MEDICAL INSURANCE CHARGES PREDICTION

BHAVAN'S VIVEKANANDA COLLEGE

PRESENTED BY:
Group 6:

MD SAFWAN SHAWN | K ROHIT KUMAR | MOHAMMED AYAZ | M BHARGAVI REDDY

Abstract

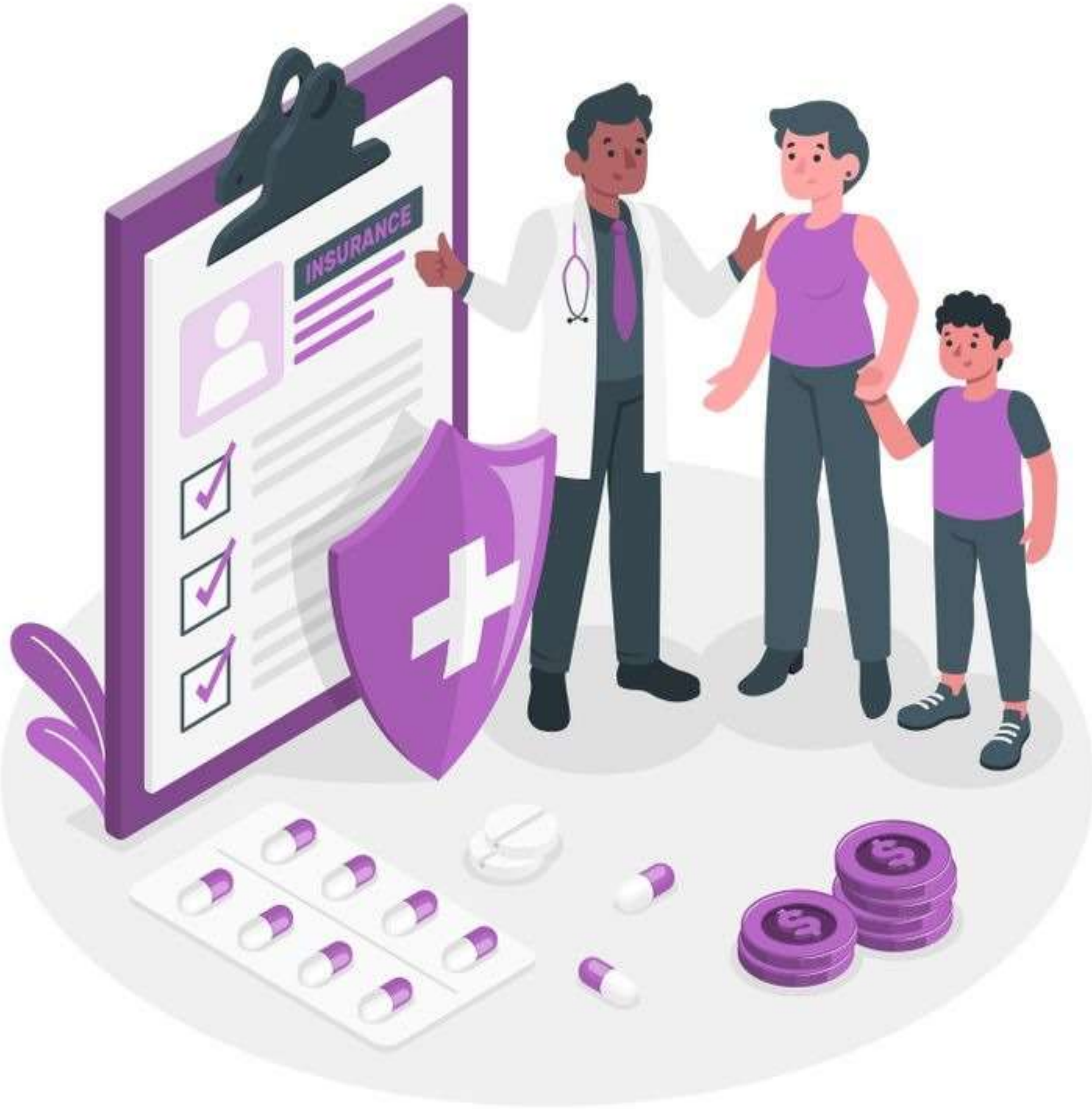
The project focuses on building a model that accurately predicts the insurance charges, based on the individual's demographic and lifestyle information. The aim is likely to predict or understand factors influencing insurance costs.

The study explores machine learning algorithms like Linear Regression , Multiple Regression, K-Nearest Neighbors , Support Vector Machine, Random Forest and Boosting to predict insurance charges.

Objective

To find the most optimal machine learning model that fits and predicts the insurance charges for identifying which variables significantly impact the insurance charges.

Content



Introduction

- Medical insurance is a type of coverage that helps individuals manage healthcare costs by providing financial support for medical expenses, including hospital stays, treatments, and medications.

- It's a contract between you and an insurance company where you pay a regular premium, and in return, the insurer agrees to cover a portion of your medical expenses.

- This coverage promotes timely treatment and preventive care, making healthcare more affordable and accessible.

LITERATURE REVIEW

Literature Review - 1

ul Hassan, C.A., Iqbal, J., Hussain, S., AlSalman, H., Mosleh, M.A. and Sajid Ullah, S., 2021. A computational intelligence approach for predicting medical insurance cost.

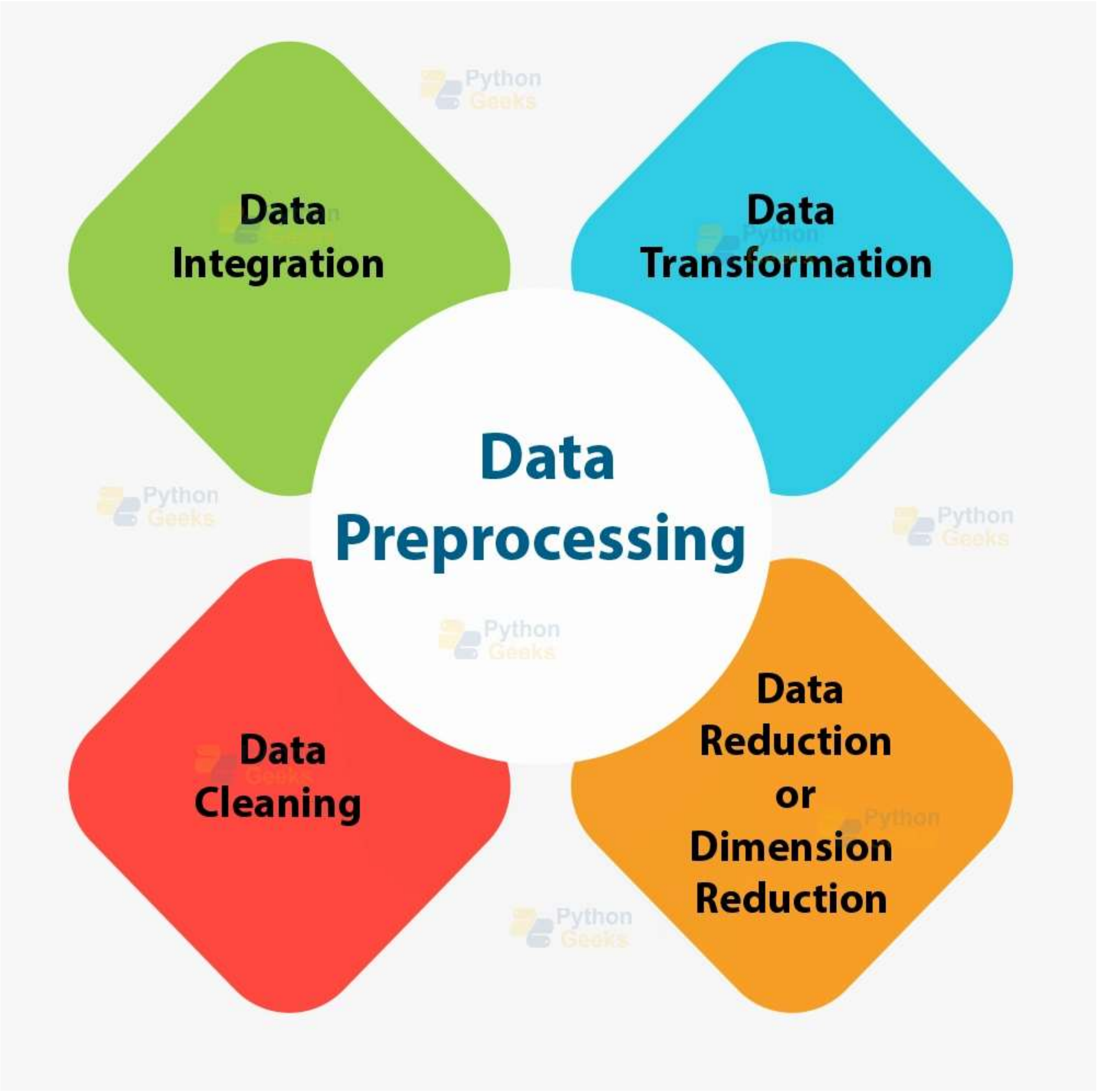
Ul Hassan et al.'s 2021 study proposes a machine learning framework to predict medical insurance costs, comparing models like linear regression, decision trees, random forests, and neural networks (with factors like age, gender, BMI, and smoking status).The research finds that ensemble methods, especially random forests, deliver the highest prediction accuracy by effectively handling complex variable relationships. This approach, validated by metrics such as mean absolute error and R-squared, enables insurers to price premiums more accurately and manage risk better.

Literature Review - 2

US, S. and Mathew, A., 2024. Medical Insurance Cost Prediction.

