National College of Ireland

Forecasting Medical Insurance Claim Cost with Data Mining Techniques

MSc Research Project
Data Analytics

Aditya Naresh Sahare
Student ID: X21140677

School of Computing
National College of Ireland

Supervisor: Dr. Cristina Hava Muntean

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aditya Naresh Sahare
Student ID:	X21140677
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Cristina Hava Muntean
Submission Due Date:	15/12/2022
Project Title:	Forecasting Medical Insurance Claim Cost with Data Mining Techniques
Word Count:	557
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	1st February 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Forecasting Medical Insurance Claim Cost with Data Mining Techniques

Aditya Naresh Sahare
X21140677

1 Hardware Requirements

The computer has AMD Ryzen 5 5600H Processor with Radeon Graphics, 3301 Mhz, 6 Core(s), 12 Logical Processor(s) with 8GB RAM, 512GB SSD, 4GB NVIDIA GEFORCE RTX Graphic Card.(Figure 1)

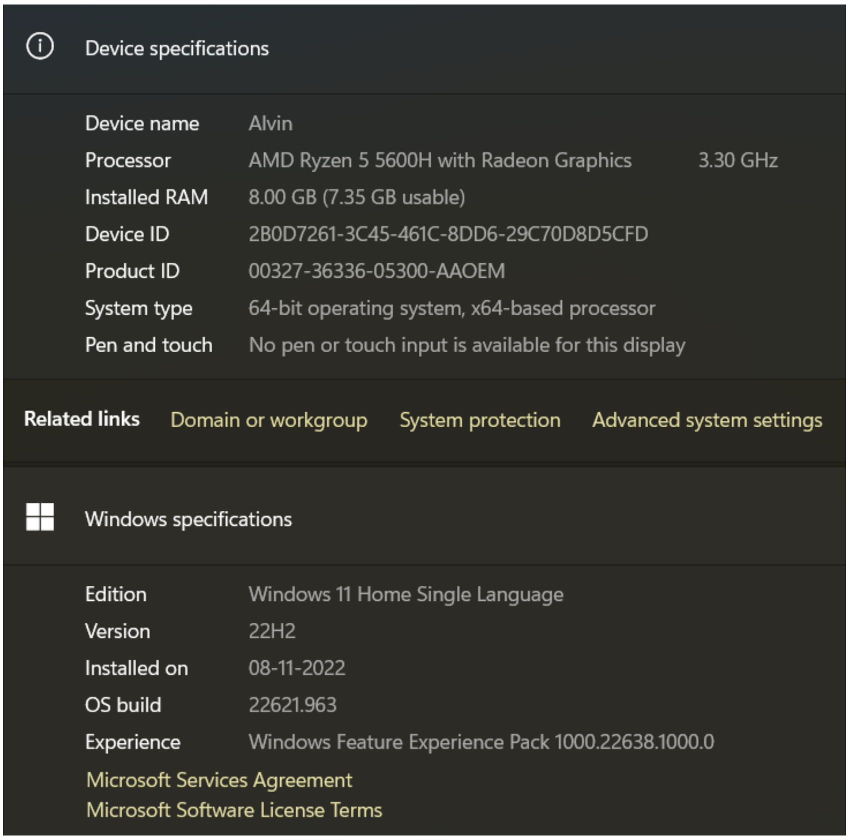

Figure 1: Hardware Requirements

2 Software Requirements

The code has been written in Python Language. Jupyter Notebook has been used which is an Integrated Development Environment(IDE) for programming. This IDE is present in Ananconda Application (Figure 2).

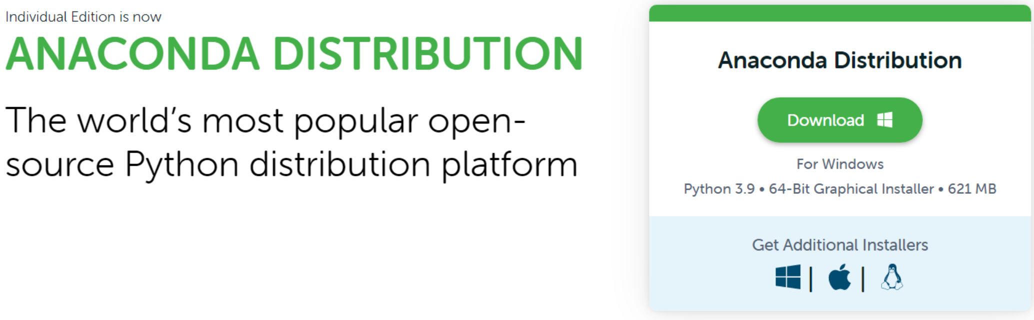

Figure 2: Anaconda navigator specification

Install this Anaconda Distribution which launches the Anaconda Navigator home. This consist of Jupyter Notebook (Figure 3).

