PROJECT REPORT

Insurance Premium Prediction

Created by *Vikram Singh*

GitHub Profile : [VkasRajpurohit (github.com)]

ACKNOWLEDGEMENT

"It is not the brain that matter the most, but that which guide them: The character, the heart, generous qualities and progressive force."

I would like to make a number of acknowledgments to the **iNeuron Intelligence Pvt Ltd**, who gave me this opportunity to work on this project.

Vikram Singh

TABLE OF CONTENTS

ABSTRACT

While taking insurance policy, an individual have to contact an insurance agent or insurance representative and provide required information to reach at the premium for the insurance policy. However, it is hectic and time-consuming process. To fill-up this gap, built this application which allows an individual to insert the required information and get the **Insurance Premium** as result.

Insurance Premium Prediction is a Machine Learning Algorithms based model. For the Problem statement data collected via Kaggle platform and built an end-to-end deployment ML model.

After data preprocessing and EDA, came to the conclusion that **Age** and **Smoker** are the most significant features which affecting the target/dependent feature. After feature engineering, the **performance metrics RMSE, MAE & R2 Score** selected. RMSE & MAE to see outlier impact and compare it and R2 Score, since in feature engineering, not creating huge number of new features, hence no impact on target/dependent feature. Built different-different ML models & compared R2_Score with Cross_Validation_Score, found **StackingRegressor** model as the **best model** with the **cross_validation_score 80.93 %**. In addition, with the **Hyper-parameters tuning** using RandomizedSearchCV, **cross_validation_score improved to 83.21%** i.e. 2.28% performance increased. **Saved the best model in pickle file** format.

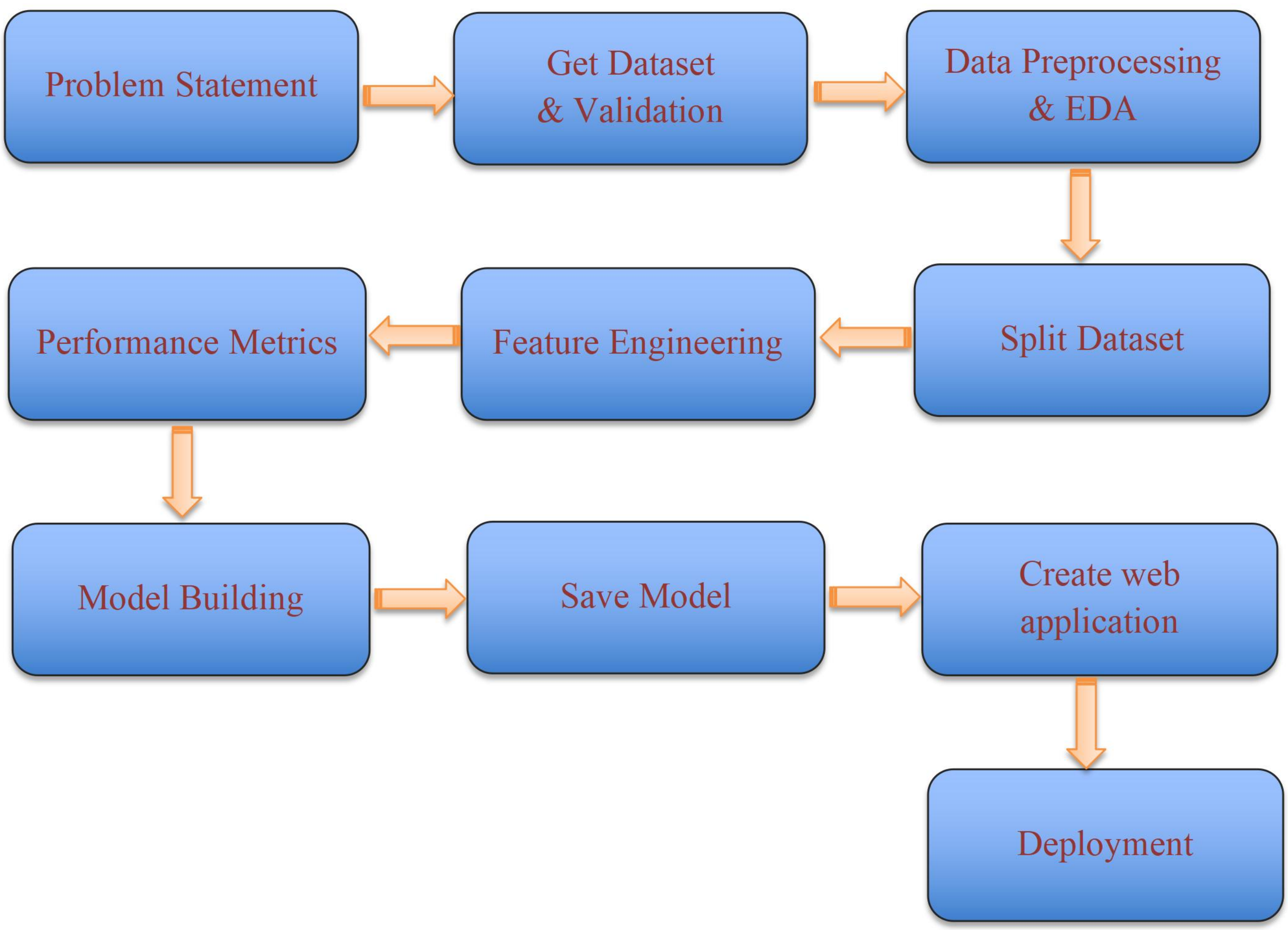
After the training part, created a **User Interface web application** using **Flask-API & HTML** and deployed the model for productionisation using **Heroku & AWS**. User will enter the required values & hit Get Premium, and it will show the **premium** as result.

Web app-- Insurance Premium Prediction (insurance-premium-webapp.herokuapp.com)

Overview:

Title	:	Insurance Premium Prediction
Domain	:	Insurance
Tools & Technology	:	Python | Data-Preprocessing | EDA | Feature Engineering | Feature Transformation & Scaling | Machine Learning | Flask-API | HTML | Bootstrap | GitHub | Heroku | AWS
ML Algorithms	:	Linear Regressor | SVM Regressor | DT Regressor | RF Regressor | Gradient Boosting Regressor | Stacking Regressor
IDE	:	PyCharm | Google Colab

Process Flow:



CHAPTER 1
INTRODUCTION

1. Introduction:

The **Insurance** word means **to the protection from financial loss**, basically it is a form of risk management against the risk of a contingent or uncertain loss. Insurance have many different-different types, some of the examples are health and life insurance (an individual must have these two).
The **Premium** word means to **an amount paid periodically** to the insured/nominee by the insurer **for covering risk**. In an insurance contract, the risk is transferred from the insured to the insurer. For taking this risk, the insurer charges an amount called the premium.

- Regarding the health/life insurance, during the **COVID**, people became more aware about the importance of health/life insurances, however still there is some kind of lag to understand the premium of the health/life insurances for an individuals.

1.1 Business Problem:

When we see insurance industries in business prospective- If an individual willing to take insurance plan, so they have to call the insurance company's helpline number and have to consult as per the need and different-different parameters, this is hectic process, also in this process, there are many of the negative points mentioned below.

1. Sometimes they have to wait for long hold.
2. May be they did not find the enough educated insurance representative, might lead to wrong information.
3. Delay in taking final decision and many more.

- Hence trying to resolve the above problems using Machine Learning Algorithms.
- Trying to build a user friendly ML model which can save time and effort for an individuals to reach at the insurance premium estimate with accuracy.

The purposes of this case study is to look into different-different **features** (related to the insurance domain) and observe their relationship between/among them. Based on several features of an individual such as **age**, **physical/family condition** and **location** against their existing medical expense, to be used for predicting future medical expenses of individuals that help medical insurance or individuals to make decision on charging the premium.