US and Mathew (2024) focus on predicting medical insurance costs by using advanced machine learning techniques to identify cost drivers such as demographic and lifestyle factors. This study aims to enhance the accuracy of premium estimation through methods that capture nonlinear interactions between variables, building on recent improvements in prediction accuracy within this domain. Expected to be published in International Journal of Data Communication and Networking (IJDCN), the paper seeks to contribute to better financial forecasting and policy pricing in the insurance sector.

DATA PREPROCESSING



What is Data Preprocessing?

Data preprocessing is an essential step in any data science or machine learning project. It involves preparing raw data to ensure it is clean, consistent, and suitable for analysis or modeling. Here's an overview of its key steps:

1) Data Quality Assessment: Evaluates the quality of dataset by assessing its accuracy , completeness, etc
2) Data Cleaning: Address missing, duplicate, or incorrect data.
3) Data Transformation: Convert data into a format that machine learning can understand.
4) Data Reduction: Reducing the number of attributes in a dataset while preserving as much of the original dataset.

Data

- Dataset : Our Dataset consists of 7 variables and 1,338 observations

- Source: https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/insurance.csv

- Variables:

Categorical Variables	Continuous Variables
Sex	Age
Smoker	BMI
Region	Children
	Charges

age	sex	bmi	children	smoker	region	charges
19	female	27.9	0	yes	southwest	16884.92
18	male	33.77	1	no	southeast	1725.552
28	male	33	3	no	southeast	4449.462
33	male	22.705	0	no	northwest	21984.47
32	male	28.88	0	no	northwest	3866.855
31	female	25.74	0	no	southeast	3756.622
46	female	33.44	1	no	southeast	8240.59
37	female	27.74	3	no	northwest	7281.506
37	male	29.83	2	no	northeast	6406.411
60	female	25.84	0	no	northwest	28923.14
25	male	26.22	0	no	northeast	2721.321
62	female	26.29	0	yes	southeast	27808.73
23	male	34.4	0	no	southwest	1826.843

DATA CLEANING

- Data cleaning is the process of fixing or removing incorrect, incomplete, or duplicate data from a dataset.

- Data cleaning is an important step in data preprocessing. It helps to ensure that the data used for analysis or machine learning is reliable and high quality.

- In Data cleaning, we usually check with our dataset for any missing or incorrect data. In this model the data cleaning process was as follows:

  - ❖ Checked for missing and unique values. There were no missing or unique values.

  - ❖ Calculated the mean value for the “charges” column and then converted the column into categorical by relabelling the values below the mean value as 0 and above the mean value as 1.

❖ Renaming the new charges column as "ChargesNew".

❖ Performed dummy variable encoding on categorical variables to convert them to continuous.

charges	ChargesNew
16884.92400	1
1725.55230	0
4449.46200	0
21984.47061	1
3866.85520	0
...	...
10600.54830	0
2205.98080	0
1629.83350	0
2007.94500	0
29141.36030	1

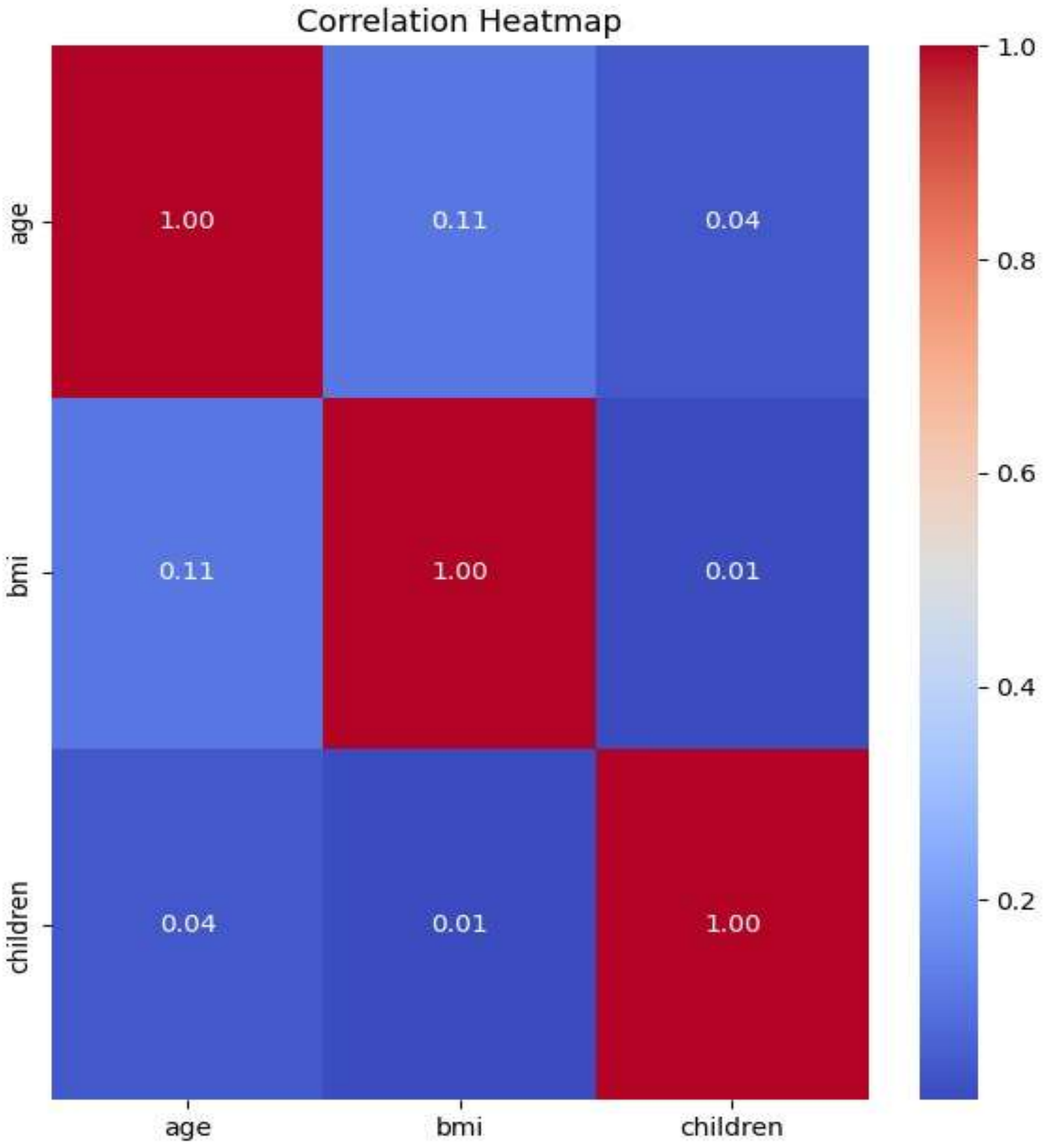
The new charges column.

Original Variables	Renamed Variables
sex	sex_male
smoker	smoker_yes
charges	ChargesNew

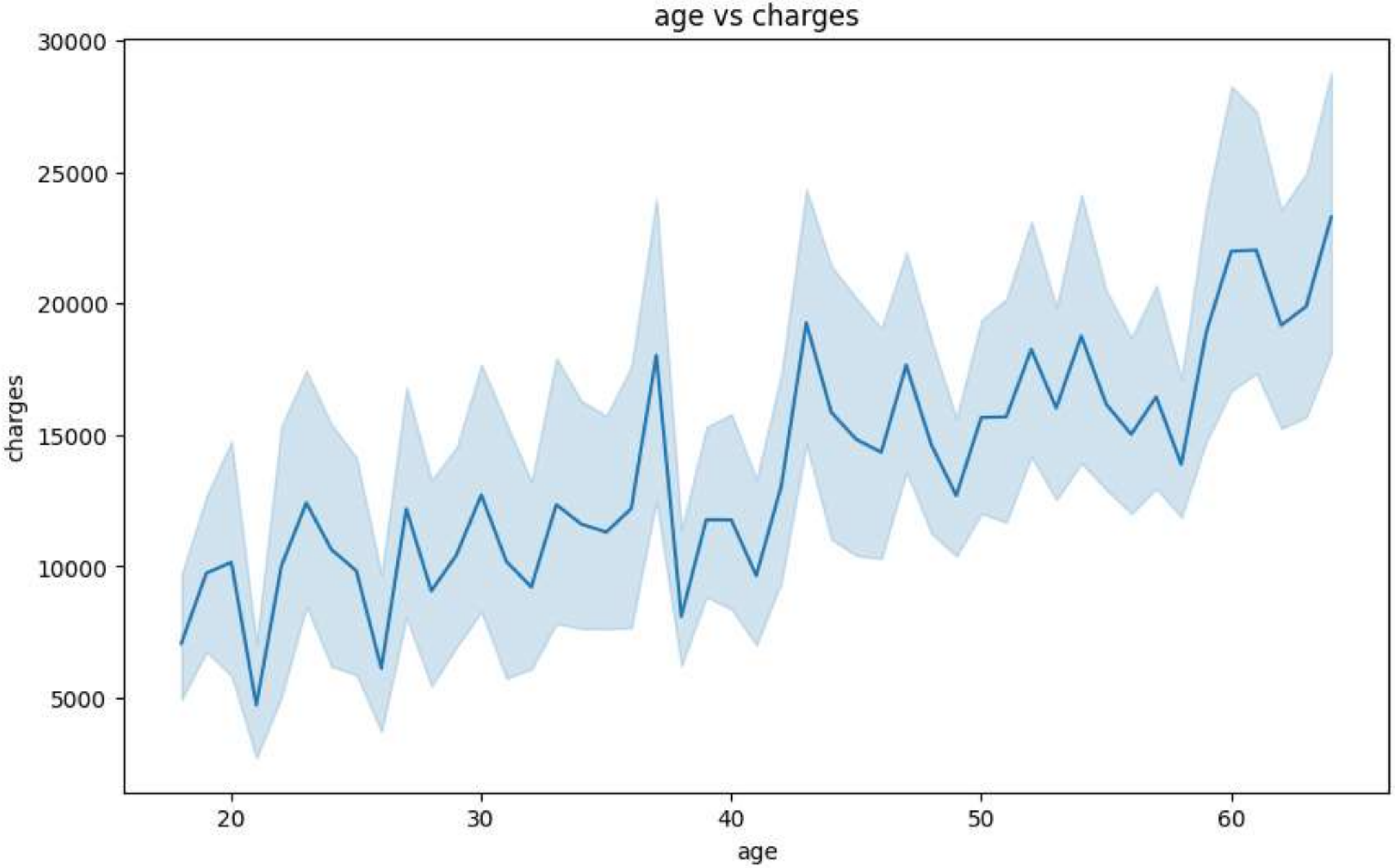
EXPLORATORY DATA ANALYSIS

Correlation Matrix

- There is a weak positive correlation between age and BMI, indicated by the value of 0.11.
- The heatmap reveals that there are no strong correlations between any of the three variables. The relationships are either very weak or non-existent.

