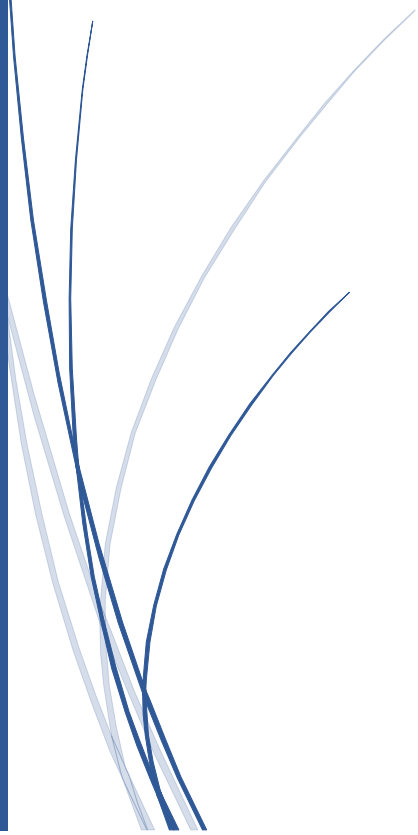
28 April 2022

Data analytics and Visualization of a healthcare dataset



Contents

1. Steps

The basic methods used in this study involve data collection, data preparation, data exploration and data visualization, data modelling and evaluation and applying the model for various use cases.

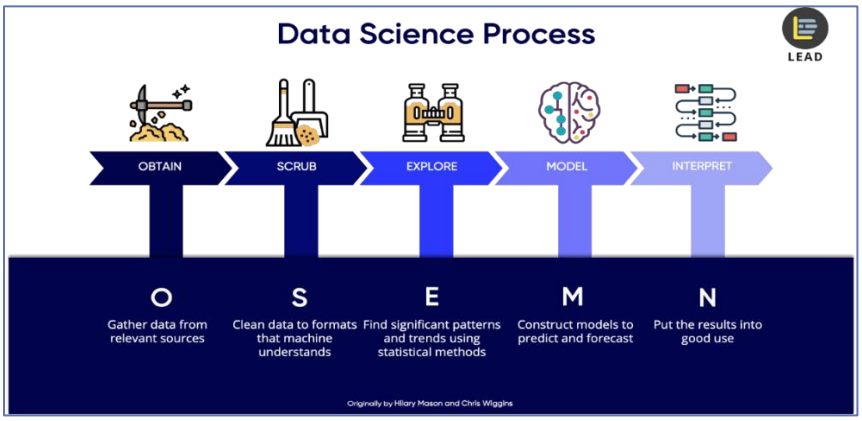


Figure 1: Data Science process flow

1.1 Data collection

The data can be collected either by querying a database, using a web API or just by reading a file in excel or CSV format.

1.1.1 Data source

The raw data for this study i.e., the stroke prediction dataset is downloaded from the Kaggle website as a CSV file.

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

5110 rows × 12 columns

Figure 2: Dataset

1.1.2 Dependent and Independent variables

The “stroke” feature is identified as the target (dependent) variable and the remaining features are independent variables.

Field variables	Field Description	Variable type	Data type	Unique values	Counts & Statistics	
id	Unique value for each person	Independent	Interval	5110 unique values	5110	100%
gender	Sex of the person	Independent	Nominal	Female	2994	58.59%
				Male	2115	41.39%
				Other	1	0.02%
age	Age of the person	Independent	Ratio	104 unique values	5110	100% Min: 0.08 Max: 82 Mean: 43.23 SD: 22.61
hypertension	Whether the person has hypertension or not	Independent	Nominal	0 (Yes)	4612	90.25%
				1 (No)	498	9.75%
heart_disease	Whether the person has heart disease or not	Independent	Nominal	0 (Yes)	4834	94.6%
				1 (No)	276	5.4%
ever_married	Marital status of the person	Independent	Nominal	Yes	3353	65.62%
				No	1757	34.38%
work_type	Employment type of the person	Independent	Nominal	Private	2925	57.24%
				Self-employed	819	16.03%
				Govt_job	657	12.86%
				children	687	13.44%
				Never_worked	22	0.43%
Residence_type	Location of the person's residence	Independent	Nominal	Urban	2596	50.80%
				Rural	2514	49.20%
avg_glucose_level	Average glucose level in the person's body	Independent	Interval	3979 unique values	5110	100% Min: 55.12 Max: 271.74 Mean: 106.15 SD: 45.28
bmi	Body mass index of the person	Independent	Interval	419 unique values and 201 N/A values	5110	100% Min: 10.30 Max: 97.60 Mean: 28.89 SD: 7.7
smoking_status	Whether the person has the habit of smoking or not	Independent	Nominal	formerly smoked	885	17.32%
				never smoked	1892	37.03%
				smokes	789	15.44%
				Unknown	1544	30.21%
stroke	Whether the person has stroke or not	Dependent (Target)	Nominal	0	4861	95.13%
				1	249	4.87%

Figure 3: Dataset details

1.2 Data Preparation (Data Cleaning/Data Wrangling)

The first step is to analyze the data type and the statistics of each feature.

```
#Data size and datatype analysis
print(data.shape)
data.info()
```

```
(5110, 12)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Figure 4: Data size and types

The dataset contains 5110 rows with 12 columns and consists of both numeric and categorical type variables.

Variable Type	Variable Name
Categorical	gender
	hypertension
	heart_disease
	ever_married
	work_type
	Residence_type
	smoking_status
	stroke
Numerical	id
	age
	avg_glucose_level

Figure 5: Categorical and Numerical variables

```
data.describe()
```

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

Figure 6: Statistics of feature variables

The standard deviation values of numerical features are not very high indicating that the values are not too much spread out from the mean value.

```
#Print unique values for the columns with object datatype(categorical columns)
col_list = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']
for col in col_list:
  print(data[col].unique())
```

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

Figure 7: Unique values of Categorical columns

1.2.1 Feature elimination

The column “id” is removed from the dataframe as it does not contain any significant information for the prediction of the target variable

```
#Remove the 'id' column as it is not relevant
data.drop(columns=['id'], inplace=True)
print(list(data.columns))
```

```
['gender', 'age', 'hypertension', 'heart_disease', 'ever_married', 'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
'smoking_status', 'stroke']
```

Figure 8: Feature elimination

1.2.2 Feature Encoding

The categorical variables are converted to numerical variables between 0 and total class count minus 1 using the Label Encoding technique since the machine learning models can work only with numerical values. The LabelEncoder class of the sklearn library is used to perform feature encoding for nominal variables.

```
#Label encoding for categorical column values
from sklearn.preprocessing import LabelEncoder

for i in range(len(col_list)):
    data[col_list[i]] = LabelEncoder().fit_transform(data[col_list[i]])

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             5110 non-null   int32
 1   age                5110 non-null   float64
 2   hypertension       5110 non-null   int64
 3   heart_disease      5110 non-null   int64
 4   ever_married       5110 non-null   int32
 5   work_type          5110 non-null   int32
 6   Residence_type     5110 non-null   int32
 7   avg_glucose_level  5110 non-null   float64
 8   bmi                4909 non-null   float64
 9   smoking_status     5110 non-null   int32
 10  stroke             5110 non-null   int64
dtypes: float64(3), int32(5), int64(3)
memory usage: 339.5 KB
```

Figure 9: Label encoding

Categorical value	Label Encoded value
Male	0
Female	1
Other	2

Figure 10: Label encoding for gender feature

There are other types of encoders like one-hot encoding, frequency encoding etc.

1.2.3 Missing data handling

The missing value ratio of each column showed that about 3.93% of the observations i.e., 201 out of 5110 rows have missing values denoted as “NaN” for the “bmi” column.

```
#Check for missing data
check_missing_data = data[data.isna().any(axis=1)]
percentage_missing = data.isnull().sum()/len(data)*100
print(percentage_missing)
print(f'\nThe {len(check_missing_data)} rows with missing data are listed below\n')
check_missing_data
```

```
gender              0.000000
age                 0.000000
hypertension        0.000000
heart_disease       0.000000
ever_married        0.000000
work_type           0.000000
Residence_type      0.000000
avg_glucose_level   0.000000
bmi                 3.933464
smoking_status      0.000000
stroke              0.000000
dtype: float64

The 201 rows with missing data are listed below
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
1	0	61.0	0	0	1	3	0	202.21	NaN	2	1
8	0	59.0	0	0	1	2	0	76.15	NaN	0	1
13	1	78.0	0	1	1	2	1	219.84	NaN	0	1
19	1	57.0	0	1	0	0	1	217.08	NaN	0	1
27	1	58.0	0	0	1	2	0	189.84	NaN	0	1
...	...	...	...	...	...	...	...	...	...	...	...
5039	1	41.0	0	0	0	2	0	70.15	NaN	1	0
5048	1	40.0	0	0	1	2	1	191.15	NaN	3	0
5093	0	45.0	1	0	1	0	0	95.02	NaN	3	0
5099	1	40.0	0	0	1	2	0	83.94	NaN	3	0
5105	0	80.0	1	0	1	2	1	83.75	NaN	2	0

201 rows × 11 columns

Figure 11: Missing values

Since missing values exist only for the “bmi” feature, instead of removing the entire row, the “NaN” values can be simply replaced with the mean of the column values. Here the missing values are replaced using the “SimpleImputer” class of the sklearn library with the “mean” value (28.893237) as the imputation strategy. Other imputation strategies include “mean”, “median”, “most-frequent” or “constant”.

```
#Missing value handling using simple imputation
from sklearn.impute import SimpleImputer

new_headers = ['gender','age','hypertension','heart_disease','ever_married','work_type',
           'Residence_type','avg_glucose_level','bmi','smoking_status','stroke']

data.reset_index(drop=True, inplace=True)

imputer = SimpleImputer(strategy='mean')
imputer.fit(data)
new_data = imputer.transform(data)
data = pd.DataFrame(new_data, columns=new_headers)

#Recheck to ensure the missing data replacement
check_missing_data = data[data.isna().any(axis=1)]
print(f'\nThere are {len(check_missing_data)} rows with missing data.\n', check_missing_data)
data
```

```
There are 0 rows with missing data.
 Empty DataFrame
Columns: [gender, age, hypertension, heart_disease, ever_married, work_type, Residence_type, avg_glucose_level, bmi, smoking_st
atus, stroke]
Index: []
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1.0	67.0	0.0	1.0	1.0	2.0	1.0	228.69	36.600000	1.0	1.0
1	0.0	61.0	0.0	0.0	1.0	3.0	0.0	202.21	28.893237	2.0	1.0
2	1.0	80.0	0.0	1.0	1.0	2.0	0.0	105.92	32.500000	2.0	1.0
3	0.0	49.0	0.0	0.0	1.0	2.0	1.0	171.23	34.400000	3.0	1.0
4	0.0	79.0	1.0	0.0	1.0	3.0	0.0	174.12	24.000000	2.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
5105	0.0	80.0	1.0	0.0	1.0	2.0	1.0	83.75	28.893237	2.0	0.0
5106	0.0	81.0	0.0	0.0	1.0	3.0	1.0	125.20	40.000000	2.0	0.0
5107	0.0	35.0	0.0	0.0	1.0	3.0	0.0	82.99	30.600000	2.0	0.0
5108	1.0	51.0	0.0	0.0	1.0	2.0	0.0	166.29	25.600000	1.0	0.0
5109	0.0	44.0	0.0	0.0	1.0	0.0	1.0	85.28	26.200000	0.0	0.0

5110 rows × 11 columns

Figure 12: Mean imputation

1.3 Exploratory Data Analysis and Data Visualization

1.3.1 Univariate analysis

Univariate analysis is used to perform an analysis of a single variable at a time.

Histogram

Histogram analysis can reveal important information regarding the frequency and distribution of data. It is observed that most of the categorical features especially “stroke” has imbalanced data while the numerical features seem to have a skewed normal distribution.

