A Project Phase II Report on

**Brain Stroke Prediction Using Data Science and Machine Learning**

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)**

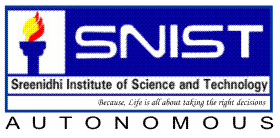
**Venkatesh. V(17311A12A8)**

**…**

under the guidance of

**Ms. N. Sreevidya**

**Asst. 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**

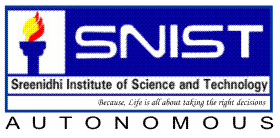
Hyderabad – 500 085

**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 Group-Project report on **“Brain Stroke Prediction Using Data Science and Machine Learning”** 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 Group-Project work have not been submitted to any other University or Institute for the award of any degree or diploma.

**Internal guide**

**(Mr. Mekala Sreenivas)**

**Project Coordinator**

**(Ms.N.Sreevidya)**

**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 Group project work, titled **“Brain Stroke Prediction Using Data Science and Machine Learning”** 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, Asst. Professor** in Information Technology for her guidance, valuable suggestions and encouragement in completing the Group-Project 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, **Ms.N.Sreevidya**, Associate Professor & Group-Project Work Coordinator of the Department of Information Technology, Sreenidhi Institute of Science and Technology (An Autonomous Institution), Hyderabad for permitting us to do our Group-Project 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 Group-Project work.

**Anil. M (17311A1289)**

**Abhinay. L(17311A12A7)**

**Venkatesh. V(17311A12A9)**

**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 datasets and build a predictive model to estimate whether a person is suffering from a brain stroke or not.

**Project Guide:**

Anil. M (17311A1289) Ms. N. Sreevidya

Abhinay. L (17311A12A7) Assistant Professor

Venkatesh. V (17311A12A8) Dept. of IT, SNIST

**IV-2 (2021), Dept. of IT, SNIST**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **LIST OF FIGURES** | |  |
| **S. No** | **Fig. No** | **Title of Figure** | **Page. No** |
| 1 | 3.1 | Architectural Design | 05 |
| 2 | 3.2 | Use case Diagram for Administrator’s Module | 07 |
| 3 | 3.3 | Use case Diagram for Event Manager’s Module | 08 |
| 4 | 3.4 | Use case Diagram for Alumni/Student Module | 09 |
| 5 | 3.5 | Class Diagram for Online College Alumni System | 10 |
| 6 | 3.6 | Sequence Diagram for Administrator’s Module | 11 |
| 7 | 3.7 | Sequence Diagram for Event Manager’s Module | 12 |
| 8 | 3.8 | Sequence Diagram for Alumni/Student Module | 13 |
| 9 | 3.9 | Activity Diagram | 14 |
| 10 | 5.1 | Screenshot of Importing the Necessary Libraries | 21 |
| 11 | 5.2 | Screenshot of Importing and Skimming the Data Set | 22 |
| 12 | 5.3 | Screenshot of Exploring Numerical Columns | 23 |
| 13 | 5.4 | Screenshot of Anomaly and Outliers Detection | 24 |
| 14 | 5.5 | Screenshot of Exploring Categorical Columns and Treating Missing Values | 25 |
| 15 | 5.6 | Screenshot of Various Charts | 26 |
| 16 | 5.7 | Screenshot of auto-viz Description | 27 |
| 17 | 5.8 | Screenshots of auto viz charts | 28 |
| 18 | 5.9 | Screenshot of Exploring Data Using Interactive Widgets | 29 |
| 19 | 5.10 | Screenshot of Feature Transformation | 30 |
| 20 | 5.11 | Screenshot of Feature Selection | 31 |
| 21 | 5.12 | Screenshot of Handling Imbalanced Data, Train Test Split and Creating Test Data. | 32 |
| 22 | 5.13 | Screenshot of Predictive Models.  Decision Tree | 33 |
| 23 | 5.14 | Screenshot of Tree Plot | 34 |
| 24 | 5.15 | Screenshot of Logistic Regression Model | 35 |
| 25 | 5.16 | Screenshot of Tuning LR Model | 36 |
| 26 | 5.17 | Screenshot of Random Forest Model | 37 |
| 27 | 5.18 | Screenshot of Tuning RF Model | 38 |

**INDEX**

**1. ABSTRACT** I

**2. LIST OF FIGURES** II

**3. INTRODUCTION**

1.1 Scope 12

1.2 Existing System 12

1.3 Proposed System 12

**4. SYSTEM ANALYSIS**

2.1 Functional Requirement Specifications 14

2.2 Performance Requirements 14

2.3 Software Requirements 14

2.4 Hardware Requirements 16

**5. SYSTEM DESIGN**

3.1 Architecture Design 5

3.2 Modules 6

3.3 UML Diagrams 7

3.3.1 Use Case Diagrams 7

3.3.2 Class Diagrams 10

3.3.3 Sequence Diagrams 11

3.3.4 Activity Diagrams 14

**6. SYSTEM IMPLEMENTATION** 5

**7. OUTPUT SCREENS** 21

**8. CONCLUSION AND FUTURE SCOPE** 40

**9. REFERENCES** 41

**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 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 seaborn

#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 balanced 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))

### 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)

predicate = mod\_model\_ccp.predict(lucas)

print(accuracy\_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)

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.\*\*

jim = lr.predict(lucas)

print(accuracy\_score(jim, resampled\_y))

print(roc\_auc\_score(jim, 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))

### \*\*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')

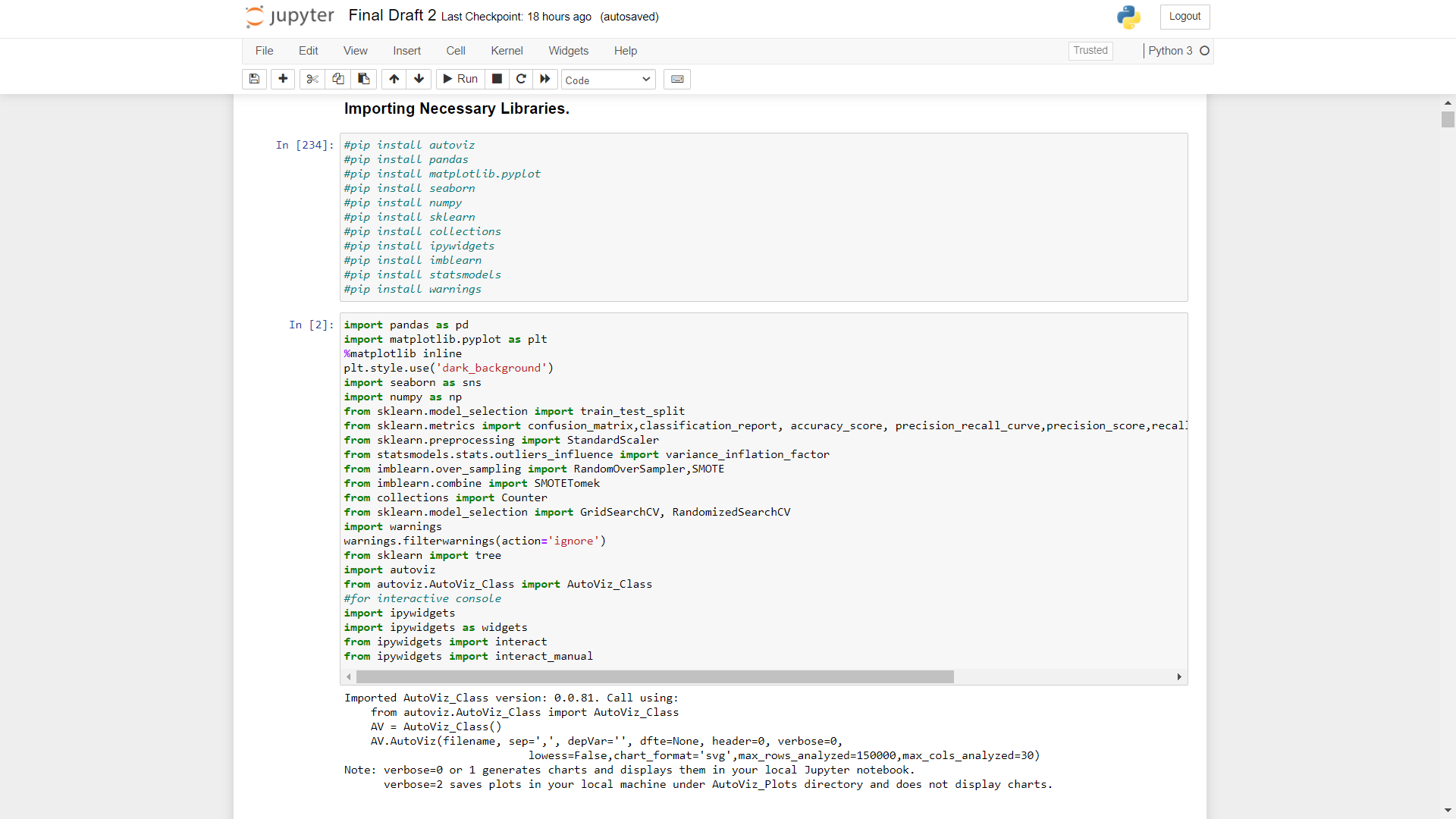
kite.fit(x\_train\_ss,y\_train)

lion = kite.predict(lucas)

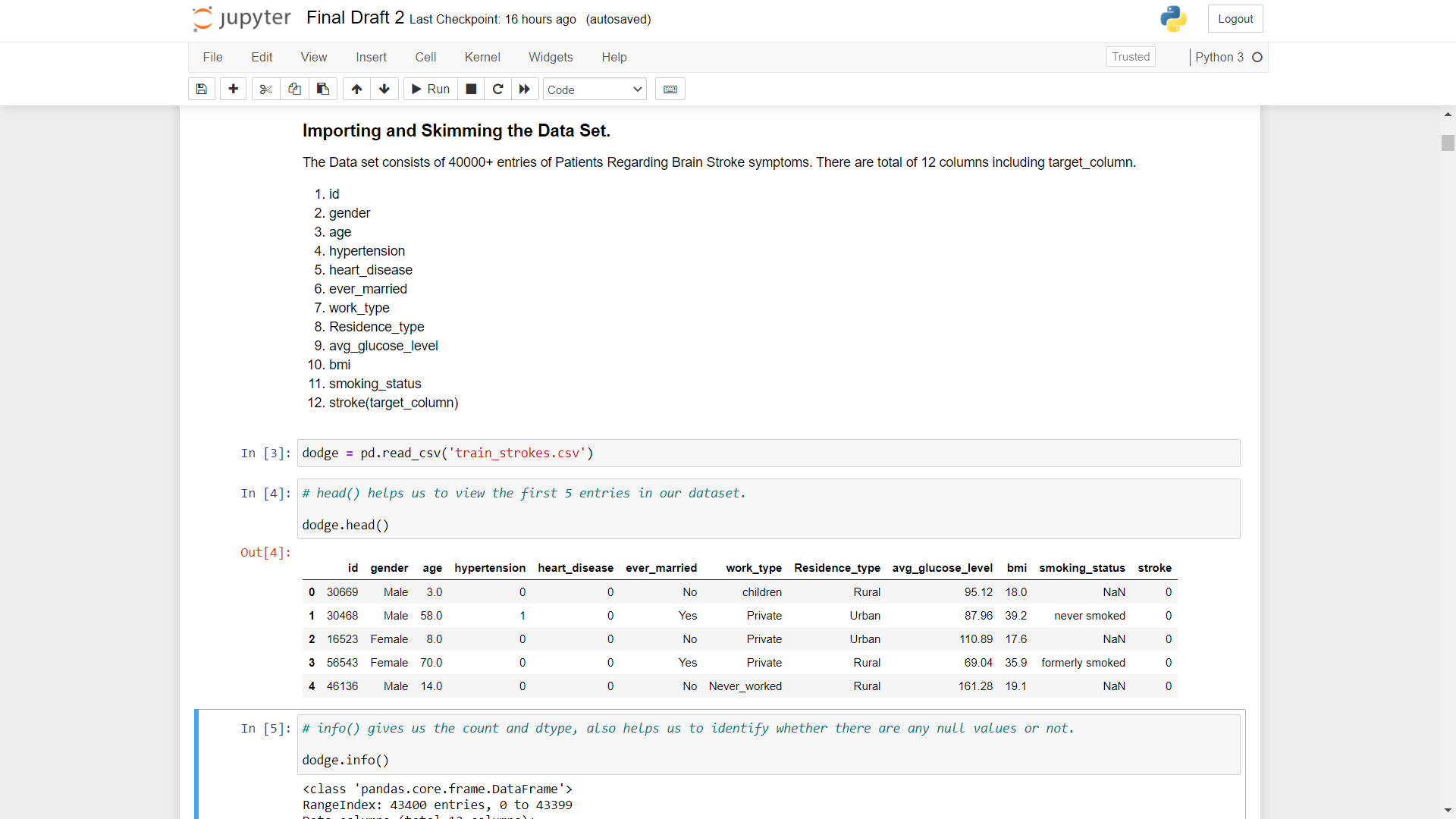
print(accuracy\_score(lion,resampled\_y))