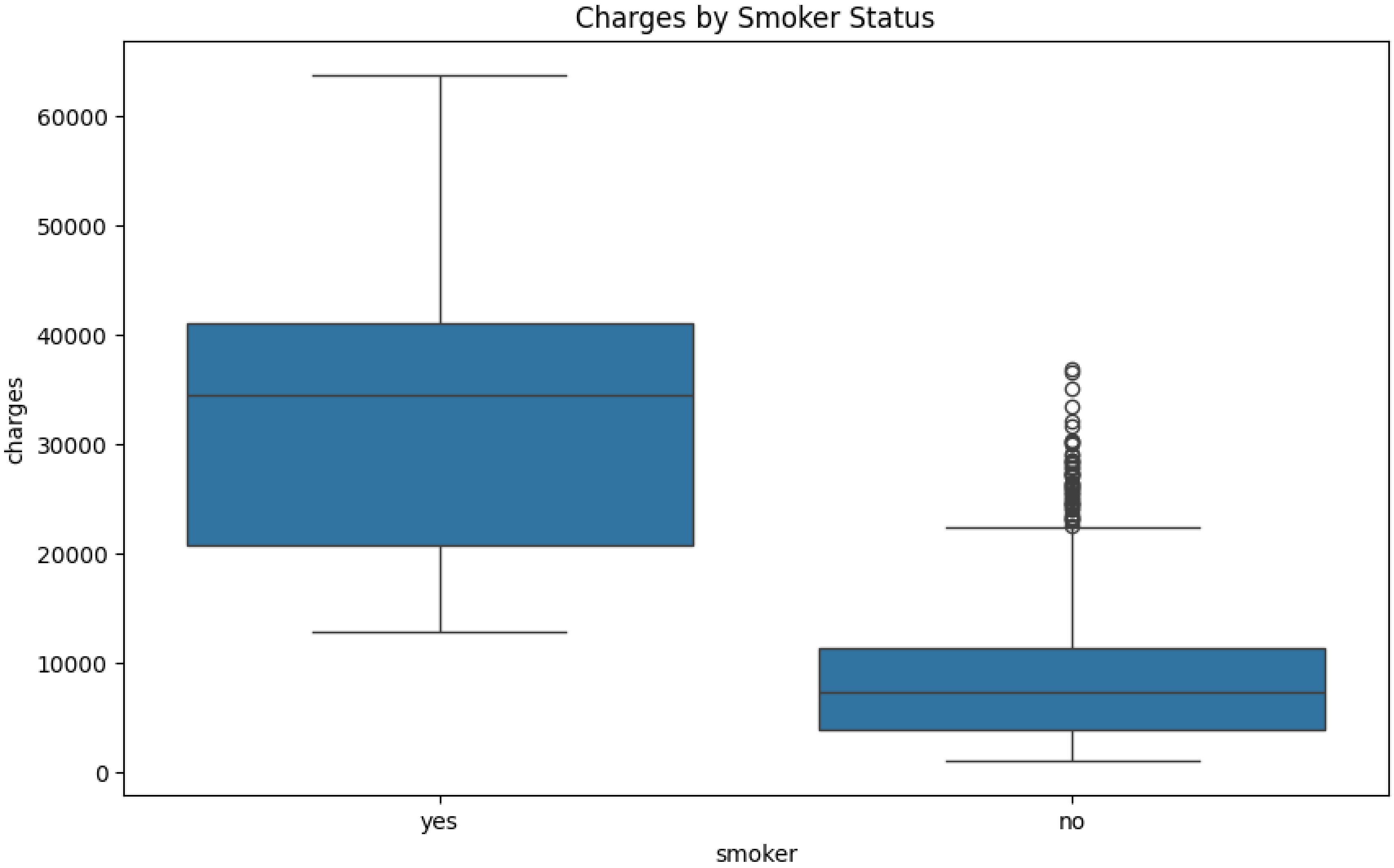


Line Plot



- As age increases, average charges also tend to increase. This suggests that age is an important feature and could have a positive correlation with charges.
- The shaded area widens as age increases, meaning that the variation in charges tends to increase for older individuals.

Box Plot

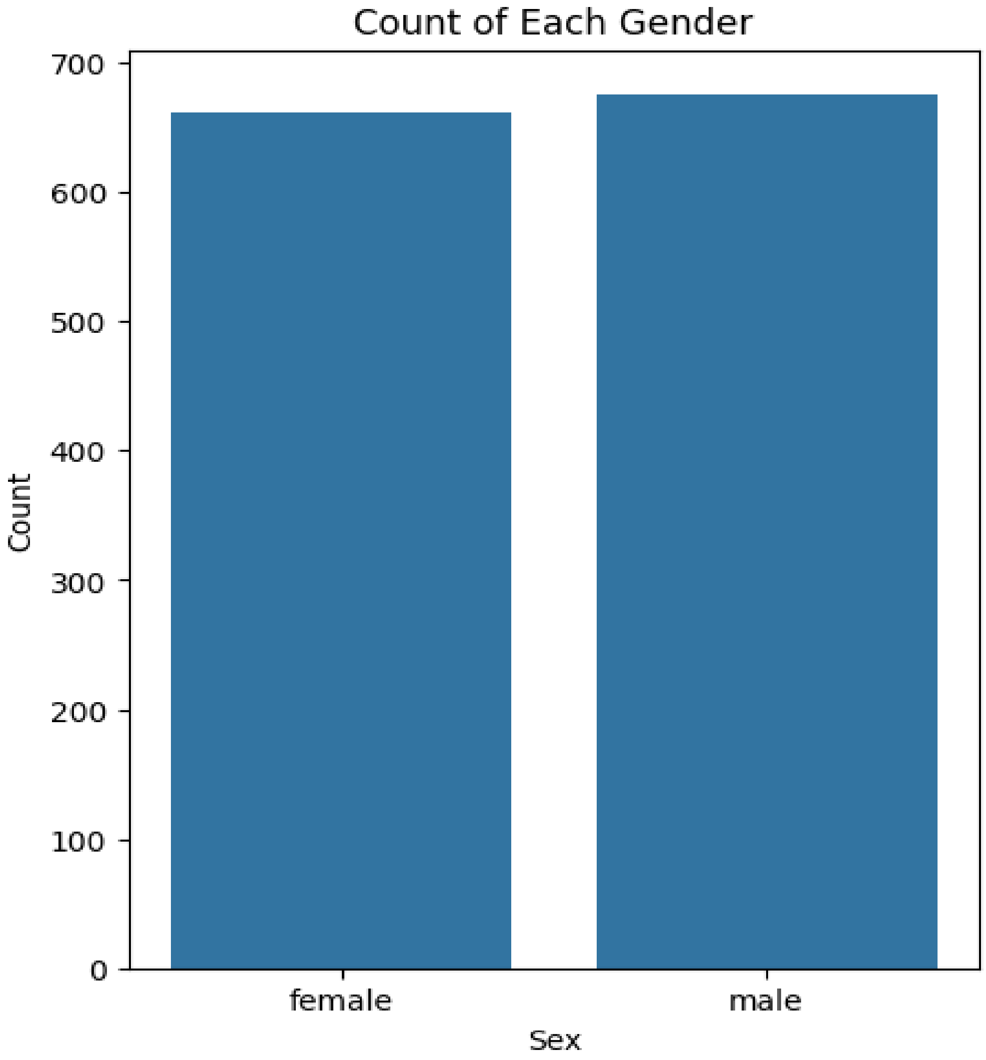


- The box plot for smokers is much higher than for non-smokers. This indicates that smokers, on average, incur significantly higher charges compared to non-smokers.

- This suggests greater variability in charges among smokers.

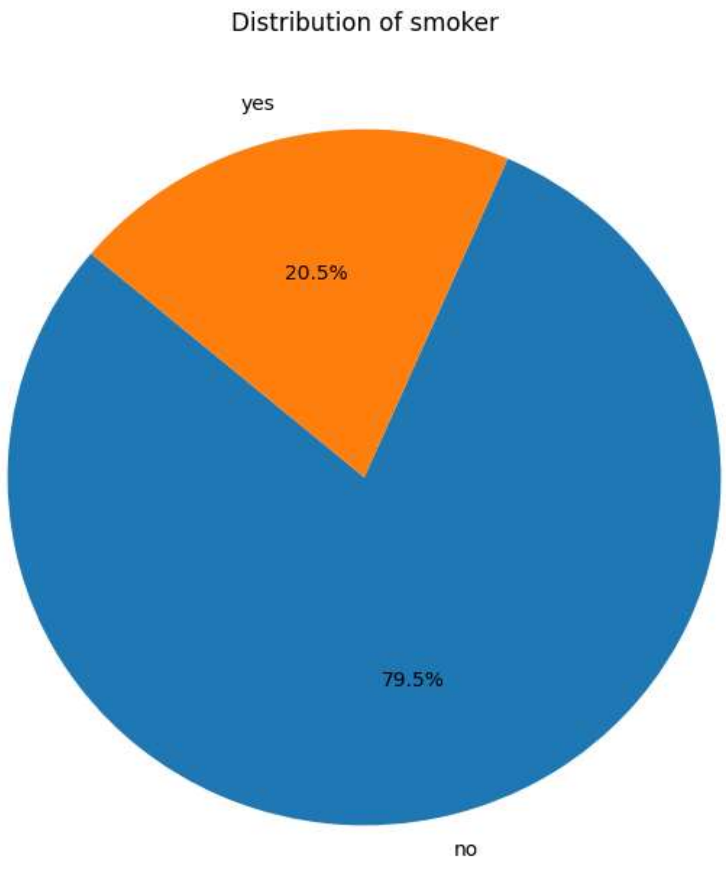
- The plot shows some outliers for non-smokers, represented by small circles above the main box. These are cases where certain non-smokers have unusually high charges, but they are not common.

