

# Medical Insurance Price Predictor

A Project of

INTRODUCTION TO MACHINE LEARNING

20 by ANAND CHOUDHARY



# Introduction

In this project, we are going to develop various machine learning models for medical insurance price prediction on a dataset and we will compare their accuracy(R-squared score).

This research will help insurance companies, customers, policymakers and researchers and analysts while understanding and predicting the high medical insurance bills.

The annual premium of medical insurances is constantly increasing across the globe. The average annual premium for employer-sponsored health insurance in the US was \$7,470 for single coverage and \$21,342 for family coverage in 2020.

# Literature Review

We have studied 2 papers and multiple Kaggle articles to understand the progress made on this topic till now.

The first paper was titled "Implementation of Medical Insurance Price Prediction System using Regression Algorithms" and it was about implementing various regression models for the problem and their explanation.

The second paper was titled "Random forest regression with hyper parameter tuning for medical insurance premium prediction" and it was about optimizing random forest regression for more accurate predictions (98%).



# Data Collection

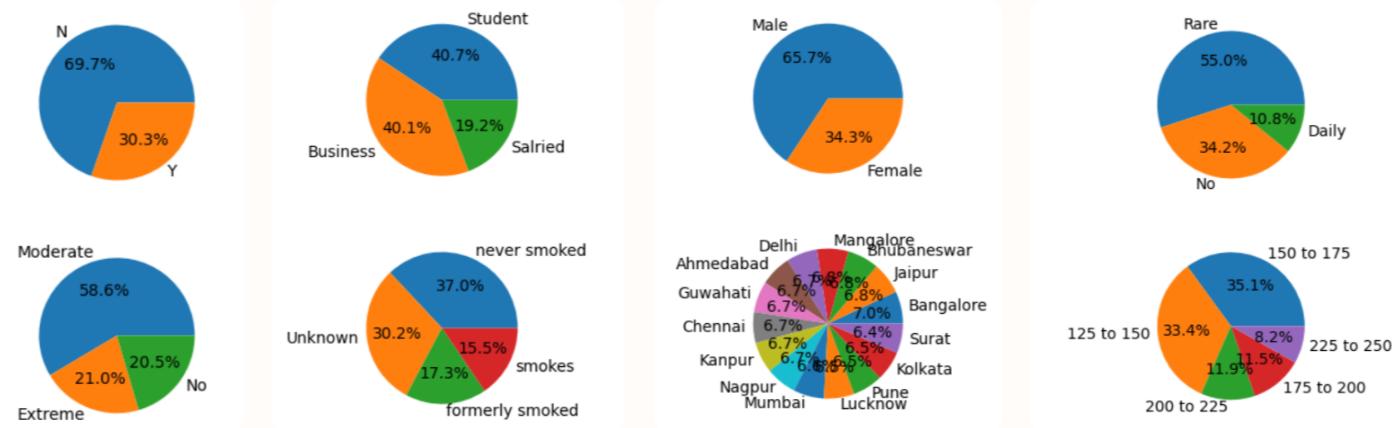
Dataset was downloaded from Kaggle ([www.kaggle.com/datasets/gdeepakreddy/insurance](http://www.kaggle.com/datasets/gdeepakreddy/insurance)).

It consists of twenty-four features in which 23 are independent variables and 1 is target variable .  
There were 25000 records in this dataset.