**OUTPUT SCREENS:**

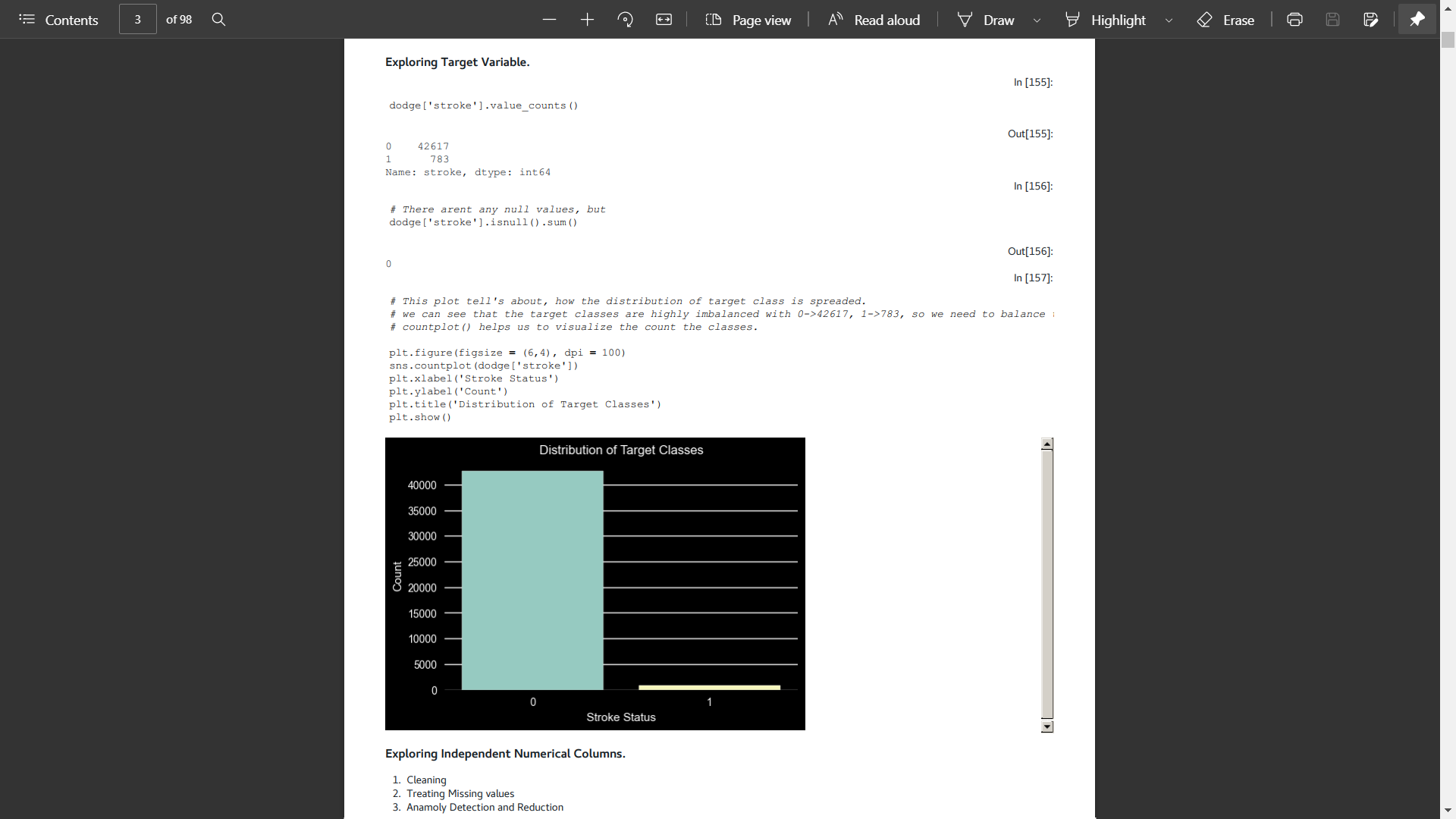
1.Importing Libraries.



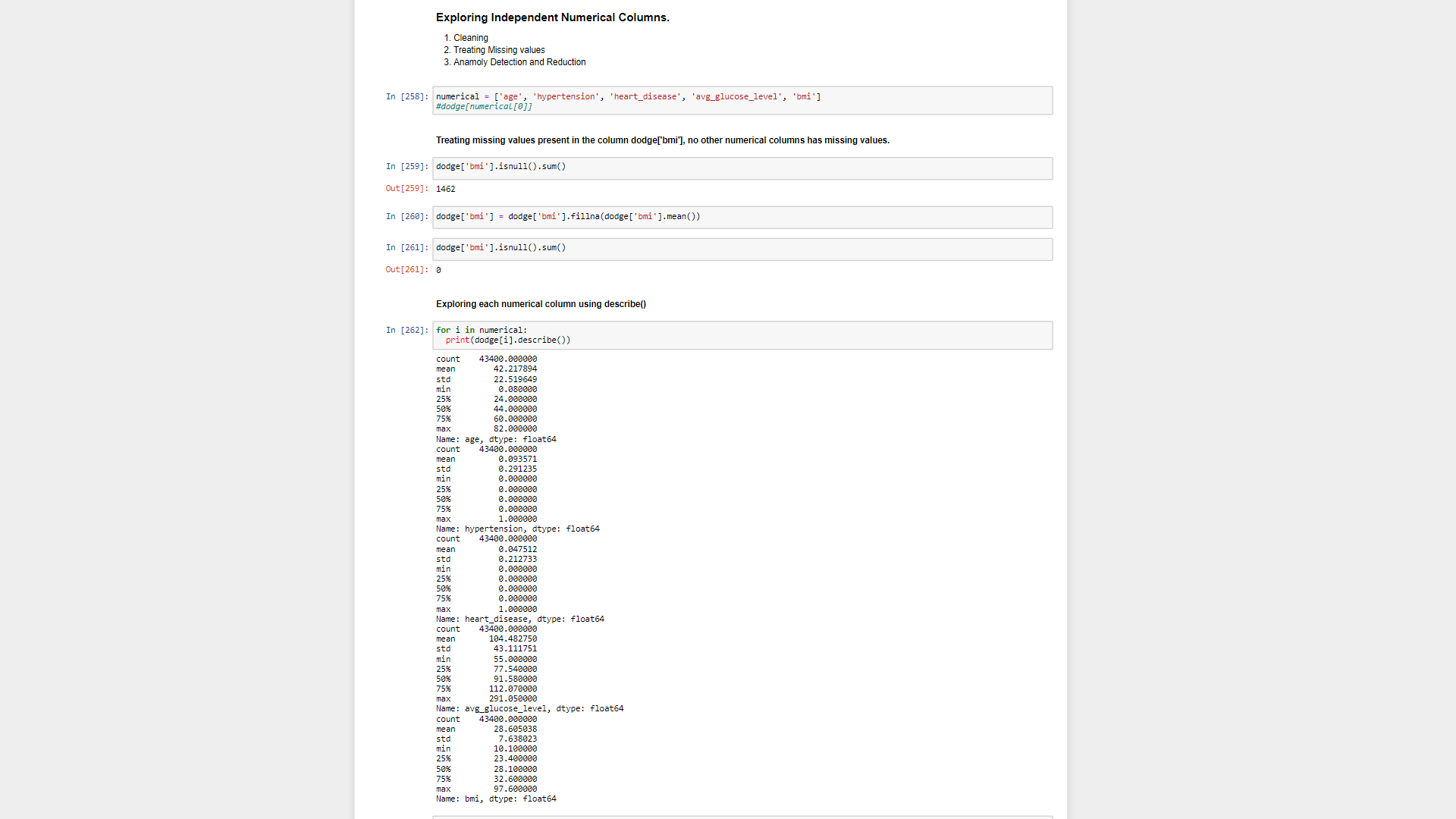
2.Importing and Skimming of Data set.



