A Project Phase II Report on

Brain Stroke Prediction Using Machine Learning and Data Science

Submitted to the Dept. of Information Technology, SNIST
in the partial fulfillment of the academic requirements for the award of

B. Tech (Information Technology)

under JNTUH

by

Anil. M (17311A1289)
Abhinay. L(17311A12A7)
Venkatesh. V(17311A12A8)
…

under the guidance of

Ms. N. Sreevidya
Assistant Professor

Department of Information Technology

School of Computer Science and Informatics
Sreenidhi Institute of Science and Technology (An Autonomous Institution)
Yamnampet, Ghatkesar Mandal, R. R. Dist., Hyderabad – 501301

affiliated to

Jawaharlal Nehru Technological University Hyderabad
Hyderabad – 500 085

2020–21

Department of Information Technology

School of Computer Science and Informatics
Sreenidhi Institute of Science and Technology

Certificate

This is to certify that the Project Phase II report on **"Brain Stroke Prediction Using Machine Learning and Data Science"** is a Bonafide work carried out by Anil. M (17311A1289), Abhinay. L (17311A12A7), Venkatesh. V (17311A12A8) in the partial fulfillment for the award of B-Tech degree in Information Technology, Sreenidhi Institute of Science and Technology, Hyderabad, affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH), Hyderabad under our guidance and supervision.

The results embodied in the Project Phase II work have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal guide (Ms.N.Sreevidya)	**Project Coordinator (Mr.M.Srinivas)**	**Head of the Department (Dr. V. V. S. S. S. Balaram)**

External Examiner

Date:

DECLARATION

We, **Anil.M(17311A1289), Abhinay.L(17311A12A7) and Venkatesh.V (17311A12A8),** students of **Sreenidhi Institute of Science and Technology, Yamnampet, Ghatkesar,** studying IVth year IInd semester, **Information Technology** solemnly declare that the Project Phase II work, titled **"Brain Stroke Prediction Using Machine Learning and Data Science"** is submitted to **Sreenidhi Institute of Science and Technology** for partial fulfillment for the award of degree of Bachelor of technology in **Information Technology**.

It is declared to the best of our knowledge that the work reported does not form part of any dissertation submitted to any other University or Institute for award of any degree.

ACKNOWLEDGMENTS

We would like to express our immense gratitude and sincere thanks to **Ms.N.Sreevidya, Assistant Professor** in Information Technology for **her** guidance, valuable suggestions and encouragement in completing the Project Phase II work within the stipulated time.

We would like to express our sincere thanks to **Dr. P. Narasimha Reddy**, Executive Director, **Dr. T. Ch. Siva Reddy**, Principal, **Dr. V. V. S. S. S. Balaram,** Professor & Head of the Department of Information Technology, **Mr.M.Srinivas**, Associate Professor & Project Phase II Work Coordinator of the Department of Information Technology, Sreenidhi Institute of Science and Technology (An Autonomous Institution), Hyderabad for permitting us to do our Project Phase II work.

Finally, we would also like to thank the people who have directly or indirectly helped us and parents and friends for their cooperation in completing the Project Phase II work.

Anil. M (17311A1289)
Abhinay. L(17311A12A7)
Venkatesh. V(17311A12A8)

ABSTRACT

BRAIN STROKE PREDICTION USING ML AND DATA SCIENCE

Stroke is one of the leading causes of the death worldwide these days. About 1/5th of patients with an acute stroke dies within a month of event and at least 1/2 of those who survive are left with physical disability. As we study some stats, we can see that, there are 15 million people worldwide who suffer a stroke each year. According to the World Health Organization (WHO), stroke is the second leading cause of death for people above the age of 60 years, and the fifth leading cause in people aged 15 to 59 years old. Each year, nearly six million people worldwide die from stroke. One in six people worldwide will have a stroke in their lifetime. Every six seconds, stroke kills some. As the study suggests hypertension remained the most common risk factor for Stroke followed by Smoking and diabetes Mellitus and dyslipidemia. In fact Strokes continues to play and pivotal role in killing as many humans getting killed by Aids, Tuberculosis and Malaria combined. So, brain stroke is a medical emergency and can lead to death or permanent disability. One needs to react fast and need to get emergency medical attention by calling to 1-0-8 or 9-1-1(International). According to the World stroke organization reports of the year 2019 suggests that,

1. Brain Attacks devastates lives around the world.
2. 13.7 M new strokes each year.
3. 80M stroke survivors worldwide.
4. 5.5M death due to stroke each year.
5. 1 in 4 people over age 25 will experience stroke in their lifetime.

So, as a group we came up with an idea of fine tuning the data set collected form Kaggle data repository and build a predictive model to estimate whether a person is suffering from a Brain stroke or not.

Anil. M (17311A1289)

Abhinay. L (17311A12A7)

Venkatesh. V (17311A12A8)

IV-II (2021), Dept. of IT, SNIST

Project Guide:

Ms. N. Sreevidya

Assistant Professor

Dept. of IT, SNIST

LIST OF FIGURES

INDEX

INTRODUCTION:

Scope:

1. As We All know how the world is Revolutionizing with the likes of Machine Learning, Artificial Intelligence and Deep Learning, where everyday a new approach of solving existing Problem is been discovered and helping us save a lot of time and resources.
2. As, The Machine Learning (ML) Algorithms are Rapidly evolving, delivering an accurate and quick prediction outcome and it has become a powerful tool in health settings, offering personalized clinical care for all the patients suffering from various Diseases.
3. Our Project Aims to build the better model In terms of all the evaluation metrics and we hope it might help in the field of Healthcare, so that the death rates related to a particular disease sector might reduce.
4. The Main Objective of our Project is to Extract the Patterns and Insights from the Data collected form the Various Sources and Use It to Detect/Predict Whether a Person is Suffering From Brain Stroke or Not.
5. Apart from this Prediction we can also find some Useful information in the Exploratory Data Analysis(EDA) Stage, where we can find information like, whether the Males/Females are more likely to be suffering from this Disease, What is the min Age groups of the people suffering from this disease and All other Permutations and combinations.

Existing System:

As we all know NON contrast CT scan is the current standard for initial screening of the head trauma and Brain Related Diseases, Some other tests that we may include are:

1. A physical exam
2. Blood Tests
3. Computerized tomography(CT) scan
4. Magnetic resonance imaging(MRI)
5. carotid ultrasound
6. Cerebral angiogram
7. Electrocardiogram

Proposed System:

We have proposed a system/model which might help us to detect a patient who is suffering from Brain Stroke as early as possible based on the inputs the patient provides.
Even though we predict an Accurate Result Still we might want to take the above tests so as to know the exact location where the blood clot has occurred, in order to take necessary actions.
So, This model helps us to predict the stroke in patients and helps the patients not to proceed to further stages of testing, which might in turn save a lot of time and money.

SYSTEM ANALYSIS:

Functional Requirements Specifications:

As the project is comprised of a model which helps us to detect whether the person is suffering from Brain Stroke or not, Where the Patient Details should be fed into the model to get the result.

User Story 1:

As a Patient, I must be able to feed the data to the model.

Functional Requirements:

1. Gather the Necessary Data Prescribed by medical officers/personals.
2. Verify the documents.
3. Insert the data into the model through key board.

Performance Requirements:

Response Time:

1. Time Taken to execute the entire notebook: 17:10:87, It might be more or less depending on the system we are working on.
2. Amount of time taken to access Jupyter notebook: 2sec
3. Amount of time taken to access Google colab: 14sec
4. Time taken to load all the jupyter notebook cells: 10sec
5. Time complexity for each cell in the notebook differs drastically cell to cell.

Software Requirements:

1. Google Colab
2. Jupyter Notebook
3. Libre Office
4. Python
5. Microsoft Windows 10
6. Ms Office
7. Vs code
8. Spotify

Hardware Requirements:

1. Intel core i5 or AMD Ryzen 5
2. Minimum 10 GB SSD/HDD
3. Ram Preferably >8 for smooth execution, if not 4 GB might be sufficient.
4. System Compatible with Windows/Linux/Mac Operating System.

SYSTEM DESIGN:

UML Diagrams:

1. Use Case Diagram

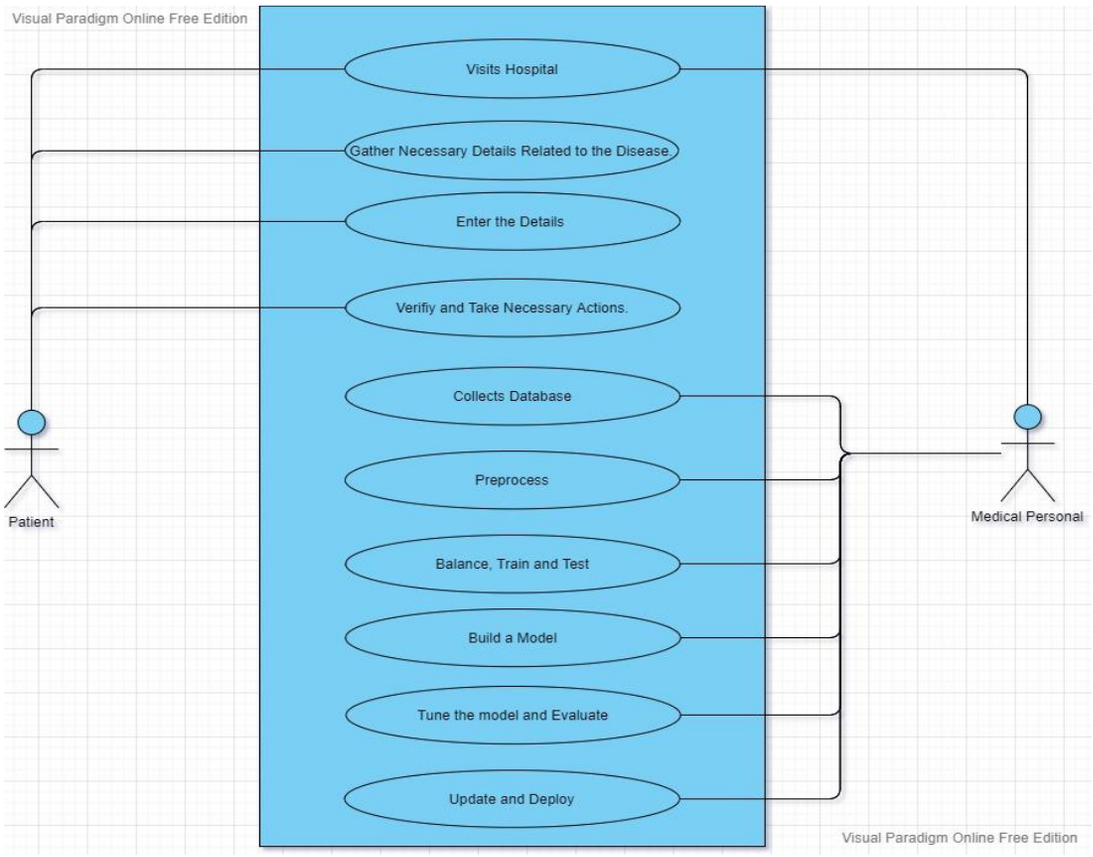


Fig 1.1

2. Class Diagram

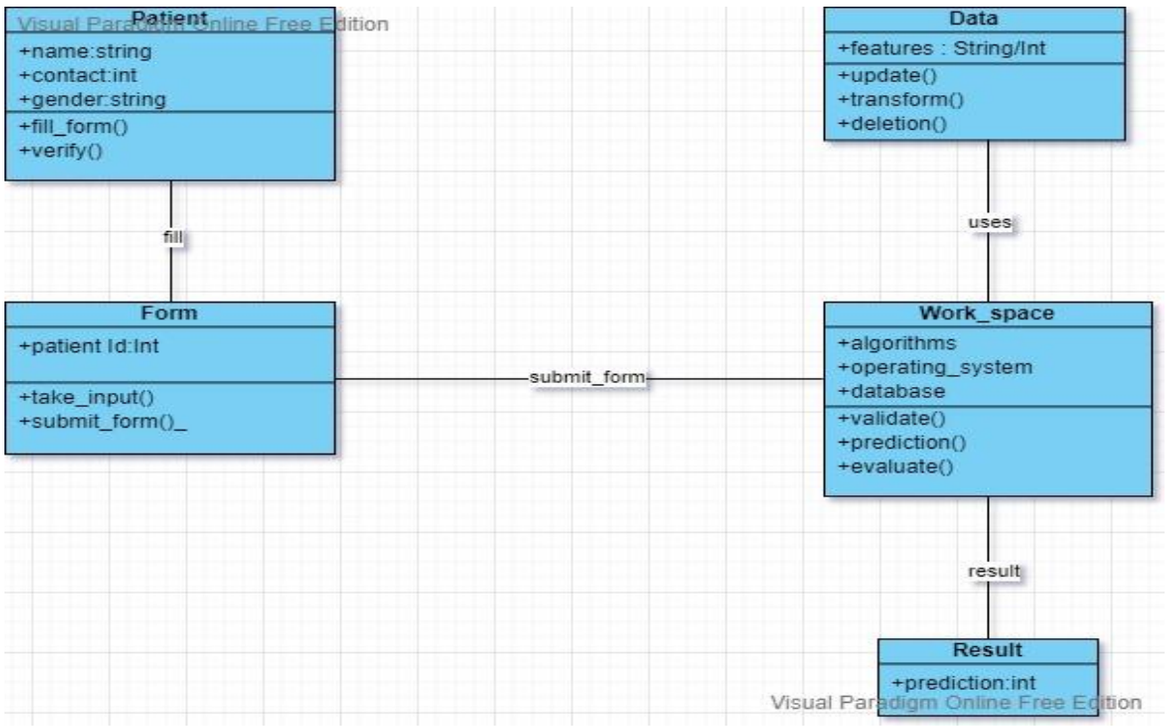


Fig 1.2

3. Activity Diagram

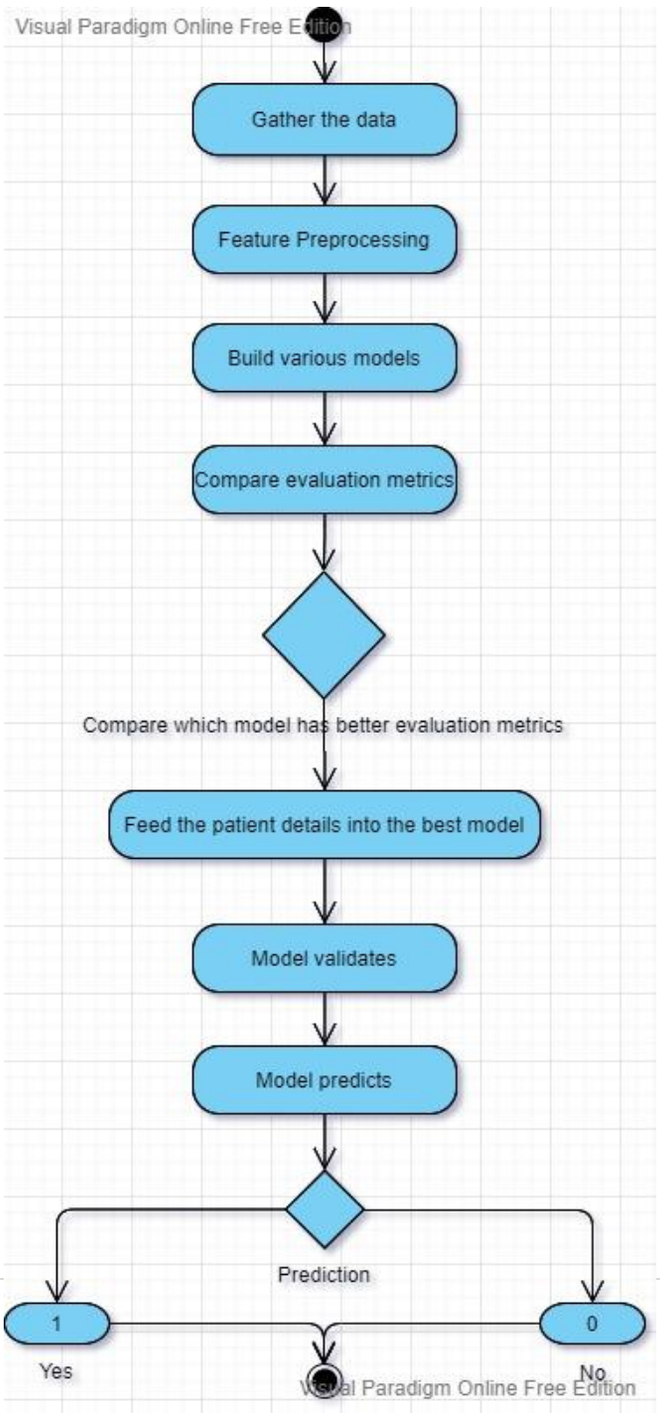


Fig 1.3

SYSTEM IMPLEMENTATION

```
# **Brain Stroke Prediction Using Machine Learning and Data Science.**

### **Importing Necessary Libraries.**

#pip install autoviz
#pip install pandas
#pip install matplotlib.pyplot
#pip install seaborns
#pip install numpy
#pip install sklearn
#pip install collections
#pip install ipywidgets
#pip install imblearn
#pip install statsmodels
#pip install warnings

import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('dark_background')
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
precision_recall_curve, precision_score, recall_score,f1_score, roc_auc_score, roc_curve, auc
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import RandomOverSampler,SMOTE
from imblearn.combine import SMOTETomek
from collections import Counter
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import warnings
warnings.filterwarnings(action='ignore')
```

```
from sklearn import tree
import autoviz
from autoviz.AutoViz_Class import AutoViz_Class
#for interactive console
import ipywidgets
import ipywidgets as widgets
from ipywidgets import interact
from ipywidgets import interact_manual
```

Importing and Skimming the Data Set.
The Data set consists of 40000+ entries of Patients Regarding Brain Stroke symptoms.
There are total of 12 columns including target_column.
1. id
2. gender
3. age
4. hypertension
5. heart_disease
6. ever_married
7. work_type
8. Residence_type
9. avg_glucose_level
10. bmi
11. smoking_status
12. stroke(target_column)

```
dodge = pd.read_csv('train_strokes.csv')

# head() helps us to view the first 5 entries in our dataset.

dodge.head()

# info() gives us the count and dtype, also helps us to identify whether there are any null values or not.

dodge.info()

# describe() gives us a breif description about the columns(count, min, max, mean, median etc)

dodge.describe()

# In the case of object columns we get(count, unique values, top, freq)
dodge.describe(include = 'object')
```

Exploring Target Variable.

```
dodge['stroke'].value_counts()

# There arent any null values, but
dodge['stroke'].isnull().sum()

# This plot tell's about, how the distribution of target class is spreaded.
# we can see that the target classes are highly imbalanced with 0->42617, 1->783, so we need to balance these target classes which we will see in the later part.
# countplot() helps us to visualize the count the classes.

plt.figure(figsize = (6,4), dpi = 100)
```

```
sns.countplot(dodge['stroke'])
plt.xlabel('Stroke Status')
plt.ylabel('Count')
plt.title('Distribution of Target Classes')
plt.show()
```

Exploring Independent Numerical Columns.

