



# MEDICAL INSURANCE PREMIUM

**Group 12 | Bhavan's Vivekananda  
College**

Y.Shan Koushik | Y.Sruthi | A.Rahul | V.Akash

## **Abstract**

- The project focuses on developing a model that accurately predicts medical insurance premiums, enabling better financial planning and efficient cost management for healthcare providers and insurers.
- The study explores machine learning algorithms such as Linear Regression, Decision Tree, Random Forest, K-Nearest Neighbours, Support Vector Machines, Bagging, and Boosting to predict insurance premiums based on demographic and health-related factors.

## **Objective**

- To identify the most suitable machine learning model for predicting medical insurance premiums to aid in cost-effective and personalized insurance policy pricing.

# CONTENT

- Introduction
- Literature Review
- Data Preprocessing
- Exploratory Data Analysis
- Data Modeling & Evaluation
- Summary
- Appendix



# Introduction

- **Topic Overview:** Medical insurance premiums are influenced by multiple factors, including age, BMI, smoking status, and geographic region.
- **Objective:** This project aims to use regression techniques to predict medical insurance premiums based on key factors.
- **Dataset:** Utilized a publicly available dataset containing variables such as age, sex, BMI, number of children, smoking status, and region to predict premium costs.
- **Significance:** Accurate premium predictions help insurers set fair prices, aid consumers in financial planning, and assist policymakers in understanding healthcare cost determinants.



# LITERATURE REVIEW

# Literature Review -1

1. K. Bhatia
2. S. S. Gill
3. N. Kamboj
4. M. Kumar
5. R. K. Bhatia

- This paper represents a machine learning-based health insurance prediction system. Recently, many attempts have been made to solve this problem, as after Covid-19 pandemic, health insurance has become one of the most prominent areas of research.
- We have used the USA's medical cost personal dataset from kaggle, having 1338 entries. Features in the dataset that are used for the prediction of insurance cost include: Age, Gender, BMI, Smoking Habit, number of children etc.
- We used linear regression and also determined the relation between price and these features. We trained the system using a 70-30

The background is a deep blue gradient. It features several glowing hexagonal shapes, some of which are interconnected to form a honeycomb-like pattern. Scattered throughout are plus signs (+) of varying sizes and brightness. In the lower right, there are sharp, intersecting lines that create a sense of depth and movement, resembling a stylized network or data flow.

# DATA PREPROCESSIN G

# DATA

**Dataset:** Our Dataset Consists **1338** rows and **7** columns

**Source** : <https://drive.google.com/file/d/1CZBa0RBm88cfdQzSyLVkn6n9Peuo3fsm/view?usp=drivesdk>

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



# DATA CLEANING

- Checked for null /NaN values : We replaced the Nan values with mean and mode of the column based on the attribute type.
- Garbage values : We replaced the garbage values of the column based on the domain knowledge of the dataset.
- Dummification: enabled and dummified the categorical variables

CATEGORICAL VARIABLES	CONTINUOUS VARIABLES
SEX	BMI
SMOKER	CHARGES
REGION	



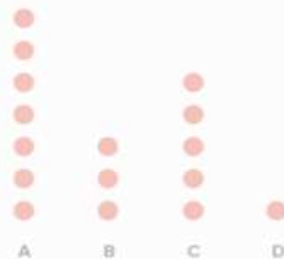
Multi-level Donut Chart



Angular Gauge



Dot Plot



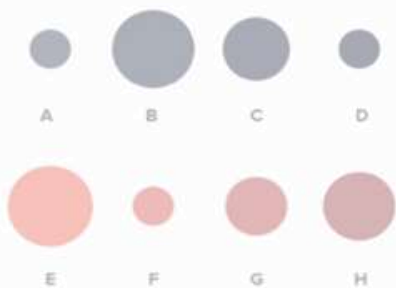
Pie Chart



Sociogram



Proportional Area Chart (Circle)



