Medical Insurance Cost Predictor for Coverage

Week 4: Deployment on Flask

Name: Elissa Kuon
Batch Code: LISUM17
Date: January 28, 2023
Submitted to: Data Glacier

1. INTRODUCTION

Most people now have a relationship with a public or commercial health insurance firm, making health insurance a need for daily living. Each company has different criteria for determining the insurance coverage amount. People are easily duped regarding the cost of the insurance and may mistakenly purchase pricey medical coverage. Additionally, this will assist insurers in building an accurate pricing model, planning for a specific insurance outcome, or managing a large portfolio by giving them an idea of more appropriate medical costs for each user. This study gives us a general notion of the cost involved with an individual for his or her own health insurance even though we are unable to estimate the precise amount needed for any health insurance business. The fact that prediction is unreliable and does not adhere to any certain business means that it cannot be the sole factor considered when choosing a health insurance. Based on a small sample of the US population and several distinctive characteristics, our study concentrates on them.

Through this project, we sought to construct a machine learning model that would assist in estimating the annual cost of medical insurance coverage using the Python micro-framework, Flask Framework. This will provide consumers access to the online browser where they may enter their personal information to receive a general estimate of how much a user's medical insurance will cost.

2. DATA INFORMATION

Data Source Link & Overview

The primary source of the dataset was from Kaggle. There are a total of 1,338 rows (individuals) and 7 columns (characteristics of each individual). The data was kept in a CSV file and was structured. Our goal is to predict the medical insurance costs on the Medical Cost Personal Datasets based on some of the user's personal information.

Columns and Data Type - Brief Description

1. 'age': Age of primary beneficiary (discrete numeric variable)
2. 'sex': Insurance contractor gender (categorical variable)
3. 'bmi': Body mass index, providing an understanding of body, weights that are relatively high or low relative to height, objective index of body weight (kg/m^2) using the ratio of height to weight, ideally 18.5 to 24.9 (continuous numeric variable)
4. 'children': Number of children covered by health insurance (discrete numeric variable)
5. 'smoker': Smoking (categorical variable)
6. 'region': The beneficiary's residential area in the US, northeast, southeast, southwest, northwest (categorical variable)

7. 'charges': Individual medical costs billed by health insurance (continuous numeric variable)

The format of this dataset was still raw. As a result, we had to clean the data before using it with the model's algorithm to produce an adequate and precise analysis of the model.

3. BUILDING MACHINE LEARNING MODELS

In order to compare the evaluation of the models, three models—linear regression, polynomial regression (2nd degree), and random forest regression—were taken into consideration.

Importing Libraries and Dataset

We import the relevant dataset and libraries in this section to get the personal health insurance information:

```
## basically the model.py
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import pickle

insurance_df = pd.read_csv("/Users/elissakuon/data_glacier_repos/First-Flask/insurance.csv")
insurance_df.head()
```

After searching for any missing or redundant values, we discovered one duplicate value. However, because there was a potential that another person would have the same personal information given the low number of data (columns or attributes), we decided to leave this value in our dataset.

```
In [15]: # Checking for missing values and duplicated values

In [16]: insurance_df.isna().sum()
Out[16]:
age                 0
bmi                 0
children            0
charges             0
sex_male            0
smoker_yes          0
region_northwest    0
region_southeast    0
region_southwest    0
dtype: int64

In [17]: insurance_df.duplicated().sum()
Out[17]: 1

In [18]: insurance_df[insurance_df.duplicated(keep = False)]
Out[18]:
     age    bmi  children  ...  region_northwest  region_southeast  region_southwest
195   19  30.59         0  ...                 1                 0                 0
581   19  30.59         0  ...                 1                 0                 0

[2 rows x 9 columns]

In [19]: # Checking on the skewness

In [20]: insurance_df.skew(axis = 0, skipna = True)
Out[20]:
age                 0.055673
bmi                 0.284047
children            0.938380
charges             1.515880
sex_male           -0.020951
smoker_yes          1.464766
region_northwest    1.200409
region_southeast    1.025621
region_southwest    1.200409
dtype: float64
```

