

CMPE - 255 Data Mining

Insurance Premium Prediction

Project Report submitted by

Data Miners

Chaitra Bengaluru Vishweshwaraiah
Dhanasree Rajamani
Sai Pragna Kancheti
Sravani Thota

Introduction

Medical insurance is the contract which requires the health insurer company to pay some or all costs of an individual's healthcare costs in exchange for a premium. It is considered very important for an individual to possess insurance for healthcare, as the charges incurred for medium to serious medical conditions are untenable without the aid of health insurance to cover at least some of the costs. One of the primary focuses of health insurance companies is to make healthcare more affordable by protecting the individuals from unexpected illness or accidents which can easily cost people up to thousands of dollars. While health insurance is essential and beneficial to paying customers, it is also important for the companies providing these services to benefit from the deal. For this purpose, it is important for these companies to collect and understand the data of their customers to understand the features of the individuals, and how these features affect their medical costs.

CRISP-DM

Business Understanding

This project aims to incorporate data mining within insurance - by using various data features to determine the cost of premium to be charged by the health insurance provider for the customer. This helps the insurance provider understand the customer needs and provide services and charge them appropriately.

Demo

https://youtu.be/dnX6y_2rwmQ

Colab

https://colab.research.google.com/drive/1-LlomVakXuUrOFMI341a_8R00tKBxPie?usp=sharing#scrollTo=L-JSAbG8usb3

Data

The dataset consists of the following parameters:

Age : Age of the primary beneficiary

Sex : Sex of the person

BMI : Body Mass Index of the person

Children : Number of children/dependants covered in the insurance

Smoker : Whether the person is a smoker or not

Region : The region in which the person lives in the US

Expenses : The target variable (which is predicted) - denotes the price of the health insurance premium paid annually by the customer

```
[ ] 1 df = df.rename(columns={'expenses': 'premium'})
```

```
[ ] 1 df = df.rename(columns={'children': 'dependants'})
```

```
[ ] 1 df.head()
```

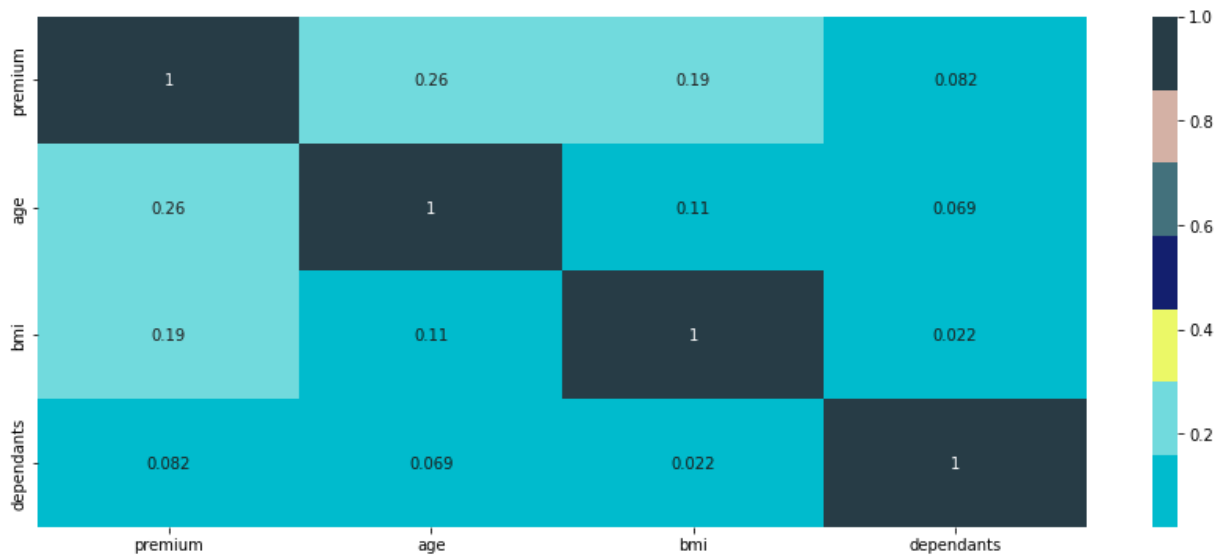
	age	sex	bmi	dependants	smoker	region	premium
0	52	female	32.6	0	no	northwest	9641.14
1	26	female	23.6	1	no	northwest	8209.23
2	24	male	22.7	3	no	northwest	8987.67
3	25	male	31.3	2	no	southwest	4503.26
4	26	female	40.9	0	no	southeast	1581.19

<https://www.kaggle.com/datasets/noordeen/insurance-premium-prediction>

Data Understanding

The main goal of data understanding is to gain general insights about the data that will potentially be helpful for the further steps in the data analysis process.