```
#Histogram of the dataframe columns
data.hist(figsize=(10, 10))
```

Figure 13: Histogram-plotting

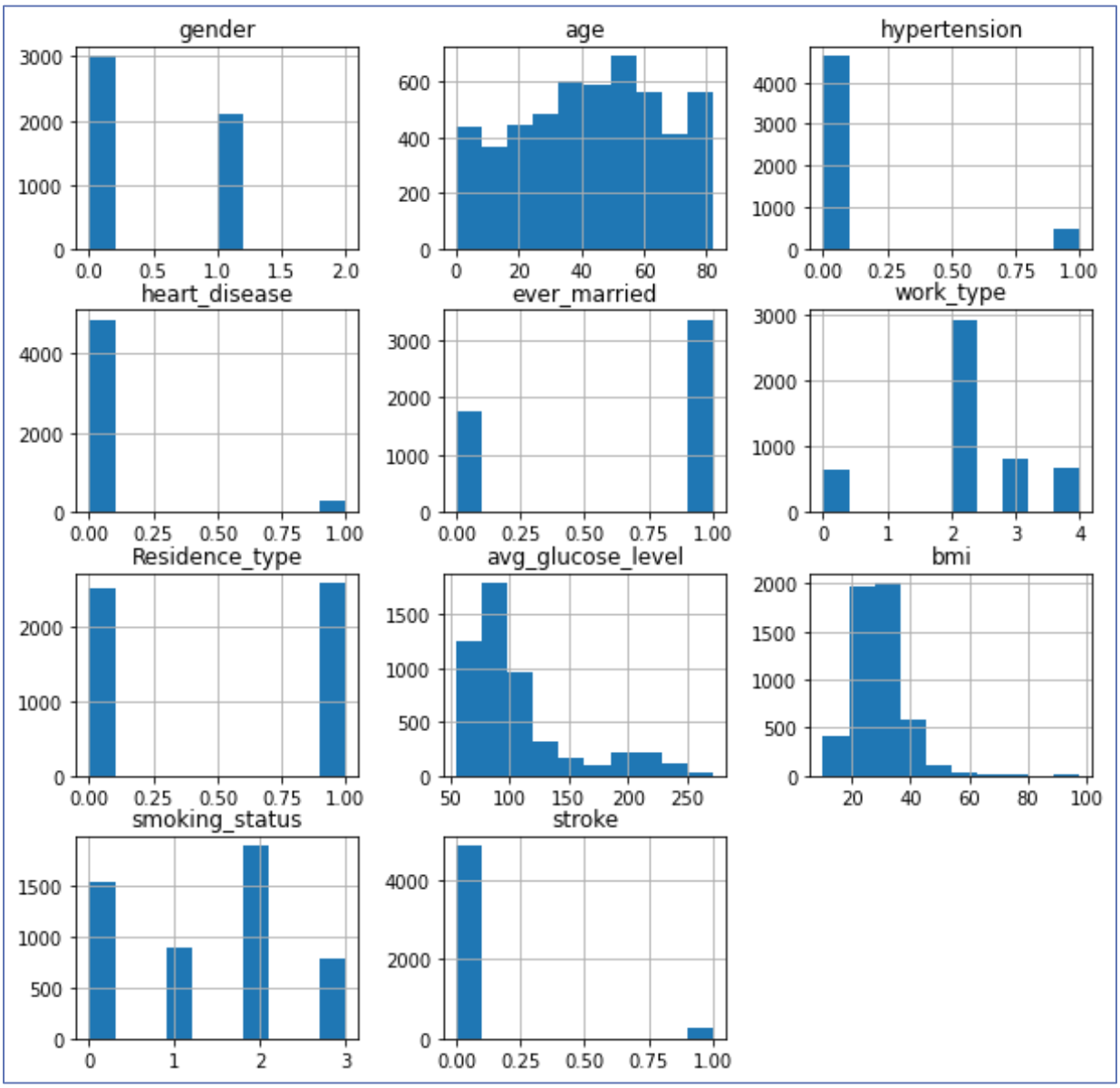


Figure 14: Histogram plot

Box-Whisker plots

Boxplots display the data distribution based on “minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum” values. It can be used to observe the outliers, symmetry, and skewness of data.

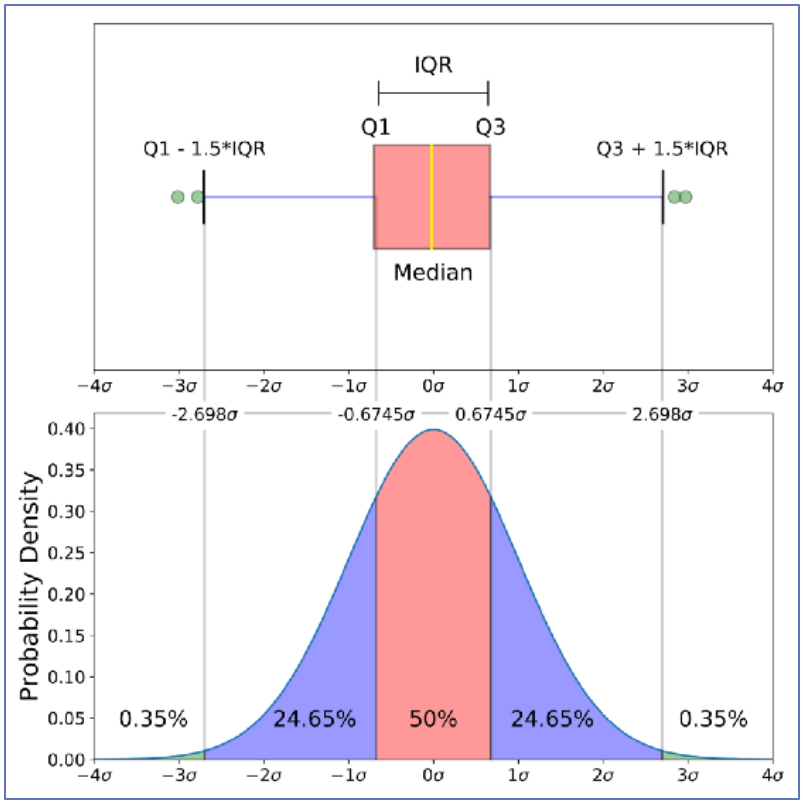


Figure 15: Box-plot interpretation

```
#perform uni-variate analysis using boxplot
data.plot(kind='box', subplots= True, layout = (3,4), sharex =False, figsize=(15, 8))
```

Figure 16: Box-whisker plotting

It is observed that the variables “avg_glucose_level” and “bmi” have many outliers beyond the maximum value.

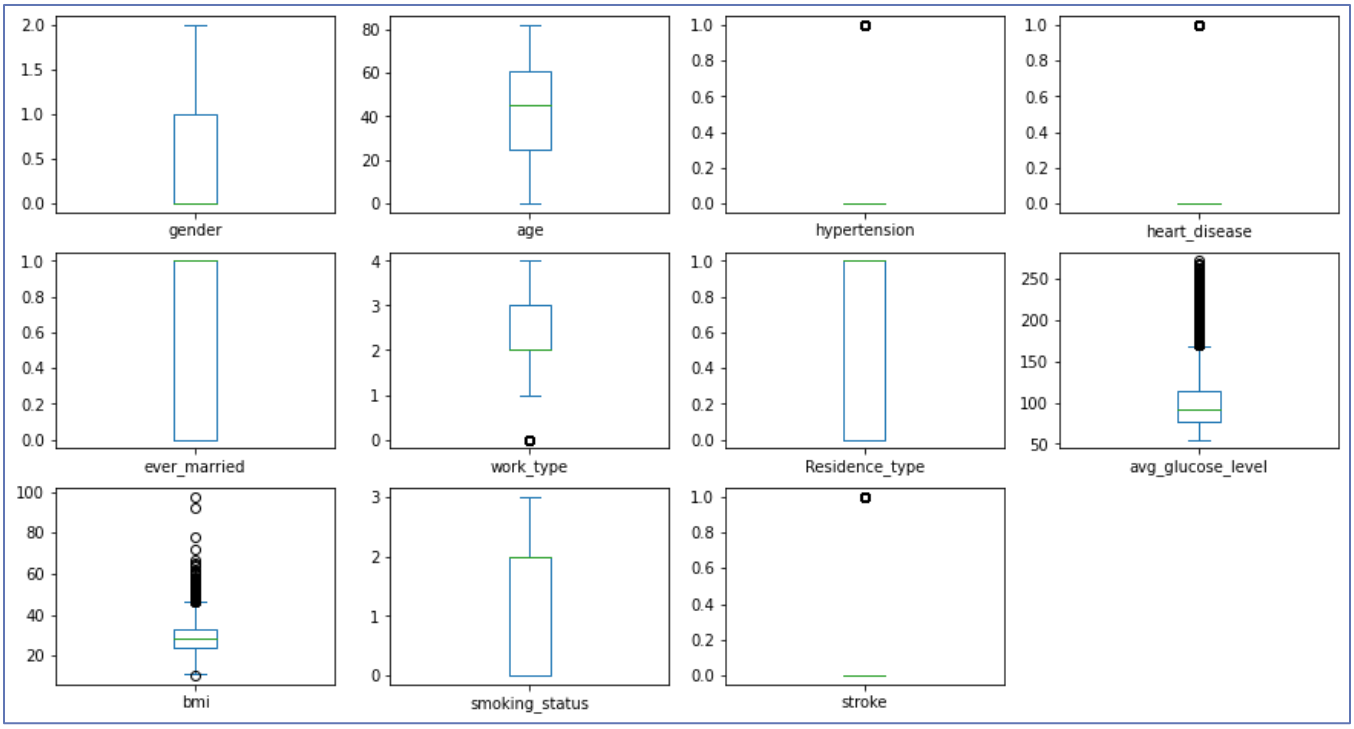


Figure 17: Box-whisker plot

Density plots

The density plots are Kernel Density Estimate plots using Gaussian kernels. Here, it appears to be a smoothened version of the histogram and does not provide additional information.

