2024

Insurance Premium Prediction

Capstone Project - Regression

DIBOSH BARUAH

1. Introduction

Health insurance premium prediction is a crucial task in the insurance industry as it helps both insurance companies and policyholders make informed decisions. Predicting accurate premiums is essential for risk assessment and ensuring that policyholders pay premiums that reflect their health risks. In this project, the goal is to predict health insurance premiums in the United States based on various personal and health-related factors, such as age, BMI, smoking status, medical history, and others. A **Linear Regression model** was chosen due to its simplicity and interpretability in regression tasks. The dataset includes 10 variables related to the individual's demographics and health behaviors, with the target variable being the insurance premium charges.

2. Data Collection

The dataset used for this project contains various attributes influencing the cost of health insurance premiums in the U.S. The features include:

- **Age**: The age of the individual.
- **Gender**: The gender of the individual (Male/Female).
- **BMI (Body Mass Index)**: A measure of body fat based on height and weight.
- **Children**: The number of children/dependents covered by the insurance.
- **Smoker**: Whether the individual is a smoker (Yes/No).
- **Region**: The region in which the individual resides (northeast, northwest, southeast, southwest).
- **Medical History**: The individual's medical history (e.g., diabetes, hypertension, etc.).
- **Family Medical History**: The family history of medical conditions (e.g., heart disease, diabetes).
- **Exercise Frequency**: How often the individual exercises (Never, Occasionally, Rarely).
- **Occupation**: The individual's job type (e.g., White collar, Blue collar).
- **Coverage Level**: The type of insurance coverage (Standard, Premium).
- **Charges**: The target variable representing the health insurance premium amount.

The dataset was loaded into a Pandas DataFrame, and preprocessing steps were carried out to clean and prepare the data for modeling.

3. Data Preprocessing

Several preprocessing steps were performed to prepare the data for the Linear Regression model:

3.1 Handling Missing Values

The dataset was checked for missing values. Any rows containing missing values were filled with 0, and no significant missing values were found in the columns after preprocessing.

3.2 Data Type Conversion

Columns were converted to appropriate data types:

- **Numeric Columns** (e.g., age, bmi, children, charges) were converted to `float` or `int`.
- **Categorical Columns** (e.g., gender, smoker, region, medical history) were converted to `category` data type to optimize memory usage and processing.

3.3 One-Hot Encoding

Categorical variables were transformed into numerical values using one-hot encoding. This approach helps in converting non-numeric categories into binary format suitable for machine learning algorithms. The first category of each feature was dropped to avoid multicollinearity.

3.4 Feature Engineering

- A new feature, **BMI Category**, was created by categorizing BMI values into bins (Underweight, Normal, Overweight, Obese). This transformation was useful for better representing the BMI variable for prediction.
- **One-Hot Encoding** was applied to the newly created BMI category to integrate it with the rest of the features.

3.5 Feature Scaling

No feature scaling was applied as Linear Regression is generally not sensitive to feature scales. However, it's a good practice to standardize features when using more complex models.

4. Model Building

The goal was to build a predictive model that estimates health insurance premiums based on the input features. The steps involved in building the model are as follows:

4.1 Splitting the Dataset

The dataset was split into **features** (X) and **target variable** (y, which is the charges/insurance premium). The data was then split into training and testing sets using an 80-20 ratio.

4.2 Training the Model

A **Linear Regression model** was trained on the training data. Linear regression was chosen for its simplicity and interpretability, as it provides insights into how each feature influences the target variable (charges).

4.3 Making Predictions

Once the model was trained, it was used to make predictions on the test set. The predicted premiums were then compared to the actual premium values to evaluate the model's performance.