To make sure our model is error-free, we also made the necessary conversion of the categorical variables into dummy variables.

```
In [21]: # Dummy Variable on Categorical Variables

In [22]: insurance_df = pd.get_dummies(insurance_df, drop_first=True)

In [23]: insurance_df.head()
Out[23]:
   age     bmi  children  ...  region_northwest  region_southeast  region_southwest
0   19  27.900         0  ...                 0                 0                 1
1   18  33.770         1  ...                 0                 1                 0
2   28  33.000         3  ...                 0                 1                 0
3   33  22.705         0  ...                 1                 0                 0
4   32  28.880         0  ...                 1                 0                 0

[5 rows x 9 columns]
```

<u>Splitting the Dataset</u>

The dataset was divided into training and test sets prior to each model, with the target variable "charges" designated as the variable we intend to predict using the other predictors. Looking at the code snapshot under "Building the Model," we can see that one of the models' codes for partitioning the dataset is a tiny bit different.

<u>Building the Model</u>

We proceeded to fit the training dataset produced by dividing the original dataset after importing and initializing each model.

```
# Building the Model
## LINEAR REGRESSION
x = insurance_df.drop(['charges'], axis = 1)
y = insurance_df.charges
# Split between training and testing = 80/20
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size = 0.2,random_state = 0)
lr = LinearRegression().fit(x_train,y_train)

# Get the R-squared value
print(lr.score(x_test,y_test))

## POLYNOMIAL REGRESSION
poly = PolynomialFeatures(degree = 2)
X_poly = poly.fit_transform(x)
# Split between training and testing = 80/20
x_train,x_test,y_train,y_test = train_test_split(X_poly, y, test_size = 0.2,random_state = 0)
plr = LinearRegression().fit(x_train, y_train)

# Get the R-squared value
print(plr.score(x_test, y_test))

## RANDOM FOREST REGRESSOR
# Split between training and testing = 80/20
x_train,x_test,y_train,y_test = train_test_split(x, y, test_size = 0.2,random_state = 0)
rf_model = RandomForestRegressor(n_estimators = 1000, criterion = 'mse', random_state = 0)
rf = rf_model.fit(x_train, y_train)

rf_train_pred = rf_model.predict(x_train)
rf_test_pred = rf_model.predict(x_test)
print('R2 train data: %.3f'%(r2_score(y_train,rf_train_pred)))
print('R2 test data: %.3f'% (r2_score(y_test,rf_test_pred)))
```

Then, after making predictions using the testing set, we evaluate each model, yielding an overall R^2 score for each model:

Table 1: R^2 Score for Each Models

Model	**R^2 Score**
Linear Regression	0.799
Polynomial Regression	0.849
Random Forest Regressor	0.882

We can see that all three models are fairly accurate in estimating how much users' medical insurance will cost. The Random Forest Regressor, however, performed a little bit better than the other models.

Saving the Model

The best model is then saved using Pickle, a Python object structure serializer and deserializer. And then we'll move on to the following action, which is deploying the model onto Flask.

```
# Creating a pickle file for Random Forest Regressor
filename = 'model.pkl'
pickle.dump(rf, open(filename, 'wb'))
```

4. DEPLOYMENT ON FLASK FRAMEWORK

Our objective is to create a web application that consists of a straightforward web page with a form field where visitors can input their personal data based on the dataset's features. The information will render onto a new page after being sent to the online program, giving us the projected cost of the medical care.

For this project, we first made a new folder named First-Flask, which contains the directory tree of the necessary files to launch Flask that we got from the resources page and used as a template:

Table 2: Application Folder File Directory

app-copy.py
insurance_df.py
insurance.csv
model.pkl
templates/
index.html
result.html
static/
style.css
original svg

The primary code for both generating the model and cleaning the data is contained in the Insurance df.py file, which was covered in part 3. (Building Machine Learning Models). The resulting pickle object, Model.pkl, is where we save our best model within the folder. The directory that Flask will search for to render in the web browser is the subdirectory templates. Two HTML files, result.html and index.html, as well as style.css and original.svg, are present in this instance.