1. Cleaning
2. Treating Missing values
3. Anamoly Detection and Reduction

```
numerical = ['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi']
#dodge[numerical[0]]
```

Treating missing values present in the column dodge['bmi'], no other numerical columns has missing values.

```
dodge['bmi'].isnull().sum()

dodge['bmi'] = dodge['bmi'].fillna(dodge['bmi'].mean())

dodge['bmi'].isnull().sum()
```

Exploring each numerical column using describe()

```
for i in numerical:
  print(dodge[i].describe())
```

Anamoly Detection and Reduction in Numericals.

1. age

```
dodge['age'].describe()

dodge['age'].value_counts()
```

Function to check the Anamolies in the column using upper_limit and lower_limit.

1. If the upper_limit > max(df['col']), then we replace the upper_limit with the max value.
2. Similarly, if the lower_limit < min(df['col']), we replace the lower_limit with the min value.

```
anamolies = []
def outliers(data):
  random_state_mean = np.mean(data)
  random_state_std = np.std(data)
  anamoly = random_state_std * 3

  upper_limit = random_state_mean + anamoly
  lower_limit = random_state_mean - anamoly
  lp_lower_limit = 1.00
  up_upper_limit = max(dodge['age'])
  print(upper_limit)
  print(lower_limit)

  print(lp_lower_limit)
  print(up_upper_limit)

  for i in data:
      if i < lp_lower_limit or i > up_upper_limit:
        anamolies.append(i)

outliers(dodge['age'])
print(len(anamolies))

dodge.shape
```

Here all the values below 1 are termed as outliers, although in rarest of cases Intrauterine stroke occur to unborn childre in the womb.

But in this project we drop those values, but in future we can even work on these values.

```
dodge[dodge['age'] < 1.00]

dodge[dodge['age'] < 1.00].index

chevy = dodge.drop(index = dodge[dodge['age'] < 1.00].index, axis = 0, inplace=True)

dodge.drop(index = dodge[(dodge.age > 1.0) & (dodge.age < 2.0)].index, axis = 0, inplace = True)

dodge.shape
```

2. avg_glucose_level(Average Glucose Level)

```
anamolies = []
def outliers(data):
  random_state_mean = np.mean(data)
  random_state_std = np.std(data)
  anamoly = random_state_std * 3

  upper_limit = random_state_mean + anamoly
  lower_limit = random_state_mean - anamoly
  ll_p = min(dodge['avg_glucose_level'])
```

```
  print(upper_limit)
  print(lower_limit)
  print(ll_p)
  for i in data:
    if i < ll_p or i > upper_limit:
      anamolies.append(i)

outliers(dodge['avg_glucose_level'])
print(len(anamolies))

dodge['avg_glucose_level'].describe()

dodge[dodge['avg_glucose_level'] > 234.40827023316058 ]

dodge[dodge['avg_glucose_level'] > 234.40827023316058].index

dodge.drop(index = dodge[dodge['avg_glucose_level'] > 234.40827023316058].index, axis = 0,
inplace = True)

dodge.shape

#### 3. bmi(Body Mass Index)

anamolies = []
def outliers(data):
  random_state_mean = np.mean(data)
  random_state_std = np.std(data)
  anamoly = random_state_std * 3

  upper_limit = random_state_mean + anamoly
  lower_limit = random_state_mean - anamoly
  lll_p = min(dodge['bmi'])

  print(upper_limit)
  print(lower_limit)
  print(lll_p)
  for i in data:
    if i < lll_p or i > upper_limit:
      anamolies.append(i)

outliers(dodge['bmi'])
print(len(anamolies))

dodge['bmi'].describe()

dodge[dodge['bmi'] > 51.35486554902225]

dodge[dodge['bmi'] > 51.35486554902225].index

dodge.drop(index = dodge[dodge['bmi'] > 51.35486554902225].index, axis = 0, inplace = True)

dodge.shape

### **Exploring Independent Categorical(Object/String) Columns.**
```

1. Cleaning
2. Treating Missing values

```
dodge.isnull().sum()

categorical = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']

for i in categorical:
  print(dodge[i].describe())

dodge.describe(include = 'object')

dodge['smoking_status'].value_counts()

dodge.describe(include = 'all')
```

Treating Missing values in Object columns using,
1. mean/median/mode
2. Based on frequency Distribution.

```
dodge['smoking_status'].mode()

dodge['smoking_status'].fillna('never smoked',inplace = True)

dodge['smoking_status'].isnull().sum()

dodge.info()

dodge.head()
```

Exploratory Data Analysis.
Exploratory Data Analysis helps us the understand the insights and extract the patterns from the dataset, which might be helpful to explain about the problem statement given to our clients.
This can also be done by using traditional python code, But Visualizing the data looks more eye catching than looking at some numbers and letters.
so, hence we are going to use various plots and graphs to visualize, which comes from the libraries such as,
seaborn and matplotlib.pyplot.
1. bar
2. countplot
3. piechart
4. hist
5. box
6. scatterplot
7. pairplot

Apart from this we have also used and auto visualization tool, "autoviz"

```
dodge.isnull().sum()

dodge.drop(columns = 'id', inplace=True)

dodge.head()
```

-> pd.crosstab() function is a very useful and most advanced fuction in the python dataframe, it helps us to compare 2 variables, due to which we can plot the distribution of thsoe variables.

1. Bar plot for crosstab distribution between gender and stroke.

```
plt.figure(figsize = (8,6))
x = pd.crosstab(dodge['gender'], dodge['stroke'])
x.plot(kind = 'bar')
#x.div(x.sum(1).astype(float), axis = 0).plot(kind='bar', stacked = False)
plt.xlabel('Gender_distribution')
plt.ylabel('Count')
plt.title('Gender Distribution over Target Class')
plt.show()
```

2. Pie Chart for distribution of gender.

```
# PIE CHART for dodge['gender'] column.

plt.figure(figsize = (8,6), dpi = 90)
labels = dodge['gender'].value_counts().index
sizes = dodge['gender'].value_counts()
explode = [0,0,0.1]
colors = plt.cm.autumn(np.linspace(0,1,3))
plt.pie(sizes, colors=colors, labels=labels, explode=explode, shadow =True, startangle=90, autopct = '%.2f%%' )
plt.title('Gender',fontsize=12)
plt.legend()
plt.show()
```

3. Bar chart for gender-hypertentsion distribution.

```
plt.figure(figsize = (8,6), dpi = 90)
x = pd.crosstab(dodge['gender'],dodge['hypertension'])
x.plot(kind = 'bar')
plt.xlabel('Gender')
plt.ylabel('Hypertension')
plt.title("Gender_Hypertension_Distribution")
plt.show()
```

4. Bar Chart for age-hypertension distribution

```
plt.rcParams['figure.figsize'] = (20,12)
x = pd.crosstab(dodge['age'], dodge['hypertension'])
x.plot(kind = 'bar')
plt.xlabel('Age')
plt.ylabel('Count')
plt.title("Age Hypertension Distrubition")
plt.show()
```

5. Bar Chart for gender-heart_disease distribution

```
plt.figure(figsize=(12,10))
```

```
ab = pd.crosstab(dodge['gender'], dodge['heart_disease'])
ab.plot(kind = 'bar')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

6. age-stroke distribution

```
plt.rcParams['figure.figsize'] = (20,12)
x = pd.crosstab(dodge['age'], dodge['stroke'])
x.plot(kind = 'bar')
plt.xlabel('Age')
plt.ylabel('Count')
plt.title("Age_Stroke Distrubition")
plt.show()
```

7. age-heart_disease distribution.

```
plt.rcParams['figure.figsize'] = (20,12)
#plt.figure(figsize =(13,6))
x = pd.crosstab(dodge['age'], dodge['heart_disease'])
x.plot(kind = 'bar')
plt.xlabel('Age')
plt.ylabel('Count')
plt.title("Age Heart_Disease Distrubition")
plt.show()
```

8. Distribution of people getting stroke with respect to whether they are married or not.

```
plt.rcParams['figure.figsize'] = (8,6)
h = pd.crosstab(dodge['ever_married'], dodge['stroke'])
h.plot(kind ='bar')
plt.show()
```

9. Scatterplot for avg_glucose level and bmi with hue as stroke, hue is an additional parameter which seperates the classes using different colors.

```
plt.rcParams['figure.figsize'] = (20,12)
sns.relplot(dodge['avg_glucose_level'], dodge['bmi'], hue = dodge['stroke'], kind = 'scatter')
plt.xlabel('Avg_Glucose_Level')
plt.ylabel('BMI')
plt.show()
```

10. Countplot() for checking distribution of work_type.

```
plt.figure(figsize = (12,10))
sns.countplot(dodge['work_type'], color ='red')
plt.xlabel("Work Type")
plt.ylabel('Count')
plt.title("Distribution of Work_type")
plt.show()
```

11. Distribution of work_type with respect to stroke occurence.

```
plt.rcParams['figure.figsize'] = (8,6)
h = pd.crosstab(dodge['work_type'], dodge['stroke'])
```

```
h.plot(kind ='bar')
plt.xlabel("Work_type")
plt.ylabel("Count")
plt.title("Distribution of Work_type and stroke")
plt.show()
```

autoviz -> An AutoVisualization tool, which helps to visualize the features in the dataset more in depth.

```
AV = AutoViz_Class()
autovis = AV.AutoViz(filename = 'train_strokes.csv', sep=',', depVar='', dfte=None, header=0, verbose=2,

lowess=False,chart_format='svg',max_rows_analyzed=150000,max_cols_analyzed=30)
autovis
```

Exploring data using Traditional python code, with the help of interactive widgets.

```
abg = dodge[['hypertension', 'heart_disease']].groupby(['hypertension']).count().style.background_gradient(cmap = 'viridis')
```

Sum of Heart Disease values with respect to hypertension, This can be easily eaxplained by crosstab()

```
abg

dre = pd.crosstab(dodge['hypertension'], dodge['heart_disease'])
dre
```

@interact:
The interact function (ipywidgets.interact) automatically creates user interface (UI) controls for exploring code and data interactively.

The function gets called each time the slider is moved.

```
@interact
def abc(x = 50):
  y = dodge[dodge['avg_glucose_level'] > x]
  return y['stroke'].value_counts()
abc()

@interact
def hyp_heart(x=0, y=0):
  g = dodge[(dodge['hypertension'] == x) & (dodge['heart_disease'] == y)]
  return g['stroke'].value_counts()
hyp_heart()

@interact
def hy_he_eve(x=0,y=0,z='No'):
  j = dodge[(dodge['hypertension'] == x) & (dodge['heart_disease'] == y) & (dodge['ever_married'] == z)]
  return j['stroke'].value_counts(), j['smoking_status'].value_counts()
hy_he_eve()
```

Feature Transformation.

Feature Transformation is the technique of transforming the variable into other form like Strings -> Numeric, splitting the Date Column in to pieces etc.

Types of encoding.
1. Nominal Encoding.
* one hot encoding -> Creating Dummy variables.
* one hot encoding with multi categories(more than 20 categories)
* mean encoding

2. Ordinal Encoding.
* Label Encoder
* target_guided_encoding

-> For the columns with less than 5 categories we can manually perform encoding, usinf map().

-> For Columns with more than 20 Categories we can perform one hot encoding with multi categories, where we tend to select the top categories based on their value_counts().

```
dodge.head()

dodge['smoking_status'].unique()

mapping = {'Male':2, 'Female':1, 'Other':0}
mapping1 = {'No':0, 'Yes':1}
mapping2 = {'never smoked':0, 'formerly smoked':1, 'smokes':2}

dodge['gender'] = dodge['gender'].map(mapping)

dodge['ever_married'] = dodge['ever_married'].map(mapping1)

dodge['smoking_status'] = dodge['smoking_status'].map(mapping2)

dodge[['gender', 'smoking_status', 'ever_married']].head()

dodge['work_type'].unique()

dodge['Residence_type'].unique()

dodge['home_town'] = pd.get_dummies(dodge['Residence_type'], drop_first = True)
```

Creating a new dataframe with respect to work_type.

```
f150 = pd.get_dummies(dodge['work_type'], drop_first = True)
```

Merging 2 DataFrames(dodge,f150) with the default join.

```
camero = pd.concat([dodge,f150], axis = 1)

camero.head()

camero.rename(columns = {'Never_worked':'w_t_n_w', 'Private':'w_t_p', 'Self-employed':'w_t_s_e', 'children':'w_t_c'}, inplace = True)
```

Droping the columns ['work_type', 'Residence_type'], as we have already created dummy variables for them.

```
camero.drop(columns = ['work_type','Residence_type'], inplace = True)

camero.head()

camero.info()
```

Feature Scaling
Feature Scaling is the technique to scale down all the values in the datset to same level, so that there will be no partiality while we train the model like bmi -> 56 getting high priority than heart_disease -> 0, so in order to remove this error, feature scaling is done.

Feature Scaling Tools.
1. Standardisation (values are centered around the mean with unit standard deviation.)
2. Normalisation/min_max scaling.(values range from 0 to 1)

StandardScaler()

```
se = StandardScaler()
abh = se.fit_transform(camero.drop(columns=['stroke']))
mercury = pd.DataFrame(data = abh, columns = camero.drop(columns = ['stroke']).columns)
mercury.head()
```

Feature Selection
Selecting the best features which best contribute to our model.

Correlation Diagram.

```
plt.rcParams['figure.figsize'] = (20,12)
corr = mercury.corr()
sns.heatmap(corr, annot=True, cmap='autumn')
plt.show()
```

Function to select the best features with some threshold value.

```
def correlation(dataset,threshold):
    corr_list = []
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j] > threshold):
                column = corr_matrix.columns[[i,j]]
                corr_list.append(column)
    print(len(corr_list))
    return corr_list
correlation(mercury,0.6)
```

Although, we can see ['ever_married', 'age'] are somewhat correlated, but where as if we use 'Variance Inflation Factor'', we ended up with fixing the Multicollinearirty.

variance_inflation_factor -> it is used to remove multicollinearity between variables by removing as few variables as possible.

VIF->Variance Inflation Factor

```
vif = variance_inflation_factor
earth1 = pd.Series([vif(mercury.values, i) for i in range(mercury.shape[1])], index = mercury.columns)
earth1
```

Function to check and remove multicollinearity between independent variables.

```
def mc(data):
  earth = pd.Series([vif(data.values, i) for i in range(data.shape[1])], index = data.columns)
  if earth.max() > 6:
    print(earth[earth == earth.max()].index[0], 'Has Been Removed.')
    data = data.drop(columns = earth[earth == earth.max()].index[0])
  else:
    print("MultiCollinearity Has Been Removed.")
    return data

for i in range(5):
  mercury = mc(mercury)
mercury.head()
```

Splitting Data
Splitting the dataset

1. target_var
2. Independent_var

```
target_var = camero['stroke']
inde_vars = camero.drop(columns=['stroke'], axis = 1)

target_var

inde_vars.head()
```

Handling Imbalanced Dataset.
As we saw the target_calss was highly imbalanced, so we try to balance the target_class using Oversampling method, using "SMOTETomek" tool.

```
camero.head()

#### SMOTETomek Tool

so = SMOTETomek()
x_resample,y_resample = so.fit_sample(inde_vars, target_var.values.ravel())
brad = pd.DataFrame(data=x_resample, columns = inde_vars.columns)

#Before resampling
print("Before Resampling Target_Variable: ")
print(target_var.value_counts())

# After resampling
y_resample = pd.DataFrame(y_resample)
print("After Resampling Target_Variable:")
print(y_resample[0].value_counts())
```

Train Test Split.

Splitting the data into train and test datasets.

```
x_train,x_test,y_train,y_test = train_test_split(x_resample, y_resample, test_size = 0.3, random_state = 50)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

Feature Scaling Balanced Data.
Now, as we have balanced our data, we need to perform feature scaling to the banlanced data.

```
x_train_ss = se.fit_transform(x_train)
x_test_ss = se.transform(x_test)
```

Creating Test Data.

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

ford.head()

ford.info()

ford.drop(index = ford[(ford.age > 1.0) & (ford.age < 2.0)].index, axis = 0, inplace = True)

ford.info()

ford.shape

anamolies = []
def outliers(data):
  random_state_mean = np.mean(data)
  random_state_std = np.std(data)
  anamoly = random_state_std * 3

  upper_limit = random_state_mean + anamoly
  lower_limit = random_state_mean - anamoly
  uu =  max(ford['avg_glucose_level'])
  ll = min(ford['avg_glucose_level'])

  print(upper_limit)
  print(lower_limit)
  for i in data:
    if i < ll or i > uu:
      anamolies.append(i)

outliers(ford['avg_glucose_level'])
print(len(anamolies))

dodge['avg_glucose_level'].describe()

anamolies = []
def outliers(data):
  random_state_mean = np.mean(data)
  random_state_std = np.std(data)
```

```
  anamoly = random_state_std * 3

  upper_limit = random_state_mean + anamoly
  lower_limit = random_state_mean - anamoly
  ll = min(ford['bmi'])

  print(upper_limit)
  print(lower_limit)
  for i in data:
    if i < ll or i > upper_limit:
      anamolies.append(i)

outliers(ford['bmi'])
print(len(anamolies))

ford[ford['bmi'] > 52.45615973942819]

ford.drop(index = ford[ford['bmi'] > 52.45615973942819].index, axis = 0, inplace = True)

ford.shape

ford.isnull().sum()

ford['bmi'].mean()

ford['bmi'].fillna(ford['bmi'].mean(), inplace = True)

ford['bmi'].isnull().sum()

ford['smoking_status'].replace('Unknown', 'never smoked')

ford.info()

ford.drop(columns = ['id'], axis=1, inplace = True)

ford['smoking_status'].replace({'Unknown':'never smoked'}, inplace = True)

ford['gender'] = ford['gender'].map(mapping)

ford['ever_married'] = ford['ever_married'].map(mapping1)

ford['smoking_status'] = ford['smoking_status'].map(mapping2)

ford[['gender', 'smoking_status', 'ever_married']].head()

ford['work_type'].unique()

ford['Residence_type'].unique()

ford['home_town'] = pd.get_dummies(ford['Residence_type'], drop_first = True)

rap = pd.get_dummies(ford['work_type'], drop_first = True)

cam = pd.concat([ford,rap], axis = 1)

cam.head()
```

```
cam.rename(columns = {'Never_worked':'w_t_n_w', 'Private':'w_t_p', 'Self-employed':'w_t_s_e',
'children':'w_t_c'}, inplace = True)

cam.drop(columns = ['work_type','Residence_type'], inplace = True)

target = cam['stroke']
original = cam.drop(columns = ['stroke'])

resampled_x,resampled_y = so.fit_resample(original,target.values.ravel())
pitt = pd.DataFrame(data = resampled_x, columns=original.columns)

#Before resampling
print("Before Resampling Target_Variable: ")
print(target.value_counts())

# After resampling
resampled_y = pd.DataFrame(resampled_y)
print("After Resampling Target_Variable:")
print(resampled_y[0].value_counts())

fish = se.fit_transform(resampled_x)
lucas = pd.DataFrame(data = fish, columns = original.columns)

lucas.head()

lucas.info()

## **Building Predictive Models.**
1. Decision Tree
2. Random Forest
3. Logistic Regression

## **Decision Tree Classifier**

from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train_ss,y_train)
predictions = dt.predict(x_test_ss)

print('The Training Accuracy of x_train and y_train is', dt.score(x_train_ss,y_train))
print("The Testing Accuracy of x_test and y_test is", dt.score(x_test_ss,y_test))

print(confusion_matrix(predictions,y_test))

print(classification_report(predictions,y_test))

print(accuracy_score(predictions, y_test))

### **Tree Plot.**

plt.figure(figsize = (15,10))
tree.plot_tree(dt, filled = True)

### Performing Cross Validation on DT Model.
```

Performing cross validation on the dataset using StratifiedKFold and calculating the mean Accuracy that can be achieved by the model.