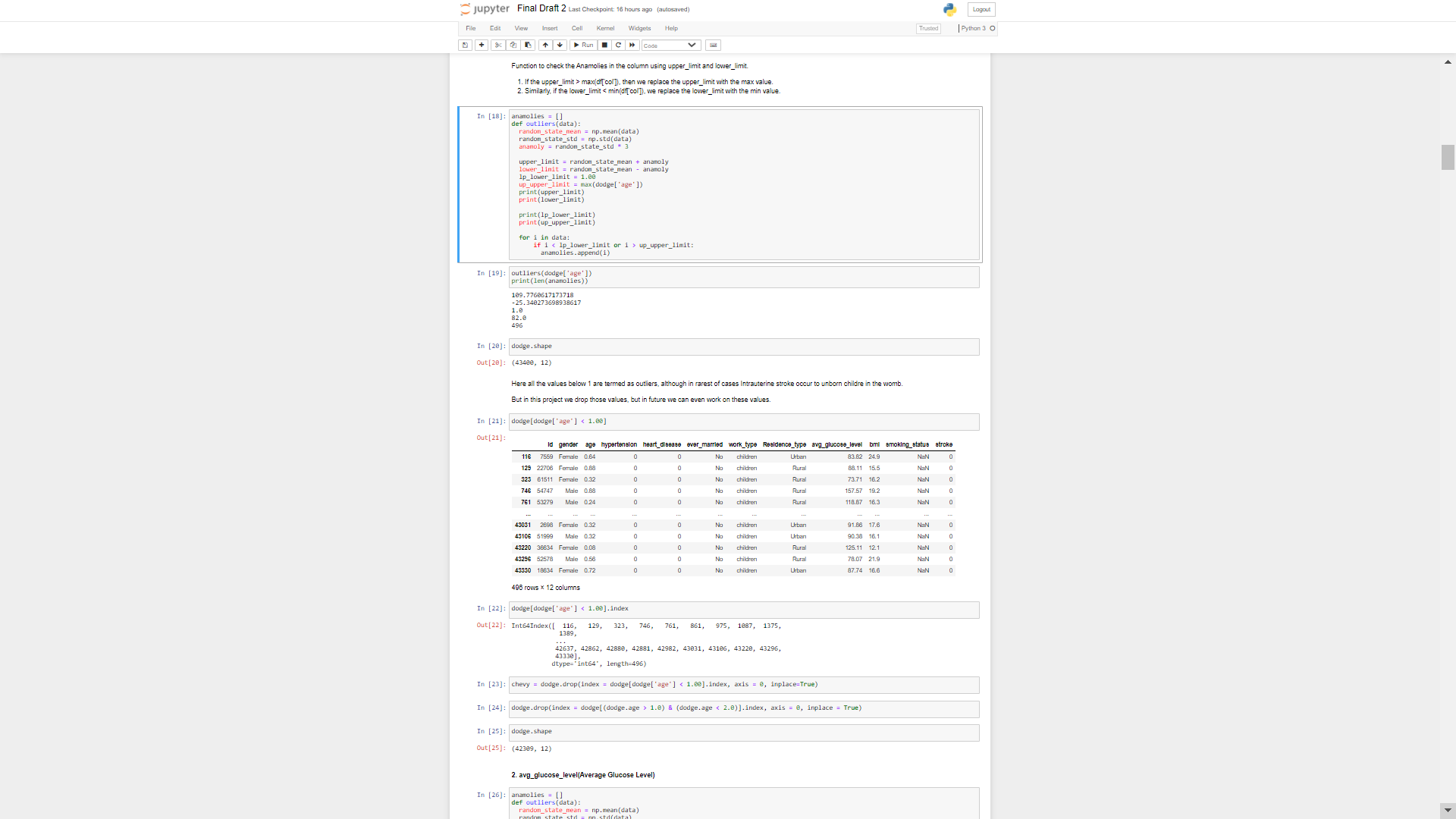
3.Exploring Target Variable.

******

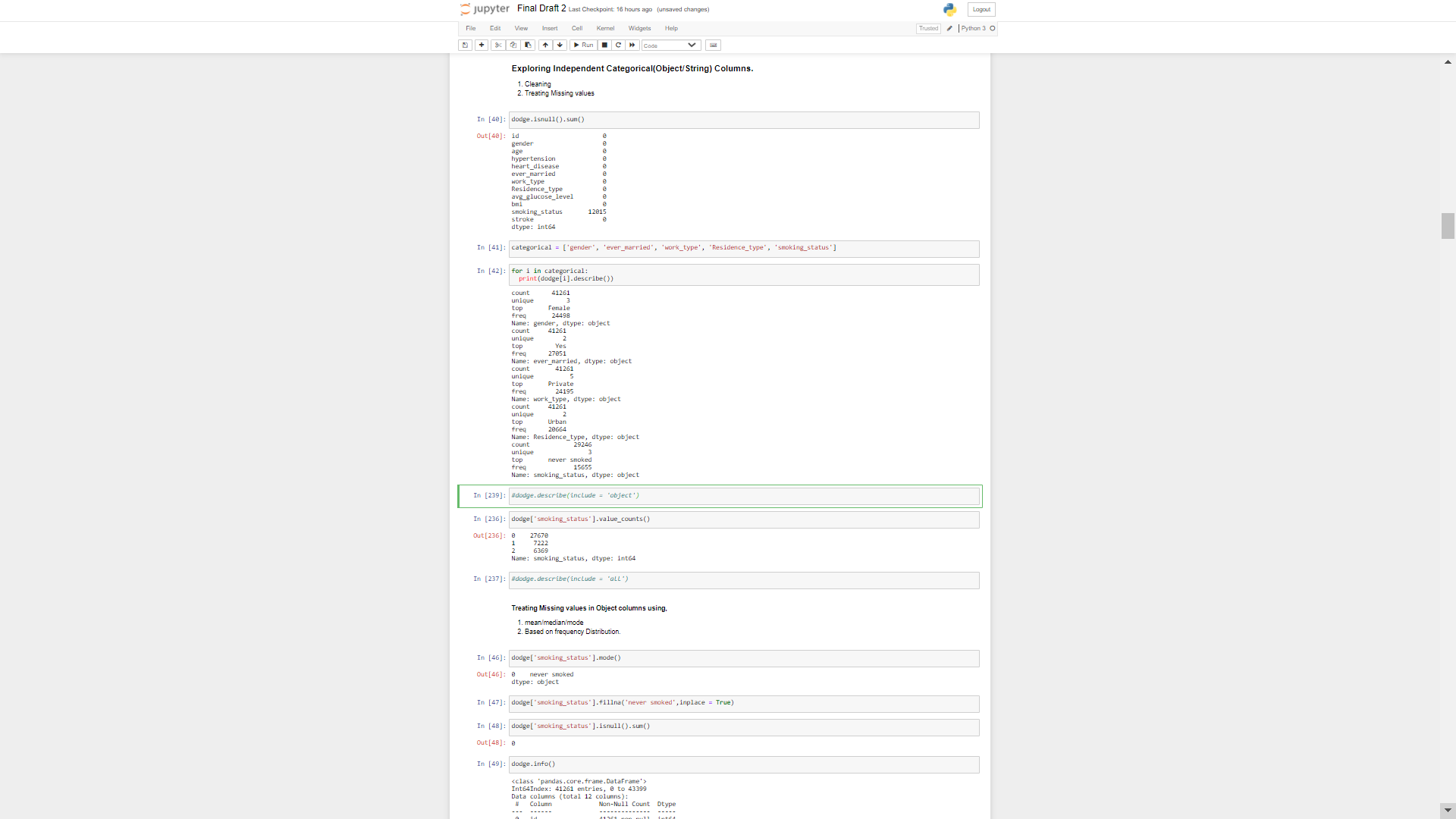
4.Exploring Independent Columns.