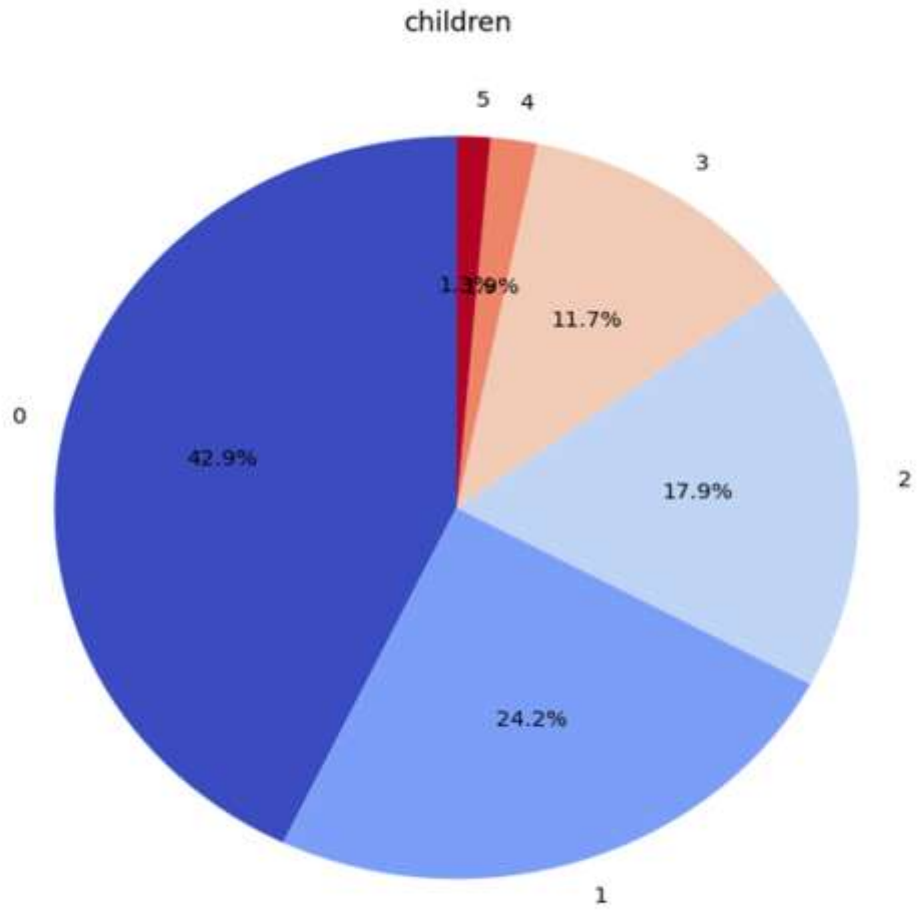
Count plot



This bar chart shows that the counts of females and males are nearly equal in the dataset.

PIE CHARTS





The majority (79.5%) of respondents indicated they are not smokers, while 20.5% stated they are smokers.

The majority (42.9%) of respondents have no child, followed by respondents having 1 child(24.2%).

Multicollinearity

- Multicollinearity happens when two or more variables in a model are highly correlated with each other.

- This makes it difficult to determine the individual effect of each variable on dependent variable.

Multicollinearity Check

We removed the variables that had VIF > 3, to get independent variables that have very less multicollinearity.

	variables	VIF
0	age	7.7
1	bmi	11.4
2	children	1.8
3	sex_male	2.0
4	smoker_yes	1.3
5	region_northwest	1.9
6	region_southeast	2.3
7	region_southwest	2.0

We can observe "BMI" has the highest VIF.

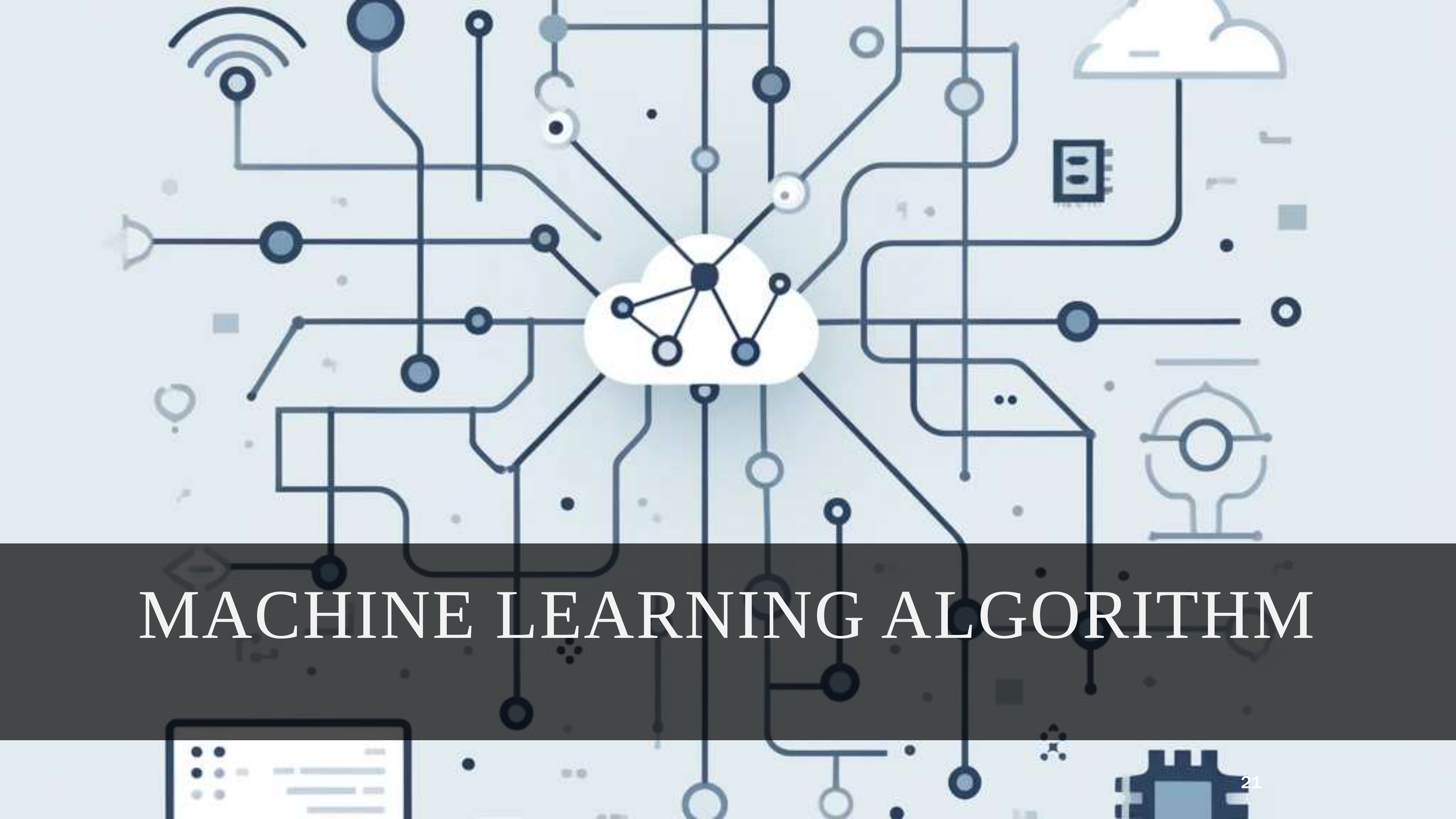
	variables	VIF
0	age	3.9
1	children	1.8
2	sex_male	1.9
3	smoker_yes	1.2
4	region_northwest	1.7
5	region_southeast	1.8
6	region_southwest	1.7