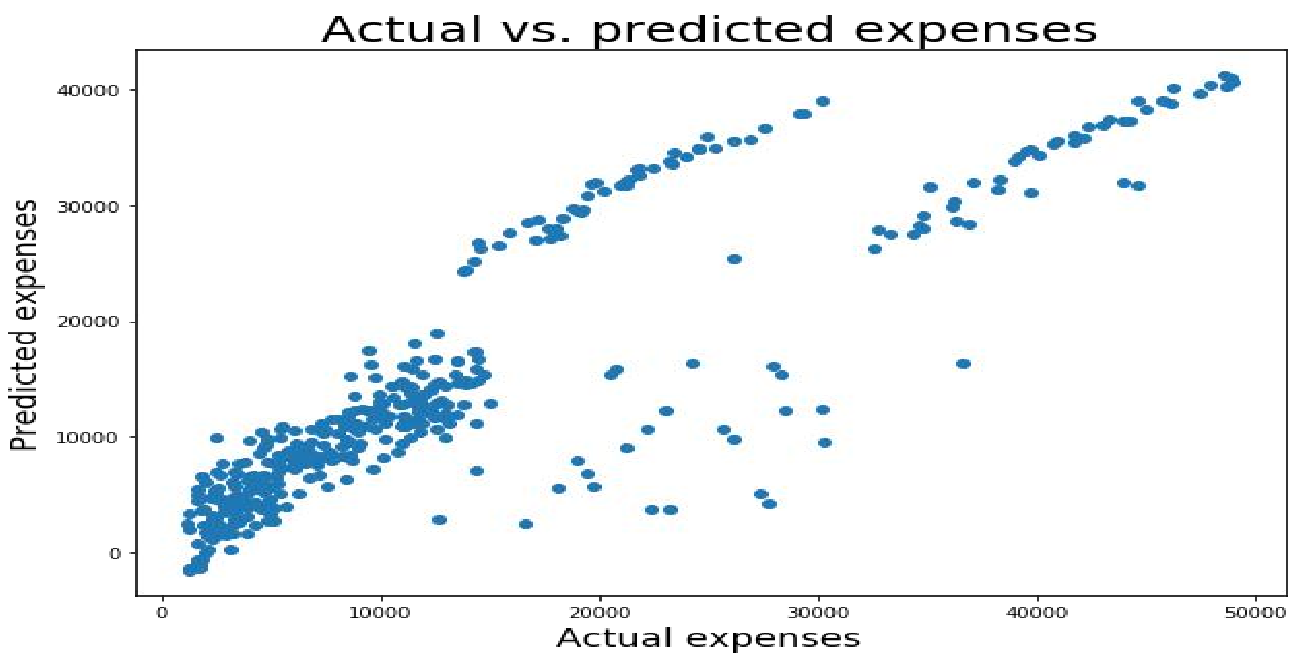
1.2 Expected Solution:

- Build a solution that should be able to predict the premium of the an individual health insurance based on given features in dataset.

1.3 Existing solutions:

- Insurance Premium | Kaggle - In this, shown EDA and Linear regression Technique.

1) For categorical features used Label Encoding & One-Hot Encoding, however no measure difference found in results by using these techniques.
2) Also predicted Expenses not found linear with actual expenses.
3) Actual Expenses vs Predicted Expenses shown below-



- Improvised Accuracy - Predicting Insurance Premium | Kaggle - In this, shown EDA and different-different machine learning algorithms to improve the prediction R2_Score.

1) Below are the some of the techniques used and respective R2_Score -

ML Model	R2_Score for 20 - Fold Cross Predicted
Multiple_linear_Regression	: 71.64%
Support_Vector_Regressor	: 70.09%
Polynomial Features	: 83.93%
Decision_Tree_Regression	: 85.19%
Random_Forest_Regression	: 85.77%

2) Random_Forest_Regression Model have the highest accuracy compared to all other models.

CHAPTER 2
ML formulation of the Business Problem

2. ML formulation of the business problem:

First Cut Approach

As per the problem statement, we need to predict **Expenses.**

- As checked dataset, it is labeled data i.e. we can go with the supervised machine learning techniques.
- **It is a Regression problem**.
  - For Regression problem, we can go with Linear Regression, SVM, Decision Tree Regressor and Ensemble techniques (RF, GB), also the **StackingRegressor**.
  - With Linear Regression, have to keep in mind basic 4 assumptions

    1. **Linearity**: The relationship between X and the mean of Y is linear.
    2. **Homoscedasticity**: The variance of residual is the same for any value of X.
    3. **Independence**: Observations are independent of each other.
    4. **Normality**: For any fixed value of X, Y is normally distributed.

       - QQ plot to check if normally distributed.
       - If not normally distributed, convert to normal distribution using Log transform, box-cox transform, exponential transform, power-low transform.

  - If data is non linear, we can go with SVM, Decision Tree Regressor and Ensemble techniques, however time complexity is more for these compared to Linear Regression.

- First we will perform EDA.
- We can check, dataset is linear or non linear by correlation.
- We will try to build different-different ML models and try to improve performance by hyper-parameter tuning, the model which will give the best **cross_validation_score**, we will select that one.

CHAPTER 3
Business Constraints & Dataset Column Analysis

3.1 Business Constraints:

1) **Time :** Latency is really not a major issue, even a few seconds of the latency is considerable.
2) **Accuracy:** Accuracy is a vital constraint, high accuracy in predicted premium with actual premium will build-up a trust, which will lead to referral i.e. more user, more business.
3) **Interpretability:** Model should be easy to interpret and user friendly.

3.2 Data set Column Analysis :

Dataset- Kaggle platform.

The dataset contains 1338 observations (rows) and 7 features (columns).

- There are **4 numerical features** (age, bmi, children and expenses) and **3 categorical features** (sex, smoker and region).
- Unique values for categorical features- sex: 2, smoker: 2, region: 4.

Features:

- **Age:** Age is the domain feature, as the age increases, insurance premium is more.
- **BMI:** it is Body mass index. It is a value derived from the mass and height of a person. The BMI is defined as the body mass divided by the square of the body height, and is expressed in units of kg/m^2, resulting from mass in kilograms and height in meters.
- **Children:** Number of children an individual have.
- **Expenses:** It is the target/dependent feature. It is the overall medical expense/premium yearly.
- **Sex:** Sex of an individual i.e. male or female.
- **Smoker:** It is also a domain feature, an smoker individual have to pay high premium compared to a non-smoker.
- **Region:** It is representing region of an individual belongs. It have four unique values i.e. southeast, southwest, northwest, northeast.

CHAPTER 4
Data Preprocessing

4. Data Preprocessing:

4.1 Basic information about the data -

- Shape of the dataset: (1338, 7)
- Column names: ['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'expenses']
- More data information -

```
<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   expenses  1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

4.2 Basic statistics information -

- An individual's maximum age is 64 and minimum is 18.
- Maximum bmi is 53.1.
- An individuals have maximum 5 children.
- Also observed that more number of children tends to have more expenses.
- Average expenses is 13k.

4.3 Checking nominal/categorical features -