App-copy.py

The machine learning code for prediction is included in the app-copy.py file, which also contains the primary code that will be performed by the Python interpreter to operate the Flask web application. To define the URL that should cause the home function to run, we used the route decorator (@app.route("|")). We pre-process the variables, create predictions, and then store the model inside the predict function. We retrieve the newly typed message by the user and apply it to our model to forecast the label of the message. Last but not least, we utilized the run function to only launch the server-side application when the Python interpreter performed this script directly.

```python
from flask import Flask, render_template, request
import numpy as np
import pickle

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/',methods=['GET'])
def Home():
    return render_template('index.html')

@app.route("/predict", methods=['POST'])
def predict():
    if request.method == 'POST':
        age = float(request.form['age'])

        sex = request.form['sex']
        if (sex == 'male'):
            sex_male = 1
            sex_female = 0
        else:
            sex_male = 0
            sex_female = 1

        smoker = request.form['smoker']
        if (smoker == 'yes'):
            smoker_yes = 1
            smoker_no = 0
        else:
            smoker_yes = 0
            smoker_no = 1

        bmi = float(request.form['bmi'])
        children = int(request.form['children'])

        region = request.form['region']
        if (region == 'northwest'):
            region_northwest = 1
            region_southeast = 0
            region_southwest = 0
            region_northeast = 0
        elif (region == 'southeast'):
            region_northwest = 0
            region_southeast = 1
            region_southwest = 0
            region_northeast = 0
        elif (region == 'southwest'):
            region_northwest = 0
            region_southeast = 0
            region_southwest = 1
            region_northeast = 0
        else:
            region_northwest = 0
            region_southeast = 0
            region_southwest = 0
            region_northeast = 1
```

```python
        values = np.array([[age,sex_male,smoker_yes,bmi,children,region_northwest,region_southeast,region_southwest]])
        prediction = model.predict(values)
        prediction = round(prediction[0],2)
        if prediction < 0:
            return render_template('result.html', prediction_text='Predicted Medical Insurance Cost Per Year is $0.00')
        else:
            return render_template('result.html', prediction_text='Predicted Medical Insurance Cost Per Year is ${:,}'.format(prediction))

if __name__ == "__main__":
    app.run(debug=True)
```

Index.html

The contents of the index.html file, which will display a text form for users to enter their information, as well as the general layout of the web application, are shown in the screenshot below.

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Medical Insurance Cost for Coverage Model</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>
<body>
    <div style="color:white;" class="container">
        <h2 class='container-heading'><span class="heading_font">Medical Insurance Cost Predicter for Coverage</span></h2>
    </div>

    <div style="color:white;" class="ml-container">
        <form action="{{ url_for('predict') }}" method="POST">
            <br>
            <h3>Age</h3>
            <input id="first" name="age" placeholder="in year" required="required">
            <br>
            <h3>Sex</h3>
            <select id="second" class="form-input align-center" name="sex">
                <option value="None">Sex</option>
                <option value="sex_male">Male</option>
                <option value="sex_female">Female</option>
            </select>
            <br>
            <h3>Smoker</h3>
            <select id="third" class="form-input align-center" name="smoker">
                <option value="None">Smoker</option>
                <option value="smoker_yes">Yes</option>
                <option value="smoker_no">No</option>
            </select>
            <br>
            <h3>Body Mass Index</h3>
            <input id="fourth" name="bmi" placeholder="BMI" required="required">
            <br>
            <h3>Number of Childrens?</h3>
            <input id="fifth" name="children" placeholder="0, 1, 2, 3, 4, or 5" required="required">
            <br>
            <h3>Region</h3>
            <select id="sixth" class="form-input align-center" name="region">
                <option value="None">Region</option>
                <option value="region_northwest">North-West</option>
                <option value="region_southeast">South-East</option>
                <option value="region_southwest">South-West</option>
                <option value="region_northeast">North-East</option>
            </select>
            <br>
            <br>
            <br>
            <button id="sub" type="submit ">Submit</button>