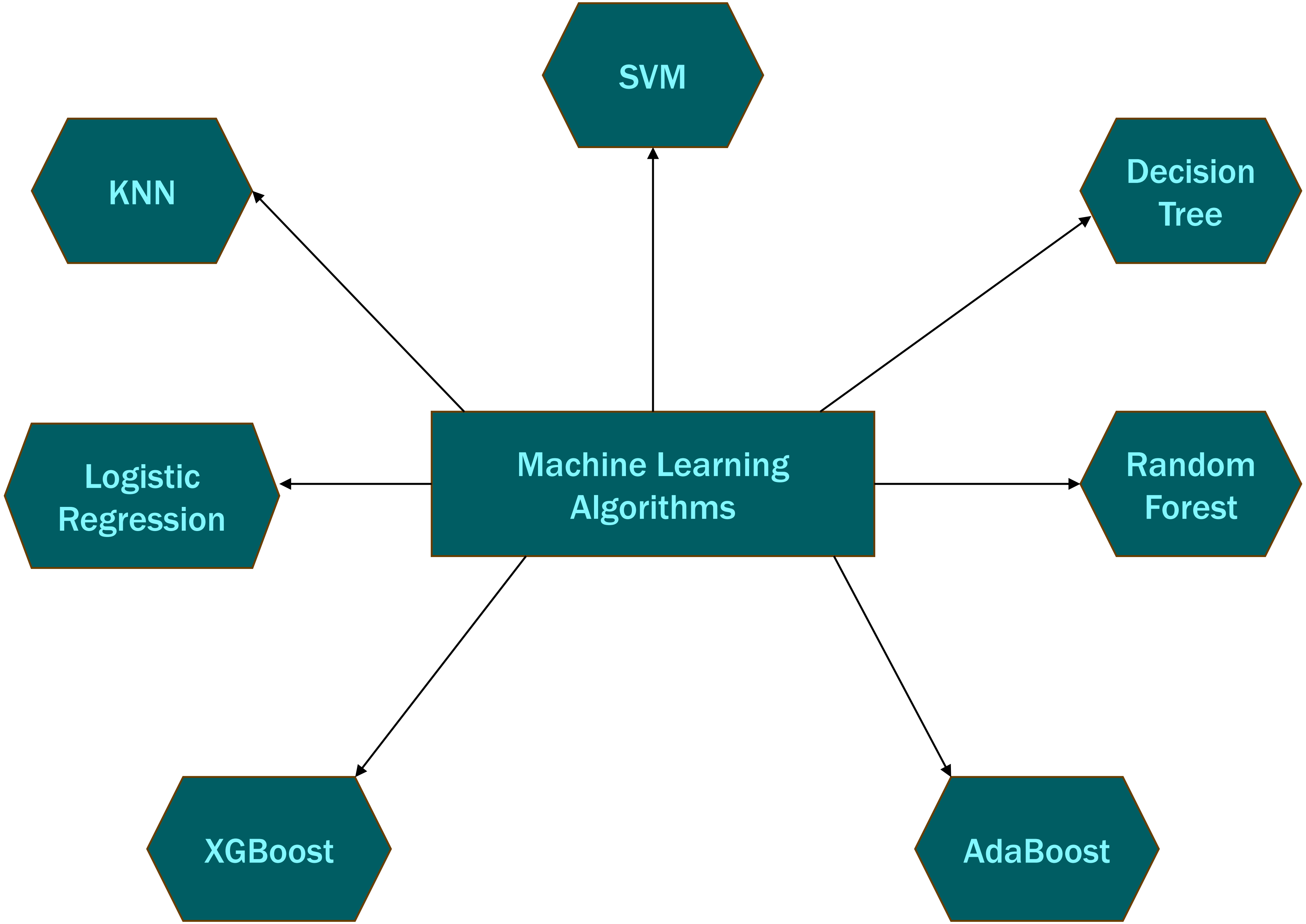
BMI is removed and now we see that the variable "age" has a VIF > 3.

	variables	VIF
0	children	1.6
1	sex_male	1.7
2	smoker_yes	1.2
3	region_northwest	1.3
4	region_southeast	1.4
5	region_southwest	1.3

The final variables with VIF<3.

MACHINE LEARNING ALGORITHM




SVM
KNN
Decision Tree
Logistic Regression
Machine Learning Algorithms
Random Forest
XGBoost
AdaBoost

80-20 Train-Test Split

Algorithms	MODEL - 1 Accuracy	MODEL – 2 Accuracy
Logistic Regression	0.889	0.899
KNN	0.739	0.895
SVM	0.899	0.899
Decision Tree	0.918	0.899
Random Forest	0.914	0.895
AdaBoost	0.918	0.895
XGBoost	0.914	0.895

Model 1: With all features | Model 2: After removing multi-collinear variables

75-25 Train-Test Split

Algorithms	MODEL - 1 Accuracy	MODEL – 2 Accuracy
Logistic Regression	0.907	0.907
KNN	0.755	0.892
SVM	0.907	0.907
Decision Tree	0.928	0.907
Random Forest	0.919	0.904
AdaBoost	0.922	0.904
XGBoost	0.922	0.905

Model 1: With all features | Model 2: After removing multi-collinear variables

70-30 Train-Test Split

Algorithms	MODEL - 1 Accuracy	MODEL – 2 Accuracy
Logistic Regression	0.905	0.905
KNN	0.749	0.880
SVM	0.905	0.903
Decision Tree	0.930	0.905
Random Forest	0.922	0.902
AdaBoost	0.925	0.905
XGBoost	0.920	0.903

Model 1: With all features | Model 2: After removing multi-collinear variables

60-40 Train-Test Split

Algorithms	MODEL - 1 Accuracy	MODEL – 2 Accuracy
Logistic Regression	0.910	0.908
KNN	0.744	0.889
SVM	0.909	0.908
Decision Tree	0.931	0.904
Random Forest	0.922	0.904
AdaBoost	0.919	0.901
XGBoost	0.916	0.916

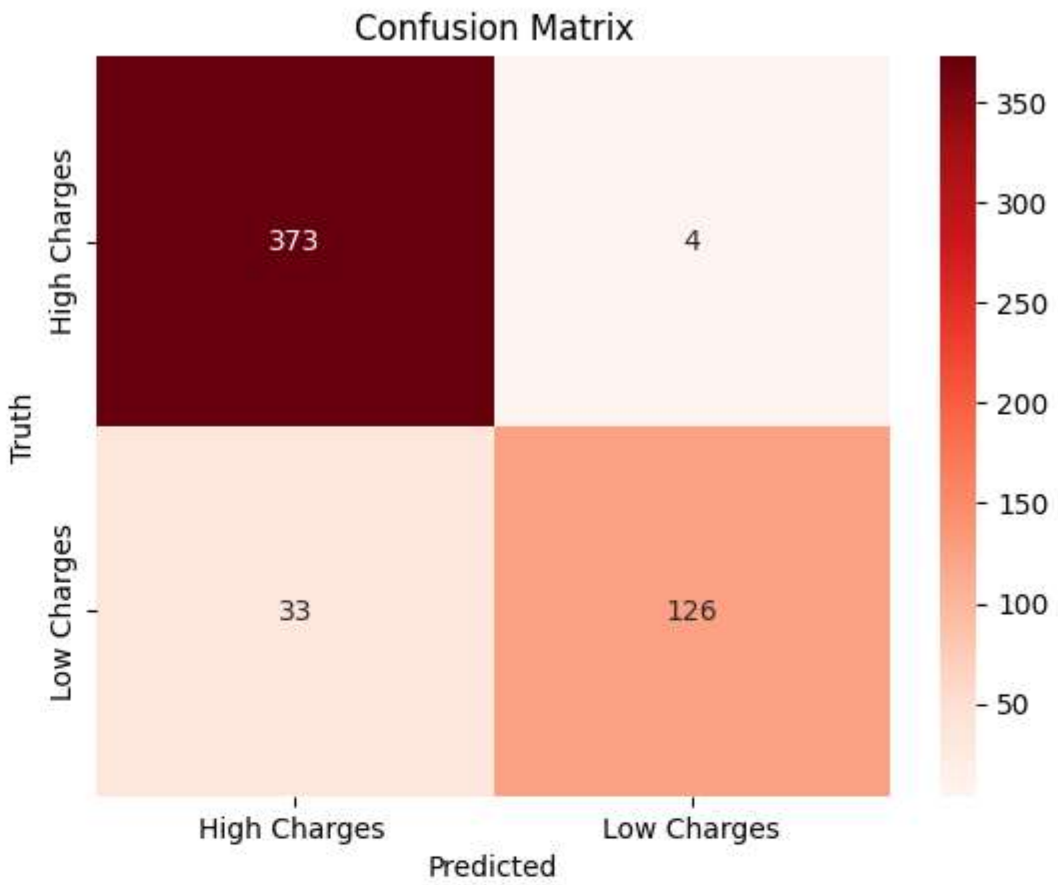
Model 1: With all features | Model 2: After removing multi-collinear variables

Algorithms Comparison For MODEL - 1

60-40 Split Before Applying VIF

Algorithms	Accuracy
Logistic Regression	0.910
KNN	0.744
SVM	0.909
Decision Tree	0.931
Random Forest	0.922
AdaBoost	0.919
XGBoost	0.916