```
#density plot
data.plot(kind='density', subplots= True, layout = (4,3), sharex =False, figsize=(20, 10))
```

Figure 18: Density plotting

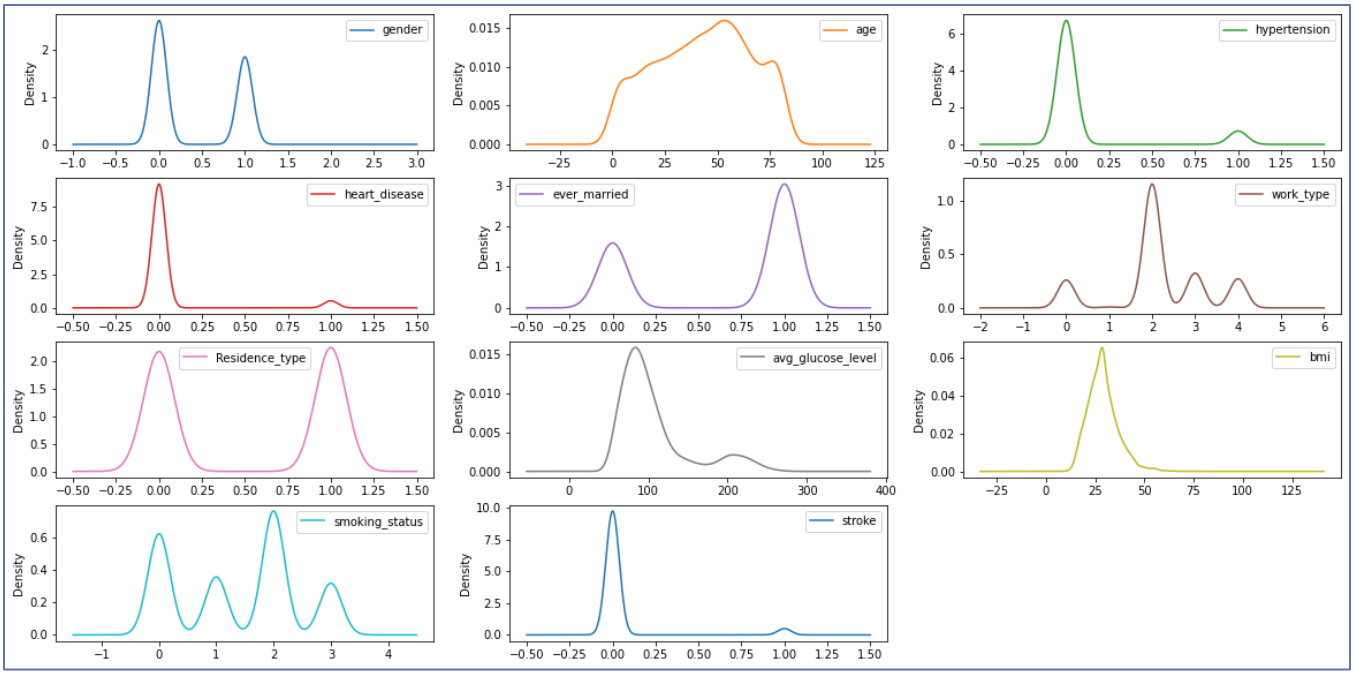


Figure 19: Density plot

Skewness and Kurtosis

Skewness is the measure of symmetry whereas kurtosis measures the peakedness of the data concerning a normal distribution.

```
print("Skewness:\n", data.skew())
print("\nKurtosis:\n", data.kurtosis())
```

```
Skewness:
 gender              0.353012
age                -0.137059
hypertension        2.715392
heart_disease       3.947244
ever_married       -0.657745
work_type          -0.308617
Residence_type     -0.032107
avg_glucose_level   1.572284
bmi                 1.076716
smoking_status     -0.039234
stroke              4.193284
dtype: float64

Kurtosis:
 gender             -1.862882
age                -0.991010
hypertension        5.375456
heart_disease      13.586052
ever_married       -1.567985
work_type           0.144290
Residence_type     -1.999752
avg_glucose_level   1.680479
bmi                 3.623061
smoking_status     -1.317830
stroke             15.589736
dtype: float64
```

Figure 20: Skewness-Kurtosis analysis

Feature	Skewness	Kurtosis
avg_glucose_level	Positively skewed, Mode shifted to left	Platykurtic, Thinner tails
bmi	Positively skewed, Mode shifted to left	Leptokurtic, Thicker tails

Figure 21: Skewness and Kurtosis

1.3.2 Multivariate analysis

Multivariate analysis is used to perform analysis of more than one variable at a time.

The Scatter-Facet plot

The scatter facet plot using the features “age”, “bmi”, “smoking_status”, “stroke” and “gender” is implemented using the Plotly library.