```
Unique value_counts for categorical features-

male      676
female    662
Name: sex, dtype: int64

no     1064
yes     274
Name: smoker, dtype: int64

southeast    364
southwest    325
northwest    325
northeast    324
Name: region, dtype: int64
```

- **Number of samples** taken for male, female and region wise are **balanced**.

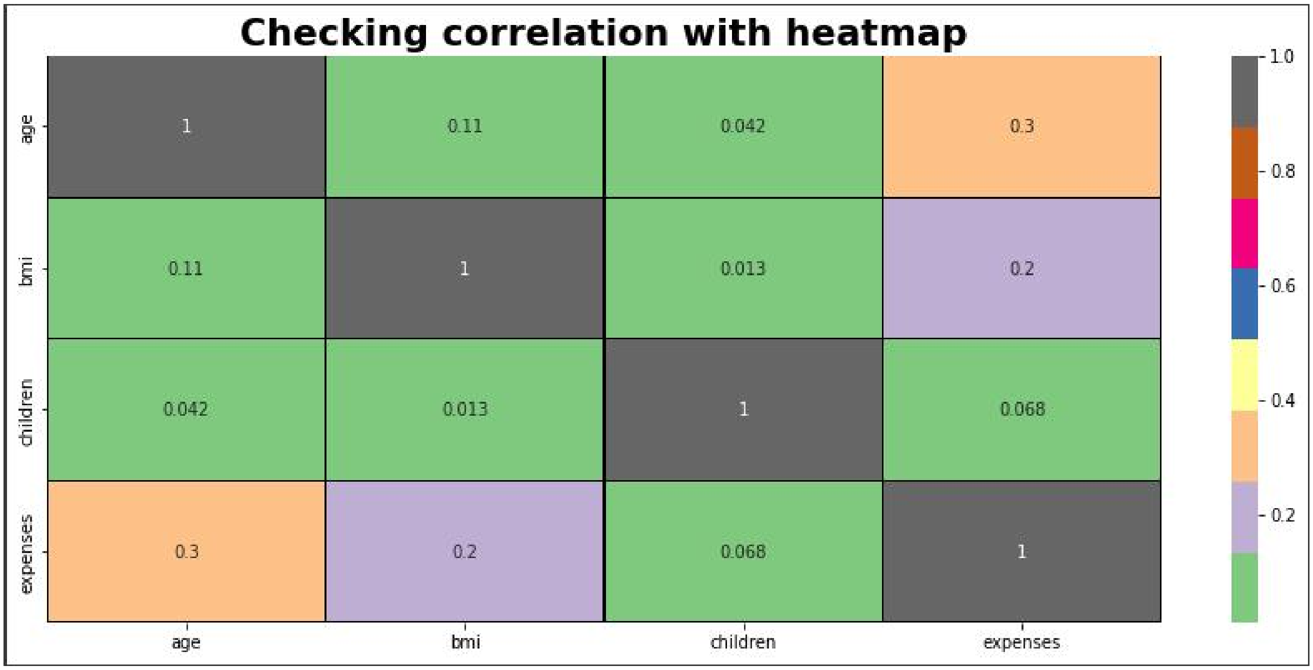
4.4 Missing Values -

- No missing values found.
- Since not filling missing values, hence there is no data leakage problem.

CHAPTER 5
Exploratory Data Analysis

5. Exploratory Data Analysis:

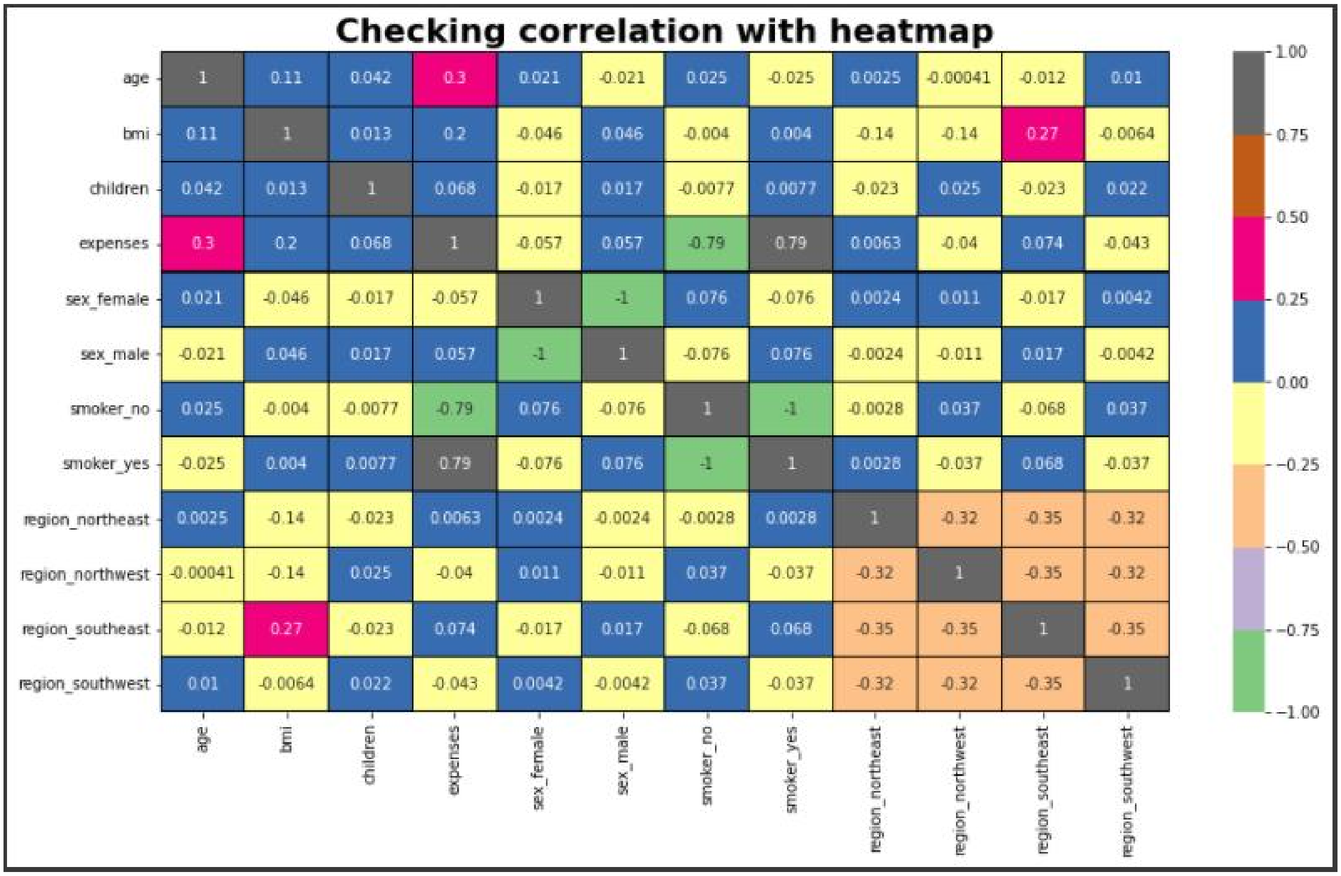
5.1 Checking correlation -



5.1.1 Analyzing heatmap -

1) Target feature is expenses.
2) No feature is negatively correlated, all are positively correlated.
3) **Age** is less correlated with bmi & children, a little bit positively correlated with expenses.
4) **BMI** is less positively correlated with expenses as compared to Age.
5) **Children** is less positively correlated with expenses as compared to Age and bmi.
6) As the conclusion, **Age** is the most significantly feature, which is correlated with the expenses.
7) Age, BMI & Children are not highly correlated with each-other, hence cannot be dropped one of these feature.

5.1.2 Correlation with dummy variables of categorical features -



Analyzing heatmap with all the dummy data -

1) **Age** significantly correlated with the expenses only, not correlated with any of the other features.
2) **BMI** is not significantly correlated with any of the other features. BMI is the most correlated with the region_southeast (0.27) and also a little bit correlated with the expenses (0.2).
3) **Children** is not significantly correlated with any of the features. Individually it is correlated with expenses only (0.068).
4) **Sex** have two categories i.e. female & male.

➢ **Female & Male** both of the features is not significantly correlated with any of the other features, even not correlated with expenses.

5) **Smoker** have two categories i.e. yes & no.

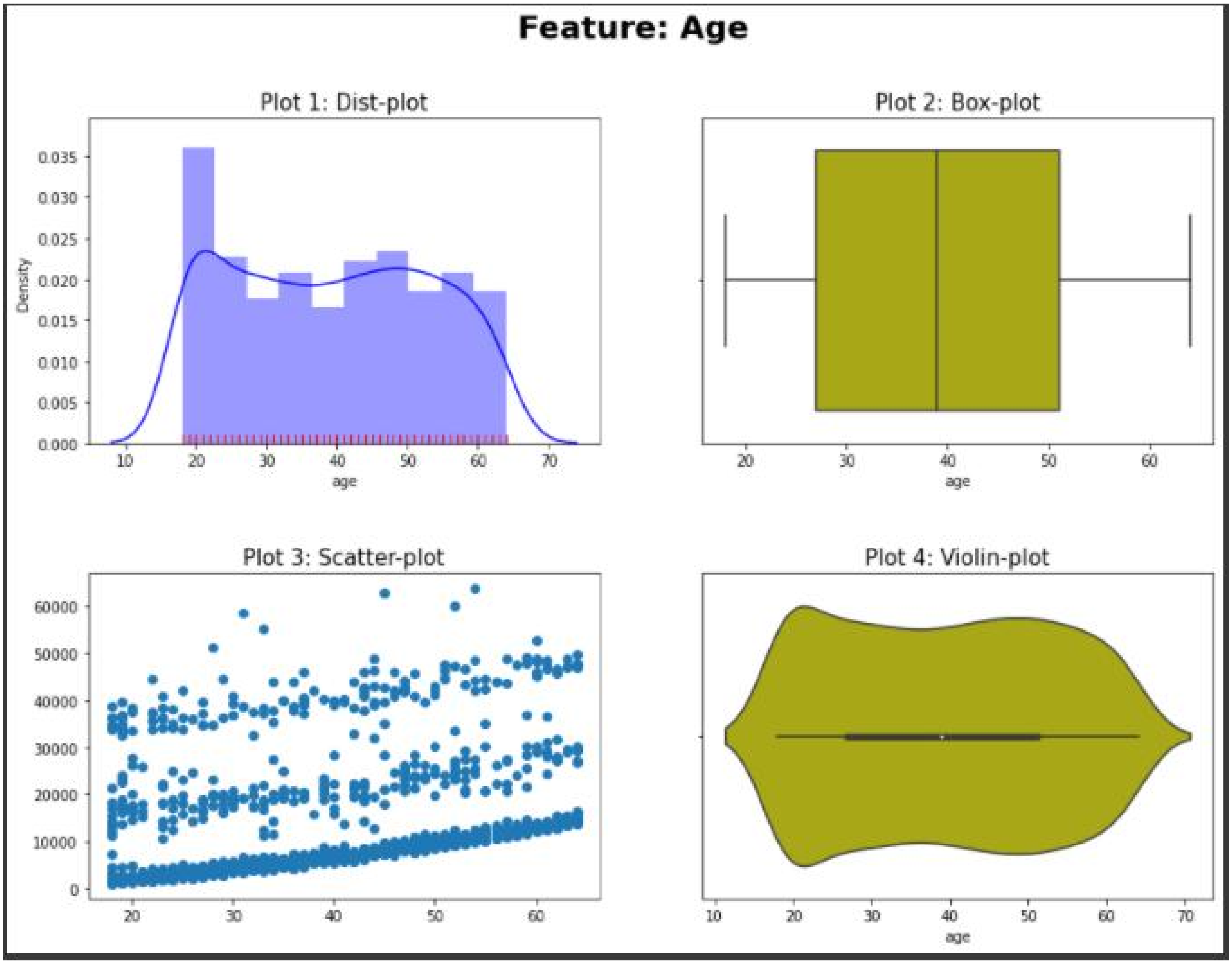
➢ **Yes & No** both of the features is not significantly correlated with any of the other features, individually **Yes is highly positively correlated with expenses** only (0.79) & **No is highly negatively correlated with expenses** only (-0.79).

