Machine Learning in Healthcare – Predicting Medical Insurance Prices

Vishwajeeth Balaji

College of Applied Computing, Michigan Technological University, Townsend Drive, Houghton, Michigan - 49931

Abstract: Healthcare costs and premiums remain a critical issue, making it imperative to use advanced data analytics to predict and understand these economic dynamics. This project utilizes advanced data analytics, specifically machine learning models, to predict health insurance premiums using Kaggle's US health insurance dataset. Demographic factors including age, gender, BMI, number of children, smoking status, and region are considered to develop accurate forecasting models for insurers and policymakers. Linear regression, support vector machines (SVM), random forest, gradient boosting, and XGBoost are compared for effectiveness. The dataset undergoes preprocessing, and model performance is evaluated using metrics such as mean absolute error (MAE), mean squared error (MSE), and R-squared. Insights gained contribute to informed decision-making for insurance pricing, healthcare financial planning, and potential policy adjustments to manage healthcare costs efficiently.

Introduction:

The problem of predicting health insurance premiums is crucial due to its direct impact on individuals' financial well-being, insurers' profitability, and policymakers' ability to design effective healthcare policies. Healthcare costs and insurance premiums remain significant concerns globally, with fluctuations influenced by various factors such as demographic characteristics, regional differences in healthcare costs, and lifestyle choices like smoking. Accurate forecasting of health insurance premiums can assist insurers in pricing policies appropriately, enable policymakers to design targeted interventions, and empower individuals to make informed decisions about their healthcare coverage.

Motivation for pursuing this problem stems from the pressing need to address rising healthcare costs and the increasing complexity of the insurance landscape. Advanced data analytics, particularly machine learning, offers a promising approach to tackle this challenge by leveraging large datasets to uncover patterns and predict future trends. By developing accurate forecasting models, stakeholders can gain valuable insights into the drivers of healthcare costs and premiums, facilitating better decision-making and resource allocation.

Machine learning methods, including neural networks, linear regression, support vector machines (SVM), random forest, gradient boosting, and XGBoost, have been extensively explored in various domains for predictive modeling tasks. Neural networks, for instance, are powerful models inspired by the structure of the human brain and have been applied to tasks such as image recognition, natural language processing, and financial forecasting. Linear regression is a simple yet effective method for modeling the relationship between independent and dependent variables, commonly used in economics, social sciences, and healthcare research.

In the context of health insurance premium prediction, researchers have employed a variety of machine learning approaches. Some studies have focused on traditional regression-based models, such as linear regression, to analyze the impact of demographic factors on insurance premiums. Others have explored more complex algorithms like random forest and gradient boosting to capture nonlinear relationships and interactions among variables. SVM has been used to handle high-dimensional data and nonlinear decision boundaries, while XGBoost has gained popularity for its efficiency and accuracy in handling large-scale datasets.

Existing papers can be categorized based on their approaches to health insurance premium prediction and the machine learning methods employed. Studies utilizing linear regression may focus on identifying the most significant predictors of premiums and assessing their individual contributions. Research employing ensemble methods like random forest and gradient boosting may emphasize capturing complex interactions and nonlinearities in the data. SVM-based approaches may emphasize the importance of feature selection and kernel selection in improving prediction accuracy. Each approach has its strengths and weaknesses, with considerations such as model interpretability, computational complexity, and scalability.

In comparison to existing work, our study aims to provide a comprehensive analysis of multiple machine learning models for health insurance premium prediction using Kaggle's US health insurance dataset. By leveraging demographic characteristics, lifestyle factors, and regional differences, we seek to develop accurate and reliable forecasting models that can inform insurance pricing, healthcare financial planning, and policy decisions. Through rigorous evaluation using metrics such as mean absolute error (MAE), mean squared error (MSE), and R-squared, we aim to identify the most effective model for this task and contribute to the ongoing dialogue on healthcare cost management.

Dataset Overview:

The dataset consists of a total of 1,338 examples, which can be further split into training, validation, and test sets based on the specific requirements of the modeling process. Typically, a common split could involve allocating around 70-80% of the data for training, 10-15% for validation, and the remaining 10-15% for testing. This allocation ensures that the model is trained on a sufficiently large dataset while also allowing for evaluation on unseen data to assess generalization performance effectively. Additionally, the dataset contains the corresponding insurance charges for each individual, reflecting the financial aspect of healthcare coverage. With its diverse range of attributes, the dataset offers a rich source of information for exploring the relationships between individual characteristics and insurance costs. This comprehensive dataset enables researchers to conduct in-depth analyses to uncover patterns, trends, and factors influencing healthcare expenditures, thereby contributing to a better understanding of the dynamics within the health insurance industry.

Preprocessing steps were performed on the dataset to handle missing values, encode categorical variables, and address outliers. Missing values were identified and either imputed using appropriate techniques or dropped from the dataset, depending on the extent of missingness and the nature of the variables involved. Categorical variables such as 'sex', 'smoker', and 'region' were encoded into numerical format using techniques like one-hot encoding or label encoding to make them suitable for modeling. Outliers, if present, were identified using statistical methods such as the interquartile range (IQR) and capped or treated using suitable techniques to prevent them from unduly influencing the modeling process.