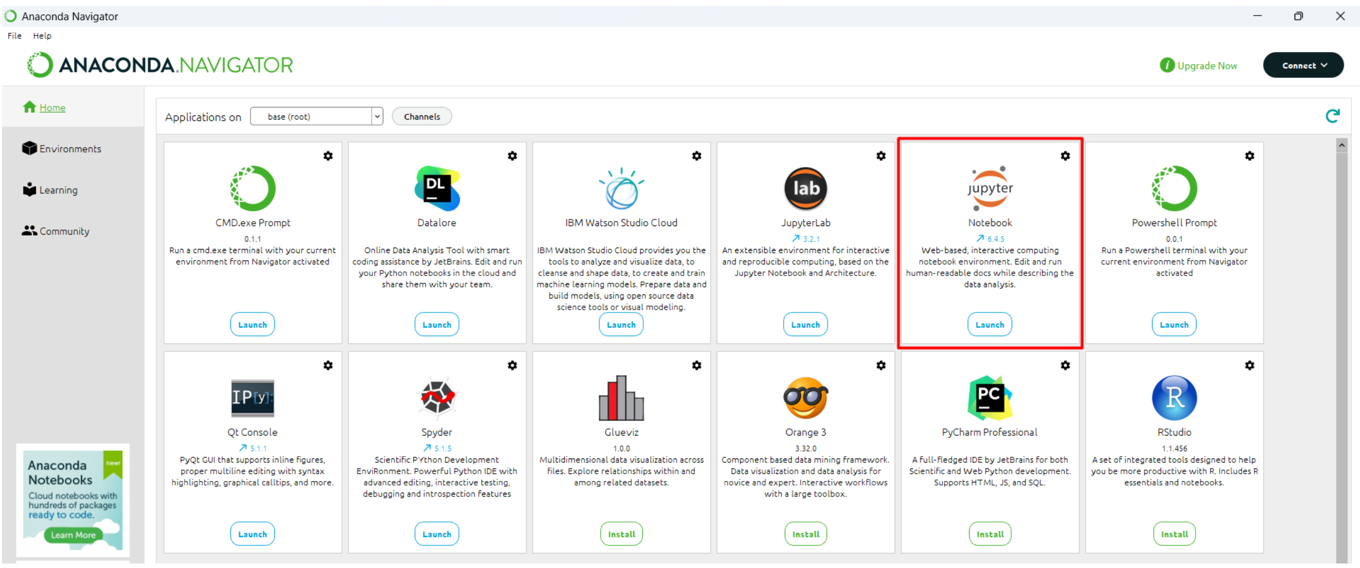

Figure 3: Anaconda navigator overview

Install Jupyter Notebook in this pack in Anaconda Navigator. The best part of Jupyter Notebook is it automatically update the system environment variables to run Python.exe

3 Libraries required for Python

Following are the libraries used to run the code. If the libraries are not found in JUpyter Notebook, then write 'pip install library name', here you can library name to required library listed below.

- numpy
- matplotlib
- pandas
- seaborn
- sklearn
- statsmodels
- scipy

4 Dataset Description

- Health insurance dataset can be found in this URL: `https://www.kaggle.com/datasets/sureshgupta/health-insurance-data-set`.
- The dataset is uploaded with the code artifacts.
- Save the daaset in the same file as Python code file and give the file name in pd.read_csv("file name") like in Figure 4

3. Read Data

```
df_insurance = pd.read_csv("health_insurance_final.csv")
df_insurance.head()
```

Figure 4: Reading the data in code

5 Data pre-processing

- In data pre-processing, the missing values are handled first.
- Exploratory Data Analysis of the dataset (Figure 6).
- Since the city variable has 91 cities, a new feature is introduced combining some cities which are from the same region.
- Hot encoding of categorical variables needs to be done for Linear Regression model

Deal with Missing Values

```
df_insurance['age'].groupby(df_insurance['sex'], axis=0).mean()
```

```
sex
female    39.361040
male      39.738395
Name: age, dtype: float64
```

The average age for the male and female is nearly the same. We will fill in missing values with the mean age of the policyholder.

```
df_insurance['age'].fillna(df_insurance['age'].mean(), inplace=True)
```

Replace missing values by mean for the BMI.

```
df_insurance['bmi'].fillna(df_insurance['bmi'].mean(), inplace=True)
```

We have seen that the the minimum bloodpressure is 0, which is absurd. It implies that these are missing values. Let us replace these missing values with the median value.

```
median_bloodpressure = df_insurance['bloodpressure'].median()
df_insurance['bloodpressure'] = df_insurance['bloodpressure'].replace(0,median_bloodpressure)
```

Figure 5: Code for Missing values

4.2.3 Dummy Encoding of Categorical Variables

To build linear regression models we use OLS method. The OLS method fails to perform in presence of categorical variables. To overcome this we use dummy encoding.