Waterfall Chart

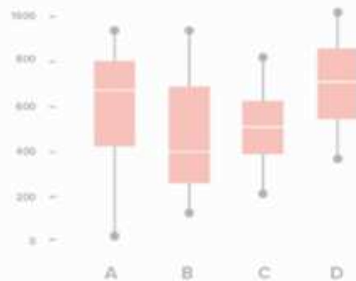


# EXPLORATORY DATA ANALYSIS

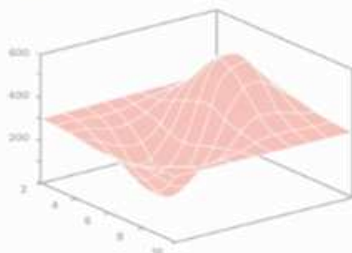
Population Pyramid



Boxplot



Three-dimensional Stream Graph



Semi Circle Donut Chart



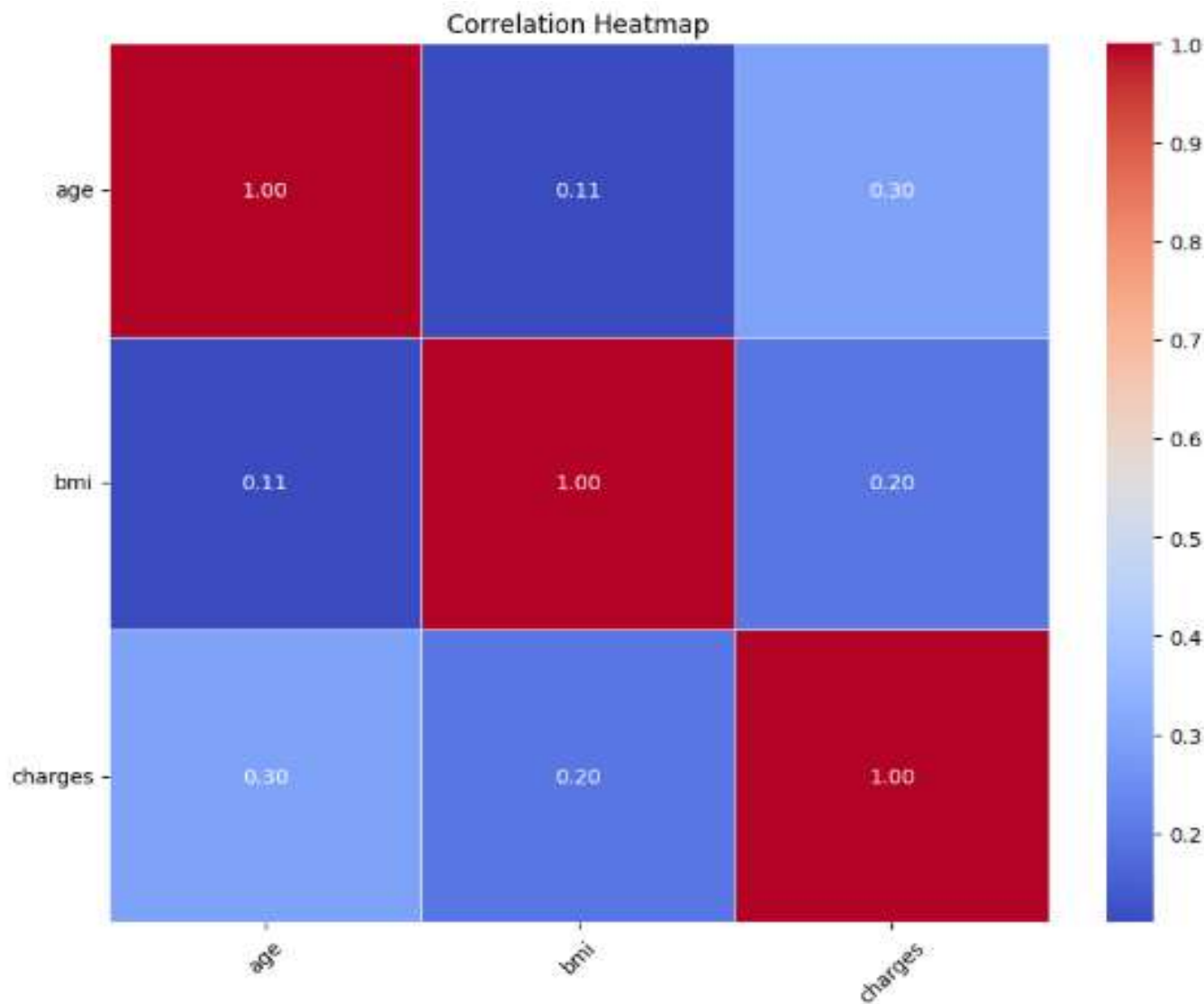
Topographic Map



Radar Diagram



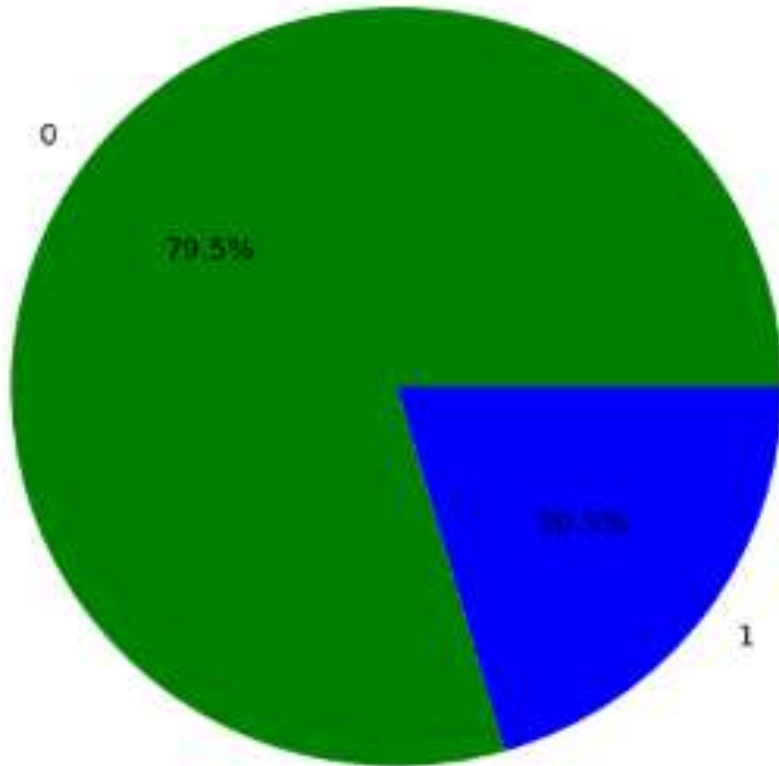
# CORRELATION MATRIX



- The Most Positively Correlated Variables Are Age and Charges
- The Next Most Positively Correlated Variables Are Bmi and Charges