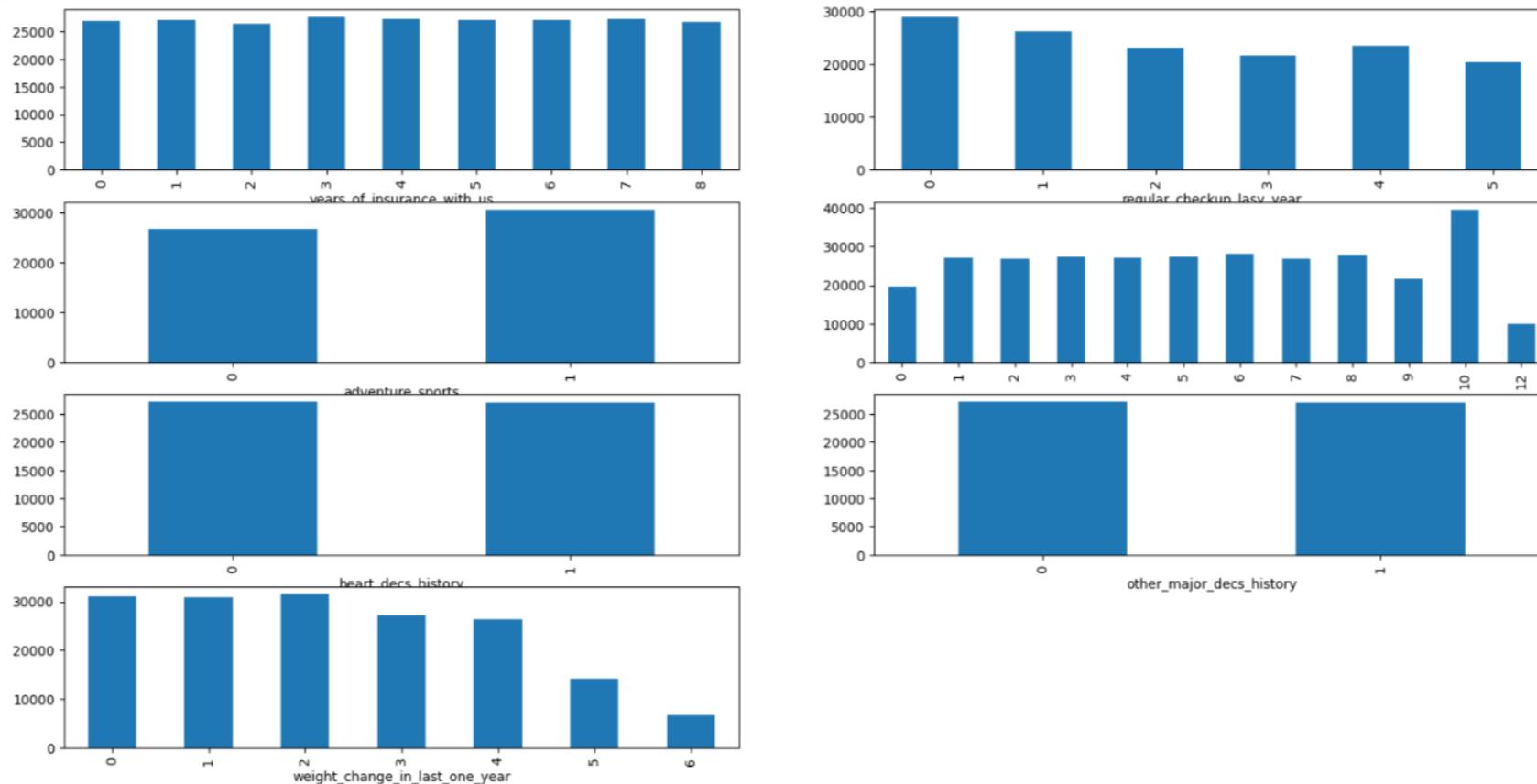
The insurance\_cost is target variable with continuous values from 2468 to 67870. The decision is taken based on the values of the independent features.

The attribute named 'applicant\_id' was dropped. After that, we are left with 22 attributes among which 8 are string literals and 2 are decimal values and the remaining 12 are integers.

# Dataset Features



Pie Chart type data visualization of different features



Bar Graph type data visualization of different features

# Data Preprocessing

- We drop any duplicate rows from the dataframe using df.drop\_duplicates (inplace=True).
- We create a boxplot of all the columns using the seaborn library and it shows the outliers and other important feature distribution of the attributes.

```
# Removing duplicate rows
df.drop_duplicates(inplace=True)

# Box Plots

fig, axs = plt.subplots(dpi=95, figsize=(47,10),ncols=len(df.columns))

# Loop through each attribute and create a box plot
i=0
for i, col in enumerate(df.columns):
    axs[i].boxplot(df[col], vert=False)
    axs[i].set_ylabel(col)

# Show the plot
plt.show()
```

- We calculate the interquartile range (IQR) of the bmi column using df['bmi'].quantile() and prints the lower and upper limits for detecting outliers using the 1.5\*IQR rule. We use it for attributes like 'bmi', 'fat\_percentage' and 'avg\_glucose\_level' and then remove the outliers using it.

```
# Finding upper and lower limit for bmi
Q1=df['bmi'].quantile(0.25)
Q2=df['bmi'].quantile(0.5)
Q3=df['bmi'].quantile(0.75)
iqr=Q3-Q1
lowlim=Q1-1.5*iqr
upplim=Q3+1.5*iqr
print('The lower and upper limit for bmi is :')
print(lowlim)
print(upplim)

# Drop the outliers
clean_data = df[(df['bmi'] >= lowlim) & (df['bmi'] <= upplim)]
```