6) **Region** have four categories i.e. northeast, northwest, southeast & southwest.

➢ These features are a little bit negatively correlated with each other.

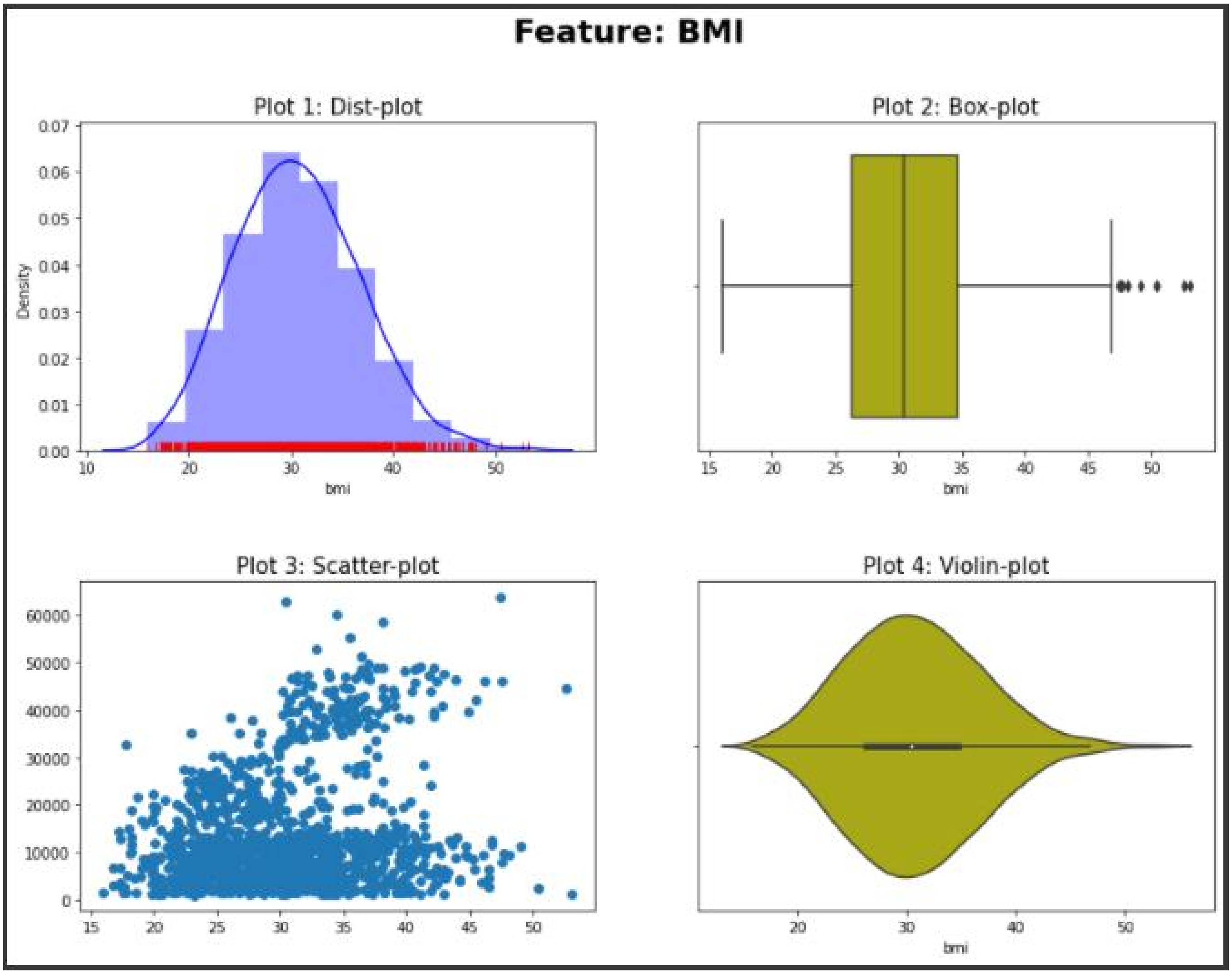
➢ However, region_southeast have a little bit higher expenses compared to the other region.

5.2 Analysis of Feature: Age



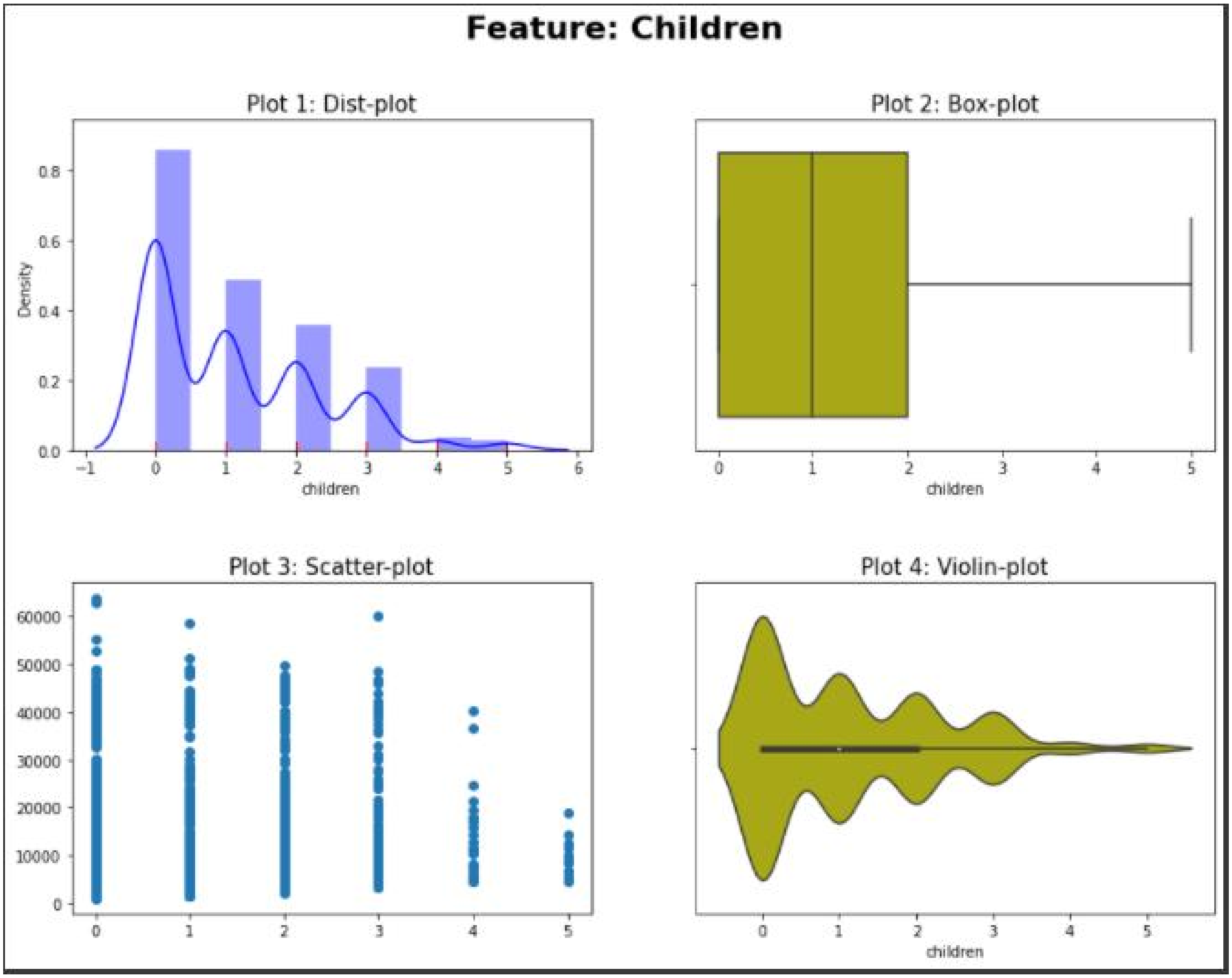
- Plot 1: Dist-plot, shows that it is uniformly distributed, the minimum age is 18 & maximum age is 64.
- Plot 2: Box-plot & Plot 4: Violin-plot, shows that there is no outlier present in age feature.
- Plot 3: Scatter-plot, shows that as the age increases, slightly the expenses are higher.

5.3 Analysis of Feature: BMI



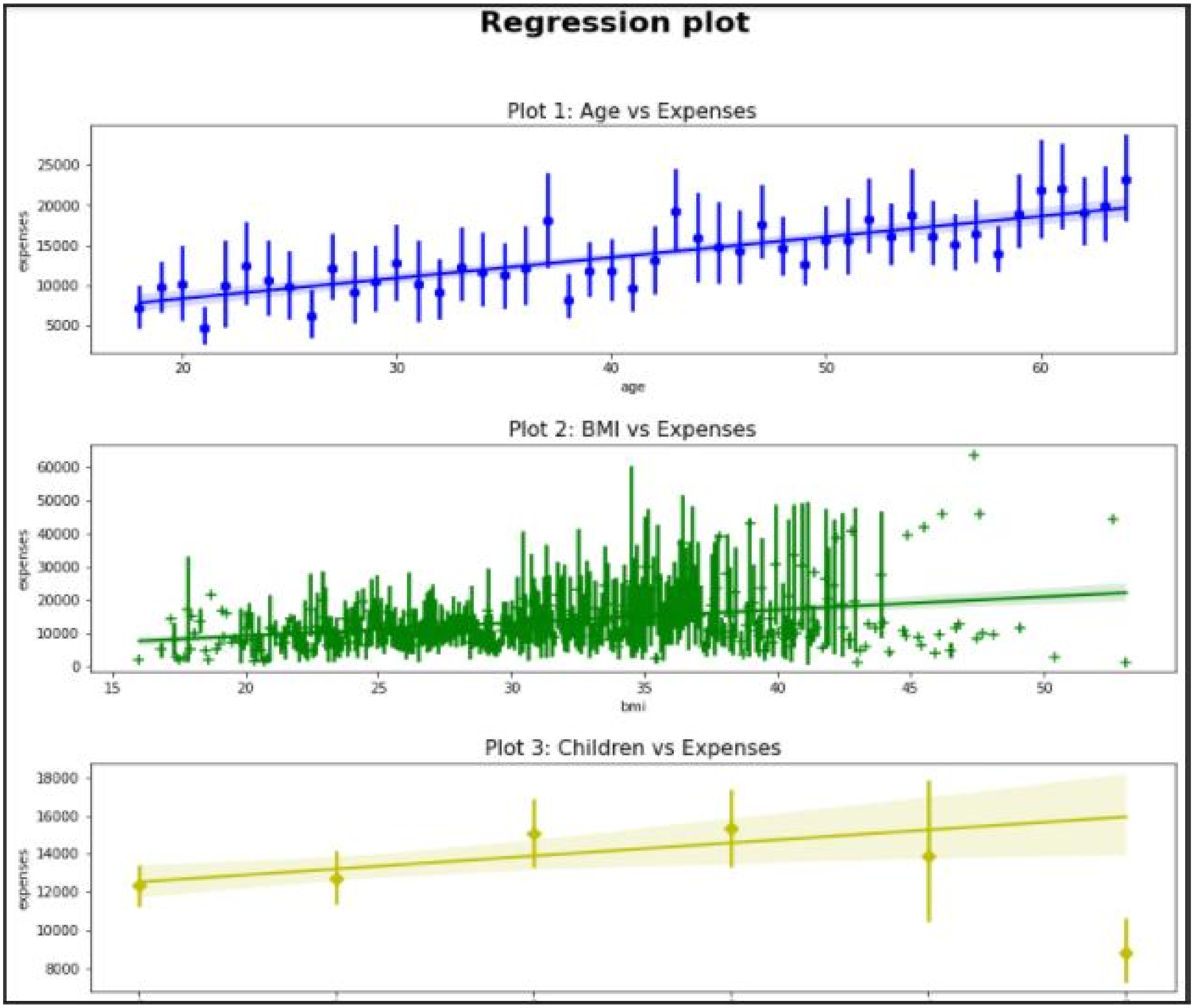
- Plot 1: Dist-plot, shows that it is normally distributed.
- Plot 2: Box-plot & Plot 4: Violin-plot, shows that there are outliers present in bmi feature.

5.4 Analysis of Feature: Children



- Children is the ordinal feature.
- It shows that the majority of the family have 1-2 children, however an individual number are the highest with no child.

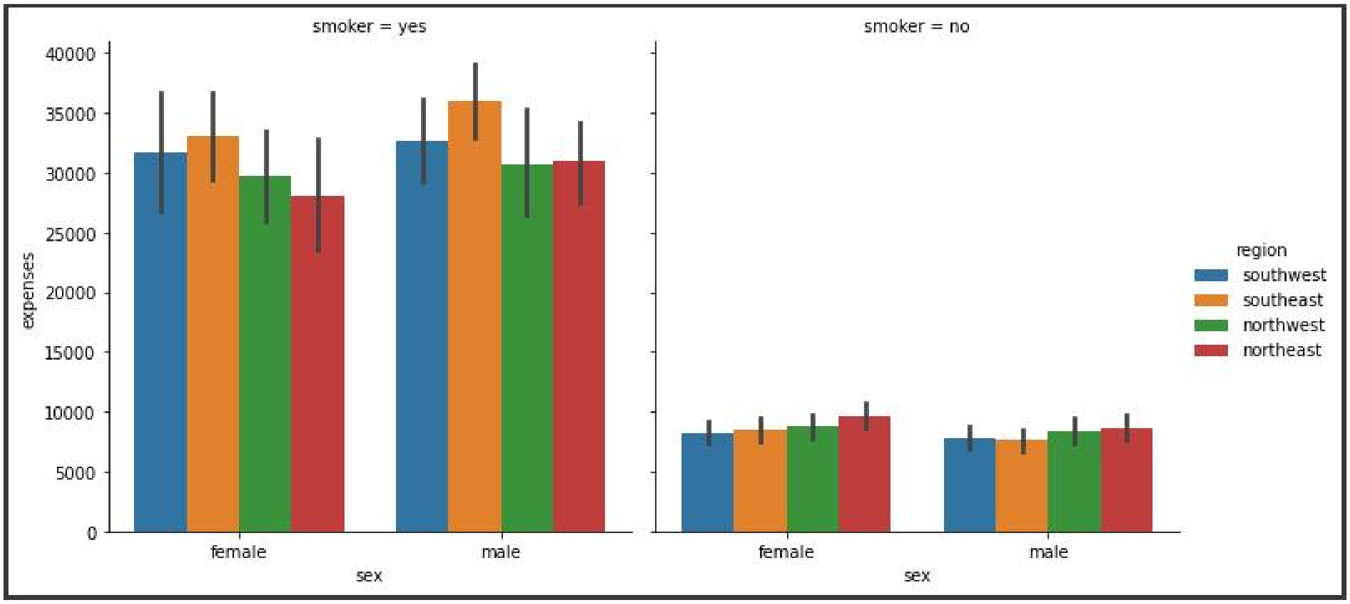
5.5 Regression plot with features: Age, BMI, Children & Expenses



Analyzing Regression plot -

1) **Age** is slightly linear with expenses.
2) However, unable to conclude linearity regarding bmi & children features with expenses.

5.6 Categorical plot with features: Sex, Smoker, Region & Expenses



Analyzing Categorical plot -

1) Features **region & sex** wise not a major change in expenses, however **region_southeast** have a little bit **higher expenses** compared to the other region.
2) It is a **major conclusion** that **smoker have very high expenses** compared to the non-smoker.

CHAPTER 6
Feature Engineering

6. Feature Engineering :

Feature engineering is the process of using domain knowledge to **extract the features from raw data** and use these extra features **to improve the quality of results** from a machine learning algorithms, compared with supplying only the raw data to the machine learning algorithms.

6.1 Split Dataset:

Splitting the dataset into train data & test data, it is a good practice to **split** the dataset before Feature Engineering to avoid the **data leakage** problem.

6.1.1 Independent & dependent features -