******

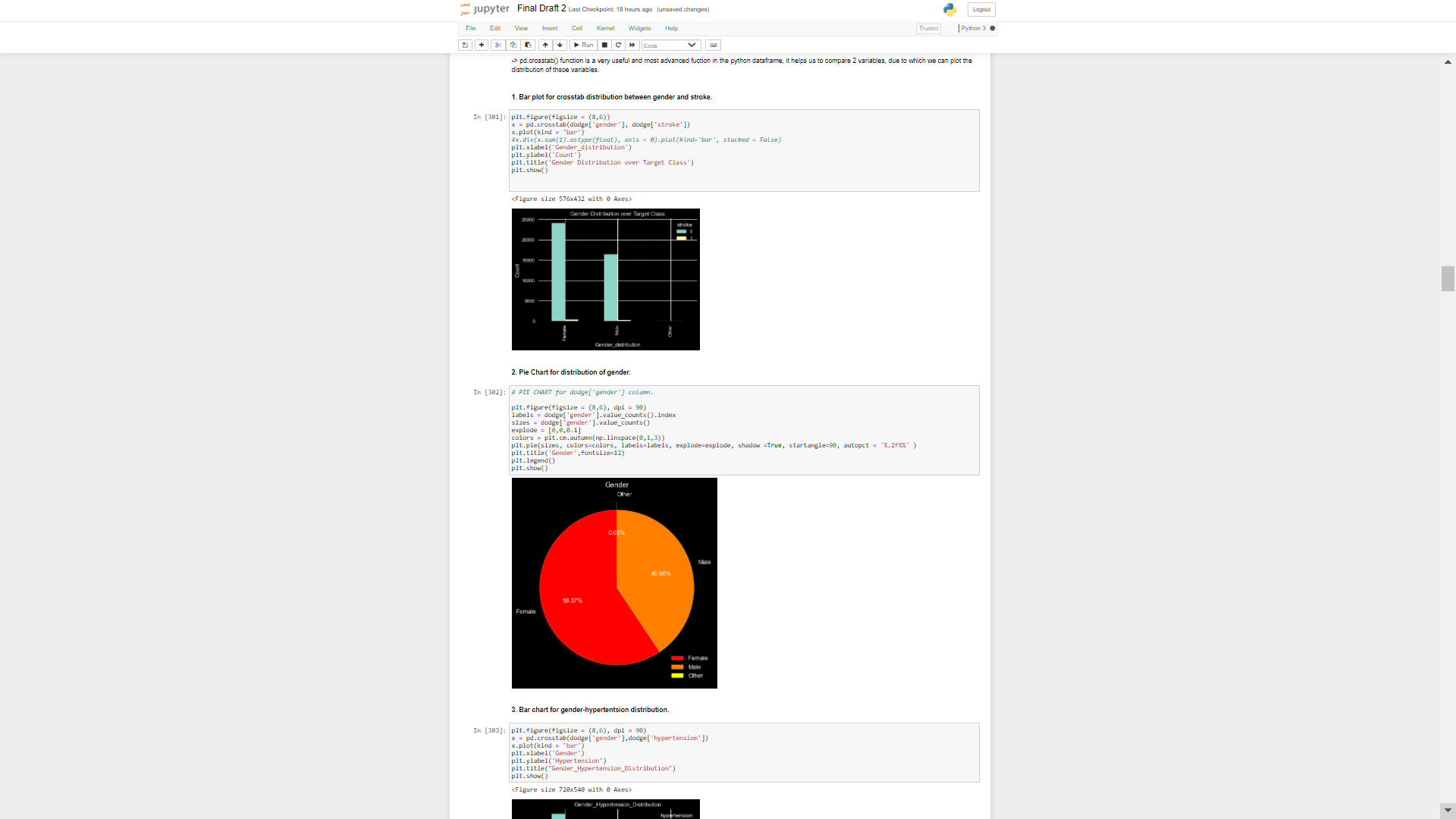
5.Anamoly Detection and Reduction

******

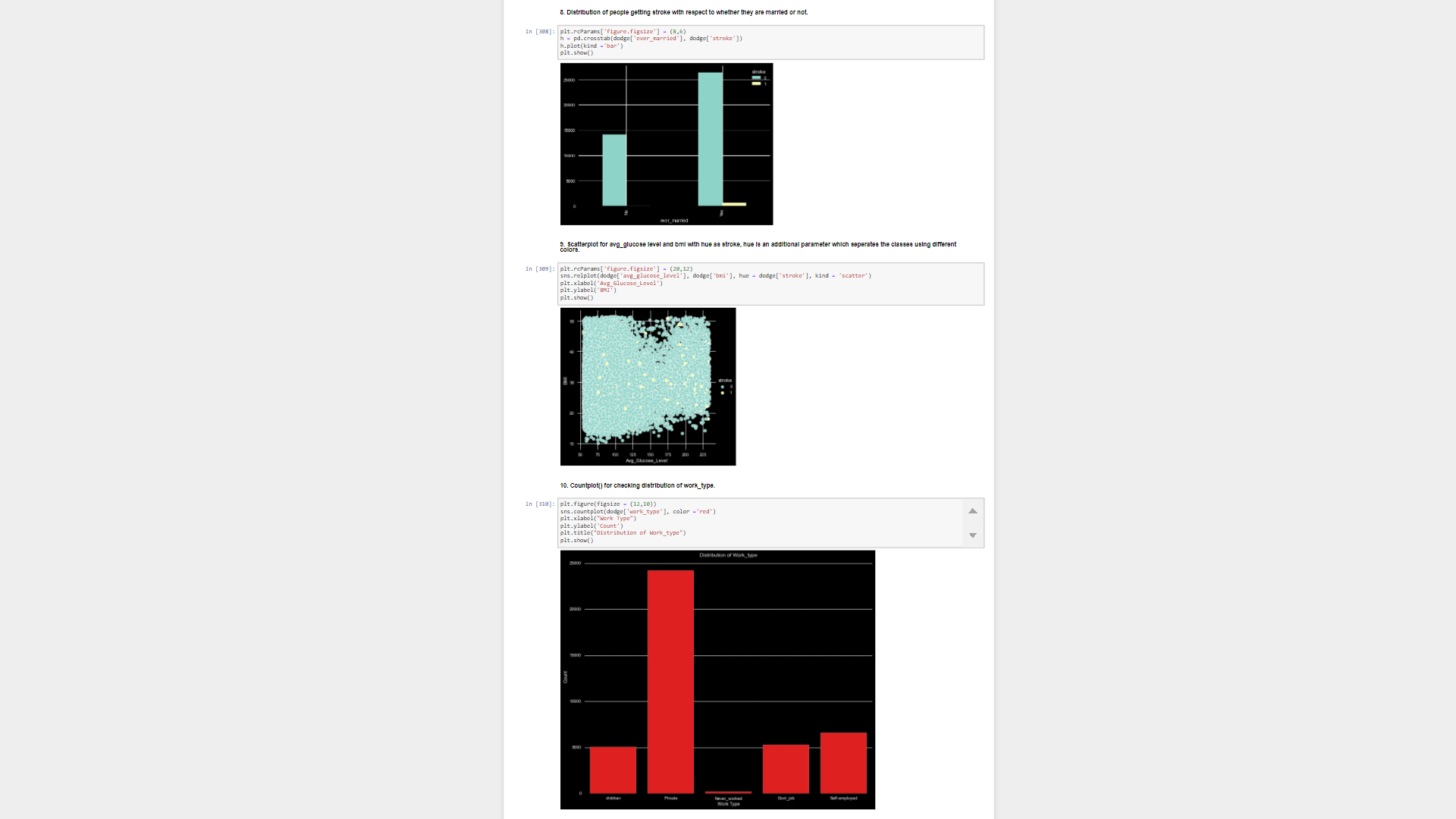
6.Exploring Independent Categorical Columns.

******

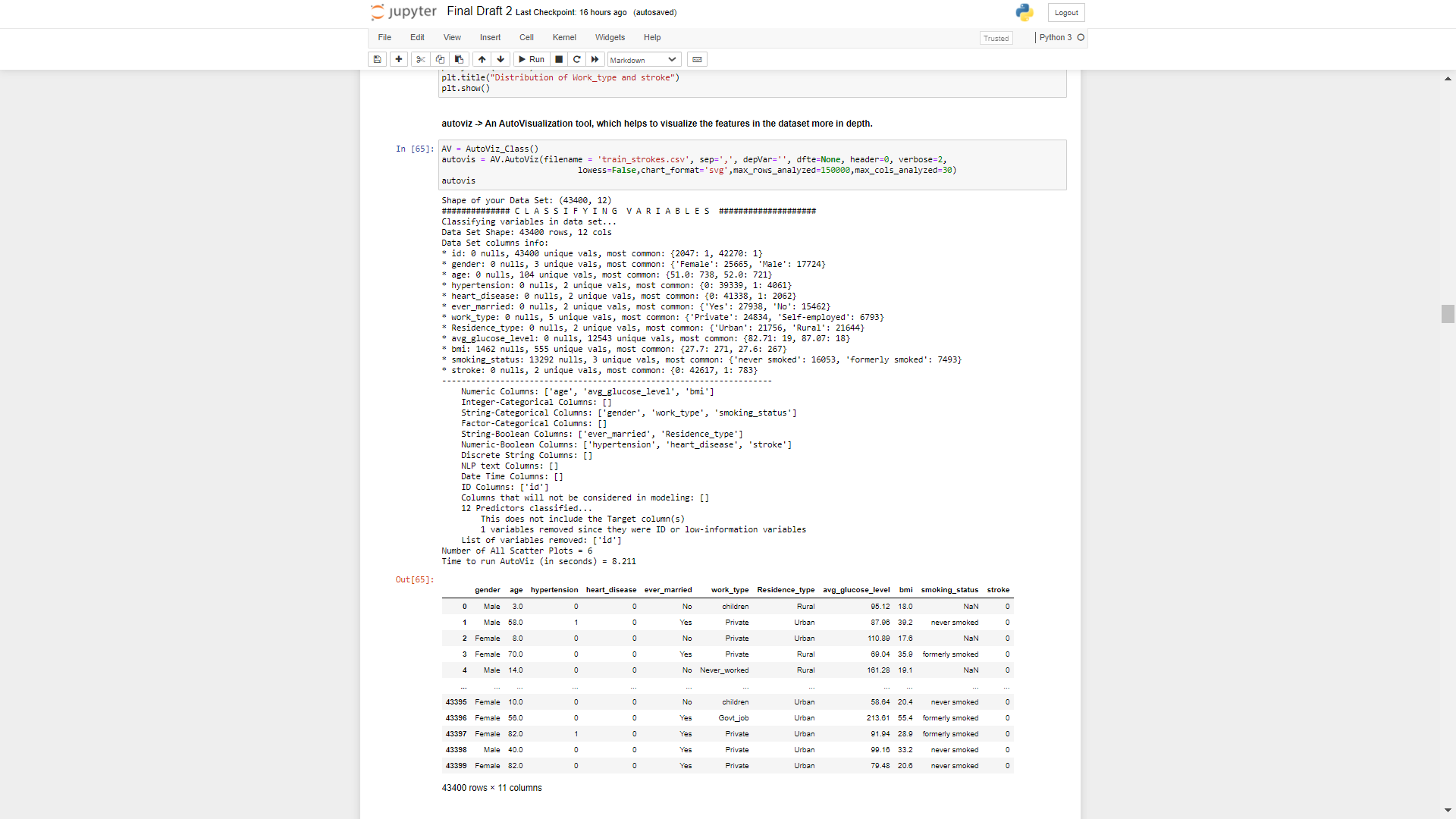
7.EDA.

