
```
(4909, 16)
```

```
y.shape
```

```
(4909,)
```

```
sns.countplot(y)
```

```
<AxesSubplot:ylabel='count'>
```




```
from imblearn.over_sampling import SMOTE
sm = SMOTE()
X_res, y_res = sm.fit_resample(X, y)
print("Before OverSampling, counts of label '1': {}".format(sum(y==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y==0)))
print('After OverSampling, the shape of train_X: {}'.format(X_res.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_res.shape))
print("After OverSampling, counts of label '1': {}".format(sum(y_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_res==0)))
```

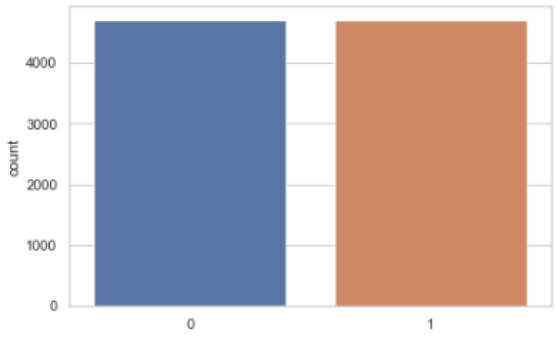
```
Before OverSampling, counts of label '1': 209
Before OverSampling, counts of label '0': 4700

After OverSampling, the shape of train_X: (9400, 16)
After OverSampling, the shape of train_y: (9400,)

After OverSampling, counts of label '1': 4700
After OverSampling, counts of label '0': 4700
```

```
sns.countplot(y_res)
```

```
<AxesSubplot:ylabel='count'>
```



ACTIVITY:6

Splitting Data Into Train And Test

Here we split the dataset into train and test data so that we can use train data to build the model.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size = 0.2, random_state = 42)
```

Milestone 4:

Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying the classification algorithms. The best model is saved based on its performance.

ACTIVITY:1

Decision Tree Model

DecisionTreeClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
dtc_pred = dtc.predict(X_test)
print('***Decision Tree Model Results***')
print(confusion_matrix(dtc_pred, y_test))
print(classification_report(dtc_pred, y_test))
```

ACTIVITY:2

Random Forest Model

RandomForestClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
print('*** Random Forest ***')
print(confusion_matrix(rf_pred, y_test))
print(classification_report(rf_pred, y_test))
```

ACTIVITY:3

Logistic Regression Model

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.

Logistic Regression algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
print(confusion_matrix(lr_pred, y_test))
print('*** Logistic Regression ***')
print(classification_report(lr_pred, y_test))
```

ACTIVITY:4

Support Vector Classifier Model

SVC algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
svc = SVC()
svc.fit(X_train, y_train)
svc_pred = svc.predict(X_test)
print('*** Support Vector Classifier ***')
print(confusion_matrix(svc_pred, y_test))
print(classification_report(svc_pred, y_test))
```

ACTIVITY:5

K-Nearest Neighbors Model

KNeighborsClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
print('*** K-Nearest Neighbours ***')
print(confusion_matrix(knn_pred, y_test))
print(classification_report(knn_pred, y_test))
```

ACTIVITY:6

Decision Tree Model Using Grid Search

Here we apply the grid search to get best hyper tuning parameters

```
cross_valid_scores = {}
```

```
 %%time
parameters = {
    "max_depth": [3, 5, 7, 9, 11, 13],
}
model_dtc = DecisionTreeClassifier(
    random_state=42,
    class_weight='balanced',
)
model_dtc = GridSearchCV(
      model_dtc,
      parameters,
      cv=5,
)
model_dtc.fit(X_train, y_train)
model_dtc_pred = model_dtc.predict(X_test)
print('** Applying Grid Search to Decision Tree**')
print(classification_report(model_dtc_pred, y_test))
print(f'Best parameters {model_dtc.best_params_}')
print(
     f'Mean cross_validated accuracy score of the best_estimator: ' + \
     f'{model_dtc.best_score_:.3f}'
    )
cross_valid_scores['desicion_tree'] = model_dtc.best_score_
 print('-----')
```

ACTIVITY:7
Random Forest Model Using Grid Search

```
parameters = {
    "n_estimators": [5, 10, 15, 20, 25],
    "max_depth": [3, 5, 7, 9, 11, 13],
    }

model_rf = RandomForestClassifier(
    random_state=42,
            class_weight='balanced',
    )

model_rf = GridSearchCV(
            model_rf,
            parameters,
    cv=5,
    )

model_rf.fit(X_train, y_train)
model_rf_pred = model_rf.predict(X_test)
print('** Applying Grid Search to Random Forest**')
print(classification_report(model_rf_pred, y_test))
print(f'Best parameters {model_rf.best_params_}')
print(
    f'Mean cross_validated accuracy score of the best_estimator: ' + \
    f'{model_rf.best_score_:.3f}'
    )
cross_valid_scores['random_forest'] = model_rf.best_score_
```