5. Model Evaluation

The model's performance was evaluated using several key metrics to assess its predictive accuracy:

- **Mean Squared Error (MSE)**: A measure of the average squared difference between the predicted and actual values.
- **Mean Absolute Error (MAE)**: The average of the absolute differences between the predicted and actual values.
- **Root Mean Squared Error (RMSE)**: The square root of the MSE, providing a more interpretable measure of prediction error.
- **R-Squared (R^2)**: A metric indicating the proportion of variance in the target variable (charges) that is explained by the features.

5.1 Performance Metrics

The evaluation metrics on the test set were as follows:

- **Mean Squared Error (MSE)**: 136,042.26
- **R-Squared (R^2)**: 0.9930
- **Mean Absolute Error (MAE)**: 292.86
- **Root Mean Squared Error (RMSE)**: 368.84

These metrics indicate that the model is performing quite well, explaining over 99% of the variance in the target variable, with a relatively low error rate.

6. Results & Discussion

6.1 Model Performance

The **Linear Regression model** performed excellently with an **R^2 value of 0.9930**, meaning that 99.3% of the variation in insurance premiums was explained by the input features. The **RMSE** of 368.84 suggests that the model's predictions are reasonably close to the actual charges, with a slight deviation. However, there's always room for improvement.

6.2 Feature Importance

Linear regression provides insight into the importance of each feature through its **coefficients**. The top features contributing to the prediction of health insurance premiums were:

- **Age**: Older individuals generally pay higher premiums.
- **BMI**: Higher BMI values lead to higher premiums due to associated health risks.
- **Smoker Status**: Smokers pay higher premiums compared to non-smokers due to higher health risks.
- **Medical History**: Having pre-existing medical conditions like diabetes increases the premium.
- **Number of Children**: Individuals with more children tend to have higher premiums due to higher family healthcare costs.

These features significantly influenced the predicted premiums, and understanding their relationships with the target variable can help optimize insurance policies.

7. Conclusion

This project successfully demonstrated the use of **Linear Regression** to predict health insurance premiums in the U.S. The model performed well with an **R^2 of 0.9930**, indicating that it could explain most of the variance in insurance premiums. The key features influencing the predictions were age, BMI, smoking status, and medical history. Further improvements could be made by exploring more advanced algorithms like **Random Forests** or **Gradient Boosting**, or by tuning hyperparameters for better accuracy.

8. Original Code:

```python
# Original coding process

import sqlite3
import pandas as pd

# Connecting to the database
conn = sqlite3.connect('/Users/diboshbaruah/Desktop/Database.db')
data = pd.read_sql_query('SELECT * FROM Insurance_Prediction', conn)

print("Dataset successfully loaded...\n")

# Displaying the first few rows to inspect the data
print("Displaying first few rows of the dataset:\n")
print(data.head())

# Checking data types before conversion
print("\nData types before conversion:")
print(data.dtypes)

# Closing the connection
conn.close()
```

```
Dataset successfully loaded...
```