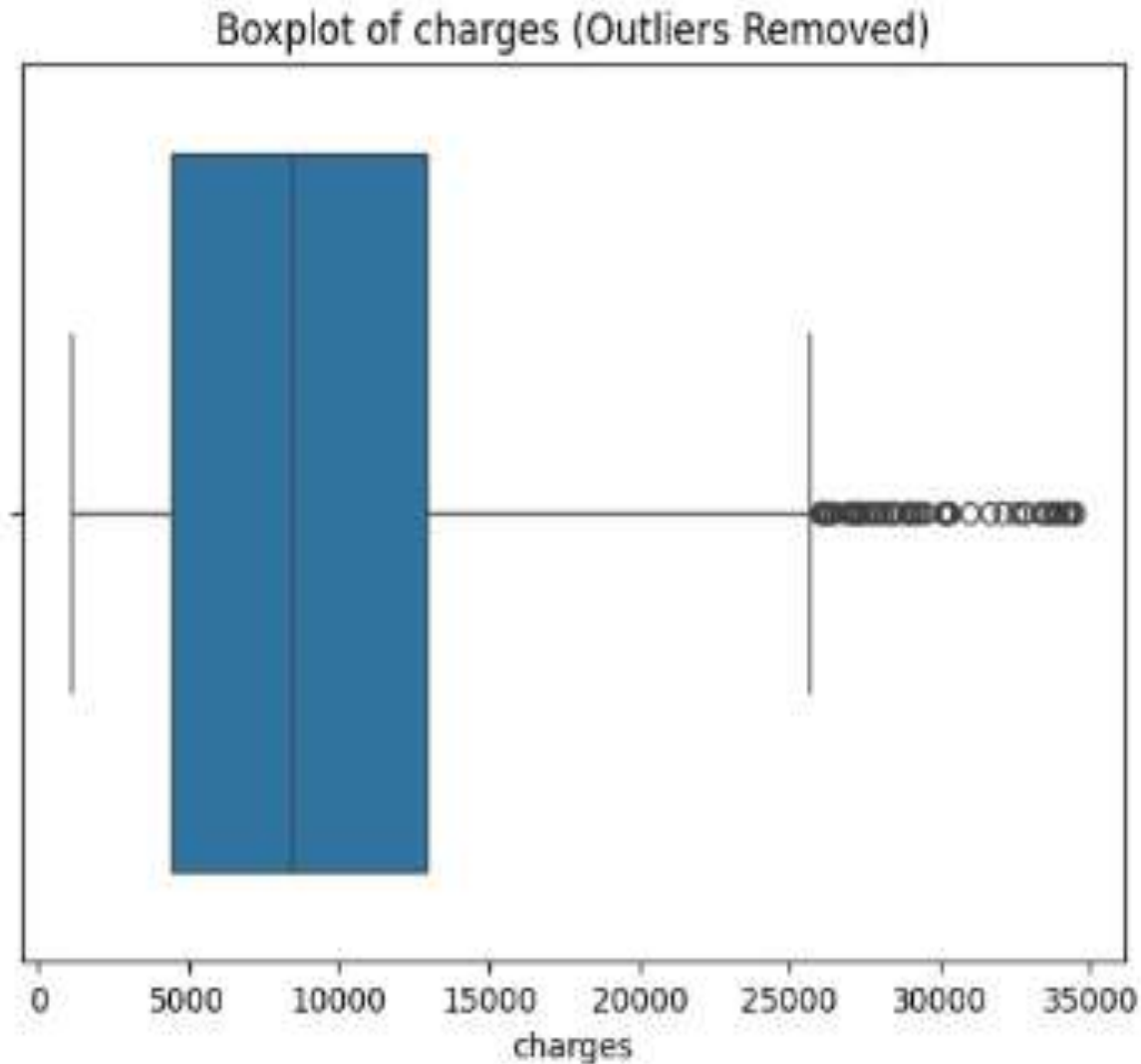
# PIE CHART

Proportion of Smokers vs. Non-Smokers



- THE PORTION OF SMOKERS IS LESS THAN NON SMOKERS
- SMOKERS-20.5%
- NON-SMOKERS-79.5%

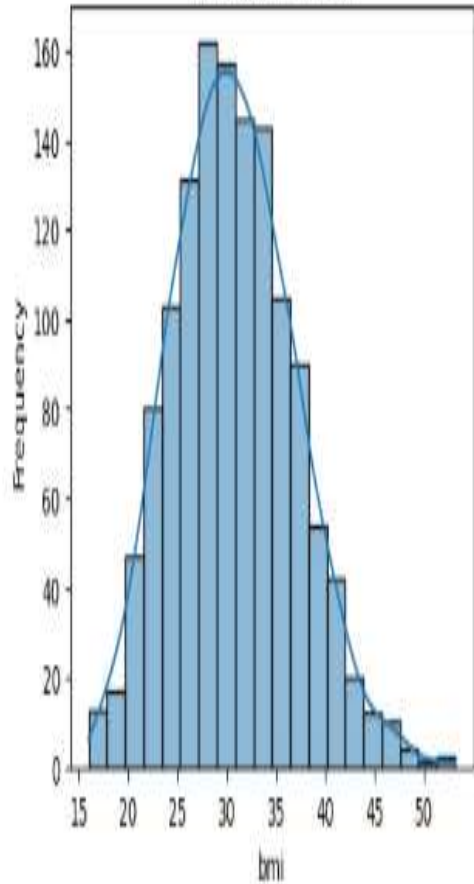
# BOX PLOT



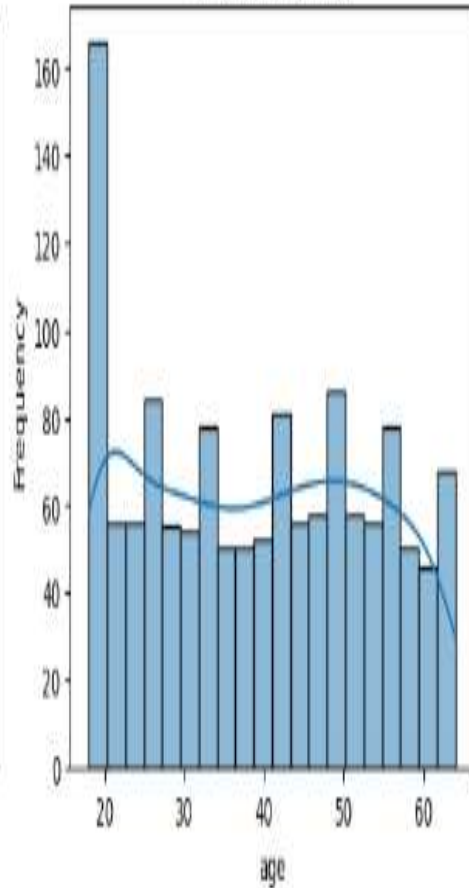
- The majority of data lies between approximately **0 and 25,000**.
- A significant number of outliers are present beyond the **25,000 mark**, with some values approaching **35,000**.
- The median (central line) is roughly in the middle of the box, suggesting a moderately symmetric distribution for non-outlier data.

# HISTOGRAM

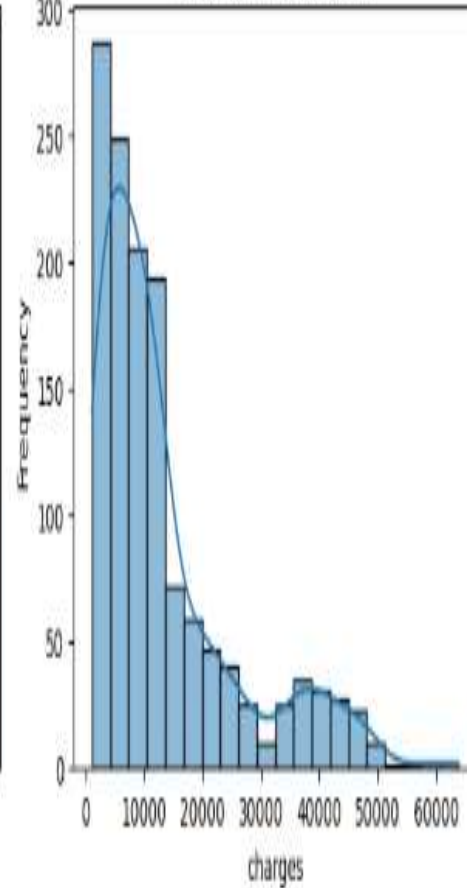
Distribution of bmi



Distribution of age



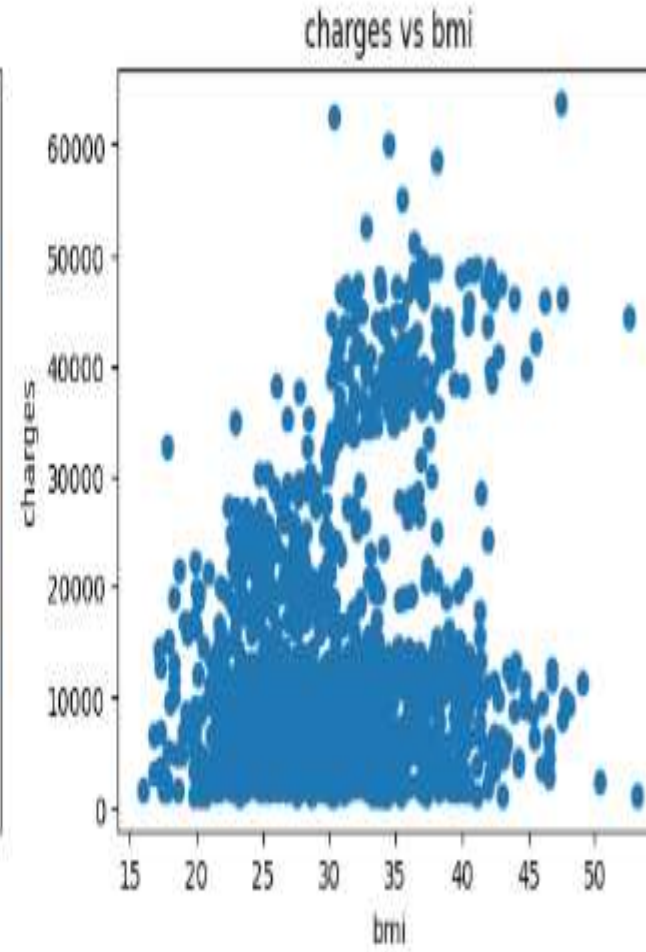
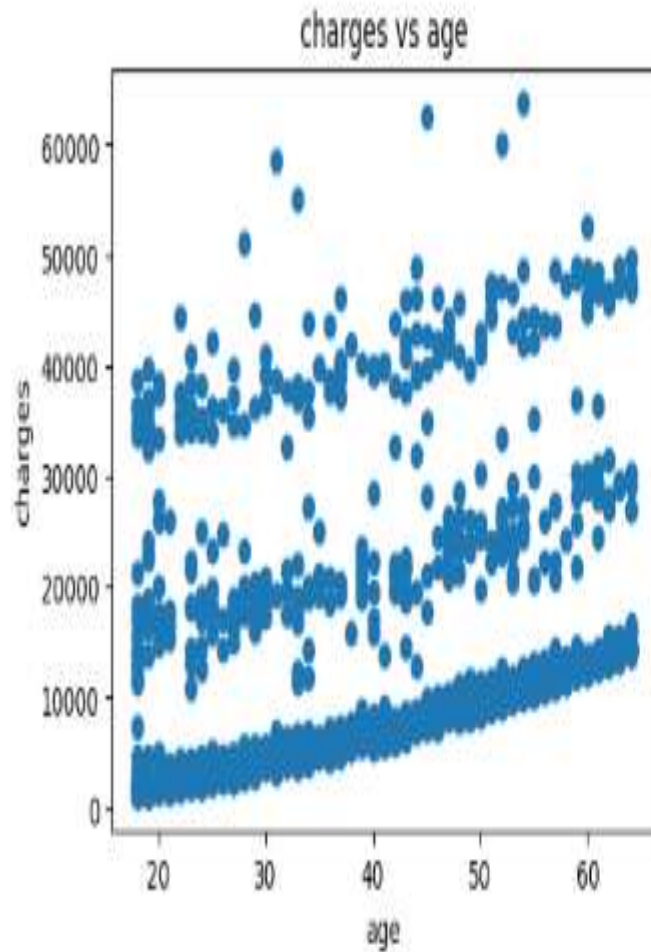
Distribution of charges



- **Distribution of BMI** (left plot):
  - The data is approximately **bell-shaped**, resembling a normal distribution.
  - The majority of BMI values range between **20 and 40**, peaking around **30–35**.
- **Distribution of Age** (middle plot):
  - The distribution is **relatively uniform**, with some peaks, especially at the lower ages (~20s).
  - Ages are spread across **20 to 65** without a dominant concentration.
- **Distribution of Charges** (right plot):
  - The data is **right-skewed** (positively skewed), with most charges concentrated under **15,000**.
  - A few higher values extend beyond **50,000**, suggesting the presence of outliers.



# SCATTER PLOT



- **Charges vs Age** (left plot):

- There is a **positive trend** where charges generally increase with age.