ACTIVITY:8

Evaluating Performance Of The Model And Saving The Model

From the above models, random forest is performing well. From the below image, we can see the train accuracy of the model is 97% accuracy. So, here random forest is selected as the best performing algorithm and evaluated with cross validation. Additionally, we can tune the model with hyper parameter tuning techniques.

```
***Decision Tree Model Results***
[[886  49]
 [ 36 909]]
              precision    recall  f1-score   support

           0       0.96      0.95      0.95       935
           1       0.95      0.96      0.96       945

    accuracy                           0.95      1880
   macro avg       0.95      0.95      0.95      1880
weighted avg       0.95      0.95      0.95      1880


*** Random Forest ***
[[913  44]
 [  9 914]]
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       957
           1       0.95      0.99      0.97       923

    accuracy                           0.97      1880
   macro avg       0.97      0.97      0.97      1880
weighted avg       0.97      0.97      0.97      1880


   [[688 168]
    [234 790]]
   *** Logistic Regression ***
                 precision    recall  f1-score   support

              0       0.75      0.80      0.77       856
              1       0.82      0.77      0.80      1024

       accuracy                           0.79      1880
      macro avg       0.79      0.79      0.79      1880
   weighted avg       0.79      0.79      0.79      1880
```

```
*** Support Vector Classifier ***
[[654 151]
 [268 807]]
              precision    recall  f1-score   support

           0       0.71      0.81      0.76       805
           1       0.84      0.75      0.79      1075

    accuracy                           0.78      1880
   macro avg       0.78      0.78      0.78      1880
weighted avg       0.79      0.78      0.78      1880
```

```
*** K-Nearest Neighbours ***
[[764  19]
 [158 939]]
              precision    recall  f1-score   support

           0       0.83      0.98      0.90       783
           1       0.98      0.86      0.91      1097

    accuracy                           0.91      1880
   macro avg       0.90      0.92      0.91      1880
weighted avg       0.92      0.91      0.91      1880
```

```
** Applying Grid Search to Decision Tree**
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       940
           1       0.95      0.97      0.96       940

    accuracy                           0.96      1880
   macro avg       0.96      0.96      0.96      1880
weighted avg       0.96      0.96      0.96      1880

Best parameters {'max_depth': 13}
Mean cross_validated accuracy score of the best_estimator: 0.950
-----
CPU times: total: 1.69 s
Wall time: 1.74 s
```

```
** Applying Grid Search to Random Forest**
              precision    recall  f1-score   support

           0       1.00      0.95      0.97       962
           1       0.95      1.00      0.97       918

    accuracy                           0.97      1880
   macro avg       0.97      0.97      0.97      1880
weighted avg       0.98      0.97      0.97      1880

Best parameters {'max_depth': 13, 'n_estimators': 25}
Mean cross_validated accuracy score of the best_estimator: 0.973
```

Milestone 5:

Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script

ACTIVITY:1

Building HTML Pages

We Build an HTML page to take the values from the user in a form and upon clicking on the predict button we get the prediction

```
<html>
    <head>
    <title>prediction</title>
    <link href='https://fonts.googleapis.com/css?family=Montserrat' rel='stylesheet
    <style>

    * {
        box-sizing:border-box;
    }
    body{
        font-family: 'Montserrat';
    }
        .header {
            top:0px;
            margin:0px;
            left: 0px;
            right: 0px;
            position: fixed;
            background-color: black;
            color: white;
            overflow: hidden;
            padding: 15px;
            font-size: 2vw;
            width: 100%;
            text-align: left;
            padding-left:100px;
            font-family: 'Merriweather';
            box-shadow: 0px 8px 4px grey;
            opacity: 0.9;
        }
```

```
.header_text{
    font-size: 40px;
    text-align:center;
}
.content{
    margin-top: 100px;
}
.result{
     color:red;
}
.text{
    font-size: 20px;
    margin-top: 10px;
    text-align: center;
}
   input[type=number] ,select {
    width: 50%;
    padding:12px 20px;
    margin:8px 0;
    display:inline-block;
    border:1 px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
   }
   input[type=text] ,select {
    width: 50%;
    padding:12px 20px;
    margin:8px 0;
    display:inline-block;
    border:1 px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
   }
```

```
    input[type=submit] ,select {
     width: 50%;
     background-color: #000000;
     color: white;
     padding:14px 20px;
     margin-bottom: 20px;
     font-size: 25px;
     color:red;
    }
```

```
margin-bottom:20px;
font-size:25px;
color:red;
}
</style>
</head>
<body align=center>
<div class="header">
        <div>Stroke Prediction </div>
</div>
<div class="content">
<div class="header_text">Stroke Prediction</div>
<div class="text">Fill in and below details to predict whether a person might get a stroke.</div>
<div class="result">
  {{ prediction_text }}
  </div>
<form action="{{ url_for('y_predict') }}" method="POST">
    <input type="text" id="gender" name="Gender" placeholder="gender">
    <input type="number" id="age" name="Age" placeholder="age">
    <input type="number"  id="hypertension" name="Hypertension" placeholder="hypertension(0/1)">
    <input type="number"  id="heart_disease" name="Heart Disease" placeholder="heart_disease(0/1)">
    <input type="text"  id="ever_married" name="Ever Married" placeholder="ever_married">
    <input type="text"  id="work_type" name="Work Type" placeholder="work_type">
    <input type="text"  id="Residence_type" name="Residence Type" placeholder="Residence_type">
    <input type="number" step="any" id="avg_glucose_level" name="avg glucose level" placeholder="Avg Glucose Level">
    <input type="number" step="any" id="bmi" name="BMI" placeholder="BMI">
    <input type="text"  id="smoking_status" name="smoking_status" placeholder="smoking_status">

    <input type="submit" value="Submit">
  </form>

</div>
</body>
</html>
```