```
x = pd.DataFrame(data = x_train_ss, columns = inde_vars.columns)
y = y_train
from sklearn.model_selection import  StratifiedKFold
accuracy = []
skf = StratifiedKFold(n_splits = 10, random_state = None)
skf.get_n_splits(x,y)
for train_index, test_index in skf.split(x,y):
  print('Train:', train_index, 'Validation',test_index)
  x1_train,x1_test = x.iloc[train_index],x.iloc[test_index]
  y1_train,y1_test = y.iloc[train_index],y.iloc[test_index]
  dt.fit(x1_train,y1_train)
  pred = dt.predict(x1_test)
  score = accuracy_score(pred,y1_test)
  accuracy.append(score)
print(accuracy)

arr = np.array(accuracy)

np.mean(arr)
```

Hyper Parameter Tuning the model to overcome Overfitting model.

Determining the parameters by plotting f1_score metrics.

1. Function to calculate f1_score.
2. Function to plot the f1_score that we have calculated.
3. Pass the parameter values in the model and call the functions.

```
def cal_score(model, x1,y1,x2,y2):
  model.fit(x1,y1)
  p = model.predict(x1)
  f1 = f1_score(y1, p)
  p1 = model.predict(x2)
  f2 = f1_score(y2,p1)
  return f1,f2

def effect(train, test, x_axis, title):
  plt.figure(figsize = (12,10), dpi = 100)
  plt.plot(x_axis, train, color = 'red', label = 'train_score')
  plt.plot(x_axis, test, color = 'blue', label = 'test_score')
  plt.legend()
  plt.show()

max_depth = [i for i in range(1,50)]
train = []
test = []
for i in max_depth:
  model =DecisionTreeClassifier(max_depth=i, random_state=50)
  f1,f2 = cal_score(model, x_train, y_train, x_test, y_test)
  train.append(f1)
  test.append(f2)
effect(train,test, range(1,50), 'Max_Depth')

min_samples = [i for i in range(2,5000,25)]
train = []
```

```
test = []
for i in min_samples:
  model =DecisionTreeClassifier(max_depth=20, random_state=50, min_samples_split=i)
  f1,f2 = cal_score(model, x_train, y_train, x_test, y_test)
  train.append(f1)
  test.append(f2)
effect(train,test, range(2,5000,25), 'Min_Samples_Split')

max_leaf = [i for i in range(2,200,10)]
train = []
test = []
for i in max_leaf:
  model      =DecisionTreeClassifier(max_depth=20,min_samples_split=4250,max_leaf_nodes=i,
random_state=50)
  f1,f2 = cal_score(model, x_train, y_train, x_test, y_test)
  train.append(f1)
  test.append(f2)
effect(train,test, range(2,200,10), 'Max_Leaf_Nodes')

### Hyper Parameter Tuning the model by using roc_auc_curve.


def cal_score1(model, x1,y1,x2,y2):
  model.fit(x1,y1)
  p = model.predict(x1)
  false_positive_rate, true_positive_rate, thresholds = roc_curve(y1, p)
  roc_auc_1 = auc(false_positive_rate, true_positive_rate)
  p1 = model.predict(x2)
  false_positive_rate, true_positive_rate, thresholds = roc_curve(y2, p1)
  roc_auc_2 = auc(false_positive_rate, true_positive_rate)
  return roc_auc_1,roc_auc_2

def effect1(train, test, x_axis, title):
  plt.figure(figsize = (12,10), dpi = 100)
  plt.plot(x_axis, train, color = 'red', label = 'train_score')
  plt.plot(x_axis, test, color = 'blue', label = 'test_score')
  plt.legend()
  plt.show()

max_depth = [i for i in range(1,100)]
train = []
test = []
for i in max_depth:
  roc_auc_model =DecisionTreeClassifier(max_depth=i, random_state=50)
  roc_auc_1,roc_auc_2 = cal_score1(roc_auc_model, x_train, y_train, x_test, y_test)
  train.append(roc_auc_1)
  test.append(roc_auc_2)
effect1(train,test, range(1,100), 'Max_Depth')

min_sample_leaff = [i for i in range(25,4000,25)]
train = []
test = []
for i in min_sample_leaff:
  roc_auc_model              =DecisionTreeClassifier(max_depth=20,              min_samples_leaf=i,
random_state=50)
  roc_auc_1,roc_auc_2 = cal_score1(roc_auc_model, x_train, y_train, x_test, y_test)
```

```
  train.append(roc_auc_1)
  test.append(roc_auc_2)
effect1(train,test, range(25,4000,25), 'Min_Samples_Leaf')

max_leaf_node = [i for i in range(2,200,10)]
train = []
test = []
for i in max_leaf_node:
  roc_auc_model =DecisionTreeClassifier(max_depth=20,max_leaf_nodes=i, min_samples_leaf=3700, random_state=50)
  roc_auc_1,roc_auc_2 = cal_score1(roc_auc_model, x_train, y_train, x_test, y_test)
  train.append(roc_auc_1)
  test.append(roc_auc_2)
effect1(train,test, range(2,200,10), 'Max_Leaf_Nodes')
```

Hyper parameter Tuning the model using ccp(cost complexity pruning)
which helps us to select the best values for max_depth and max_samples_leaf parameter for Decision Tree.

```
path = dt.cost_complexity_pruning_path(x_train_ss,y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

ccp_alphas

clfs = []
for i in ccp_alphas:
  dt = DecisionTreeClassifier(random_state = 0, ccp_alpha=i)
  dt.fit(x_train_ss,y_train)
  clfs.append(dt)
  print('Number of Nodes in the Last Tree is: {} with ccp_alpha: {}'.format(clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

Plotting a graph with Respect to Accuracy score and various clfs(classifiers)

```
train_set = [dt.score(x_train_ss,y_train) for dt in clfs]
test_set = [dt.score(x_test_ss,y_test) for dt in clfs]

plt.figure(figsize = (12,10), dpi = 100)
fig,ax = plt.subplots()
ax.plot(ccp_alphas, train_set, marker = 'o', label = 'Train', drawstyle = 'steps-post')
ax.plot(ccp_alphas, test_set, marker = 'o', label = 'Test', drawstyle = 'steps-post')
ax.set_xlabel('ccp_alphas')
ax.set_ylabel('Accuracy')
ax.set_title("Accuracy and ccp_alphas Distribution")
ax.legend()
plt.show()
```

So, After applying Hyper Parameter Tuning **With Respect to Evaluation Metrics**, our model has successfully overcomed the problem of overfitting which has occured earlier.

```
modified_model = DecisionTreeClassifier(max_depth = 18, min_samples_split=4250, min_samples_leaf=3700, max_leaf_nodes=21)
modified_model.fit(x_train_ss, y_train)
pr = modified_model.predict(x_test_ss)
```

```
print(modified_model.score(x_train_ss,y_train))
print(modified_model.score(x_test_ss, y_test))
print(accuracy_score(pr,y_test))

### Tree Plot With Respect to Modified Model.

plt.figure(figsize = (15,10))
tree.plot_tree(modified_model, filled = True)



### **Evaluating Tuned Model on Test Data.**

hash = modified_model.predict(lucas)

print(accuracy_score(hash,resampled_y))

print(classification_report(hash,resampled_y))

print(confusion_matrix(hash,resampled_y))

print(precision_score(hash, resampled_y))

print(recall_score(hash, resampled_y))

print(f1_score(hash, resampled_y))



### Verifying With Respect to ccp_alpha value.

pathh = dt.cost_complexity_pruning_path(x_train_ss,y_train)
ccp_alphass, impurities = path.ccp_alphas, path.impurities

clfss = []
for i in ccp_alphass:
  dt = DecisionTreeClassifier(max_depth = 18, min_samples_split=4250, min_samples_leaf=3700, max_leaf_nodes=21, random_state = 0, ccp_alpha=i)
  dt.fit(x_train_ss,y_train)
  clfss.append(dt)
print('Number of Nodes in the Last Tree is: {} with ccp_alpha: {}'.format(clfs[-1].tree_.node_count, ccp_alphas[-1]))

train_sett = [dt.score(x_train_ss,y_train) for dt in clfss]
test_sett = [dt.score(lucas,resampled_y) for dt in clfss]

plt.figure(figsize = (12,10), dpi = 100)
fig,ax = plt.subplots()
ax.plot(ccp_alphass, train_sett, marker = '*', label = 'Train', drawstyle = 'steps-post')
ax.plot(ccp_alphass, test_sett, marker = '*', label = 'Test', drawstyle = 'steps-post')
ax.set_xlabel('ccp_alphas')
ax.set_ylabel('Accuracy')
ax.set_title("Accuracy and ccp_alphas Distribution")
ax.legend()
plt.show()
```

```
mod_model_ccp = DecisionTreeClassifier(random_state = 0, ccp_alpha = 0.04)
mod_model_ccp.fit(x_train_ss,y_train)

print(mod_model_ccp.score(x_train_ss,y_train))
print(mod_model_ccp.score(x_test_ss, y_test))

predicate = mod_model_ccp.predict(lucas)

print(accuracy_score(predicate,resampled_y))

print(precision_score(predicate, resampled_y))

print(recall_score(predicate, resampled_y))

print(f1_score(predicate, resampled_y))
```

```
### Tree plot with respect to ccp_modified_model

plt.figure(figsize = (12,10))
tree.plot_tree(mod_model_ccp, filled=True)

## **Logistic Regression**

from sklearn.linear_model import LogisticRegression
lg = LogisticRegression()
lg.fit(x_train_ss, y_train)
lg_pred = lg.predict(x_test_ss)
predicted_values = lg.predict_proba(x_test_ss)

print('Training Accuracy:', lg.score(x_train_ss,y_train))
print('Test Accuracy:', lg.score(x_test_ss,y_test))

recall_score(y_test, lg_pred)

precision_score(y_test,lg_pred)

f1_score(y_test,lg_pred)

y_testt = y_test.squeeze()

precision_points, recall_points, threshold_points = precision_recall_curve(y_testt, predicted_values[:,1])

precision_points.shape, recall_points.shape, threshold_points.shape

precision_points

recall_points
```

```
threshold_points

plt.figure(figsize = (12,10), dpi = 100)
plt.plot(threshold_points, recall_points[:-1], color = 'red')
plt.plot(threshold_points, precision_points[:-1], color = 'blue')
plt.show()

### Feature Importance

lg.coef_

f_imp = lg.coef_[0]
print(f_imp)
for i,v in enumerate(f_imp):
   print('Feature: %0d, Score: %.5f' % (i,v))

plt.figure(figsize =(8,6), dpi = 100)
plt.bar([i for i in range(len(f_imp))], f_imp)
plt.xlabel('len(f_imp)')
plt.ylabel('f_importances')
plt.title('LR Feature Importances')
plt.show()

### Evaluating LogisticRegresssion using roc_auc_score metric.

tpr,fpr, threshold = roc_curve(y_testt, predicted_values[:,1])
tpr.shape, fpr.shape, threshold.shape

plt.figure(figsize = (12,10), dpi = 100)
plt.plot(tpr,fpr, color = 'red')
plt.plot([0,1],[0,1], color = 'blue')
plt.title("roc_curve")
plt.show()

print(roc_auc_score(y_test, predicted_values[:,1]))

print("Training Accuracy ", lg.score(x_train_ss,y_train))
print("Testing Accuracy ", lg.score(x_test_ss,y_test))

print(classification_report(lg_pred, y_test))

print(confusion_matrix(lg_pred, y_test))

print(accuracy_score(lg_pred,y_test))

### Performing Cross Validating LR Model.

x = pd.DataFrame(data = x_train_ss, columns = inde_vars.columns)
y = y_train
from sklearn.model_selection import  StratifiedKFold
accuracy1 = []
skf = StratifiedKFold(n_splits = 10, random_state = None)
skf.get_n_splits(x,y)
for train_index, test_index in skf.split(x,y):
  print('Train:', train_index, 'Validation',test_index)
  x1_train,x1_test = x.iloc[train_index],x.iloc[test_index]
```

```
  y1_train,y1_test = y.iloc[train_index],y.iloc[test_index]
  lg.fit(x1_train,y1_train)
  pred = lg.predict(x1_test)
  score = accuracy_score(pred,y1_test)
  accuracy1.append(score)
print(accuracy1)

### Hyper Parameter Tuning Logistic regression model, using RandomizedSearchCV tool.

lo = LogisticRegression()


parameters = {'penalty':['l1','l2','elasticnet','none'],
              'solver':['newton-cg','lbfgs','sag','saga'],
              'max_iter':[i for i in range(100,2000,100)],
              'warm_start':['True','False']}


print(parameters)

lg_tuned_model = RandomizedSearchCV(estimator=lo, param_distributions = parameters, scoring='accuracy', n_jobs = -1, cv = 10, n_iter = 10, verbose = 2, random_state = 50)

lg_tuned_model.fit(x_train_ss,y_train)

lg_tuned_model.best_params_

lg_tuned_model.get_params

lg_tuned_model.best_score_

### Testing the Accuracy using Tuned LR Model.

lr = LogisticRegression(max_iter = 1300,
                        penalty='l2',
                        solver= 'newton-cg',
                        warm_start=True)
lr.fit(x_train_ss,y_train)

tuned_pred = lr.predict(x_test_ss)
print(accuracy_score(tuned_pred,y_test))

### **Evaluating Tuned Model on Test Data.**

print('Tuned Training Accuracy:', lr.score(x_train_ss, y_train))

jim = lr.predict(lucas)
print(accuracy_score(jim, resampled_y))

print(roc_auc_score(jim, resampled_y))

print(precision_score(jim, resampled_y))

print(recall_score(jim, resampled_y))

print(f1_score(jim, resampled_y))
```

```
###### Results of LR Model without Tuning.

pam = lg.predict(lucas)
print(accuracy_score(pam, resampled_y))

print(precision_score(pam, resampled_y))

print(recall_score(pam, resampled_y))

print(f1_score(pam, resampled_y))

## **Random Forest Classifier**

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train_ss, y_train)
rf_pred = rf.predict(x_test_ss)

print("Training accuracy is", rf.score(x_train_ss, y_train))
print('Testing Accuracy is', rf.score(x_test_ss, y_test))
print(accuracy_score(y_test, rf_pred))

print(confusion_matrix(y_test, rf_pred))

print(classification_report(y_test, rf_pred))

recall_score(y_test, rf_pred)

precision_score(y_test, rf_pred)

f1_score(y_test, rf_pred)

### Feature Importance

rf.feature_importances_

fe_imp = rf.feature_importances_
for i,v in enumerate(fe_imp):
  print('Feature: %0d, Score: %.5f' % (i,v))

plt.figure(figsize =(8,6), dpi = 100)
plt.bar([i for i in range(len(fe_imp))], fe_imp)
plt.xlabel('len(f_imp)')
plt.ylabel('f_importances')
plt.title('RF Feature Importances')
plt.show()

### Performing Cross Validation on RFC Model.

x = pd.DataFrame(data = x_train_ss, columns = inde_vars.columns)
y = y_train
from sklearn.model_selection import  StratifiedKFold
accuracy2 = []
skf = StratifiedKFold(n_splits = 10, random_state = None)
skf.get_n_splits(x,y)
```

```
for train_index, test_index in skf.split(x,y):
  print('Train:', train_index, 'Validation',test_index)
  x1_train,x1_test = x.iloc[train_index],x.iloc[test_index]
  y1_train,y1_test = y.iloc[train_index],y.iloc[test_index]
  rf.fit(x1_train,y1_train)
  pred = rf.predict(x1_test)
  score = accuracy_score(pred,y1_test)
  accuracy2.append(score)
print(accuracy2)
```

Hyper Parameter Tuning Random Forest Model Using RandomizedSearchCV.

```
rfc = RandomForestClassifier()
#rfc

param = {'n_estimators' : [i for i in range(100,1500,100)],
        'max_depth' : [i for i in range(10,100,10)],
        'max_features' : ['auto','sqrt','log2'],
        'min_samples_split' :  np.linspace(0.1,1.0,10, endpoint = True),
        'min_samples_leaf' : np.linspace(0.1,0.5,5, endpoint =True),
        'warm_start' : ['True', 'False']
}

#param

rf_tuned_model = RandomizedSearchCV(estimator =rfc, param_distributions=param, scoring =
'roc_auc', verbose = 2, n_jobs = -1, random_state = 50)

rf_tuned_model.fit(x_train_ss,y_train)

rf_tuned_model.best_score_

rf_tuned_model.get_params

rf_tuned_model.best_estimator_
```

Before Tuning the Random Forest Model.

```
dwight = rf.predict(lucas)
print(accuracy_score(dwight, resampled_y))

print('Testing Accuracy:', rf.score(lucas, resampled_y))

print(precision_score(dwight, resampled_y))

print(recall_score(resampled_y, dwight))

print(f1_score(resampled_y, dwight))
```

Evaluating Tuned Model on Test Data.

```
kite = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                    criterion='gini', max_depth=40, max_features='sqrt',
                    max_leaf_nodes=None, max_samples=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=0.1, min_samples_split=0.1,
```

```
                min_weight_fraction_leaf=0.0, n_estimators=100,
                n_jobs=None, oob_score=False, random_state=None,
                verbose=0, warm_start='True')

kite.fit(x_train_ss,y_train)

print('Tuned Training Score:', kite.score(x_train_ss, y_train))

print('Testing Accuracy:', kite.score(lucas, resampled_y))

lion = kite.predict(lucas)
print(accuracy_score(lion, resampled_y))

print(recall_score(resampled_y, lion))

print(precision_score(resampled_y, lion))

print(f1_score(resampled_y, lion))
```

OUTPUT SCREENS:

1.Importing Libraries.