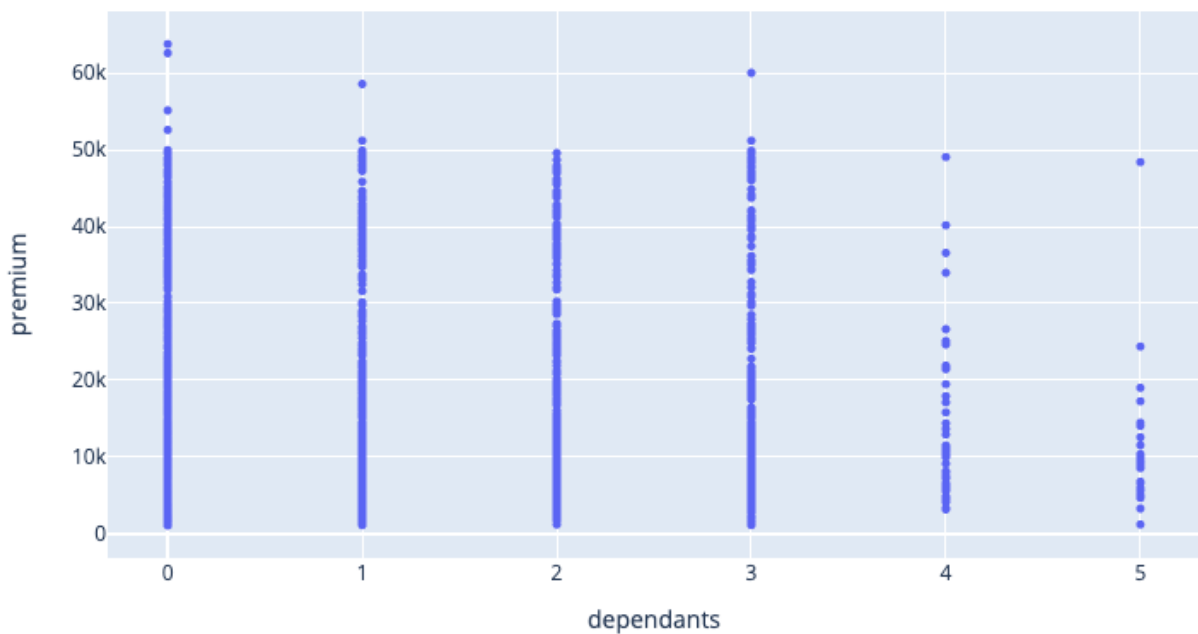
HeatMap to see the correlation coefficient between target label and other features.



From this heatmap, premium is 30% correlated to age and 20% to bmi.

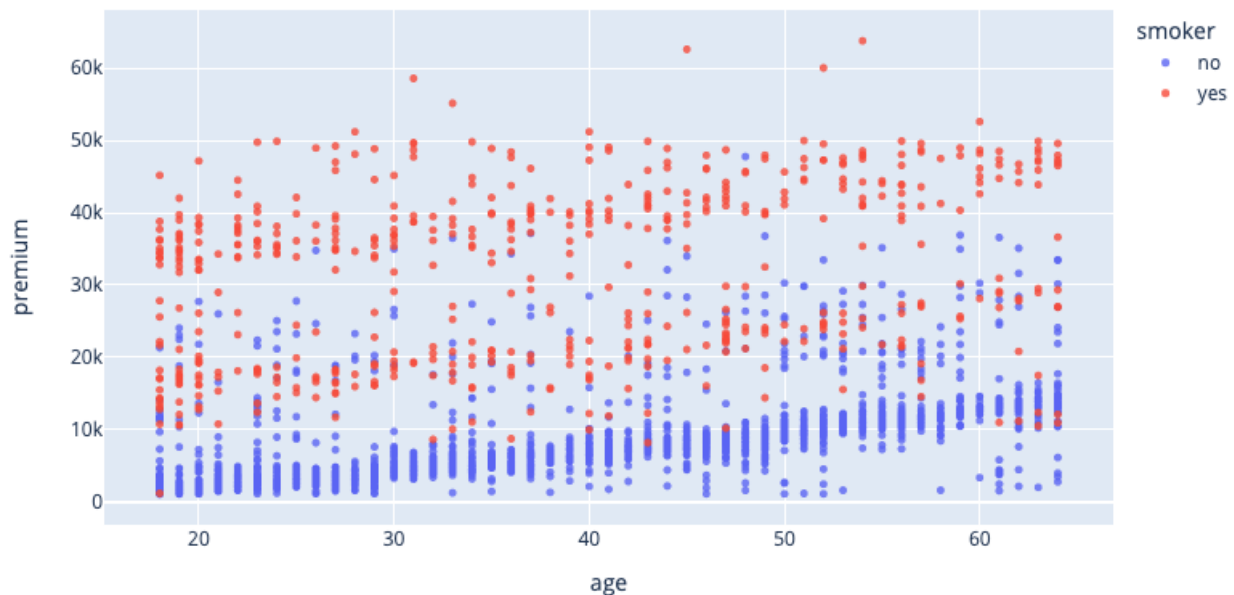
Dependency of features with target variable

dependants vs premium



Even if there are no dependants, the premium seems to be high. Maybe they are smokers or they have major illnesses or diseases.

