



MEDICAL INSURANCE PREMIUM PREDICTION USING ML ALGORITHMS

BHAVANS VIVEKANANDA COLLEGE

Group-11

N.AISHWARYA(107222546029)

RISHI BHARADWAJ (107222546047)

AJAY KUMAR (107222546045)

JAI KUMAR(107222546046)

Abstract

This project seeks to create a predictive model for estimating medical insurance premiums based on factors like age, BMI, smoking status, and region.

Utilizing machine learning, the model aids insurance companies in determining fair premiums and helps consumers understand the financial effects of their health choices, promoting transparency in healthcare insurance.

Objective

To develop a strong machine learning framework to predict medical insurance premiums using health and demographic data for fair, data-driven calculations.



CONTENT

- **Introduction** 4
- **Literature Review** 5
- **Data Preprocessing** 8
- **Exploratory Data Analysis** 12
- **Data Modelling & Evaluation** 19
- **Summary** 28
- **Appendix** 32

Introduction

Medical insurance is vital for financial security, covering healthcare costs. As medical expenses rise, accurately predicting insurance premiums is crucial for affordability and transparency for both insurers and policyholders.

Advancements in data analytics and machine learning are transforming healthcare by enabling insurance providers to predict premiums based on health and demographic factors. This allows for fair, personalized premiums and helps customers make informed health and financial choices.



LITERATURE REVIEW



Literature Review - 1

Author(s): Smith & Johnson

Title: Predictive Modeling of Medical Insurance Premiums Using Machine Learning

Published In: Journal of Health Economics, 2021

This study investigates machine learning models for predicting medical insurance premiums. The authors evaluated algorithms, including linear regression, decision trees, and deep learning, to assess their effectiveness in forecasting costs. It emphasizes the influence of factors like age, medical history, and demographics on premium calculations.

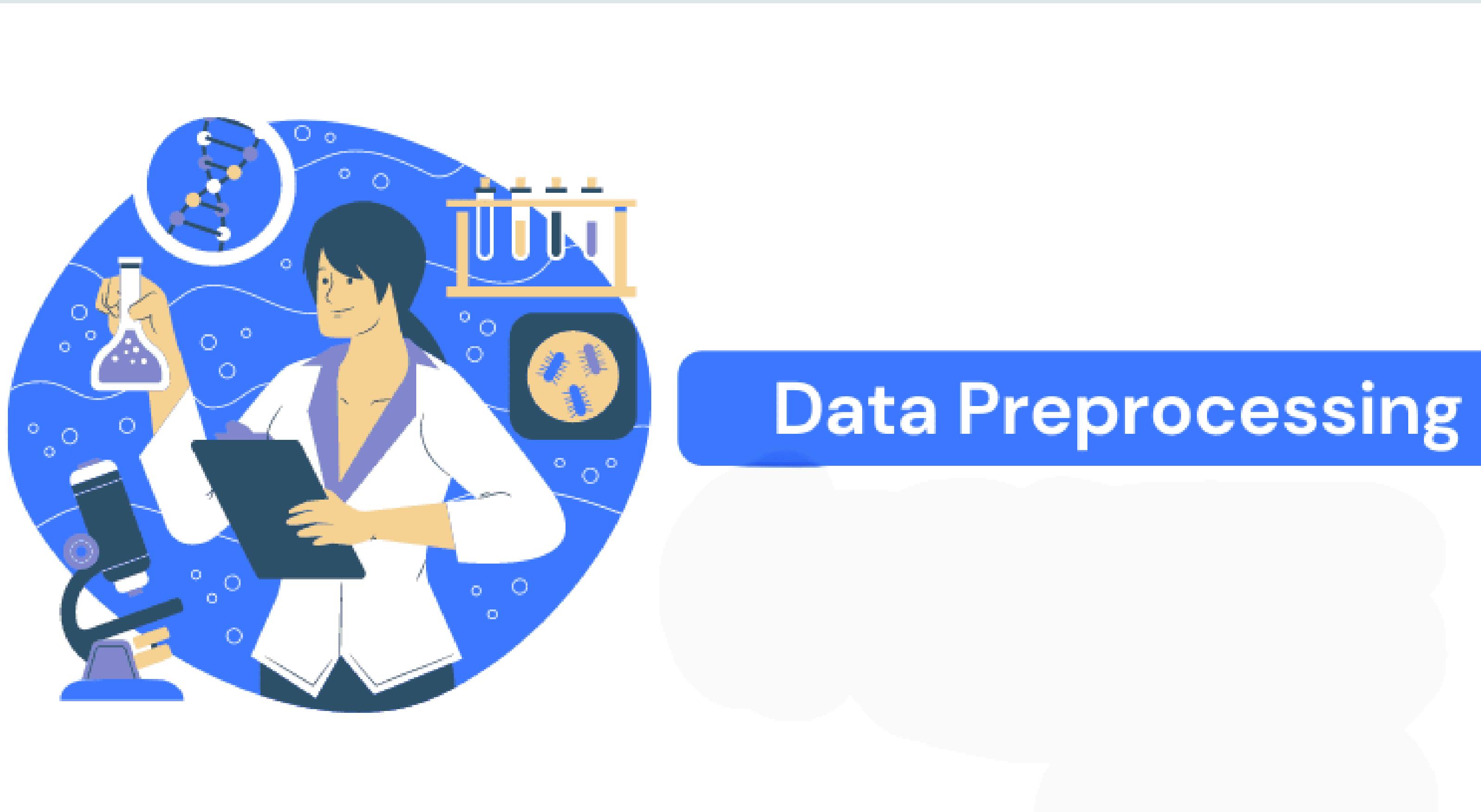
The paper highlights the advantages of machine learning techniques in improving the accuracy of premium predictions over traditional actuarial methods. The authors emphasize that machine learning models can help insurance companies personalize premium structures, thereby enhancing customer satisfaction and operational efficiency

Literature Review - 2

Authors: Satyabrata Dash, Bhawani Sankar Panigrahi, B K Madhavi, Susanta Kumar Sahoo, Vijay Venkata Bhaskar Reddy Sanikomu
Title: A Comparative Analysis of Different Machine Learning Techniques for Medical Insurance Premium Prediction
Published in : International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), IEEE -2023

To create predictive models, the study combined various regression approaches, including logistic regression, Random Forest Regression, Decision Tree Regression, Gradient Boosting Regression, and linear regression.

The predictive models were highly accurate, with a mean absolute error of less than 5% across all examined samples. The findings have significant implications for both insurers and customers, as accurate premium pricing is essential for insurers to maintain their financial health and offer customers coverage at reasonable prices.



Data Preprocessing

DATA

- **Dataset :** Our Dataset consists of 7 variables and 1338 records
- **Source :** https://drive.google.com/drive/folders/1vGSRChqSxEH53BgLqhh32F1qNKqfOim?usp=drive_link
- **About the dataset:**

A medical insurance company aims to deepen its understanding of the determinants impacting individual health insurance costs. They have gathered a dataset featuring diverse individual profiles, encompassing details such as age, gender, BMI, number of children, smoking status, and region.

Age (Age of the person)

Sex (Gender of the person)

BMI (Body Mass Index)

Children (Number of children)

Smoker (smoker or non smoker)

Region

Charges (Insurance price)

• Data Variables :