- We calculate the skewness of the bmi and age columns using df['bmi'].skew() and df['age'].skew() and shows how symmetric or asymmetric the distributions are.
- We convert all non-numeric attribute values to integers using mapping and by assigning them some values for better performance.

```
# Showing the skewness of age and bmi
df['bmi'].skew()

1.0564277700656508

df['age'].skew()

0.01386002805629118

# Mapping all non-numeric column values to integers
df['Gender']=df['Gender'].map({'Male':0,'Female':1})
df['covered_by_any_other_company']=df['covered_by_any_other_company'].map({'Y':0,
df['smoking_status']=df['smoking_status'].map({'Unknown':1,'never smoked':0,'formal':2,'former':3,'smokes':4})
df['Location']=df['Location'].map({'Chennai':0, 'Jaipur':1, 'Bangalore':2, 'Guwahati':3, 'Kolkata':4})
df['Occupation']=df['Occupation'].map({'Salried':0, 'Student':1, 'Business':2})
df['cholesterol_level']=df['cholesterol_level'].map({'125 to 150':0, '150 to 175':1, '175 to 200':2, '200 to 225':3, '225 to 250':4, '250 to 275':5, '275 to 300':6, '300 to 325':7, '325 to 350':8, '350 to 375':9, '375 to 400':10, '400 to 425':11, '425 to 450':12, '450 to 475':13, '475 to 500':14, '500 to 525':15, '525 to 550':16, '550 to 575':17, '575 to 600':18, '600 to 625':19, '625 to 650':20, '650 to 675':21, '675 to 700':22, '700 to 725':23, '725 to 750':24, '750 to 775':25, '775 to 800':26, '800 to 825':27, '825 to 850':28, '850 to 875':29, '875 to 900':30, '900 to 925':31, '925 to 950':32, '950 to 975':33, '975 to 1000':34, '1000 to 1025':35, '1025 to 1050':36, '1050 to 1075':37, '1075 to 1100':38, '1100 to 1125':39, '1125 to 1150':40, '1150 to 1175':41, '1175 to 1200':42, '1200 to 1225':43, '1225 to 1250':44, '1250 to 1275':45, '1275 to 1300':46, '1300 to 1325':47, '1325 to 1350':48, '1350 to 1375':49, '1375 to 1400':50, '1400 to 1425':51, '1425 to 1450':52, '1450 to 1475':53, '1475 to 1500':54, '1500 to 1525':55, '1525 to 1550':56, '1550 to 1575':57, '1575 to 1600':58, '1600 to 1625':59, '1625 to 1650':60, '1650 to 1675':61, '1675 to 1700':62, '1700 to 1725':63, '1725 to 1750':64, '1750 to 1775':65, '1775 to 1800':66, '1800 to 1825':67, '1825 to 1850':68, '1850 to 1875':69, '1875 to 1900':70, '1900 to 1925':71, '1925 to 1950':72, '1950 to 1975':73, '1975 to 2000':74, '2000 to 2025':75, '2025 to 2050':76, '2050 to 2075':77, '2075 to 2100':78, '2100 to 2125':79, '2125 to 2150':80, '2150 to 2175':81, '2175 to 2200':82, '2200 to 2225':83, '2225 to 2250':84, '2250 to 2275':85, '2275 to 2300':86, '2300 to 2325':87, '2325 to 2350':88, '2350 to 2375':89, '2375 to 2400':90, '2400 to 2425':91, '2425 to 2450':92, '2450 to 2475':93, '2475 to 2500':94, '2500 to 2525':95, '2525 to 2550':96, '2550 to 2575':97, '2575 to 2600':98, '2600 to 2625':99, '2625 to 2650':100, '2650 to 2675':101, '2675 to 2700':102, '2700 to 2725':103, '2725 to 2750':104, '2750 to 2775':105, '2775 to 2800':106, '2800 to 2825':107, '2825 to 2850':108, '2850 to 2875':109, '2875 to 2900':110, '2900 to 2925':111, '2925 to 2950':112, '2950 to 2975':113, '2975 to 3000':114, '3000 to 3025':115, '3025 to 3050':116, '3050 to 3075':117, '3075 to 3100':118, '3100 to 3125':119, '3125 to 