Decision Tree 60-40 split

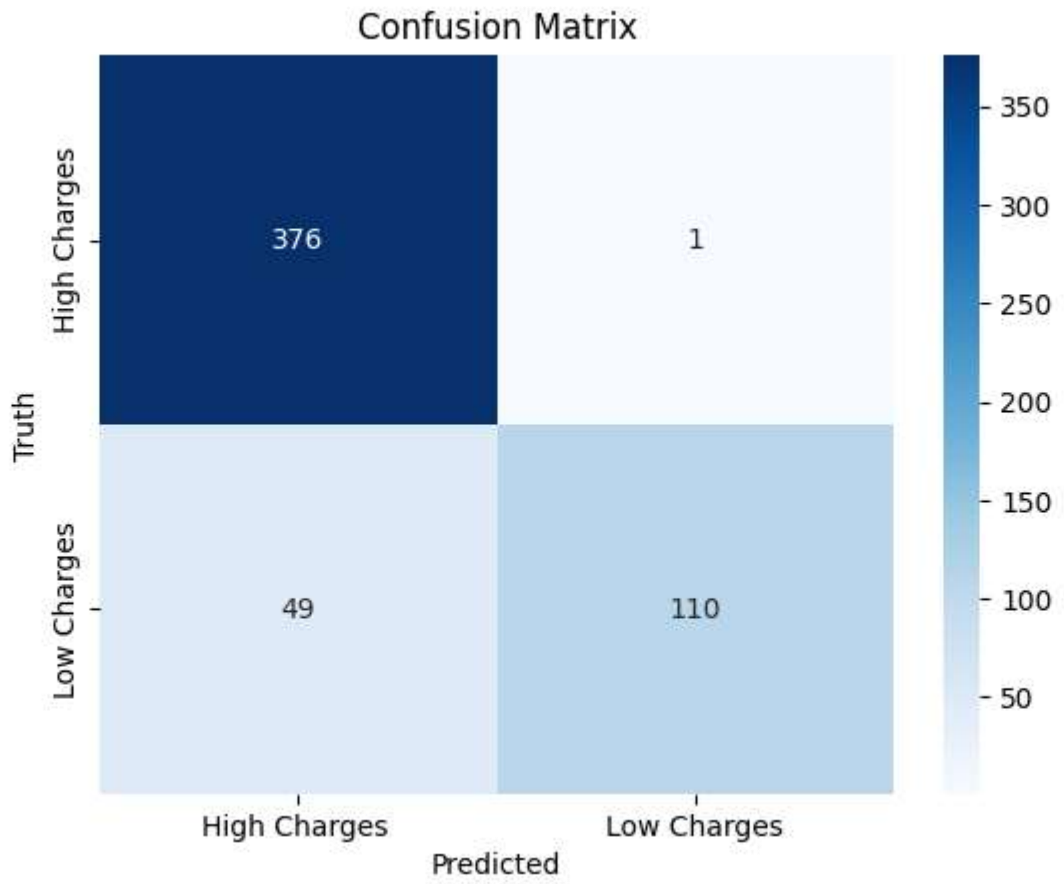


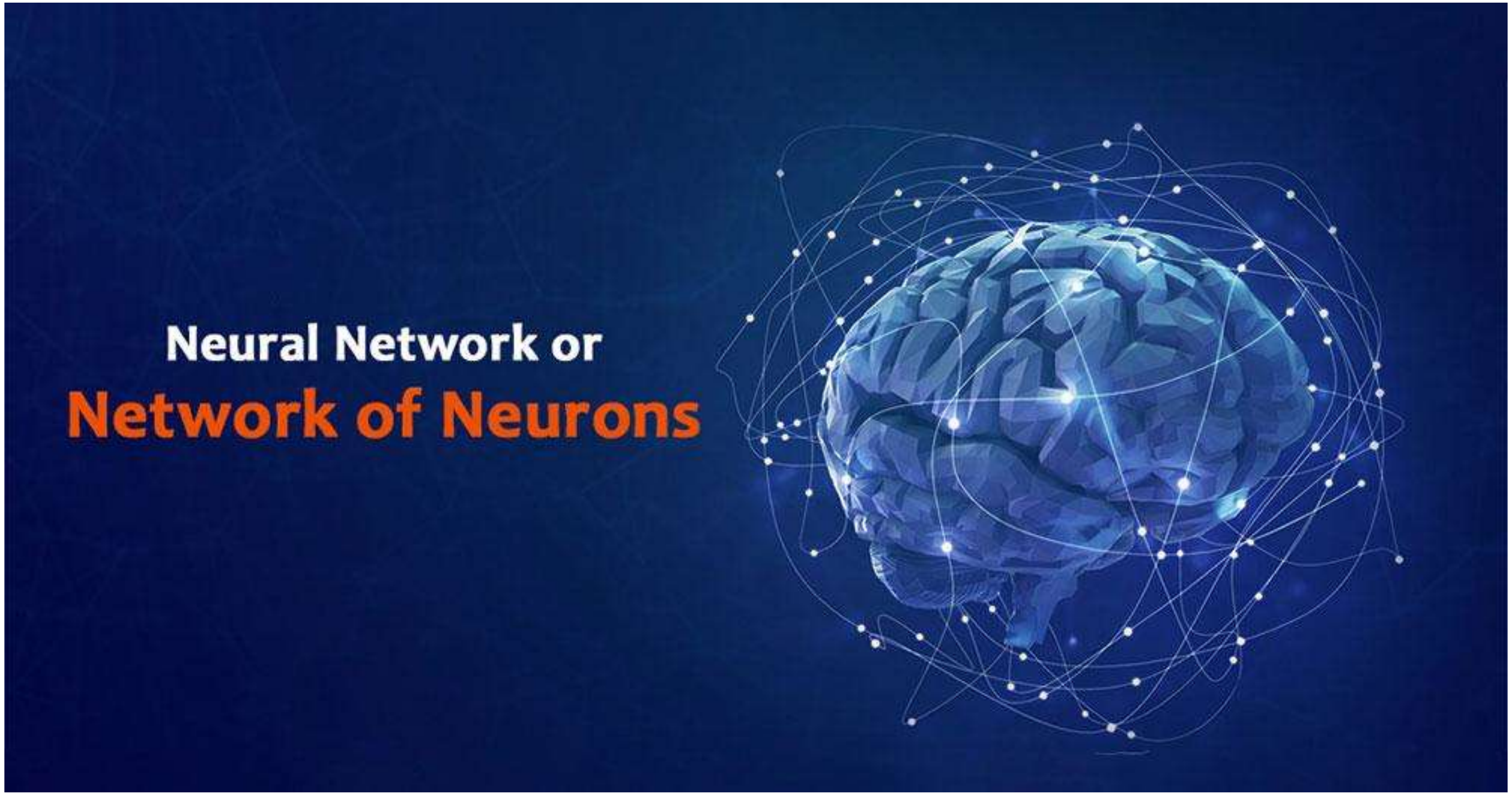
Algorithms Comparison For MODEL - 2

60-40 Split After Applying VIF

Algorithms	Accuracy
Logistic Regression	0.908
KNN	0.889
SVM	0.908
Decision Tree	0.904
Random Forest	0.904
AdaBoost	0.901
XGBoost	0.916

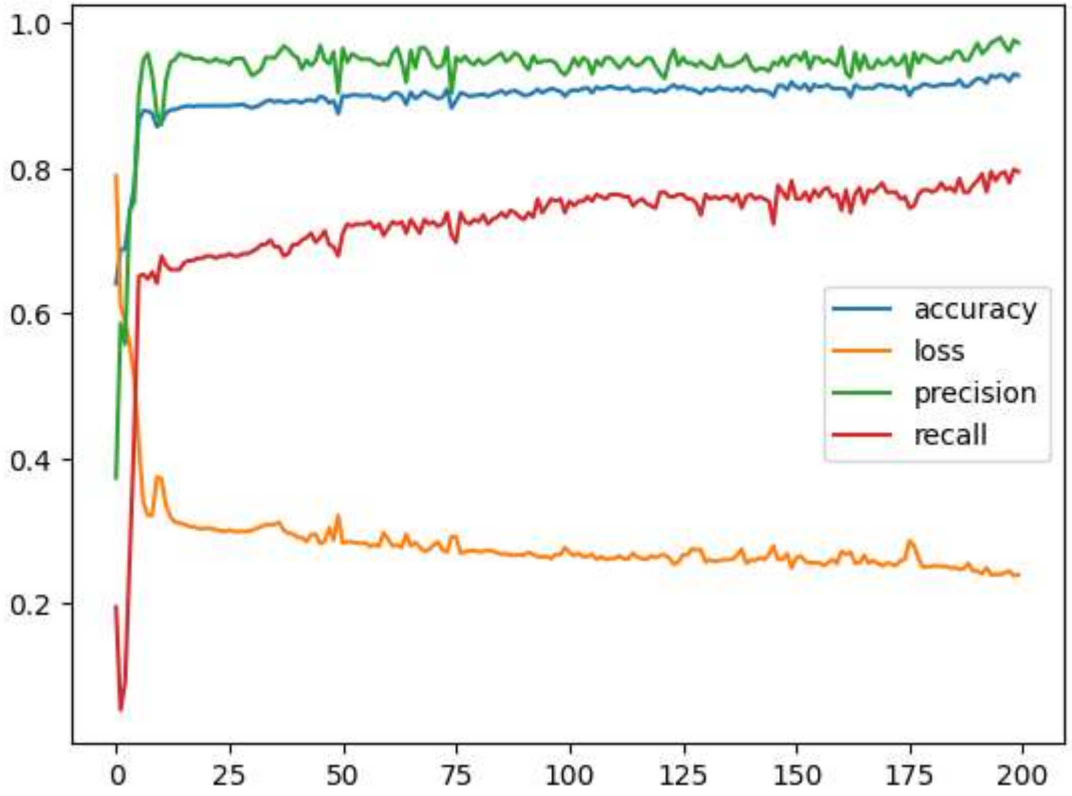
XGBoost 60-40 split After VIF




Neural Network or
Network of Neurons

Train test	Architecture	Optimizer	Epochs	Accuracy
80-20	10-7-5-1	Adam	200	0.8821
80-20	10-7-5-1	Adam	200	0.8793
75-25	10-7-5-1	Adam	200	0.8941
75-25	10-7-5-1	Adam	200	0.8846
70-30	10-7-5-1	Adam	200	0.6846
70-30	10-7-5-1	Adam	200	0.8921
60-40	10-7-5-1	Adam	200	0.6665

Neural Network Plot



Train Test Split	75-25
Architecture	10-7-5-1
Optimizer	Adam
Epochs	200

- The purpose of this research is to determine the best performing machine learning algorithms to predict the charges of Medical Insurance.
- For the Model 1, the best fit model is Decision Tree with accuracy of 93.1%.
- Whereas for Model 2, the best model is XGBoost algorithm.
- However, since the Model 2 accuracy does not contribute enough, the Model 1 Decision Tree is the best fit overall.

Insights

- **Personalized Premiums**: Tailor premium rates based on demographic data , potentially lowering costs for healthier lifestyles.
- **Preventive Health Programs**: Offer wellness programs (e.g., smoking cessation, fitness) to lower risk profiles and long-term costs.
- **Early Risk Intervention & Risk Management**: Identify and provide support for high-risk customers to reduce severe claims and improves risk management.
- **Competitive Edge**: Attract customers through personalized, flexible premiums and proactive health incentives.