Categorical variables

Continuous variables

smoker

age

region

bmi

sex

children

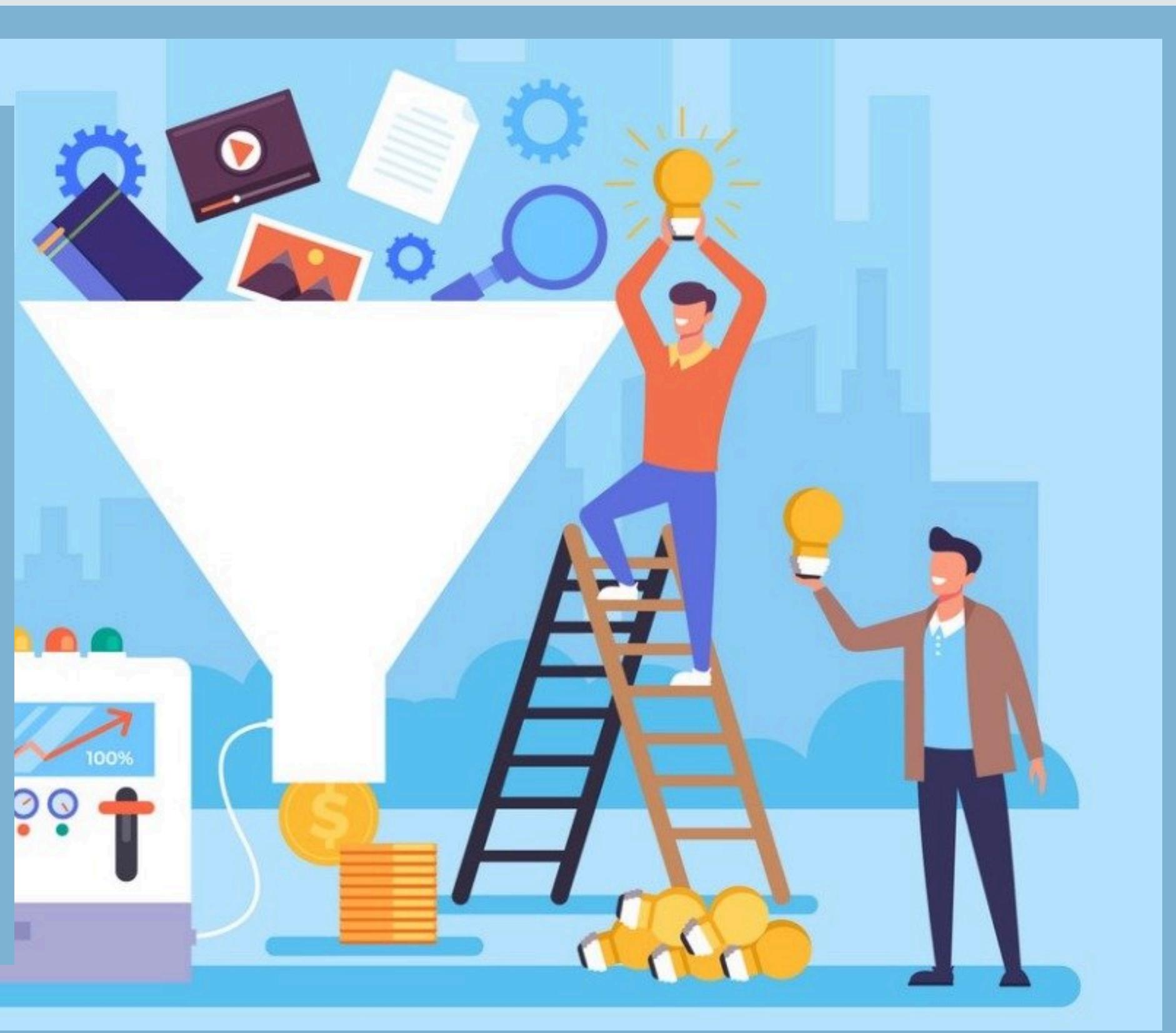
charges

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

1338 rows × 7 columns

Data Cleaning

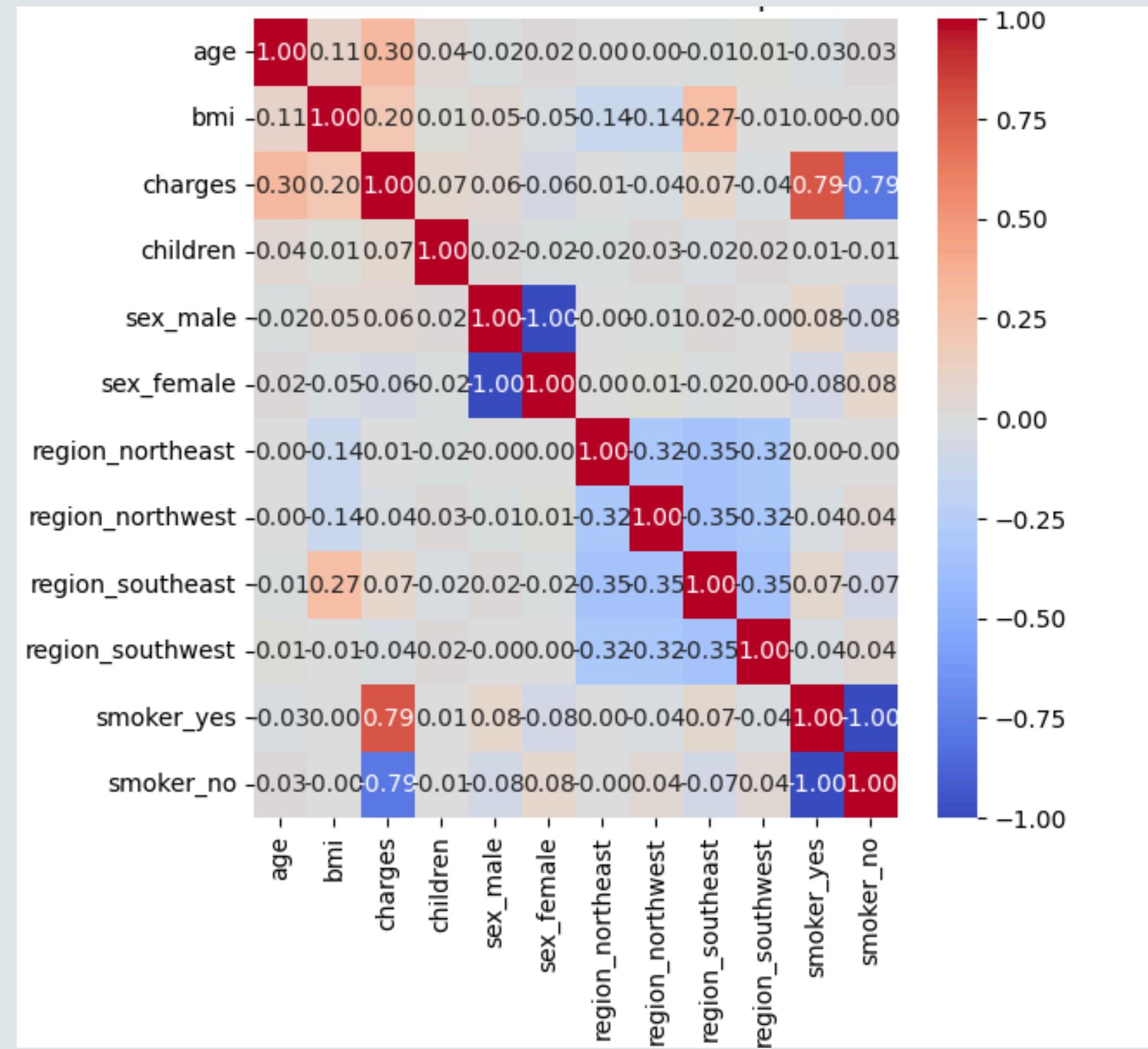
- **Removed duplicate row from the data.**
- **Checked for missing and unique values.**
- **Dummified categorical variables:**
 - **Sex as sex_male and sex_female.**
 - **Region as region_northeast, region_northwest, region_southeast, and region_southwest.**
 - **Smoker column in binary format (0 and 1).**



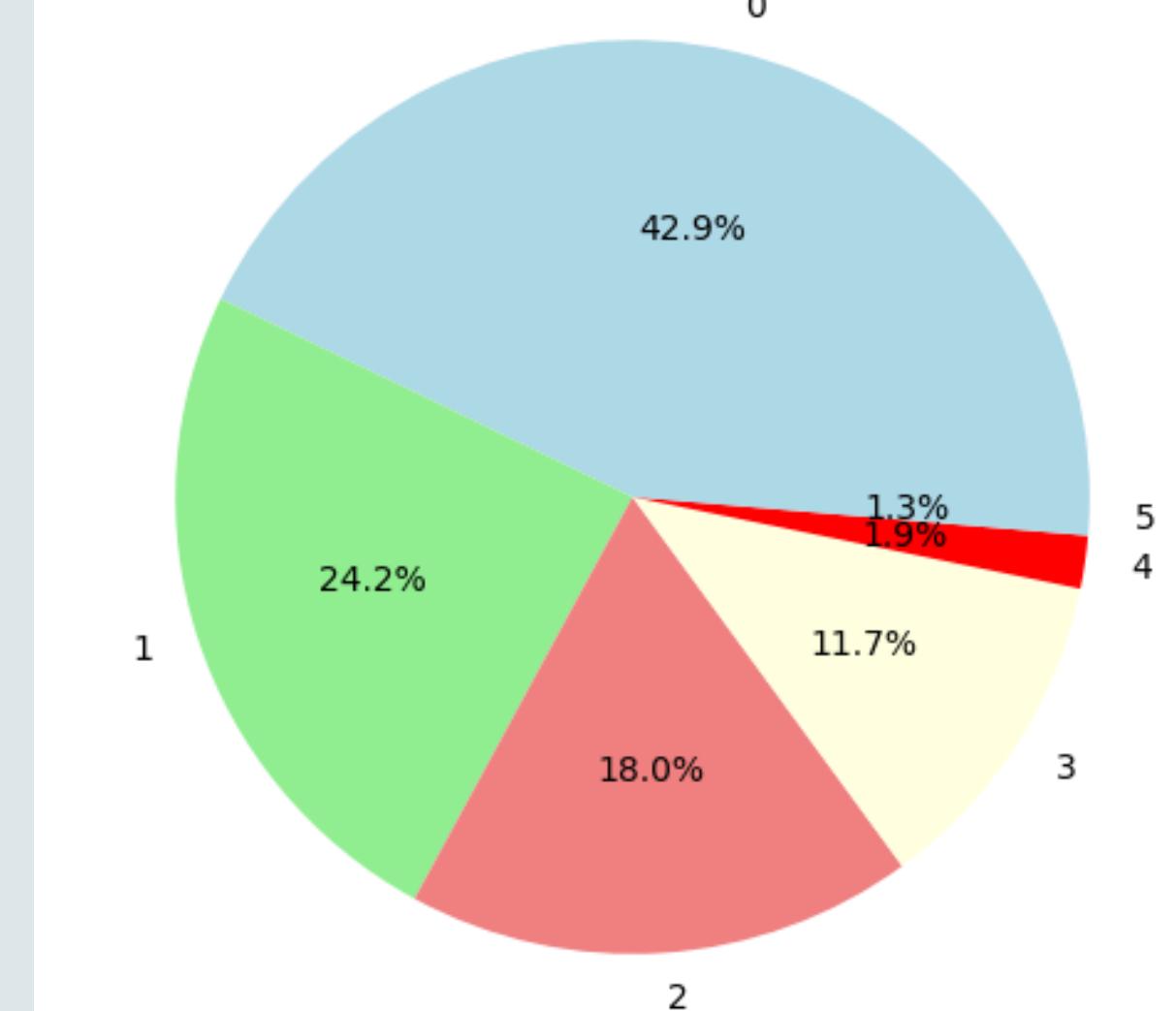
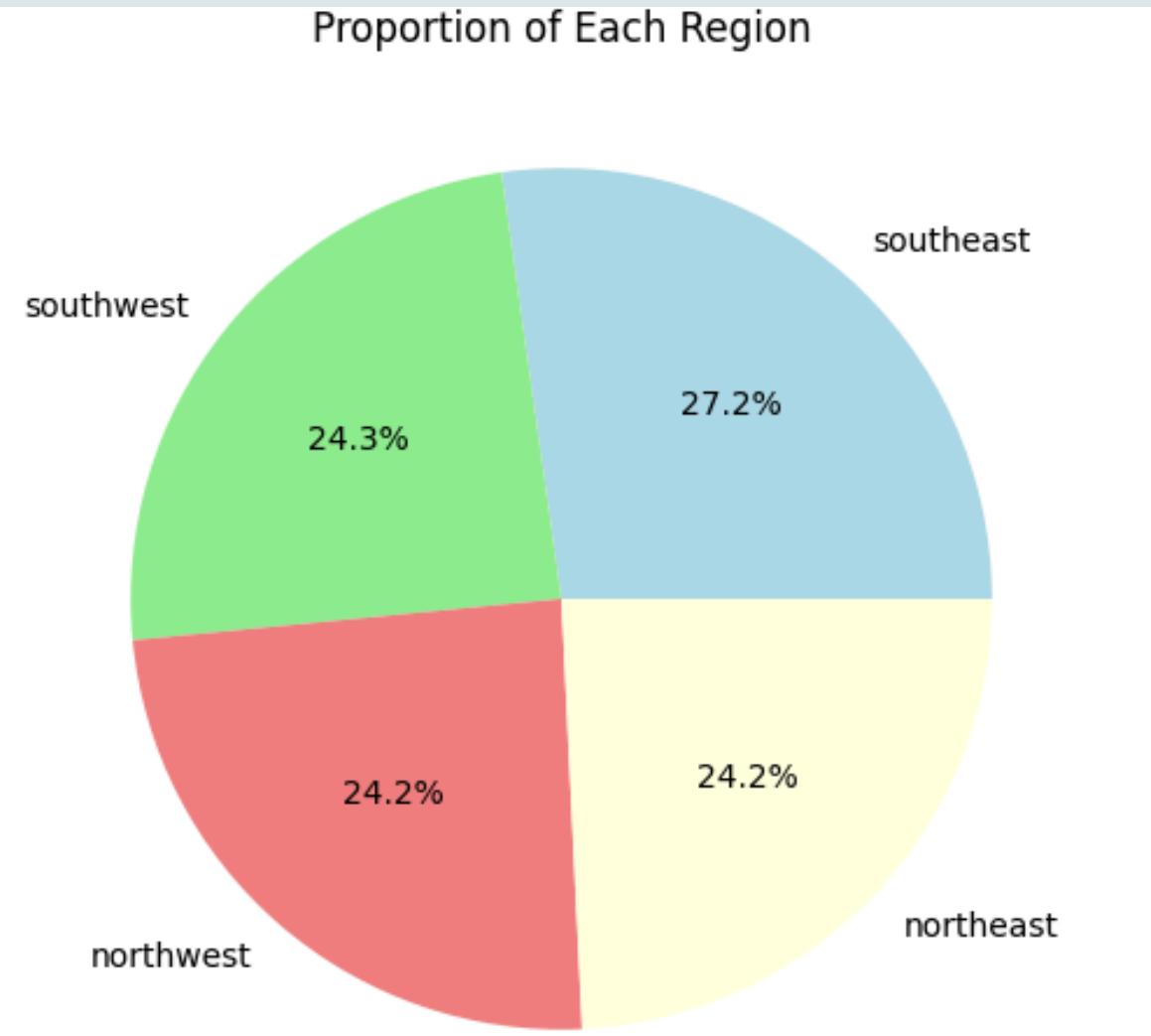
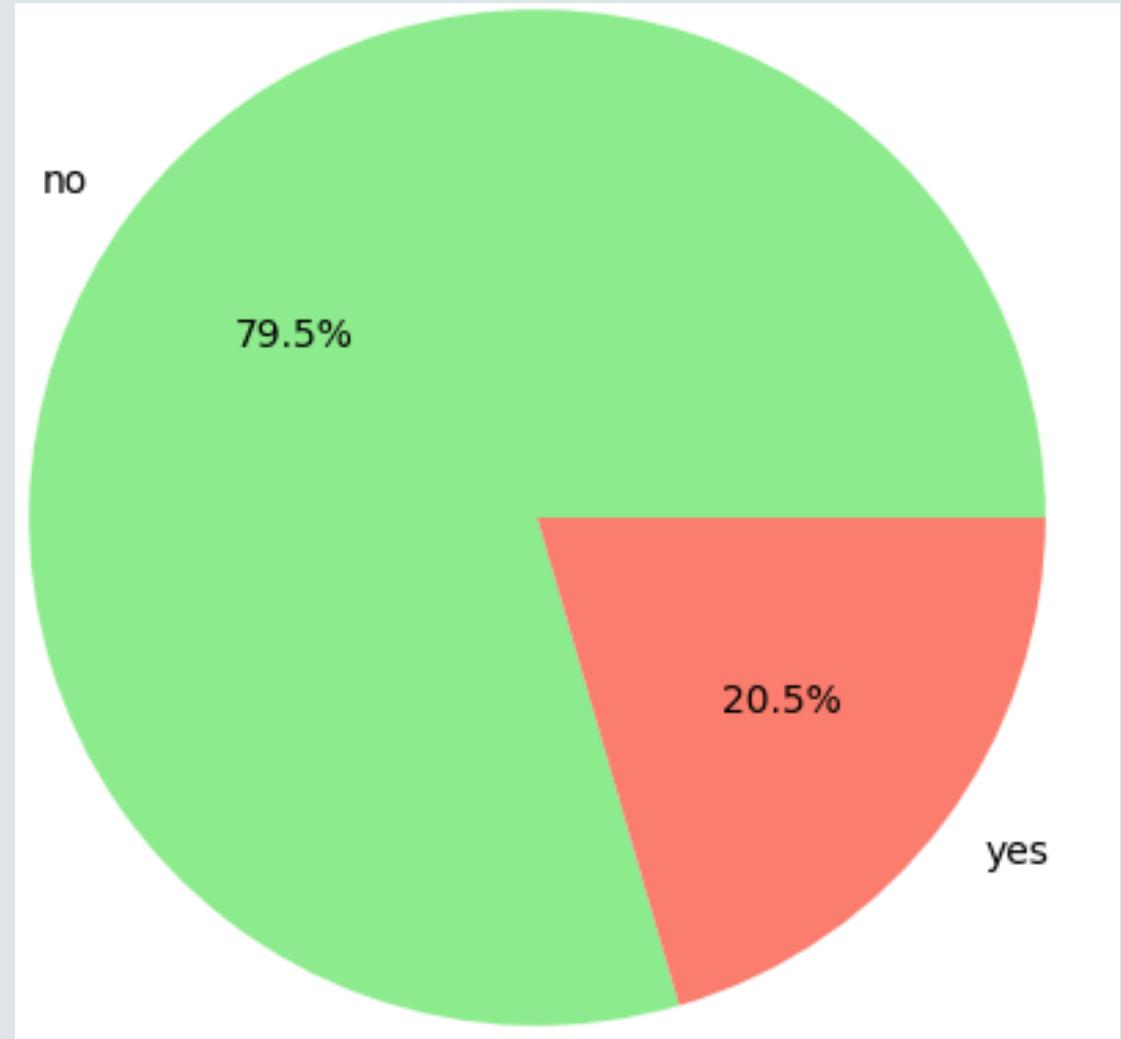
Exploratory Data Analysis