1. Filter numerical and categorical variables

```
df_numeric_features = df_insurance.select_dtypes(include=np.number)
df_numeric_features.columns
```

```
Index(['age', 'weight', 'bmi', 'no_of_dependents', 'bloodpressure', 'claim',
       'sqrt_claim'],
      dtype='object')
```

```
df_categoric_features = df_insurance.select_dtypes(include=[np.object])
df_categoric_features.columns
```

```
Index(['sex', 'hereditary diseases', 'smoker', 'state', 'diabetes',
       'regular_ex', 'job_title', 'region'],
      dtype='object')
```

2. Dummy encode the catergorical variables

```
for col in df_categoric_features.columns.values:
    dummy_encoded_variables = pd.get_dummies(df_categoric_features[col], prefix=col, drop_first=True)
    df_categoric_features = pd.concat([df_categoric_features, dummy_encoded_variables],axis=1)
    df_categoric_features.drop([col], axis=1, inplace=True)
```

Figure 8: Creating new feature "Region"

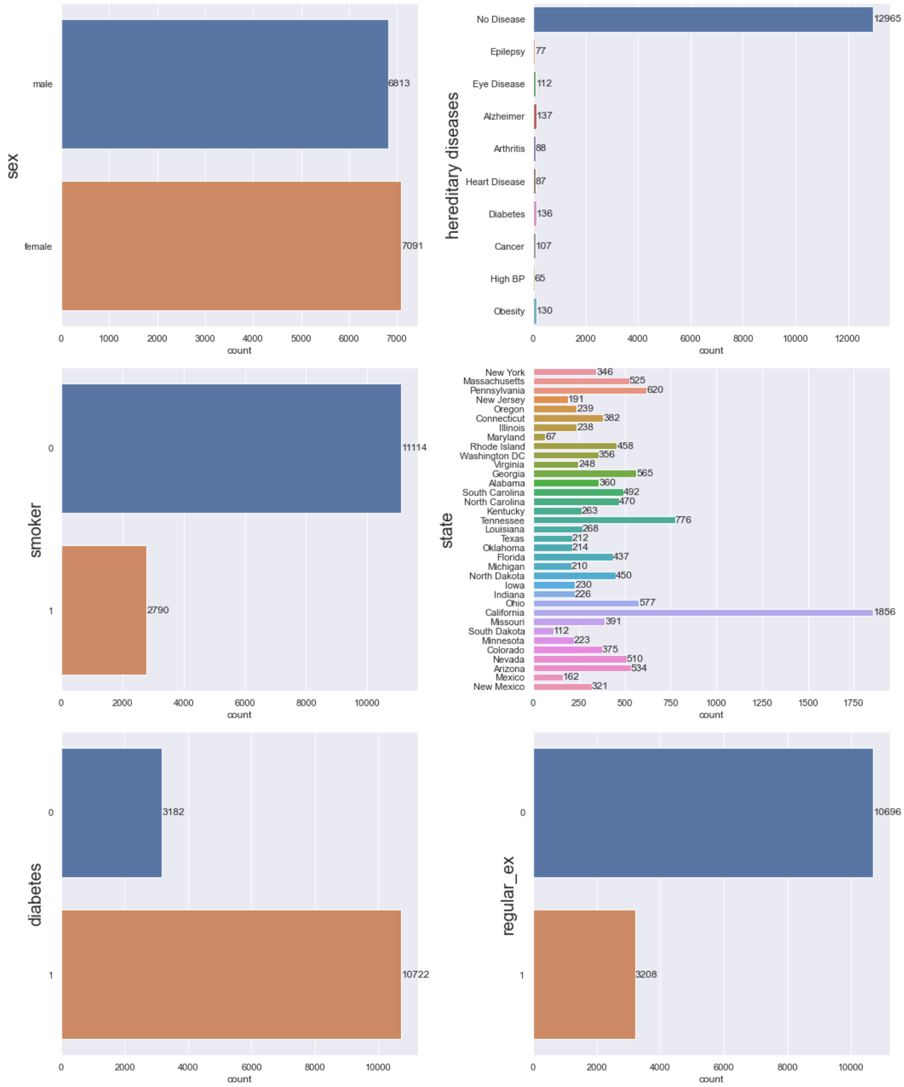

Figure 6: EDA of Numerical and Categorical values

4.1.8 Feature Engineering

Create a new feature 'region' by combining the cities.