Future Scope

- Incorporating additional demographic and health-related variables.
- Exploring deep learning models for more complex patterns.
- Real-Time Health Tracking: Integrate wearable health data for dynamic, real-time premium adjustments based on lifestyle changes.
- Explainable AI: Implement explainable AI techniques to make model decisions more transparent, helping customers understand premium calculations.

Work Distribution

M. BHARGAVI REDDY	Collecting basic information about medical insurance and Literature Review.
MOHAMMED AYAZ	Data preprocessing
K. ROHIT KUMAR	Exploratory Data Analysis
MD SAFWAN SHAWN	Applying Machine Learning Algorithms

Colab Notebook

THANK YOU

MD SAFWAN SHAWN
K. ROHIT KUMAR
MOHAMMED AYAZ
M. BHARGAVI REDDY

APPENDIX

Loading the dataset

```
[1]  import pandas as pd
     import numpy as np
```

```
[2]  from google.colab import files
     uploaded = files.upload()
```

```
Choose Files  Insurance.csv
• Insurance.csv(text/csv) - 55628 bytes, last modified: 8/26/2024 - 100% done
Saving Insurance.csv to Insurance.csv
```

Checking for null values

```
[7] df.isna().sum()

           0
age        0
sex        0
bmi        0
children   0
smoker     0
region     0
charges    0

dtype: int64
```

To know the data type

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Checking for Unique Values

```
for i in range(df.shape[1]):
    print(df.iloc[:,i].unique())
    print(df.iloc[:,i].value_counts())
```

```
62      23
64      22
Name: count, dtype: int64
['female' 'male']
sex
male      676
female    662
Name: count, dtype: int64
[27.9   33.77  33.    22.705 28.88  25.74  33.44  27.74  29.83  25.84
 26.22  26.29  34.4   39.82  42.13  24.6   30.78  23.845 40.3   35.3
 36.005 32.4   34.1   31.92  28.025 27.72  23.085 32.775 17.385 36.3
 35.6   26.315 28.6   28.31  36.4   20.425 32.965 20.8   36.67  39.9
 26.6   36.63  21.78  30.8   37.05  37.3   38.665 34.77  24.53  35.2
 35.625 33.63  28.    34.43  28.69  36.955 31.825 31.68  22.88  37.335
 27.36  33.66  24.7   25.935 22.42  28.9   39.1   36.19  23.98  24.75
 28.5   28.1   32.01  27.4   34.01  29.59  35.53  39.805 26.885 38.285
 37.62  41.23  34.8   22.895 31.16  27.2   26.98  39.49  24.795 31.3
 38.28  19.95  19.3   31.6   25.46  30.115 29.92  27.5   28.4   30.875
 27.94  35.09  29.7   35.72  32.205 28.595 49.06  27.17  23.37  37.1
 23.75  28.975 31.35  33.915 28.785 28.3   37.4   17.765 34.7   26.505
 22.04  35.9   25.555 28.05  25.175 31.9   36.    32.49  25.3   29.735
 38.83  30.495 37.73  37.43  24.13  37.145 39.52  24.42  27.83  36.85
 39.6   29.8   29.64  28.215 37.    33.155 18.905 41.47  30.3   15.96
 33.345 37.7   27.835 29.2   26.41  30.69  41.895 30.9   32.2   32.11
```

Converting the target variable to categorical

```
[15] df['ChargesNew']=pd.cut(df['charges'],bins=[0,13270,63772],labels=['0','1'])
     print(df)

      age     sex     bmi  children smoker     region      charges ChargesNew
0      19  female  27.900         0    yes  southwest  16884.92400          1
1      18    male  33.770         1     no  southeast   1725.55230          0
2      28    male  33.000         3     no  southeast   4449.46200          0
3      33    male  22.705         0     no  northwest  21984.47061          1
4      32    male  28.880         0     no  northwest   3866.85520          0
...   ...     ...     ...       ...    ...        ...          ...        ...
1333   50    male  30.970         3     no  northwest  10600.54830          0
1334   18  female  31.920         0     no  northeast   2205.98080          0
1335   18  female  36.850         0     no  southeast   1629.83350          0
1336   21  female  25.800         0     no  southwest   2007.94500          0
1337   61  female  29.070         0    yes  northwest  29141.36030          1

[1338 rows x 8 columns]
```

The converted Charges variable

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         1338 non-null   int64
 1   sex         1338 non-null   object
 2   bmi         1338 non-null   float64
 3   children    1338 non-null   int64
 4   smoker      1338 non-null   object
 5   region      1338 non-null   object
 6   charges     1338 non-null   float64
 7   ChargesNew  1338 non-null   category
dtypes: category(1), float64(2), int64(2), object(3)
memory usage: 74.7+ KB
```

Defining the Dependent and Independent Variables

```
[18] x= df.drop(["ChargesNew"],axis=1)
     y= df["ChargesNew"]
```

```
[19] x=pd.get_dummies(x,dtype='int',drop_first=True)
     print(x)
```

```
      age     bmi  children  sex_male  smoker_yes  region_northwest  \
0      19  27.900         0         0           1                 0
1      18  33.770         1         1           0                 0
2      28  33.000         3         1           0                 0
3      33  22.705         0         1           0                 1
4      32  28.880         0         1           0                 1
...   ...     ...       ...       ...         ...               ...
1333   50  30.970         3         1           0                 1
1334   18  31.920         0         0           0                 0
1335   18  36.850         0         0           0                 0
1336   21  25.800         0         0           0                 0
1337   61  29.070         0         0           1                 1

      region_southeast  region_southwest
0                    0                 1
1                    1                 0
2                    1                 0
```

Variables after dummifying

```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   age               1338 non-null   int64
 1   bmi               1338 non-null   float64
 2   children          1338 non-null   int64
 3   sex_male          1338 non-null   int64
 4   smoker_yes        1338 non-null   int64
 5   region_northwest  1338 non-null   int64
 6   region_southeast  1338 non-null   int64
 7   region_southwest  1338 non-null   int64
dtypes: float64(1), int64(7)
memory usage: 83.8 KB
```

Train-Test Split

```
[ ] import statsmodels.formula.api as smf
    from sklearn import metrics
    from sklearn.model_selection import train_test_split

    X_train1, X_test1, y_train1, y_test1 = train_test_split(x, y, test_size=0.20, train_size=0.80,random_state=0)
    X_train2, X_test2, y_train2, y_test2 = train_test_split(x, y, test_size=0.25, train_size=0.75,random_state=0)
    X_train3, X_test3, y_train3, y_test3 = train_test_split(x, y, test_size=0.30, train_size=0.70,random_state=0)
    X_train4, X_test4, y_train4, y_test4 = train_test_split(x, y, test_size=0.40, train_size=0.60,random_state=0)
```

Logistic Regression Before VIF

80-20

```
logreg = LogisticRegression(C=1e9)

logreg.fit(X_train1, y_train1)
predictions1 = logreg.predict(X_test1)
print(predictions1)
```

```
['0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0'
 '0' '1' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '0'
 '0' '1' '1' '0' '0' '0' '1' '1' '1' '0' '0' '0' '0' '0' '0' '0' '1' '1'
 '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '1' '1' '0' '0' '0' '0'
 '1' '0' '0' '0' '0' '0' '1' '1' '0' '1' '0' '0' '1' '1' '0' '0' '0' '1'
 '0' '0' '0' '0' '0' '0' '1' '1' '0' '0' '0' '0' '0' '1' '1' '0' '0' '0'
 '0' '1' '0' '0' '0' '1' '1' '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '0'
 '1' '1' '1' '0' '0' '0' '0' '0' '1' '1' '1' '0' '1' '0' '0' '0' '0' '0'
 '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 '0' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0'
 '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '1' '0' '0' '1' '0' '0' '0' '0'
 '0' '0' '0' '1' '0' '1' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 '0' '0' '0' '1' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '1' '1'
 '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '1' '0' '0' '0' '1' '0']
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
[ ] model=KNeighborsClassifier(n_neighbors=25)
    model.fit(X_train1, y_train1)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=25)
```

```
y_pred1 = model.predict(X_test1)
knn = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
knn
```

	Predicted	Actual
578	0	0
610	0	0
569	0	1
1034	1	0
198	0	0
...	...	...

KNN Before VIF

```
[ ] model1 = SVC(kernel='linear')
    model1.fit(X_train1, y_train1)
```

```
SVC
SVC(kernel='linear')
```

```
y_pred1 = model1.predict(X_test1)
svm = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
print(svm)
```

```
      Predicted Actual
578           0      0
610           0      0
569           1      1
1034          0      0
198           0      0
...         ...    ...
1084          0      1
726           0      0
1132          0      1
725           1      1
963           0      0

[268 rows x 2 columns]
```

SVM Before VIF

Decision Tree Before VIF

```
[ ] dt=DecisionTreeClassifier()
    dt.fit(X_train1,y_train1)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
[ ] y_pred1=dt.predict(X_test1)
```

```
[ ] print("Accuracy:",accuracy_score(y_test1,y_pred1))
    print(classification_report(y_test1, y_pred1))
```

```
Accuracy: 0.8656716417910447
              precision    recall  f1-score   support

           0       0.91      0.89      0.90       186
           1       0.77      0.80      0.79        82

    accuracy                           0.87       268
   macro avg       0.84      0.85      0.84       268
weighted avg       0.87      0.87      0.87       268
```

Random Forest Before VIF

```
[ ] rf=RandomForestClassifier()
    rf.fit(X_train1,y_train1)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
[ ] y_pred1=rf.predict(X_test1)
    print("Accuracy:",accuracy_score(y_test1,y_pred1))
    print(classification_report(y_test1, y_pred1))
    print(confusion_matrix(y_test1, y_pred1))
```

```
Accuracy: 0.9216417910447762
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       186
           1       0.97      0.77      0.86        82

    accuracy                           0.92       268
   macro avg       0.94      0.88      0.90       268
weighted avg       0.93      0.92      0.92       268