```
X= data.drop('expenses', axis=1)   # Independent features
y= data['expenses']                # dependent feature
```

```
Independent features X_shape:  (1338, 6)
dependent feature y_shape:  (1338,)
```

E.g. - Sample dataset for independent & dependent features

Independent Features

	age	sex	bmi	children	smoker	region
0	19	female	27.9	0	yes	southwest
1	18	male	33.8	1	no	southeast
2	28	male	33.0	3	no	southeast
3	33	male	22.7	0	no	northwest
4	32	male	28.9	0	no	northwest

Dependent Feature

```
0    16884.92
1     1725.55
2     4449.46
3    21984.47
4     3866.86
Name: expenses, dtype: float64
```

6.1.2 Split dataset using sklearn.model_selection.train_test_split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=.30, random_state=0)
```

```
X_train: (936, 6) ; X_test: (402, 6)
y_train: (936,) ; y_test: (402,)
```

Now, come to the **Feature Engineering** part, In data preprocessing step, no missing values found in the dataset, here need to handle categorical features only. **Categorical features** can be divided into **Nominal & Ordinal** categorical features.

6.2 Handling Categorical Features :

Machines unable to understand the categorical values, have to convert these values into numeric or float values. Encoding techniques used to handle categorical features.

6.2.1 Nominal Categorical Features -

Nominal categorical features are the features that have two or more categories, however don't have to worry about the order/arrangement of categories. These can be handled by the following techniques.