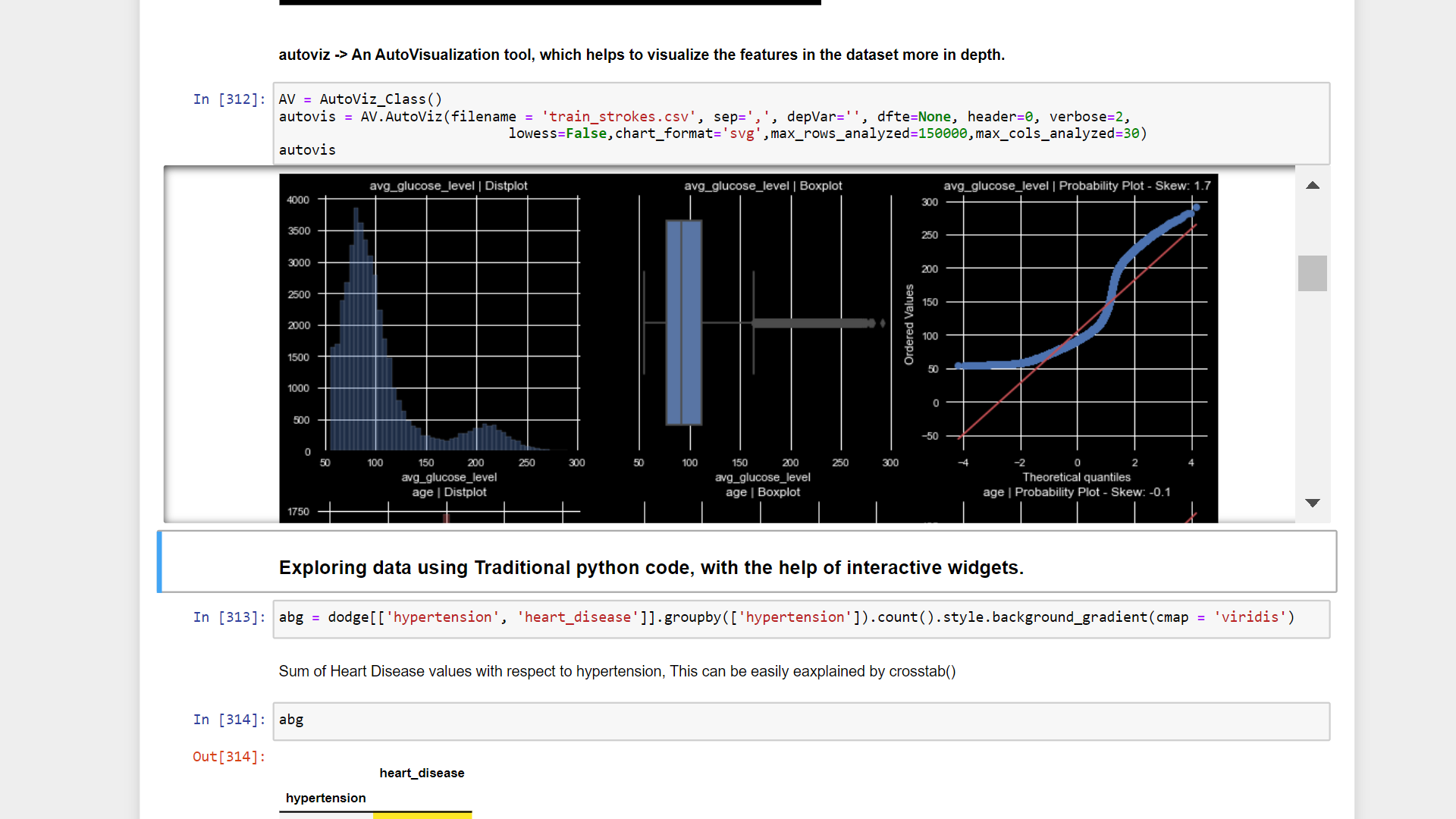
******

8.EDA 2

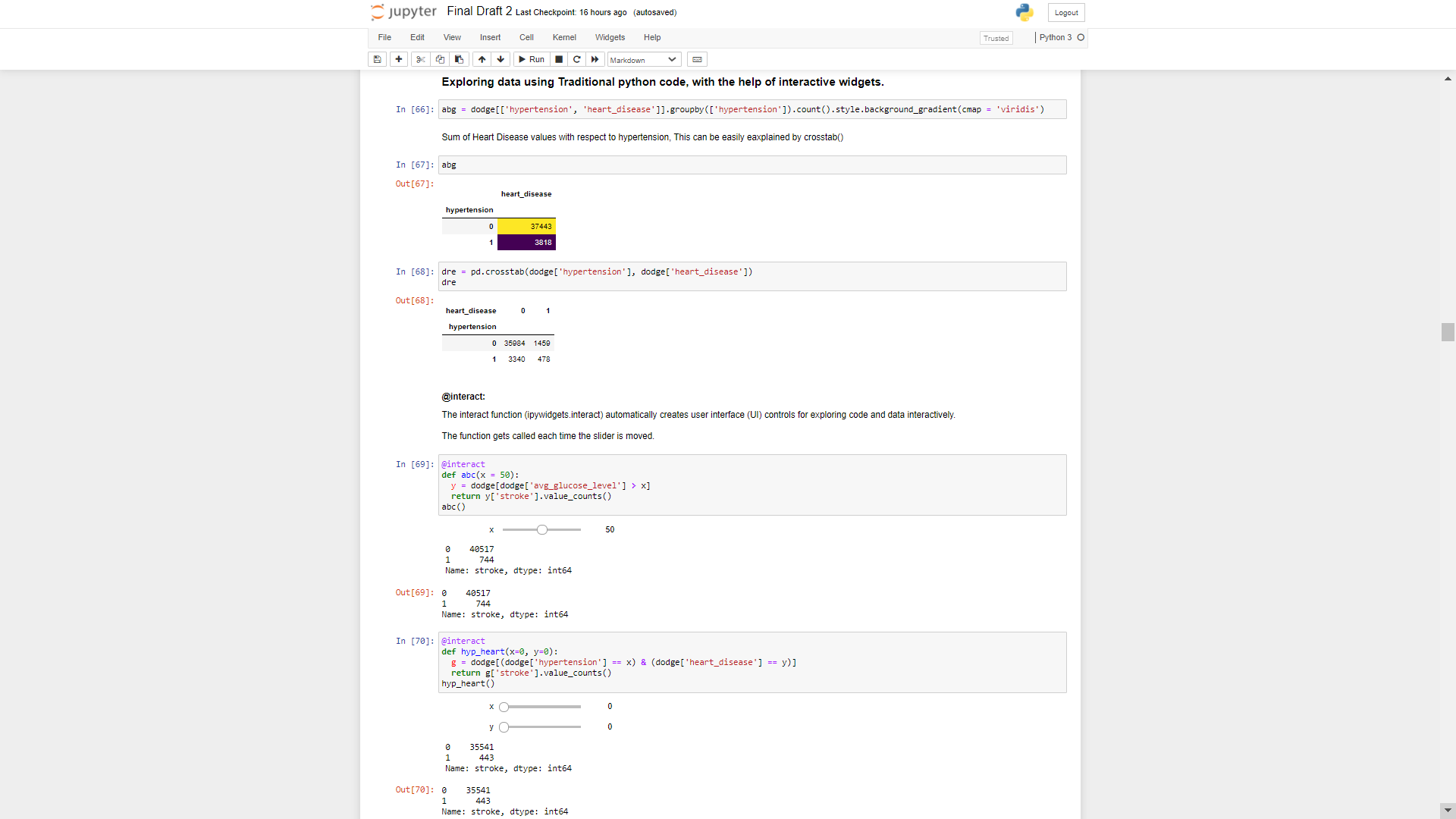
******

9.AutoVisualizing Tool.

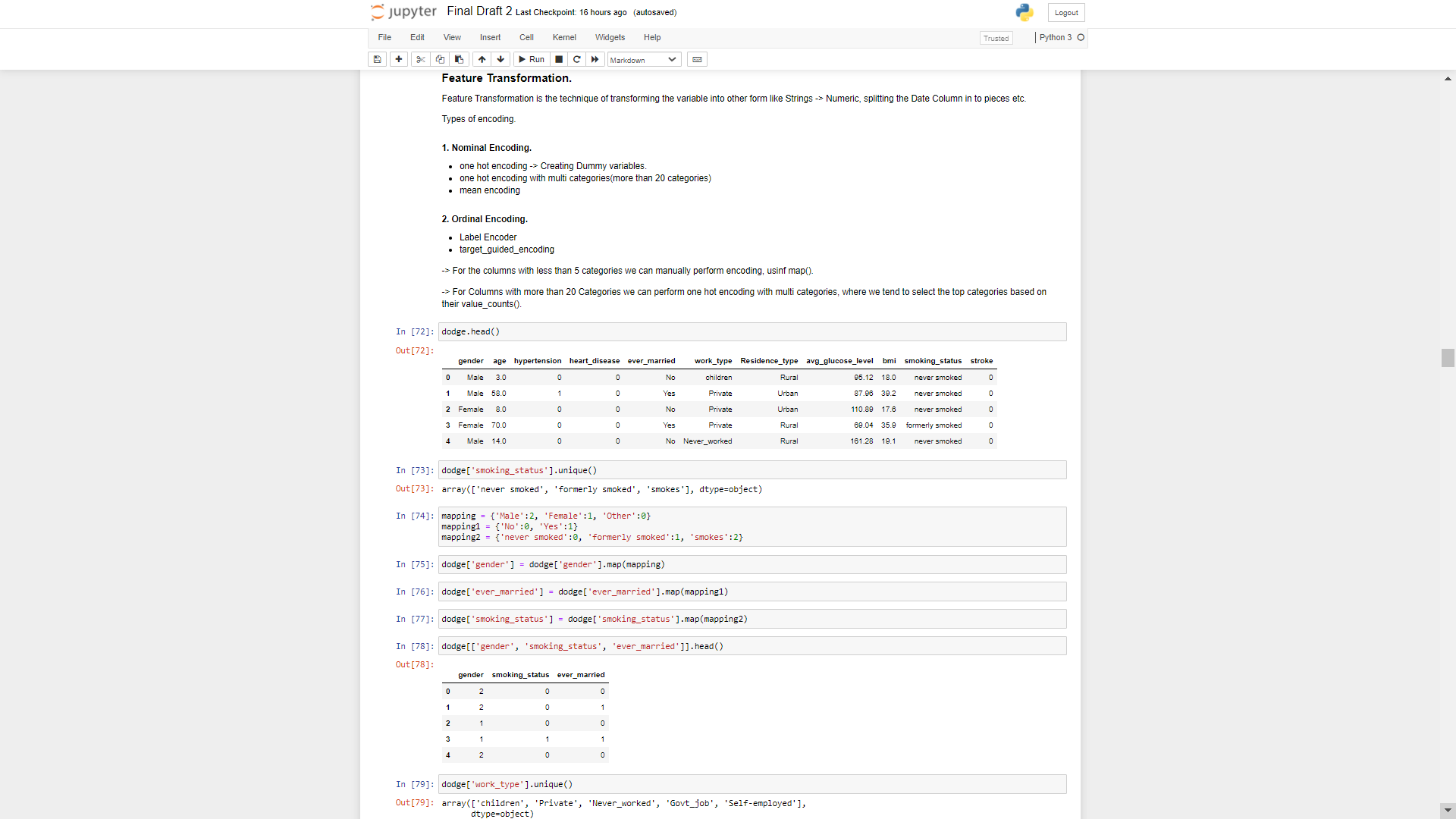
10.Sample Charts.

******

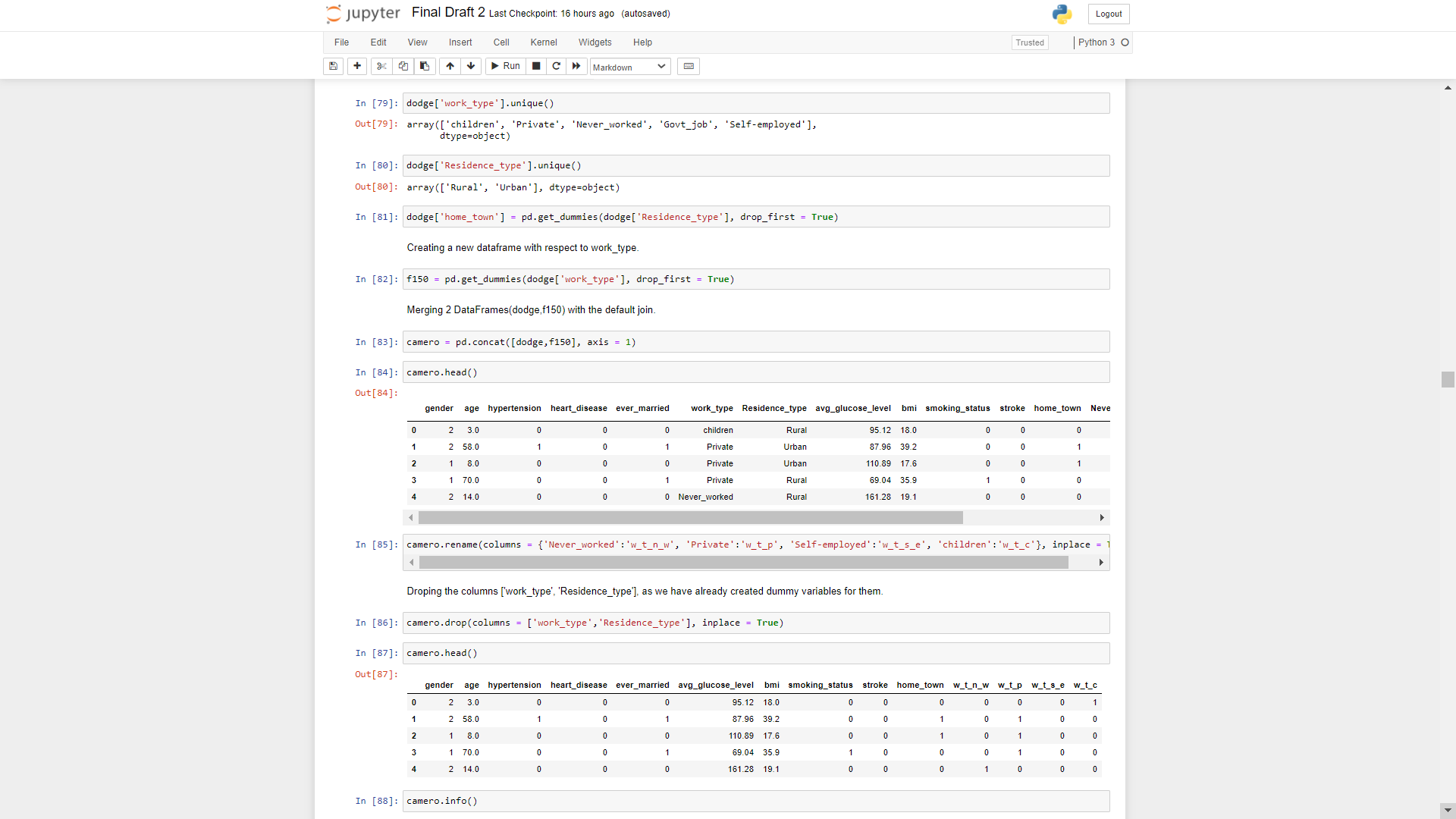
11.Exploring Data Using Interact Manual.