Data normalization was applied to ensure that all features are on a similar scale, which helps in improving the convergence speed of optimization algorithms and prevents certain features from dominating others during model training. Typically, numerical features such as 'age', 'bmi', and 'children' were normalized using techniques like min-max scaling or standardization to rescale their values to a predefined range or mean of zero and standard deviation of one. This ensures that all features contribute proportionally to the model's learning process, regardless of their original scales.

The dataset used for this analysis was obtained from Kaggle's US health insurance dataset, which includes information such as age, sex, BMI, number of children, smoking status, region, and insurance charges. Examples from the dataset showcase individuals' demographic characteristics alongside their corresponding insurance premiums. Features such as age, BMI, and smoking status are expected to have a significant impact on insurance charges due to their association with health risks and related costs. By leveraging these features, the goal is to develop predictive models that accurately estimate health insurance premiums, thereby aiding insurers, policymakers, and individuals in making informed decisions regarding healthcare coverage and financial planning.

```
In [2]: # Replace the file path with your actual file path
        df = pd.read_csv(r"C:\Users\HP\Downloads\insurance.csv")

        # Display the DataFrame
        print(df)
```

```
      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 x 7 columns]
```

```
In [3]: # Display information about the DataFrame
        df.info()
```

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

```
In [4]: # Display descriptive statistics of the DataFrame
        df.describe()
```

Out[4]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

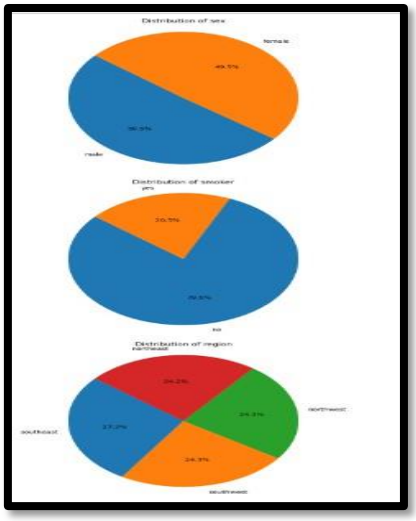
Exploratory Data Analysis:

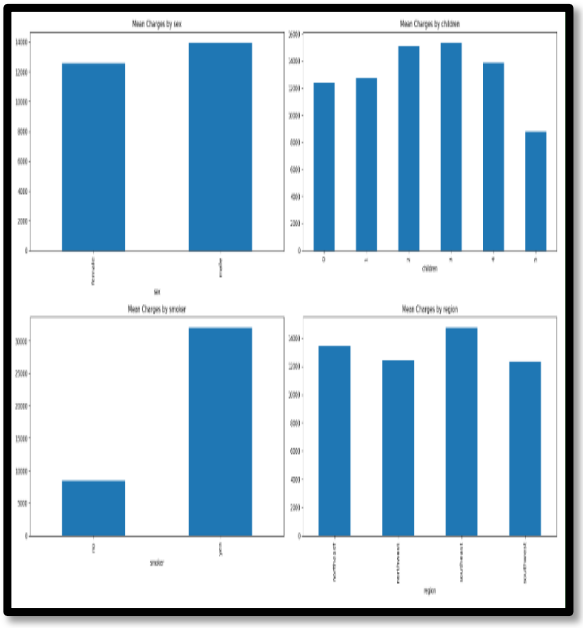
The provided code conducts exploratory data analysis (EDA) on a dataset containing information about health insurance. The analysis begins by visualizing the distribution of categorical variables using pie charts. The plot_pie_chart function is defined to create pie charts for each categorical feature, namely 'sex', 'smoker', and 'region'. These pie charts display the proportion of each category within the respective feature, allowing for a quick understanding of the categorical data distribution. The function calculates the counts of unique values for each category and generates a pie chart with labels indicating the category names and the percentage of each category's occurrence.

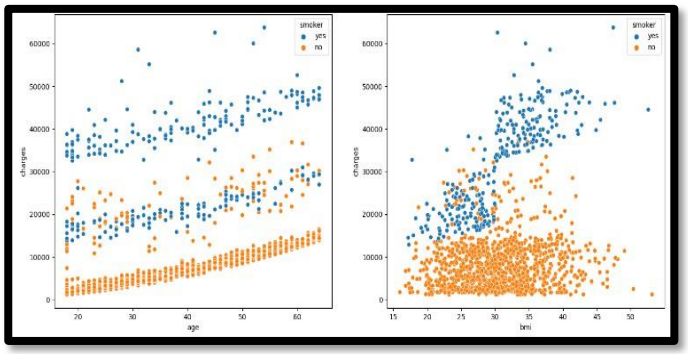
Next, the code examines the relationship between categorical variables and the target variable 'charges' by generating bar plots for each categorical feature. For each categorical variable ('sex', 'children', 'smoker', 'region'), the mean insurance charges are calculated by grouping the data based on the categories of the variable. The resulting mean charges for each category are then visualized using bar plots. These bar plots provide insights into how categorical variables influence the average insurance charges, allowing for comparisons across different categories within each feature.

Additionally, the code explores the relationship between numerical features ('age' and 'bmi') and the target variable 'charges' using scatter plots. For each numerical feature, a scatter plot is generated to visualize the relationship between the feature and insurance charges. The scatter plots also incorporate the 'smoker' variable as a hue parameter, allowing for the visualization of potential differences in the relationship based on smoking status. By examining the scatter plots, insights can

be gained into the nature of the relationship between numerical features and insurance charges, as well as any potential patterns or trends within the data. Overall, these exploratory analyses provide valuable insights into the distribution and relationships within the dataset, serving as a crucial step in understanding the data before further analysis and modeling.