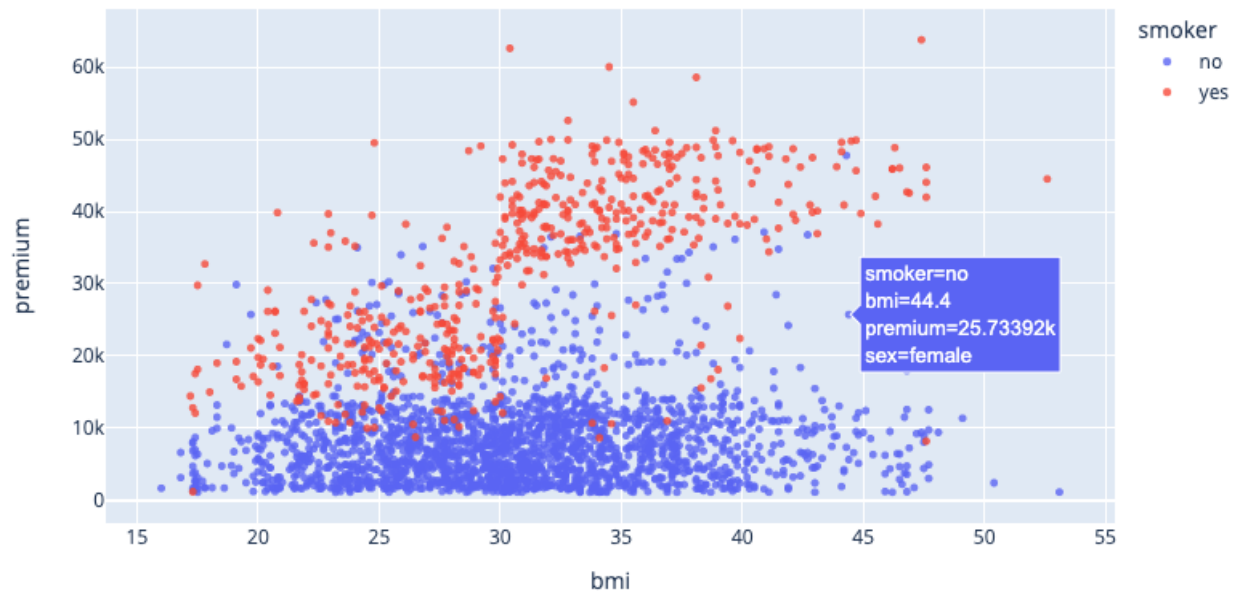
Age vs premium



We can see three "clusters" of points, each of which seems to form a line with an increasing slope:

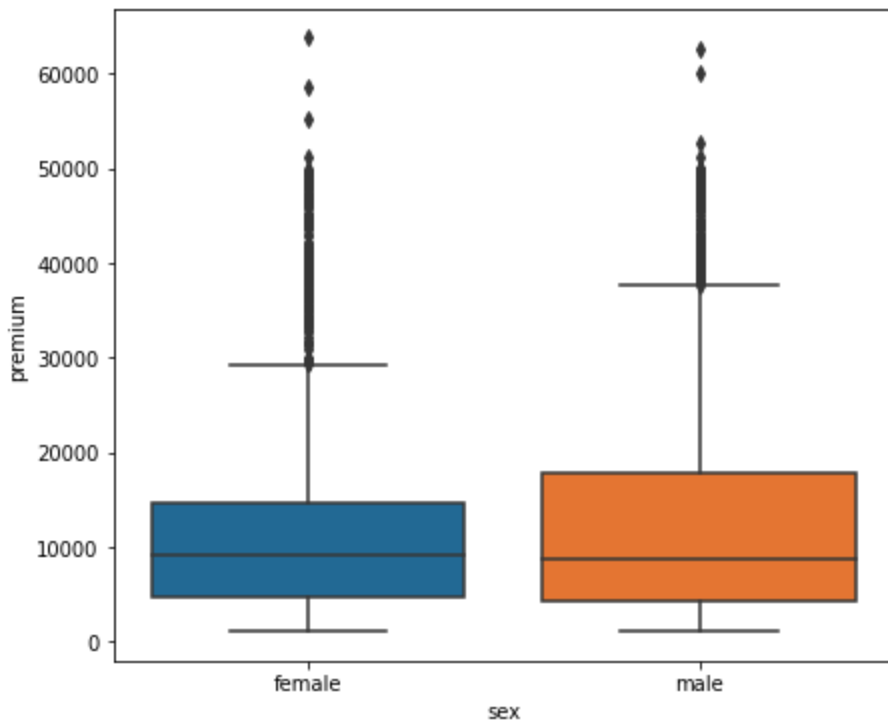
- The first and the largest cluster consists primarily of presumably "healthy non-smokers" who have relatively low medical expenses compared to others
- The second cluster contains a mix of smokers and non-smokers. It's possible that these are actually two distinct but overlapping clusters: "non-smokers with medical issues" and "smokers without major medical issues".
- The final cluster consists exclusively of smokers, presumably smokers with major medical issues that are possibly related to or worsened by smoking.

BMI Vs Premium

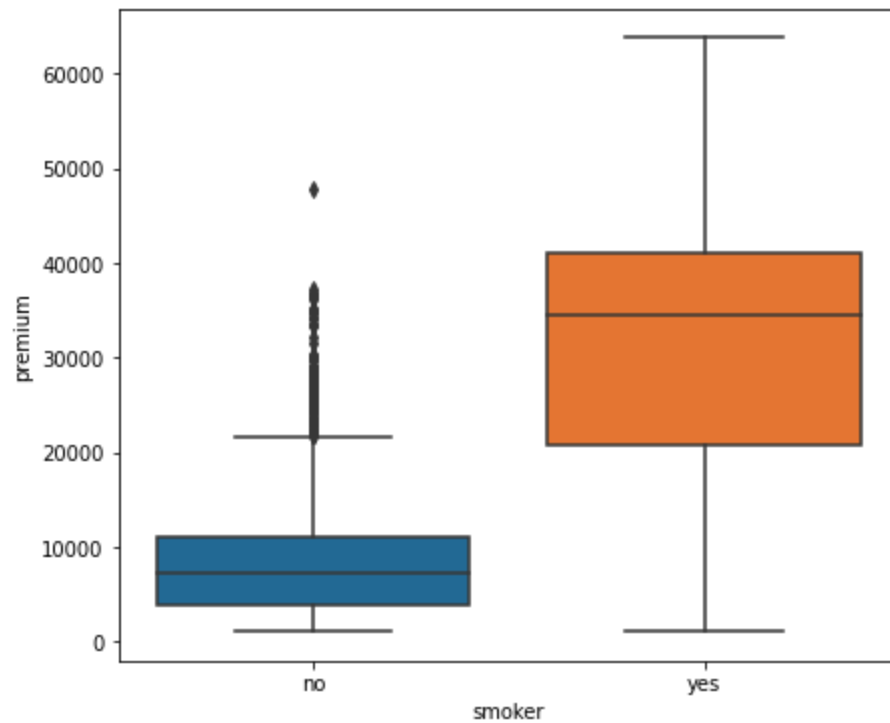


It appears that for non-smokers, an increase in BMI doesn't seem to be related to an increase in medical expenses. However, medical expenses seem to be significantly higher for smokers with a BMI greater than 30.