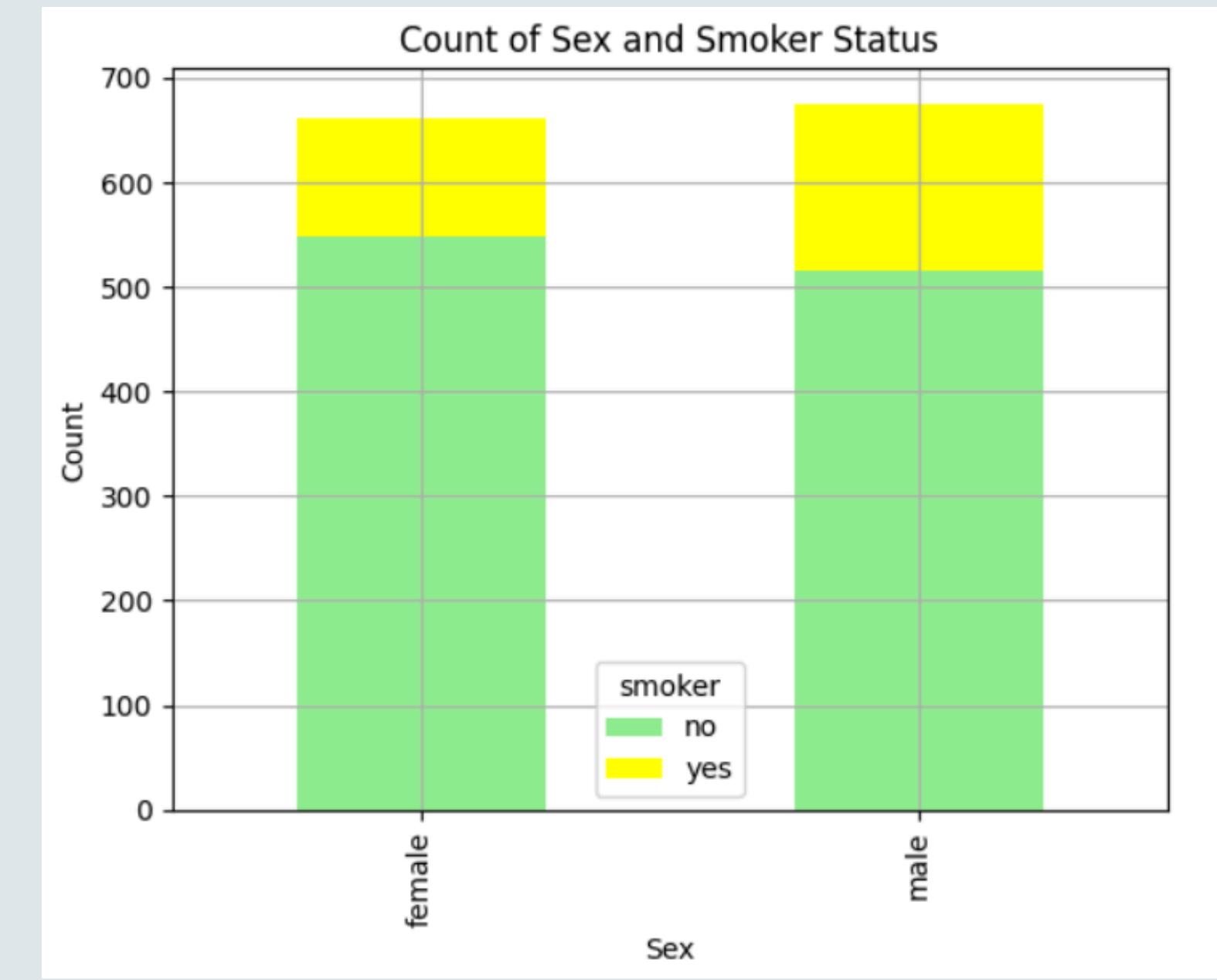
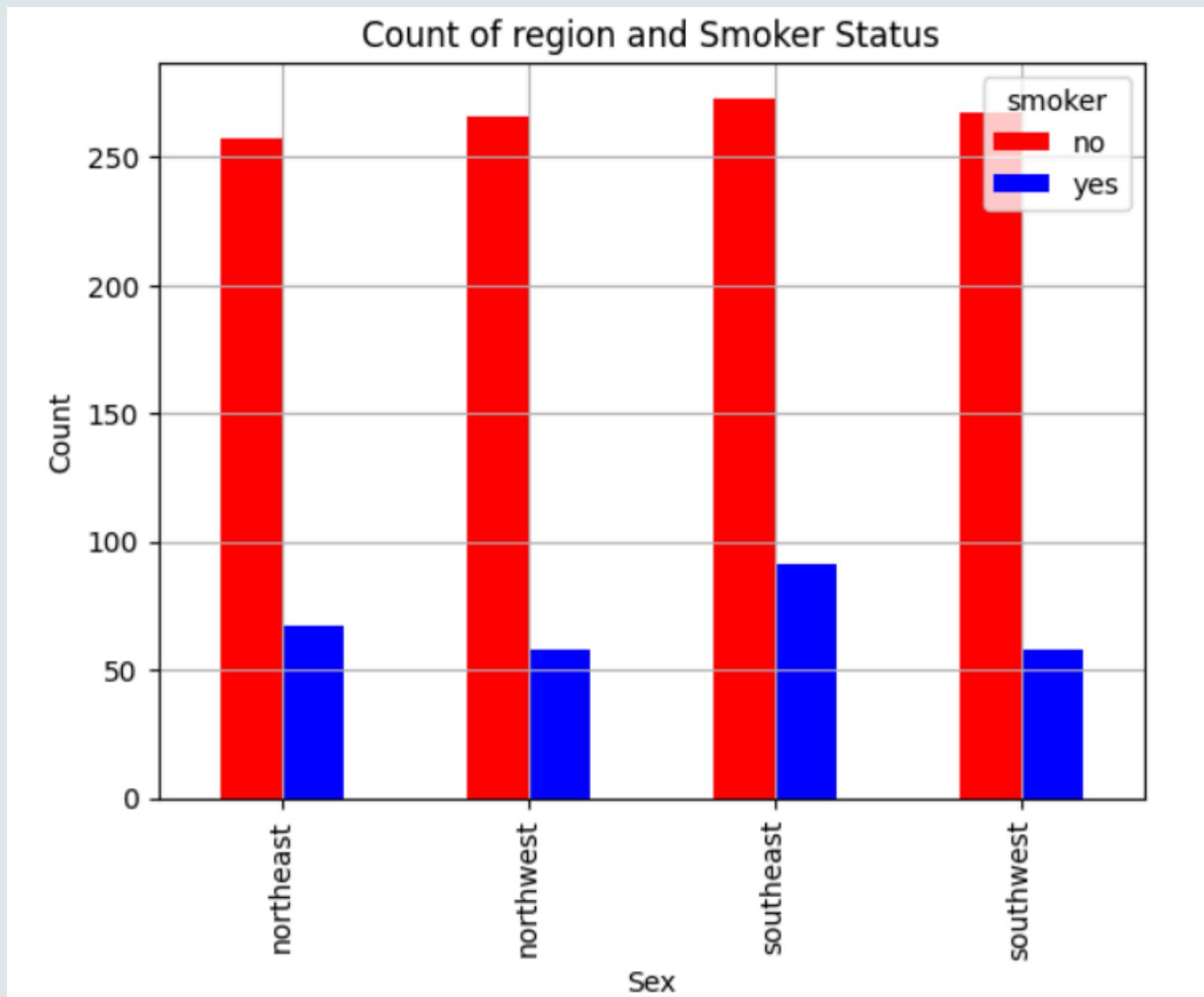
CORRELATION MATRIX