******

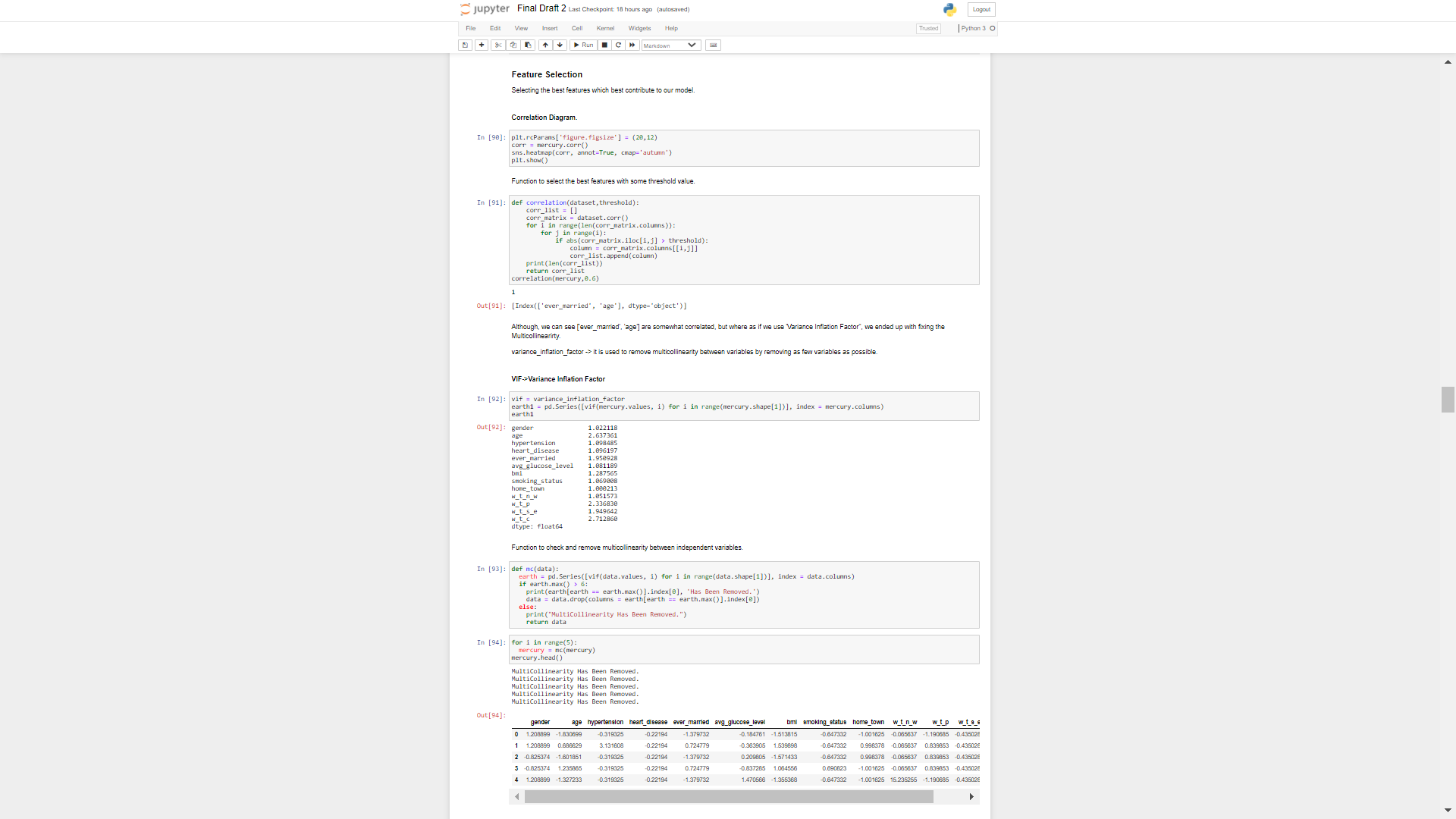
12.Feature Transformation.

******

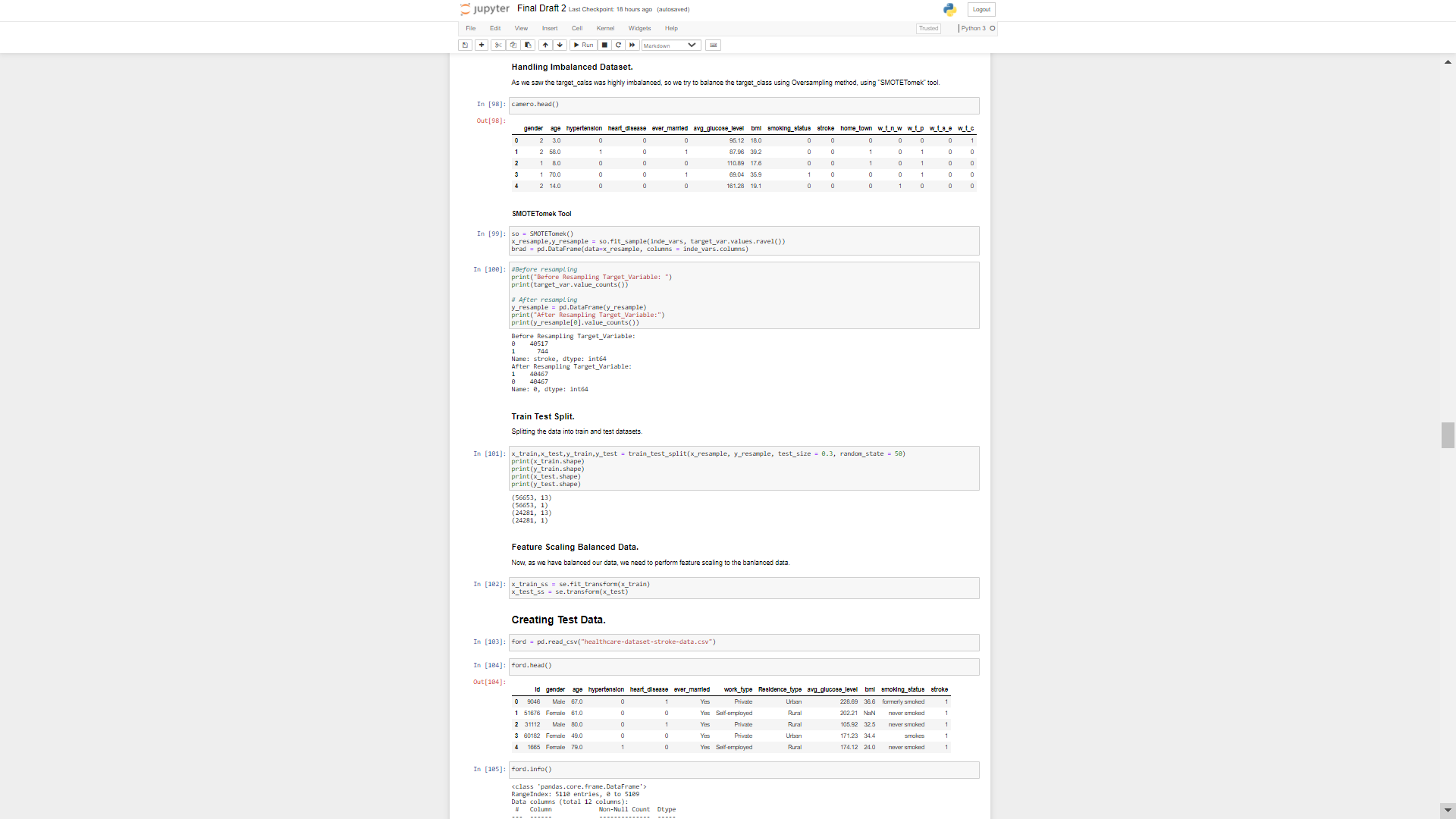
13.Feature Transformation 2.

******

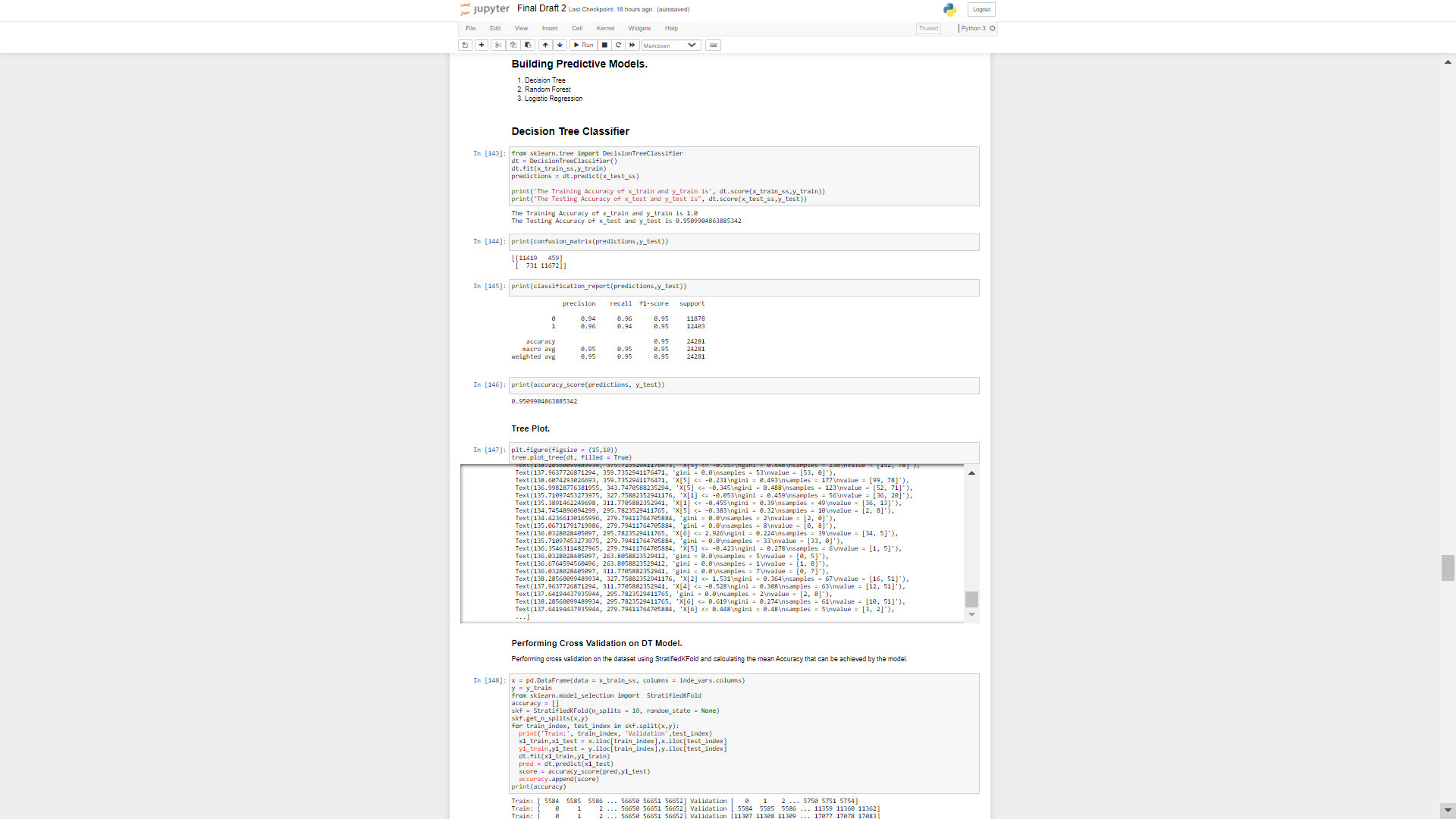
14.Feature Scaling.