Visualizing the correlation of each categorical feature to premium

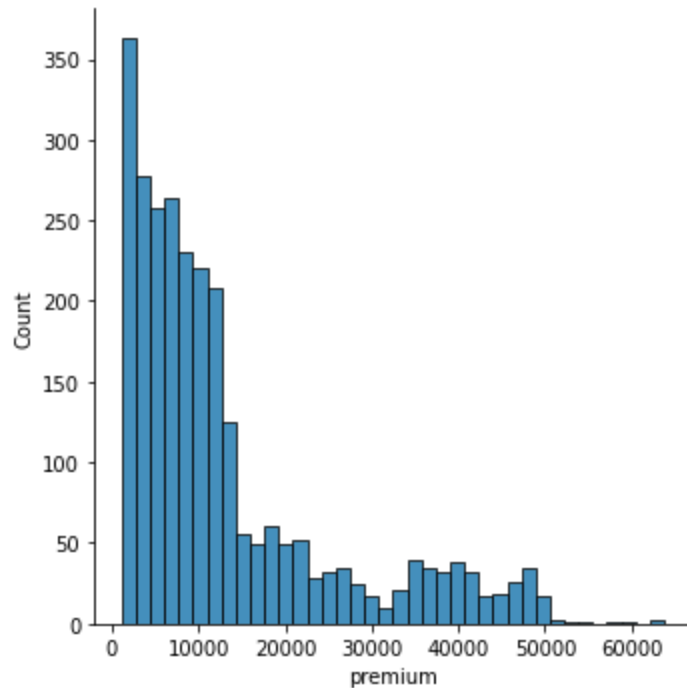


From this box plot, Although the median looks quite same , 75% of female premium is less than male.



From this, a person who smokes tends to pay more premium than non-smoker.

Distribution of features:



For most customers, the annual medical expenses are under 10k. Only a small fraction of customers have higher medical expenses.

We can say that, premium data is:

- Deviated from the normal distribution
- Has positive skewness
- And shows peakedness

So, we need to normalize the data using Normalizing techniques (scaling, log-scaling, z-score).

Data Preparation

Feature Engineering

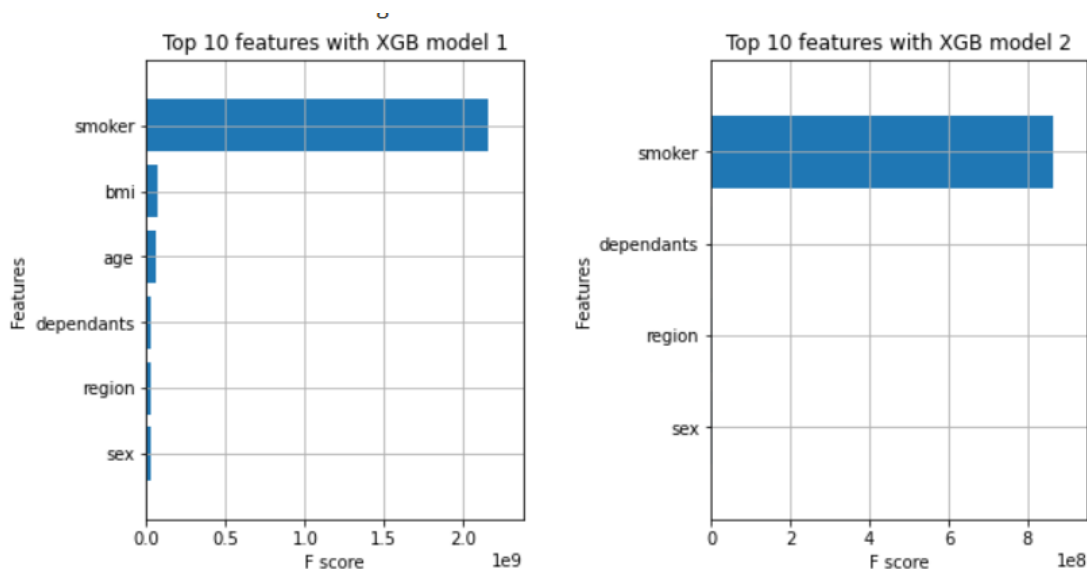
We have Performed feature engineering to understand the different features in the dataset, and their relationship with the target variable. It

helps us understand the importance of every feature in the dataset and how it influences the target variable. We have used FeatureWiz to perform feature engineering. FeatureWiz is a brand new Python library that can automatically help to select the best features from the dataset. FeatureWiz does various steps such as SULO and XGBoost in order to identify the crucial features in the dataset in determining the target label. SULO removes highly correlated features and XGBoost does further feature selection after SULO. FeatureWiz identifies the most important features, and reduces the dataset to have top 4 important features.

