Medical Insurance Price Predictor

A Project of

INTRODUCTION TO MACHINE LEARNING

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

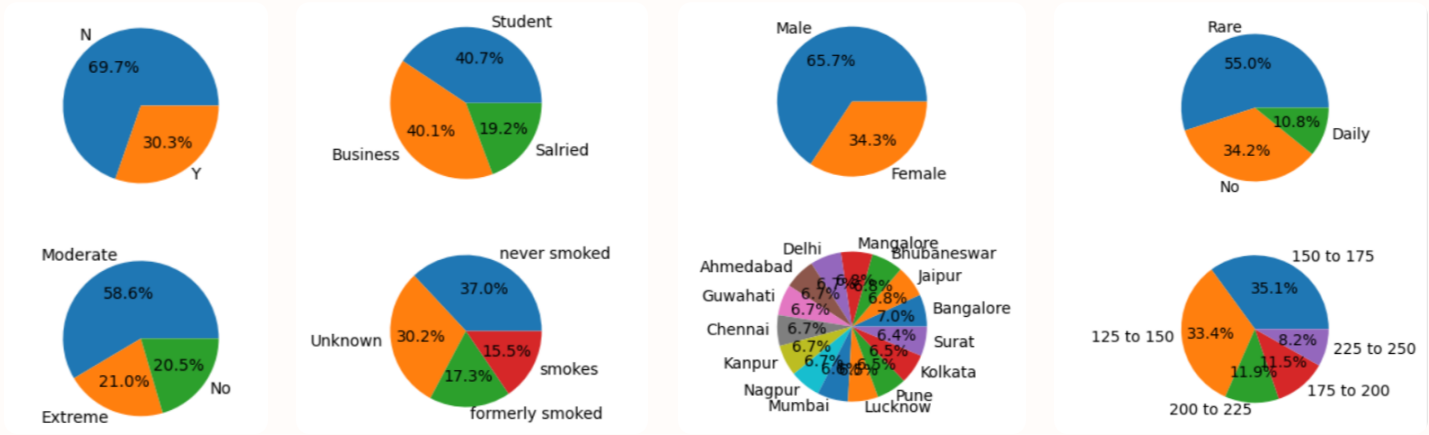
Dataset was downloaded from Kaggle (www.kaggle.com/datasets/gdeepakreddy/insurance).

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

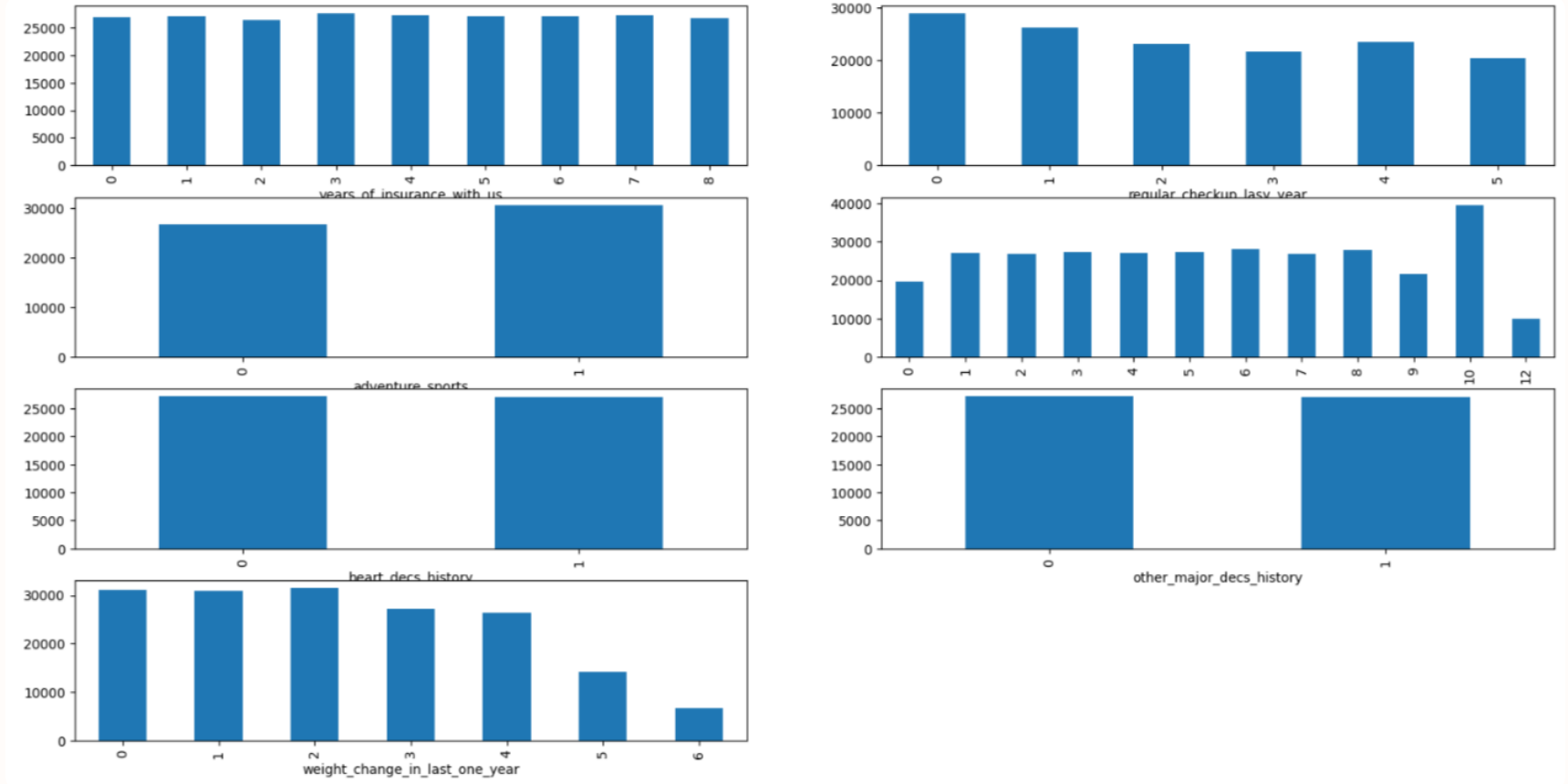
The insurance_cost is target variable with continuous values from 2468 to 67870. The decision is taken based on the values of the independent features.