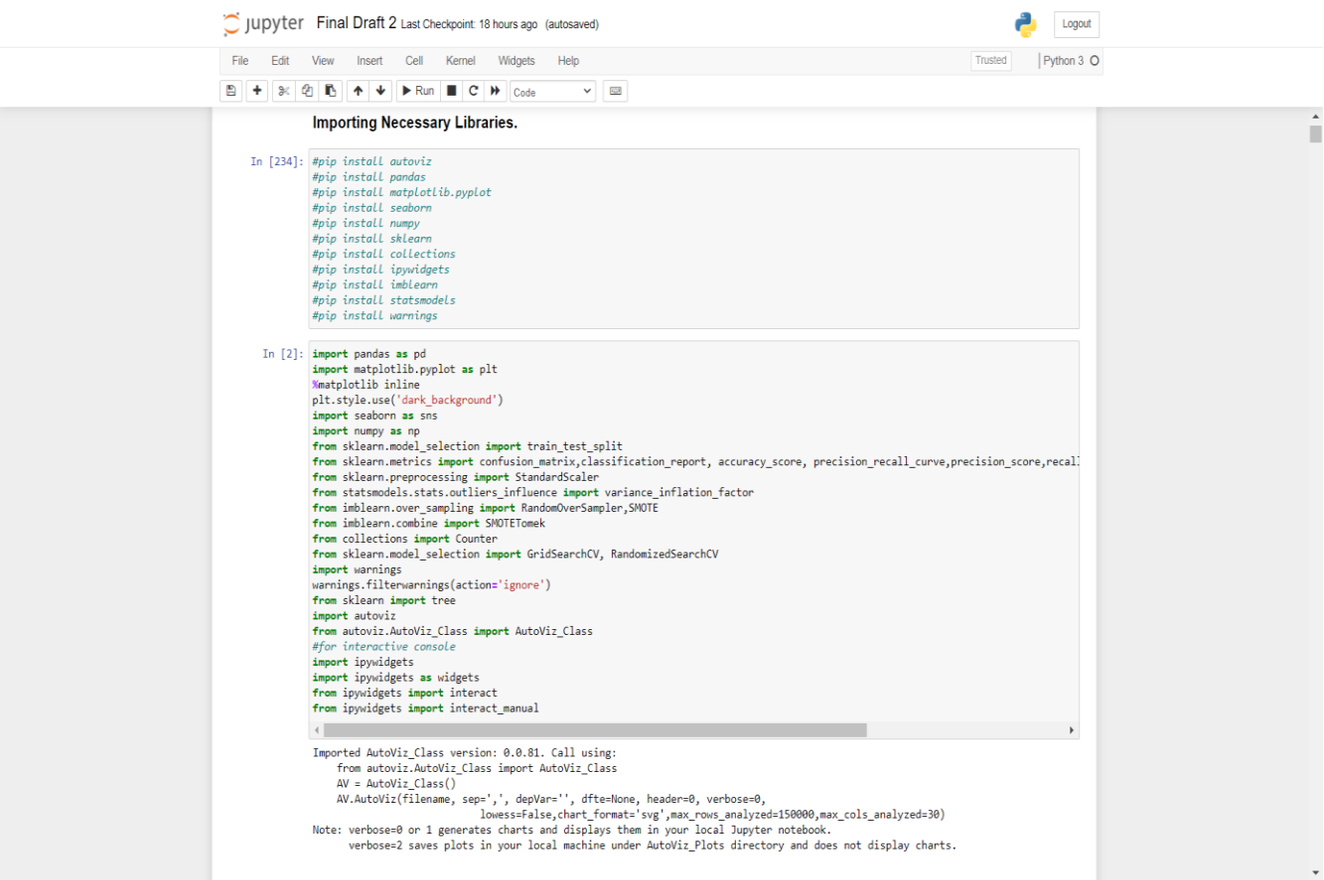
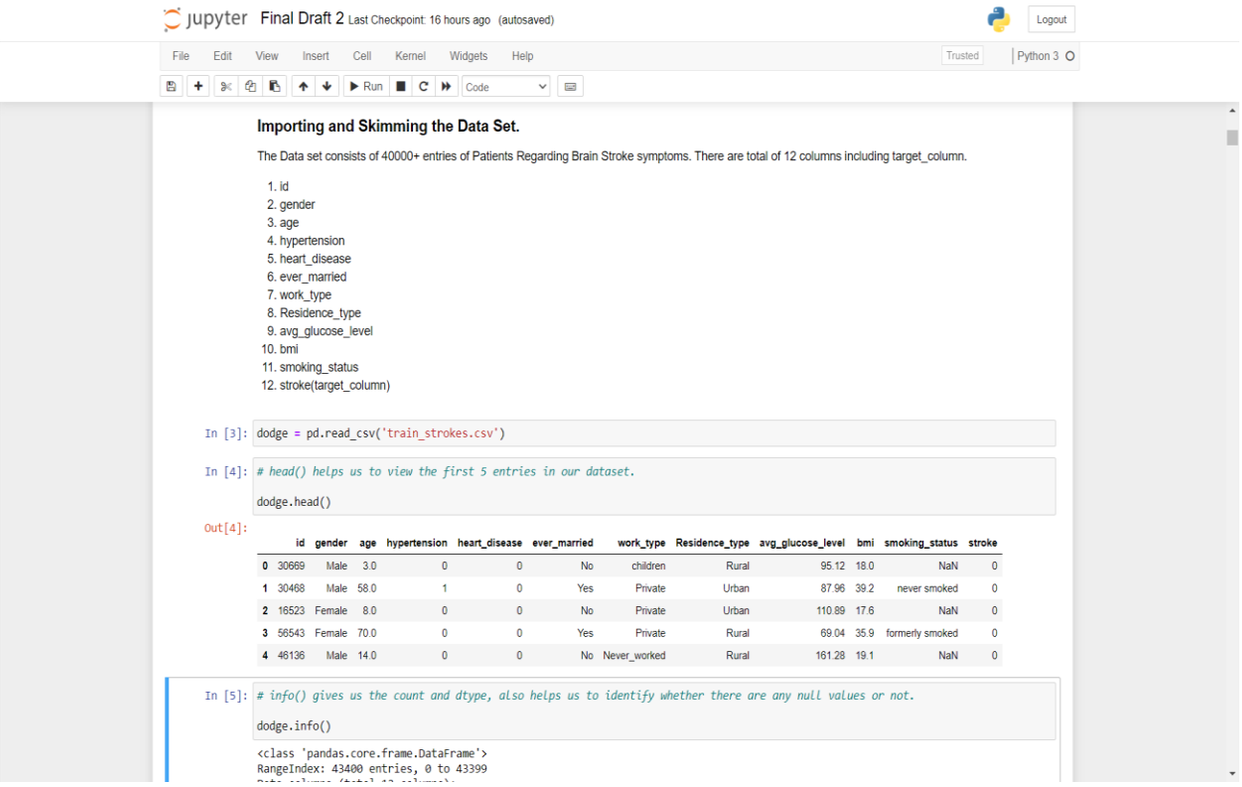


Fig 2.1

2.Importing and Skimming of Data set.



jupyter Final Draft 2 Last Checkpoint: 16 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Importing and Skimming the Data Set.

The Data set consists of 40000+ entries of Patients Regarding Brain Stroke symptoms. There are total of 12 columns including target_column.

1. id
2. gender
3. age
4. hypertension
5. heart_disease
6. ever_married
7. work_type
8. Residence_type
9. avg_glucose_level
10. bmi
11. smoking_status
12. stroke(target_column)

In [3]:
```
dodge = pd.read_csv('train_strokes.csv')
```

In [4]:
```
# head() helps us to view the first 5 entries in our dataset.

dodge.head()
```

Out[4]:

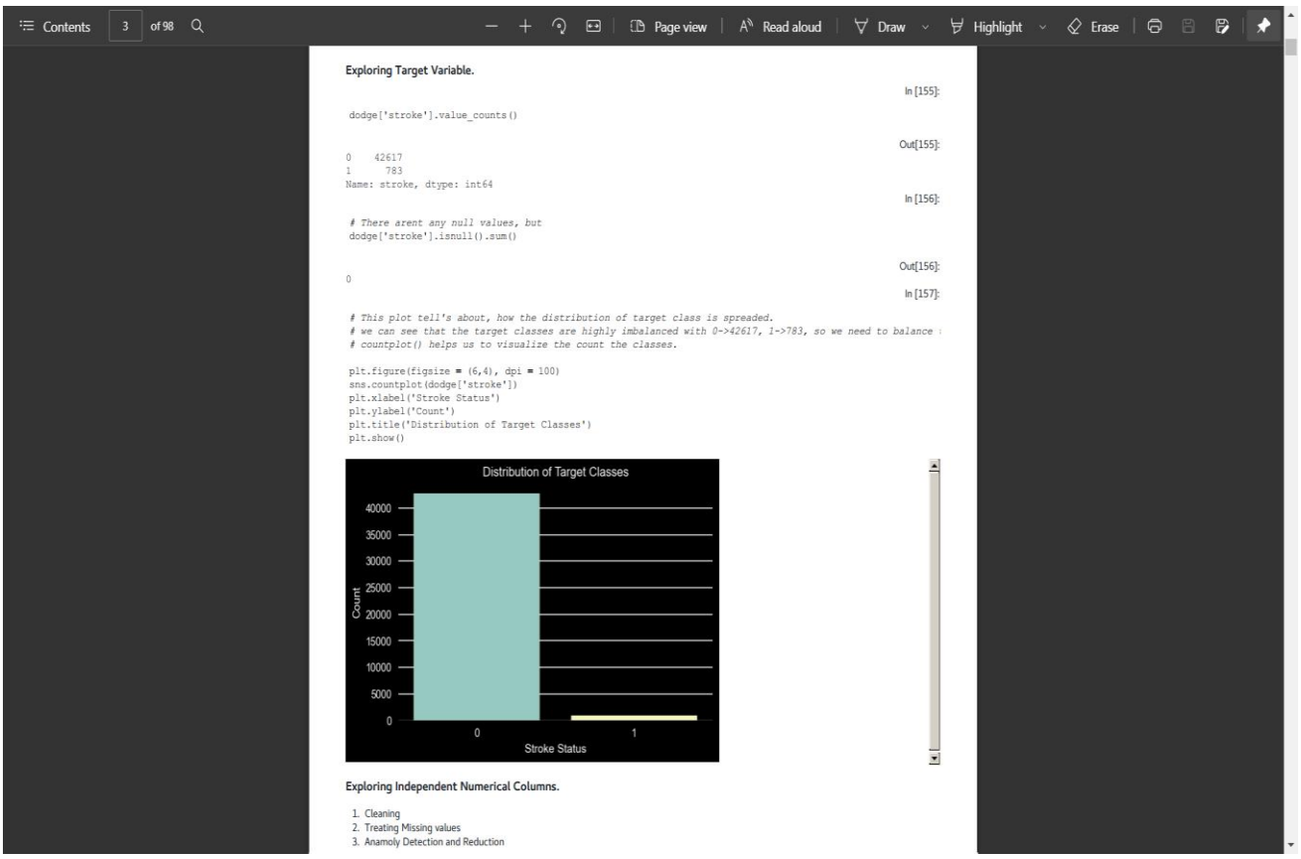
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	30669	Male	3.0	0	0	No	children	Rural	95.12	18.0	NaN	0
1	30468	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	16523	Female	8.0	0	0	No	Private	Urban	110.89	17.6	NaN	0
3	56543	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	46136	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	NaN	0

In [5]:
```
# info() gives us the count and dtype, also helps us to identify whether there are any null values or not.

dodge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43400 entries, 0 to 43399
```

Fig 2.2

3.Exploring Target Variable.



Contents 3 of 98 Page view Read aloud Draw Highlight Erase

Exploring Target Variable.

In [155]:
```
dodge['stroke'].value_counts()
```

Out[155]:
```
0    42617
1      783
Name: stroke, dtype: int64
```

In [156]:
```
# There arent any null values, but
dodge['stroke'].isnull().sum()
```

Out[156]:
```
0
```

In [157]:
```
# This plot tell's about, how the distribution of target class is spreaded.
# we can see that the target classes are highly imbalanced with 0->42617, 1->783, so we need to balance
# countplot() helps us to visualize the count the classes.

plt.figure(figsize = (6,4), dpi = 100)
sns.countplot(dodge['stroke'])
plt.xlabel('Stroke Status')
plt.ylabel('Count')
plt.title('Distribution of Target Classes')
plt.show()
```


Exploring Independent Numerical Columns.

1. Cleaning
2. Treating Missing values
3. Anamoly Detection and Reduction

Fig 2.3

4.Exploring Independent Columns.

Exploring Independent Numerical Columns.

1. Cleaning
2. Treating Missing values
3. Anamoly Detection and Reduction

```
In [258]: numerical = ['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi']
          #dodge[numerical[0]]
```

Treating missing values present in the column dodge['bmi'], no other numerical columns has missing values.

```
In [259]: dodge['bmi'].isnull().sum()

Out[259]: 1462

In [260]: dodge['bmi'] = dodge['bmi'].fillna(dodge['bmi'].mean())

In [261]: dodge['bmi'].isnull().sum()

Out[261]: 0
```

Exploring each numerical column using describe()

```
In [262]: for i in numerical:
            print(dodge[i].describe())

count    43400.000000
mean        42.217894
std         22.519649
min          0.080000
25%         24.000000
50%         44.000000
75%         60.000000
max         82.000000
Name: age, dtype: float64
count    43400.000000
mean         0.093571
std          0.291235
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: hypertension, dtype: float64
count    43400.000000
mean         0.047512
std          0.212733
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: heart_disease, dtype: float64
count    43400.000000
mean       104.482750
std         43.111751
min         55.000000
25%         77.540000
50%         91.580000
75%        112.070000
max        291.050000
Name: avg_glucose_level, dtype: float64
count    43400.000000
mean        28.605038
std          7.638023
min         10.100000
25%         23.400000
50%         28.100000
75%         32.600000
max         97.600000
Name: bmi, dtype: float64
```

Fig 2.4

5.Anomaly Detection and Reduction

jupyter Final Draft 2 Last Checkpoint: 16 hours ago (autosaved)

Function to check the Anamolies in the column using upper_limit and lower_limit.

1. If the upper_limit > max(df['col']), then we replace the upper_limit with the max value.
2. Similarly, if the lower_limit < min(df['col']), we replace the lower_limit with the min value.

```
In [18]: anamolies = []
         def outliers(data):
           random_state_mean = np.mean(data)
           random_state_std = np.std(data)
           anamoly = random_state_std * 3

           upper_limit = random_state_mean + anamoly
           lower_limit = random_state_mean - anamoly
           lp_lower_limit = 1.00
           up_upper_limit = max(dodge['age'])
           print(upper_limit)
           print(lower_limit)

           print(lp_lower_limit)
           print(up_upper_limit)

           for i in data:
               if i < lp_lower_limit or i > up_upper_limit:
                   anamolies.append(i)

In [19]: outliers(dodge['age'])
         print(len(anamolies))

         109.77606171737218
         -25.348273608938617
         1.0
         82.0
         496

In [20]: dodge.shape

Out[20]: (43400, 12)
```

Here all the values below 1 are termed as outliers, although in rarest of cases Intrauterine stroke occur to unborn children in the womb.

But in this project we drop those values, but in future we can even work on these values.

```
In [21]: dodge[dodge['age'] < 1.00]

Out[21]:
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
116	7559	Female	0.64	0	0	No	children	Urban	83.62	24.9	NaN	0
129	22706	Female	0.88	0	0	No	children	Rural	88.11	15.5	NaN	0
323	61511	Female	0.32	0	0	No	children	Rural	73.71	16.2	NaN	0
746	54747	Male	0.88	0	0	No	children	Rural	157.57	19.2	NaN	0
761	53279	Male	0.24	0	0	No	children	Rural	118.87	16.3	NaN	0
...	...	...	...	...	...	...	...	...	...	...	...	...
42001	2898	Female	0.32	0	0	No	children	Urban	91.06	17.6	NaN	0
42106	51969	Male	0.32	0	0	No	children	Urban	90.38	16.1	NaN	0
43220	38654	Female	0.08	0	0	No	children	Rural	125.11	12.1	NaN	0
43296	32576	Male	0.56	0	0	No	children	Rural	78.07	21.9	NaN	0
43330	16834	Female	0.72	0	0	No	children	Urban	87.74	16.6	NaN	0

496 rows × 12 columns

```
In [22]: dodge[dodge['age'] < 1.00].index

Out[22]: Int64Index([ 116,  129,  323,  746,  761,  861,  975, 1087, 1375,
                     1389,
                     ...
                     42637, 42862, 42880, 42881, 42982, 43031, 43106, 43220, 43296,
                     43330],
                    dtype='int64', length=496)

In [23]: chevy = dodge.drop(index = dodge[dodge['age'] < 1.00].index, axis = 0, inplace=True)

In [24]: dodge.drop(index = dodge[(dodge.age > 1.0) & (dodge.age < 2.0)].index, axis = 0, inplace = True)

In [25]: dodge.shape

Out[25]: (42389, 12)
```

2. avg_glucose_level(Average Glucose Level)