- Distinct clusters suggest groups with higher charges, possibly influenced by other factors like health conditions or insurance tiers.

- A lower band of points indicates a baseline level of charges that increases steadily with age.

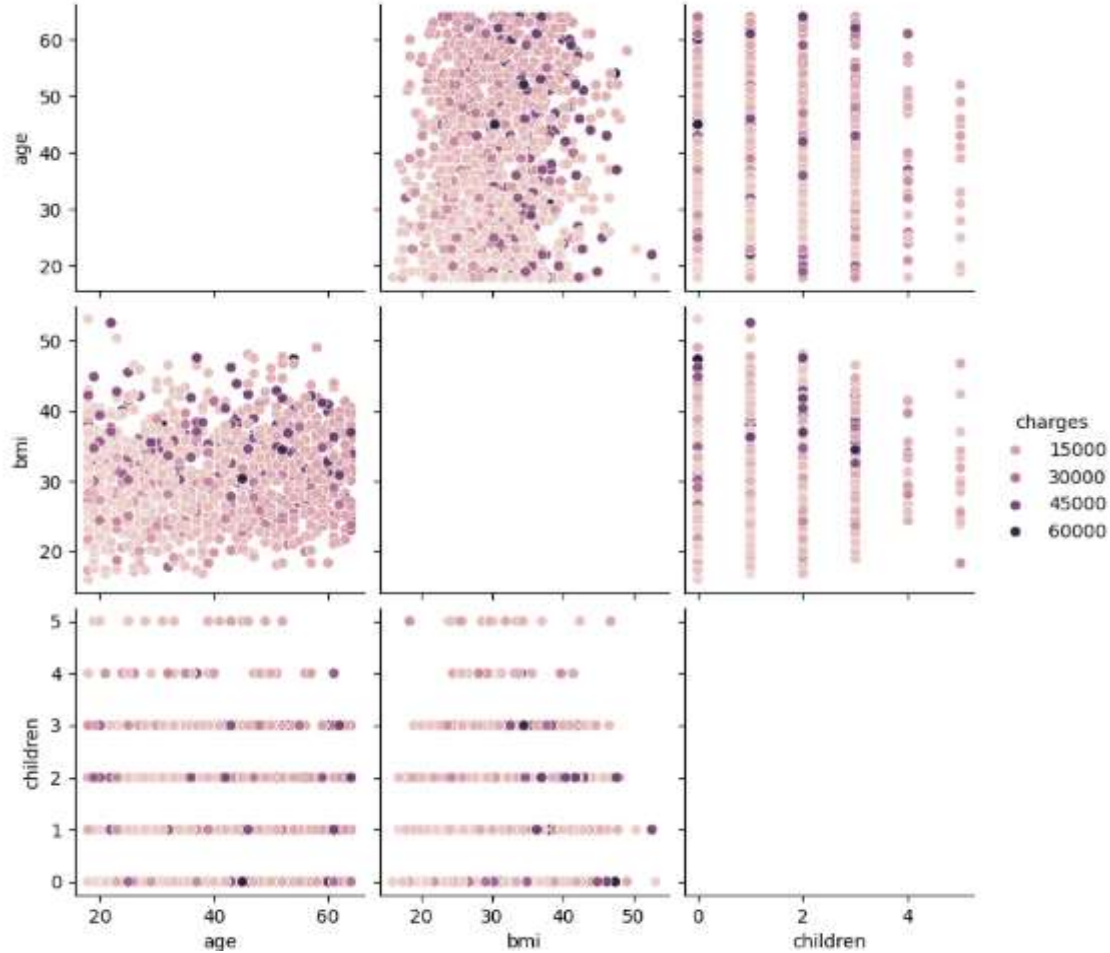
- **Charges vs BMI** (right plot):

- The relationship between charges and BMI shows **scattered data** without a strong linear trend.

- Higher charges appear concentrated for **BMI values between 30 and 40**, suggesting that elevated BMI may influence higher costs.

- Many individuals with low charges exist across the entire BMI range.