There are 91 unique cities. We will divide these cities into North-East, West, Mid-West, and South regions.

Let's create a new variable region. We will replace the original variable `city` with it.

```
]: # create a region column and combine the north-east cities
df_insurance['region'] = df_insurance['city'].replace(['New York', 'Boston', 'Philadelphia', 'Pittsburgh', 'Buffalo',
                                                     'Atlantic City','Portland', 'Cambridge', 'Hartford',
                                                     'Springfield', 'Syracuse', 'Baltimore', 'York', 'Trenton',
                                                     'Warwick', 'Washington DC', 'Providence', 'Harrisburg',
                                                     'Newport', 'Stamford', 'Worcester'],
                                                    'North-East')
```

```
]: # combine all the southern cities into the 'region' column
df_insurance['region'] = df_insurance['region'].replace(['Atlanta', 'Birmingham', 'Charleston', 'Charlotte',
                                                       'Louisville', 'Memphis', 'Nashville', 'New Orleans',
                                                       'Raleigh', 'Houston', 'Georgia', 'Oklahoma', 'Orlando',
                                                       'Macon', 'Huntsville', 'Knoxville', 'Florence', 'Miami',
                                                       'Tampa', 'Panama City', 'Kingsport', 'Marshall'],
                                                       'Southern')
```

```
]: # combine all the mid-west cities into the 'region' column
df_insurance['region'] = df_insurance['region'].replace(['Mandan', 'Waterloo', 'Iowa City', 'Columbia',
                                                       'Indianapolis', 'Cincinnati', 'Bloomington', 'Salina',
                                                       'Kanas City', 'Brookings', 'Minot', 'Chicago', 'Lincoln',
                                                       'Falls City', 'Grand Forks', 'Fargo', 'Cleveland',
                                                       'Canton', 'Columbus', 'Rochester', 'Minneapolis',
                                                       'Jefferson City', 'Escanaba','Youngstown'],
                                                       'Mid-West')
```

```
]: # combine all the western cities into the 'region' column
df_insurance['region'] = df_insurance['region'].replace(['Santa Rosa', 'Eureka', 'San Francisco', 'San Jose',
                                                       'Los Angeles', 'Oxnard', 'San Deigo', 'Oceanside',
                                                       'Carlsbad', 'Montrose', 'Prescott', 'Fresno', 'Reno',
                                                       'Las Vegas', 'Tucson', 'San Luis', 'Denver', 'Kingman',
                                                       'Bakersfield', 'Mexicali', 'Silver City', 'Phoenix',
                                                       'Santa Fe', 'Lovelock'],
                                                       'West')
```

Figure 7: Creating new feature "Region"

6 Model Implementation

6.1 Linear Regression

```
df_insurance_dummy = sm.add_constant(df_insurance_dummy)
X = df_insurance_dummy.drop(['claim','sqrt_claim'], axis=1)
y = df_insurance_dummy[['sqrt_claim','claim']]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
print("The shape of X_train is:",X_train.shape)
print("The shape of X_test is:",X_test.shape)
print("The shape of y_train is:",y_train.shape)
print("The shape of y_test is:",y_test.shape)
```

```
The shape of X_train is: (10238, 90)
The shape of X_test is: (3413, 90)
The shape of y_train is: (10238, 2)
The shape of y_test is: (3413, 2)
```