```
import plotly.express as px
fig = px.scatter(csv_data, x="age", y="bmi", color="smoking_status", facet_row="stroke", facet_col="gender" )
fig.show()
```

Figure 22: Scatter-Facet plotting

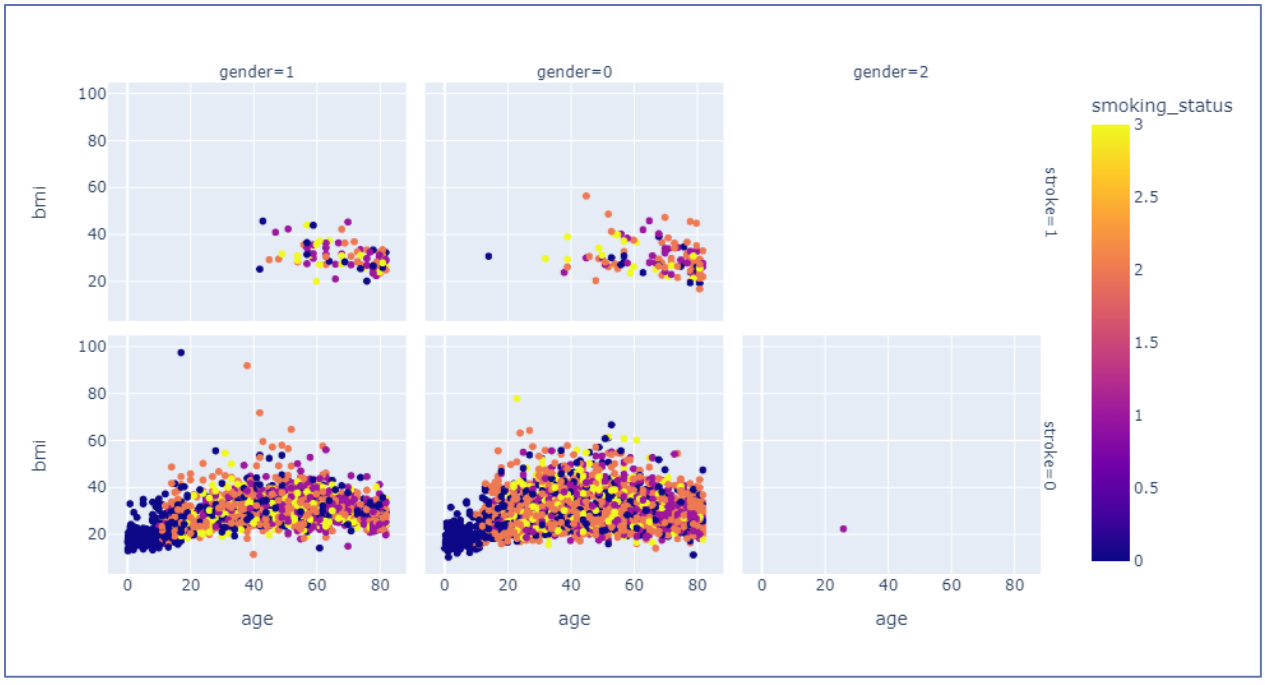


Figure 23: Scatter-Facet plot

Features	High Risk values (stroke =1)
gender	Female
age	>40 years
bmi	>25
smoking_status	formerly smoked', 'never smoked' or 'smokes'

Figure 24: Observations

Tableau dashboards

Tableau is an advanced data visualization tool used to create dashboards containing multiple views to compare a variety of data simultaneously.

The pie chart shows the number of stroke and healthy patients while the grouped bar chart represents hypertensive and non-hypertensive patients grouped by gender. The butterfly (diverging bar) chart depicts the number of patients with heart disease and hypertension in each age group. The average blood glucose level and the BMI values are denoted by the bubble chart and the tree-map, respectively.

Figure 25: Dashboard using Tableau

Power BI dashboards

Microsoft's Power BI is also a powerful tool for interactive data visualization and reports for business intelligence.

The pie-chart and tree-map in the dashboard show the count grouped by marital status and employment type, respectively. The gender-wise count of people with different smoking habits is indicated in the bottom stacked bar chart.

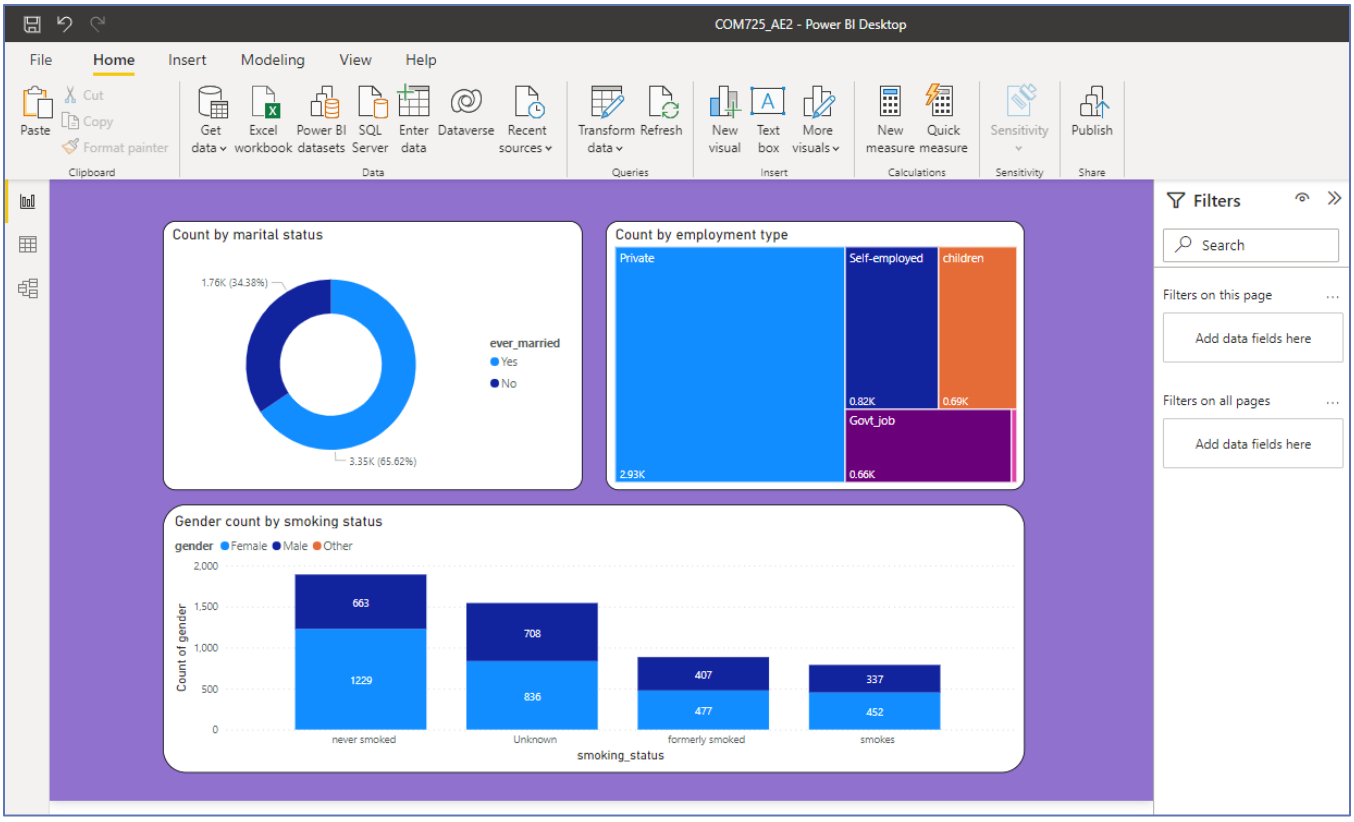


Figure 26: Dashboard using Power BI

1.3.3 Test for Normality

IBM SPSS Statistics is a tool for data processing, advanced analytics, and business intelligence. This tool has been used to perform the normality tests on the continuous values "avg_glucose_level" and "bmi".

Generally, the Kolmogorov-Smirnov Goodness of Fit Test (K-S test) is for large sample sizes (n ≥ 50) and the Shapiro-Wilk test for small sample sizes (< 50).

As the "Sig" column value is small (e.g., under .05 for a 5% alpha level), the null hypothesis which states that the data is normally distributed can be rejected. This means the data is not normally distributed.

Tests of Normality

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
avg_glucose_level	.183	5110	.000
SMEAN(bmi)	.073	5110	<.001

a. Lilliefors Significance Correction

Figure 27: KS Test - Sig values

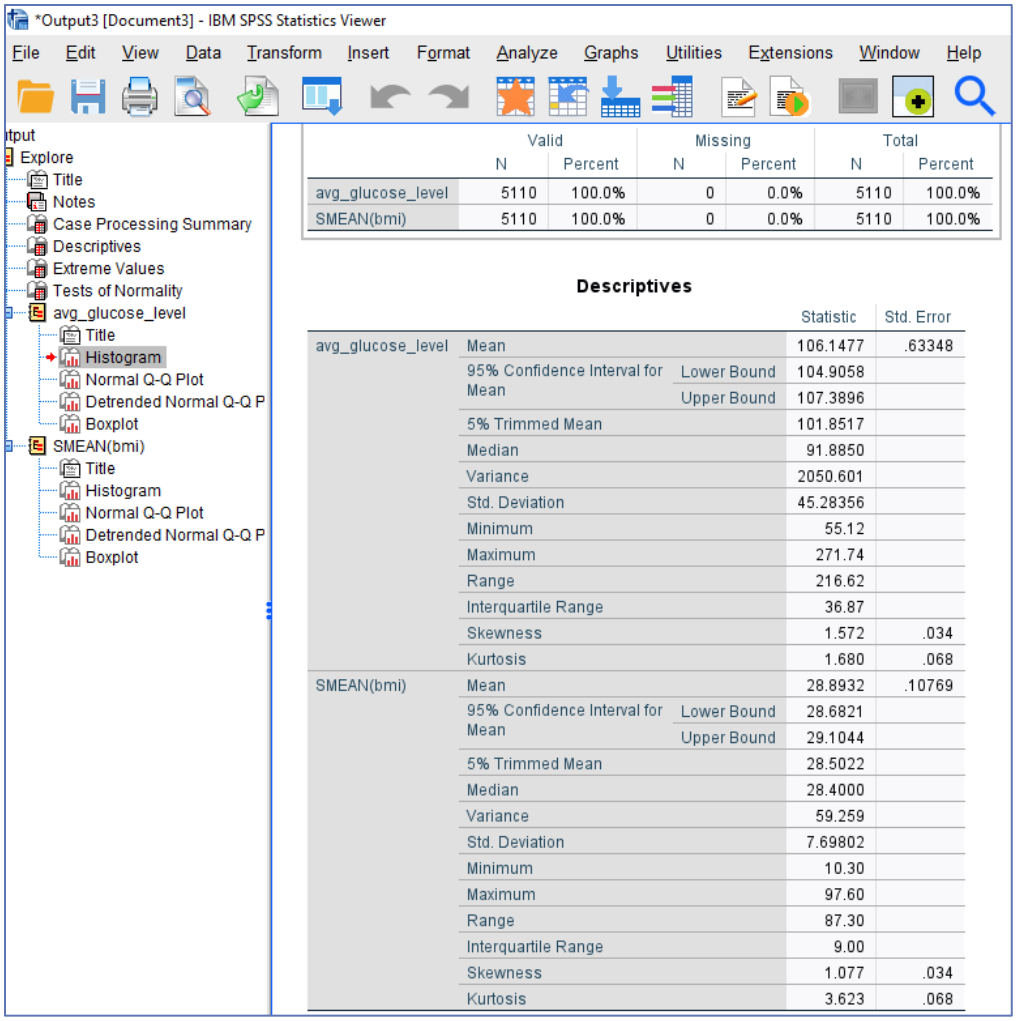


Figure 28: KS Test Descriptives

1.3.4 Feature correlation

The correlation matrix is used to determine the correlation between different features. The value ranges between -1 and +1 where the absolute value determines the strength of correlation and the sign represents the direction of correlation, positive or negative.

```
#Feature correlation
import seaborn as sns

#Correlation matrix
data_corr = data.corr()
data_corr
```

Figure 29: Correlation-matrix

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
gender	1.000000	-0.028202	0.020994	0.085447	-0.031005	0.056422	-0.006738	0.055180	-0.026109	-0.062581
age	-0.028202	1.000000	0.276398	0.263796	0.679125	-0.361642	0.014180	0.238171	0.325942	0.265199
hypertension	0.020994	0.276398	1.000000	0.108306	0.164243	-0.051761	-0.007913	0.174474	0.160189	0.111038
heart_disease	0.085447	0.263796	0.108306	1.000000	0.114644	-0.028023	0.003092	0.161857	0.038899	0.048460
ever_married	-0.031005	0.679125	0.164243	0.114644	1.000000	-0.352722	0.006261	0.155068	0.335705	0.259647
work_type	0.056422	-0.361642	-0.051761	-0.028023	-0.352722	1.000000	-0.007316	-0.050513	-0.299448	-0.305927
Residence_type	-0.006738	0.014180	-0.007913	0.003092	0.006261	-0.007316	1.000000	-0.004946	-0.000120	0.008237
avg_glucose_level	0.055180	0.238171	0.174474	0.161857	0.155068	-0.050513	-0.004946	1.000000	0.168751	0.063437
bmi	-0.026109	0.325942	0.160189	0.038899	0.335705	-0.299448	-0.000120	0.168751	1.000000	0.219148
smoking_status	-0.062581	0.265199	0.111038	0.048460	0.259647	-0.305927	0.008237	0.063437	0.219148	1.000000
stroke	0.008929	0.245257	0.127904	0.134914	0.108340	-0.032316	0.015458	0.131945	0.038947	0.028123

Figure 30: Correlation-matrix values

Correlation heatmap

Heatmaps are used to visualize the correlation matrix in a better way using colour codes. There exists a significant correlation between the features “age” and “ever_married”.

```
#Correlation heatmap using matplotlib library
corr_fig = plt.figure(figsize=(10, 8))
axes = corr_fig.add_subplot()
axescorr = axes.matshow(data_corr, vmin=-1, vmax=1)
corr_fig.colorbar(axescorr)
ticks = np.arange(0, len(data.columns), 1)
axes.set_xticks(ticks)
axes.set_yticks(ticks)
axes.set_xticklabels(new_headers)
# Rotate the tick labels and set their alignment.
plt.setp(axes.get_xticklabels(), rotation=45, ha="left", va="top",
         rotation_mode="anchor")
axes.set_yticklabels(new_headers)
plt.show()