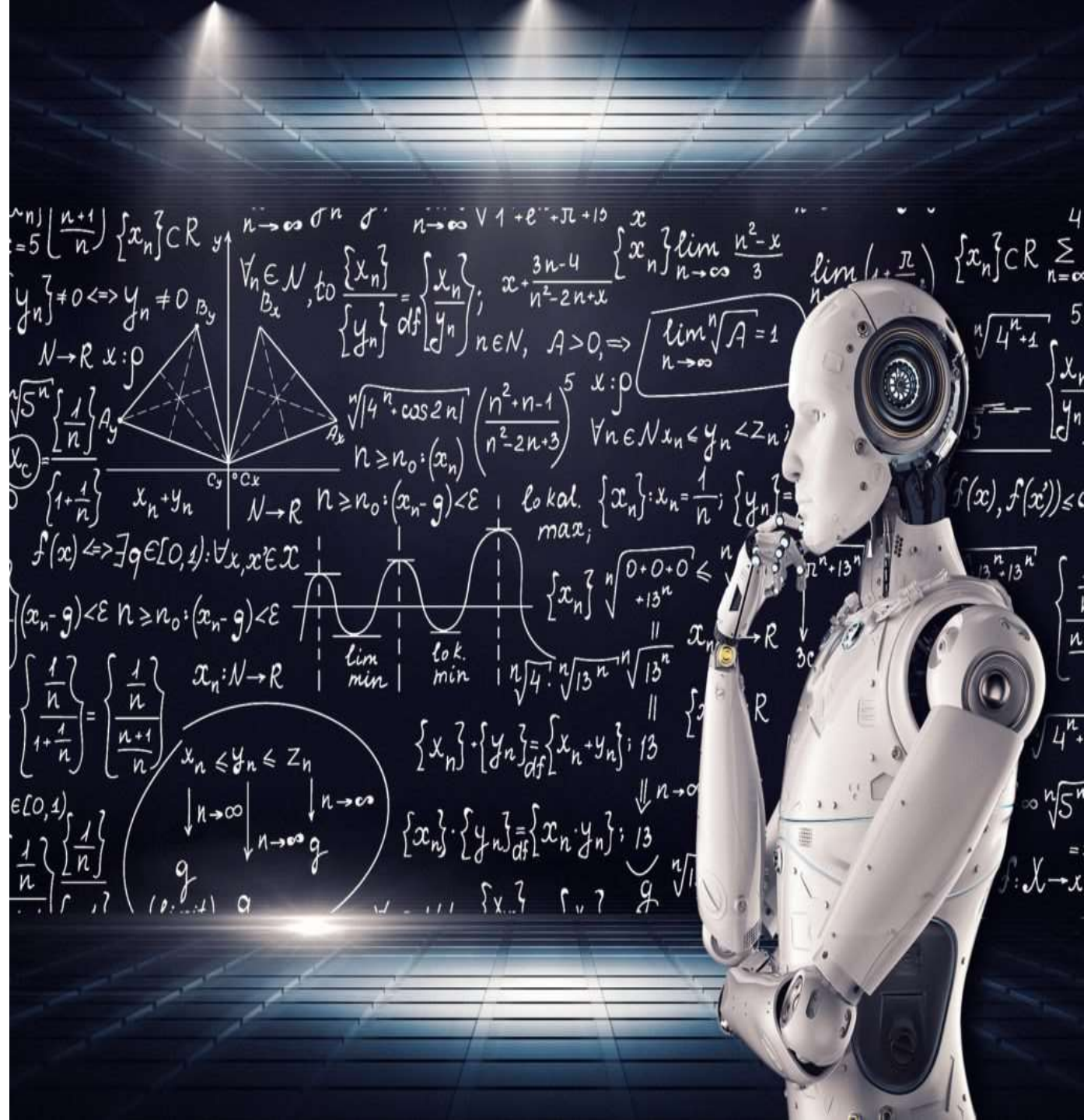
# PAIR PLOT

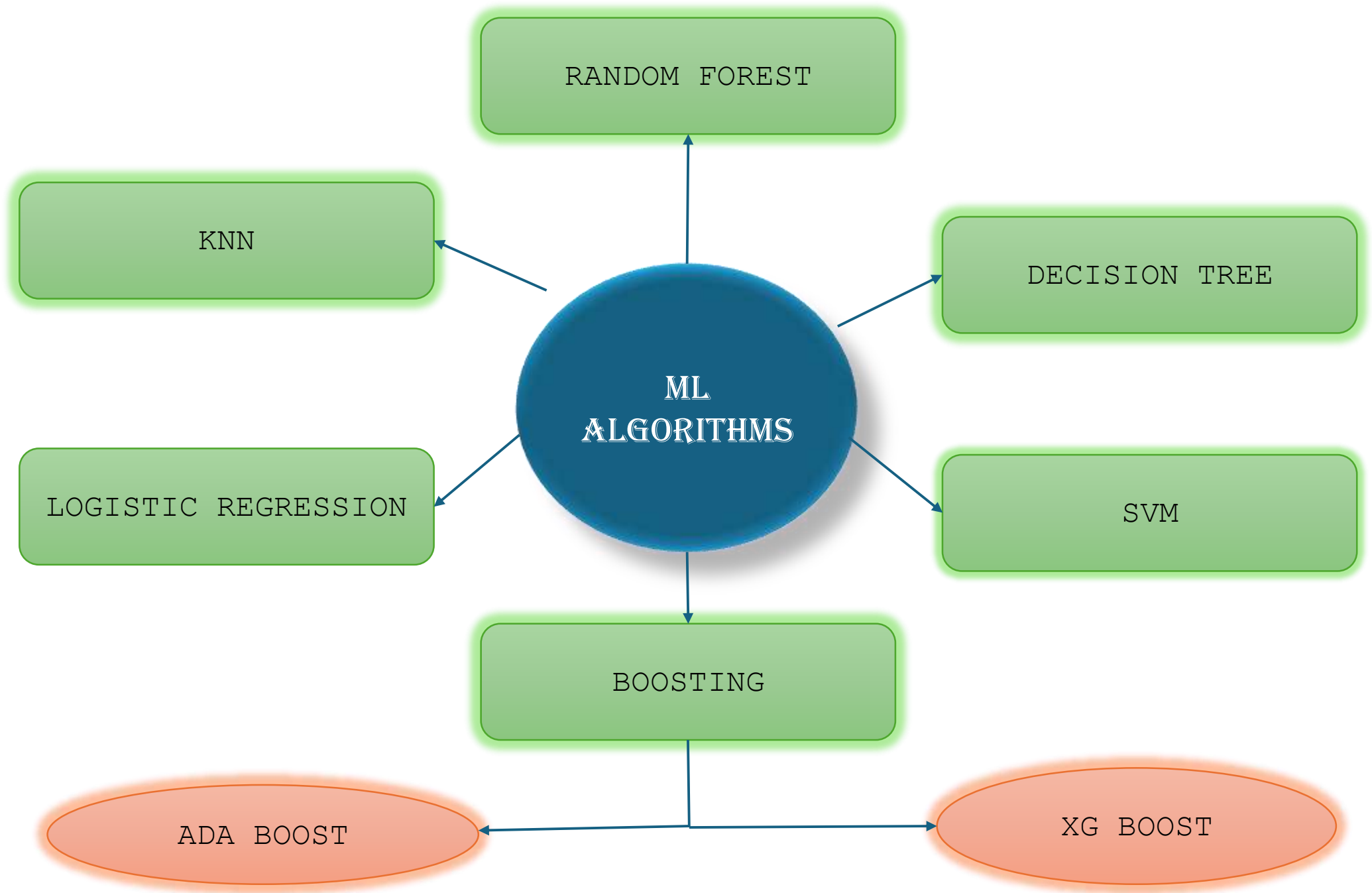


- Age vs BMI: Data is widely spread with no clear pattern between age and BMI.
- Age vs Children: Data shows that people of various ages have different numbers of children, with most individuals having 0–3 children.
- BMI vs Children: BMI values are consistent across different numbers of children.
- Charges: Larger and darker dots represent higher charges. These appear more scattered, but higher charges seem to cluster for higher BMI and age.



# MACHINE LEARNING ALGORITHMS





# 60:40 TRAIN TEST SPLIT

ALGORITHMS	MODEL 1
LINEAR REGRESSION	11431.265806495396
KNN	11729.448347067131
SVM	13125.787740041129
DECISION TREE	16179.484384739933
RANDOM FOREST	13122.963924673244
XG BOOST	13968.427863698407
ADA BOOST	11811.288756729913

# 70:30 TRAIN TEST SPLIT

ALGORITHMS	MODEL 1
LINEAR REGRESSION	11182.436478201516
KNN	11036.302509526595
SVM	12216.363130385913
DECISION TREE	16179.484384739933
RANDOM FOREST	12491.832175681928
XG BOOST	13099.292006046864
ADA BOOST	11040.990002151222

# 75:25 TRAIN TEST SPLIT

ALGORITHMS	MODEL 1
LINEAR REGRESSION	10847.593384283526
KNN	11261.344306644703
SVM	12724.896113492025
DECISION TREE	15996.586496155332
RANDOM FOREST	12789.831896644697
XG BOOST	13901.025024984147
ADA BOOS	11565.838526413283

# 80:20 TRAIN TEST SPLIT

ALGORITHMS	MODEL 1
LINEAR REGRESSION	11425.44386403564
KNN	11540.148271602695
SVM	13033.447076242333
DECISION TREE	16429.20930806319
RANDOM FOREST	13225.56472929329
XG BOOST	13713.484479004674
ADA BOOST	11655.560366092102

# ALGORITHMS COMPARISION

ALGORITHMS	MODEL 1
LOGISTIC REGRESSION	11425.44386403564
KNN	11540.148271602695
SVM	13033.447076242333
DECISION TREE	16429.20930806319
RANDOM FOREST	13225.56472929329
XG BOOST	13713.484479004674
ADA BOOST	11655.560366092102