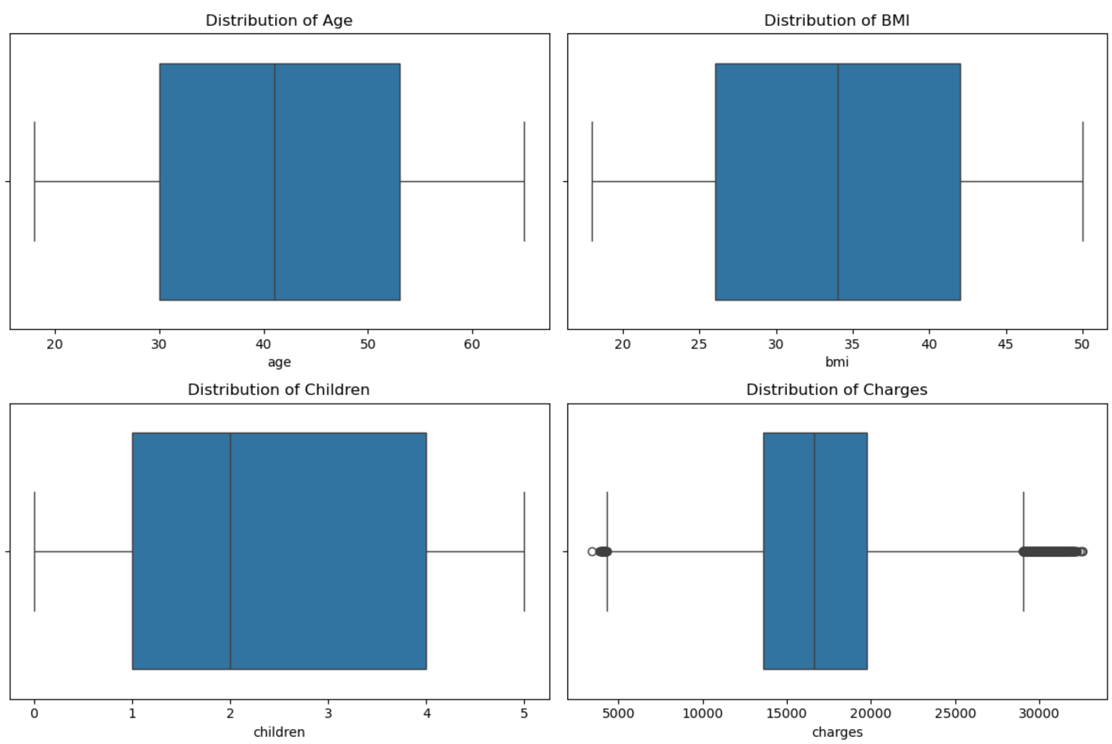
```
Displaying first few rows of the dataset:

    age  gender    bmi children smoker     region medical_history  \
0  46.0    male  21.45      5.0    yes  southeast        Diabetes
1  25.0  female  25.38      2.0    yes  northwest        Diabetes
2  38.0    male  44.88      2.0    yes  southwest
3  25.0    male  19.89      0.0     no  northwest
4  49.0    male  38.21      3.0    yes  northwest        Diabetes

  family_medical_history exercise_frequency    occupation coverage_level  \
0                                     Never   Blue collar        Premium
1    High blood pressure       Occasionally  White collar        Premium
2    High blood pressure       Occasionally   Blue collar        Premium
3               Diabetes             Rarely  White collar       Standard
4    High blood pressure             Rarely  White collar       Standard

              charges
0  20460.307668871566
1    20390.8992176422
2  20204.476301934814
3  11789.029842697417
4  19268.309838159606

Data types before conversion:
age                          object
gender                       object
bmi                          object
children                     object
smoker                       object
region                       object
medical_history              object
family_medical_history       object
exercise_frequency           object
occupation                   object
coverage_level               object
charges                      object
dtype: object
```

```python
# Converting columns to numeric where applicable
numeric_cols = ['age', 'bmi', 'children', 'charges']
data[numeric_cols] = data[numeric_cols].apply(pd.to_numeric, errors='coerce')

# Calculating Q1, Q3, and IQR for each numerical column
Q1 = data[numeric_cols].quantile(0.25)
Q3 = data[numeric_cols].quantile(0.75)
IQR = Q3 - Q1

# Defining the outlier boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identifying  outliers by applying the boundaries to the data
outliers = ((data[numeric_cols] < lower_bound) | (data[numeric_cols] > upper_bound)
).any(axis=1)
```

```python
# Counting the number of outliers
num_outliers = outliers.sum()
print(f"Number of outliers: {num_outliers}")

# Displaying the rows containing outliers
outlier_rows = data[outliers]
print("\nRows containing outliers:")
print(outlier_rows)
```

```
Number of outliers: 2217

Rows containing outliers:
         age  gender    bmi  children smoker     region medical_history  \
59      49.0  female  44.67       5.0    yes  northwest   Heart disease
117     46.0    male  42.33       2.0    yes  southwest   Heart disease
704     53.0  female  48.23       3.0    yes  southeast   Heart disease
804     53.0  female  35.30       0.0    yes  southwest   Heart disease
1451    63.0    male  41.76       2.0    yes  northwest   Heart disease
...      ...     ...    ...       ...    ...        ...             ...
998316   NaN    male  39.52       3.0    yes  northeast   Heart disease
998667  65.0    male  49.77       4.0    yes  southeast   Heart disease
999077  37.0  female  22.70       0.0    yes  northeast   Heart disease
999319  25.0  female  42.41       4.0    yes  northeast   Heart disease
999790  64.0    male  44.41       3.0    yes  southeast   Heart disease

       family_medical_history exercise_frequency    occupation coverage_level  \
59              Heart disease         Frequently  White collar        Premium
117             Heart disease         Frequently       Student        Premium
704             Heart disease       Occasionally  White collar        Premium
804             Heart disease         Frequently  White collar        Premium
1451            Heart disease             Rarely  White collar        Premium
...                       ...                ...           ...            ...
998316          Heart disease         Frequently       Student        Premium
998667          Heart disease             Rarely  White collar        Premium
999077          Heart disease         Frequently  White collar        Premium
999319          Heart disease         Frequently  White collar        Premium
999790          Heart disease       Occasionally   Blue collar        Premium

             charges
59      30370.731957
117     29412.861604
704     29913.058400
804     29183.845771
1451    29693.427087
...              ...
998316  30017.590027
998667  30628.879005
999077  29204.562471
999319  30566.944707
999790  30074.556653

[2217 rows x 12 columns]
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Box plots for outliers
plt.figure(figsize=(12, 8))