#Correlation heatmap using seaborn
plt.subplots(figsize=(10, 8))
sns.heatmap(data_corr, linewidths=.1, annot=True, fmt="0.2f", vmin=0, vmax=1, cmap='coolwarm' )
plt.show()
```

Figure 31: Heatmaps

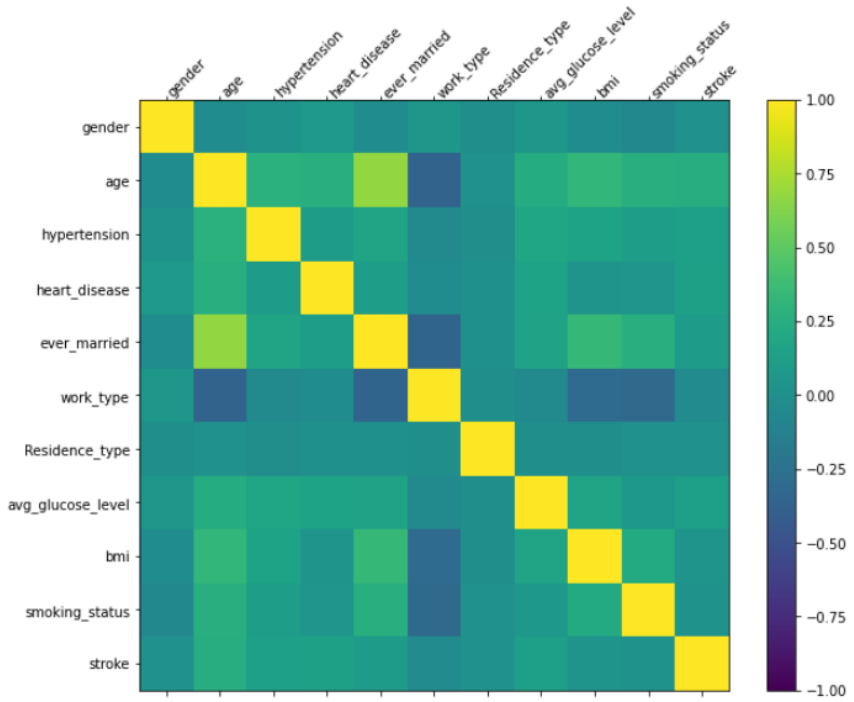


Figure 32: Correlation heatmap using matplotlib

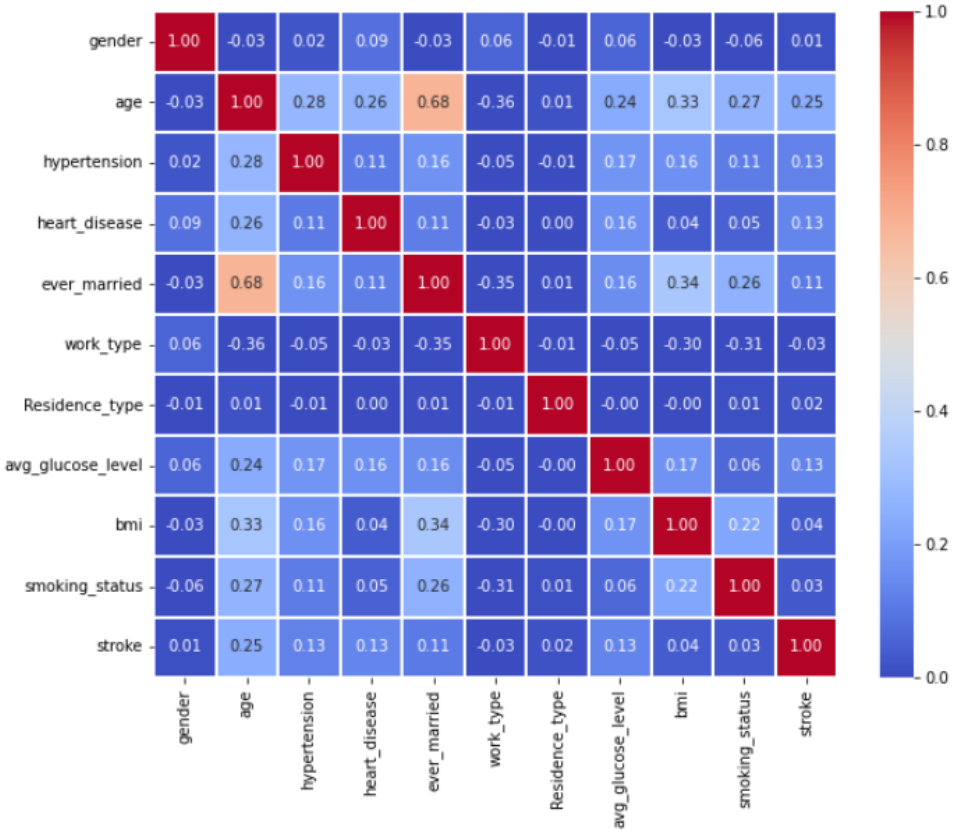


Figure 33: Correlation heatmap using seaborn

1.3.5 Feature importance

The relative importance of the feature in the target value prediction is calculated using three different methods i.e., RandomForestClassifier, XGBClassifier and LGBMClassifier.

```
#Feature importances using RandomForestClassifier(sklearn), XGBClassifier(xgboost) and LGBMClassifier(lightgbm)
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import matplotlib.pyplot as plt

feature_data = data.drop(columns=['stroke'])
def plot_feature_importances(clf):
    clf.fit(feature_data, data.stroke)
    train_features = feature_data.columns
    importances = clf.feature_importances_
    indices = np.argsort(importances)[-10:]
    print(indices)
    plt.title(f"Feature Importance using {clf.__class__.__name__}")
    plt.barh(range(len(indices)), importances[indices], color="teal", align="center")
    plt.yticks(range(len(indices)), [train_features[i] for i in indices])
    plt.xlabel("Relative Importance")
    plt.show()

clfs = [RandomForestClassifier(random_state=1, max_depth=4), XGBClassifier(use_label_encoder=False), LGBMClassifier()]
for clf in clfs:
    try:
        plot_feature_importances(clf)
    except AttributeError as e:
        print(e)
```

Figure 34: Feature importance analysis

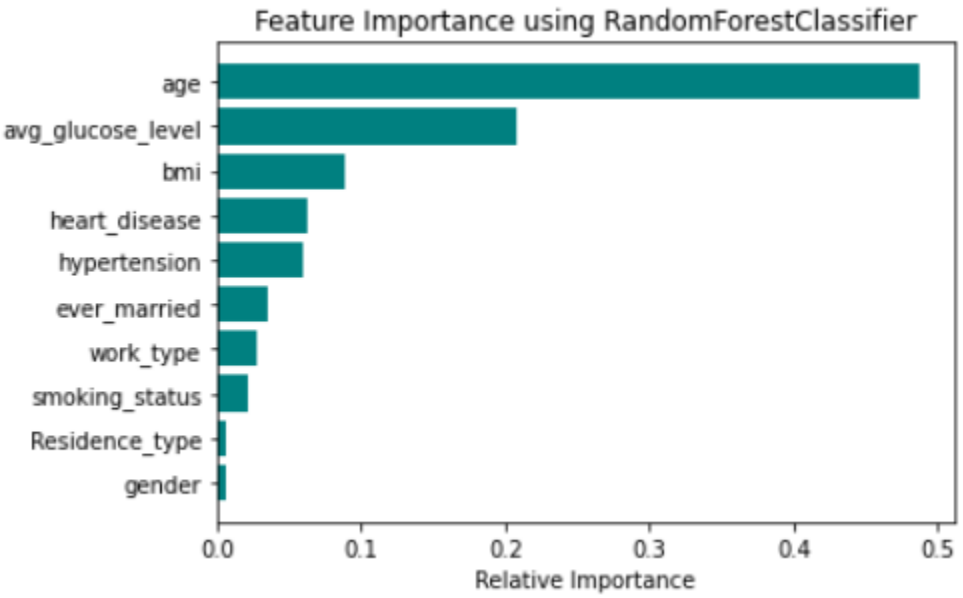


Figure 35: FI - RandomForestClassifier

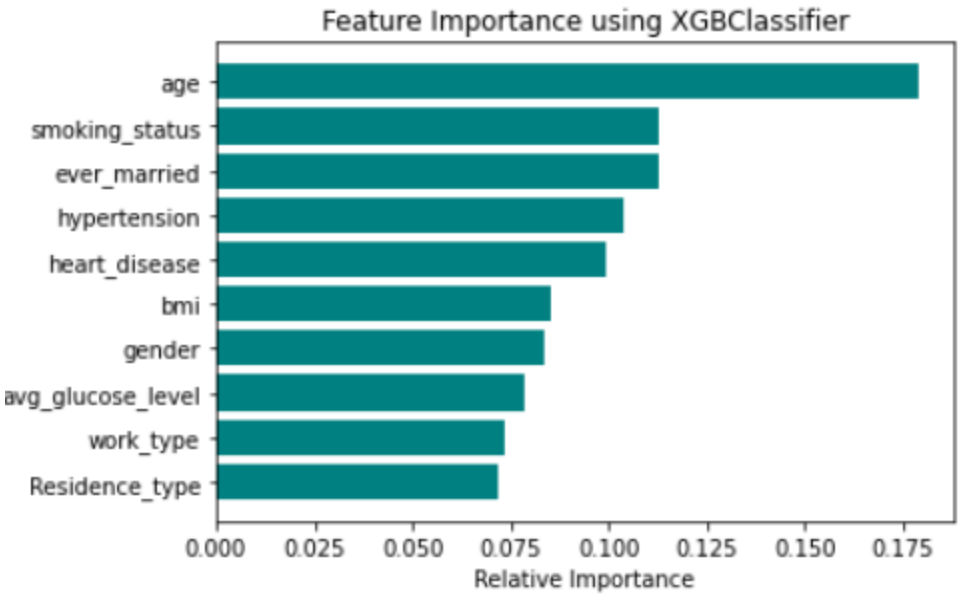


Figure 36: FI – XGBoostClassifier

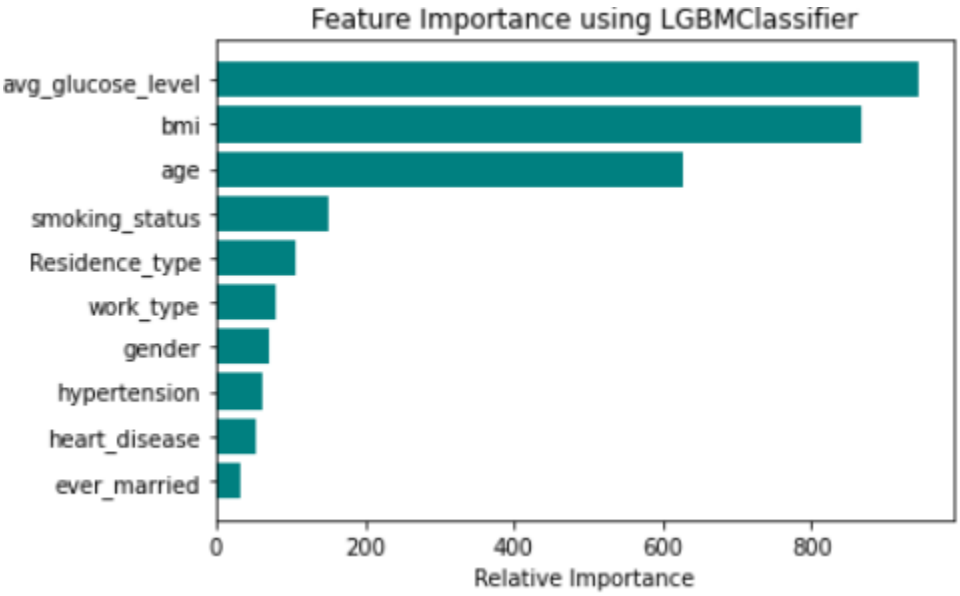


Figure 37: FI – LGBMClassifier

1.3.6 Feature Scaling

The feature scaling is implemented through the normalization (min-max scaling) of the data which rescales the values between 0 and 1. Another technique is standardisation where the mean and standard deviation is shifted to 0 and 1 respectively.