******

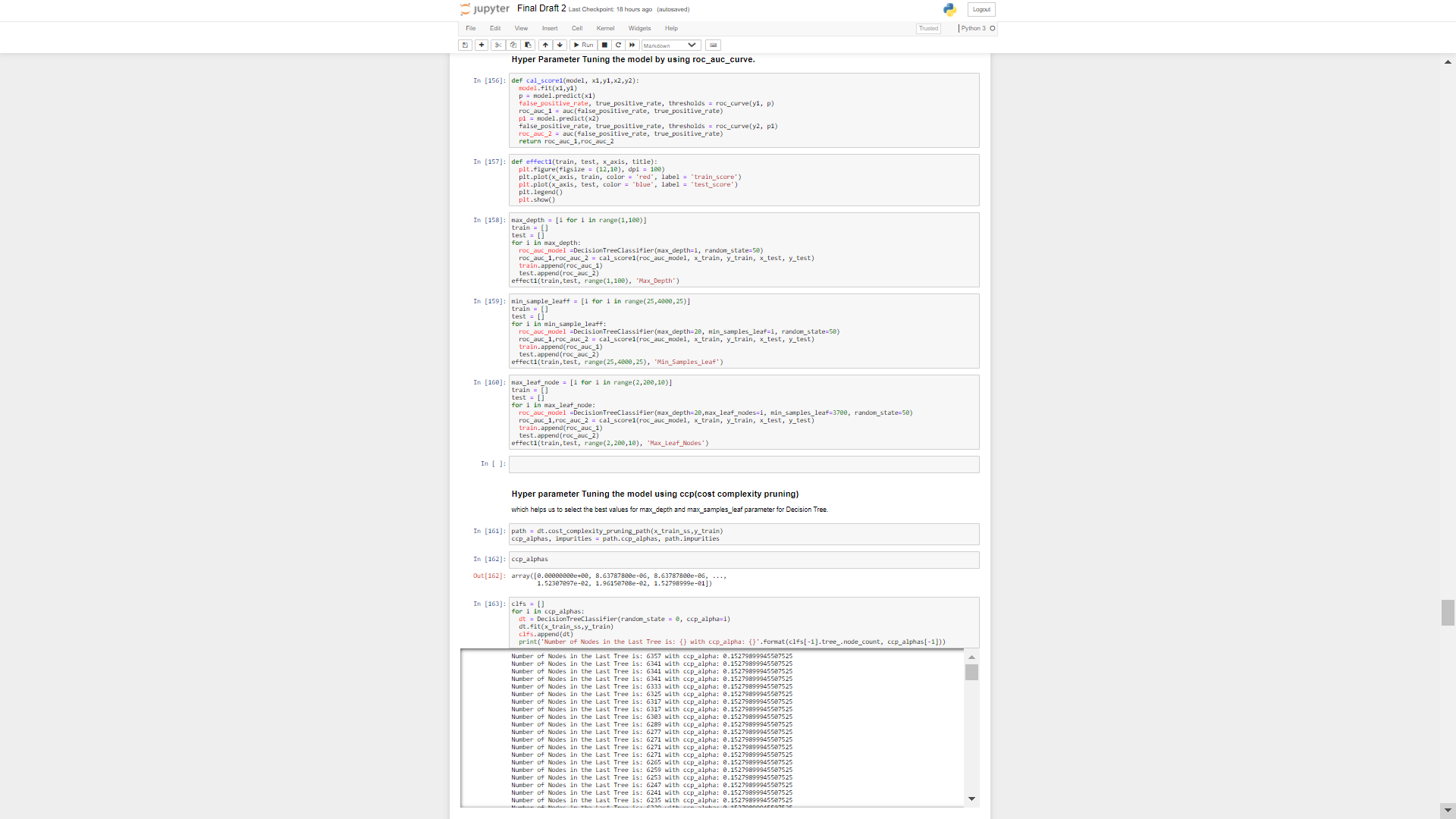
16.Handling Imbalanced Data.

******

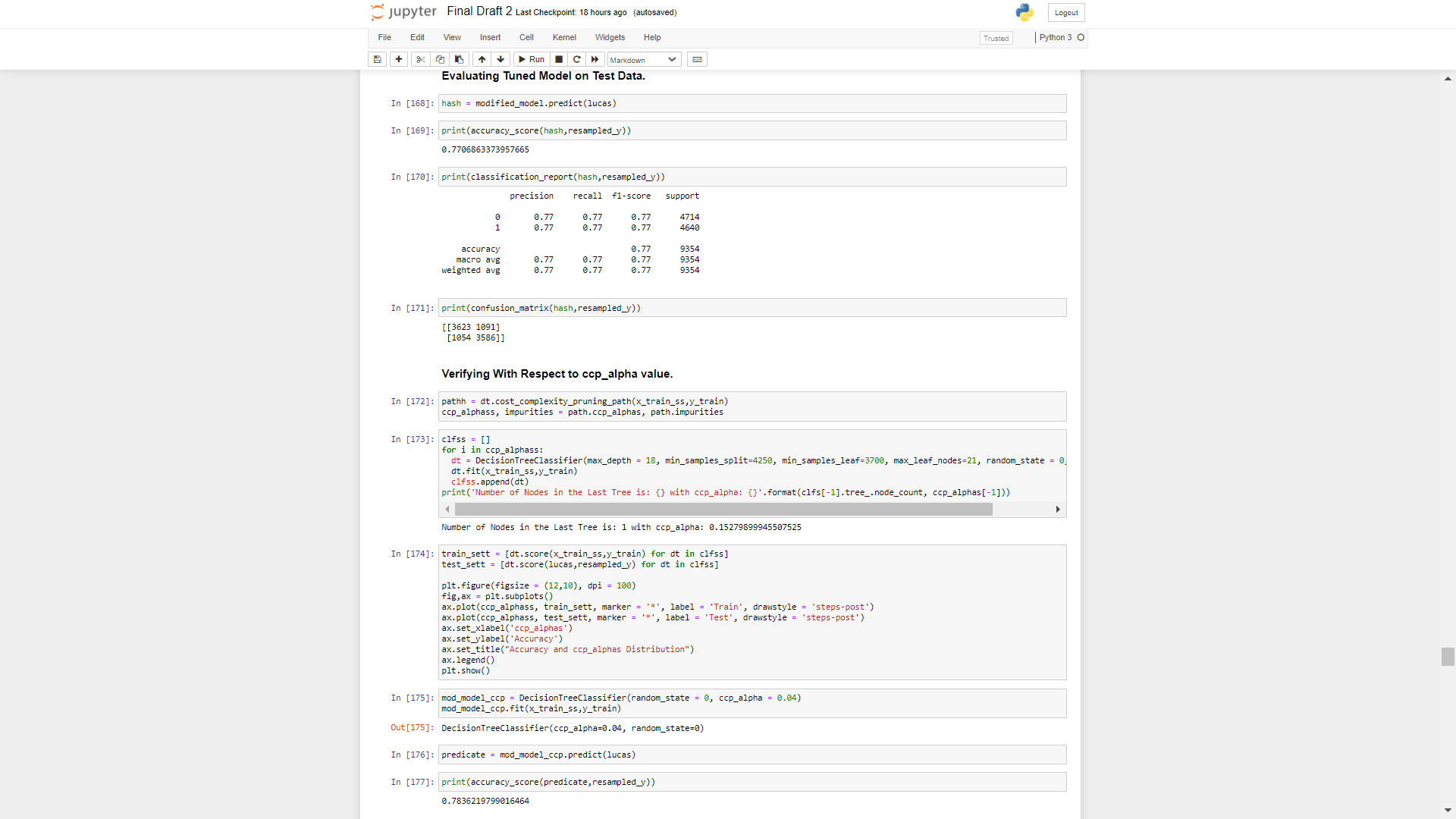
17.Building Predictive Models.



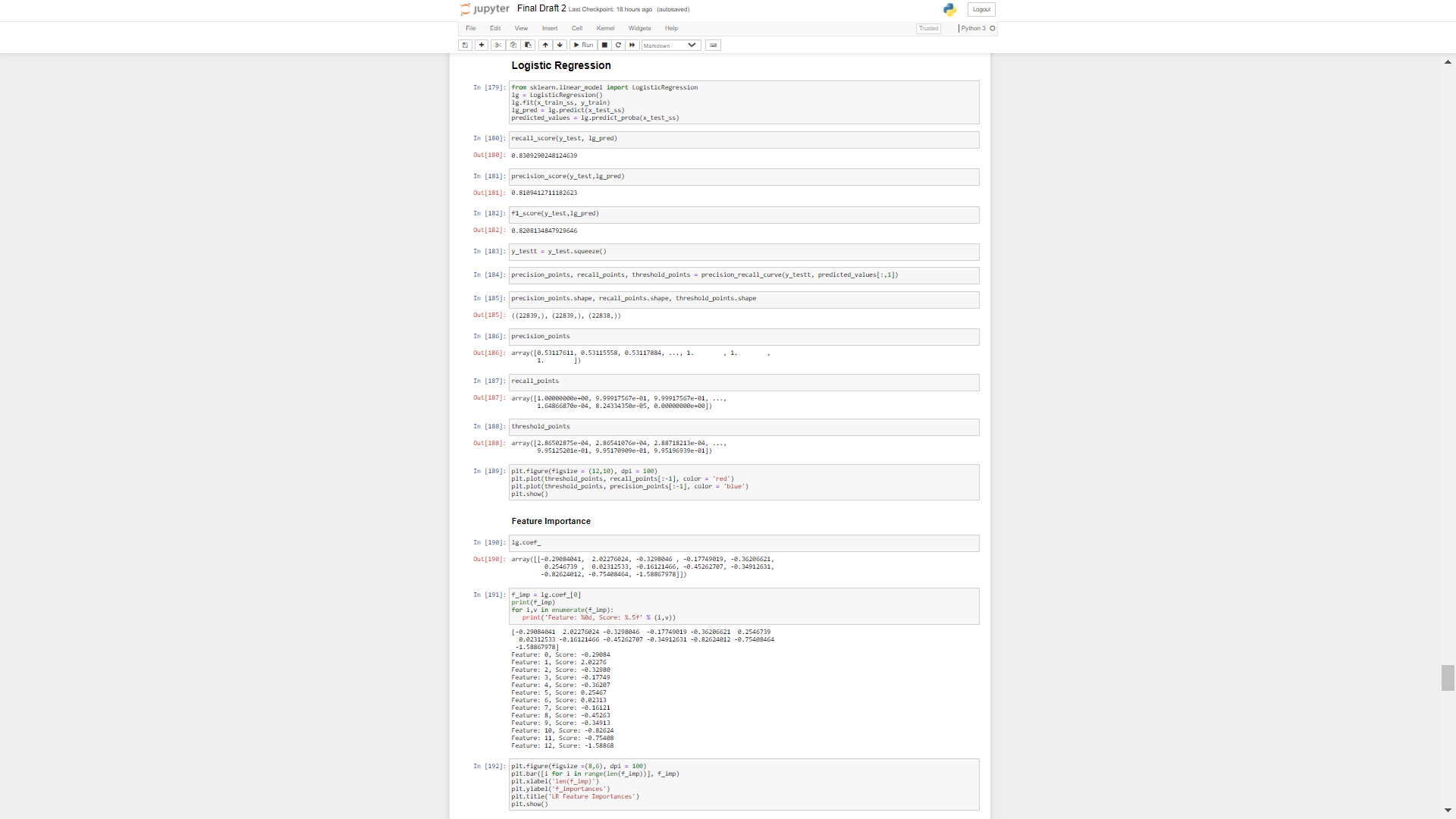
18.Hyper Tuning Decision Tree Model.