# NEURAL NETWORK

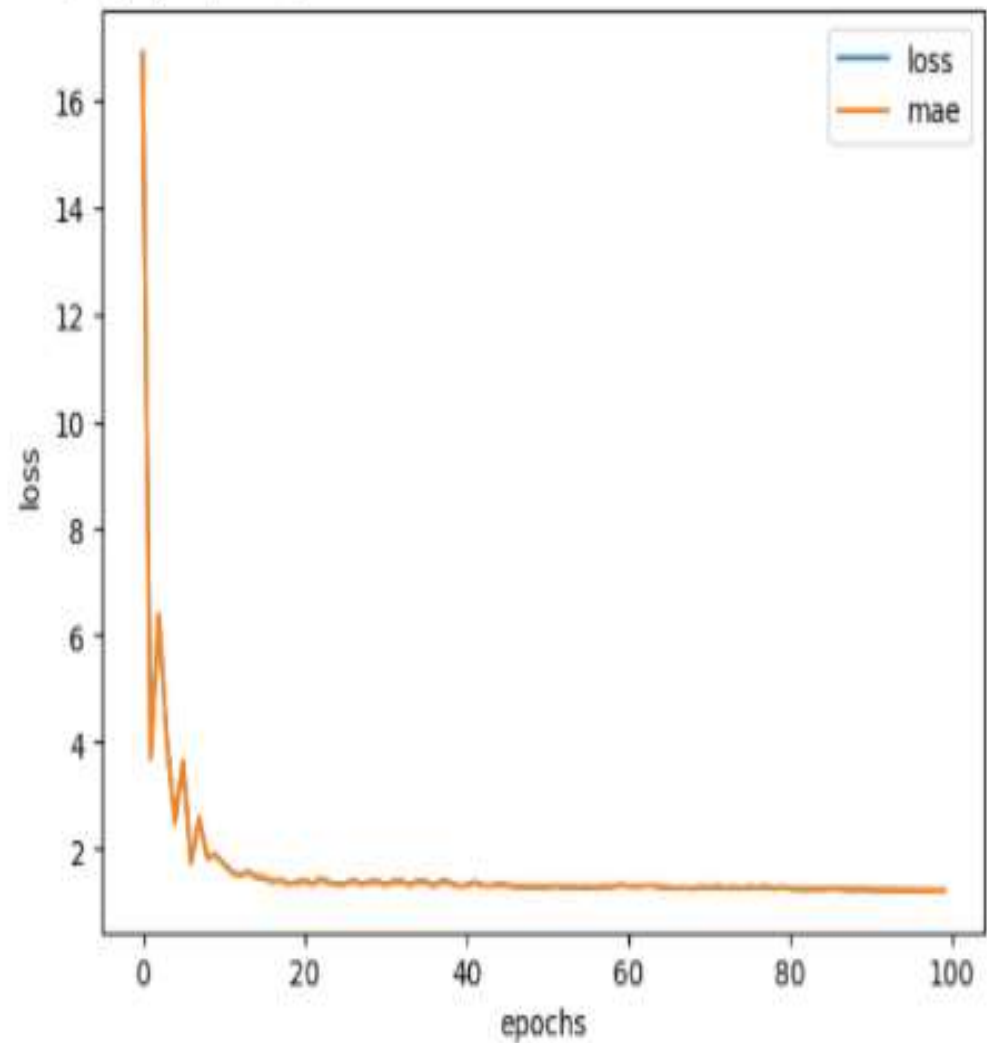




Train test	Architecture	Optimizer	Epochs	Mae
60-40	34-32-30-26	Adam	100	1.5688951015472412
70-30	34-32-30-26	Adam	100	1.2450635433197021
75-25	34-32-30-26	Adam	100	2.426121234893799
80-20	34-32-30-26	Adam	100	1.582599401473999

# NEURAL NETWORK PLOT

Text(0.5, 0, 'epochs')



Train-test split	80-20
Architecture	34-32-30-26
Optimizer	Adams
Epochs	100

# SUMMARY

- The main aim of this research is to predict the charges of the medical insurance based on the BMI and Gender.
- Based on the table, the algorithm with the lowest score (which typically indicates better performance for metrics like error or loss) is **Logistic Regression**, with a score of **11425.44**. This suggests that Logistic Regression is the best-performing algorithm in this scenario.

# WORK DISTRIBUTION

TEAM MEMBER	WORK DONE
A RAHUL	COLLECTED REQUIRED INFORMATION AND DATA
AKASH	DATA PRE PROCESSING
Y SRUTHI	EXPLORATORY DATA ANALYSIS
SHAN KOUSHIK	IMPLMENTATION OF ML ALGORITHMS



Thank You

SHAN KOUSHIK  
A RAHUL  
Y SRUTHI  
V AKASH

Collab Notebook  
Link

# LOADING THE DATASET

```
data= pd.read_excel('Insurance1.xlsx',names = ["age","sex","bmi","children","smoker","region","charges"], header=None)  
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()
```



```
0
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   object
 4   smoker      1338 non-null   object
 5   region      1338 non-null   object
 6   charges     1338 non-null   float64
dtypes: float64(2), int64(1), object(4)
memory usage: 73.3+ KB
```

# REPLACING THE VARIABLES



```
df= df.replace(to_replace='yes', value='1')  
df= df.replace(to_replace='no', value='0')  
df
```





	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	southwest	16884.92400
1	18	1	33.770	1	0	southeast	1725.55230
2	28	1	33.000	3	0	southeast	4449.46200
3	33	1	22.705	0	0	northwest	21984.47061
4	32	1	28.880	0	0	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	northwest	10600.54830
1334	18	0	31.920	0	0	northeast	2205.98080
1335	18	0	36.850	0	0	southeast	1629.83350
1336	21	0	25.800	0	0	southwest	2007.94500
1337	61	0	29.070	0	1	northwest	29141.36030

1338 rows × 7 columns



# DROPPING THE TARGET VARIABLES

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

```

   age  sex    bmi  children  smoker    region
0    19    0  27.900         0        1  southwest
1    18    1  33.770         1        0  southeast
2    28    1  33.000         3        0  southeast
3    33    1  22.705         0        0  northwest
4    32    1  28.880         0        0  northwest
...  ...  ..  ...         ...      ...  ...
1333  50    1  30.970         3        0  northwest
1334  18    0  31.920         0        0  northeast
1335  18    0  36.850         0        0  southeast
1336  21    0  25.800         0        0  southwest
1337  61    0  29.070         0        1  northwest

[1338 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: 1338, dtype: float64
```

# LINEAR REGRESSION

```
▶ from sklearn import metrics
X = data[['age', 'bmi']]
y = data.charges
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
lm2 = LinearRegression()
lm2.fit(X_train, y_train)
y_pred = lm2.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

➡ 10847.593384283526

```
[ ] test_sizes = [0.20]

# Loop over different test sizes
for test_size in test_sizes:
    X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=test_size, random_state=1)

    # Train and fit the model
    lm2 = LinearRegression()
    lm2.fit(X_train1, y_train1)
    y_pred = lm2.predict(X_test1)

    # Calculate the Root Mean Squared Error (RMSE)
    rmse = np.sqrt(metrics.mean_squared_error(y_test1, y_pred))
    print(f" for Test size {test_size * 100}% RMSE: {rmse}")
```

➡ for Test size 20.0% RMSE: 11431.265806495396

```
[ ] test_sizes = [0.30]

# Loop over different test sizes
for test_size in test_sizes:
    X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=test_size, random_state=1)

    # Train and fit the model
    lm2 = LinearRegression()
    lm2.fit(X_train3, y_train3)
    y_pred = lm2.predict(X_test3)

    # Calculate the Root Mean Squared Error (RMSE)
    rmse = np.sqrt(metrics.mean_squared_error(y_test3, y_pred))
    print(f" for Test size {test_size * 100}% RMSE: {rmse}")
```

 for Test size 30.0% RMSE: 11182.436478201516

```
▶ test_sizes = [0.40]

# Loop over different test sizes
for test_size in test_sizes:
    X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=test_size, random_state=1)

    # Train and fit the model
    lm2 = LinearRegression()
    lm2.fit(X_train4, y_train4)
    y_pred = lm2.predict(X_test4)
```

# KNN

```
[ ] #knn
    from sklearn.neighbors import KNeighborsRegressor
    model=KNeighborsRegressor(n_neighbors=25)
    model.fit(X_train1, y_train1)
```



KNeighborsRegressor  
KNeighborsRegressor(n\_neighbors=25)



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



```
16073.5700056 , 6348.9743044 , 22857.6255396 , 15080.1226792 ,
10722.2342716 , 20278.9248068 , 15184.2099496 , 21683.7448564 ,
8720.0872444 , 15470.41036 , 10224.110586 , 9292.5937892 ,
11638.26348 , 16956.290602 , 24481.1012072 , 17945.9389416 ,
5064.7335008 , 11078.91848 , 9622.10569836 , 11514.7706336 ,
9007.9871828 , 25457.1968124 , 4856.8487148 , 11756.582768 ,
15284.8238696 , 26054.9786516 , 7749.489838 , 7147.27076 ,
21807.9889884 , 15644.3422528 , 13546.9324108 , 13005.453128 ,
5273.4814348 , 6249.4965072 , 18703.6134092 , 11367.2384544 ,
9427.15934476 , 16615.1606348 , 22127.0971132 , 12147.9669216 ,
5102.712322 , 8535.8044756 , 14277.114974 , 17125.7769888 ,
5745.0147488 , 17989.0402408 , 17415.6802224 , 10085.0134236 ,
9619.186014 , 14122.2885772 , 13094.5477408 , 16384.71948 ,
17030.3138888 , 9679.58257 , 7881.514732 , 15558.2373244 ,
22355.8506024 , 15568.9653328 , 6000.4769472 , 16093.896042 ,
17281.9492908 , 19186.3510904 , 20785.8332268 , 8803.3746076 ,
15528.9044252 , 5870.6278188 , 8818.7633284 , 9784.3601392 ,
18910.710548 , 7749.489838 , 5296.9582268 , 15126.9496672 ,
```

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

⇒ 0.0783804324137155

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

⇒ 9154.393822653283