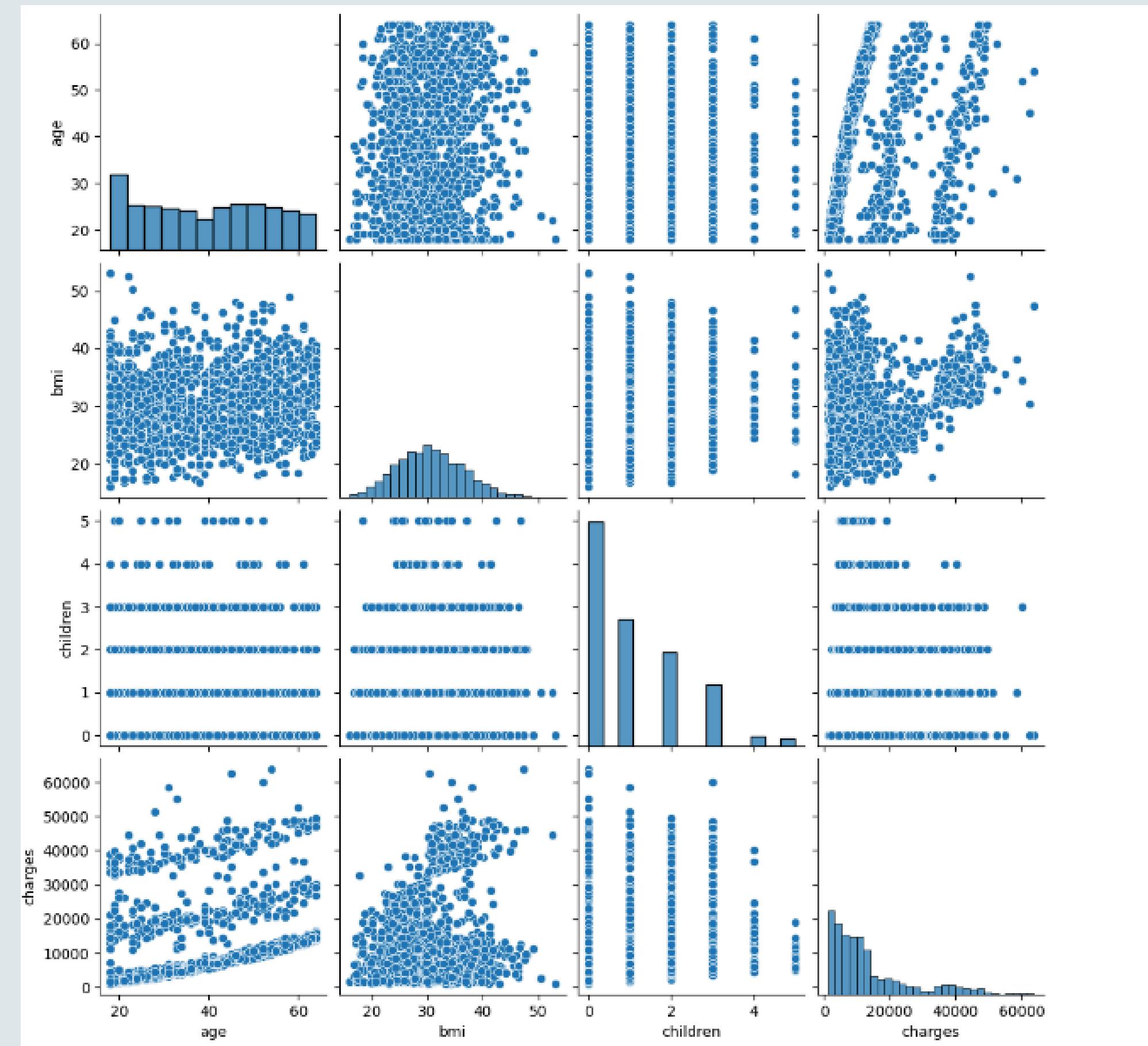
PIE CHART



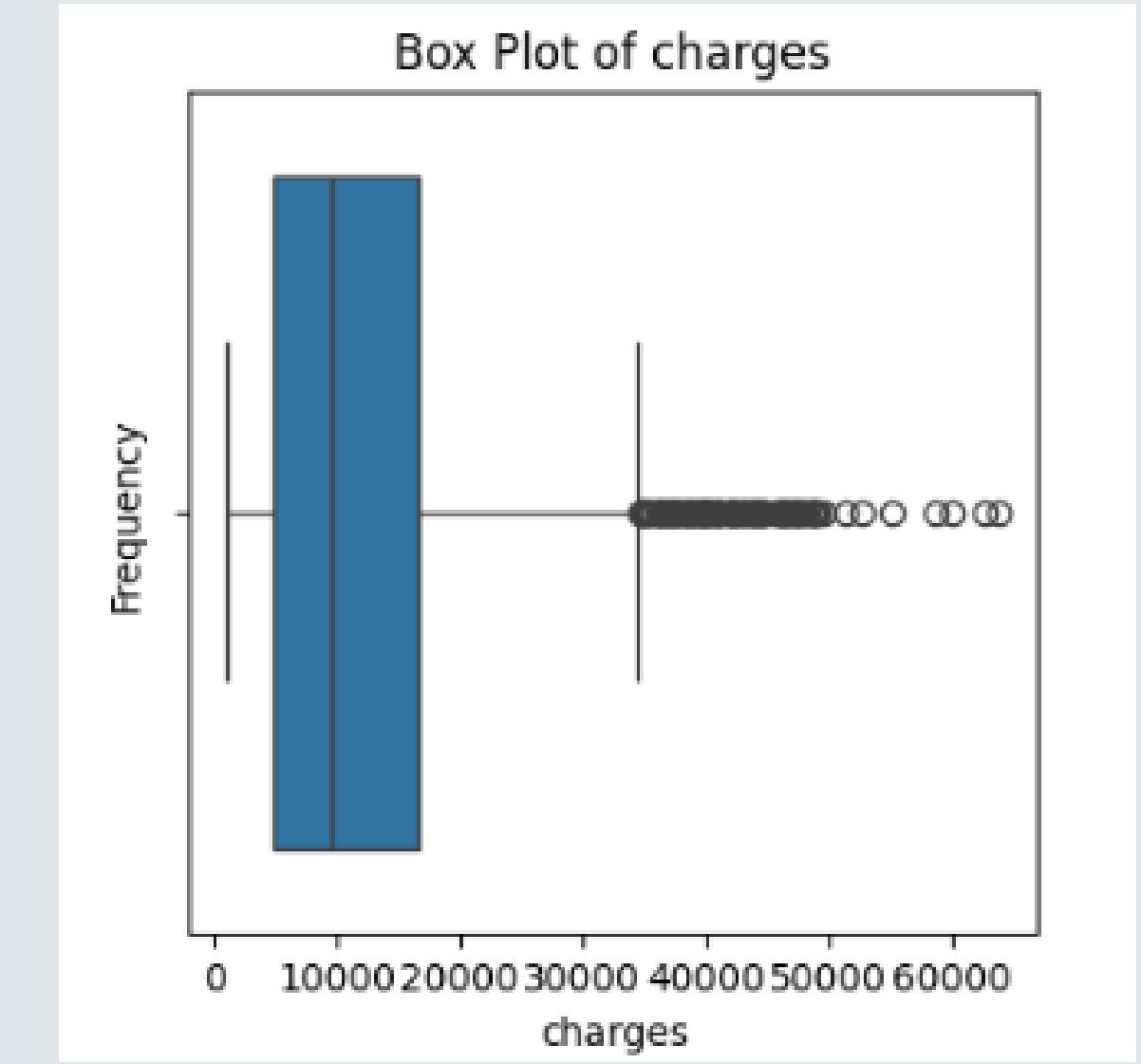
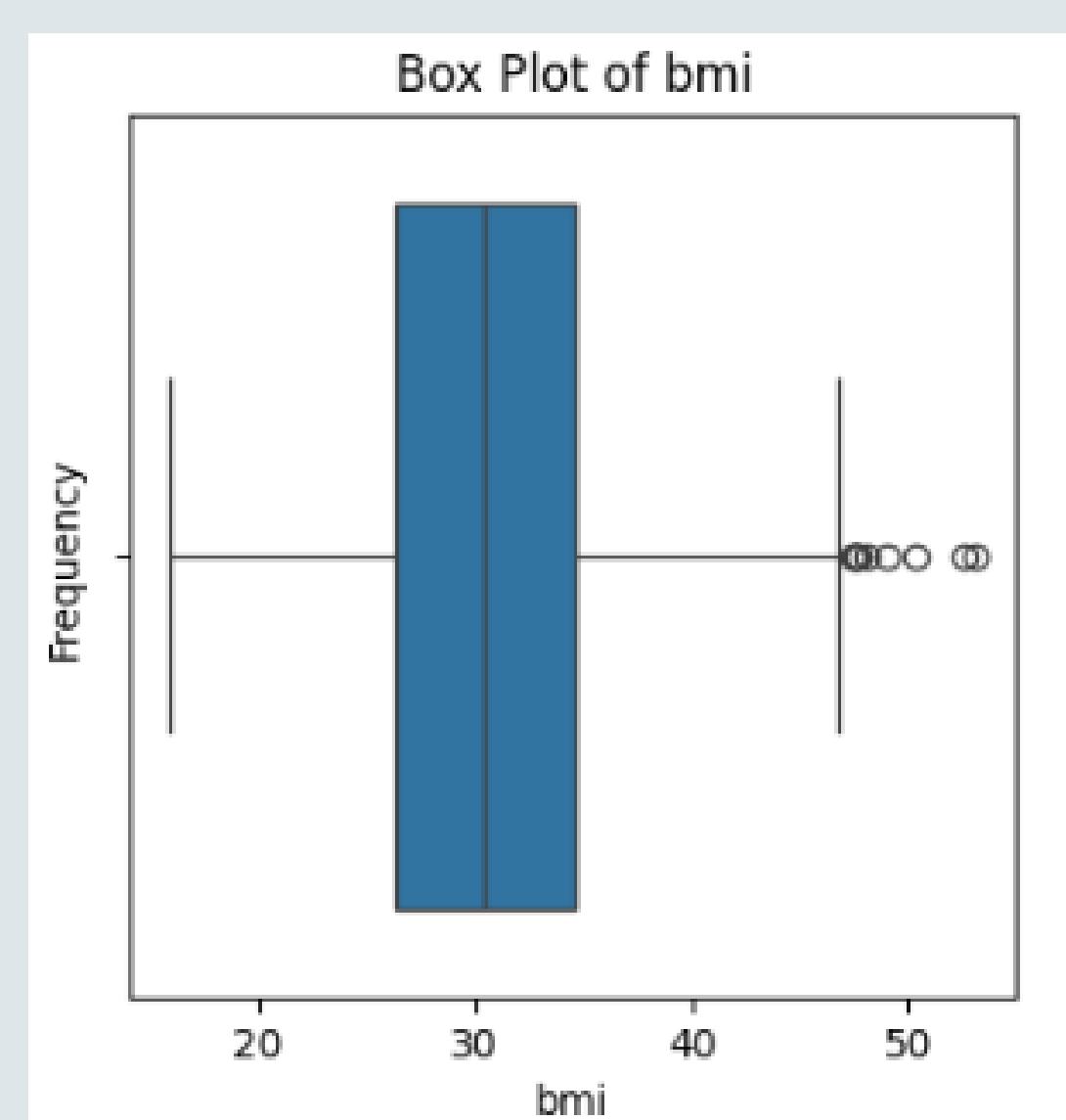
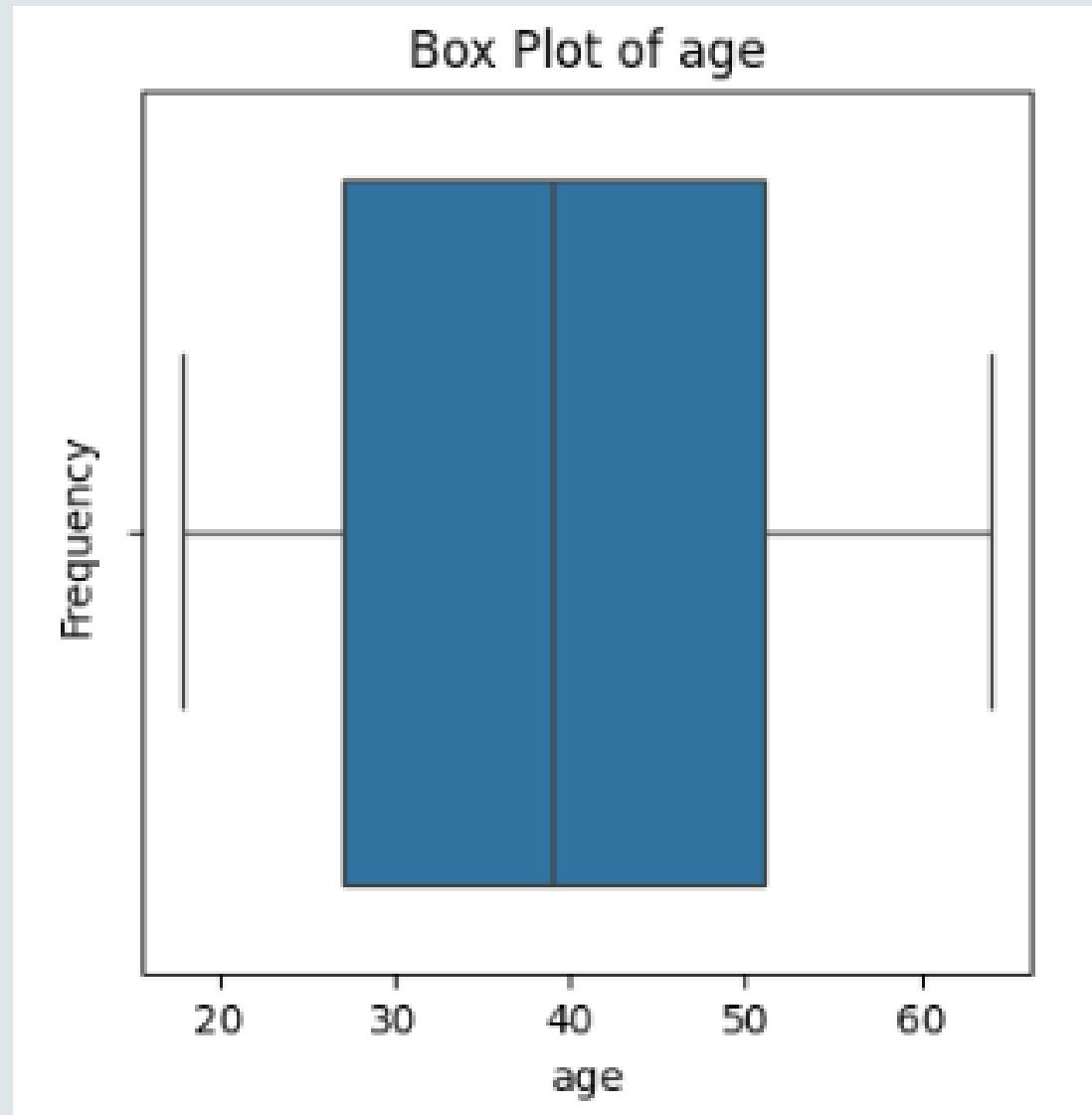
BAR PLOT