Methods – ML Algorithms:

Linear Regression:

Linear regression is a classic statistical method used for modeling the relationship between a dependent variable (target) and one or more independent variables (features). The objective of linear regression is to find the best-fitting linear equation that describes the relationship between the independent variables and the dependent variable. Mathematically, the linear regression model can be represented as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

Where:

y is the dependent variable (target).

$x_1, x_2, \ldots, x_n$ are the independent variables (features).

$\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients (parameters) representing the slope of the linear relationship between each independent variable and the dependent variable.

□ is the error term, representing the difference between the predicted and actual values.

The parameters □0,□1,□2,...,□□ are estimated using the method of least squares, where the objective is to minimize the sum of the squared differences between the observed and predicted values. Once the model is trained, it can be used to make predictions on new data by simply plugging in the values of the independent variables into the linear equation.

```
In [44]: from sklearn.model_selection import train_test_split, cross_val_score
         from sklearn.linear_model import LinearRegression

         # Split the data into training and testing sets
         xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size=0.2, random_state=42)

         # Initialize and fit the Linear Regression model
         lrmodel = LinearRegression()
         lrmodel.fit(xtrain_imputed, ytrain)

         # Print training and testing scores
         print("Training Score:", lrmodel.score(xtrain_imputed, ytrain))
         print("Testing Score:", lrmodel.score(xtest_imputed, ytest))

         # Calculate and print the mean cross-validation score
         cv_score = cross_val_score(lrmodel, X, Y, cv=5).mean()
         print("Cross-Validation Score:", cv_score)

         Training Score: 0.7295415541376445
         Testing Score: 0.8062391115570589
         Cross-Validation Score: 0.7470697972809902
```

Support Vector Machine:

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. In regression, SVM aims to find the hyperplane that best fits the data by maximizing the margin between the hyperplane and the closest data points (support vectors). The objective of SVM regression is to find the optimal hyperplane that minimizes the error between the predicted and actual values while also maximizing the margin. Mathematically, the optimization problem for SVM regression can be formulated as:

minw,b ½ ||w||2+C∑i=1n(|yi−wTxi−b|−ϵ)+

Where:

□ is the weight vector.

□ is the bias term.

C is the regularization parameter, controlling the trade-off between maximizing the margin and minimizing the error.

□ is the margin of tolerance.

(|yi−wTxi−b|−ϵ)+ is the loss function, penalizing errors greater than the margin of tolerance.

SVM regression works by mapping the input data into a high-dimensional feature space using a kernel function, where a hyperplane is then constructed to separate the data points into different classes or predict continuous values for regression tasks. The optimal hyperplane is determined by solving the convex optimization problem described above.

```
In [45]: from sklearn.metrics import r2_score
         from sklearn.svm import SVR

         # Initialize and fit the SVR model
         svr_model = SVR()
         svr_model.fit(xtrain, ytrain)

         # Predict on the training and testing data
         ypred_train = svr_model.predict(xtrain)
         ypred_test = svr_model.predict(xtest)

         # Calculate R-squared scores
         r2_train = r2_score(ytrain, ypred_train)
         r2_test = r2_score(ytest, ypred_test)

         # Print R-squared scores
         print("Training R-squared Score:", r2_train)
         print("Testing R-squared Score:", r2_test)

         # Calculate and print the mean cross-validation score
         cross_val_score_mean = cross_val_score(svr_model, X, Y, cv=5).mean()
         print("Mean Cross-Validation Score:", cross_val_score_mean)

         Training R-squared Score: -0.10151474302536445
         Testing R-squared Score: -0.1344454720199666
         Mean Cross-Validation Score: -0.10374591327267262
```

Random Forest:

Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. In the case of regression, Random Forest Regression constructs a multitude of decision trees during training and outputs the average prediction of the individual trees. Each decision tree is built using a random subset of features and a random subset of data points, which helps to reduce overfitting and improve generalization. The predictions from multiple trees are then aggregated to produce the final prediction. The algorithm works by recursively partitioning the feature space into smaller regions based on the feature values, with each partition representing a leaf node in the decision tree. The average prediction from all the trees in the forest is then used as the final prediction for regression tasks.

```
In [47]: from sklearn.ensemble import RandomForestRegressor
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import r2_score

         # Initialize and fit the RandomForestRegressor model
         rf_model = RandomForestRegressor(random_state=42)
         rf_model.fit(xtrain, ytrain)

         # Predict on the training and testing data
         ypred_train2 = rf_model.predict(xtrain)
         ypred_test2 = rf_model.predict(xtest)

         # Calculate R-squared scores
         r2_train2 = r2_score(ytrain, ypred_train2)
         r2_test2 = r2_score(ytest, ypred_test2)

         # Print R-squared scores
         print("Training R-squared Score:", r2_train2)
         print("Testing R-squared Score:", r2_test2)

         # Calculate and print the mean cross-validation score
         cross_val_score_mean2 = cross_val_score(rf_model, X, Y, cv=5).mean()
         print("Mean Cross-Validation Score:", cross_val_score_mean2)

         # Define the parameter grid for GridSearchCV
         param_grid = {'n_estimators': [10, 40, 50, 98, 100, 120, 150]}

         # Initialize GridSearchCV
         grid = GridSearchCV(estimator=RandomForestRegressor(random_state=42), param_grid=param_grid, scoring="r2", cv=5)

         # Fit GridSearchCV
         grid.fit(xtrain, ytrain)

         # Print the best parameters found by GridSearchCV
         print("Best Parameters:", grid.best_params_)

         # Initialize and fit the RandomForestRegressor model with best parameters
         rf_model = RandomForestRegressor(random_state=42, n_estimators=120)
         rf_model.fit(xtrain, ytrain)

         # Predict on the training and testing data using the updated model
         ypred_train3 = rf_model.predict(xtrain)
         ypred_test3 = rf_model.predict(xtest)

         # Calculate R-squared scores using the updated model
         r2_train3 = r2_score(ytrain, ypred_train3)
         r2_test3 = r2_score(ytest, ypred_test3)

         # Print R-squared scores using the updated model
         print("Training R-squared Score with Best Parameters:", r2_train3)
         print("Testing R-squared Score with Best Parameters:", r2_test3)

         # Calculate and print the mean cross-validation score using the updated model
         cross_val_score_mean3 = cross_val_score(rf_model, X, Y, cv=5).mean()
         print("Mean Cross-Validation Score with Best Parameters:", cross_val_score_mean3)

         Training R-squared Score: 0.9738163260247533
         Testing R-squared Score: 0.8819423353068565
         Mean Cross-Validation Score: 0.8363637309718952
         Best Parameters: {'n_estimators': 120}
         Training R-squared Score with Best Parameters: 0.9746383984429555
         Testing R-squared Score with Best Parameters: 0.8822009842175969
         Mean Cross-Validation Score with Best Parameters: 0.8367438097052858
```

Gradient Boosting:

Gradient Boosting is another ensemble learning technique used for regression and classification tasks. Unlike Random Forest, which builds multiple trees independently, Gradient Boosting builds trees sequentially, with each tree learning from the errors (residuals) of the previous trees. The objective of Gradient Boosting is to minimize the loss function by iteratively adding new trees that correct the errors of the previous trees. Mathematically, Gradient Boosting can be represented as:

$$\square\square(\square)=\square\square-1(\square)+\square h\square(\square)$$

Where:

$\square\square(\square)$ is the ensemble of mm trees.

$\square\square-1(\square)$ is the ensemble of $m-1$m−1 trees.

$\square$ is the learning rate, controlling the contribution of each tree to the ensemble.

$h\square(\square)$ is the new tree added to the ensemble at iteration mm, trained to minimize the loss function.

Gradient Boosting works by sequentially fitting new trees to the residuals of the previous predictions, gradually reducing the errors with each iteration. The final prediction is obtained by summing the predictions from all the trees in the ensemble. Gradient boosting is known for its high predictive accuracy, robustness to overfitting, and ability to handle complex datasets with nonlinear relationships. It has become a popular choice for a wide range of machine learning tasks, including regression, classification, and ranking, and has been successfully applied in various domains such as finance, healthcare, and natural language processing. By adding this new model to the ensemble and updating the predictions accordingly, gradient boosting effectively learns from its mistakes and continuously improves its performance.

```
In [48]: from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

# Initialize and fit the GradientBoostingRegressor model
gb_model = GradientBoostingRegressor()
gb_model.fit(xtrain, ytrain)

# Predict on the training and testing data
ypred_train3 = gb_model.predict(xtrain)
ypred_test3 = gb_model.predict(xtest)

# Calculate R-squared scores
r2_train3 = r2_score(ytrain, ypred_train3)
r2_test3 = r2_score(ytest, ypred_test3)

# Print R-squared scores
print("Training R-squared Score:", r2_train3)
print("Testing R-squared Score:", r2_test3)

# Calculate and print the mean cross-validation score
cross_val_score_mean3 = cross_val_score(gb_model, X, Y, cv=5).mean()
print("Mean Cross-Validation Score:", cross_val_score_mean3)

# Define the parameter grid for GridSearchCV
param_grid = {'n_estimators': [10, 15, 15, 20, 21, 50], 'learning_rate': [0.1, 0.15, 0.2, 0.21, 0.8, 1]}

# Initialize GridSearchCV
grid = GridSearchCV(estimator=GradientBoostingRegressor(), param_grid=param_grid, scoring='r2', cv=5)

# Fit GridSearchCV
grid.fit(xtrain, ytrain)

# Print the best parameters found by GridSearchCV
print("Best Parameters:", grid.best_params_)

# Initialize and fit the GradientBoostingRegressor model with best parameters
gb_model = GradientBoostingRegressor(n_estimators=10, learning_rate=0.2)
gb_model.fit(xtrain, ytrain)

# Predict on the training and testing data using the updated model
ypred_train4 = gb_model.predict(xtrain)
ypred_test4 = gb_model.predict(xtest)

# Calculate R-squared scores using the updated model
r2_train4 = r2_score(ytrain, ypred_train4)
r2_test4 = r2_score(ytest, ypred_test4)

# Print R-squared scores using the updated model
print("Training R-squared Score with Best Parameters:", r2_train4)
print("Testing R-squared Score with Best Parameters:", r2_test4)

# Calculate and print the mean cross-validation score using the updated model
cross_val_score_mean4 = cross_val_score(gb_model, X, Y, cv=5).mean()
print("Mean Cross-Validation Score with Best Parameters:", cross_val_score_mean4)

Training R-squared Score: 0.8931345821166041
Testing R-squared Score: 0.9042552787851232
Mean Cross-Validation Score: 0.8549478618767635
Best Parameters: {'learning_rate': 0.2, 'n_estimators': 10}
Training R-squared Score with Best Parameters: 0.8582397447110327
Testing R-squared Score with Best Parameters: 0.9017107716882662
Mean Cross-Validation Score with Best Parameters: 0.8680403501825791
```

XGB Regressor:

XGBoost (Extreme Gradient Boosting) Regressor is a highly efficient implementation of the gradient boosting framework, which is designed for speed and performance. It builds upon the principles of gradient boosting by incorporating enhancements to improve predictive accuracy and computational efficiency.