```
▶ from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test1,y_pred1)
```

⇒ 137579958.52651587

```
[ ] mse = mean_squared_error(y_test1, y_pred1)
    rmse = np.sqrt(mse)
    rmse
```

⇒ 11729.448347067131

```
[ ] from sklearn.metrics import mean_absolute_percentage_error
    mape = mean_absolute_percentage_error(y_test1, y_pred1)
    print(mape)
    mape = mape * 100
```

```
[ ] from sklearn import metrics  
    metrics.mean_absolute_error(y_test4,y_pred4)
```

```
⇒ 9034.187735966567
```

```
[ ] from sklearn.metrics import mean_squared_error  
    mean_squared_error(y_test4,y_pred4)
```

```
⇒ 133175022.13057467
```

```
▶ mse = mean_squared_error(y_test4, y_pred4)  
  rmse = np.sqrt(mse)  
  rmse
```

```
⇒ 11540.148271602695
```

```
[ ] from sklearn.metrics import mean_absolute_percentage_error  
    mape = mean_absolute_percentage_error(y_test4, y_pred4)  
    print(mape)  
    mape = mape * 100  
    mape
```

```
⇒ 1.1683706233560116
```

---

# SVM

```
#svm
from sklearn.svm import SVR
model = SVR(kernel='linear')
model.fit(X_train1, y_train1)
```



SVR

SVR(kernel='linear')

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



```
array([ 1905.20936523, 12088.60091979, 10451.71193829,  9919.22456837,
        2519.31095978,  5120.29709905,  9913.94364795, 11838.92012159,
        4101.05032919,  6772.36872676, 12932.72988712, 11195.56342814,
        7271.43027085,  8093.53539962,  1644.45663729,  9690.45741524,
        5725.12707764,  6229.85961014, 12593.42346798, 13415.40857583,
       10741.56973897,  4824.97834657,  9170.51223131, 10246.76893749,
        1670.35115048,  7829.75214349,  8370.7309934 ,  9671.67414148,
        5663.9464144 ,  4906.53256074, 11795.68258568,  5966.70646383,
       11072.77474611,  1926.87314103,  9716.56196503, 11062.51295757,
        4847.4822688 ,  2498.90740362, 12310.88694332,  8886.62547132,
        4326.57691753, 12338.85181735, 10239.387651 , 12589.94286134,
        5185.04838462, 12908.33563542,  2173.88347379,  4334.49829815,
       10565.43175864, 12105.13380132, 13157.83640223, 11564.93508739,
        3313.24117791,  8915.40048655, 1686.88403201,  5328.66069603,
        8339.52555457, 12869.29883165,  3042.46670327, 1592.96766323,
        4578.74814978, 13146.04434698,  3016.87224239,  3019.09262938,
       12858.16689145,  9677.85521879,  9966.87287304, 12133.96882701,
```

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

⇒ -0.15411016623814744

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

⇒ 6638.806276256675

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

⇒ 172286303.79661402

```
▶ mse = mean_squared_error(y_test1, y_pred1)
  rmse = np.sqrt(mse)
  rmse
```

⇒ 13125.787740041129

```
[ ] from sklearn.metrics import mean_absolute_percentage_error
    mape = mean_absolute_percentage_error(y_test1, y_pred1)
    print(mape)
    mape = mape * 100
    mape
```



# DECISSION TREE

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

⇒ -0.5953894661491503

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

⇒ 9840.463405257462

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

⇒ 238160759.93405038

```
▶ mse = mean_squared_error(y_test1, y_pred1)  
  rmse = np.sqrt(mse)  
  rmse
```

⇒ 15432.45800039807

```
[ ] from sklearn.metrics import mean_absolute_percentage_error  
    mape = mean_absolute_percentage_error(y_test1, y_pred1)  
    print(mape)  
    mape = mape * 100  
    mape
```

```
[ ] #evaluation metrix
    from sklearn.metrics import r2_score
    r2_score(y_test4,y_pred4)
```

⇒ -0.8778575040057566

```
[ ] from sklearn import metrics
    metrics.mean_absolute_error(y_test4,y_pred4)
```

⇒ 10601.602630725745

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_test4,y_pred4)
```

⇒ 269918918.4881502

```
▶ mse = mean_squared_error(y_test4, y_pred4)
  rmse = np.sqrt(mse)
  rmse
```

⇒ 16429.20930806319

```
[ ] from sklearn.metrics import mean_absolute_percentage_error
    mape = mean_absolute_percentage_error(y_test4, y_pred4)
    print(mape)
    mape = mape * 100
    mape
```

# RANDOM FOREST

```
[ ] #random forest
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.tree import plot_tree
    rf=RandomForestRegressor()
    rf.fit(X_train1,y_train1)
```



RandomForestRegressor ① ②  
RandomForestRegressor()



```
y_pred1=rf.predict(X_test1)
y_pred1
```



```
18831.5991559 , 2906.145957 , 15403.7342934 , 15869.037168 ,
 6071.69111817, 13899.2869611 , 17827.9928102 , 27452.9548311 ,
11540.3286458 , 13739.5136059 , 12121.320012 , 11945.438202 ,
10078.070628 , 12998.9869255 , 28122.1670988 , 19408.4177548 ,
 8839.6924825 , 9282.9755485 , 12779.7482906 , 6183.7768159 ,
 8127.3467855 , 33268.1579711 , 3353.5000062 , 27821.725882 ,
36772.4968488 , 30523.2474944 , 13873.67089863, 12146.98637113,
34187.265596 , 11966.820441 , 9240.4917561 , 11584.7064055 ,
 5325.516317 , 7405.4222255 , 19331.601352 , 11340.7966155 ,
14153.03985119, 21238.342086 , 16294.2820805 , 15214.6667204 ,
 8069.9575783 , 15222.6637165 , 13416.1151212 , 16944.4850301 ,
 7701.2199195 , 15508.9844468 , 20840.0487416 , 8194.18276367,
29646.296648 , 8259.1937961 , 12020.2596226 , 20636.4459017 ,
18628.3051758 , 8033.0073406 , 17591.2261975 , 14089.4268326 ,
38271.3875094 , 11859.145185 , 3243.5791585 , 14547.3252038 ,
16149.7764742 , 20278.377667 , 18310.009187 , 7059.2926134 ,
12772.699334 , 4170.0304498 , 10945.7706676 , 25013.8575045 ,
```

```
[ ] #evaluation metrix  
    print("RMSE",np.sqrt(metrics.mean_squared_error(y_test4,y_pred4)))  
    print("R2 score:",metrics.r2_score(y_test4,y_pred4))
```

⇒ RMSE 13225.56472929329  
R2 score: -0.21690803770405154

```
[ ] from sklearn import metrics  
    metrics.mean_absolute_error(y_test4,y_pred4)
```

⇒ 9722.17949024921

▶ 

```
from sklearn.metrics import mean_squared_error  
mean_squared_error(y_test4,y_pred4)
```

⇒ 174915562.4087267

```
[ ] from sklearn.metrics import mean_absolute_percentage_error  
    mape = mean_absolute_percentage_error(y_test4, y_pred4)  
    print(mape)  
    mape = mape * 100  
    mape
```