```
In [26]: anamolies = []
         def outliers(data):
           random_state_mean = np.mean(data)
           random_state_std = np.std(data)
```

Fig 2.5

6.Exploring Independent Categorical Columns.

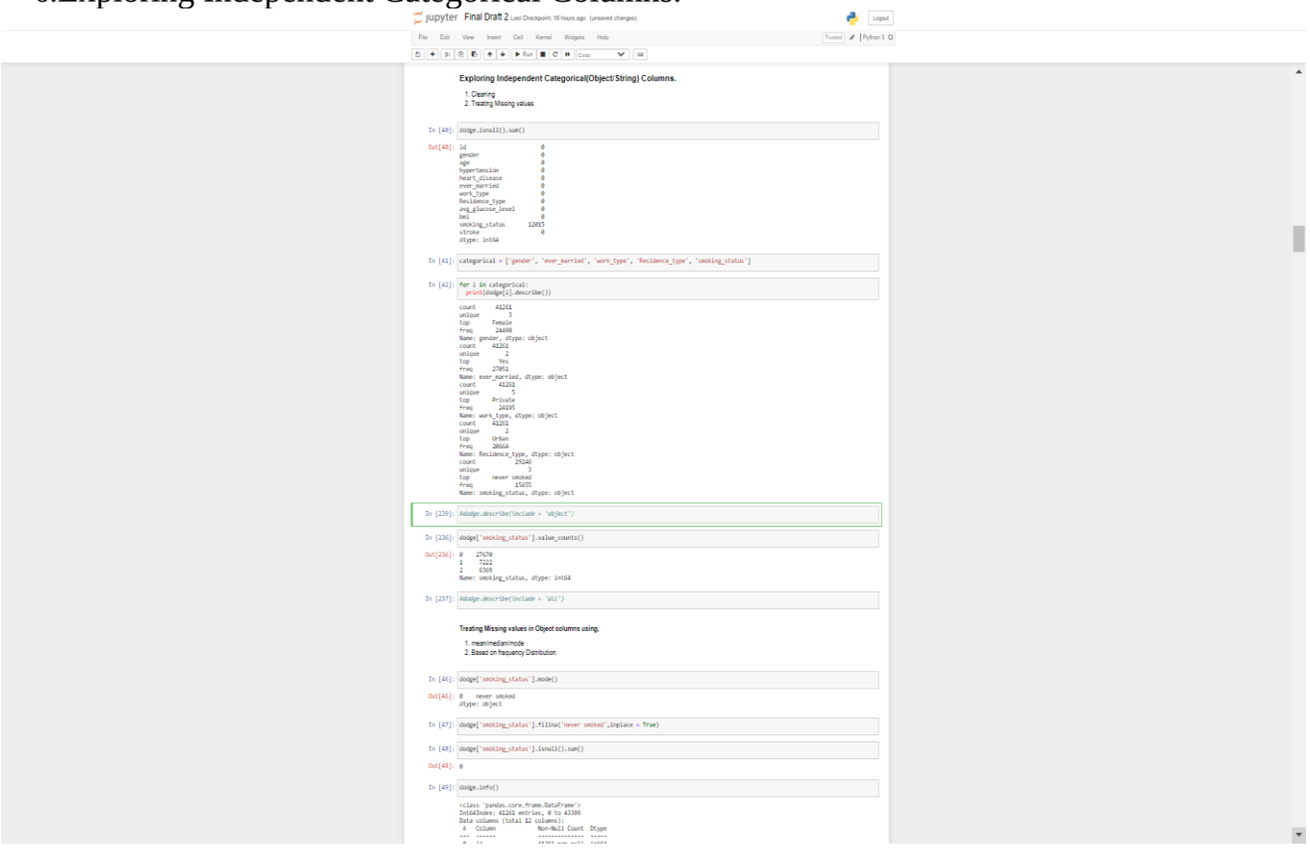


Fig 2.6

7.Exploratory Data Analysis.

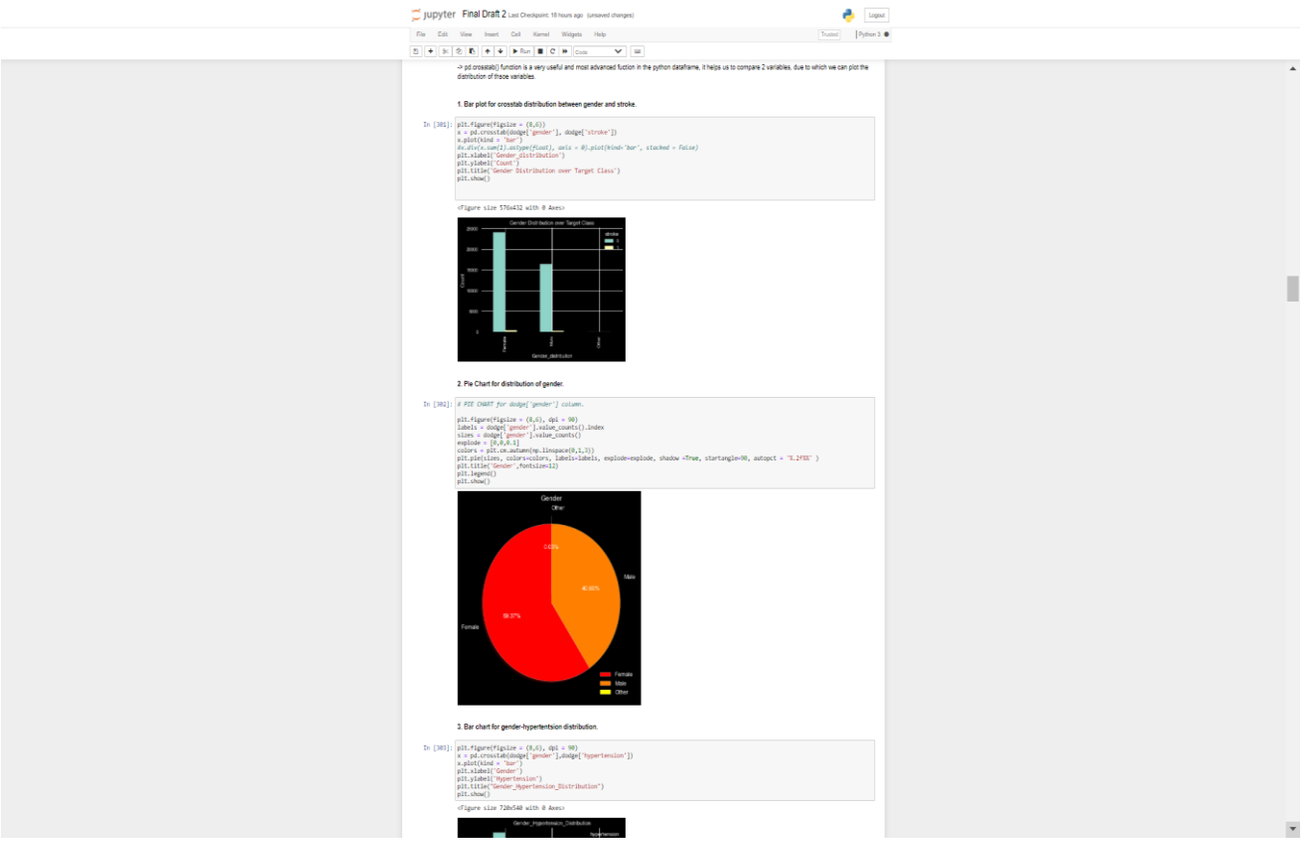


Fig 2.7

8.Exploratory Data Analysis 2

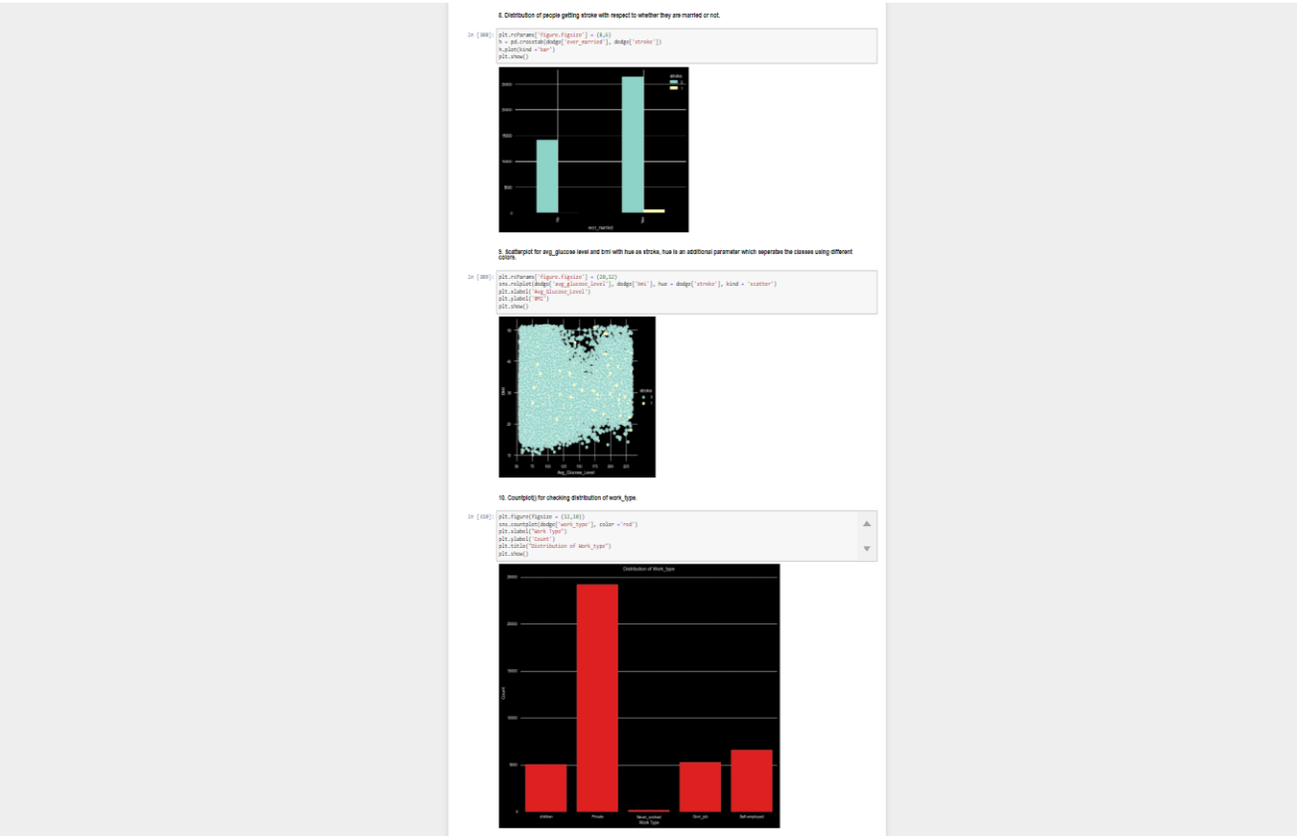
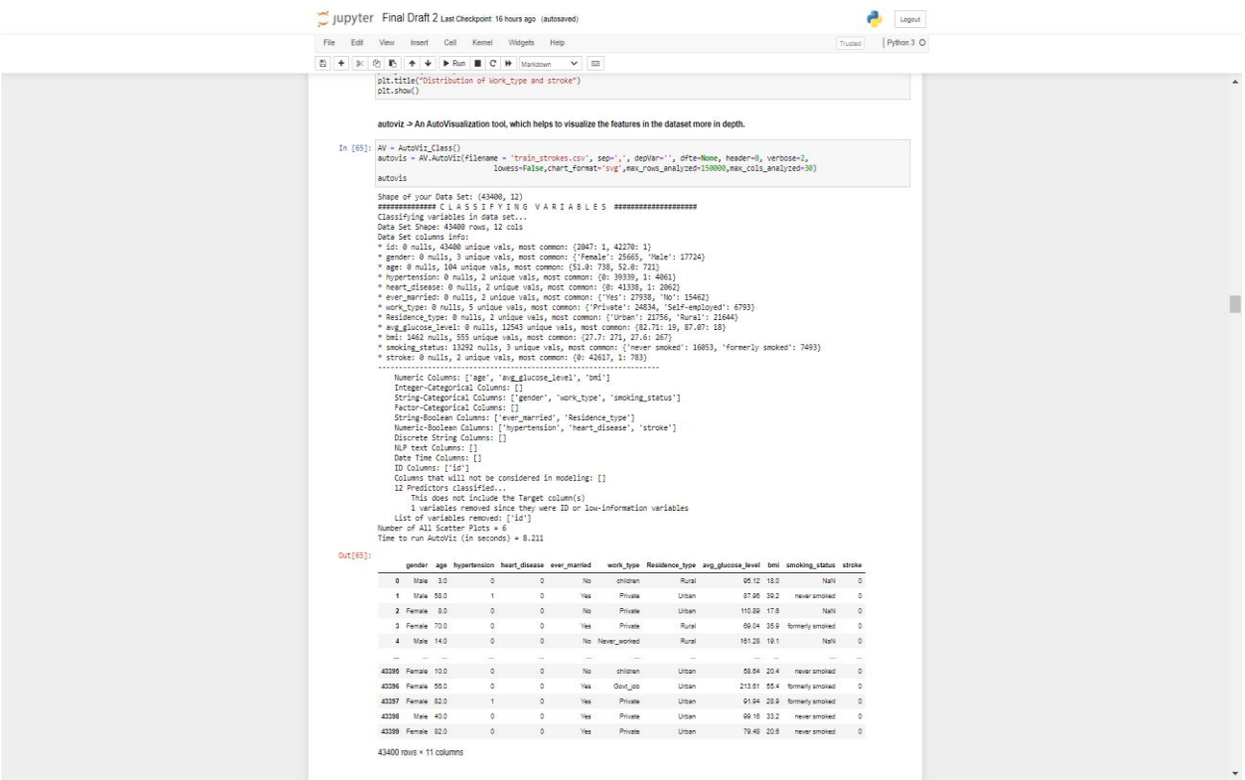


Fig 2.8

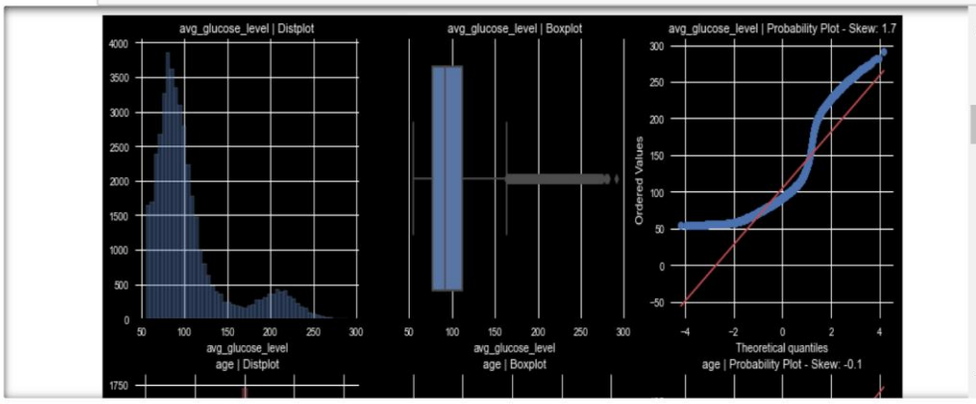
9.AutoVisualizing Tool.



10.Sample Auto Viz Charts.

autoviz -> An AutoVisualization tool, which helps to visualize the features in the dataset more in depth.

```
In [312]: AV = AutoViz_Class()
autovis = AV.AutoViz(filename = 'train_strokes.csv', sep=',', depVar='', dfte=None, header=0, verbose=2,
                     lowess=False,chart_format='svg',max_rows_analyzed=150000,max_cols_analyzed=30)
autovis
```



Exploring data using Traditional python code, with the help of interactive widgets.

```
In [313]: abg = dodge[['hypertension', 'heart_disease']].groupby(['hypertension']).count().style.background_gradient(cmap = 'viridis')
```

Sum of Heart Disease values with respect to hypertension, This can be easily eaxplained by crosstab()