```
#Normalize data for numerical features
from sklearn.preprocessing import MinMaxScaler

# copy of datasets
data_stand = data.copy()

# numerical features
num_cols = ['avg_glucose_level','bmi']

for i in num_cols:
    scale = MinMaxScaler().fit(data_stand[[i]])
    data_stand[i] = scale.transform(data_stand[[i]])

#Write pre-processed data to a csv file
data_stand.to_csv("train_data.csv", index=False)

y_data = data_stand['stroke']
X_data_stand = data_stand.drop(columns=['stroke'])
data_stand
```

Figure 38: Data Normalization

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1.0	67.0	0.0	1.0	1.0	2.0	1.0	0.801265	0.301260	1.0	1.0
1	0.0	61.0	0.0	0.0	1.0	3.0	0.0	0.679023	0.212981	2.0	1.0
2	1.0	80.0	0.0	1.0	1.0	2.0	0.0	0.234512	0.254296	2.0	1.0
3	0.0	49.0	0.0	0.0	1.0	2.0	1.0	0.536008	0.276060	3.0	1.0
4	0.0	79.0	1.0	0.0	1.0	3.0	0.0	0.549349	0.156930	2.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
5105	0.0	80.0	1.0	0.0	1.0	2.0	1.0	0.132167	0.212981	2.0	0.0
5106	0.0	81.0	0.0	0.0	1.0	3.0	1.0	0.323516	0.340206	2.0	0.0
5107	0.0	35.0	0.0	0.0	1.0	3.0	0.0	0.128658	0.232532	2.0	0.0
5108	1.0	51.0	0.0	0.0	1.0	2.0	0.0	0.513203	0.175258	1.0	0.0
5109	0.0	44.0	0.0	0.0	1.0	0.0	1.0	0.139230	0.182131	0.0	0.0

5110 rows × 11 columns

Figure 39: Normalization output

1.3.7 Imbalanced data handling

The data for the target variable (stroke) is highly imbalanced with 4861 healthy people and just 249 stroke patients which can significantly affect the precision and recall values.

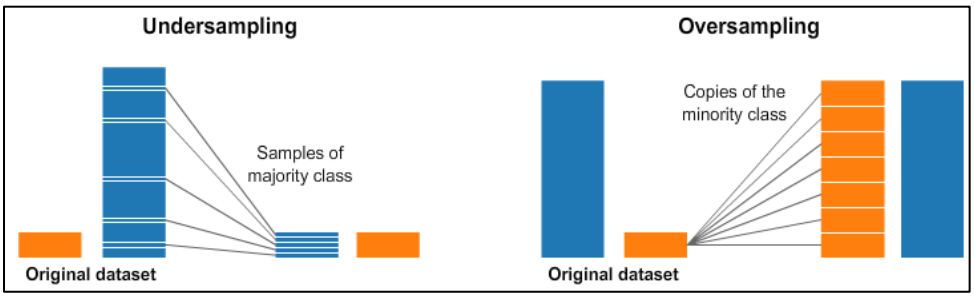


Figure 40: Resampling data

Over-sampling can result in over-fitting and under-sampling can lead to loss of information. Hence, the data has been balanced using a method which combines both under-sampling and over-sampling implemented by the "SMOTETomek" class of imblearn library.

```
#Undersampling for imbalanced stroke column data
# from imblearn.under_sampling import NearMiss
# sampler = NearMiss(version=1)
# X_sampled,y_sampled = sampler.fit_resample(X_data_stand, y_data)
# X_sampled.shape,y_sampled.shape

#Oversampling for imbalanced stroke column data
# from imblearn.over_sampling import SMOTE
# sampler = SMOTE()
# X_sampled,y_sampled = sampler.fit_resample(X_data_stand, y_data)
# X_sampled.shape,y_sampled.shape

# Combination of over- and under-sampling using SMOTE and Tomek links.
#Over-sampling using SMOTE and cleaning using Tomek links.
from imblearn.combine import SMOTETomek
sampler = SMOTETomek(random_state=42)
X_sampled,y_sampled = sampler.fit_resample(X_data_stand, y_data)
X_sampled.shape,y_sampled.shape
```

```
((9674, 10), (9674,))
```

Figure 41: Data resampling

1.4 Data Modelling

The entire dataset is split into training, validation, and test dataset in the ratio of 60:20:20.

```
#Data-splitting, Model configuration
#Split into training, validation and test dataset in the ratio 60:20:20
X_train, X_test, y_train, y_test = train_test_split(X_sampled, y_sampled, test_size=0.4, random_state=1)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=1)# stratify=y_sampled, shuffle=True

rf = RandomForestClassifier()
parameters = {
    'n_estimators': [5, 50, 100],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [2, 4, 8, 16],
    'criterion' :['gini', 'entropy']
}
print("X_train:", len(X_train))
print("y_train:", len(y_train))
print("X_val:", len(X_val))
print("y_val:", len(y_val))
print("X_test:", len(X_test))
print("y_test:", len(y_test))
```

```
X_train: 5804
y_train: 5804
X_val: 1935
y_val: 1935
X_test: 1935
y_test: 1935
```

Figure 42: Initial model configuration

The optimal hyperparameter values for the RandomForestClassifier model are calculated using fivefold cross-validation implemented by the GridSearchCV class of the sklearn library. Finally, the best model is saved to a file with a pkl extension for future use.

```
#Five fold cross validation using GridSearchCV and training
import joblib
from sklearn.model_selection import GridSearchCV, StratifiedKFold

def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))

kfold_split = StratifiedKFold(n_splits=5, shuffle=False)
cv = GridSearchCV(rf, parameters, cv=kfold_split, scoring='precision', verbose=1 )
cv.fit(X_train, y_train)
print_results(cv)
print(cv.best_estimator_)

joblib.dump(cv.best_estimator_, 'GB_model.pkl')
```

Figure 43: Five-fold Cross-validation

During the cross-validation process, the model is run for every possible combination of the specified parameter values and the most suitable one is determined.

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
BEST PARAMS: {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 100}

0.745 (+/-0.037) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'auto', 'n_estimators': 5}
0.751 (+/-0.014) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'auto', 'n_estimators': 50}
0.752 (+/-0.019) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'auto', 'n_estimators': 100}
0.748 (+/-0.013) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 5}
0.752 (+/-0.022) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 50}
0.754 (+/-0.018) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 100}
0.747 (+/-0.02) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'log2', 'n_estimators': 5}
0.752 (+/-0.022) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'log2', 'n_estimators': 50}
0.754 (+/-0.019) for {'criterion': 'gini', 'max_depth': 2, 'max_features': 'log2', 'n_estimators': 100}
0.788 (+/-0.031) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 5}
0.775 (+/-0.018) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 50}
0.778 (+/-0.019) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 100}
0.779 (+/-0.029) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 5}
0.779 (+/-0.018) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 50}
0.778 (+/-0.014) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 100}
0.783 (+/-0.019) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 5}
0.781 (+/-0.024) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 50}
0.774 (+/-0.02) for {'criterion': 'gini', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 100}
0.88 (+/-0.02) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 5}
0.886 (+/-0.019) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 50}
0.89 (+/-0.01) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 100}
0.884 (+/-0.031) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 5}
0.887 (+/-0.019) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 50}
0.893 (+/-0.01) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 100}
0.886 (+/-0.015) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 5}
0.895 (+/-0.021) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 50}
0.891 (+/-0.01) for {'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 100}
0.932 (+/-0.017) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'auto', 'n_estimators': 5}
0.954 (+/-0.015) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'auto', 'n_estimators': 50}
0.95 (+/-0.016) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'auto', 'n_estimators': 100}
0.93 (+/-0.014) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 5}
0.954 (+/-0.013) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 50}
0.951 (+/-0.013) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 100}
0.931 (+/-0.007) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'log2', 'n_estimators': 5}
0.95 (+/-0.014) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'log2', 'n_estimators': 50}
0.95 (+/-0.013) for {'criterion': 'gini', 'max_depth': 16, 'max_features': 'log2', 'n_estimators': 100}
0.736 (+/-0.024) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'auto', 'n_estimators': 5}
0.745 (+/-0.033) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'auto', 'n_estimators': 50}
0.751 (+/-0.019) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'auto', 'n_estimators': 100}
0.748 (+/-0.009) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 5}
0.744 (+/-0.035) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 50}
0.747 (+/-0.025) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 100}
0.729 (+/-0.055) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'log2', 'n_estimators': 5}
0.747 (+/-0.026) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'log2', 'n_estimators': 50}
0.749 (+/-0.021) for {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'log2', 'n_estimators': 100}
0.783 (+/-0.035) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 5}
0.779 (+/-0.021) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 50}
0.774 (+/-0.01) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 100}
0.778 (+/-0.03) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 5}
0.778 (+/-0.011) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 50}
0.781 (+/-0.024) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 100}
0.771 (+/-0.03) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 5}
0.779 (+/-0.017) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 50}
0.776 (+/-0.012) for {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'n_estimators': 100}
0.886 (+/-0.027) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 5}
0.884 (+/-0.018) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 50}
0.888 (+/-0.008) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'n_estimators': 100}
0.877 (+/-0.019) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 5}
0.89 (+/-0.01) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 50}
0.892 (+/-0.011) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 100}
0.873 (+/-0.022) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 5}
0.892 (+/-0.016) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 50}
0.887 (+/-0.016) for {'criterion': 'entropy', 'max_depth': 8, 'max_features': 'log2', 'n_estimators': 100}
0.937 (+/-0.017) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'auto', 'n_estimators': 5}
0.952 (+/-0.01) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'auto', 'n_estimators': 50}
0.952 (+/-0.009) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'auto', 'n_estimators': 100}
0.935 (+/-0.02) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 5}
0.952 (+/-0.013) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 50}
0.955 (+/-0.012) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqrt', 'n_estimators': 100}
0.939 (+/-0.014) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'log2', 'n_estimators': 5}
0.951 (+/-0.02) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'log2', 'n_estimators': 50}
0.954 (+/-0.014) for {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'log2', 'n_estimators': 100}
RandomForestClassifier(criterion='entropy', max_depth=16, max_features='sqrt')
```

Figure 44: Cross-validation output

1.5 Model Evaluation

The model was able to predict stroke with 96.7% accuracy on the validation dataset and 96.8% accuracy on the test dataset respectively. The precision and recall values are approximately the same on both datasets. But the metric specific for imbalanced datasets i.e., Index Balanced Accuracy (IBA) is 0.241 in the validation dataset and 0.938 in the test dataset.

```
#Model validation using validation data set
from time import time
from sklearn.metrics import accuracy_score, precision_score, recall_score
from imblearn.metrics import geometric_mean_score
from imblearn.metrics import make_index_balanced_accuracy

def evaluate_model(features, labels):
    start = time()
    pred = model.predict(features)
    end = time()
    accuracy = round(accuracy_score(labels, pred), 3)
    precision = round(precision_score(labels, pred), 3)
    recall = round(recall_score(labels, pred), 3)
    print('{} -- Accuracy: {} / Precision: {} / Recall: {} / Latency: {}ms'.format('RandomForest',
                                                                                   accuracy,
                                                                                   precision,
                                                                                   recall,
                                                                                   round((end - start)*1000, 1)))
    print(f"The geometric mean is {geometric_mean_score(y_test, pred):.3f}")

    alpha = 0.5
    geo_mean = make_index_balanced_accuracy(alpha=alpha, squared=True)(geometric_mean_score)

    print(f"The IBA using alpha={alpha} and the geometric mean: "
          f"{geo_mean(y_test, pred):.3f}")

model = cv.best_estimator_
evaluate_model(X_val, y_val)
```

```
RandomForest -- Accuracy: 0.967 / Precision: 0.962 / Recall: 0.972 / Latency: 87.4ms
The geometric mean is 0.490
The IBA using alpha=0.5 and the geometric mean: 0.241
```