⇒ 1.3045008883519413  
130.45008883519412

# XG BOOST

```
[ ] #XGboost
import xgboost as xgb
model1 = xgb.XGBRegressor()
model2 = xgb.XGBRegressor(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5)

train_model1 = model1.fit(X_train1, y_train1)
train_model2 = model2.fit(X_train1, y_train1)
```

```
[ ] pred1 = train_model1.predict(X_test1)
pred2 = train_model2.predict(X_test1)
```

```
[ ] #evaluation metrix
print("RMSE1:",np.sqrt(metrics.mean_squared_error(y_test1, pred1)))
print("RMSE2:",np.sqrt(metrics.mean_squared_error(y_test1, pred2)))
print("R2 score1:",metrics.r2_score(y_test1,pred1))
print("R2 score2:",metrics.r2_score(y_test1,pred2))
```



```
RMSE1: 14355.489943493878
RMSE2: 13968.427863698407
R2 score1: -0.38048773140260383
R2 score2: -0.30704810411377537
```



```
mae1=metrics.mean_absolute_error(y_test1,pred1)
mae2=metrics.mean_absolute_error(y_test1,pred2)
print("MAE1:",mae1)
print("MAE2:",mae2)
```



```
MAE1: 10111.485606117854
```

```
[ ] #evaluation metrix
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test4, pred1)))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test4, pred2)))
print("R2 score:",metrics.r2_score(y_test4,pred1))
print("R2 score:",metrics.r2_score(y_test4,pred2))
```

RMSE: 14464.566003670305  
RMSE: 13713.484479004674  
R2 score: -0.4555935549630441  
R2 score: -0.3083530389136093

```
mae1=metrics.mean_absolute_error(y_test4,pred1)
mae2=metrics.mean_absolute_error(y_test4,pred2)
print("MAE1:",mae1)
print("MAE2:",mae2)
```

MAE1: 10253.346605412888  
MAE2: 9907.443675122599

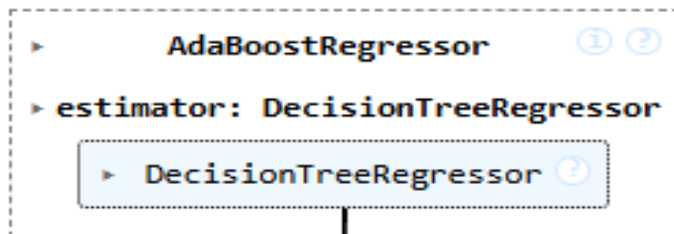
```
[ ] from sklearn.metrics import mean_squared_error
mse1=mean_squared_error(y_test4,pred1)
mse2=mean_squared_error(y_test4,pred2)
print("MSE1:",mse1)
print("MSE2:",mse2)
```

MSE1: 209223669.67453474  
MSE2: 188059656.5559021

# ADA BOOST

```
[ ] #ADABOOST
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
# Replace 'base_estimator' with 'estimator'
base_estimator = DecisionTreeRegressor(max_depth=3, random_state=0)
adaboost = AdaBoostRegressor(estimator=base_estimator, # Changed argument name here
                             n_estimators=3, random_state=0)
```

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



```
y_pred1 = adaboost.predict(X_test1)
y_pred1
```



```
array([16463.52109117, 24067.46585327, 19149.97186129, 19149.97186129,
       7837.68691867, 16463.52109117, 19149.97186129, 16059.58574176,
       9359.48958517, 9359.48958517, 16210.85696644, 19149.97186129,
       16463.52109117, 15807.66626925, 13869.39064146, 16059.58574176,
       9359.48958517, 9359.48958517, 24067.46585327, 24067.46585327,
       17629.42631379, 16463.52109117, 15807.66626925, 15807.66626925,
       6750.59661578, 15807.66626925, 15807.66626925, 17629.42631379,
       16463.52109117, 9359.48958517, 24067.46585327, 9359.48958517,
```

```
[ ] #evaluation metrix
    print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test4,y_pred4)))
    print("R2 score:",metrics.r2_score(y_test4,y_pred4))
```

➞ RMSE: 11655.560366092102  
R2 score: 0.05486112912254548

```
[ ] metrics.mean_absolute_error(y_test4,y_pred4)
```

➞ 9539.25543417836

▶ mean\_squared\_error(y\_test4,y\_pred4)


➞ 135852087.44761705

```
[ ] mape = mean_absolute_percentage_error(y_test4, y_pred4)
    print(mape)
    mape = mape * 100
    mape
```

➞ 1.3917498934940569  
139.17498934940568




# ANN

```
✓ 1s  #ann
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
```

```
✓ 0s [544] # Convert 'Approved' column to numeric (if it's not already)
y = pd.to_numeric(y, errors='coerce')


# Convert all columns in X to numeric
for column in X.columns:
    X[column] = pd.to_numeric(X[column], errors='coerce')
```



 <ipython-input-544-a855d7058886e>:6: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead


See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
X[column] = pd.to\_numeric(X[column], errors='coerce')


```
✓ 0s [545] from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.40, random_state=42)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.30, random_state=42)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.25, random_state=42)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.20, random_state=42)
```

# 60-40

✓ 0s  `model.evaluate(X_test1, y_test1)`

 17/17  1s 5ms/step - accuracy: 0.0000e+00 - loss: -399050964992.0000 - precision: 1.0000 - recall: 1.0000  
[-391140868096.0, 0.0, 1.0, 1.0]


✓ 1s  `model.summary();`

 Model: "sequential\_13"

Layer (type)	Output Shape	Param #
dense_52 (Dense)	(None, 30)	90
dense_53 (Dense)	(None, 20)	620
dense_54 (Dense)	(None, 10)	210
dense_55 (Dense)	(None, 1)	11

Total params: 2,795 (10.92 KB)  
Trainable params: 931 (3.64 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 1,864 (7.29 KB)

✓ 2s [552] `pd.DataFrame(history.history).plot()`  
`plt.ylabel("loss")`  
`plt.xlabel("epochs")`

 `Text(0.5, 0, 'epochs')`

## 70-30

✓ [558] model.evaluate(X\_test2, y\_test2)

13/13 ————— 1s 5ms/step - accuracy: 0.0000e+00 - loss: -945034952704.0000 - precision: 1.0000 - recall: 1.0000  
[-911839592448.0, 0.0, 1.0, 1.0]

✓ [559] model.summary();

Model: "sequential\_17"

Layer (type)	Output Shape	Param #
dense_68 (Dense)	(None, 30)	90
dense_69 (Dense)	(None, 20)	620
dense_70 (Dense)	(None, 10)	210
dense_71 (Dense)	(None, 1)	11

Total params: 2,795 (10.92 KB)  
Trainable params: 931 (3.64 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 1,864 (7.29 KB)

✓ [561] pd.DataFrame(history.history).plot()  
plt.ylabel("loss")  
plt.xlabel("epochs")

Text(0.5, 0, 'epochs')

# 75-25

✓ [564] model.evaluate(X\_test3, y\_test3)

11/11 ————— 2s 7ms/step - accuracy: 0.0000e+00 - loss: -618509041664.0000 - precision: 1.0000 - recall: 1.0000  
[-603104608256.0, 0.0, 1.0, 1.0]

✓ 0s model.summary();

Model: "sequential\_19"

Layer (type)	Output Shape	Param #
dense_76 (Dense)	(None, 30)	90
dense_77 (Dense)	(None, 20)	620
dense_78 (Dense)	(None, 10)	210
dense_79 (Dense)	(None, 1)	11

Total params: 2,795 (10.92 KB)  
Trainable params: 931 (3.64 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 1,864 (7.29 KB)

✓ [567] pd.DataFrame(history.history).plot()  
plt.ylabel("loss")  
plt.xlabel("epochs")

Text(0.5, 0, 'epochs')

## 80-20

✓ [569] model.evaluate(X\_test4, y\_test4)

2s

9/9 ————— 1s 12ms/step - accuracy: 0.0000e+00 - loss: -1266557779968.0000 - precision: 1.0000 - recall: 1.0000  
[-1209574752256.0, 0.0, 1.0, 1.0]

✓ model.summary();

0s

Model: "sequential\_20"

Layer (type)	Output Shape	Param #
dense_80 (Dense)	(None, 30)	90
dense_81 (Dense)	(None, 20)	620
dense_82 (Dense)	(None, 10)	210
dense_83 (Dense)	(None, 1)	11

Total params: 2,795 (10.92 KB)  
Trainable params: 931 (3.64 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 1,864 (7.29 KB)

✓ [571] pd.DataFrame(history.history).plot()  
plt.ylabel("loss")  
plt.xlabel("epochs")

0s

Text(0.5, 0, 'epochs')