PAIR PLOT



BOX PLOT



Multicollinearity Check

Variables with the greatest variance inflation factor (VIF > 2) were removed

	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



	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



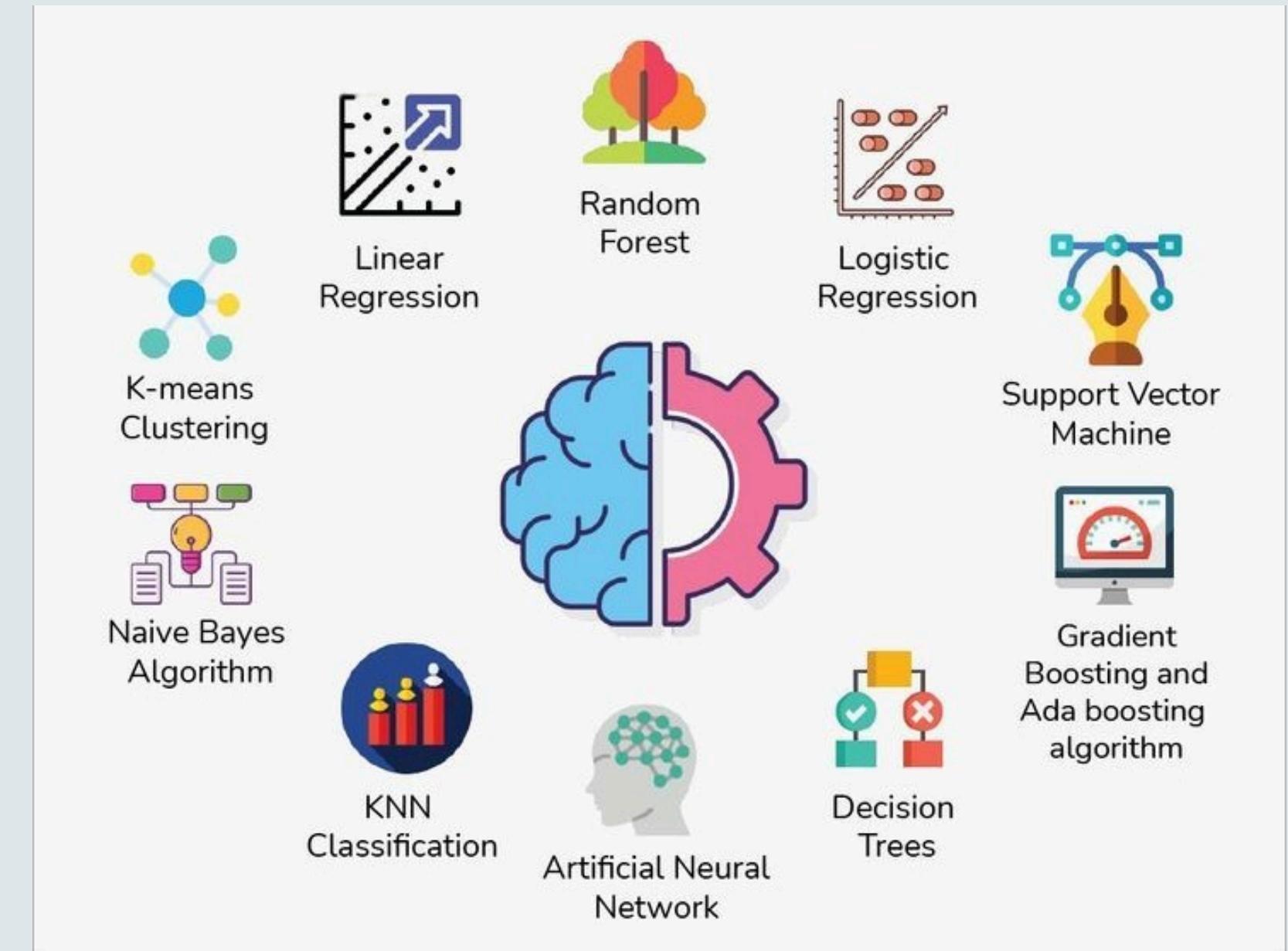
	variables	VIF
0	children	1.6
1	sex_male	1.7
2	smoker_yes	1.2
3	region_northwest	1.3
4	region_southeast	1.4
5	region_southwest	1.3

bmi was removed

age was removed

ML Algorithms

- **Linear Regression**
- **K-Nearest Neighbors(KNN)**
- **Decision Tree**
- **Random Forest**
- **XG Boosting**
- **ADA Boosting**
- **Support Vector Machine(SVM)**



80-20

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.750	0.647	3924.64	5184.0847
KNN	0.164	0.586	7452.49	5464.843
Decision Tree	0.632	0.635	3251.76	5126.588
Random Forest	0.829	0.638	2631.44	5080.645
XGBoost	0.794	0.638	3030.09	5080.645
ADABOOST	0.820	0.640	3928.00	5080.65
SVM	-0.037	-0.037	7409.86	7450.926

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

r2 score- represents goodness of fit of a model

MAE- absolute difference b/w predicted and actual values

75-25

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.720	0.811	4151.721	3954.255
KNN	0.160	0.546	7430.074	5628.108
Decision Tree	0.628	0.606	3216.599	5346.590
Random Forest	0.816	0.592	2769.744	5367.993
XGBoost	0.793	0.592	3010.910	5367.993
ADABOOST	0.810	0.59	3954.25	5367.99
SVM	-0.046	-0.045	7420.140	7454.331

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

r2 score- represents goodness of fit of a model

MAE- absolute difference b/w predicted and actual values

70-30

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.733	0.811	4207.80	3987.05
KNN	0.125	0.559	7821.08	5723.50
Decision Tree	0.656	0.615	3296.05	5388.69
Random Forest	0.825	0.606	2817.397	5417.70
XGBoost	0.815	0.607	2995.79	5417.70
ADABOOST	0.810	0.610	3987.05	5417.70
SVM	-0.056	-0.055	7753.014	7781.744

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

r2 score- represents goodness of fit of a model

MAE- absolute difference b/w predicted and actual values

60-40

Algorithms	Model-1 (r2 score)	Model-2 (r2 score)	Model-1 (MAE)	Model-2 (MAE)
Linear Regression	0.720	0.799	4274.29	4148.44
KNN	0.078	0.493	7949.71	6018.83
Decision Tree	0.622	0.573	3523.32	5583.30
Random Forest	0.819	0.573	2782.36	5583.30
XGBoost	0.785	0.573	3034.98	5583.31
ADABOOST	0.800	0.570	4148.44	5583.31
SVM	-0.054	-0.053	7754.86	7778.46

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

r2 score- represents goodness of fit of a model

MAE- absolute difference b/w predicted and actual values

Algorithms Comparison :

80-20

Model-1

75-25

Model -2

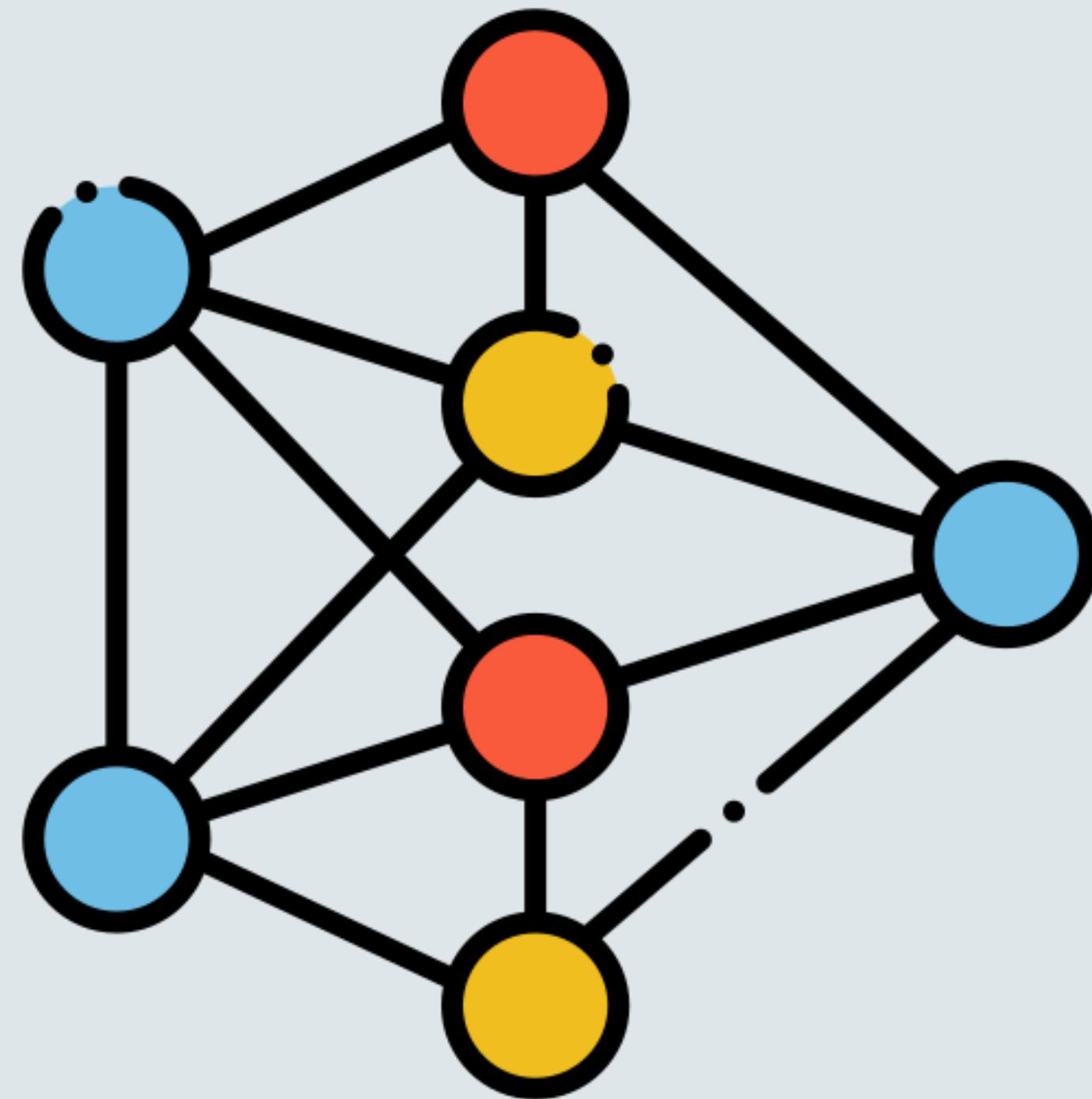
Algorithms	r2 score	MAE	Algorithms	r2 score	MAE
Linear Regression	0.750	3924.64	Linear Regression	0.811	3954.26
K- nearest numbers	0.164	7452.49	K- nearest numbers	0.546	5628.11
Decision tree	0.632	3251.76	Decision tree	0.606	5346.59
Random forest	0.829	2631.44	Random forest	0.592	5367.993
XG Boosting	0.794	3030.09	XG Boosting	0.592	5367.993
ADA Boosting	0.820	3928.00	ADA Boosting	0.590	5367.99
SVM	-0.037	7409.86	SVM	-0.045	7454.33