[[184   2]
```

AdaBoost Before VIF

```
from sklearn.ensemble import AdaBoostClassifier
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Replace 'base_estimator' with 'estimator'
base_estimator = DecisionTreeClassifier(max_depth=3, random_state=0)
adaboost = AdaBoostClassifier(estimator=base_estimator, # Changed argument name here
                              n_estimators=3,random_state=0)
```

80-20

```
adaboost.fit(X_train1, y_train1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is depre
  warnings.warn(
AdaBoostClassifier
estimator: DecisionTreeClassifier
```

XGBoost Before VIF

```
import xgboost as xgb
```

```
model1 = xgb.XGBClassifier()
model2 = xgb.XGBClassifier(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5)
```

```
y_train1 = y_train1.astype('int')
y_train2 = y_train2.astype('int')
y_train3 = y_train3.astype('int')
y_train4 = y_train4.astype('int')
y_test1= y_test1.astype('int')
y_test2= y_test2.astype('int')
y_test3= y_test3.astype('int')
y_test4= y_test4.astype('int')
```

80-20

```
model1.fit(X_train1, y_train1)
model2.fit(X_train1,y_train1)
```

Checking for Multicollinearity

VIF

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(x):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = x.columns
    vif["VIF"] = [variance_inflation_factor(x.values, i).round(1) for i in range(x.shape[1])]

    return(vif)

calc_vif(x)
```

	variables	VIF
0	age	7.7
1	bmi	11.4
2	children	1.8
3	sex_male	2.0
4	smoker_yes	1.3

Train-Test Split After VIF

AFTER REMOVING MULTICOLLINEAR VARIABLES

```
[ ] x_nomulti_train1,x_nomulti_test1,y_nomulti_train1,y_nomulti_test1=train_test_split(x_nomulti,y,test_size=0.20,random_state=0)
    x_nomulti_train2,x_nomulti_test2,y_nomulti_train2,y_nomulti_test2=train_test_split(x_nomulti,y,test_size=0.25,random_state=0)
    x_nomulti_train3,x_nomulti_test3,y_nomulti_train3,y_nomulti_test3=train_test_split(x_nomulti,y,test_size=0.30,random_state=0)
    x_nomulti_train4,x_nomulti_test4,y_nomulti_train4,y_nomulti_test4=train_test_split(x_nomulti,y,test_size=0.40,random_state=0)
```

Logistic Regression After VIF

```
logreg.fit(x_nomulti_train1, y_nomulti_train1)
nomulti_predictions1 = logreg.predict(x_nomulti_test1)
print(nomulti_predictions1)
```

```
['0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0'
 '0' '1' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '0'
 '0' '1' '1' '0' '0' '0' '1' '1' '1' '0' '0' '0' '0' '0' '0' '0' '1' '1'
 '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '1' '1' '0' '0' '0' '0'
 '1' '0' '0' '0' '0' '0' '1' '1' '0' '1' '0' '0' '1' '1' '0' '0' '0' '1'
 '0' '0' '0' '0' '0' '0' '1' '1' '0' '0' '0' '0' '0' '1' '1' '0' '0' '0'
 '0' '1' '0' '0' '0' '1' '1' '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '0'
 '1' '1' '1' '0' '0' '0' '0' '0' '1' '1' '1' '0' '1' '0' '0' '0' '0' '0'
 '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 '0' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0'
 '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '1' '0' '0' '1' '0' '0' '0' '0'
 '0' '0' '0' '1' '0' '1' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 '0' '0' '0' '1' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '1' '1'
 '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '1' '0' '0' '0' '1' '0']
```

```
[ ] z=confusion_matrix(y_nomulti_test1,nomulti_predictions1)
    print(z)
    print("Accuracy",accuracy_score(y_nomulti_test1,nomulti_predictions1))
    print(classification_report(y_nomulti_test1,nomulti_predictions1))
```

```
[[186   0]
 [ 27  55]]
```

KNN After VIF

```
[ ] model.fit(x_nomulti_train1, y_nomulti_train1)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=25)
```

```
y_pred1_nomulti = model.predict(x_nomulti_test1)
print("Accuracy:",accuracy_score(y_nomulti_test1,y_pred1_nomulti))
print(classification_report(y_nomulti_test1, y_pred1_nomulti))
cm = confusion_matrix(y_nomulti_test1,y_pred1_nomulti)
```

```
Accuracy: 0.8955223880597015
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       186
           1       1.00      0.66      0.79        82

    accuracy                           0.90       268
   macro avg       0.93      0.83      0.86       268
weighted avg       0.91      0.90      0.89       268
```

SVM After VIF

```
model1.fit(x_nomulti_train1, y_nomulti_train1)
```

```
SVC
SVC(kernel='linear')
```

```
y_pred1_nomulti = model1.predict(x_nomulti_test1)
print("Accuracy:",accuracy_score(y_nomulti_test1,y_pred1_nomulti))
print(classification_report(y_nomulti_test1, y_pred1_nomulti))
cm = confusion_matrix(y_nomulti_test1,y_pred1_nomulti)
```

```
Accuracy: 0.8992537313432836
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       186
           1       1.00      0.67      0.80        82

    accuracy                           0.90       268
   macro avg       0.94      0.84      0.87       268
weighted avg       0.91      0.90      0.89       268
```

Decision Tree After VIF

```
dt.fit(x_nomulti_train1,y_nomulti_train1)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred1_nomulti=dt.predict(x_nomulti_test1)
print("Accuracy:",accuracy_score(y_nomulti_test1,y_pred1_nomulti))
print(classification_report(y_nomulti_test1, y_pred1_nomulti))
```

```
Accuracy: 0.8955223880597015
              precision    recall  f1-score   support

           0       0.87      0.99      0.93       186
           1       0.98      0.67      0.80        82

    accuracy                           0.90       268
   macro avg       0.93      0.83      0.86       268
weighted avg       0.91      0.90      0.89       268
```

Random Forest After VIF

```
[ ] rf.fit(x_nomulti_train1,y_nomulti_train1)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
y_pred1_nomulti=rf.predict(x_nomulti_test1)
print("Accuracy:",accuracy_score(y_nomulti_test1,y_pred1_nomulti))
print(classification_report(y_nomulti_test1, y_pred1_nomulti))
print(confusion_matrix(y_nomulti_test1, y_pred1_nomulti))
```

```
Accuracy: 0.8955223880597015
              precision    recall  f1-score   support

           0       0.88      0.98      0.93       186
           1       0.95      0.70      0.80        82

    accuracy                           0.90       268
   macro avg       0.91      0.84      0.87       268
weighted avg       0.90      0.90      0.89       268

[[183   3]
 [ 25  57]]
```

AdaBoost After VIF

```
adaboost.fit(x_nomulti_train1, y_nomulti_train1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_bo
  warnings.warn(
    AdaBoostClassifier
  estimator: DecisionTreeClassifier
    DecisionTreeClassifier
```

```
[ ] y_pred1_nomulti = adaboost.predict(x_nomulti_test1)
    print("Accuracy:",accuracy_score(y_nomulti_test1,y_pred1_nomulti))
    print(classification_report(y_nomulti_test1, y_pred1_nomulti))
    print(confusion_matrix(y_nomulti_test1, y_pred1_nomulti))
```

```
Accuracy: 0.8955223880597015
              precision    recall  f1-score   support

           0       0.87      0.99      0.93       186
           1       0.98      0.67      0.80        82

    accuracy                           0.90       268
   macro avg       0.93      0.83      0.86       268
weighted avg       0.91      0.90      0.89       268

[[185   1]
```

XGBoost After VIF

```
model1.fit(x_nomulti_train1, y_nomulti_train1)
model2.fit(x_nomulti_train1,y_nomulti_train1)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=8, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
[ ] pred1_nomulti = model1.predict(x_nomulti_test1)
    pred2_nomulti = model2.predict(x_nomulti_test1)

    print('Model 1 XGboost Report %r' % (classification_report(y_nomulti_test1, pred1_nomulti)))
    print('Model 2 XGboost Report %r' % (classification_report(y_nomulti_test1, pred2_nomulti)))
```

ANN

```
[ ] tf.random.set_seed(42)

    # STEP 1: Creating the model
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(10, activation='relu'),
        tf.keras.layers.Dense(7, activation='relu'),
        tf.keras.layers.Dense(5, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    # STEP 2: Compiling the model
    model.compile(
        loss=tf.keras.losses.binary_crossentropy,
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),  # Corrected here
        metrics=[
            tf.keras.metrics.BinaryAccuracy(name='accuracy'),
            tf.keras.metrics.Precision(name='precision'),
            tf.keras.metrics.Recall(name='recall')  # Removed typo 'a=recall'
        ]
    )

    # STEP 3: Fit the model
    history = model.fit(X_train1, y_train1, epochs=200)
```

```
Epoch 172/200
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8821 - loss: 0.3001 - precision: 1.0000 - recall: 0.6400
Epoch 173/200
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8793 - loss: 0.3007 - precision: 0.9870 - recall: 0.6400
Epoch 174/200
```

After VIF

```
tf.random.set_seed(42)

# STEP 1: Creating the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(7, activation='relu'),
    tf.keras.layers.Dense(5, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP 2: Compiling the model
model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),  # Corrected here
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')  # Removed typo 'a=recall'
    ]
)

# STEP 3: Fit the model
history = model.fit(x_nomulti_train1, y_nomulti_train1, epochs=200)
```

```
Epoch 172/200
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8793 - loss: 0.3257 - precision: 0.9870 - recall: 0.6400
Epoch 173/200
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8793 - loss: 0.3265 - precision: 0.9870 - recall: 0.6400
Epoch 174/200
34/34 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8793 - loss: 0.3296 - precision: 0.9870 - recall: 0.6400
```