3150':120, '3150 to 3175':121, '3175 to 3200':122, '3200 to 3225':123, '3225 to 3250':124, '3250 to 3275':125, '3275 to 3300':126, '3300 to 3325':127, '3325 to 3350':128, '3350 to 3375':129, '3375 to 3400':130, '3400 to 3425':131, '3425 to 3450':132, '3450 to 3475':133, '3475 to 3500':134, '3500 to 3525':135, '3525 to 3550':136, '3550 to 3575':137, '3575 to 3600':138, '3600 to 3625':139, '3625 to 3650':140, '3650 to 3675':141, '3675 to 3700':142, '3700 to 3725':143, '3725 to 3750':144, '3750 to 3775':145, '3775 to 3800':146, '3800 to 3825':147, '3825 to 3850':148, '3850 to 3875':149, '3875 to 3900':150, '3900 to 3925':151, '3925 to 3950':152, '3950 to 3975':153, '3975 to 4000':154, '4000 to 4025':155, '4025 to 4050':156, '4050 to 4075':157, '4075 to 4100':158, '4100 to 4125':159, '4125 to 4150':160, '4150 to 4175':161, '4175 to 4200':162, '4200 to 4225':163, '4225 to 4250':164, '4250 to 4275':165, '4275 to 4300':166, '4300 to 4325':167, '4325 to 4350':168, '4350 to 4375':169, '4375 to 4400':170, '4400 to 4425':171, '4425 to 4450':172, '4450 to 4475':173, '4475 to 4500':174, '4500 to 4525':175, '4525 to 4550':176, '4550 to 4575':177, '4575 to 4600':178, '4600 to 4625':179, '4625 to 4650':180, '4650 to 4675':181, '4675 to 4700':182, '4700 to 4725':183, '4725 to 4750':184, '4750 to 4775':185, '4775 to 4800':186, '4800 to 4825':187, '4825 to 4850':188, '4850 to 4875':189, '4875 to 4900':190, '4900 to 4925':191, '4925 to 4950':192, '4950 to 4975':193, '4975 to 5000':194, '5000 to 5025':195, '5025 to 5050':196, '5050 to 5075':197, '5075 to 5100':198, '5100 to 5125':199, '5125 to 5150':200, '5150 to 5175':201, '5175 to 5200':202, '5200 to 5225':203, '5225 to 5250':204, '5250 to 5275':205, '5275 to 5300':206, '5300 to 5325':207, '5325 to 5350':208, '5350 to 5375':209, '5375 to 5400':210, '5400 to 5425':211, '5425 to 5450':212, '5450 to 5475':213, '5475 to 5500':214, '5500 to 5525':215, '5525 to 5550':216, '5550 to 5575':217, '5575 to 5600':218, '5600 to 5625':219, '5625 to 5650':220, '5650 to 5675':221, '5675 to 5700':222, '5700 to 5725':223, '5725 to 5750':224, '5750 to 5775':225, '5775 to 5800':226, '5800 to 5825':227, '5825 to 5850':228, '5850 to 5875':229, '5875 to 5900':230, '5900 to 5925':231, '5925 to 5950':232, '5950 to 5975':233, '5975 to 6000':234, '6000 to 6025':235, '6025 to 6050':236, '6050 to 6075':237, '6075 to 6100':238, '6100 to 6125':239, '6125 to 6150':240, '6150 to 6175':241, '6175 to 6200':242, '6200 to 6225':243, '6225 to 6250':244, '6250 to 6275':245, '6275 to 6300':246, '6300 to 6325':247, '6325 to 6350':248, '6350 to 6375':249, '6375 to 6400':250, '6400 to 6425':251, '6425 to 6450':252, '6450 to 6475':253, '6475 to 6500':254, '6500 to 6525':255, '6525 to 6550':256, '6550 to 6575':257, '6575 to 6600':258, '6600 to 6625':259, '6625 to 6650':260, '6650 to 6675':261, '6675 to 6700':262, '6700 to 6725':263, '6725 to 6750':264, '6750 to 6775':265, '6775 to 6800':266, '6800 to 6825':267, '6825 to 6850':268, '6850 to 6875':269, '6875 to 6900':270, '6900 to 6925':271, '6925 to 6950':272, '6950 to 6975':273, '6975 to 7000':274, '7000 to 7025':275, '7025 to 7050':276, '7050 to 7075':277, '7075 to 7100':278, '7100 to 