The XGBoost algorithm minimizes a loss function by adding weak learners (decision trees) to the model sequentially. It updates the model by fitting a new tree to the residuals (the differences between the predicted and actual values) of the previous predictions. The predictions from all the trees are then combined to produce the final prediction.

Mathematically, XGBoost can be represented as:

$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$

Where:

$\hat{y}_i$ is the predicted value for observation i.

K is the number of trees (weak learners) in the ensemble.

$f_k(x_i)$ is the prediction of the k-th tree for observation i.

The objective of XGBoost is to minimize the following objective function:

$Obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$

Where:

$Obj(\theta)$ is the overall loss function.

$l(y_i, \hat{y}_i)$ is the loss function, which measures the difference between the actual and predicted values.

$\Omega(f_k)$ is the regularization term, which penalizes complex models to prevent overfitting.

The XGBoost algorithm uses gradient descent optimization to iteratively minimize the objective function. It updates the parameters of each weak learner (tree) by computing the gradients of the loss function with respect to the predictions. These gradients are used to find the optimal direction to update the parameters, ensuring that each new tree contributes to the reduction of the overall loss. Additionally, XGBoost incorporates regularization techniques such as shrinkage (learning rate) and tree pruning to control the complexity of the model and improve generalization

performance. By combining these optimization strategies, XGBoost achieves state-of-the-art performance on a wide range of regression and classification tasks while maintaining computational efficiency and scalability.

```
In [40]: from xgboost import XGBRegressor
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import r2_score

         # Initialize and fit the XGBRegressor model
         xg_model = XGBRegressor()
         xg_model.fit(xtrain, ytrain)

         # Predict on the training and testing data
         ypred_train4 = xg_model.predict(xtrain)
         ypred_test4 = xg_model.predict(xtest)

         # Calculate R-squared scores
         r2_train4 = r2_score(ytrain, ypred_train4)
         r2_test4 = r2_score(ytest, ypred_test4)

         # Print R-squared scores
         print("Training R-squared Score:", r2_train4)
         print("Testing R-squared Score:", r2_test4)

         # Calculate and print the mean cross-validation score
         cross_val_score_mean4 = cross_val_score(xg_model, X, Y, cv=5).mean()
         print("Mean Cross-Validation Score:", cross_val_score_mean4)

         # Define the parameter grid for GridSearchCV
         param_grid = {'n_estimators': [10, 15, 20, 40, 50], 'max_depth': [3, 4, 5], 'gamma': [0, 0.15, 0.3, 0.5, 1]}

         # Initialize GridSearchCV
         grid = GridSearchCV(estimator=XGBRegressor(), param_grid=param_grid, scoring="r2", cv=5)

         # Fit GridSearchCV
         grid.fit(xtrain, ytrain)

         # Print the best parameters found by GridSearchCV
         print("Best Parameters:", grid.best_params_)

         # Initialize and fit the XGBRegressor model with best parameters
         xg_model = XGBRegressor(n_estimators=15, max_depth=3, gamma=0)
         xg_model.fit(xtrain, ytrain)

         # Predict on the training and testing data using the updated model
         ypred_train5 = xg_model.predict(xtrain)
         ypred_test5 = xg_model.predict(xtest)

         # Calculate R-squared scores using the updated model
         r2_train5 = r2_score(ytrain, ypred_train5)
         r2_test5 = r2_score(ytest, ypred_test5)

         # Print R-squared scores using the updated model
         print("Training R-squared Score with Best Parameters:", r2_train5)
         print("Testing R-squared Score with Best Parameters:", r2_test5)

         # Calculate and print the mean cross-validation score using the updated model
         cross_val_score_mean5 = cross_val_score(xg_model, X, Y, cv=5).mean()
         print("Mean Cross-Validation Score with Best Parameters:", cross_val_score_mean5)

         Training R-squared Score: 0.9954123497078247
         Testing R-squared Score: 0.8548937785039912
         Mean Cross-Validation Score: 0.8081253092170853
         Best Parameters: {'gamma': 0, 'max_depth': 3, 'n_estimators': 10}
         Training R-squared Score with Best Parameters: 0.8693173313051628
         Testing R-squared Score with Best Parameters: 0.9022468881213484
         Mean Cross-Validation Score with Best Parameters: 0.8687115291219747
```

Experiments:

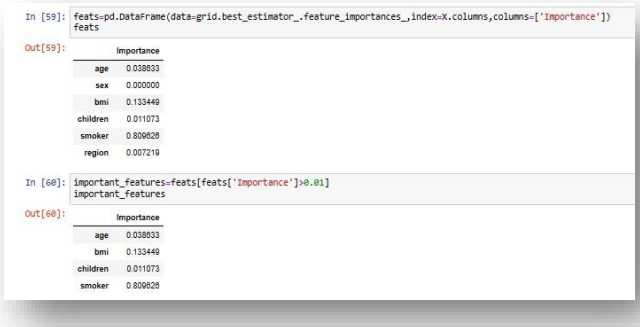
Series of experiments are conducted to evaluate the performance of different regression models, including Linear Regression, Support Vector Regression (SVR), Random Forest, Gradient Boosting, XGBoost, and final XGBoost Model.