Figure 45: Model validation

```
#Model evaluation using test data set
evaluate_model(X_test, y_test)
```

```
RandomForest -- Accuracy: 0.968 / Precision: 0.964 / Recall: 0.973 / Latency: 84.9ms
The geometric mean is 0.968
The IBA using alpha=0.5 and the geometric mean: 0.938
```

Figure 46: Model evaluation

RandomForestClassifier using Weka

The RandomForestClassifier model with 10-fold cross-validation gave 94.83% accuracy, 91% precision and 94.8% recall values.

```
=== Run information ===

Scheme:       weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1
Relation:     train_data_all
Instances:    5110
Attributes:   11
              gender
              age
              hypertension
              heart_disease
              ever_married
              work_type
              Residence_type
              avg_glucose_level
              bmi
              smoking_status
              stroke
Test mode:    10-fold cross-validation
=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.58 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        4846               94.8337 %
Incorrectly Classified Instances       264                5.1663 %
Kappa statistic                          0.0081
Mean absolute error                      0.0872
Root mean squared error                  0.2147
Relative absolute error                 93.8821 %
Root relative squared error             99.7095 %
Total Number of Instances             5110

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.008    0.003    0.105      0.008    0.015      0.016    0.799     0.134     Yes
                 0.997    0.992    0.951      0.997    0.973      0.016    0.799     0.985     No
Weighted Avg.    0.948    0.944    0.910      0.948    0.927      0.016    0.799     0.944

=== Confusion Matrix ===

    a    b   <-- classified as
    2  247 |    a = Yes
   17 4844 |    b = No
```

Figure 47: RF classification-Weka

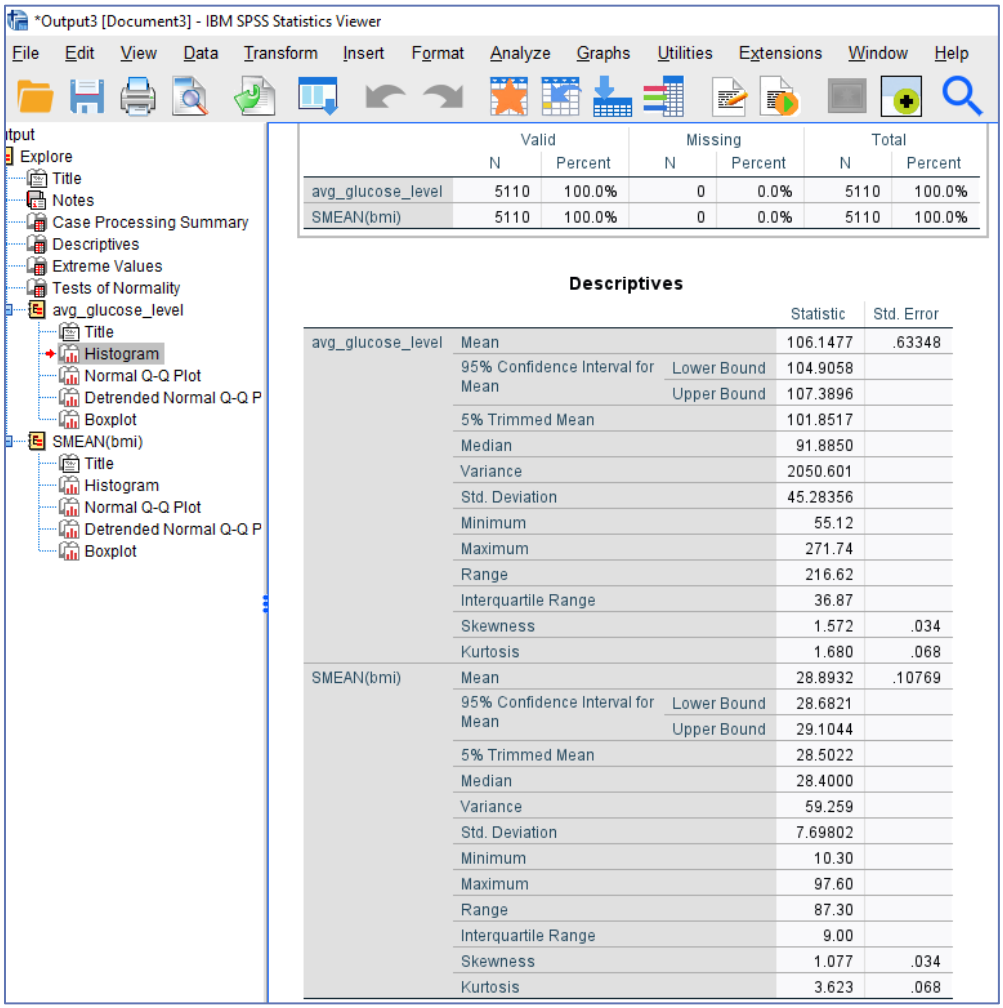
Appendix

KS Normality test results

Tests of Normality

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
avg_glucose_level	.183	5110	.000
SMEAN(bmi)	.073	5110	<.001

a. Lilliefors Significance Correction



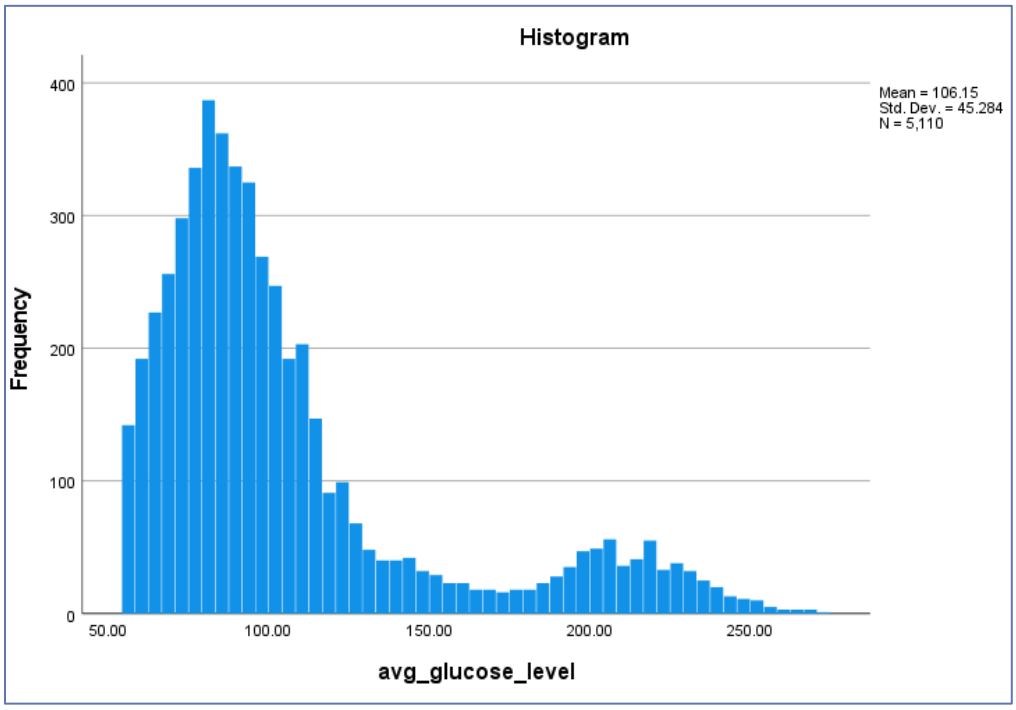
	Valid		Missing		Total	
	N	Percent	N	Percent	N	Percent
avg_glucose_level	5110	100.0%	0	0.0%	5110	100.0%
SMEAN(bmi)	5110	100.0%	0	0.0%	5110	100.0%