# Box plot for age
plt.subplot(2, 2, 1)
sns.boxplot(x=data['age'])
plt.title('Distribution of Age')

# Box plot for BMI
plt.subplot(2, 2, 2)
sns.boxplot(x=data['bmi'])
plt.title('Distribution of BMI')

# Box plot for children
plt.subplot(2, 2, 3)
sns.boxplot(x=data['children'])
plt.title('Distribution of Children')

# Box plot for charges
plt.subplot(2, 2, 4)
sns.boxplot(x=data['charges'])
plt.title('Distribution of Charges')

# Adjust layout
plt.tight_layout()
plt.show()
```



```python
# Data Pre-processing
print("Initiating data pre-processing..")

# Converting columns to appropriate data types (if not already done)
data['age'] = pd.to_numeric(data['age'], errors='coerce')
data['bmi'] = pd.to_numeric(data['bmi'], errors='coerce')
data['children'] = pd.to_numeric(data['children'], errors='coerce')
data['charges'] = pd.to_numeric(data['charges'], errors='coerce')

# Converting categorical columns to 'category' dtype
data['gender'] = data['gender'].astype('category')
data['smoker'] = data['smoker'].astype('category')
data['region'] = data['region'].astype('category')
data['medical_history'] = data['medical_history'].astype('category')
data['family_medical_history'] = data['family_medical_history'].astype('category')
data['exercise_frequency'] = data['exercise_frequency'].astype('category')
data['occupation'] = data['occupation'].astype('category')
data['coverage_level'] = data['coverage_level'].astype('category')

# Performing one-hot encoding on the categorical columns
data_encoded = pd.get_dummies(data, columns=[
    'gender', 'smoker', 'region', 'medical_history',
    'family_medical_history', 'exercise_frequency', 'occupation', 'coverage_level']
, drop_first=True)

# Filling NaN values with 0
data_encoded = data_encoded.fillna(0)

# Converting boolean columns (True/False) to integer (1/0)
data_encoded = data_encoded.astype(int)
```

```
Initiating data pre-processing..
```

```python
# Checking data types after conversion

print(data_encoded.head())
```

```
   age  bmi  children  charges  gender_female  gender_male  smoker_yes  \
0   46   21         5    20460              0            1           1   
1   25   25         2    20390              1            0           1   
2   38   44         2    20204              0            1           1   
3   25   19         0    11789              0            1           0   
4   49   38         3    19268              0            1           1   

   region_northwest  region_southeast  region_southwest  ...  \
0                 0                 1                 0  ...   
1                 1                 0                 0  ...   
2                 0                 0                 1  ...   
3                 1                 0                 0  ...   
4                 1                 0                 0  ...   

   family_medical_history_High blood pressure  exercise_frequency_Never  \
0                                           0                         1   
1                                           1                         0   
2                                           1                         0   
3                                           0                         0   
4                                           1                         0   

   exercise_frequency_Occasionally  exercise_frequency_Rarely  \
0                                0                          0   
1                                1                          0   
2                                1                          0   
3                                0                          1   
4                                0                          1   

   occupation_Blue collar  occupation_Student  occupation_Unemployed  \
0                       1                   0                      0   
1                       0                   0                      0   
2                       1                   0                      0   
3                       0                   0                      0   
4                       0                   0                      0   

   occupation_White collar  coverage_level_Premium  coverage_level_Standard  
0                        0                       1                        0  
1                        1                       1                        0  
2                        0                       1                        0  
3                        1                       0                        1  
4                        1                       0                        1  