7125':279, '7125 to 7150':280, '7150 to 7175':281, '7175 to 7200':282, '7200 to 7225':283, '7225 to 7250':284, '7250 to 7275':285, '7275 to 7300':286, '7300 to 7325':287, '7325 to 7350':288, '7350 to 7375':289, '7375 to 7400':290, '7400 to 7425':291, '7425 to 7450':292, '7450 to 7475':293, '7475 to 7500':294, '7500 to 7525':295, '7525 to 7550':296, '7550 to 7575':297, '7575 to 7600':298, '7600 to 7625':299, '7625 to 7650':300, '7650 to 7675':301, '7675 to 7700':302, '7700 to 7725':303, '7725 to 7750':304, '7750 to 7775':305, '7775 to 7800':306, '7800 to 7825':307, '7825 to 7850':308, '7850 to 7875':309, '7875 to 7900':310, '7900 to 7925':311, '7925 to 7950':312, '7950 to 7975':313, '7975 to 8000':314, '8000 to 8025':315, '8025 to 8050':316, '8050 to 8075':317, '8075 to 8100':318, '8100 to 8125':319, '8125 to 8150':320, '8150 to 8175':321, '8175 to 8200':322, '8200 to 8225':323, '8225 to 8250':324, '8250 to 8275':325, '8275 to 8300':326, '8300 to 8325':327, '8325 to 8350':328, '8350 to 8375':329, '8375 to 8400':330, '8400 to 8425':331, '8425 to 8450':332, '8450 to 8475':333, '8475 to 8500':334, '8500 to 8525':335, '8525 to 8550':336, '8550 to 8575':337, '8575 to 8600':338, '8600 to 8625':339, '8625 to 8650':340, '8650 to 8675':341, '8675 to 8700':342, '8700 to 8725':343, '8725 to 8750':344, '8750 to 8775':345, '8775 to 8800':346, '8800 to 8825':347, '8825 to 8850':348, '8850 to 8875':349, '8875 to 8900':350, '8900 to 8925':351, '8925 to 8950':352, '8950 to 8975':353, '8975 to 9000':354, '9000 to 9025':355, '9025 to 9050':356, '9050 to 9075':357, '9075 to 9100':358, '9100 to 9125':359, '9125 to 9150':360, '9150 to 9175':361, '9175 to 9200':362, '9200 to 9225':363, '9225 to 9250':364, '9250 to 9275':365, '9275 to 9300':366, '9300 to 9325':367, '9325 to 9350':368, '9350 to 9375':369, '9375 to 9400':370, '9400 to 9425':371, '9425 to 9450':372, '9450 to 9475':373, '9475 to 9500':374, '9500 to 9525':375, '9525 to 9550':376, '9550 to 9575':377, '9575 to 9600':378, '9600 to 9625':379, '9625 to 9650':380, '9650 to 9675':381, '9675 to 9700':382, '9700 to 9725':383, '9725 to 9750':384, '9750 to 9775':385, '9775 to 9800':386, '9800 to 9825':387, '9825 to 9850':388, '9850 to 9875':389, '9875 to 9900':390, '9900 to 9925':391, '9925 to 9950':392, '9950 to 9975':393, '9975 to 10000':394, '10000 to 10025':395, '10025 to 10050':396, '10050 to 10075':397, '10075 to 10100':398, '10100 to 10125':399, '10125 to 10150':400, '10150 to 10175':401, '10175 to 10200':402, '10200 to 10225':403, '10225 to 10250':404, '10250 to 10275':405, '10275 to 10300':406, '10300 to 10325':407, '10325 to 10350':408, '10350 to 10375':409, '10375 to 10400':410, '10400 to 10425':411, '10425 to 10450':412, '10450 to 10475':413, '10475 to 10500':414, '10500 to 10525':415, '10525 to 10550':416, '10550 to 10575':417, '10575 to 10600':418, '10600 to 10625':419, '10625 to 10650':420, '10650 to 10675':421, '10675 to 10700':422, '10700 to 10725':423, '10725 to 10750':424, '10750 to 10775':425, '10775 to 10800':426, '10800 to 10825':427, '10825 to 10850':428, '10850 to 10875':429, '10875 to 10900':430, '10900 to 10925':431, '10925 to 10950':432, '10950 to 10975':433, '10975 to 110
```