Converting categorical to numeric, for FeatureWiz:

```
1 df_enc.head()
```

	age	sex	bmi	dependants	smoker	region	premium
0	52	0	32.6	0	0	1	9641.14
1	26	0	23.6	1	0	1	8209.23
2	24	1	22.7	3	0	1	8987.67
3	25	1	31.3	2	0	3	4503.26
4	26	0	40.9	0	0	2	1581.19



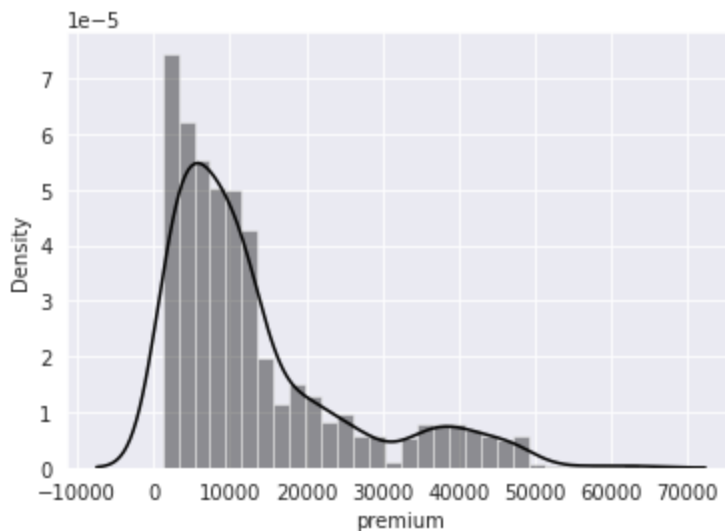
```
Completed XGBoost feature selection in 0 seconds
#####
#####      FEATURE SELECTION COMPLETED      #####
#####
Selected 4 important features:
['smoker', 'bmi', 'age', 'dependants']
Total Time taken for featurewiz selection = 3 seconds
Output contains a list of 4 important features and a train dataframe
```

From performing this feature engineering step, we understand that the attribute “Smoker” has the most influence on the target label “Premium”. We have also used PCA to determine the most important features and perform dimensionality reduction. These steps help us understand the features in the dataset, the importance of features, relationship between the features and their influence on the target label.

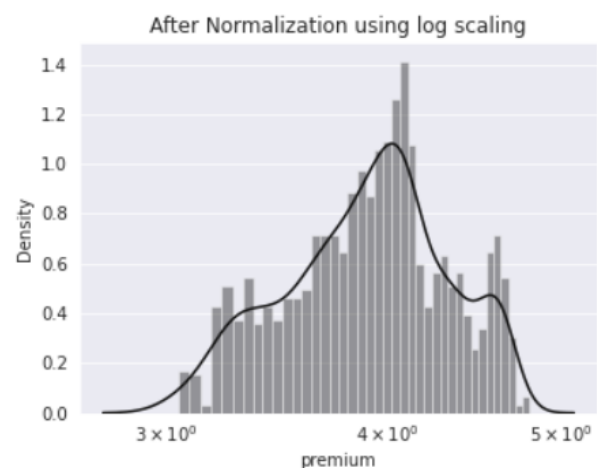
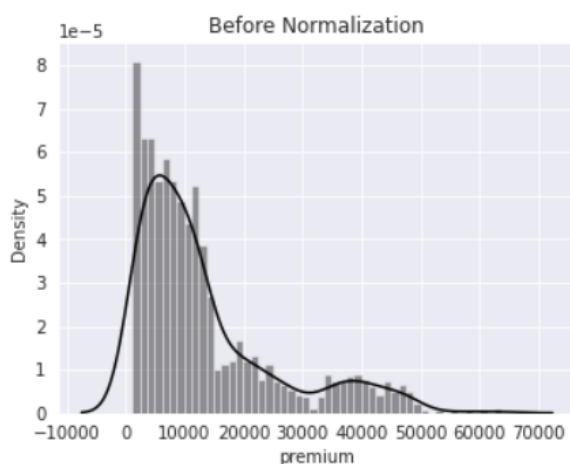
Normalization

The goal of normalization is to transform features to be on a similar scale. This improves the performance and training stability of the model.

```
min 1121.87
median 9382.029999999999
max 63770.43
```



Observing the above graph for insurance premium, it varies from 1121.87 to 63770.43, it seems to be significantly skewed as the median 8944.12 (50 percentile) is much lower than the maximum value 63770.43.



Log Scaling Log scaling is helpful when a handful of values have many points, while most other values have few points. This data distribution is known as the power law distribution. Observing the above graph, data

distribution of insurance premium follows power law distribution.
Hence, Log scaling is used to normalize.

Log scaling changes the distribution, helping to improve linear model performance.

Standard Scaler

The StandardScaler will transform data such that its distribution will have a mean value 0 and standard deviation of 1. This is done feature-wise (in other words independently for each column of the data). Given the distribution of the data, each value in the dataset will have the mean value subtracted, and then divided by the standard deviation of the feature in dataset

Modeling

Regression can be defined as a Machine learning problem where we have to predict discrete values like price, Rating, Fees, etc.

We are using Auto-ML and SKlearn to train regression models

- Pycaret
- LazyPredict
- SKlearn

Linear Regression

Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values.

```
linear_reg = Pipeline(steps=[('preprocessor', preprocessor), ('linear_regressor', LinearRegression())])

linear_reg.fit(X_train, y_train)

y_pred = linear_reg.predict(X_test)

linear_reg_mse = mean_squared_error(y_test, y_pred)
linear_reg_rmse = mean_squared_error(y_test, y_pred, squared=False)
linear_reg_r2_score = r2_score(y_test, y_pred)

# Evaluation Metrics
print("The Mean Squared Error using Linear Regression :{}".format(linear_reg_mse))
print(('The Root Mean Squared Error using Linear Regression :{}'.format(linear_reg_rmse)))
print(('The accuracy using Linear Regression :{}'.format(linear_reg_r2_score)))

The Mean Squared Error using Linear Regression :3360065.35507783
The Root Mean Squared Error using Linear Regression :5796.556335884076
The accuracy using Linear Regression :0.7835726930039905
```

We tried Linear regression on insurance dataset and obtained RMSE of 5796.55 and r2 as 0.78