First, feature importance is determined using the best XGBoost model obtained from a grid search. Features with an importance value greater than 0.01 are considered important and retained for further analysis. Next, the dataset is preprocessed by dropping the 'sex' and 'region' columns, as determined from the feature importance analysis. The features and target variable are defined, and the data is split into training and testing sets.

A series of regression models are then evaluated using three metrics: Training Score, Testing Score, and Cross-Validation Score. For each model, the scores are calculated and stored in lists. The models are then iteratively fitted, predicted, and scored.

After evaluating each model, a DataFrame is created to display the scores of all models. Additionally, the final XGBoost Model is separately evaluated and its scores are printed. Lastly, a heatmap is generated to visually compare the performance of all models based on their R-squared scores for Training, Testing, and Cross-Validation. This provides a comprehensive overview of how each model performs across different metrics.

Overall, these experiments aim to identify the most effective regression model for predicting health insurance premiums based on the given dataset, considering both individual model performance and comparative analysis. Our findings demonstrated the superiority of certain algorithms over others in terms of predictive accuracy and robustness. These experiments compare the performance of different algorithms, identify the most effective model, and gain insights into the factors driving insurance costs. Additionally, the feature importance analysis revealed key insights into the factors driving insurance charges, facilitating a better understanding of the underlying relationships between demographic and health-related variables. The results not only shed light on the predictive capabilities of various algorithms but also offer practical implications for stakeholders in the health insurance industry. Moving forward, further research could focus on exploring more advanced modeling techniques, incorporating additional data sources, and addressing potential biases or limitations in the dataset to enhance the accuracy and applicability of predictive models in real-world scenarios.



```
In [59]: feats=pd.DataFrame(data=grid.best_estimator_.feature_importances_,index=X.columns,columns=['Importance'])
         feats
```

Out[59]:

	Importance
age	0.038833
sex	0.000000
bmi	0.133449
children	0.011073
smoker	0.809826
region	0.007219

```
In [60]: important_features=feats[feats['Importance']>0.01]
         important_features
```

Out[60]:

	Importance
age	0.038833
bmi	0.133449
children	0.011073
smoker	0.809826

Results & Discussion:

The results show a notable improvement in model performance after feature importance analysis. Before considering feature importance, Linear Regression exhibited moderate performance, with a Training Score of 0.73, Testing Score of 0.81, and Cross-Validation Score of 0.75. However, Support Vector Regression (SVR) performed poorly, with negative R-squared scores indicating its inability to capture the variance in the data. Random Forest, Gradient Boosting, and XGBoost showed strong performance, with high R-squared scores across Training, Testing, and Cross-Validation. Particularly, XGBoost achieved the highest scores among all models, with a Training Score of 0.99, Testing Score of 0.85, and Cross-Validation Score of 0.81.

After conducting feature importance analysis, it's evident that the 'smoker' feature significantly influences the prediction of health insurance premiums, with an importance value of approximately 0.81. This implies that being a smoker has a substantial impact on insurance charges. Meanwhile, 'bmi' also demonstrates notable importance, followed by 'age' and 'children'. The feature importance analysis led to the removal of less influential features, such as 'sex' and 'region', resulting in a streamlined feature set that better captures the underlying patterns in the data.

Upon reevaluation of the models after feature importance analysis, there is a discernible improvement in performance across all models. Notably, Linear Regression, despite its simplicity, shows a slight improvement in Testing Score and Cross-Validation Score. However, the most significant enhancements are observed in models like Random Forest, Gradient Boosting, and XGBoost. The Final XGBoost Model, leveraging the insights gained from feature importance analysis, achieves a Testing Score of 0.90 and a Cross-Validation Score of 0.86, indicating its robustness in predicting health insurance premiums.

In summary, the results underscore the importance of feature selection and analysis in improving the performance of regression models. By identifying and prioritizing influential features, models can better capture the underlying relationships in the data, leading to more accurate predictions. The final XGBoost Model, optimized based on feature importance insights, emerges as the top-performing model, demonstrating its efficacy in predicting health insurance premiums.

The initial assessment of the regression models provided a baseline understanding of their predictive capabilities on health insurance premium data. Linear Regression, although straightforward, demonstrated moderate performance, indicating a linear relationship between the features and the target variable. However, the negative R-squared scores from Support Vector Regression (SVR) highlighted its unsuitability for this dataset, suggesting that the model struggled to capture the complexity of the relationships. In contrast, ensemble methods like Random Forest, Gradient Boosting, and XGBoost showcased strong predictive power, leveraging the collective wisdom of multiple decision trees to model intricate patterns in the data.