# Feature Selection and Engineering

```
#correlation  
corr = df.corr()  
  
plt.figure(dpi=250,figsize=(15,8))  
sns.heatmap(df.corr(), annot=True,  
plt.show()
```

We calculate the correlation matrix of the dataframe using df.corr() and shows the linear relationship between the variables.

We have sorted attributes according to their impact on the target variable. We have also sorted attributes with feats[importance] greater than a threshold of 0.01

```
# Dropping the attributes with least significance  
df.drop(['Occupation','cholesterol_level','Gender','bmi','exercise'],axis=1)
```

```
# Divide the outcome from attributes  
X=df.drop(['insurance_cost'],axis=1)  
Y=df[['insurance_cost']]  
l1=[]  
l2=[]  
l3=[]  
cvs=0
```

We have dropped 5 attributes with the least significance named Occupation, cholesterol\_level, Gender, bmi and exercise.

We split the dataframe into two parts: X containing the independent variables and Y containing the dependent variable (charges).

We have also divide the dataframe into 80:20 respectively for training and testing.

# Model Selection and Training

We have taken a large number of regression models and have checked their training and testing accuracy using R-squared values. We have also find cross validation score for all.

## Linear Regression

```
# For linear regression
lrmodel=LinearRegression()
lrmodel.fit(xtrain,ytrain)
ypredtrain1 = lrmodel.predict(xtrain)
ypredtest1 = lrmodel.predict(xtest)
print('By Linear Regression')
print(r2_score(ytrain,ypredtrain1))
print(r2_score(ytest,ypredtest1))
print(cross_val_score(lrmodel,X,Y,cv=5).mean())
```

## Decision Tree Regressor

```
# For Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
dtmodel = DecisionTreeRegressor(max_depth=10,random_state=42)
dtmodel.fit(xtrain, ytrain)
ypredtrain4 = dtmodel.predict(xtrain)
ypredtest4 = dtmodel.predict(xtest)
print('By Decision Tree Regression')
print(r2_score(ytrain, ypredtrain4))
print(r2_score(ytest, ypredtest4))
print(cross_val_score(dtmodel, X, Y, cv=5).mean())
```

## Random Forest Regressor

```
# For Random Forest Regressor
rfmodel=RandomForestRegressor(random_state=42)
rfmodel.fit(xtrain,ytrain)
ypredtrain2=rfmodel.predict(xtrain)
ypredtest2=rfmodel.predict(xtest)
print('By Random Forest Regressor')
print(r2_score(ytrain,ypredtrain2))
print(r2_score(ytest,ypredtest2))
print(cross_val_score(rfmodel,X,Y,cv=5).mean())

# Searching for best parameters in Random Forest Regressor
from sklearn.model_selection import GridSearchCV

estimator=RandomForestRegressor(random_state=42)
param_grid={'n_estimators':[10,40,50,98,100,120,150]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
```

## XG Boost Regressor

```
# For XGBoost Regressor
xgmodel=XGBRegressor()
xgmodel.fit(xtrain,ytrain)
ypredtrain4=xgmodel.predict(xtrain)
ypredtest4=xgmodel.predict(xtest)
print('By XGBoost Regressor')
print(r2_score(ytrain,ypredtrain4))
print(r2_score(ytest,ypredtest4))
print(cross_val_score(xgmodel,X,Y,cv=5).mean())

# Searching for best parameters in XGBoost Regressor
estimator=XGBRegressor()
param_grid={'n_estimators':[10,15,20,40,50], 'max_depth':[3,4,5], 'gamma':[0,0.15,0.3,0.5,1]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
```

## Support Vector Regression using linear kernel

```
# For Support Vector Regression with linear kernel
from sklearn.svm import LinearSVR
svrmodel = LinearSVR(random_state=42)
svrmodel.fit(xtrain, ytrain)
ypredtrain3 = svrmodel.predict(xtrain)
ypredtest3 = svrmodel.predict(xtest)
print('By Support Vector Regression with linear kernel')
print(r2_score(ytrain, ypredtrain3))
print(r2_score(ytest, ypredtest3))
print(cross_val_score(svrmodel, X, Y, cv=5).mean())
```

## K Nearest Neighbors Regressor

```
# For K-Nearest Neighbors Regression
from sklearn.neighbors import KNeighborsRegressor
knnmodel = KNeighborsRegressor(n_neighbors=3)
knnmodel.fit(xtrain, ytrain)
ypredtrain5 = knnmodel.predict(xtrain)
ypredtest5 = knnmodel.predict(xtest)
print('By K-Nearest Neighbors Regression')
print(r2_score(ytrain, ypredtrain5))
print(r2_score(ytest, ypredtest5))
print(cross_val_score(knnmodel, X, Y, cv=5).mean())
```