Decision Tree

Decision tree build regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

Decision Tree

```
[59] decision_tree = Pipeline(steps=[('preprocessor', preprocessor),
                                     ('decision_tree_regressor', DecisionTreeRegressor(max_depth=4, min_samples_split=4, random_state=42)
decision_tree.fit(X_train, y_train)
# Predicting the model
y_pred1 = decision_tree.predict(X_test)
# Evaluation Metrics
decision_tree_mse = mean_squared_error(y_test, y_pred)
decision_tree_rmse = mean_squared_error(y_test, y_pred1, squared=False)
decision_tree_r2_score = r2_score(y_test, y_pred1)

print("The Mean Squared Error using Decision Tree Regressor : {}".format(decision_tree_mse))
print("The Root Mean Squared Error using Decision Tree Regressor : {}".format(decision_tree_rmse))
print("The r2_score using Decision Tree Regressor : {}".format(decision_tree_r2_score))

The Mean Squared Error using Decision Tree Regressor : 3360065.35507783
The Root Mean Squared Error using Decision Tree Regressor : 4590.94481247192
The r2_score using Decision Tree Regressor : 0.864238671935167
```

We tried Decision Tree on insurance dataset and obtained RMSE of 4590.94 and r2 as 0.86

Random Forest

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

▼ Random Forest

```
✓ [60] random_forest_reg = Pipeline(steps=[('preprocessor', preprocessor),  
0s      ('random_forest_regressor', RandomForestRegressor(n_estimators=100, max_depth=4, random_state=42))  
      random_forest_reg.fit(X_train, y_train)  
  
      # Predicting the model  
      y_pred2 = random_forest_reg.predict(X_test)  
  
      # Evaluation Metrics  
      random_forest_mse = mean_squared_error(y_test, y_pred2)  
      random_forest_rmse = mean_squared_error(y_test, y_pred2, squared=False)  
      random_forest_r2_score = r2_score(y_test, y_pred2)  
  
      print("The Mean Squared Error using Random Forest Regressor : {}".format(random_forest_mse))  
      print("The Root Mean Squared Error using Random Forest Regressor : {}".format(random_forest_rmse))  
      print("The r2_score Error using Random Forest Regressor : {}".format(random_forest_r2_score))
```

The Mean Squared Error using Random Forest Regressor : 20054572.894839942
The Root Mean Squared Error using Random Forest Regressor : 4478.233233635777
The r2_score Error using Random Forest Regressor : 0.8708229535056713

We tried Random Forest on insurance dataset and obtained RMSE of 4478.23 and r2 as 0.87

Gradient Boosting

This estimator builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

▼ Gradient Boosting

```
✓ [61] gradient_boosting_reg = Pipeline(steps=[('preprocessor', preprocessor),  
0s      ('gradient_boosting', GradientBoostingRegressor())])  
  
gradient_boosting_reg.fit(X_train, y_train)  
  
# Predicting the model  
y_pred3 = gradient_boosting_reg.predict(X_test)  
  
# Evaluation Metrics  
gradient_boosting_mse = mean_squared_error(y_test, y_pred3)  
gradient_boosting_rmse = mean_squared_error(y_test, y_pred3, squared=False)  
gradient_boosting_r2_score = r2_score(y_test, y_pred3)  
  
print("The Mean Squared Error using Gradient Boosting Regressor : {}".format(gradient_boosting_mse))  
print("The Root Mean Squared Error using Gradient Boosting Regressor : {}".format(gradient_boosting_rmse))  
print("The r2_score using Gradient Boosting Regressor : {}".format(gradient_boosting_r2_score))  
  
The Mean Squared Error using Gradient Boosting Regressor : 18733376.419319134  
The Root Mean Squared Error using Gradient Boosting Regressor : 4328.207067518736  
The r2_score using Gradient Boosting Regressor : 0.8793331451433305
```

We tried Gradient Boosting algo on insurance dataset and obtained RMSE of 4328.20 and r2 as 0.87

KNN

Regression based on k-nearest neighbors.

The target is predicted by local interpolation of the targets associated with the nearest neighbors in the training set.

▼ KNN

```
✓ [63] knn = Pipeline(steps=[('preprocessor', preprocessor),  
0s      ('knn', KNeighborsRegressor(n_neighbors=10))])  
  
knn.fit(X_train, y_train)  
  
# Predictiong The model  
y_pred4 = knn.predict(X_test)  
  
# Evaluation Metrics  
knn_mse = mean_squared_error(y_test, y_pred4)  
knn_rmse = mean_squared_error(y_test, y_pred4, squared=False)  
knn_r2_score = r2_score(y_test, y_pred4)  
  
print("The mean squared error using KNN is {}".format(knn_mse))  
print("The root mean squared error using KNN is {}".format(knn_rmse))  
print("The r2_score using KNN is {}".format(knn_r2_score))  
  
The mean squared error using KNN is 37656326.797995195  
The root mean squared error using KNN is 6136.475111820726  
The r2_score using KNN is 0.7574451920219223
```

We tried KNN algo on insurance dataset and obtained RMSE of 6136.47 and r2 as 0.75

XGBoost

Automatically models numeric targets (squared error function) with random search hyperparameter optimization. XGBoost model trains with lossguide and histogram tree method to accelerate tuning.

```
XGBoost

[64] xgb_reg = Pipeline(steps=[('preprocessor', preprocessor),
                             ('xgb', xgb.XGBRegressor())])

xgb_reg.fit(X_train, y_train)

# Predicting the model
y_pred5 = xgb_reg.predict(X_test)

# Evaluation Metrics
xgb_reg_mse = mean_squared_error(y_test, y_pred5)
xgb_reg_rmse = mean_squared_error(y_test, y_pred5, squared=False)
xgb_reg_r2_score = r2_score(y_test, y_pred5)

print("The mean square error using XGBoost is {}".format(xgb_reg_mse))
print("The root mean_squared error using XGBoost is {}".format(xgb_reg_rmse))
print("The r2 score using XGBoost is {}".format(xgb_reg_r2_score))

[21:13:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror
The mean square error using XGBoost is 18730882.023298938
The root mean_squared error using XGBoost is 4327.91890211669
The r2 score using XGBoost is 0.8793492122374723
```

We tried XGBoost algo on insurance dataset and obtained RMSE of 4327.91 and r2 as 0.87

Model Performance Evaluation

A loss function specifies a penalty for an incorrect estimate from a statistical model. Loss functions specify the penalty as a function of the difference between the estimate and the true value. RMSE is used as loss function.