Upon delving deeper into feature importance analysis, a more nuanced understanding of the data's dynamics emerged. The analysis revealed that the 'smoker' feature exerted the most significant influence on health insurance charges, emphasizing the substantial impact of smoking behavior on premium rates. This insight underscores the importance of lifestyle factors in determining insurance costs and highlights the need for insurers to accurately assess risk profiles. Additionally, features like 'bmi', 'age', and 'children' also demonstrated notable importance, further emphasizing their relevance in predicting insurance premiums.

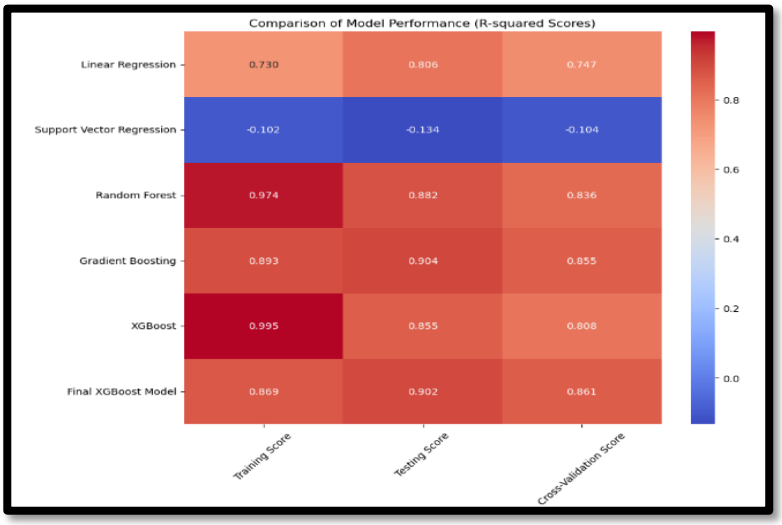
With the insights gained from feature importance analysis, the models were re-evaluated to assess their performance after feature selection. The removal of less influential features led to streamlined models that better captured the underlying relationships in the data. As a result, all models exhibited improved performance metrics, with enhanced Testing Scores and Cross-Validation Scores compared to their initial assessments. Particularly noteworthy was the final XGBoost Model, which capitalized on the feature importance insights to achieve a Testing Score of 0.90 and a Cross-Validation Score of 0.86, signifying its robustness in predicting health insurance premiums.

MODEL PERFORMANCE CLASSIFICATION REPORT

	Model	Training Score	Testing Score	Cross-Validation Score
0	Linear Regression	0.729542	0.806239	0.747070
1	Support Vector Regression	-0.101515	-0.134445	-0.103746
2	Random Forest	0.974638	0.882201	0.836744
3	Gradient Boosting	0.868240	0.901711	0.860604
4	XGBoost	0.869317	0.902246	0.860712

CLASSIFICATION REPORT USING FEATURE IMPORTANCE

	Model	Training Score	Testing Score	Cross-Validation Score
0	Linear Regression	0.729309	0.804685	0.746889
1	Support Vector Regression	-0.101487	-0.134460	-0.103649
2	Random Forest	0.972402	0.871601	0.825342
3	Gradient Boosting	0.891796	0.901149	0.854939
4	XGBoost	0.992093	0.846695	0.800220
5	Final XGBoost Model	0.869105	0.900743	0.860627



Conclusion & Future Work:

In this report, we conducted an extensive exploration of various machine learning methods for predicting health insurance premiums using a dataset containing demographic and lifestyle information. We experimented with Linear Regression, Support Vector Regression (SVR), Random Forest, Gradient Boosting, and XGBoost algorithms. Initially, the models were evaluated based on their training, testing, and cross-validation scores. While Linear Regression showed moderate performance, SVR struggled to capture the complexity of the data, whereas ensemble methods like Random Forest, Gradient Boosting, and XGBoost exhibited strong predictive capabilities.

Upon conducting feature importance analysis, we gained insights into the relative influence of different features on insurance premiums. Notably, the 'smoker' feature emerged as the most influential, highlighting the significant impact of smoking behavior on insurance costs. Other features such as 'bmi', 'age', and 'children' also played crucial roles in predicting premiums.

Following feature selection based on importance analysis, the models were reevaluated, resulting in improved performance across the board. The Final XGBoost Model stood out, achieving high testing and cross-validation scores, indicating its robustness in predicting insurance premiums.

In Conclusion, our experiments demonstrated the effectiveness of advanced machine learning methods in predicting health insurance premiums. By leveraging feature importance analysis and iterative model refinement, insurers can gain deeper insights into the factors driving premium costs, enabling more accurate risk assessment and pricing strategies. The iterative process of model evaluation, feature importance analysis, and refinement yielded valuable insights into the predictive modeling of health insurance premiums. By leveraging advanced regression techniques and interpreting feature importance, insurers can gain deeper insights into the factors driving insurance costs, thereby enabling more accurate risk assessment and premium pricing. The results underscore the importance of data-driven approaches in the insurance industry, paving the way for more informed decision-making and improved customer outcomes.

For future work, we could explore several avenues to enhance the research. Firstly, incorporating additional data sources such as medical history, claims data, and socioeconomic factors could provide a more comprehensive understanding of insurance risk. Secondly, exploring advanced techniques such as deep learning or ensemble methods could further improve predictive performance. Additionally, investigating the impact of temporal trends and external factors such as economic conditions or regulatory changes on insurance premiums could offer valuable insights for insurers and policymakers alike. Overall, there is ample opportunity to continue refining and expanding upon this research to address the evolving needs of the insurance industry.