******

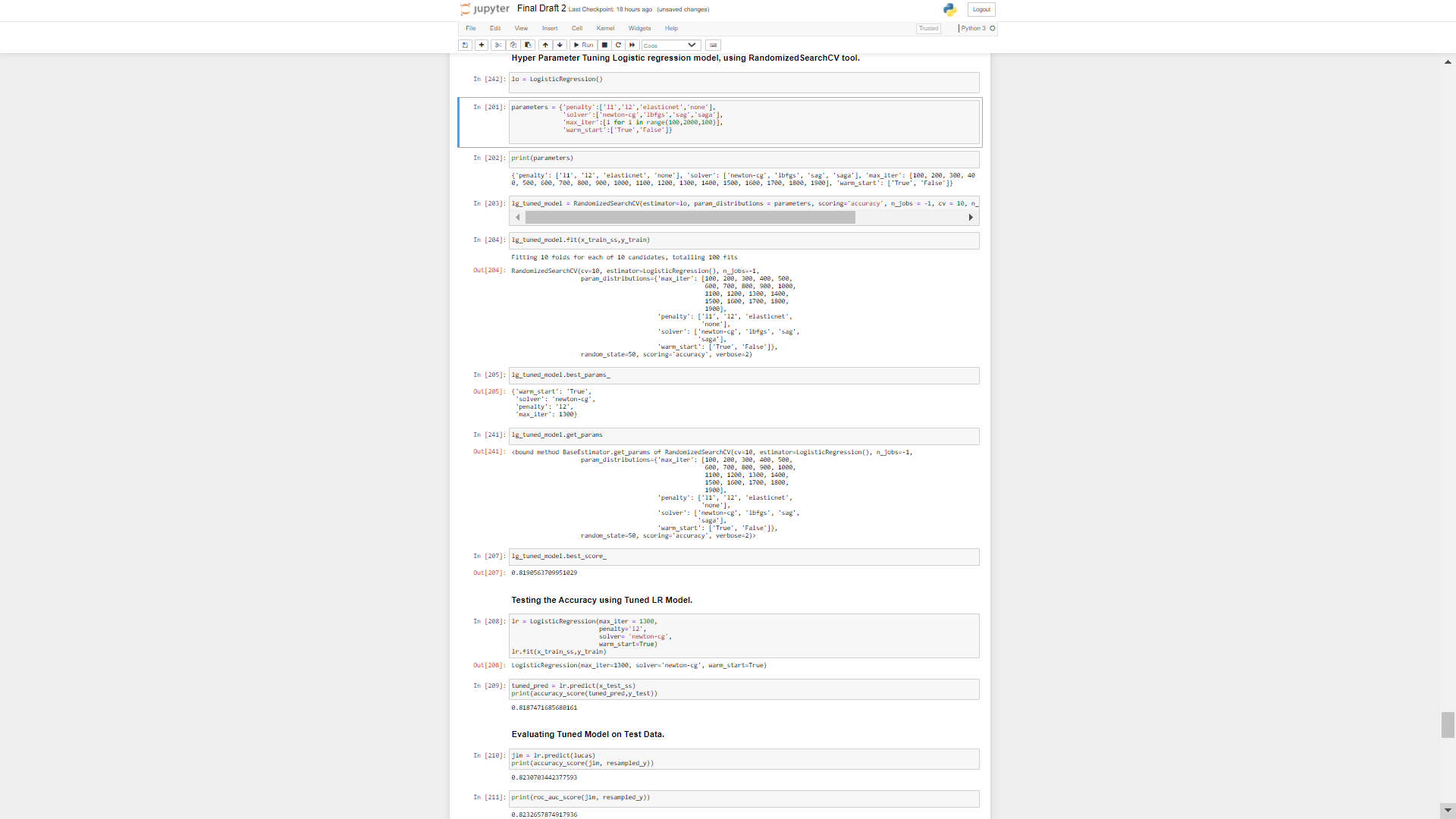
19.Evaluating the Model.

******

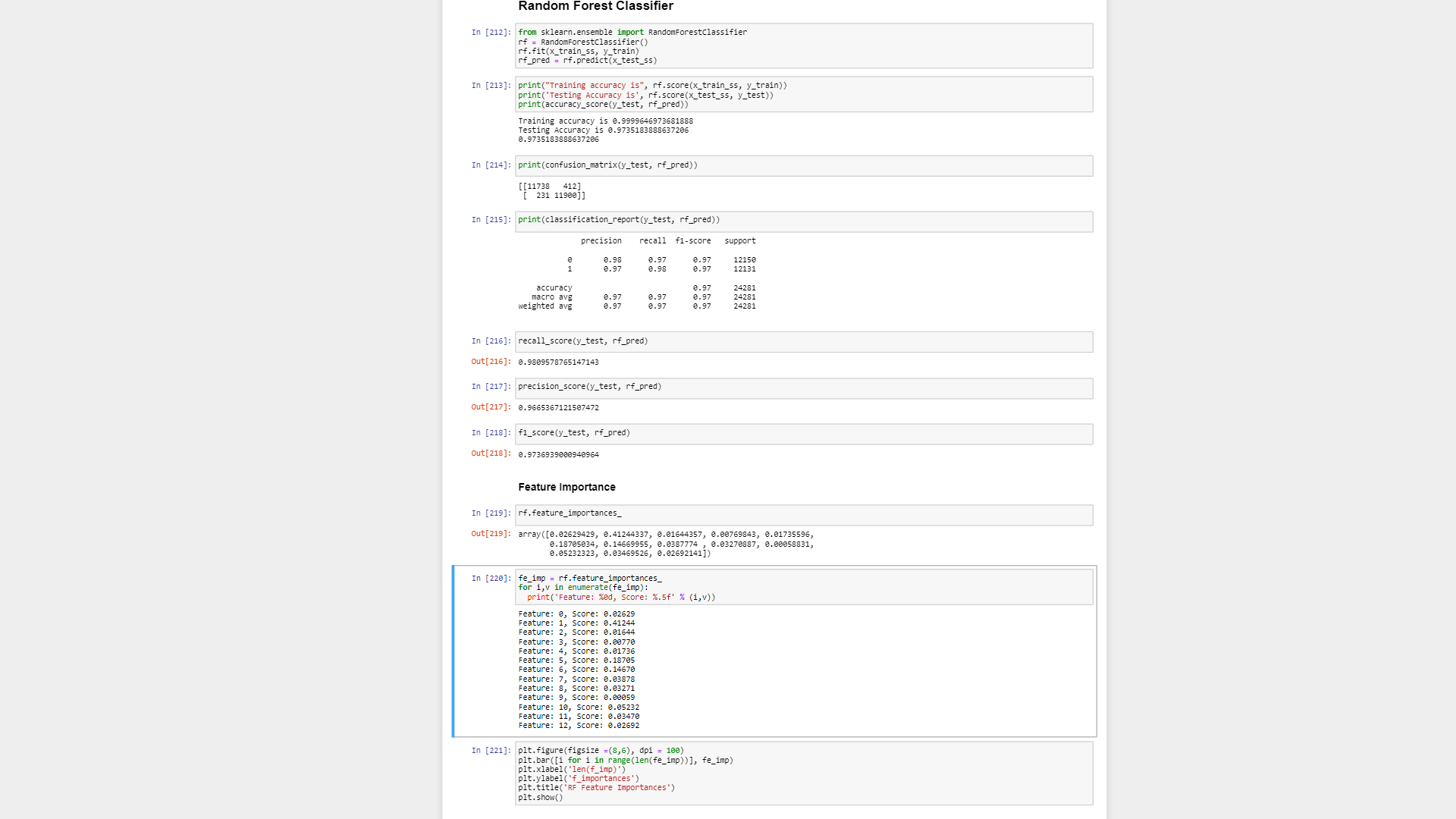
20. Logistic Regression Model.

******

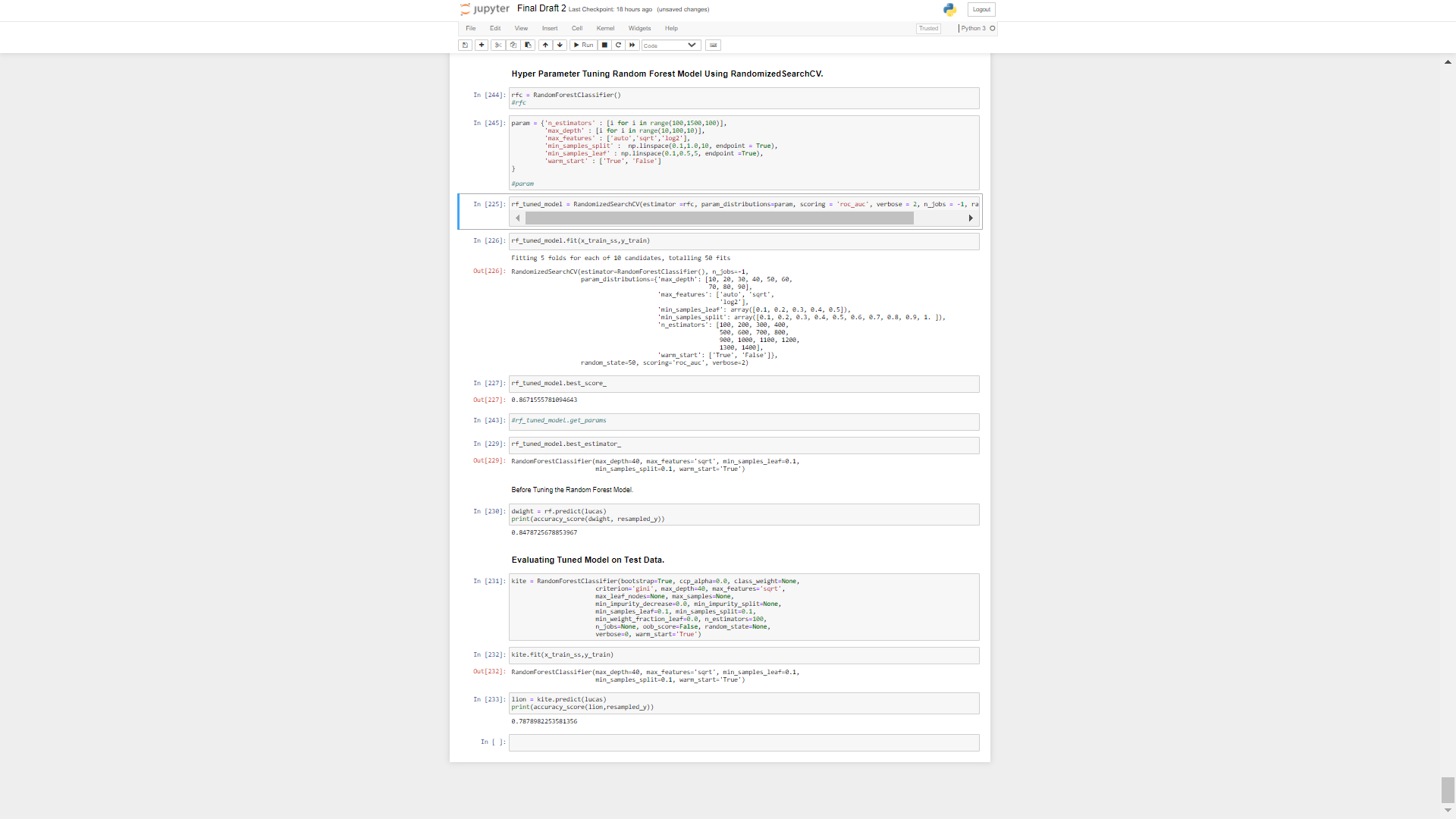
21.Tuning Logistic Regression Model.



22.Random Forest Model.

******

23. Tuning Random Forest Model.

******

**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 ENHANCEMENTS.**

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.com](https://www.medicinenet.com/) [stroke\_symptoms\_and\_treatment/article.htm](https://www.medicinenet.com/stroke_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://stackoverflow.com](https://stackoverflow.com/)