r2 score- represents goodness of fit of a model

MAE- absolute difference b/w predicted and actual values

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

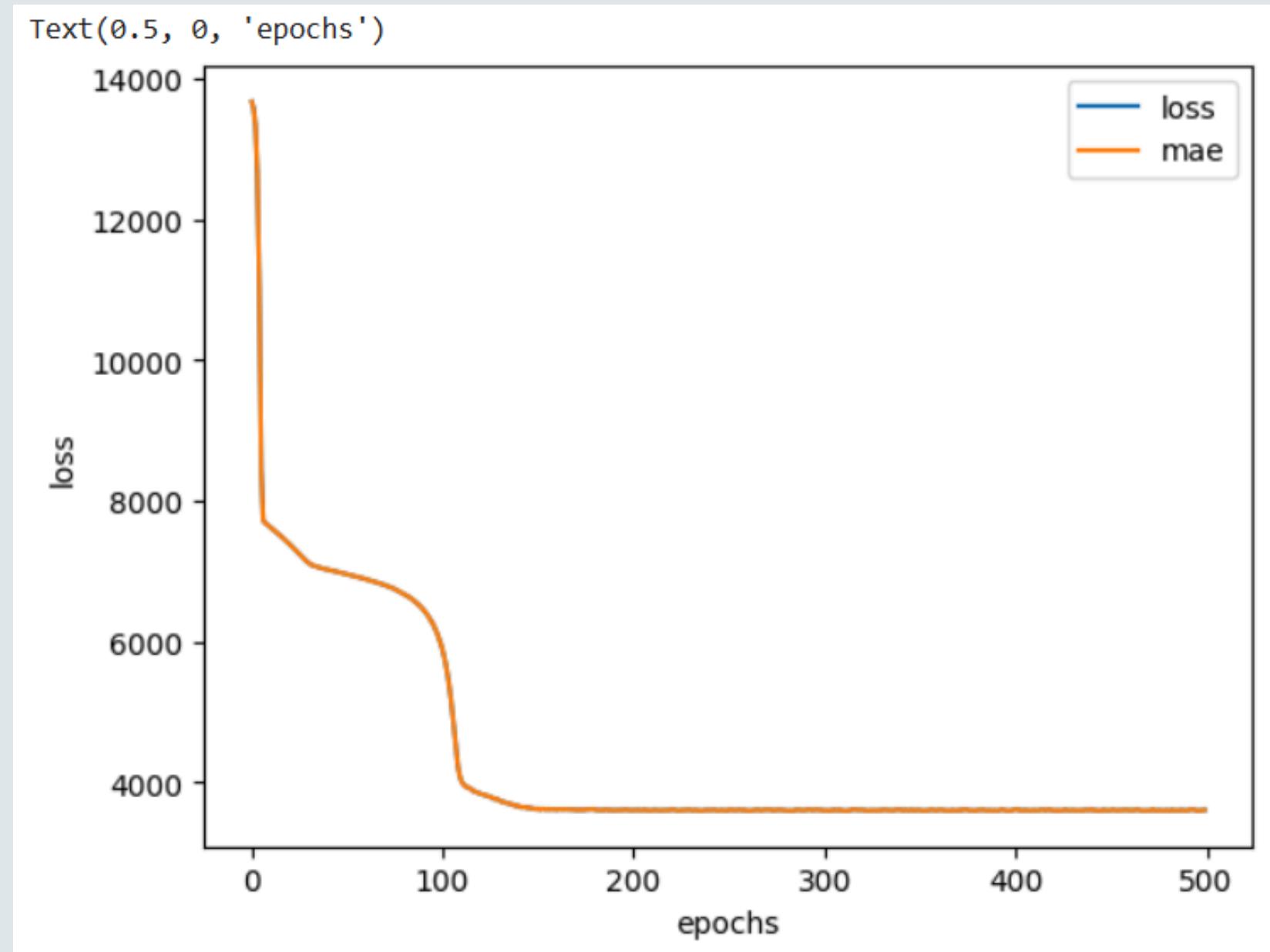
Neural Network



Neural Network

Train-Test	Architecture	Optimizer	Epochs	MAE
60-40	16-8-6-4-2-1	Adam	300	4131.75
60-40	16-8-6-4-2-1	Adam	600	2818.14
70-30	16-8-6-4-2-1	Adam	300	3596.79
70-30	16-8-6-4-2-1	Adam	250	2714.46
75-25	16-8-6-4-2-1	Adam	300	3516.62
75-25	16-8-6-4-2-1	Adam	500	2678.48
80-20	16-8-6-4-2-1	Adam	300	3483.99
80-20	16-8-6-4-2-1	Adam	400	2702.36

Neural Network plot for Observation



Train test split	75-25
Architecture	16-8-6-4-2-1
Optimizer	Adam
Epochs	500

SUMMARY

The research aims to identify factors increasing insurance costs to aid in pricing and risk assessment.

In Model 1, Random Forest outperformed with an r^2 score of 0.829 and MAE of 2631.44. Model 2, using Linear Regression, had an r^2 score of 0.811 and MAE of 3954.26.

Thus, Random Forest is the best model for forecasting medical insurance premiums, utilizing an 80:20 train split ratio.

KEY INSIGHTS

- Health Factors
- Unhealthy lifestyle
- Regional Variation
- Cost for family coverage
- Healthier behaviors

FUTURE SCOPE

- custom Premiums
- Improved Accuracy
- Adaptive Pricing
- Transparent Models

WORK DISTRIBUTION



Team member 1
Rishi Bharadwaj

Collect information about
medical insurance premium
& Literature Review

Team member 2
Ajay kumar

Data Preprocessing

Team member 3
Jai kumar

Exploratory Data Analysis

Team member 4
N.Aishwarya

Implement Machine
Learning Algorithms



Colab Notebook

Thank You

N.AISHWARYA
RISHI BHARADWAJ

AJAY KUMAR

JAI KUMAR



APPENDIX



Loading the Dataset

```
▶ data= pd.read_csv('Insurance.csv')
data
```

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

1338 rows × 7 columns

Null Values

```
[ ] data.isna().sum()
```

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

dtype: int64

Checking for the data type

```
▶ data.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 unique Values

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

```
→ [19 18 28 33 32 31 46 37 60 25 62 23 56 27 52 30 34 59 63 55 22 26 35 24  
  41 38 36 21 48 40 58 53 43 64 20 61 44 57 29 45 54 49 47 51 42 50 39]  
age  
18      69  
19      68  
50      29  
51      29  
47      29  
46      29  
45      29  
20      29  
48      29  
52      29  
22      28  
49      28  
54      28  
53      28  
21      28  
26      28  
24      28  
25      28  
28      28  
27      28  
23      28  
43      27  
29      27
```

```
▶ data.describe()
```

	age	bmi	children	charges
count	1337.000000	1337.000000	1337.000000	1337.000000
mean	39.222139	30.663452	1.095737	13279.121487
std	14.044333	6.100468	1.205571	12110.359656
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.290000	0.000000	4746.344000
50%	39.000000	30.400000	1.000000	9386.161300
75%	51.000000	34.700000	2.000000	16657.717450
max	64.000000	53.130000	5.000000	63770.428010

Replacing Duplicate Rows:

```
[147] num_duplicates = data.duplicated().sum()  
      print(f"Number of duplicate rows: {num_duplicates}")
```

→ Number of duplicate rows: 1

```
[148] data[data.duplicated()]
```

→

	age	sex	bmi	children	smoker	region	charges
581	19	male	30.59	0	no	northwest	1639.5631

▶ data_cleaned=data.drop_duplicates()

```
[150] data.drop_duplicates(inplace=True)
```

```
[151] num_duplicates = data.duplicated().sum()  
      print(f"Number of duplicate rows: {num_duplicates}")
```

→ Number of duplicate rows: 0

Dividing the data set

```
X=data.drop(['charges'],axis=1)
print(X)
y = data['charges']
print(y)

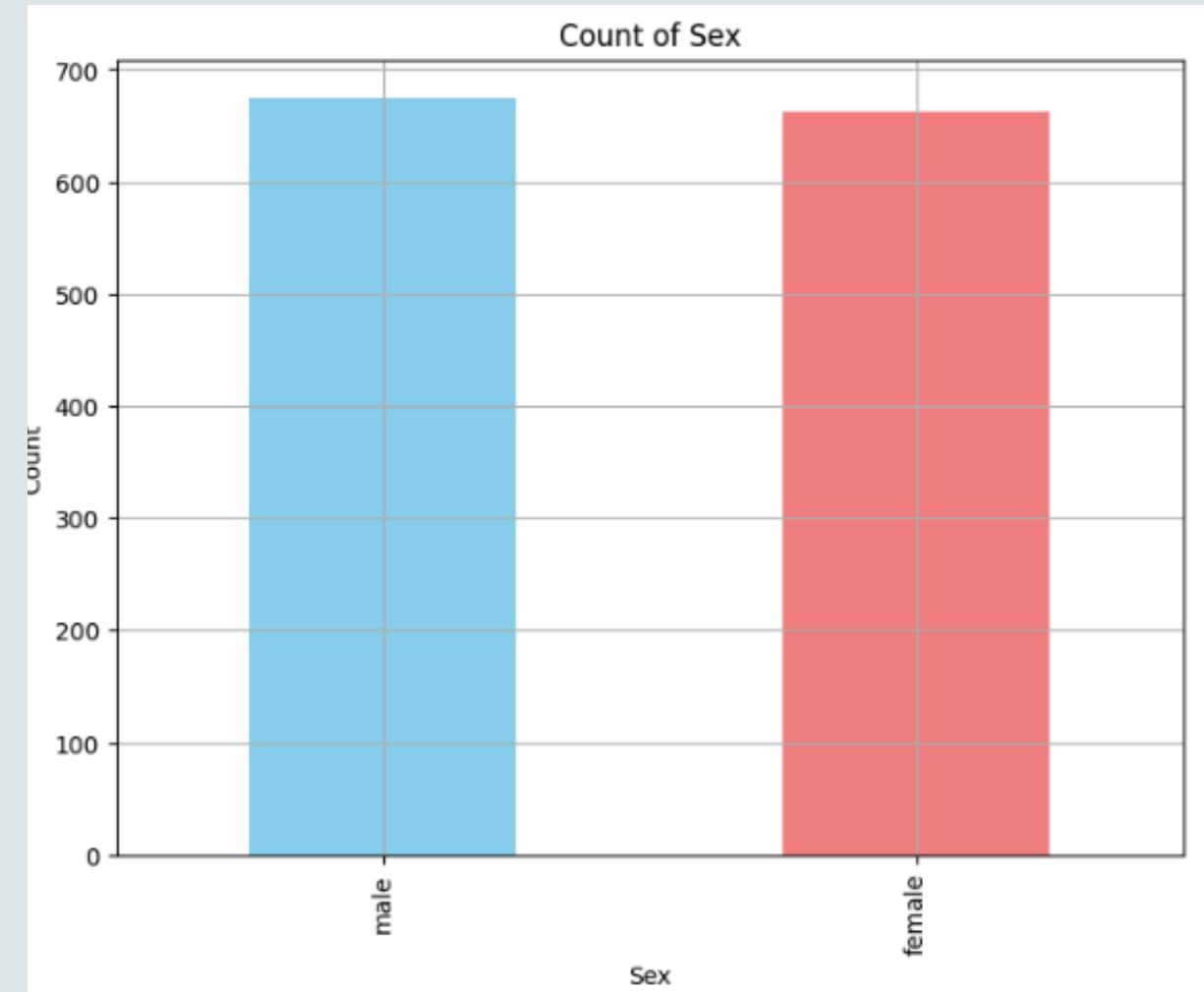
[  age   sex   bmi  children smoker  region
0    19  female  27.900      0    yes southwest
1    18    male  33.770      1     no southeast
2    28    male  33.000      3     no southeast
3    33    male  22.705      0     no northwest
4    32    male  28.880      0     no northwest
...
1333   50    male  30.970      3     no northwest
1334   18  female  31.920      0     no northeast
1335   18  female  36.850      0     no southeast
1336   21  female  25.800      0     no southwest
1337   61  female  29.070      0    yes northwest
[1337 rows x 6 columns]
0      16884.92400
1      1725.55230
2      4449.46200
3      21984.47061
4      3866.85520
...
1333   10600.54830
1334   2205.98080
1335   1629.83350
1336   2007.94500
1337   29141.36030
Name: charges, Length: 1337, dtype: float64
```

```
X=pd.get_dummies(X, dtype='int', drop_first=True)
print(X)