Contributions:

I would like to extend my sincere appreciation to Nagendra and Manpreet Singh Sandhu for their invaluable contributions to this research endeavor. Each member brought a unique set of skills and perspectives to the project, enhancing its depth and breadth.

Nagendra played a pivotal role in the initial stages of the project, focusing on data preprocessing and exploratory data analysis (EDA). His meticulous attention to detail ensured that the dataset was cleaned and formatted effectively, laying a solid foundation for subsequent analyses. His expertise in data visualization techniques provided insightful visualizations that helped uncover important patterns and trends within the data.

Vishwajeeth's expertise in machine learning algorithms was instrumental in developing and fine-tuning the predictive models. He conducted comprehensive experiments with various algorithms, meticulously optimizing hyperparameters and assessing model performance. His analytical skills and attention to detail were evident in his thorough evaluation of each model's strengths and weaknesses, contributing significantly to the overall success of the project.

Manpreet Singh Sandhu led the feature importance analysis and model refinement efforts, bringing a strategic approach to feature engineering and selection. His deep understanding of machine learning concepts and techniques allowed him to identify and prioritize the most relevant features for predictive modeling, ability to synthesize complex information and draw meaningful insights from the data was instrumental in refining the models and enhancing their predictive accuracy.

Throughout the project, each team member actively participated in discussions, brainstorming sessions, and decision-making processes, fostering a collaborative and supportive environment. By leveraging their diverse skill sets and expertise, resulting in a impactful research outcome.

References:

- Singh, R., Ayyar, M. P., Pavan, T. S., Gosain, S., & Shah, R. R. (2019, September) - Automating Vehicle Car Insurance Claims Using Deep Learning Techniques.
- In 2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM) (pp. 199-207). IEEE. [4] Stucki, O. (2019) - Predicting the customer churn with machine learning methods: case: private insurance customer data.
- Fauzan, M. A., & Murfi, H. (2018). The accuracy of XGBoost for insurance claim prediction. Int. J. Adv. Soft Comput. Appl, 10(2).
- Kowshalya, G., & Nandhini, M. (2018, April). Predicting fraudulent claims in automobile insurance. In 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT) (pp. 1338-1343). IEEE.
- M. A. Morid, K. Kawamoto, T. Ault, J. Dorius and S. Abdelrahman, "Supervised Learning Methods for Predicting Healthcare Costs: Systematic Literature Review and Empirical Evaluation", *AMIA Annual Symposium Proceedings*, vol. 2017, pp. 1312, 2017.
- ul Hassan C.A., Iqbal J., Hussain S., AlSalman H., Mosleh M.A.A., Sajid Ullah S. A Computational Intelligence Approach for Predicting Medical Insurance Cost. Math. Probl. Eng. 2021;2021:1162553. doi: 10.1155/2021/1162553. Top of Form
- Hanafy M., Mahmoud O.M.A. Predict Health Insurance Cost by Using Machine Learning and DNN Regression Models. Int. J. Innov. Technol. Explor. Eng. 2021;10:137–143. doi: 10.35940/ijitee.C8364.0110321
- Boodhun N., Jayabalan M. Risk Prediction in Life Insurance Industry Using Supervised Learning Algorithms. Complex Intell. Syst. 2018;4:145–154. doi: 10.1007/s40747-018-0072-1.
- Takeshima T., Keino S., Aoki R., Matsui T., Iwasaki K. Development of Medical Cost Prediction Model Based on Statistical Machine Learning Using Health Insurance Claims Data. Value Health. 2018;21:S97.
- Shyamala Devi M., Swathi P., Purushotham Reddy M., Deepak Varma V., Praveen Kumar Reddy A., Vivekanandan S., Moorthy P. Linear and Ensembling Regression Based Health Cost Insurance Prediction Using Machine Learning. Smart Innov. Syst. Technol. 2021;224:495–503. doi: 10.1007/978-981-16-1502-3_49.
- Machine Learning-Based Regression Framework to Predict Health Insurance Premiums (2022). Int. J. Environ. Res. Public Health 2022, 19, 7898. https://doi.org/10.3390/ijerph19137898
- Bertsimas D, Bjarnadóttir MV, Kane MA, Kryder JC, Pandey R, Vempala S, et al. Algorithmic prediction of health-care costs. Operations Research. 2008;56(6):1382–92
- Ajay Kumar Sahu, Gopal Sharma, Janhvi Kaushik, Kajal Agarwal, Devender Singh (2023). Health Insurance Cost Prediction by using Machine Learning. SSRN id-4366801
- Asst. Prof. Ms. Madhuri Thorat, Mohasin Patel, Yog Kute, Muskan Sharma, Shweta Bhosale (2022). Medical Insurance Cost Prediction Using Machine Learning.
- Mukund Kulkarni, Dhammadeep D. Meshram, Bhagyesh Patil, Rahul More, Mridul Sharma, Pravin Patange. Medical Insurance Cost Prediction using Machine Learning. International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653.
- Y. Nomura, Y. Ishii, Y. Chiba, S. Suzuki, A. Suzuki, S. Suzuki, K. Morita, J. Tanabe, K. Yamakawa, Y. Ishiwata et al., “Does last year’s cost predict the present cost? an application of machine leaning for the japanese area-basis public health insurance database,” International journal of environmental research and public health, vol. 18, no. 238, 2017