1) **One-hot-encoding with dummy variable trap**

- It will create a new columns for each category, however by applying dummy variable trap, columns can be reduced by 1.
- Disadvantage- **curse of dimensionality** i.e. many new columns created.

2) **One-hot-encoding with many categorical variables-** When the feature have many numbers of the categorical data points, then only the 10 most repeated categories can be selected as new features.
3) **Mean-encoding -** Instead converting to category, calculate mean based on dependent feature and replace the categorical data points with the mean value e.g. Pin-code.

6.2.2 Ordinal Categorical Features -

Ordinal categorical features are the features that have two or more categories just like nominal features, however have to arrange the order/rank of categories.

1) **Label encoding -** It will arrange in rank or order.
2) **Target guided ordinal encoding -** In this, take categorical feature as well as dependent feature and calculate aggregated mean of the categorical feature and apply it to the dependent feature, then assign higher integer values or a higher rank to the category with the highest mean.

In dataset, both of types categorical features present i.e. **Nominal Categorical Features** - Sex, Smoker, Region & **Ordinal Categorical Features** - Children.

- **Children** feature is already arranged in order/rank.
- For features **Sex, Smoker, Region** using **one-hot-encoding with dummy variable trap** since there aren't huge number of categories present in any particular features.

```
X_train= pd.get_dummies(X_train, drop_first= True)
X_test= pd.get_dummies(X_test, drop_first= True)
```

```
After One-hot-encoding Shape-
X_train: (936, 8) ; X_test: (402, 8)
```

E.g. - After one-hot-encoding Sample dataset for X_train

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
1163	18	28.2	0	0	0	0	0	0
196	39	32.8	0	0	0	0	0	1
438	52	46.8	5	0	0	0	1	0
183	44	26.4	0	0	0	1	0	0
1298	33	27.5	2	1	0	1	0	0

6.3 Feature Transformation & Scaling:

Feature transformation is simply a function that transforms features from one representation to another. Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data **normalization**.

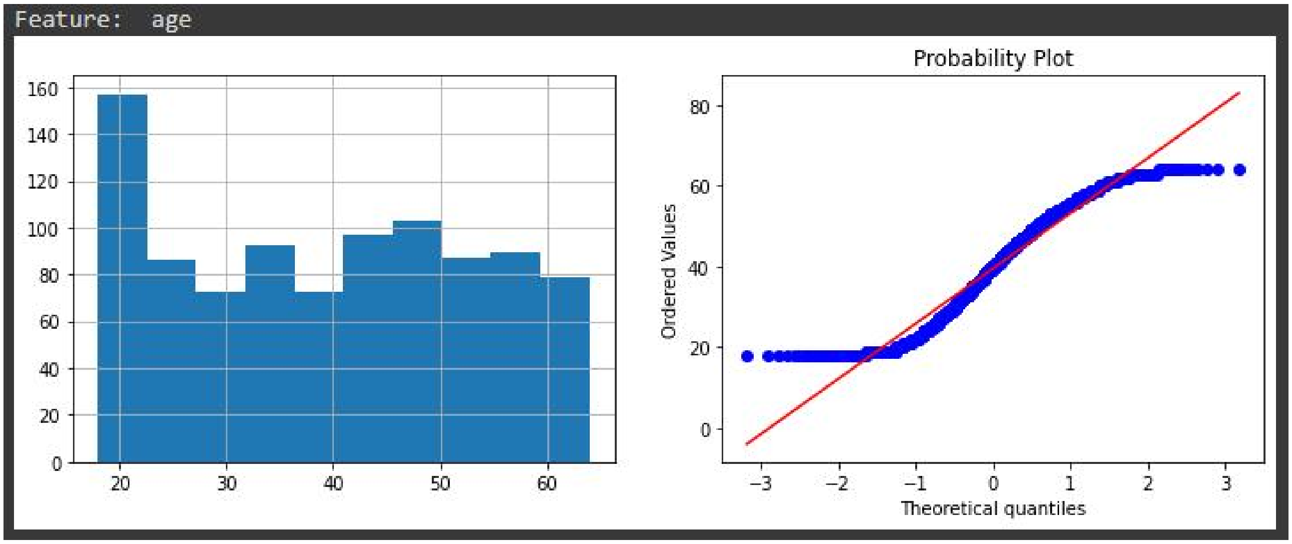
Algos require feature scaling –

- Linear regression - to converge faster
- KNN, K-mean clustering- for euclidean distance
- SVM

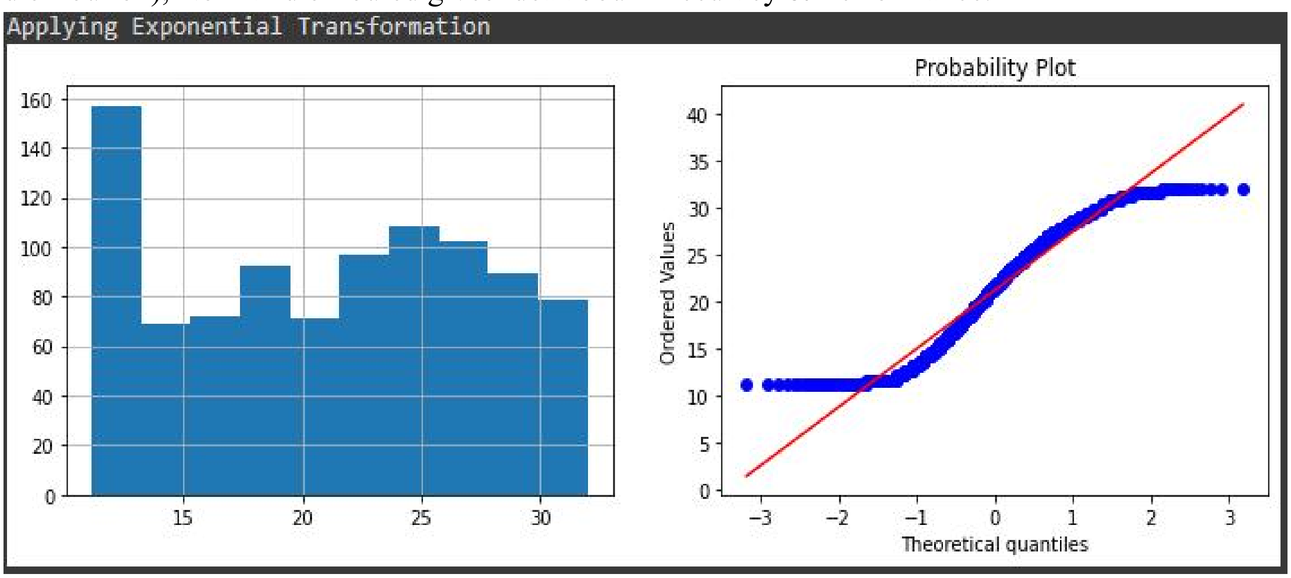
6.3.1 Q-Q plot:

- To check whether feature is Gaussian/Normal distributed.
- If Q-Q plot falls in a straight line then it is Gaussian/Normal distributed.
- Only Age & BMI are the numerical features.

QQ plot for Age -



- QQ Plot line is not straight i.e. Age is not normally distributed (have to convert it into normal distribution), Normal distributed gives idea about--Accuracy & Performance.

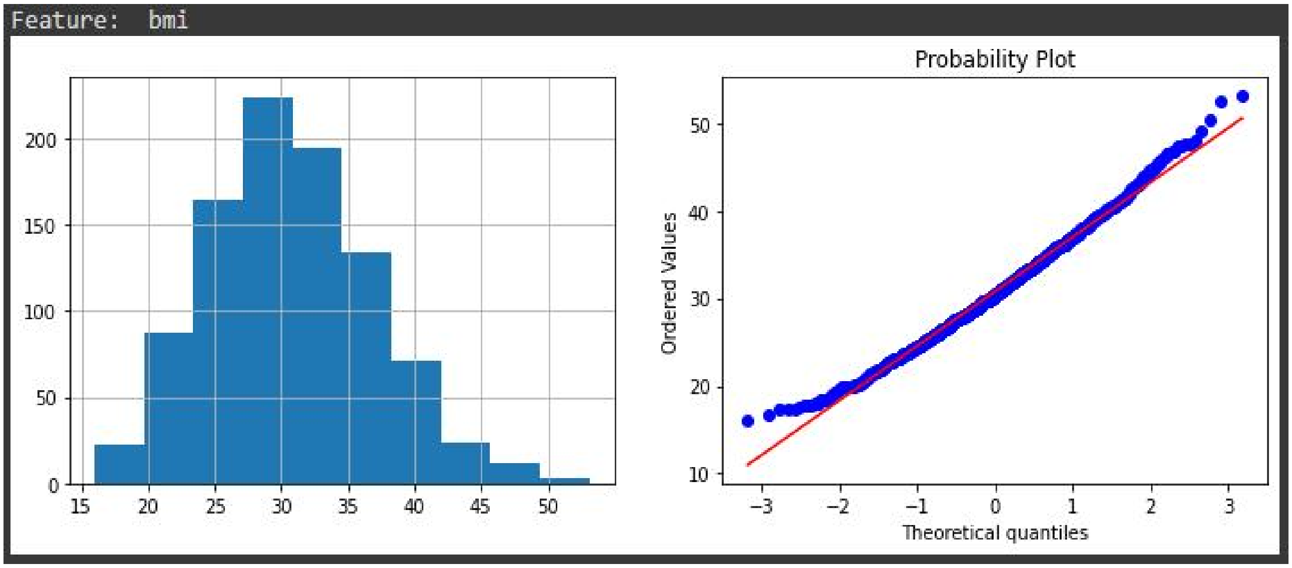


- QQ plot line is not a straight line for any of the transformation techniques for Age feature, however Exponential Transformation is the best result among all, hence Age feature will be replaced by this.

```
# Replacing feature Age with Age_exponential
X_train.age = X_train_copy.Age_exponential
```

QQ plot for BMI-

- QQ plot line is almost straight, however there are a little bit **skewness (0.3411)**, the same seen in EDA as outlier. Since skewness value is less than 0.5, hence its overall impact is low i.e. it can be ignored.