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.colum

# 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'] <= u
```

- 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,'form
df['Location']=df['Location'].map({'Chennai':0, 'Jaipur':1,'Bangalore':2,'Guwahat
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'
df['Alcohol']=df['Alcohol'].map({'Rare':0, 'Daily':1,'No':2})
df['exercise']=df['exercise'].map({'Moderate':0, 'Extreme':1,'No':2})
```

- We have checked for presence of all null values using 'df.isnull().sum()' and then have imputed all the missing values with the mean value of attribute.
- We have then use the MinMaxScalar to bring all the data in the range of 0-1 to remove any possible bias towards the size of attribute values.

```
# Impute the missing values by m
missing_col = ['bmi','Year_last_
#Technique 1: Using mean to impu
for i in missing_col:
  df.loc[df.loc[:,i].isnull(),i]
```

```
df.isnull().sum()
```

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

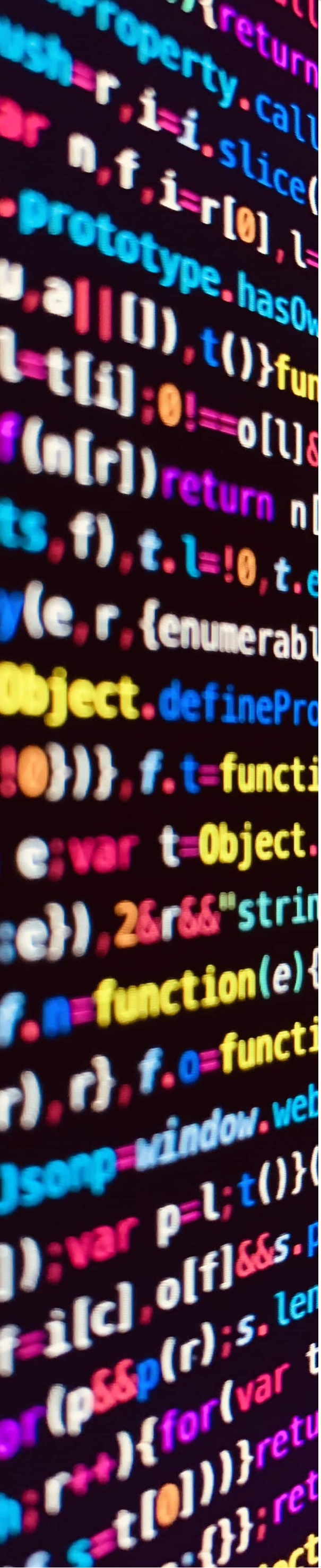
```
# Dropping the atributes with least significanc
df.drop(df[['Occupation','cholesterol_level','G
```

```
# 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())
```

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())
```

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())
```

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())
```

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_)
```

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())
```

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())
```