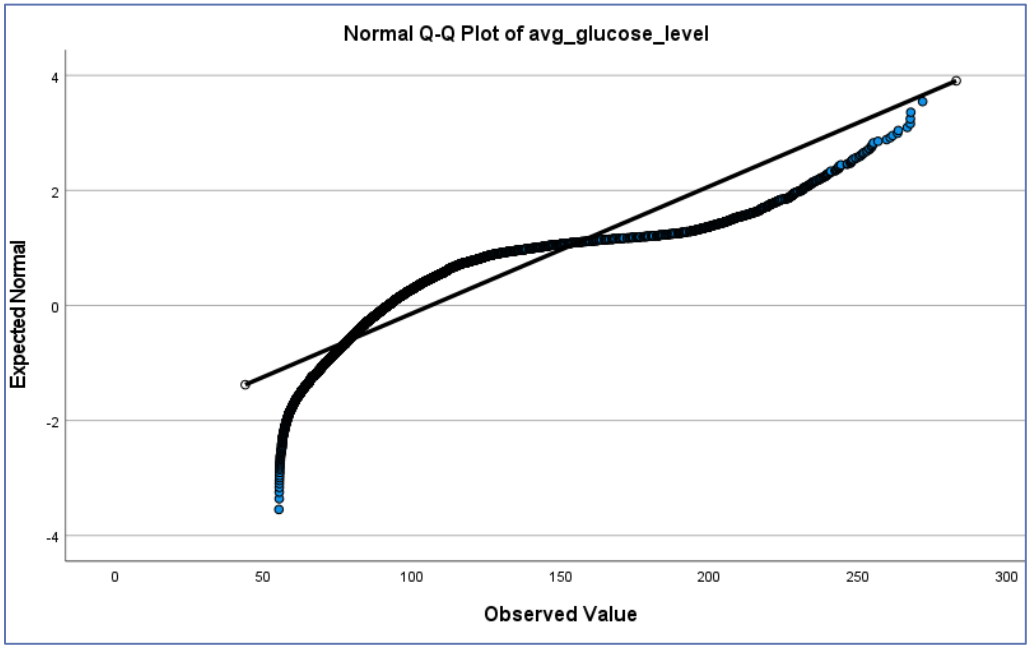
Descriptives

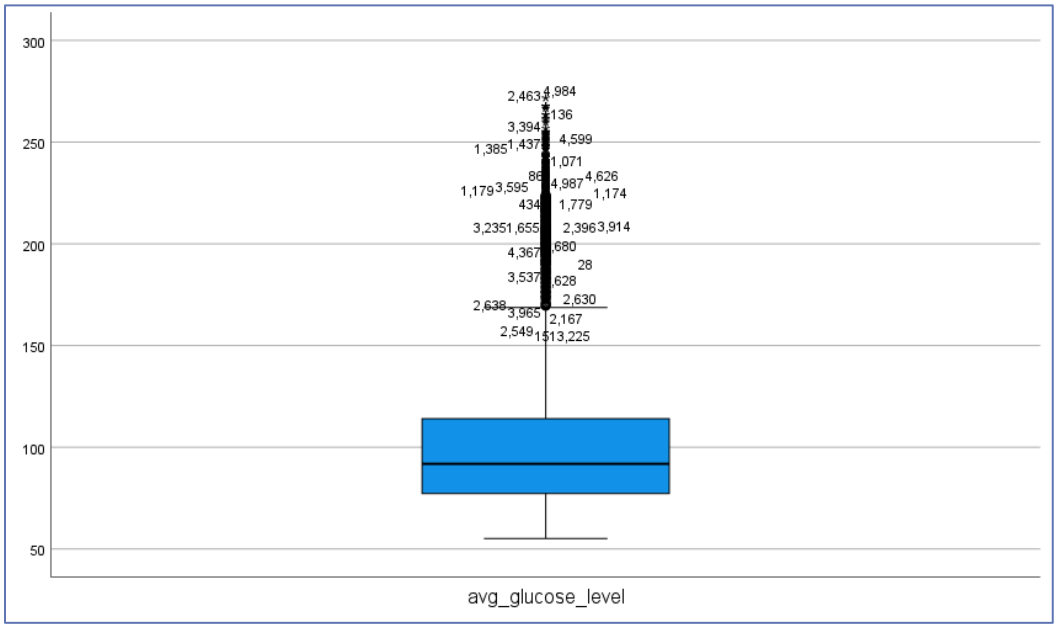
			Statistic	Std. Error
avg_glucose_level	Mean		106.1477	.63348
	95% Confidence Interval for Mean	Lower Bound	104.9058	
		Upper Bound	107.3896	
	5% Trimmed Mean		101.8517	
	Median		91.8850	
	Variance		2050.601	
	Std. Deviation		45.28356	
	Minimum		55.12	
	Maximum		271.74	
	Range		216.62	
	Interquartile Range		36.87	
	Skewness		1.572	.034
	Kurtosis		1.680	.068
SMEAN(bmi)	Mean		28.8932	.10769
	95% Confidence Interval for Mean	Lower Bound	28.6821	
		Upper Bound	29.1044	
	5% Trimmed Mean		28.5022	
	Median		28.4000	
	Variance		59.259	
	Std. Deviation		7.69802	
	Minimum		10.30	
	Maximum		97.60	
	Range		87.30	
	Interquartile Range		9.00	
	Skewness		1.077	.034
	Kurtosis		3.623	.068

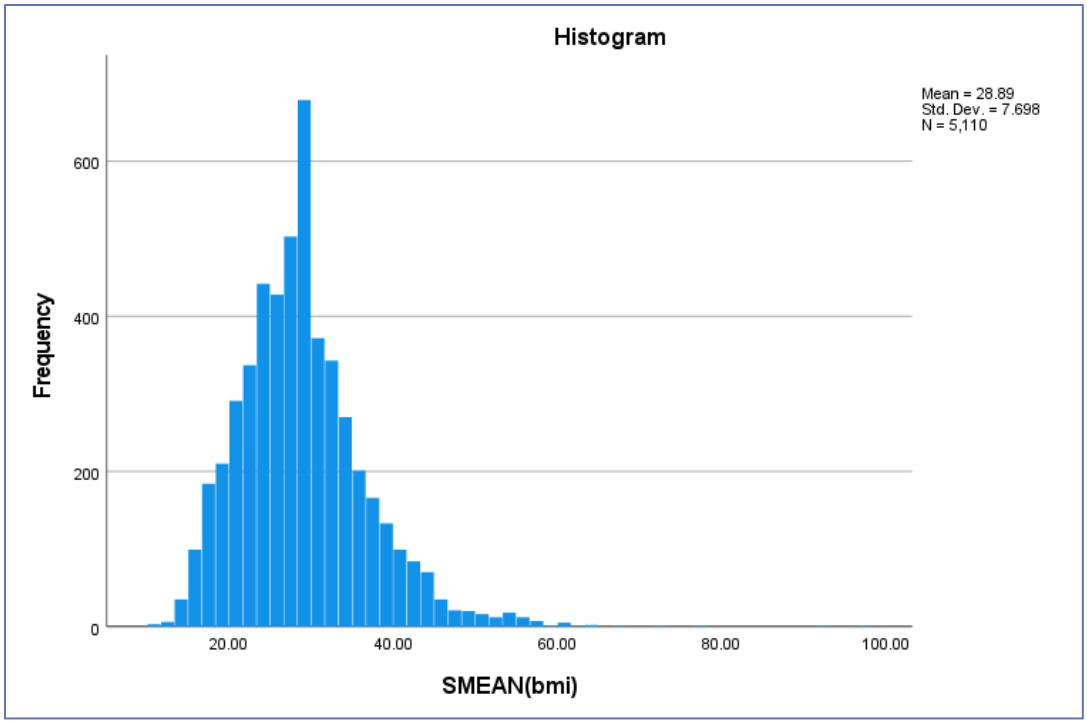
Extreme Values

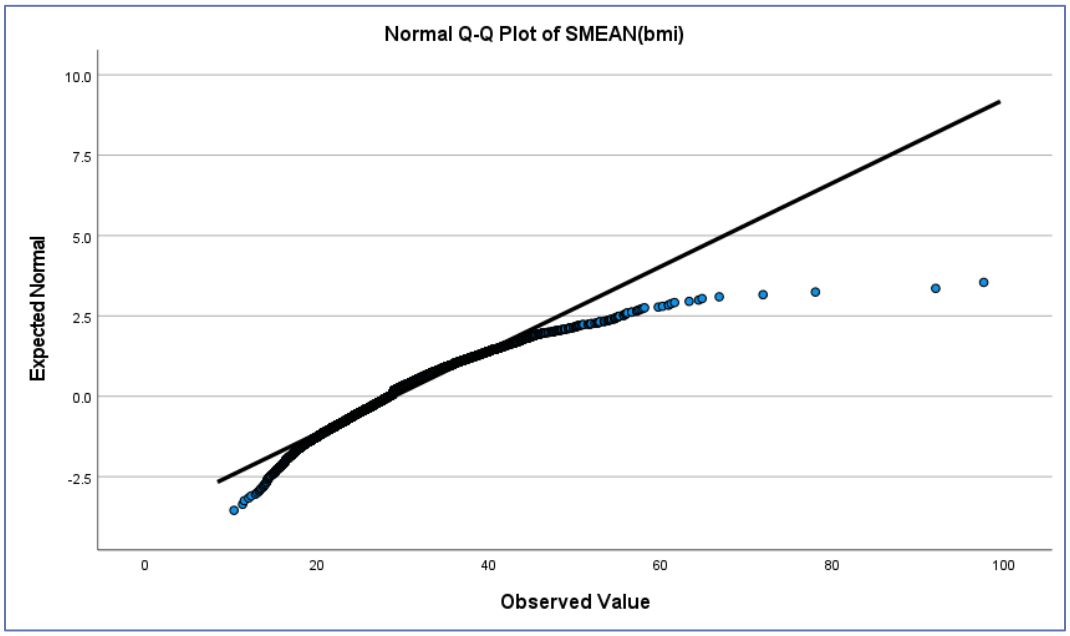
			Case Number	Value
avg_glucose_level	Highest	1	194	271.74
		2	1208	267.76
		3	3089	267.61
		4	4984	267.60
		5	2463	266.59
	Lowest	1	4887	55.12
		2	1192	55.22
		3	4111	55.23
		4	683	55.25
		5	773	55.26
SMEAN(bmi)	Highest	1	2129	97.60
		2	4210	92.00
		3	929	78.00
		4	545	71.90
		5	1560	66.80
	Lowest	1	1610	10.30
		2	3308	11.30
		3	2188	11.50
		4	658	12.00
		5	923	12.30

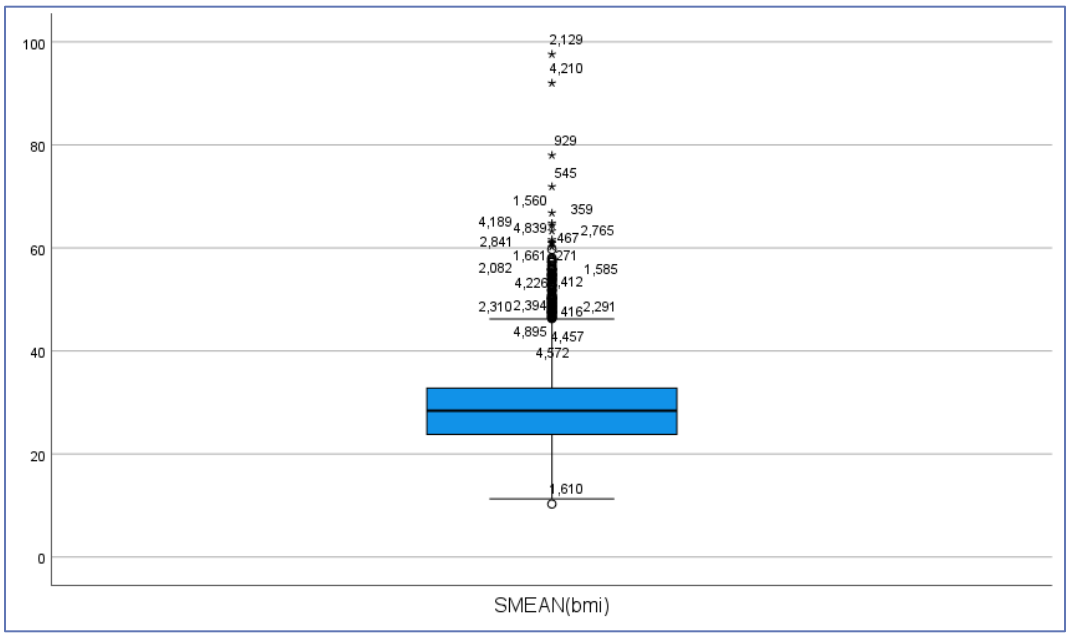



Normal Q-Q Plot of avg_glucose_level
Expected Normal
Observed Value
4
2
0
-2
-4
0
50
100
150
200
250
300


300
250
200
150
100
50
avg_glucose_level


Histogram
Mean = 28.89
Std. Dev. = 7.698
N = 5,110
Frequency
600
400
200
0
20.00
40.00
60.00
80.00
100.00
SMEAN(bmi)


Normal Q-Q Plot of SMEAN(bmi)
Expected Normal
10.0
7.5
5.0
2.5
0.0
-2.5
0
20
40
60
80
100
Observed Value


100
80
60
40
20
0
2,129
4,210
929
545
1,560
359
4,189
4,839
2,765
2,841
467
1,661
271
2,082
1,585
4,226
412
2,310
2,394
416
2,291
4,895
4,457
4,572
1,610
SMEAN(bmi)