6.3.2 Standardization:

- Standardization comes into picture when features of the input data set have large differences between their ranges or simply when they are measured in different measurement units. e.g.- Pounds, Meters, Miles … etc.
- Need to bring, all the variables or features to a similar scale.
- Standardization means centering the variable at zero.

$$Z = \frac{(x-\bar{x})}{std.dev} \quad ; \ \bar{x} - \text{mean}$$

- **Applying Standardization -**
- Only transform apply for test data since whatever parameters applied to fit for train data, same needs to be applied on test data to overcome overfitting.

```
# from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()        # Initialize
X_train= scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)
```

CHAPTER 7
Performance Metrics

7. Performance metrics:

Performance metrics are the measure of the well generalized model. If the model is 100% efficient then it will lead to the biased problem i.e. **overfitting** and **underfitting**. It is not only necessary to obtain the accuracy on training data, but also vital to get the approximate result on unseen data otherwise Model is not useful. Hence, to build and deploy a well generalized model, need to evaluate the model on different metrics which helps to better optimize the performance, fine-tune it, and obtain a better result.

As per the problem statement, we need to predict **Expenses** i.e. labeled data, we can go with the supervised machine learning techniques. It is a Regression problem.

- For Regression problem performance metrics are -
  - MSE, RMSE, MAE
  - R2, Adjusted R2

7.1 MSE: Mean Squared Error

MSE is given by the squared difference between actual and predicted value.

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Squared to avoid the cancellation of negative terms and it is the benefit of MSE.
- However, MSE penalizes the model for making large error by squaring them and also it is not robust to the outliers.

7.2 RMSE: Root Mean Squared Error

RMSE is given by the simple square root of MSE.

$$\mathbf{RMSE} = \sqrt{\mathbf{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- It overcomes the MSE disadvantage i.e. large error by squaring.
- However, it is also not robust to the outliers.

7.3 MAE: Mean Absolute Error

MAE is given the absolute difference between actual and predicted values.

$$\mathbf{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

- It solves the problem with MSE, it is robust to the outliers as compared to the MSE/RMSE.
- However, the computation is difficult.

7.4 R2 Score: R Squared

R2 Score is a metric that tells the performance of the model, not the loss in an absolute sense that how well did the model perform. It is given by -

$$\mathbf{R2} = 1 - \left(\frac{SSR}{SSM}\right)$$

SSR - Squared sum residuals/error
SSM - Squared sum error of mean

- **R2 score** lies between **0 to 1**, as **close to 1**, **better the model**.
- As new features added, R2 score usually increases, even if the new features do not affect the target/dependent feature then also R2 score increases, it is the disadvantage.

7.5 Adjusted R2 Score: Adjusted R Squared

Adjusted R2 Score given by -

$$\mathbf{Adjusted\ R2} = 1 - \left(\frac{(1-R2)(N-1)}{N-P-1}\right)$$

N- Number of total sample
P- Number of independent features

- Adjusted R2 score overcomes the disadvantage of R2 score i.e. Adjusted R2 score increases only when independent feature is significant and affect the dependent feature.
- Adjusted R2 score will **decrease** only **when** added features are **not correlated**.
- Adjusted R2 score always **less than or equals to the R2 score**.
- If a relevant feature added then the **R2 score** will **increase** and **1-R2** will **decrease** heavily and the **denominator** will **also decrease** so the complete term decreases, and on subtracting from one, the **score increases**. Hence, this metric becomes one of the most important metrics to use during the evaluation of the model.

7.6 Conclusion:

For the case study, performance metrics RMSE, MAE & R2 score selected.

- RMSE & MAE to see outlier impact and compare it.
- In feature engineering, not creating huge number of new features, hence using R2 score as performance metrics.

CHAPTER 8
Model Building

8. Model Building:

Built models for Regression problem- Linear Regression, SVM Regressor, Decision Tree Regressor and Ensemble techniques models i.e. Random Forest Regressor, Gradient Boosting Regressor & StackingRegressor.

8.1 Summarizing Models Result -

Models Result :

Model	RMSE	MAE	R2_Score
LinearRegression	5790.867	4029.379	78.97
SupportVectorRegression	13160.977	8685.885	-8.62
DecisionTreeRegressor	6658.829	3123.071	72.2
RandomForestRegressor	4714.014	2780.685	86.07
GradientBoostingRegressor	4285.81	2603.198	88.48
StackingRegressor	4743.886	2676.99	85.89

- ➢ As comparing RMSE & MAE, MAE giving less error i.e. handling outlier in a better way compared to the RMSE.
- ➢ The above R2_Score/accuracy is not the actual accuracy, this may be overfitting resultant. For correct accuracy, need to check Cross_Validation_Score.

8.2 Cross_Validation_Score -

Cross_validation_score :

Model	cross_val_score
LinearRegression	72.05
SupportVectorRegression	-9.91
DecisionTreeRegressor	65.45
RandomForestRegressor	80.47
GradientBoostingRegressor	83.35
StackingRegressor	80.93

- ➢ Best model is the one, which have minimum difference between R2_Score/Accuracy_score and Cross_val_score.
- ➢ SVR giving negative result, hence ignoring this model.

8.3 Best Model -

```
Best Model = minimum(R2_Score - cross_val_score)
+---------------------------+---------------------------+
|           Model           | R2_Score - cross_val_score |
+---------------------------+---------------------------+
|      LinearRegression     |            6.92           |
|   DecisionTreeRegressor   |            6.75           |
|   RandomForestRegressor   |            5.6            |
| GradientBoostingRegressor |            5.13           |
|     StackingRegressor     |            4.96           |
+---------------------------+---------------------------+
```

- minimum(R2_Score - cross_val_score) = 4.96 i.e. **StackingRegressor** is the **best model**.

8.4 Overall Summary -

```
Overall Summary:
+---------------------------+----------+-----------------+----------------------------+
|           Model           | R2_Score | cross_val_score | R2_Score - cross_val_score |
+---------------------------+----------+-----------------+----------------------------+
|      LinearRegression     |  78.97   |      72.05      |            6.92            |
|   DecisionTreeRegressor   |   72.2   |      65.45      |            6.75            |
|   RandomForestRegressor   |  86.07   |      80.47      |            5.6             |
| GradientBoostingRegressor |  88.48   |      83.35      |            5.13            |
|     StackingRegressor     |  85.89   |      80.93      |            4.96            |
+---------------------------+----------+-----------------+----------------------------+
```

- **StackingRegressor** is the best model with the **Cross_validation_score 80.93 %.**

8.5 Hyper-parameters tuning -

Selecting parameters -

```
{'criterion': ['squared_error', 'absolute_error', 'poisson'],
 'max_depth': [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 3, 5, 10, 50],
 'min_samples_split': [2, 3, 5, 10, 15],
 'n_estimators': [250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500]}
```

- Using RandomizedSearchCV tuning technique, it is faster compared to the GridSearchCV.