```
In [314]: abg
```

Out[314]:

hypertension	heart_disease

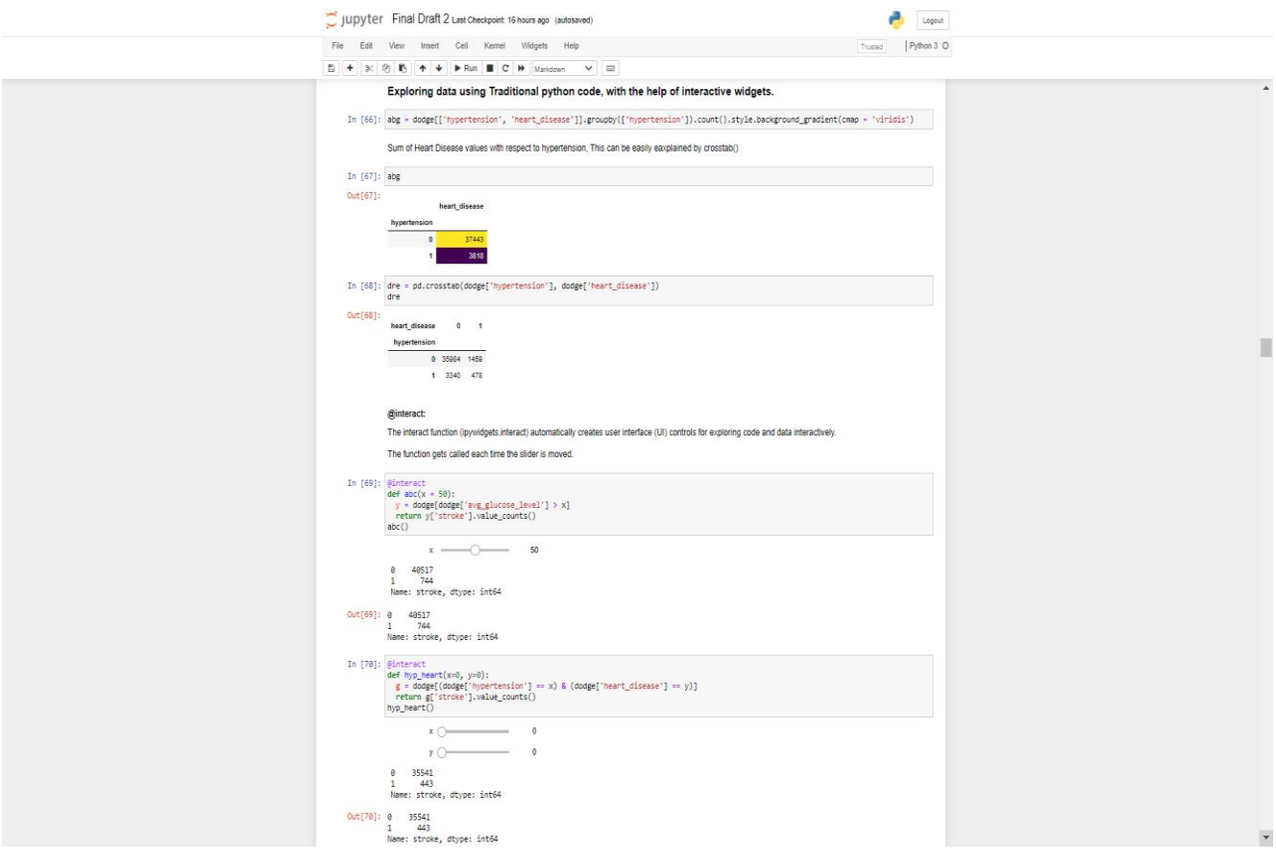


Fig 2.9 & Fig 2.10

11.Exploring Data Using Interact Manual.

Fig 2.11

12.Feature Transformation.

jupyter Final Draft 2 Last Checkpoint: 16 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Feature Transformation.

Feature Transformation is the technique of transforming the variable into other form like Strings -> Numeric, splitting the Date Column in to pieces etc.

Types of encoding.

1. Nominal Encoding.

- one hot encoding -> Creating Dummy variables.
- one hot encoding with multi categories(more than 20 categories)
- mean encoding

2. Ordinal Encoding.

- Label Encoder
- target_guided_encoding

-> For the columns with less than 5 categories we can manually perform encoding, usinf map().

-> For Columns with more than 20 Categories we can perform one hot encoding with multi categories, where we tend to select the top categories based on their value_counts().

In [72]:
```
dodge.head()
```

Out[72]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	3.0	0	0	No	children	Rural	95.12	18.0	never smoked	0
1	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	Female	8.0	0	0	No	Private	Urban	110.89	17.6	never smoked	0
3	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	never smoked	0

In [73]:
```
dodge['smoking_status'].unique()
```

Out[73]:
```
array(['never smoked', 'formerly smoked', 'smokes'], dtype=object)
```

In [74]:
```
mapping = {'Male':2, 'Female':1, 'Other':0}
mapping1 = {'No':0, 'Yes':1}
mapping2 = {'never smoked':0, 'formerly smoked':1, 'smokes':2}
```

In [75]:
```
dodge['gender'] = dodge['gender'].map(mapping)
```

In [76]:
```
dodge['ever_married'] = dodge['ever_married'].map(mapping1)
```

In [77]:
```
dodge['smoking_status'] = dodge['smoking_status'].map(mapping2)
```

In [78]:
```
dodge[['gender', 'smoking_status', 'ever_married']].head()
```

Out[78]:

	gender	smoking_status	ever_married
0	2	0	0
1	2	0	1
2	1	0	0
3	1	1	1
4	2	0	0

In [79]:
```
dodge['work_type'].unique()
```

Out[79]:
```
array(['children', 'Private', 'Never_worked', 'Govt_job', 'Self-employed'],
      dtype=object)
```

Fig 2.12

13.Feature Transformation 2.

jupyter Final Draft 2 Last Checkpoint: 16 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [79]:
```
dodge['work_type'].unique()
```

Out[79]:
```
array(['children', 'Private', 'Never_worked', 'Govt_job', 'Self-employed'],
      dtype=object)
```

In [80]:
```
dodge['Residence_type'].unique()
```

Out[80]:
```
array(['Rural', 'Urban'], dtype=object)
```

In [81]:
```
dodge['home_town'] = pd.get_dummies(dodge['Residence_type'], drop_first = True)
```

Creating a new dataframe with respect to work_type.

In [82]:
```
f150 = pd.get_dummies(dodge['work_type'], drop_first = True)
```

Merging 2 DataFrames(dodge,f150) with the default join.

In [83]:
```
camero = pd.concat([dodge,f150], axis = 1)
```

In [84]:
```
camero.head()
```

Out[84]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke	home_town	Neve
0	2	3.0	0	0	0	children	Rural	95.12	18.0	0	0	0	
1	2	58.0	1	0	1	Private	Urban	87.96	39.2	0	0	1	
2	1	8.0	0	0	0	Private	Urban	110.89	17.6	0	0	1	
3	1	70.0	0	0	1	Private	Rural	69.04	35.9	1	0	0	
4	2	14.0	0	0	0	Never_worked	Rural	161.28	19.1	0	0	0	

In [85]:
```
camero.rename(columns = {'Never_worked':'w_t_n_w', 'Private':'w_t_p', 'Self-employed':'w_t_s_e', 'children':'w_t_c'}, inplace = T
```

Droping the columns ['work_type', 'Residence_type'], as we have already created dummy variables for them.

In [86]:
```
camero.drop(columns = ['work_type','Residence_type'], inplace = True)
```

In [87]:
```
camero.head()
```

Out[87]:

	gender	age	hypertension	heart_disease	ever_married	avg_glucose_level	bmi	smoking_status	stroke	home_town	w_t_n_w	w_t_p	w_t_s_e	w_t_c
0	2	3.0	0	0	0	95.12	18.0	0	0	0	0	0	0	1
1	2	58.0	1	0	1	87.96	39.2	0	0	1	0	1	0	0
2	1	8.0	0	0	0	110.89	17.6	0	0	1	0	1	0	0
3	1	70.0	0	0	1	69.04	35.9	1	0	0	0	1	0	0
4	2	14.0	0	0	0	161.28	19.1	0	0	0	1	0	0	0

In [88]:
```
camero.info()
```

Fig 2.13

14.Feature Scaling.

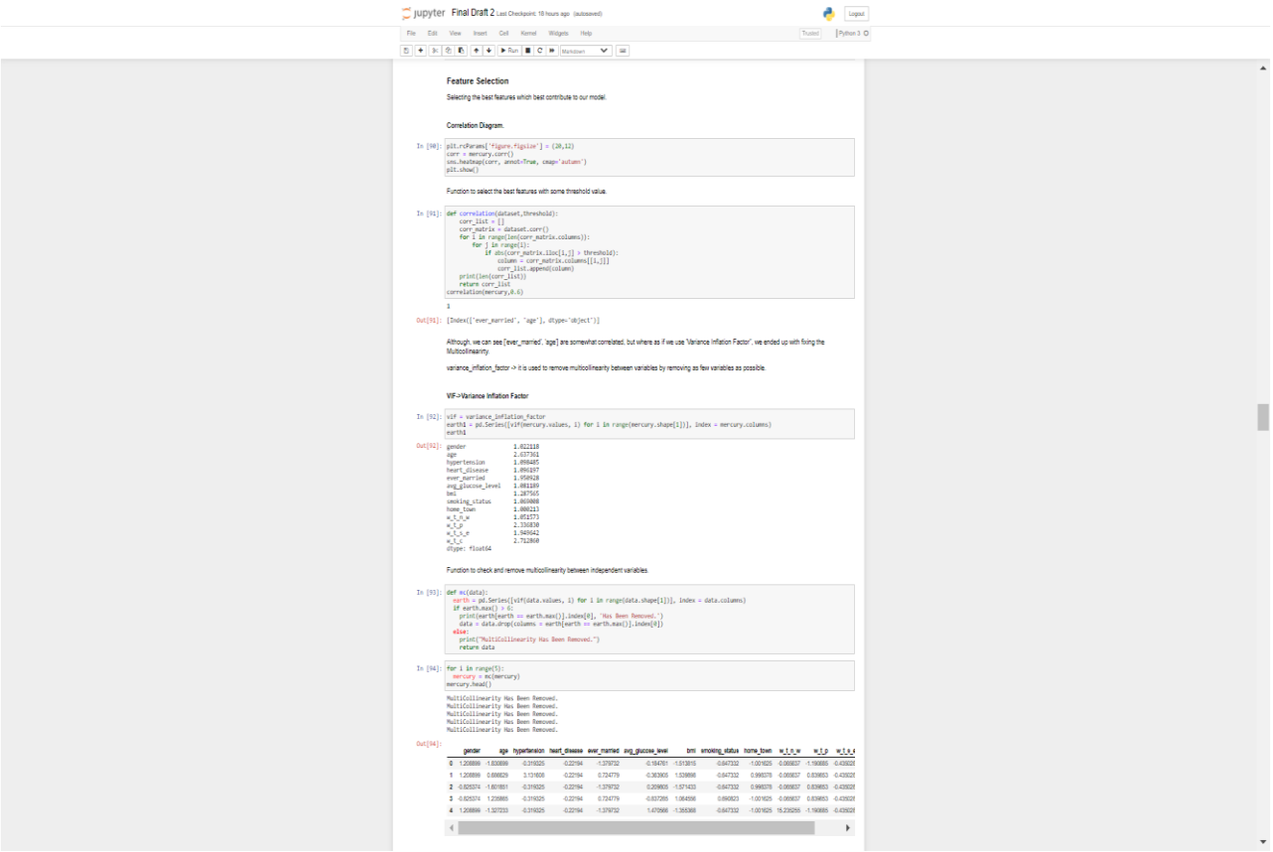


Fig 2.14

16.Handling Imbalanced Data.

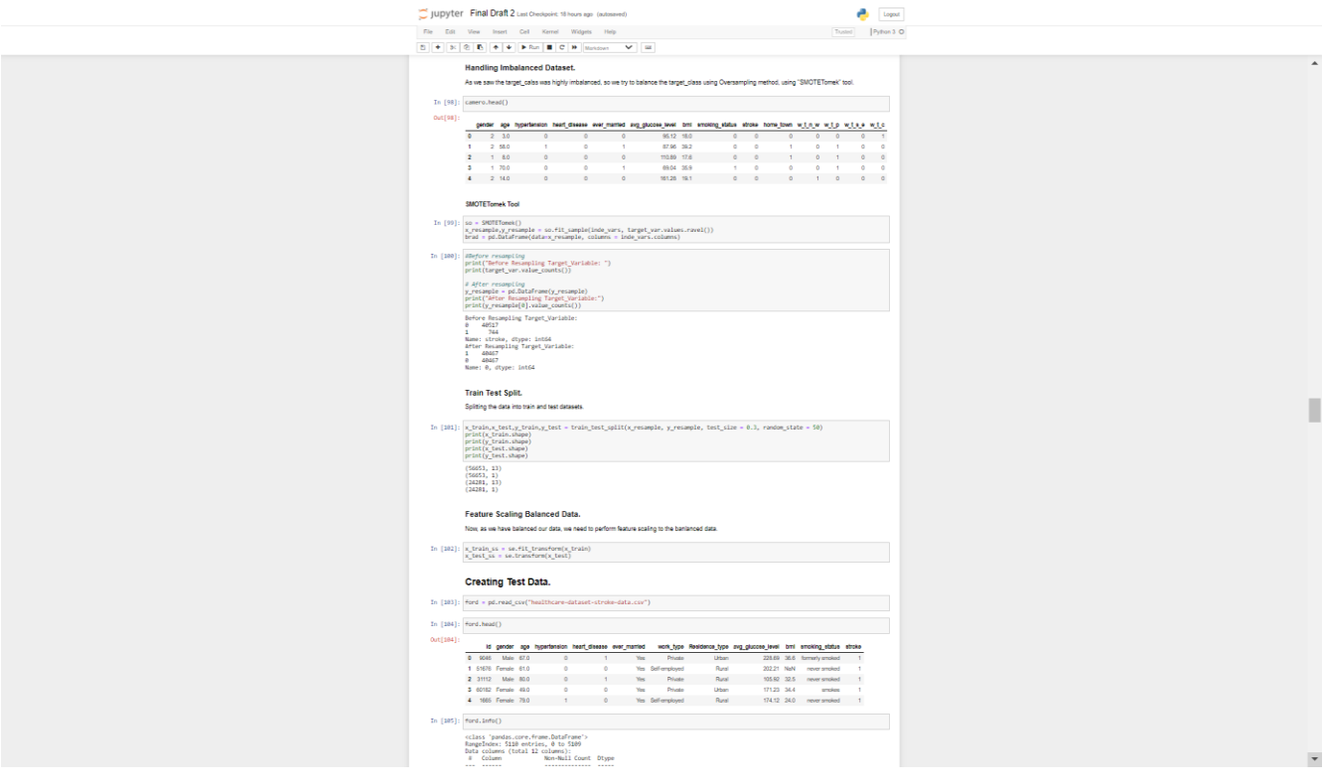
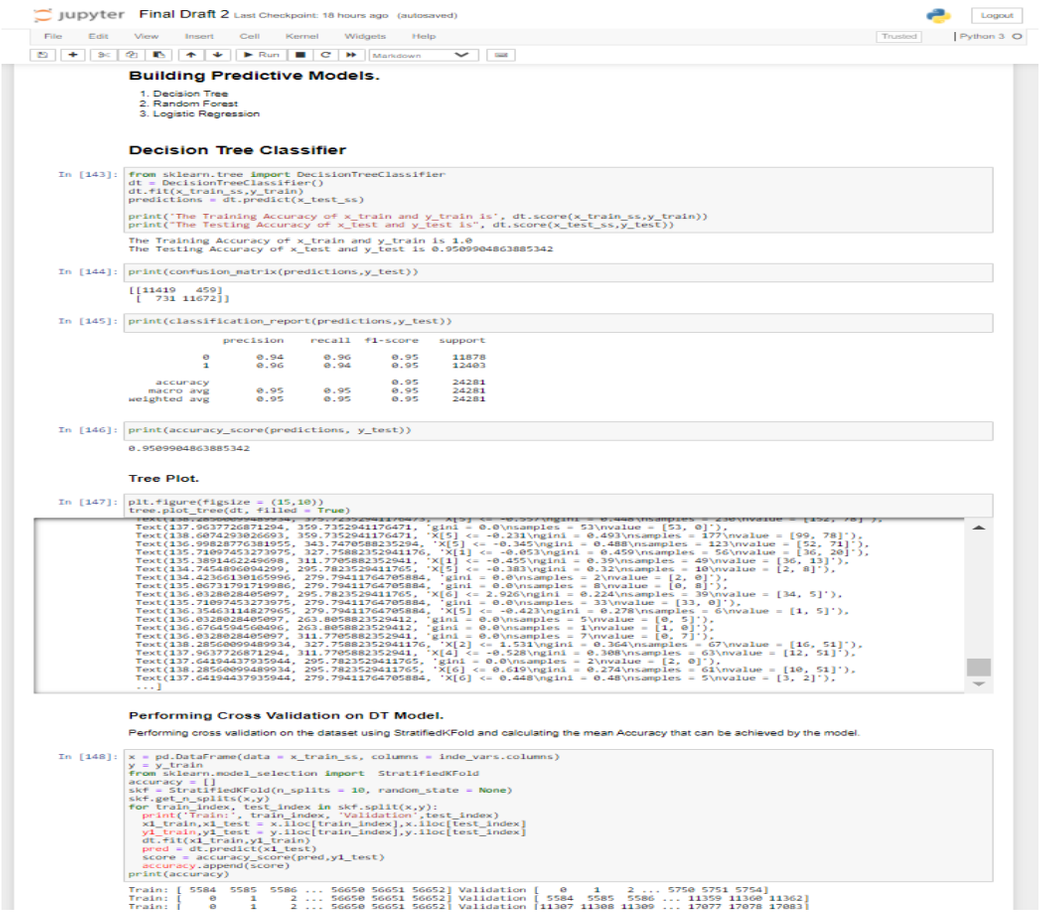


Fig 2.15

17.Building Predictive Models.



jupyter Final Draft 2 Last Checkpoint: 18 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Building Predictive Models.

1. Decision Tree
2. Random Forest
3. Logistic Regression

Decision Tree Classifier

In [143]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train_ss,y_train)
predictions = dt.predict(x_test_ss)

print('The Training Accuracy of x_train and y_train is', dt.score(x_train_ss,y_train))
print("The Testing Accuracy of x_test and y_test is", dt.score(x_test_ss,y_test))
```

```
The Training Accuracy of x_train and y_train is 1.0
The Testing Accuracy of x_test and y_test is 0.9509904863885342
```

In [144]:
```
print(confusion_matrix(predictions,y_test))
```

```
[[11419   459]
 [  731 11672]]
```

In [145]:
```
print(classification_report(predictions,y_test))
```

```
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     11878
           1       0.96      0.94      0.95     12403

    accuracy                           0.95     24281
   macro avg       0.95      0.95      0.95     24281
weighted avg       0.95      0.95      0.95     24281
```

In [146]:
```
print(accuracy_score(predictions, y_test))
```

```
0.9509904863885342
```

Tree Plot.

In [147]:
```
plt.figure(figsize = (15,10))
tree.plot_tree(dt, filled = True)
```

```
Text(137.963772687129 4, 359.7352941176471, 'gini = 0.0\nsamples = 53\nvalue = [53, 0]'),
Text(138.60742930266693, 359.7352941176471, 'X[5] <= -0.231\ngini = 0.493\nsamples = 177\nvalue = [99, 78]'),
Text(136.99828776381955, 343.7470588235294, 'X[5] <= -0.345\ngini = 0.488\nsamples = 123\nvalue = [52, 71]'),
Text(135.71097453273975, 327.7588235294117 6, 'X[1] <= -0.053\ngini = 0.459\nsamples = 56\nvalue = [36, 20]'),
Text(135.38914622496698, 311.7705882352941, 'X[1] <= -0.455\ngini = 0.39\nsamples = 49\nvalue = [36, 13]'),
Text(134.74548960042299, 295.7823529411765, 'X[5] <= -0.383\ngini = 0.32\nsamples = 10\nvalue = [2, 8]'),
Text(134.42366130165996, 279.7941176470588 4, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(135.06711791719986, 279.7941176470588 4, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
Text(136.03280284050097, 295.7823529411765, 'X[6] <= 2.926\ngini = 0.224\nsamples = 39\nvalue = [34, 5]'),
Text(135.71097453273975, 279.7941176470588 4, 'gini = 0.0\nsamples = 33\nvalue = [33, 0]'),
Text(136.35463114827965, 279.7941176470588 4, 'X[5] <= -0.423\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
Text(136.03280284050097, 263.8058823529412, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(136.67645956046 4, 263.8058823529412, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(136.03280284050097, 311.7705882352941, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(138.28560099489934, 327.7588235294117 6, 'X[2] <= 1.531\ngini = 0.364\nsamples = 67\nvalue = [16, 51]'),
Text(137.963772687129 4, 311.7705882352941, 'X[4] <= -0.528\ngini = 0.308\nsamples = 63\nvalue = [12, 51]'),
Text(137.64194437935944, 295.7823529411765, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(138.28560099489934, 295.7823529411765, 'X[6] <= 0.619\ngini = 0.274\nsamples = 61\nvalue = [10, 51]'),
Text(137.64194437935944, 279.7941176470588 4, 'X[6] <= 0.448\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
...]
```

Performing Cross Validation on DT Model.

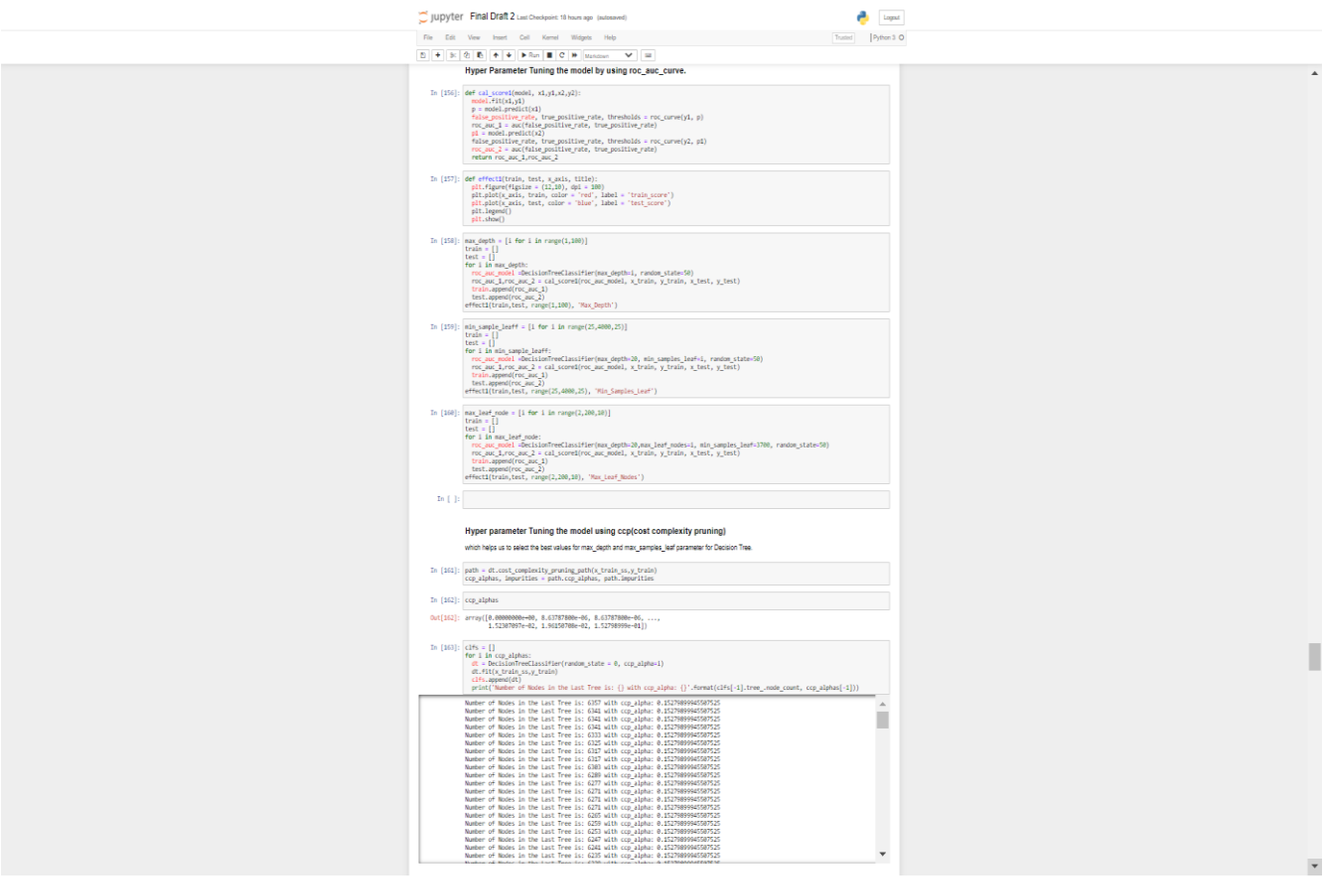
Performing cross validation on the dataset using StratifiedKFold and calculating the mean Accuracy that can be achieved by the model.

In [148]:
```
x = pd.DataFrame(data = x_train_ss, columns = inde_vars.columns)
y = y_train
from sklearn.model_selection import  StratifiedKFold
accuracy = []
skf = StratifiedKFold(n_splits = 10, random_state = None)
skf.get_n_splits(x,y)
for train_index, test_index in skf.split(x,y):
  print('Train:', train_index, 'Validation',test_index)
  x1_train,x1_test = x.iloc[train_index],x.iloc[test_index]
  y1_train,y1_test = y.iloc[train_index],y.iloc[test_index]
  dt.fit(x1_train,y1_train)
  pred = dt.predict(x1_test)
  score = accuracy_score(pred,y1_test)
  accuracy.append(score)
print(accuracy)
```

```
Train: [ 5584  5585  5586 ... 56650 56651 56652] Validation [    0     1     2 ... 5750 5751 5754]
Train: [    0     1     2 ... 56650 56651 56652] Validation [ 5584  5585  5586 ... 11359 11360 11362]
Train: [    0     1     2 ... 56650 56651 56652] Validation [11307 11308 11309 ... 17077 17078 17083]
```

Fig 2.16

18.Hyper Tuning Decision Tree Model.



jupyter Final Draft 2 Last Checkpoint: 18 hours ago (autosaved)

Hyper Parameter Tuning the model by using roc_auc_curve.