Figure 9: Test train split

```
# build a full model with significant variables using OLS()
linreg_model_with_significant_var = sm.OLS(y_train['claim'], X_train_significant).fit()

# to print the summary output
print(linreg_model_with_significant_var.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  claim   R-squared:                       0.712
Model:                            OLS   Adj. R-squared:                  0.712
Method:                 Least Squares   F-statistic:                     3166.
Date:                Wed, 14 Dec 2022   Prob (F-statistic):               0.00
Time:                        11:03:59   Log-Likelihood:            -1.0432e+05
No. Observations:               10238   AIC:                         2.087e+05
Df Residuals:                   10229   BIC:                         2.087e+05
Df Model:                           8
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const             -1.054e+04    618.974    -17.022      0.000   -1.17e+04   -9322.949
age                258.4894      4.832     53.497      0.000     249.018     267.961
weight             -50.2615      5.009    -10.035      0.000     -60.079     -40.444
bmi                271.4069     11.695     23.206      0.000     248.482     294.332
no_of_dependents   424.9461     52.801      8.048      0.000     321.445     528.447
bloodpressure       35.9958      6.038      5.962      0.000      24.161      47.831
smoker_1          2.325e+04    160.903    144.493      0.000    2.29e+04    2.36e+04
diabetes_1        1708.1415    154.186     11.078      0.000    1405.907    2010.376
regular_ex_1      -915.7722    152.448     -6.007      0.000   -1214.600    -616.944
==============================================================================
Omnibus:                     1830.077   Durbin-Watson:                   2.022
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4737.254
Skew:                           0.980   Prob(JB):                         0.00
Kurtosis:                       5.695   Cond. No.                     1.07e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.07e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Figure 10: Linear Regression Model

7 Evaluation of Implemented Methods

Evaluation metrics used are R2 score, Adjusted R2 score and RMSE value

3. Predict the values using test set

In [480]:
```
# predict the 'sqrt_claim' using predict()
linreg_full_model_withsqrt_predictions = linreg_full_model_withsqrt.predict(X_test)
```

Note that the predicted values are log transformed claim. In order to get claim values, we take the antilog of these predicted values by using the function np.exp()

In [481]:
```
predicted_claim = np.square(linreg_full_model_withsqrt_predictions)
actual_claim = y_test['claim']
```

4. Compute accuracy measures

Now we calculate accuray measures like Root-mean-square-error (RMSE), R-squared and Adjusted R-squared.

In [482]:
```
# calculate rmse using rmse()
linreg_full_model_withsqrt_rmse = rmse(actual_claim, predicted_claim)

# calculate R-squared using rsquared
linreg_full_model_withsqrt_rsquared = linreg_full_model_withsqrt.rsquared

# calculate Adjusted R-Squared using rsquared_adj
linreg_full_model_withsqrt_rsquared_adj = linreg_full_model_withsqrt.rsquared_adj
```

5. Tabulate the results

In [483]:
```
score_card = pd.DataFrame(columns=['Model_Name', 'R-Squared', 'Adj. R-Squared', 'RMSE'])
score_card
```

Out[483]:

Model_Name	R-Squared	Adj. R-Squared	RMSE

In [484]:
```
linreg_full_model_withsqrt_metrics = pd.Series({
                     'Model_Name': "Linreg full model with sqrt of target variable",
                     'RMSE':linreg_full_model_withsqrt_rmse,
                     'R-Squared': linreg_full_model_withsqrt_rsquared,
                     'Adj. R-Squared': linreg_full_model_withsqrt_rsquared_adj
                   })

score_card = score_card.append(linreg_full_model_withsqrt_metrics, ignore_index=True)
score_card
```

Out[484]:

	Model_Name	R-Squared	Adj. R-Squared	RMSE
0	Linreg full model with sqrt of target variable	0.763457	0.761383	5881.709984

Figure 11: Evaluating model on test data

```
plt.scatter(actual_claim,predicted_claim)
plt.xlabel("Actual Claim")
plt.ylabel("Predicted Claim")
plt.title("Actual Claim vs Predicted Claim")
```

```
Text(0.5, 1.0, 'Actual Claim vs Predicted Claim')
```

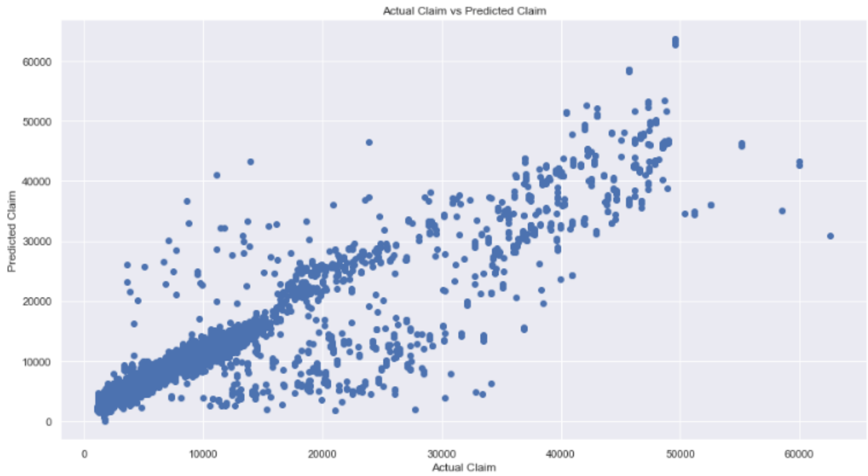

Figure 12: Actual vs Predicted plot of the best model