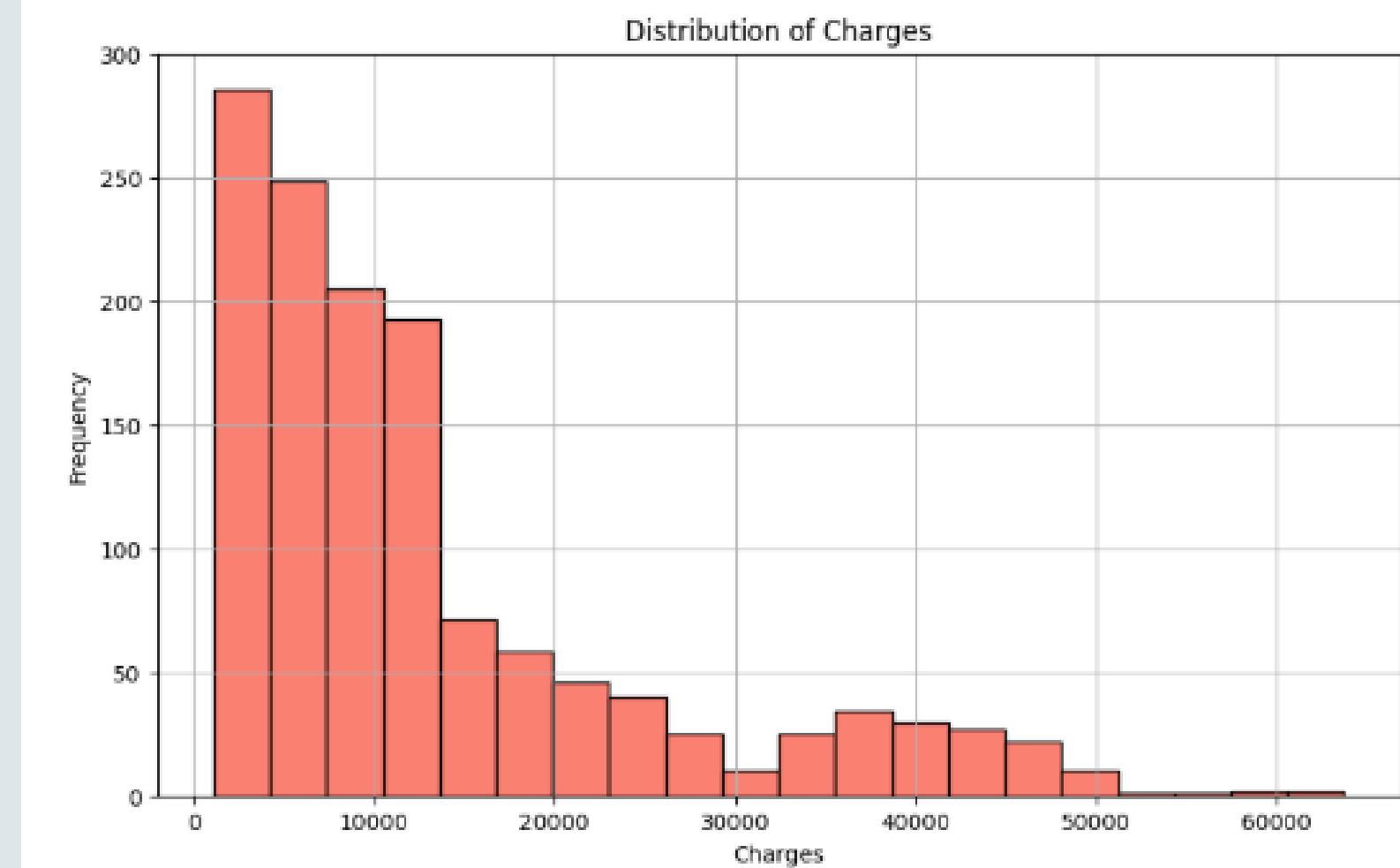
[  age   bmi  children  sex_male  smoker_yes  region_northwest \
0    19  27.900      0        0         1            0
1    18  33.770      1        1         0            0
2    28  33.000      3        1         0            0
3    33  22.705      0        1         0            1
4    32  28.880      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
3                  0            0
4                  0            0
...
1333                0            0
1334                0            0
1335                1            0
1336                0            1
1337                0            0
[1337 rows x 8 columns]
```

EDA

```
# Bar Plot for Sex
plt.figure(figsize=(8, 6))
data['sex'].value_counts().plot(kind='bar', color=['skyblue', 'lightcoral'])
plt.title('Count of Sex')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.grid(True)
plt.show()
```



```
# Histogram for Charges
plt.figure(figsize=(10, 6))
plt.hist(data['charges'], bins=20, color='salmon', edgecolor='black')
plt.title('Distribution of Charges')
plt.xlabel('Charges')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



Linear Regression

linear regression

```
[ ] import pandas as pd
import seaborn as sns
import statsmodels.formula.api as smf
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
import numpy as np

# allow plots to appear directly in the notebook
%matplotlib inline
```

```
[ ] 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=1)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.25, train_size=0.75,random_state=1)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.30, train_size=0.70,random_state=1)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.40, train_size=0.60,random_state=1)

[ ] #X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.20, train_size=0.80)
lm2 = LinearRegression()
lm2.fit(X_train1, y_train1)
y_pred1 = lm2.predict(X_test1)
print(np.sqrt(metrics.mean_squared_error(y_test1, y_pred1)))
print((metrics.r2_score(y_test1, y_pred1)))
print((metrics.mean_squared_error(y_test1, y_pred1)))
print((metrics.mean_absolute_error(y_test1,y_pred1)))

5689.30829118911
0.7497814388540698
32366228.832193147
3924.6360380217848

[ ] #X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.25, train_size=0.75)
lm2 = LinearRegression()
lm2.fit(X_train2, y_train2)
y_pred2 = lm2.predict(X_test2)
print(np.sqrt(metrics.mean_squared_error(y_test2, y_pred2)))
print((metrics.r2_score(y_test2, y_pred2)))
print((metrics.mean_squared_error(y_test2, y_pred2)))
print((metrics.mean_absolute_error(y_test2,y_pred2)))

5992.503495873213
0.7200784016998649
35910098.148052685
4151.7215200297685

[ ] #X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.30, train_size=0.70)
lm2 = LinearRegression()
lm2.fit(X_train3, y_train3)
y_pred3 = lm2.predict(X_test3)
print(np.sqrt(metrics.mean_squared_error(y_test3, y_pred3)))
print((metrics.r2_score(y_test3, y_pred3)))
print((metrics.mean_squared_error(y_test3, y_pred3)))
print((metrics.mean_absolute_error(y_test3,y_pred3)))

6043.804783305154
0.733326836658079
36527576.25870226
4207.799065841042

[ ] #X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.40, train_size=0.60)
lm2 = LinearRegression()
lm2.fit(X_train4, y_train4)
y_pred4 = lm2.predict(X_test4)
print(np.sqrt(metrics.mean_squared_error(y_test4, y_pred4)))
print((metrics.r2_score(y_test4, y_pred4)))
print((metrics.mean_squared_error(y_test4, y_pred4)))
print((metrics.mean_absolute_error(y_test4,y_pred4)))

6119.7164422330848
0.7196282704835242
37450929.333337516
4274.286033572359
```

KNN

```
KNN
[ ] from sklearn.neighbors import KNeighborsRegressor
80-20
[ ] model=KNeighborsRegressor(n_neighbors=5)
model.fit(X_train1,y_train1)

+ KNeighborsRegressor ⓘ ⓘ
KNeighborsRegressor()

y_pred1 = model.predict(X_test1)
print(y_pred1)

[16828.517086 18192.71901 10971.66693 10336.797548 18179.196618
 7182.362988 9037.28354 21986.61754 14312.35928 21926.88825
 18610.132964 6827.35397 15732.260012 8312.86098 1372.48868
 16637.80268 7331.625542 21151.78429 19167.73447 25937.297052
 4004.50979 29370.532788 18494.878844 14322.47749 7459.11708
 22671.025772 15867.571668 19689.98481 4948.15816 3823.70073
 24846.90293 3985.0733 10633.271286 25803.835716 12007.3807
 1561.134 31942.651644 19261.559494 14222.811394 15832.178256
 14248.17668 8291.637436 10355.2936 17902.479582 36689.978988
 12689.87445 32336.702692 7685.46873 11912.66588 11544.03559
 22747.741916 11633.36595 19818.911982 13898.827284 14188.608892
 6936.701148 7810.67289 11962.15698 11237.86892 33916.08512
 13534.861874 21308.474918 11393.717482 10578.2225 6427.785372
 7101.239332 2156.71866 21653.67231 13513.176664 15605.871344
 26177.87359 5929.00123 4934.41103 4167.00803 13000.478922
 4650.76186 9997.422312 6877.81562 6581.23893 18299.347248
 18274.134178 16558.12366 19623.184282 16619.632168 25804.07994
 6710.0803738 2775.06353 3171.43892 18822.322678 16387.936366
 5755.936754 12949.74305 16000.68619 15309.15557 10937.09363
 14278.12161 9766.20053 14581.99648 5042.35943 18495.71356
 18186.21812 5752.73892 6975.98214 14646.60893 4218.8003
 18710.477126 16950.269882 24713.328932 10106.7509 1687.57275
 14745.62035 6611.43847 12034.234918 8934.03378 7182.362988
 16824.57583 16756.68211 14950.1757 3952.04786 9459.23189
```

```
28327.81532 12491.291998 25124.045864 15908.8316 18714.24287
20068.935304 14371.02625 11995.21429 10833.60754 13390.4994
3259.32124 12211.85262 12598.010172 10847.44178 15548.588846
7372.148334 20862.72311 4087.79068 24384.27708 10336.034596
5038.45901 14387.52593 10013.832312 10834.69038 7620.746626
16744.98065 8217.60013 14390.92143 8935.52044 17659.44979
21079.255988 13270.34132 1793.87728 2872.04859 25526.389972
4441.25629 10672.46854 7379.43796 9968.37845 16522.75156
22525.846884 1468.52017 1719.60885 14407.842862 12908.857306
13254.19857 8469.99118 2204.13484 6854.73571 24152.910392
8199.23485 14537.41461 16690.005556 ]
```

```
[ ] df=pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
print(df)

+ Predicted      Actual
629  16828.517086  42983.45850
1087 18192.719018  11353.22760
283  10971.666930  11879.10405
798  10336.797548  5662.22500
594  18179.196618  5709.16440
...
1164  6854.735710  7153.55390
962  24152.910392  14474.67500
1158  8199.234850  2459.72010
1241  14537.414610  49577.66240
216  16690.005556  10355.64100
[268 rows x 2 columns]
```

```
[ ] from sklearn.metrics import r2_score
r2_score(y_test1,y_pred1)

+ 0.16403910584448667
```

```
[ ] from sklearn import metrics
metrics.mean_absolute_error(y_test1,y_pred1)

+ 7452.4967916171645
```

SVM

```
Support Vector Machine

[ ] from sklearn.svm import SVR
80-20

[ ] model = SVR(kernel='rbf')
model.fit(X_train1, y_train1)
SVR()
y_pred1 = model.predict(X_test1)
print(y_pred1)

[9557.36283478 9623.37438268 9614.44584998 9538.91191586 9548.96851876
 9457.47775986 9449.81471916 9618.52989882 9617.95461982 9594.99472821
 9631.55493168 9516.48953317 9602.87189233 9556.98183299 9441.88932283
 9642.67591167 9498.8915581 9485.2868371 9624.38429591 9646.98682435
 9487.23023278 9637.26239626 9563.85196072 9548.76358184 9474.4928276
 9551.39095643 9546.42669573 9632.93485383 9496.23319828 9476.96833165
 9585.23068459 9481.64291586 9468.55776567 9589.57118653 9439.25831433
 9445.95335334 9481.86715618 9635.78961276 9511.48856751 9536.88137726
 9598.77461464 9443.92348724 9586.37287922 9582.55788875 9637.67888823
 9634.58492885 9595.87951433 9585.89551774 9584.81594366 9618.41263365
 9639.72678381 9613.28113717 9613.35386657 9581.47714783 9575.82243821
 9454.22477419 9458.77474553 9628.31783696 9496.41688895 9658.26398852
 9685.42552873 9619.88769811 9445.9284455 9468.68277389 9465.43394385
 9470.7373897 9449.13123413 9567.45438515 9598.66748635 9632.57322959
 9515.79841334 9447.51130949 9440.54868664 9442.45437821 9584.61907926

9482.90938768 9448.97865694 9481.50896364 9472.53228043 9442.6841774
9452.58321084 9587.77948979 9621.8676755 9577.39541026 9615.21398906
9648.3134292 9634.58921891 9442.93959338 9453.58404682 9605.5826111
9481.66488484 9614.56525966 9487.17030438 9522.20191096 9455.61261299
9559.51566023 9448.62738292 9442.79949747 9609.51664705 9461.29176286
9496.23594536 9566.46531159 9449.515595 9535.24571152 9647.2518449
9443.21697585 9658.17052858 9601.53243202]

[ ] df = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
print(df)

   Predicted      Actual
629  9557.362835  42983.45858
1087 9623.374383  11353.22768
283   9614.445858  11879.18405
790   9538.911915  5662.22500
594   9548.968511  5709.16440
...
1164  9535.245712  7153.55398
962   9647.251845  14474.67500
1158  9443.216976  2459.72018
1241  9650.178529  49577.66248
216   9601.532432  10355.64100

[268 rows x 2 columns]

[ ] from sklearn.metrics import r2_score
r2_score(y_test1,y_pred1)

   -0.037880557580482144

[ ] from sklearn import metrics
metrics.mean_absolute_error(y_test1,y_pred1)

   7409.859428577142
```

Decision tree regressor

```
DECISION TREE

[ ] import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics

80-20

[ ] clf = DecisionTreeRegressor()
clf1 = clf.fit(X_train1,y_train1)
print(clf1)

DecisionTreeRegressor()

y_pred1 = clf.predict(X_test1)
print(y_pred1)

[43896.3763 11743.9341 18825.2537 5649.715 5699.8375 3213.62285
 1532.4697 25517.11363 12222.8983 24667.419 13284.28565 11884.84858
 9863.4718 9715.841 1137.011 13457.9608 4508.33925 37607.5277
 33471.97189 15612.19335 4137.5227 13470.86 17929.38337 7726.854
 4922.9159 48885.13561 8410.84685 48517.56315 5002.7827 2483.736
 9549.5651 3736.4647 4949.7587 30284.64294 1621.8827 2438.0552
 36197.699 13451.122 5124.1887 14478.33815 24667.419 2103.08
 18782.6424 40720.55105 13470.86 12730.9996 10848.1343 5266.3656
 9264.797 11345.519 13143.33665 12896.6512 9858.432 24915.84626
 21232.18226 2395.17155 2850.68375 11842.62375 5926.846 14394.5579
 9724.53 24227.33724 16586.49771 5289.57885 3956.07145 4949.7587
 1888.487 41661.682 18197.7722 28923.13692 39047.285 4905.4225
 4744.466 2102.08 8070.1051 4700.00525 5375.078 2404.022]
```