In [156]:
```
def cal_score2(model, x1,y1,x2,y2):
  model.fit(x1,y1)
  p = model.predict(x1)
  false_positive_rate, true_positive_rate, thresholds = roc_curve(y1, p)
  roc_auc_1 = auc(false_positive_rate, true_positive_rate)
  p1 = model.predict(x2)
  false_positive_rate, true_positive_rate, thresholds = roc_curve(y2, p1)
  roc_auc_2 = auc(false_positive_rate, true_positive_rate)
  return roc_auc_1,roc_auc_2
```

In [157]:
```
def effect1(train, test, x_axis, title):
  plt.figure(figsize = (12,10), dpi = 100)
  plt.plot(x_axis, train, color = 'red', label = 'train_score')
  plt.plot(x_axis, test, color = 'blue', label = 'test_score')
  plt.legend()
  plt.show()
```

In [158]:
```
max_depth = [i for i in range(1,100)]
train = []
test = []
for i in max_depth:
  roc_auc_model =DecisionTreeClassifier(max_depth=i, random_state=50)
  roc_auc_1,roc_auc_2 = cal_score2(roc_auc_model, x_train, y_train, x_test, y_test)
  train.append(roc_auc_1)
  test.append(roc_auc_2)
effect1(train,test, range(1,100), 'Max_Depth')
```

In [159]:
```
min_sample_leaff = [i for i in range(25,4000,25)]
train = []
test = []
for i in min_sample_leaff:
  roc_auc_model =DecisionTreeClassifier(max_depth=20, min_samples_leaf=i, random_state=50)
  roc_auc_1,roc_auc_2 = cal_score2(roc_auc_model, x_train, y_train, x_test, y_test)
  train.append(roc_auc_1)
  test.append(roc_auc_2)
effect1(train,test, range(25,4000,25), 'Min_Samples_Leaf')
```

In [160]:
```
max_leaf_node = [i for i in range(2,200,10)]
train = []
test = []
for i in max_leaf_node:
  roc_auc_model =DecisionTreeClassifier(max_depth=20,max_leaf_nodes=i, min_samples_leaf=3700, random_state=50)
  roc_auc_1,roc_auc_2 = cal_score2(roc_auc_model, x_train, y_train, x_test, y_test)
  train.append(roc_auc_1)
  test.append(roc_auc_2)
effect1(train,test, range(2,200,10), 'Max_Leaf_Nodes')
```

In [ ]:

Hyper parameter Tuning the model using ccp(cost complexity pruning)

which helps us to select the best values for max_depth and max_samples_leaf parameter for Decision Tree.

In [161]:
```
path = dt.cost_complexity_pruning_path(x_train_ss,y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

In [162]:
```
ccp_alphas
```

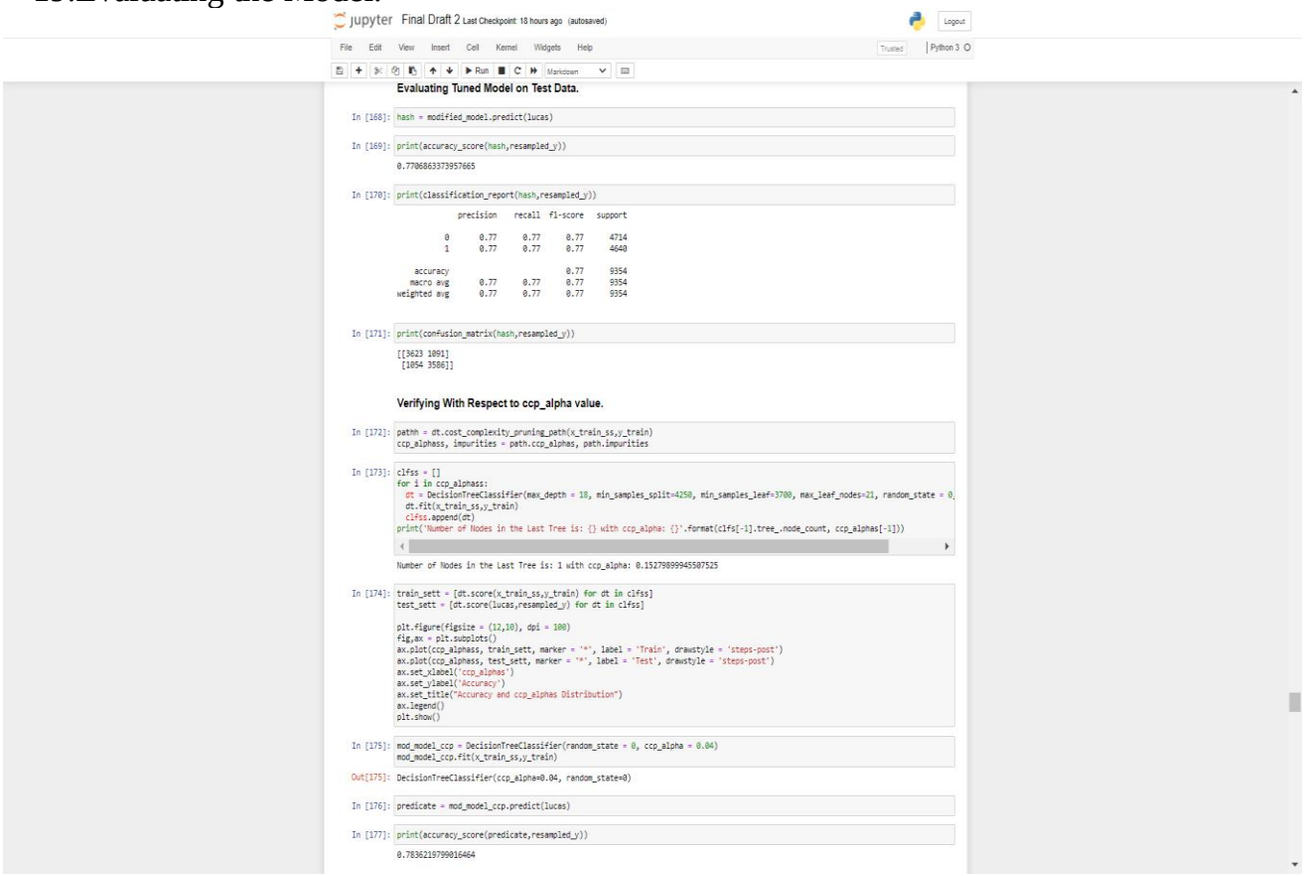
Out[162]:
```
array([0.00000000e+00, 8.63787800e-06, 8.63787800e-06, ...,
       1.52307097e-02, 1.90150708e-02, 1.52798999e-01])
```

In [163]:
```
clfs = []
for i in ccp_alphas:
  dt = DecisionTreeClassifier(random_state = 0, ccp_alpha=i)
  dt.fit(x_train_ss,y_train)
  clfs.append(dt)
  print('Number of Nodes in the Last Tree is: {} with ccp_alpha: {}'.format(clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

```
Number of Nodes in the Last Tree is: 6357 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6341 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6341 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6341 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6333 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6325 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6317 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6317 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6303 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6289 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6277 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6271 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6271 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6271 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6265 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6259 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6253 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6247 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6241 with ccp_alpha: 0.15278999450507525
Number of Nodes in the Last Tree is: 6235 with ccp_alpha: 0.15278999450507525
```

Fig 2.17

19.Evaluating the Model.



jupyter Final Draft 2 Last Checkpoint: 18 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Evaluating Tuned Model on Test Data.

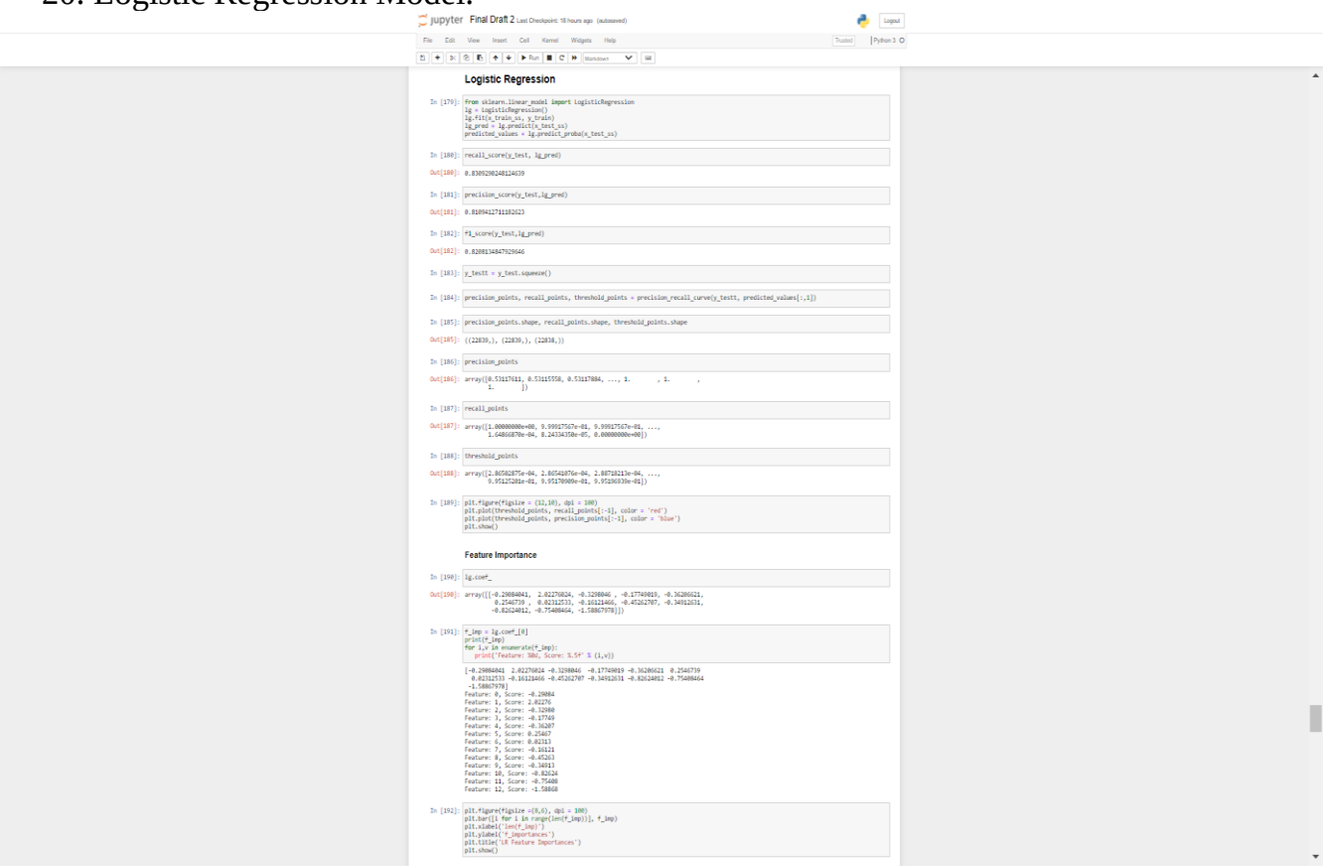
```
In [168]: hash = modified_model.predict(lucas)

In [169]: print(accuracy_score(hash,resampled_y))

          0.7708633739957665

In [170]: print(classification_report(hash,resampled_y))

                    precision    recall  f1-score   support

                 0       0.77      0.77      0.77      4714
                 1       0.77      0.77      0.77      4640

          accuracy                           0.77      9354
         macro avg       0.77      0.77      0.77      9354
      weighted avg       0.77      0.77      0.77      9354

In [171]: print(confusion_matrix(hash,resampled_y))

          [[3623 1091]
           [1054 3586]]
```

Verifying With Respect to ccp_alpha value.

```
In [172]: pathh = dt.cost_complexity_pruning_path(x_train_ss,y_train)
          ccp_alphass, impurities = path.ccp_alphas, path.impurities

In [173]: clfss = []
          for i in ccp_alphass:
            dt = DecisionTreeClassifier(max_depth = 18, min_samples_split=4250, min_samples_leaf=3700, max_leaf_nodes=21, random_state = 0,
            dt.fit(x_train_ss,y_train)
            clfss.append(dt)
          print('Number of Nodes in the Last Tree is: {} with ccp_alpha: {}'.format(clfs[-1].tree_.node_count, ccp_alphas[-1]))

          Number of Nodes in the Last Tree is: 1 with ccp_alpha: 0.15279899945507525

In [174]: train_sett = [dt.score(x_train_ss,y_train) for dt in clfss]
          test_sett = [dt.score(lucas,resampled_y) for dt in clfss]

          plt.figure(figsize = (12,10), dpi = 100)
          fig,ax = plt.subplots()
          ax.plot(ccp_alphass, train_sett, marker = '*', label = 'Train', drawstyle = 'steps-post')
          ax.plot(ccp_alphass, test_sett, marker = '*', label = 'Test', drawstyle = 'steps-post')
          ax.set_xlabel('ccp_alphas')
          ax.set_ylabel('Accuracy')
          ax.set_title("Accuracy and ccp_alphas Distribution")
          ax.legend()
          plt.show()

In [175]: mod_model_ccp = DecisionTreeClassifier(random_state = 0, ccp_alpha = 0.04)
          mod_model_ccp.fit(x_train_ss,y_train)

Out[175]: DecisionTreeClassifier(ccp_alpha=0.04, random_state=0)

In [176]: predicate = mod_model_ccp.predict(lucas)

In [177]: print(accuracy_score(predicate,resampled_y))

          0.7836219799016464
```

Fig 2.18

20. Logistic Regression Model.



jupyter Final Draft 2 Last Checkpoint: 18 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Logistic Regression

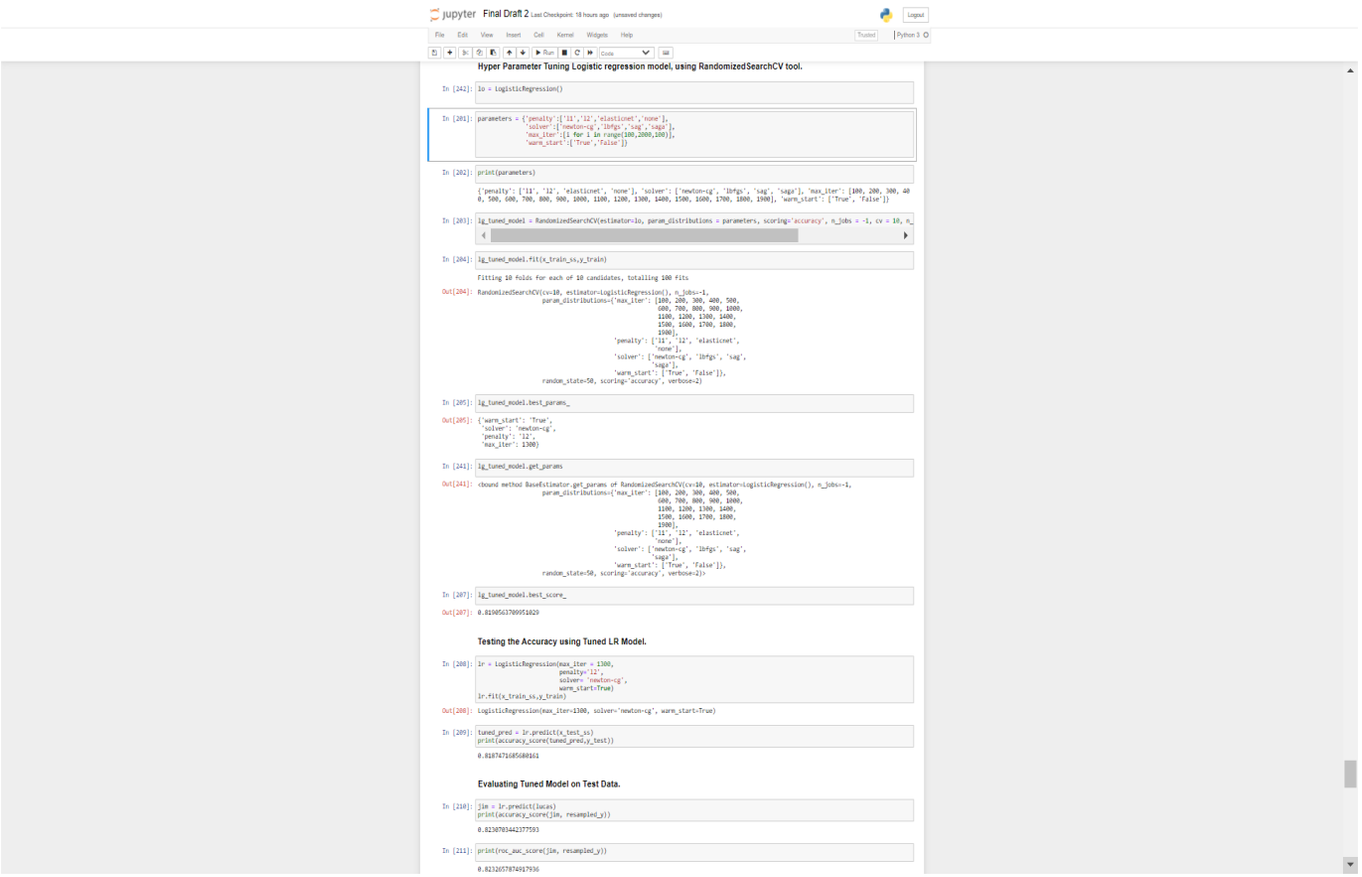
```
In [179]: from sklearn.linear_model import LogisticRegression
          lg = LogisticRegression()
          lg.fit(x_train_ss, y_train)
          lg_pred = lg.predict(x_test_ss)
          predicted_values = lg.predict_proba(x_test_ss)

In [180]: recall_score(y_test, lg_pred)

Out[180]: 0.8309290248124639

In [181]: precision_score(y_test,lg_pred)

Out[181]: 0.8109412711102623

In [182]: f1_score(y_test,lg_pred)

Out[182]: 0.8208134847929646

In [183]: y_testt = y_test.squeeze()

In [184]: precision_points, recall_points, threshold_points = precision_recall_curve(y_testt, predicted_values[:,1])

In [185]: precision_points.shape, recall_points.shape, threshold_points.shape

Out[185]: ((22839,), (22839,), (22838,))

In [186]: precision_points

Out[186]: array([0.53117611, 0.53115558, 0.53117884, ..., 1.        , 1.        ,
                 1.        ])

In [187]: recall_points

Out[187]: array([1.00000000e+00, 9.99917567e-01, 9.99917567e-01, ...,
                 1.54866870e-04, 8.24334350e-05, 0.00000000e+00])

In [188]: threshold_points

Out[188]: array([2.86502875e-04, 2.86541076e-04, 2.88718213e-04, ...,
                 9.95125816e-01, 9.95170909e-01, 9.95269339e-01])

In [189]: plt.figure(figsize = (12,10), dpi = 100)
          plt.plot(threshold_points, recall_points[:-1], color = 'red')
          plt.plot(threshold_points, precision_points[:-1], color = 'blue')
          plt.show()
```

Feature Importance

```
In [190]: lg.coef_

Out[190]: array([[-0.29084041,  2.02276024, -0.3298046 , -0.17749019, -0.36206621,
                   0.2546739 ,  0.02312533, -0.16121466, -0.45262707, -0.34912631,
                  -0.82624012, -0.75408464, -1.58867978]])

In [191]: f_imp = lg.coef_[0]
          print(f_imp)
          for i,v in enumerate(f_imp):
            print('Feature: %0d, Score: %.5f' % (i,v))

          [-0.29084041 2.02276024 -0.3298046  -0.17749019 -0.36206621 0.2546739
           0.02312533 -0.16121466 -0.45262707 -0.34912631 -0.82624012 -0.75408464
           -1.58867978]
          Feature: 0, Score: -0.29084
          Feature: 1, Score: 2.02276
          Feature: 2, Score: -0.32980
          Feature: 3, Score: -0.17749
          Feature: 4, Score: -0.36207
          Feature: 5, Score: 0.25467
          Feature: 6, Score: 0.02313
          Feature: 7, Score: -0.16121
          Feature: 8, Score: -0.45263
          Feature: 9, Score: -0.34913
          Feature: 10, Score: -0.82624
          Feature: 11, Score: -0.75408
          Feature: 12, Score: -1.58868

In [192]: plt.figure(figsize =(8,6), dpi = 100)
          plt.bar([i for i in range(len(f_imp))], f_imp)
          plt.xlabel('len(f_imp)')
          plt.ylabel('f_importances')
          plt.title('LR Feature Importances')
          plt.show()
```

Fig 2.19

21.Tuning Logistic Regression Model.



jupyter Final Draft 2 Last Checkpoint: 18 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Hyper Parameter Tuning Logistic regression model, using RandomizedSearchCV tool.