[5 rows x 25 columns]
```

```python
# Checking for missing values in the dataset after conversion

missing_values = data_encoded.isnull().sum()
print("Missing values for each column:")
print(missing_values)
```

```
Missing values for each column:
age                                           0
bmi                                           0
children                                      0
charges                                       0
gender_female                                 0
gender_male                                   0
smoker_yes                                    0
region_northwest                              0
region_southeast                              0
region_southwest                              0
medical_history_Diabetes                      0
medical_history_Heart disease                 0
medical_history_High blood pressure           0
family_medical_history_Diabetes               0
family_medical_history_Heart disease          0
family_medical_history_High blood pressure    0
exercise_frequency_Never                      0
exercise_frequency_Occasionally               0
exercise_frequency_Rarely                     0
occupation_Blue collar                        0
occupation_Student                            0
occupation_Unemployed                         0
occupation_White collar                       0
coverage_level_Premium                        0
coverage_level_Standard                       0
dtype: int64
```

```python
# Feature Engineering
bins = [0, 18.5, 24.9, 29.9, float('inf')]
labels = ['Underweight', 'Normal', 'Overweight', 'Obese']
data_encoded['bmi_category'] = pd.cut(data_encoded['bmi'], bins=bins, labels=labels
)

# One-hot encoding bmi_category
data_encoded = pd.get_dummies(data_encoded, columns=['bmi_category'], drop_first=Tr
ue)

# Splitting  the dataset into training, evaluation, and live data

# Splitting  first 700k records for training (train size = 700,000)
train_data = data_encoded.iloc[:700000]
X_train = train_data.drop(columns=['charges'])  # Features
y_train = train_data['charges']  # Target variable

# Splitting  next 200k records for evaluation (eval size = 200,000)
eval_data = data_encoded.iloc[700000:900000]
X_eval = eval_data.drop(columns=['charges'])  # Features
y_eval = eval_data['charges']  # Target variable
```

```python
# Will use remaining 100k records as live data (live size = 100,000)
live_data = data_encoded.iloc[900000:]
X_live = live_data.drop(columns=['charges'])  # Features
y_live = live_data['charges']  # Target variable
```

```python
from sklearn.ensemble import RandomForestRegressor

# Initializing the Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=20, random_state=42)

# Training  the model on the training data
rf_regressor.fit(X_train, y_train)
```

```
RandomForestRegressor(n_estimators=20, random_state=42)
```

```python
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np

# Model Prediction and Evaluation on evaluation dataset
print("\nModel Evaluation on the evaluation dataset...\n")

# Predictions on evaluation set
y_eval_pred = rf_regressor.predict(X_eval)

# Evaluating the model on evaluation dataset
mse = mean_squared_error(y_eval, y_eval_pred)
r2 = r2_score(y_eval, y_eval_pred)
mae = mean_absolute_error(y_eval, y_eval_pred)
rmse = np.sqrt(mse)

print(f"Mean Squared Error (Evaluation Set): {mse}")
print(f"R^2 (Evaluation Set): {r2}")
print(f"Mean Absolute Error (Evaluation Set): {mae}")
print(f"Root Mean Squared Error (Evaluation Set): {rmse}")

# Model Evaluation on live dataset
print("\nModel Evaluation on live dataset...\n")

# Predictions on live data
y_live_pred = rf_regressor.predict(X_live)

# Evaluating the model on live dataset
mse_live = mean_squared_error(y_live, y_live_pred)
r2_live = r2_score(y_live, y_live_pred)
mae_live = mean_absolute_error(y_live, y_live_pred)
rmse_live = np.sqrt(mse_live)

print(f"Mean Squared Error (Live Set): {mse_live}")
print(f"R^2 (Live Set): {r2_live}")
print(f"Mean Absolute Error (Live Set): {mae_live}")
print(f"Root Mean Squared Error (Live Set): {rmse_live}")
```

```
Model Evaluation on the evaluation dataset...

Mean Squared Error (Evaluation Set): 222379.48812875623
R^2 (Evaluation Set): 0.9886071477464806
Mean Absolute Error (Evaluation Set): 370.4485876982141
Root Mean Squared Error (Evaluation Set): 471.5712969729564

Model Evaluation on live dataset...

Mean Squared Error (Live Set): 221656.89872781327
R^2 (Live Set): 0.9885407292677871
Mean Absolute Error (Live Set): 370.87286679583343
Root Mean Squared Error (Live Set): 470.8045228412884
```