## Gradient Boosting Regressor

```
# For Gradient Boosting Regressor
gbmodel=GradientBoostingRegressor()
gbmodel.fit(xtrain,ytrain)
ypredtrain3=gbmodel.predict(xtrain)
ypredtest3=gbmodel.predict(xtest)
print('By Gradient Boosting')
print(r2_score(ytrain,ypredtrain3))
print(r2_score(ytest,ypredtest3))
print(cross_val_score(gbmodel,X,Y,cv=5).mean())

# Searching for best parameters in Gradient Boosting Regressor
from sklearn.model_selection import GridSearchCV

estimator=GradientBoostingRegressor()
param_grid={'n_estimators':[10,15,19,20,21,50], 'learning_rate':[0.1,0.19,0.2,0.21,0.8,1]}
grid=GridSearchCV(estimator,param_grid,scoring="r2",cv=5)
grid.fit(xtrain,ytrain)
print(grid.best_params_)
```

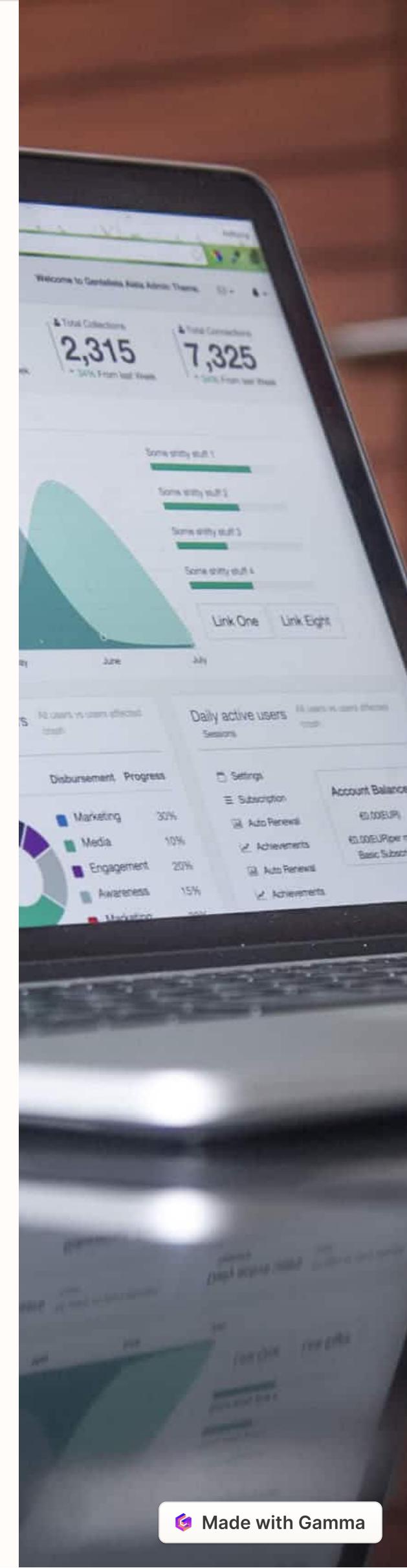
## Neural Network Regression

```
# For Neural Network Regression
from sklearn.neural_network import MLPRegressor
nnmodel = MLPRegressor(hidden_layer_sizes=(100,), max_iter=1000, random_state=42)
nnmodel.fit(xtrain, ytrain)
ypredtrain7 = nnmodel.predict(xtrain)
ypredtest7 = nnmodel.predict(xtest)
print('By Neural Network Regression')
print(r2_score(ytrain, ypredtrain7))
print(r2_score(ytest, ypredtest7))
print(cross_val_score(nnmodel, X, Y, cv=5).mean())
```

# Price Prediction

The accuracy of the above models for training data, testing data and cross validation scores based of R-squared value are given below :

MODEL	R-squared Val (Training)	R-squared Val (Testing)	Cross Validate Scor.
Linear Regression	0.9445	0.9459	0.9447
Support Vector Regression	0.9215	0.9236	0.9399
Decision Tree Regression	0.9658	0.9492	0.9482
K Nearest Neighbor Regression	0.8006	0.5997	0.5979
Random Forest Regression	0.9935	0.9552	0.9541
Random Forest Tuned Hyperparamets	0.9937	0.9553	0.9546
Gradient Boosting Regression	0.9577	0.9574	0.9569
Gradient Boosting Tuned Hyperparameters	0.9578	0.9574	0.9569
XG Boost Regression	0.9760	0.9558	0.9544
XG Boost Tuned Hyperparameters	0.9593	0.9579	0.9572
Neural Network Regressor	0.9504	0.9513	0.9501



# Comparison of Different Models

The best result in training data is for Random Forest Regression while best result in testing data is for Gradient Boosting and XG Boost with tuned hyperparameters. But the cross validation score is max in XG Boost.