RMSE

Mean squared error is finding the squared difference between actual and predicted value. Root mean squared error is square root of mean squared error

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

The output value we get is in the same unit as the required output variable which makes interpretation of loss easy

R²

R² score is zero then the above regression line by mean line is equal means 1 so $1-1$ is zero. So, in this case, both lines are overlapping means model performance is worst, It is not capable to take advantage of the output column.

Now the second case is when the R² score is 1, it means when the division term is zero and it will happen when the regression line does not make any mistake, it is perfect. In the real world, it is not possible.

So we can conclude that as our regression line moves towards perfection, R² score move towards one. And the model performance improves.

The normal case is when the R² score is between zero and one like 0.8 which means your model is capable of explaining 80 per cent of the variance of data.

	Model	RMSE	r2_score
5	XGBoost	4327.918902	0.879349
3	Gradient Boosting	4335.759655	0.878912
2	Random Forest	4478.233234	0.870823
1	Decision Tree	4590.944812	0.864239
0	Linear Regression	5796.556336	0.783573
4	KNN	6136.475112	0.757445

From the above observation we can say that the performance (RMSE & R-squared) of the XGboost model is good as compared to other models. So we will save the XGboost model for further testing of the data using the pickle library.

Pycaret

PyCaret is an open-source, low-code machine learning library in Python that helps in preparing data and deploying single or multiple models for comparison

```
[76] best_models = compare_models(exclude = ['ransac'])
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2225.5371	2.194912e+07	4633.0521	0.8497	0.3881	0.1886	0.094
rf	Random Forest Regressor	2297.9490	2.257208e+07	4704.5436	0.8455	0.4174	0.2117	0.411
lightgbm	Light Gradient Boosting Machine	2488.6675	2.412823e+07	4830.6123	0.8353	0.4157	0.2165	0.126
ada	AdaBoost Regressor	3333.7192	2.422357e+07	4901.2949	0.8353	0.4914	0.4467	0.050
et	Extra Trees Regressor	2366.5671	2.484650e+07	4935.7977	0.8299	0.4411	0.2291	0.317
dt	Decision Tree Regressor	3168.4811	4.228928e+07	6456.3041	0.7060	0.5341	0.3534	0.015
omp	Orthogonal Matching Pursuit	5622.9772	5.828418e+07	7619.0884	0.6018	0.6791	0.6751	0.021
ridge	Ridge Regression	4158.5641	6.171344e+07	7788.6469	0.5795	0.4466	0.2786	0.022
br	Bayesian Ridge	4161.6304	6.183550e+07	7796.4147	0.5787	0.4465	0.2785	0.029
lr	Linear Regression	4180.9031	6.263765e+07	7847.9271	0.5733	0.4465	0.2782	0.465

Lazy Predict

Lazy Predict : It is python library used to semi-automate Machine Learning task. It helps to understand which models work better without any hyperparameter tuning. After getting all accuracy we can choose the top 5 models and then apply hyperparameter tuning to them. It provides a Lazy Classifier to solve the classification problem and Lazy Regressor to solve the regression problem.

```
100%|██████████| 41/41 [00.11<00.00,  3.3310/3]
```

	Adjusted R-Squared	R-Squared	RMSE \
Model			
GradientBoostingRegressor	0.88	0.88	4325.57
XGBRegressor	0.88	0.88	4327.92
RandomForestRegressor	0.86	0.86	4616.66
LGBMRegressor	0.86	0.86	4655.19
HistGradientBoostingRegressor	0.86	0.86	4666.75
BaggingRegressor	0.85	0.86	4707.37
ExtraTreesRegressor	0.84	0.84	4999.89
AdaBoostRegressor	0.82	0.83	5201.54
PoissonRegressor	0.79	0.80	5612.47
SGDRegressor	0.78	0.78	5794.30
LinearRegression	0.78	0.78	5796.56
TransformedTargetRegressor	0.78	0.78	5796.56
Lars	0.78	0.78	5796.56
Lasso	0.78	0.78	5797.33
Ridge	0.78	0.78	5798.57
RidgeCV	0.78	0.78	5798.57
BayesianRidge	0.78	0.78	5798.59
KernelRidge	0.78	0.78	5798.89
LassoLars	0.78	0.78	5808.07
OrthogonalMatchingPursuitCV	0.78	0.78	5829.68
LassoCV	0.78	0.78	5837.29

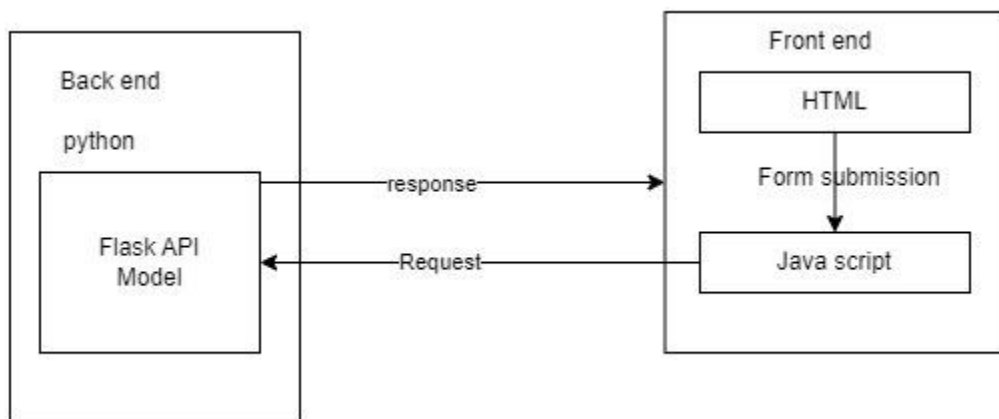
Header use times new roman 20, paragraph times new roman 16

Various Algorithms used

Deployment plan:

We are using python's flask api to make the best model trained accessible through an api.

And we are using javascript to make XHR requests to the Flask api with user inputs as request parameters.



ARCHITECTURE DIAGRAM

When a user enters the details in HTML form and submits the form a javascript function is triggered due to the form submission event. This Request will go to a python flask API in which the best fit model is loaded and used for premium prediction based on the user inputs from the request.

Flask apis

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks. Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use.

The flask object implements a WSGI application and acts as the central object. It is passed the name of the module or package of the application. Once it is created it will act as a central registry for the view functions, the URL rules, template configuration and much more.

Anaconda Prompt (anaconda3) - python deploy.py