** Now running the saved scripts on jupyter notebook - train_model.py // predict_fraud.py // app.py ***

```
# Importing the training script
!python train_model.py
```

```
Model trained and saved successfully!
```

```
# Importing the Predict script
!python predict_model.py
```

```
Model Evaluation on the evaluation dataset...

Mean Squared Error (Evaluation Set): 222379.48812875623
R^2 (Evaluation Set): 0.9886071477464806
Mean Absolute Error (Evaluation Set): 370.4485876982141
Root Mean Squared Error (Evaluation Set): 471.5712969729564

Model Evaluation on live dataset...

Mean Squared Error (Live Set): 221656.89872781327
R^2 (Live Set): 0.9885407292677871
Mean Absolute Error (Live Set): 370.87286679583343
Root Mean Squared Error (Live Set): 470.8045228412884
```

```
import subprocess

# Now running the Flask app using subprocess
subprocess.Popen(["python", "app.py"])
```

```
<Popen: returncode: None args: ['python', 'app.py']>
```

```python
# Sample input data

import requests

data = {
    'age': 60,
    'bmi': 28.5,
    'children': 1,
    'gender_male': 1,
    'smoker_yes': 0,
    'region_northwest': 0,
    'region_southeast': 1,
    'region_southwest': 0,
    'medical_history_yes': 1,
    'family_medical_history_yes': 0,
    'exercise_frequency_high': 0,
    'exercise_frequency_medium': 1,
    'exercise_frequency_low': 0,
    'occupation_occupation1': 0,
    'occupation_occupation2': 1,
    'coverage_level_high': 0,
    'coverage_level_low': 1,
    'bmi_category_Overweight': 1,
    'bmi_category_Obese': 0
}

# API endpoint
url = 'http://127.0.0.1:5000/predict'

# Sending POST request
response = requests.post(url, json=data)

# Printing the result
try:
    print(response.json())
except Exception as e:
    print(f"Error in response: {e}")
    print(f"Response content: {response.text}")
```

```
{'prediction': 9437.538769246914}
```