ACTIVITY:2

Build Python Code

In the flask application, the user values are taken from the HTML page.

```
from flask import Flask, request,render_template
import joblib
import numpy as np
app = Flask(__name__)
model = joblib.load("model")
label1 = joblib.load("mar_transform")
label2 = joblib.load("res_transform")
column = joblib.load("column")
```

Load the home page

```
@app.route('/')
def predict():
    # Main page
    return render_template('Manual_predict.html')
```

Prediction function

```
@app.route('/y_predict', methods=['POST'])
def y_predict():
    x_test = [[(x) for x in request.form.values()]]
    print('actual',x_test)
    x_test=np.array(x_test)
    x_test[:,4]=label1.transform(x_test[:,4])
    x_test[:,6]=label2.transform(x_test[:,6])
    x_test=column.transform(x_test)
    pred = model.predict(x_test)
    print(pred)
    if(pred[0]==0):
        result="no chances of stroke"
    else:
        result="chances of stroke"

    return render_template('Manual_predict.html', \
                           prediction_text=('There are \
                                            ',result))
if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

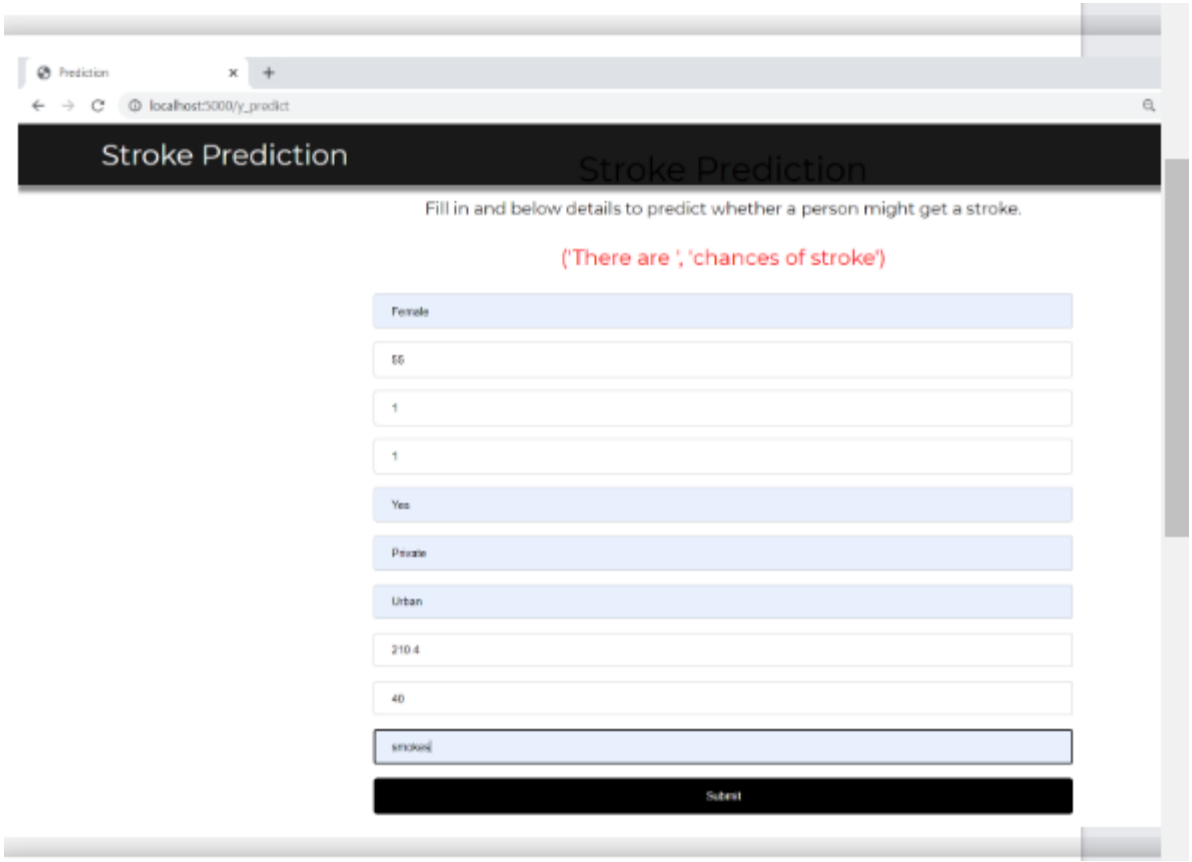
ACTIVITY:3

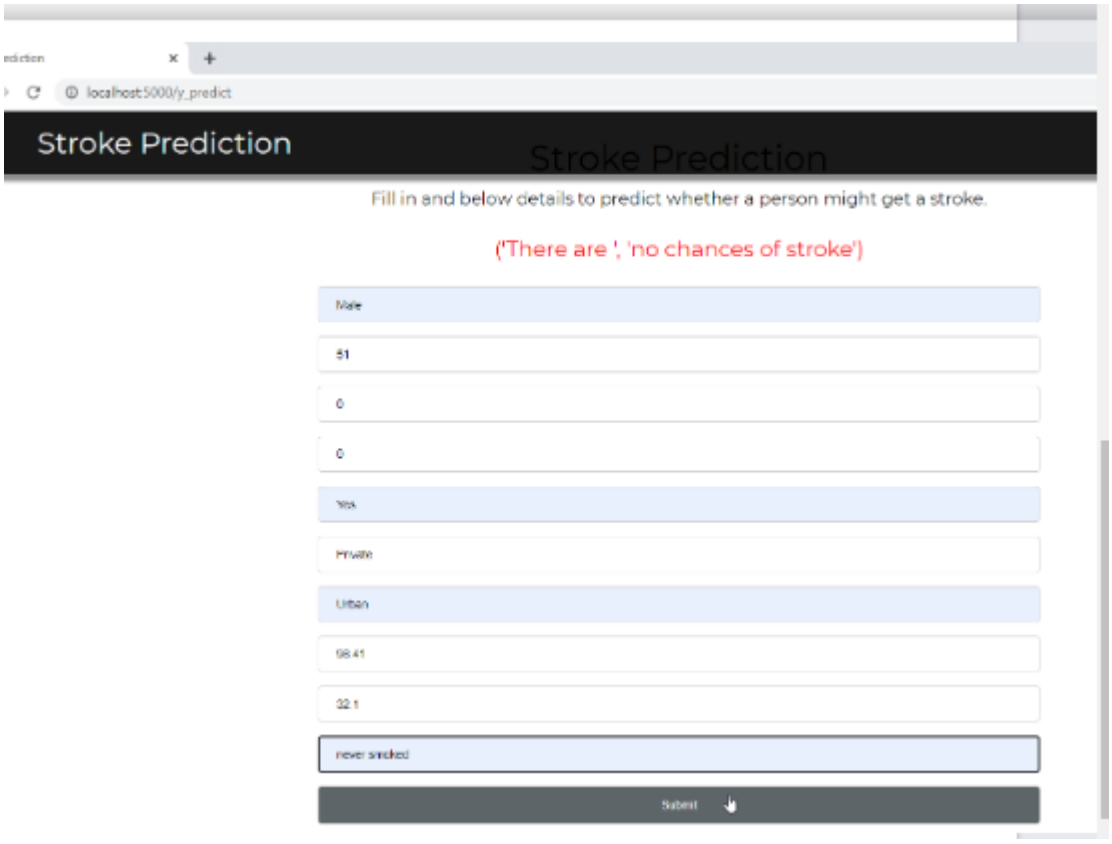
Run The App

Step 1: Open anaconda prompt go to the project folder and in that go to flask folder and run the python file by using the command “python app.py”

```
* Debugger is active!
* Debugger PIN: 764-456-992
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.115.10:5000/ (Press CTRL+C to quit)
```

Output:


Prediction
localhost:5000/y_predict
Stroke Prediction
Stroke Prediction
Fill in and below details to predict whether a person might get a stroke.
('There are ', 'chances of stroke')
Female
55
1
1
Yes
Private
Urban
210.4
40
smokes
Submit


localhost:5000/y_predict
Stroke Prediction
Stroke Prediction
Fill in and below details to predict whether a person might get a stroke.
('There are ', 'no chances of stroke')
Male
51
0
0
Yes
Private
Urban
98.41
32.1
never smoked
Submit