In [242]:
```
lo = LogisticRegression()
```

In [201]:
```
parameters = {'penalty':['l1','l2','elasticnet','none'],
              'solver':['newton-cg','lbfgs','sag','saga'],
              'max_iter':[i for i in range(100,2000,100)],
              'warm_start':['True','False']}
```

In [202]:
```
print(parameters)
```

```
{'penalty': ['l1', 'l2', 'elasticnet', 'none'], 'solver': ['newton-cg', 'lbfgs', 'sag', 'saga'], 'max_iter': [100, 200, 300, 40
0, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900], 'warm_start': ['True', 'False']}
```

In [203]:
```
lg_tuned_model = RandomizedSearchCV(estimator=lo, param_distributions = parameters, scoring='accuracy', n_jobs = -1, cv = 10, n_
```

In [204]:
```
lg_tuned_model.fit(x_train_ss,y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

Out[204]:
```
RandomizedSearchCV(cv=10, estimator=LogisticRegression(), n_jobs=-1,
                   param_distributions={'max_iter': [100, 200, 300, 400, 500,
                                                     600, 700, 800, 900, 1000,
                                                     1100, 1200, 1300, 1400,
                                                     1500, 1600, 1700, 1800,
                                                     1900],
                                        'penalty': ['l1', 'l2', 'elasticnet',
                                                    'none'],
                                        'solver': ['newton-cg', 'lbfgs', 'sag',
                                                   'saga'],
                                        'warm_start': ['True', 'False']},
                   random_state=50, scoring='accuracy', verbose=2)
```

In [205]:
```
lg_tuned_model.best_params_
```

Out[205]:
```
{'warm_start': 'True',
 'solver': 'newton-cg',
 'penalty': 'l2',
 'max_iter': 1300}
```

In [241]:
```
lg_tuned_model.get_params
```

Out[241]:
```
<bound method BaseEstimator.get_params of RandomizedSearchCV(cv=10, estimator=LogisticRegression(), n_jobs=-1,
                   param_distributions={'max_iter': [100, 200, 300, 400, 500,
                                                     600, 700, 800, 900, 1000,
                                                     1100, 1200, 1300, 1400,
                                                     1500, 1600, 1700, 1800,
                                                     1900],
                                        'penalty': ['l1', 'l2', 'elasticnet',
                                                    'none'],
                                        'solver': ['newton-cg', 'lbfgs', 'sag',
                                                   'saga'],
                                        'warm_start': ['True', 'False']},
                   random_state=50, scoring='accuracy', verbose=2)>
```

In [207]:
```
lg_tuned_model.best_score_
```

Out[207]:
```
0.8190563708951029
```

Testing the Accuracy using Tuned LR Model.

In [208]:
```
lr = LogisticRegression(max_iter = 1300,
                        penalty='l2',
                        solver= 'newton-cg',
                        warm_start=True)
lr.fit(x_train_ss,y_train)
```

Out[208]:
```
LogisticRegression(max_iter=1300, solver='newton-cg', warm_start=True)
```

In [209]:
```
tuned_pred = lr.predict(x_test_ss)
print(accuracy_score(tuned_pred,y_test))
```

```
0.8187471685608161
```

Evaluating Tuned Model on Test Data.

In [210]:
```
jim = lr.predict(lucas)
print(accuracy_score(jim, resampled_y))
```

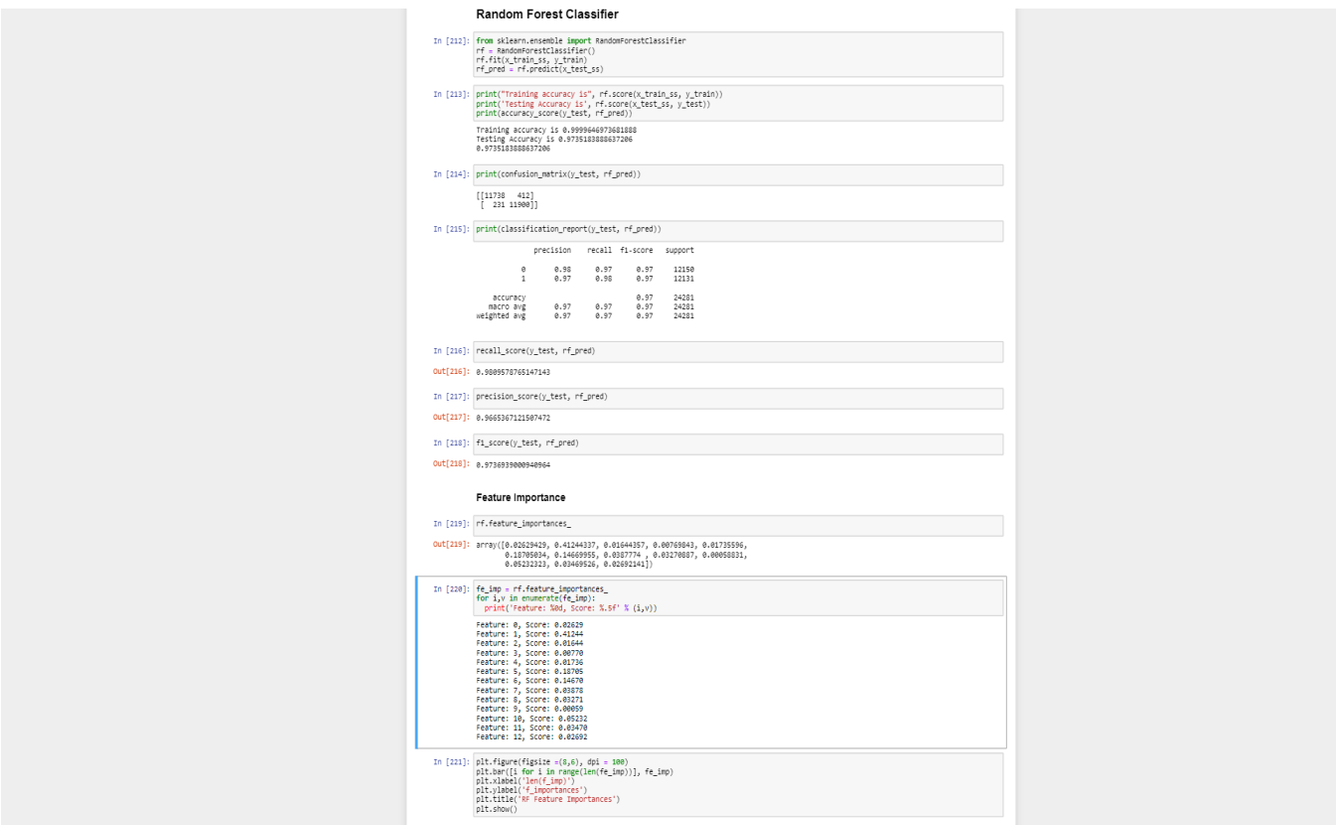
```
0.8230703442377593
```

In [211]:
```
print(roc_auc_score(jim, resampled_y))
```

```
0.8232657874917936
```

Fig 2.20

22.Random Forest Model.



Random Forest Classifier

In [212]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train_ss, y_train)
rf_pred = rf.predict(x_test_ss)
```

In [213]:
```
print("Training accuracy is", rf.score(x_train_ss, y_train))
print('Testing Accuracy is', rf.score(x_test_ss, y_test))
print(accuracy_score(y_test, rf_pred))
```

```
Training accuracy is 0.9999646973681888
Testing Accuracy is 0.9735183888637206
0.9735183888637206
```

In [214]:
```
print(confusion_matrix(y_test, rf_pred))
```

```
[[11738   412]
 [  231 11900]]
```

In [215]:
```
print(classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97     12150
           1       0.97      0.98      0.97     12131

    accuracy                           0.97     24281
   macro avg       0.97      0.97      0.97     24281
weighted avg       0.97      0.97      0.97     24281
```

In [216]:
```
recall_score(y_test, rf_pred)
```

Out[216]:
```
0.9089570765147143
```

In [217]:
```
precision_score(y_test, rf_pred)
```

Out[217]:
```
0.9665367121507472
```

In [218]:
```
f1_score(y_test, rf_pred)
```

Out[218]:
```
0.9736939000940964
```

Feature Importance

In [219]:
```
rf.feature_importances_
```

Out[219]:
```
array([0.02629429, 0.41244337, 0.01644357, 0.00769843, 0.01735596,
       0.18705034, 0.14669955, 0.0387774 , 0.03270887, 0.00058831,
       0.05232323, 0.03469526, 0.02692141])
```

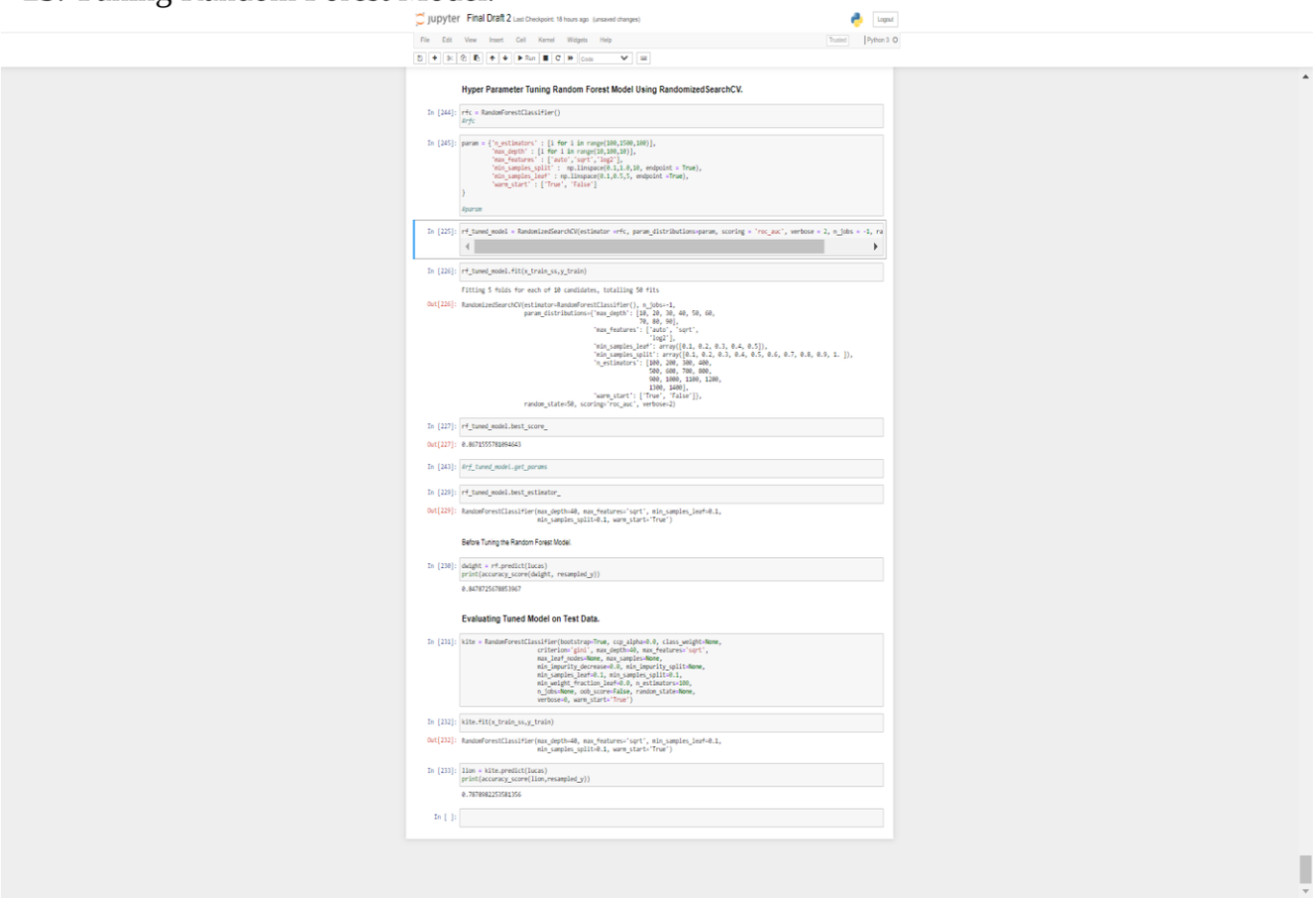
In [220]:
```
fe_imp = rf.feature_importances_
for i,v in enumerate(fe_imp):
  print('Feature: %0d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.02629
Feature: 1, Score: 0.41244
Feature: 2, Score: 0.01644
Feature: 3, Score: 0.00770
Feature: 4, Score: 0.01736
Feature: 5, Score: 0.18705
Feature: 6, Score: 0.14670
Feature: 7, Score: 0.03878
Feature: 8, Score: 0.03271
Feature: 9, Score: 0.00059
Feature: 10, Score: 0.05232
Feature: 11, Score: 0.03470
Feature: 12, Score: 0.02692
```

In [221]:
```
plt.figure(figsize =(8,6), dpi = 100)
plt.bar([i for i in range(len(fe_imp))], fe_imp)
plt.xlabel('len(f_imp)')
plt.ylabel('f_importances')
plt.title('RF Feature Importances')
plt.show()
```

Fig 2.21

23. Tuning Random Forest Model.



jupyter Final Draft 2 Last Checkpoint: 18 hours ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Hyper Parameter Tuning Random Forest Model Using RandomizedSearchCV.

```
In [244]: rfc = RandomForestClassifier()
          #rfc

In [245]: param = {'n_estimators' : [i for i in range(100,1500,100)],
                   'max_depth' : [i for i in range(10,100,10)],
                   'max_features' : ['auto','sqrt','log2'],
                   'min_samples_split' :  np.linspace(0.1,1,0.10, endpoint = True),
                   'min_samples_leaf' : np.linspace(0.1,0.5,5, endpoint =True),
                   'warm_start' : ['True', 'False']
          }
          #param

In [225]: rf_tuned_model = RandomizedSearchCV(estimator =rfc, param_distributions=param, scoring = 'roc_auc', verbose = 2, n_jobs = -1, ra

In [226]: rf_tuned_model.fit(x_train_ss,y_train)

          Fitting 5 folds for each of 10 candidates, totalling 50 fits

Out[226]: RandomizedSearchCV(estimator=RandomForestClassifier(), n_jobs=-1,
                             param_distributions={'max_depth': [10, 20, 30, 40, 50, 60,
                                                                70, 80, 90],
                                                  'max_features': ['auto', 'sqrt',
                                                                   'log2'],
                                                  'min_samples_leaf': array([0.1, 0.2, 0.3, 0.4, 0.5]),
                                                  'min_samples_split': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
                                                  'n_estimators': [100, 200, 300, 400,
                                                                   500, 600, 700, 800,
                                                                   900, 1000, 1100, 1200,
                                                                   1300, 1400],
                                                  'warm_start': ['True', 'False']},
                             random_state=50, scoring='roc_auc', verbose=2)

In [227]: rf_tuned_model.best_score_

Out[227]: 0.8671555784894643

In [243]: #rf_tuned_model.get_params

In [229]: rf_tuned_model.best_estimator_

Out[229]: RandomForestClassifier(max_depth=40, max_features='sqrt', min_samples_leaf=0.1,
                                 min_samples_split=0.1, warm_start='True')
```

Before Tuning the Random Forest Model.

```
In [230]: dwight = rf.predict(lucas)
          print(accuracy_score(dwight, resampled_y))

          0.8478725678053967
```

Evaluating Tuned Model on Test Data.

```
In [231]: kite = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                        criterion='gini', max_depth=40, max_features='sqrt',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0, min_impurity_split=None,
                                        min_samples_leaf=0.1, min_samples_split=0.1,
                                        min_weight_fraction_leaf=0.0, n_estimators=200,
                                        n_jobs=None, oob_score=False, random_state=None,
                                        verbose=0, warm_start='True')

In [232]: kite.fit(x_train_ss,y_train)

Out[232]: RandomForestClassifier(max_depth=40, max_features='sqrt', min_samples_leaf=0.1,
                                 min_samples_split=0.1, warm_start='True')

In [233]: lion = kite.predict(lucas)
          print(accuracy_score(lion,resampled_y))

          0.7878982253581356

In [ ]:
```

Fig 2.22

CONCLUSION

Finally we can conclude the Project by Understanding how Machine Learning and Artificial Intelligence are Rapidly Revolutionizing the world to make live a Better Life.
So, As we all know NON contrast CT scan and some other tests are the current standard for initial screening of the head trauma and Brain Related Diseases, so we aimed to develop a model which might reduce the usage of CT scan, Because CT scan delivers a high dose of radiation to the patient which might be very harmful for the patients in the near future.
Although, This Model doesn't prevent you from going for the Initial screening process but it helps save a lot of money and time for the people who fall in False Positive Rate, where the person is not suffering from the stroke, Even though he goes for the test wasting lot of money and time.

FUTURE SCOPE

As we have only developed a prototype model, which takes in the data and gives the result in the format of 1/0, This project has a lot of things to be seen under the microscope which we are willing to do that in the near future.

Some of the Enhancements which we might do are:

1. Building a Better Predictive model than earlier.
2. Working with the MRI Images.
3. Eliminating the Minute Errors.
4. Deploying the Project to a Workstation.
5. Making an Interactive Website.
6. Working on other Healthcare Datasets.
7. Making Healthcare Infrastructure on the Planet Better than Before.

REFERENCES

1. https://www.world-stroke.org/about-wso/annual-reports
2. https://www.medicinenet.comstroke_symptoms_and_treatment/article.htm
3. https://towardsdatascience.com/data-science/home
4. https://medium.com/
5. https://www.analyticsvidhya.com/
6. https://www.youtube.com/results?search_query=krish+naik+data+science
7. https://scholar.google.com/
8. https://www.kaggle.com/
9. https://trainings.internshala.com/
10. https://www.theedgemarkets.com/article/budget-2021-healthcare-measures-welcomed-fall-short