```
(base) C:\Users\pragna\Downloads>python deploy.py
* Serving Flask app "deploy" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 962-421-380
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```


Java script XHR requests

XMLHttpRequest (XHR) objects are used to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing.

Flow of request

User inputs are sent to flask api in the following format.

Request

```
{  
age : "age of the user",  
sex : "gender of the user",  
bmi: "BMI of the user",  
children: "number of children of the user",  
smoker: "is the user a smoker",  
region : "region of the user"  
}
```

The response from the flask api is in the following format.

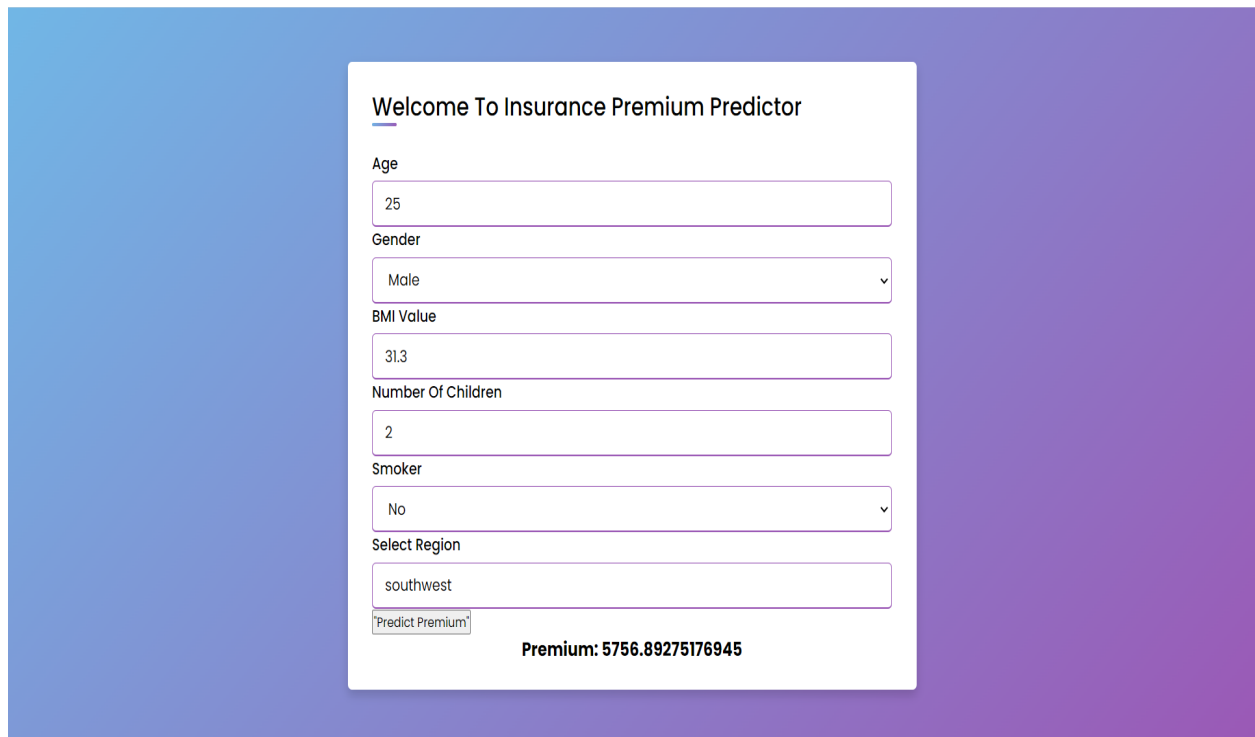
Response

```
{  
"premium prediction"  
}
```

ERROR HANDLING IN FLASK API

If there are any errors in the data types entered by the user then FLASK API will resolve it by matching the data types into the formats that the model is trained on. If there are any errors while predicting the output using user inputs and best fit model. The API handles it and sends the error message.

SNIPPET OF WEB PAGE WITH USER INPUT PREDICTION



Welcome To Insurance Premium Predictor

Age
25

Gender
Male

BMI Value
31.3

Number Of Children
2

Smoker
No

Select Region
southwest

Predict Premium

Premium: 5756.89275176945