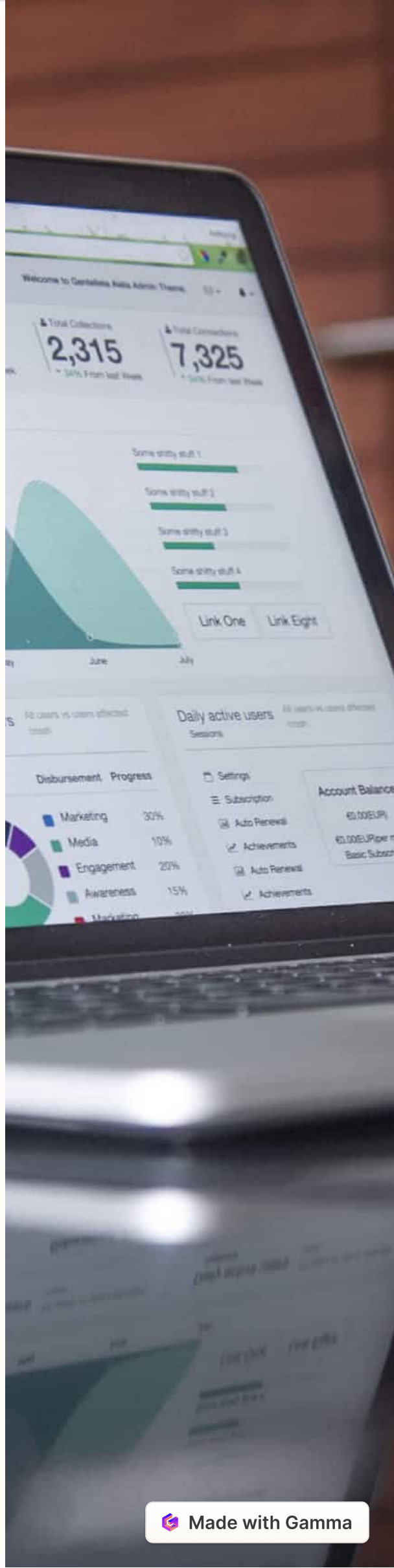
+ Code + Text

```
# 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_)
```

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	09447
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

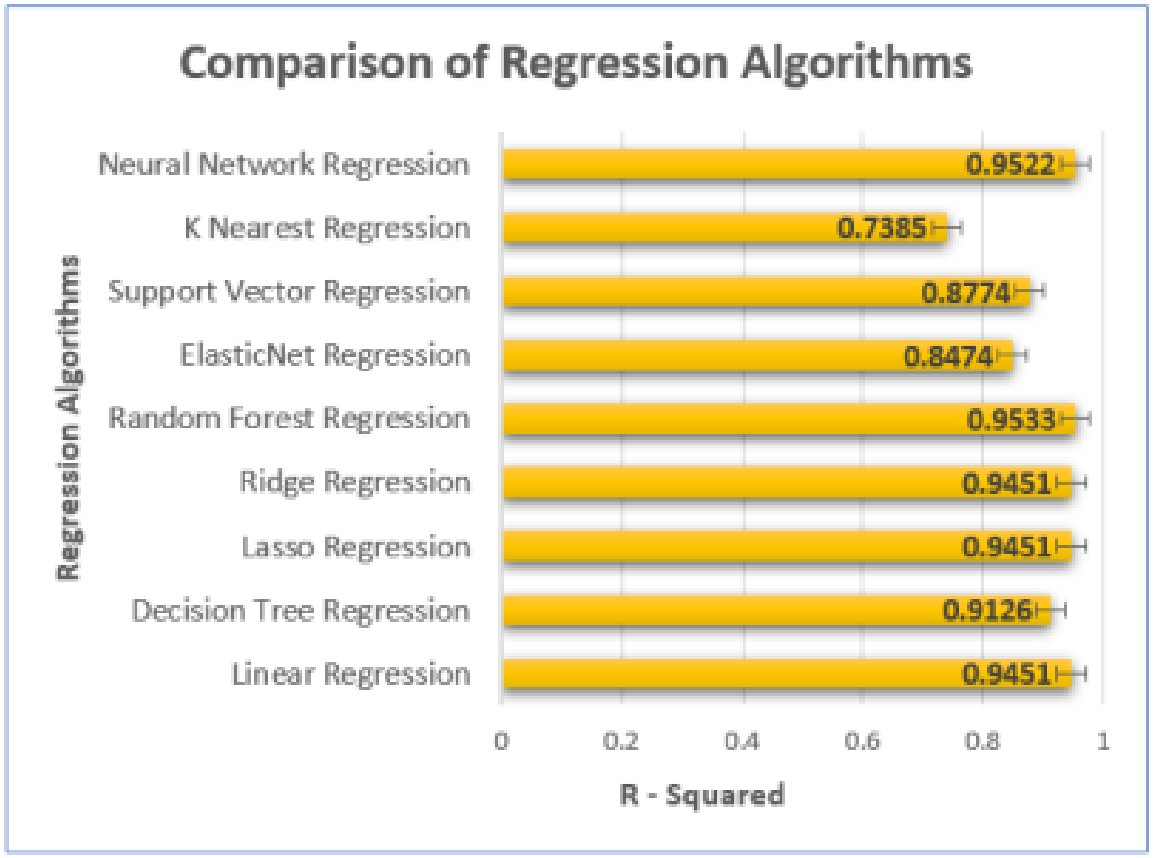


Fig. 8. Comparison of R^2 value of regression algorithms

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.