3277.161	26467.09737	63770.42881	5289.57885	26926.5144	7729.64575
12269.68865	4805.4225	13485.3983	3498.5491	5246.047	8827.2099
16884.924	33732.6867	26818.95052	39556.4945	1628.4789	9866.38485
6986.697	5327.48825	1989.52745	4934.785	48984.1995	1728.897
2020.5523	4402.233	27117.99378	10843.249	44268.7499	13143.33665
12957.118	1744.465	2352.96845	10461.9794	4237.12655	10226.2842
4074.4537	7986.47525	2020.5523	39556.4945	1621.8827	1628.4789
10825.2537	2473.3341	6933.24225	8823.98575	2775.19215	6770.1925
31620.00106	1877.9294	47496.49445	9863.4718		

```
[ ] print("R-squared:", metrics.r2_score(y_test1, y_pred1))

R-squared: 0.6346858489893212

[ ] print("Mean Squared Error:", metrics.mean_squared_error(y_test1, y_pred1))

Mean Squared Error: 47256973.988655426

[ ] print("Mean Absolute Error:", metrics.mean_absolute_error(y_test1, y_pred1))

Mean Absolute Error: 3225.587812649254
```

Random Forest Regressor

```
RANDOM FOREST

[ ] from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import plot_tree
from matplotlib import pyplot as plt

80-20

[ ] print(X_train1.shape)
print(y_train1.shape)
print(X_test1.shape)
print(y_test1.shape)

→ (1069, 8)
(1069,)
(268, 8)
(268,)

▶ rf = RandomForestRegressor()
rf.fit(X_train1, y_train1)

→ + RandomForestRegressor ⓘ ⓘ
RandomForestRegressor()

[ ] y_pred1 = rf.predict(X_test1)
print(y_pred1)

→ [47609.8972823 12428.1444087 13770.3127339 8939.8116013
 6667.336726 6410.4123499 2653.1529187 17064.4832888
11938.403051 24890.5404887 14786.6453514 8807.2285616
11882.7562533 9542.7891737 1140.607015 13958.4238374
5342.0447745 41917.8111163 16746.9889348 20228.0005332
4728.3231208 15865.9318877 13910.465246 8839.4475125
7976.5098369 44552.2249421 8991.2383276 48085.1799809
5046.576947 3660.952798 18114.2629025 4202.6061955
...]
```

1644.8299815	10512.6023285	10184.6392947	7678.8495179
2297.44262889	5376.8367587	48357.7273325	1748.7698865
3566.5597453	5645.8880857	13990.4913183	11596.8757597
45702.544574	13214.8747475	13220.489528	2789.48921752
3128.6422892	17988.2060757	4389.5144795	11187.6256922
6548.8763181	8896.7620208	3464.5083562	39989.5622315
1845.5665335	1609.4367825	14131.6349277	4762.7182132
11883.9211921	8780.3784901	2721.1230978	6944.463928
24783.4965868	3129.16573339	47631.7350585	13050.3555403]

```
[ ] print("R-squared:", metrics.r2_score(y_test1, y_pred1))

→ R-squared: 0.8265979664603367

[ ] print("Mean Squared Error:", metrics.mean_squared_error(y_test1, y_pred1))

→ Mean Squared Error: 22431256.401902393

▶ print("Mean Absolute Error:", metrics.mean_absolute_error(y_test1, y_pred1))

→ Mean Absolute Error: 2696.095002784179
```

XG Boost Regressor

```
XGBOOST
[ ] import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
80-20

[ ] model = xgb.XGBRegressor()
model.fit(X_train1, y_train1)

[ ] y_pred1 = model.predict(X_test1)
print(y_pred1)

[+] [53555.18 11461.423 12328.625 11621.41 4761.934 7750.494
 1529.4352 28414.572 11197.441 25815.15 14984.92 8241.342
 18975.551 12885.234 783.871 21445.992 5929.791 47263.584
 14852.5 25863.543 4428.945 17889.02 9728.698 7515.8193
 11295.879 46785.555 9298.674 49988.33 3783.8862 1293.941
 8658.964 2485.7837 3588.8274 12714.674 2851.182 1347.1912
 38437.418 14930.253 9465.392 9512.698 24132.133 1381.1296
 11352.383 38335.387 13748.171 12944.448 6926.9644 4855.4585
 9496.143 18398.47 14612.465 13194.611 13377.695 16814.891
 13593.789 3873.0288 2928.6475 10825.764 4624.3684 19449.688]
```

17881.783	34284.746	12668.259	43301.18	1138.8846	9087.974
14269.522	7715.562	2286.9023	7161.562	42486.285	1698.3274
2005.7002	9996.726	14614.38	9520.275	44468.668	18835.94
11878.636	2301.4807	3078.2236	27661.418	3302.828	10500.338
5086.717	9126.196	3165.3435	42321.848	1258.3187	855.5152
16849.87	2840.1963	9703.393	7843.788	1988.3663	7653.7427
21099.135	3758.2148	46147.787	11948.217		

```
[ ] mse = mean_squared_error(y_test1, y_pred1)
r2 = r2_score(y_test1, y_pred1)
mae = mean_absolute_error(y_test1, y_pred1)

[ ] print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
print(f"Mean Absolute Error: {mae}")

[+] Mean Squared Error: 26612354.118278934
R-squared: 0.7942765114303354
Mean Absolute Error: 3030.090976026156
```

ADA Boost Regressor

```
ADABOOST

[ ] import pandas as pd
import seaborn as sns
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostRegressor

80-20

[ ] model = AdaBoostRegressor(n_estimators=50, random_state=42)
model.fit(X_train1, y_train1)

+ AdaBoostRegressor ① ②
AdaBoostRegressor(random_state=42)

y_pred1 = model.predict(X_test1)
print(y_pred1)

[46699.55313864 14724.09342165 15686.0603305 6992.14362925
 6992.14362925 5484.47224727 5484.47224727 17458.23664369
 14724.09342165 26299.07696318 14724.09342165 9117.35326937
 14724.09342165 12860.44120335 5484.47224727 14724.09342165
 6992.14362925 39836.89793294 17458.23664369 17458.23664369
 6992.14362925 14724.09342165 8736.76553005 9117.35326937

46699.55313864 14724.09342165 14724.09342165 5484.47224727
5484.47224727 15686.0603305 8736.76553005 14724.09342165
8736.76553005 9117.35326937 5484.47224727 46699.55313864
5484.47224727 5484.47224727 15686.0603305 5484.47224727
9117.35326937 8736.76553005 8736.76553005 8736.76553005
17458.23664369 5484.47224727 46699.55313864 14724.09342165]

r2 = r2_score(y_test1, y_pred1)
print(f"R-squared: {r2:.2f}")

mse = mean_squared_error(y_test1, y_pred1)
print(f"Mean Squared Error: {mse:.2f}")

mae=mean_absolute_error(y_test1, y_pred1)
print(f"Mean Absolute Error: {mae:.2f}")

R-squared: 0.82
Mean Squared Error: 23615256.63
Mean Absolute Error: 3928.00
```

Neural Networks :

```
▶ tf.random.set_seed(42)

# STEP1: Creating the model

model_nom2= tf.keras.Sequential([
    tf.keras.layers.Dense(16),
    tf.keras.layers.Dense(8),
    tf.keras.layers.Dense(6),
    tf.keras.layers.Dense(4),
    tf.keras.layers.Dense(2),
    tf.keras.layers.Dense(1)
])

# STEP2: Compiling the model

model_nom2.compile(loss= tf.keras.losses.mae,
                    optimizer= tf.keras.optimizers.Adam(), #SGD
                    metrics= ["mae"])

# STEP3: Fit the model

history= model_nom2.fit(X_train2, y_train2, epochs= 500, verbose=1)
```

Epoch 1/500
32/32 6s 4ms/step - loss: 13685.7686 - mae: 13685.7686
Epoch 2/500
32/32 0s 3ms/step - loss: 13600.0674 - mae: 13600.0674
Epoch 3/500
32/32 0s 3ms/step - loss: 13409.9844 - mae: 13409.9844
Epoch 4/500
32/32 0s 3ms/step - loss: 12883.7832 - mae: 12883.7832
Epoch 5/500
32/32 0s 4ms/step - loss: 11507.2715 - mae: 11507.2715
Epoch 6/500
32/32 0s 3ms/step - loss: 8981.1768 - mae: 8981.1768
Epoch 7/500
32/32 0s 3ms/step - loss: 7741.8447 - mae: 7741.8447
Epoch 8/500
32/32 0s 6ms/step - loss: 7708.9766 - mae: 7708.9766
Epoch 9/500
32/32 0s 3ms/step - loss: 7685.9336 - mae: 7685.9336
Epoch 10/500
32/32 0s 4ms/step - loss: 7668.1367 - mae: 7668.1367
Epoch 11/500
32/32 0s 3ms/step - loss: 7647.6372 - mae: 7647.6372
Epoch 12/500
32/32 0s 4ms/step - loss: 7627.4692 - mae: 7627.4692
Epoch 13/500

```
[ ] model_nom2.evaluate(X_test2, y_test2)
[ ] 11/11 0s 2ms/step - loss: 2678.4839 - mae: 2678.4839
[2932.37939453125, 2932.37939453125]

[ ] model_nom2.summary();

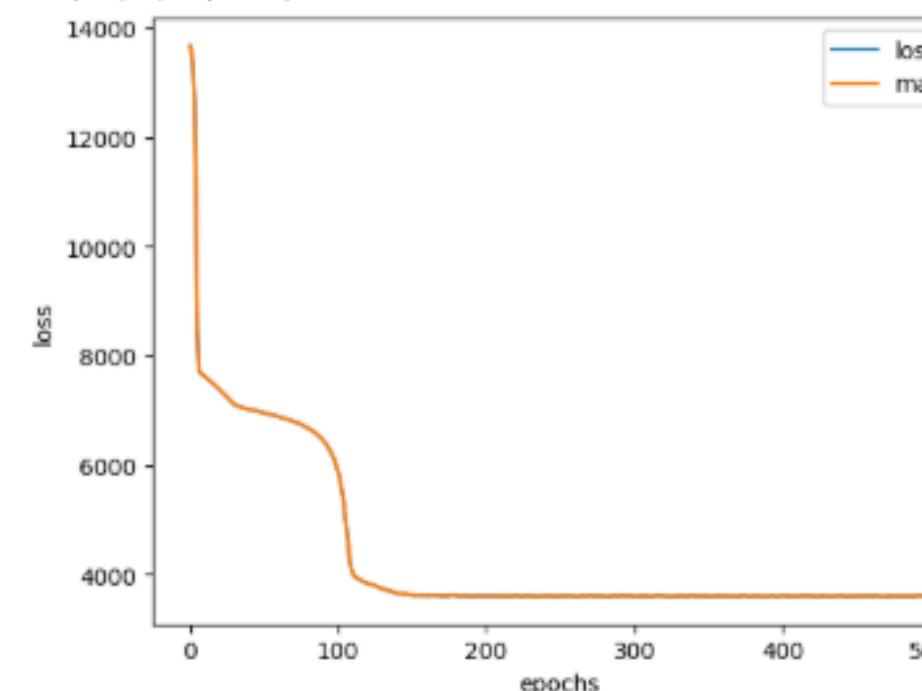
[ ] Model: "sequential_11"


| Layer (type)     | Output Shape | Param # |
|------------------|--------------|---------|
| dense_66 (Dense) | (None, 16)   | 144     |
| dense_67 (Dense) | (None, 8)    | 136     |
| dense_68 (Dense) | (None, 6)    | 54      |
| dense_69 (Dense) | (None, 4)    | 28      |
| dense_70 (Dense) | (None, 2)    | 10      |
| dense_71 (Dense) | (None, 1)    | 3       |


Total params: 1,127 (4.41 KB)
Trainable params: 375 (1.46 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 752 (2.94 KB)

▶ pd.DataFrame(history.history).plot()
plt.ylabel("loss")
plt.xlabel("epochs")

[ ] Text(0.5, 0, 'epochs')


```