Applying hyperparameters on RandomForestRegressor -

```
# from sklearn.model_selection import RandomizedSearchCV
RF_tune= RandomForestRegressor()
RF_Randomized_Search= RandomizedSearchCV(RF_tune,
                                         param_distributions= parameters,
                                         verbose= 2, cv=5, random_state=42, n_jobs=-1)

RF_Randomized_Search.fit(X_train, y_train)
```

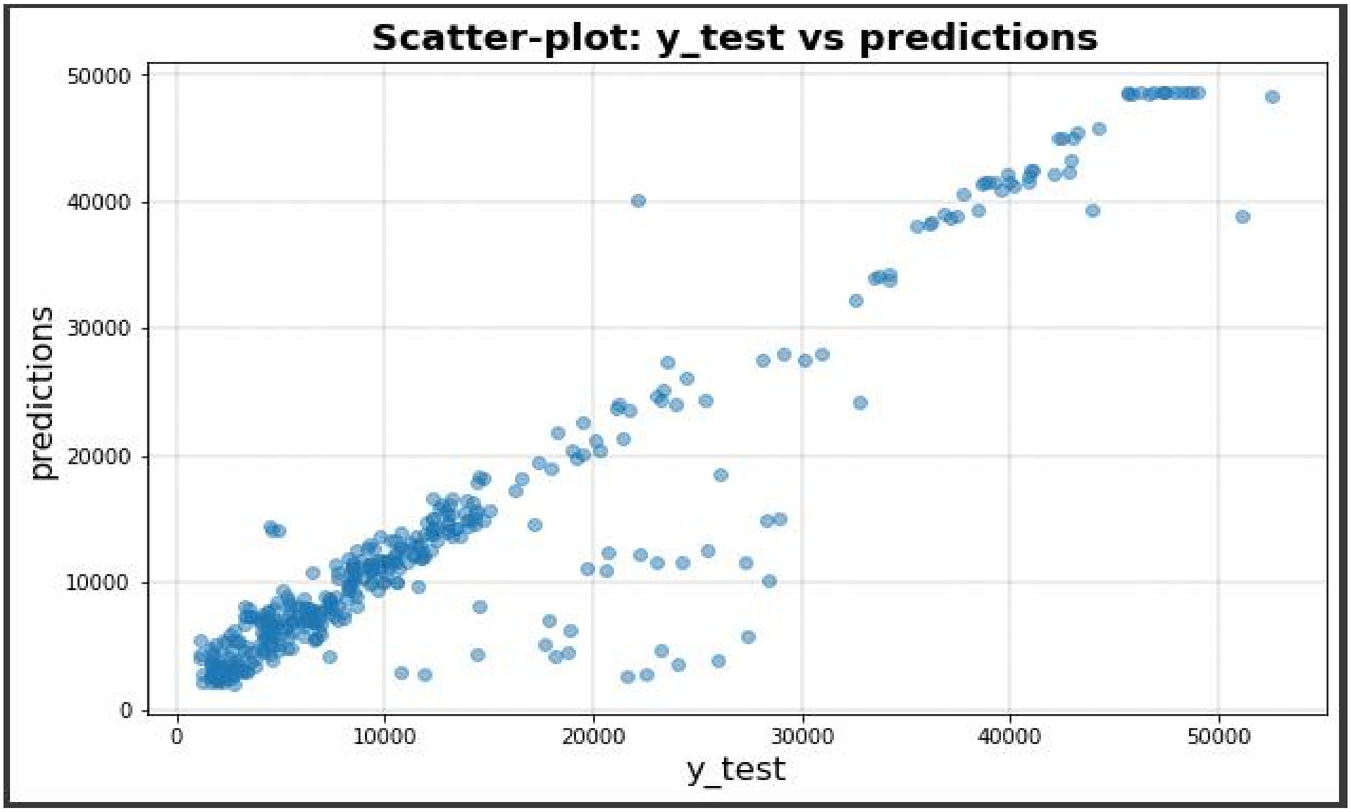
Best_tuned_model -

```
Stack_Regressor_Best_tuned = StackingRegressor(estimators= [('Model_Linear', Model_Linear),
                                                            ('Model_SVR', Model_SVR),
                                                            ('Model_DT', Model_DT),
                                                            ('Model_RF', Model_RF),
                                                            ('Model_GB', Model_GB)],
                                               final_estimator= RandomForestRegressor(max_depth=24,
                                                                                      min_samples_leaf=10,
                                                                                      min_samples_split=5,
                                                                                      n_estimators=1750,
                                                                                      random_state=42),
                                               verbose= 2,
                                               cv=5,
                                               n_jobs=-1)

Stack_Regressor_Best_tuned.fit(X_train, y_train)            # train data .fit()
```

- The Cross_validation_score was 80.93% earlier, now **performing hyper-parameter tuning using RandomizedSearchCV Cross_validation_score improved to 83.21%** i.e. 2.28% performance increased.

8.6 Final Result – Scatter-plot : y_test vs predictions -



➢ Almost linear, suitable model to deploy.

8.7 Serialization - Saving Model :

➢ Saving a model using the pickle module is also called **serialization**. When a model is serialized, it can be reused to make predictions. And **deserialization** means loading or reading the model to reuse it.

```
# import pickle

model = Stack_Regressor_Best_tuned
# open a file, where need to store the data
file = open('/content/drive/MyDrive/Colab Notebooks/Case_Study_1/Stack_Reg_model.pkl', 'wb')

# dump to file
pickle.dump(model, file)
```

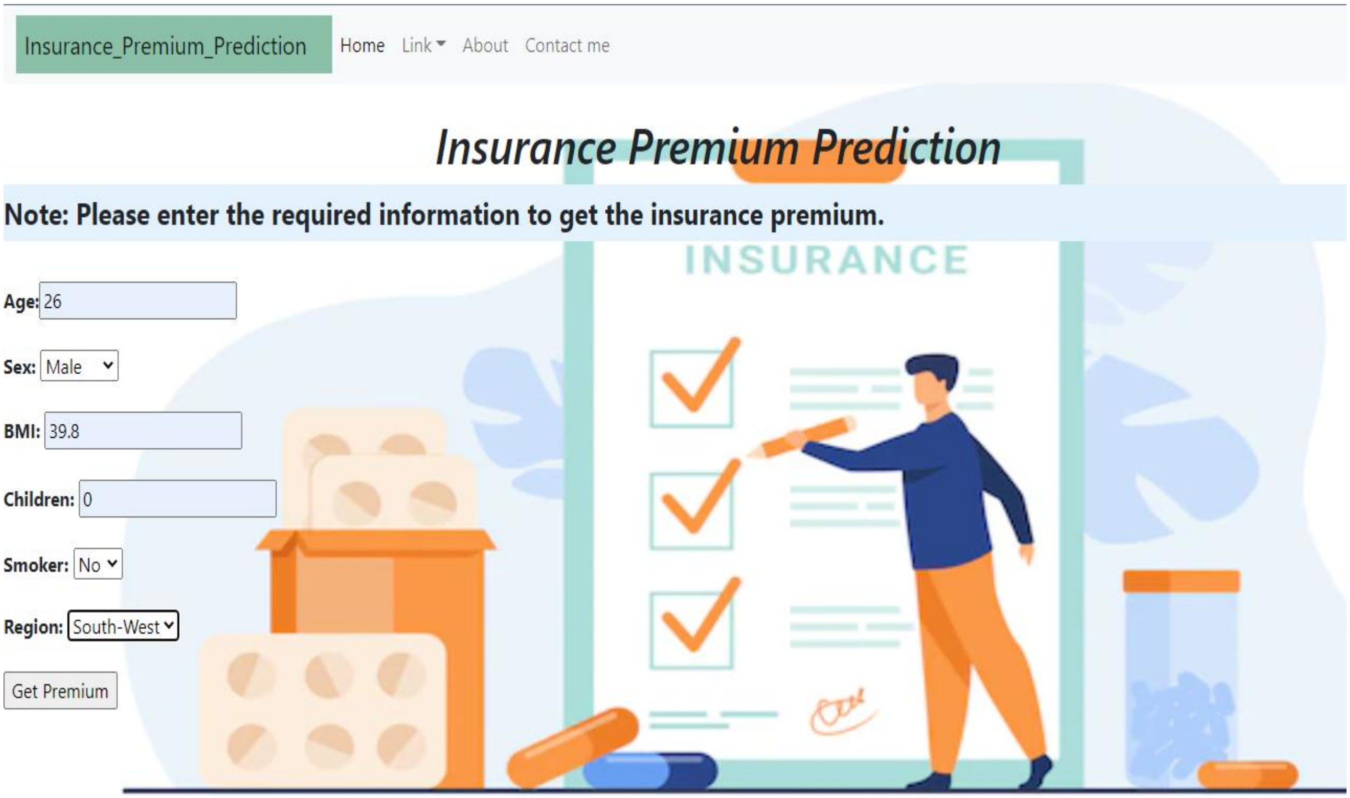
Now the training part has been completed.

CHAPTER 9
Model Deployment for Productionisation

9. Model Deployment :

After the training part, created a **User Interface web application** using **Flask-API & HTML** and deployed the model for productionisation using **Heroku & AWS**.

9.1 Project Demo :



➢ User will enter the required values & hit Get Premium, and it will show the premium as below.

References:

Definitions-- Wikipedia

HTML templates-- [Bootstrap (getbootstrap.com)](getbootstrap.com)

Created by *Vikram Singh*

GitHub Profile : VkasRajpurohit (github.com)

Code : VkasRajpurohit/Insurance_Premium_Prediction (github.com)

---------- **End of Project Report** ----------

Thank You !

...