The training accuracy is best in Random Forest is because we have not fixed the number of stumps which create a bit of overfitting. The testing data however has no such problem and hence is with the complex boosting models.

Below given is an model accuracy comparison published in the paper titled "Implementation of Medical Insurance Price Prediction System using Regression Algorithms" by Vijayalakshmi, V., A. Selvakumar, and K. Panimalar." in 2023 ICSSIT

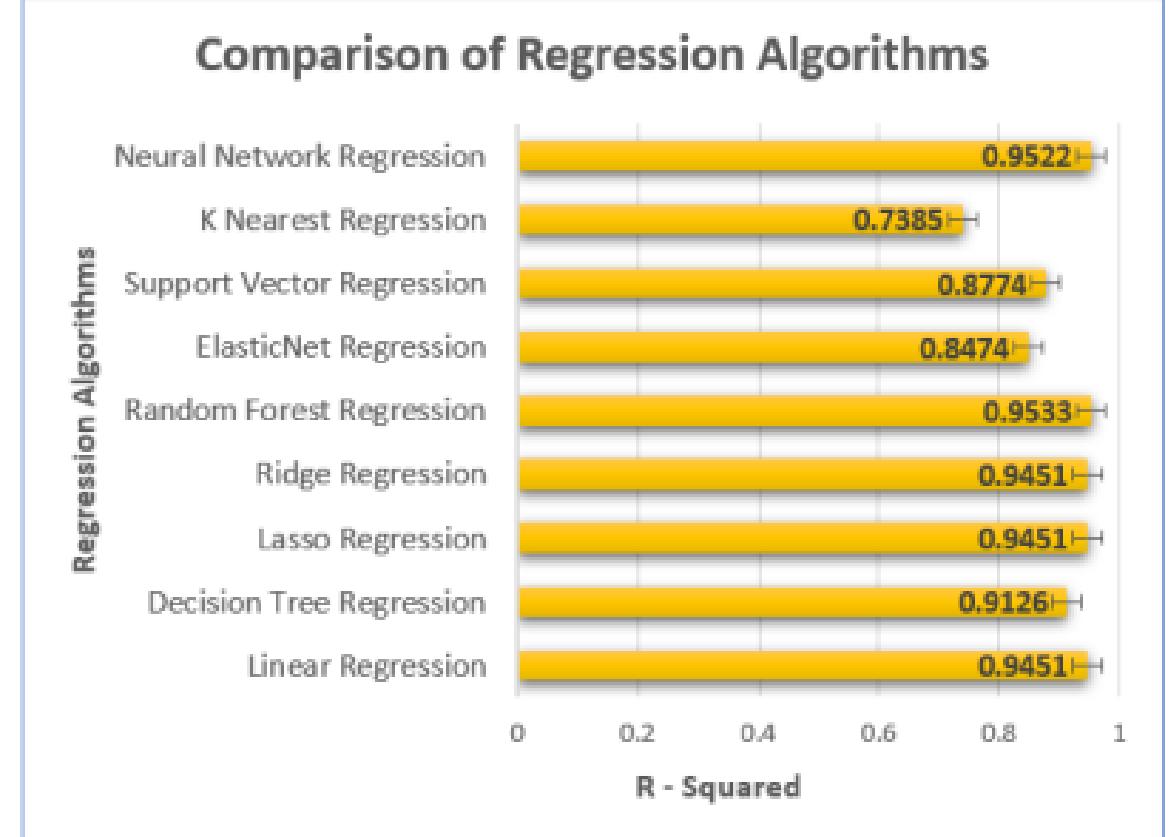


Fig. 8. Comparison of  $R^2$  value of regression algorithms





## Conclusion and Future Work

In conclusion, the comparison of model accuracies in the paper highlights the superior performance of complex boosting models like Gradient Boosting and XG Boost with tuned hyperparameters in testing data.

However, it is important to note that Random Forest Regression has the highest training accuracy, indicating potential overfitting due to an unspecified number of stumps.

In future work, it would be interesting to explore techniques to mitigate overfitting in Random Forest Regression while further improving the performance of complex boosting models.

There could also be improvement in data collection and attributes taken. Also the boosting algorithms could be implemented from scratch to have better results.