        </form>
    </div>

 <img src="/static/images/Original.svg" style="width: 400px;position: absolute;bottom: 10px;left: 10px;" alt="Company Logo"/>
```

Result.html

To show the text that a user inputs into the text box, we create a result.html file that will be rendered using the render template('result.html',...) line return inside the predict function that we defined in the app-copy.py script.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Medical Insurance Cost Result</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <div style="color:white;" class="container">
        <form action="{{ url_for('predict')}}" method="post">
            <h2 class='container-heading'><span class="heading_font">Medical Insurance Cost Predicter for Single Coverage</span></h2>
        <br><br><br><br><br><br><br><br><br><br><br><br><br>
        <!-- Result -->
        <div style="color:white;" class="results">
            <h2> {{prediction_text}}</h2>
        </div>
        </form>
    </div>
<style>
/* Color */
body{
    font-family: Impact;
    text-align: center;
    margin: 0;
    padding: 0;
    width: 100%;
    height: 100%;
    display: flex;
    flex-direction: column;
}
/* Heading Font */
.container-heading{
    margin: 0;
}
.heading_font{
    color: #white;
    font-family: 'Impact', bold;
    font-size: 50px;
    font-weight: normal;
}
</style>
 <img src="/static/images/Original.svg" style="width: 400px;position: absolute;bottom: 10px;left: 10px;" alt="Company Logo"/>
</body>
</html>
```

Style.css and Original.svg

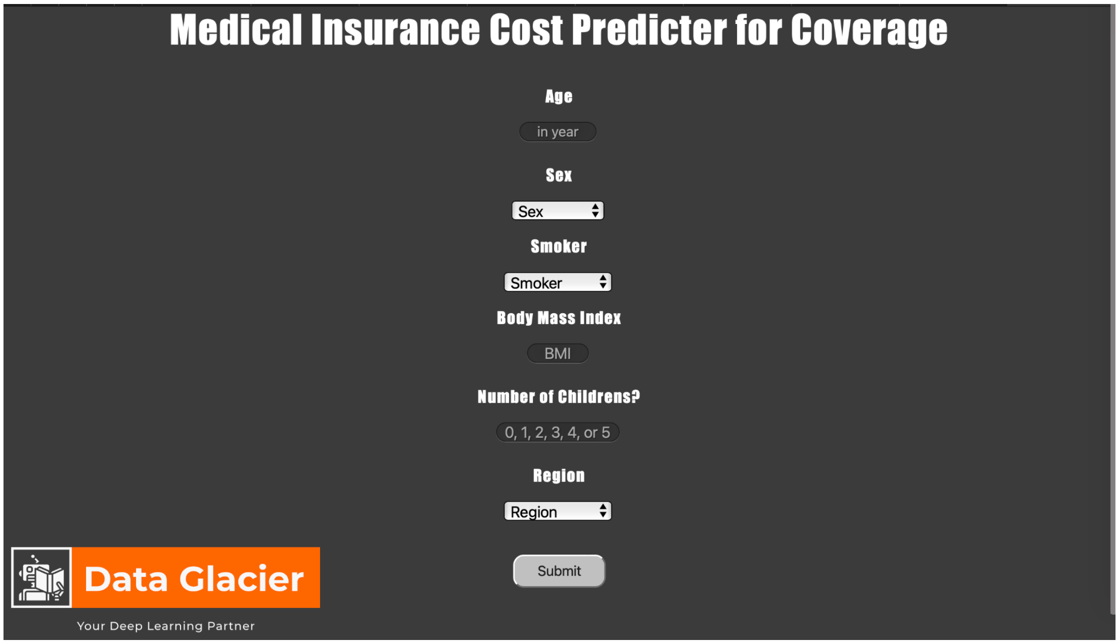
The styles.css and original.svg files were loaded into the index.html file's header section. While original.svg displays the images included in the web, CSS controls how HTML documents look and feel.

5. RUNNING THE PROCEDURE

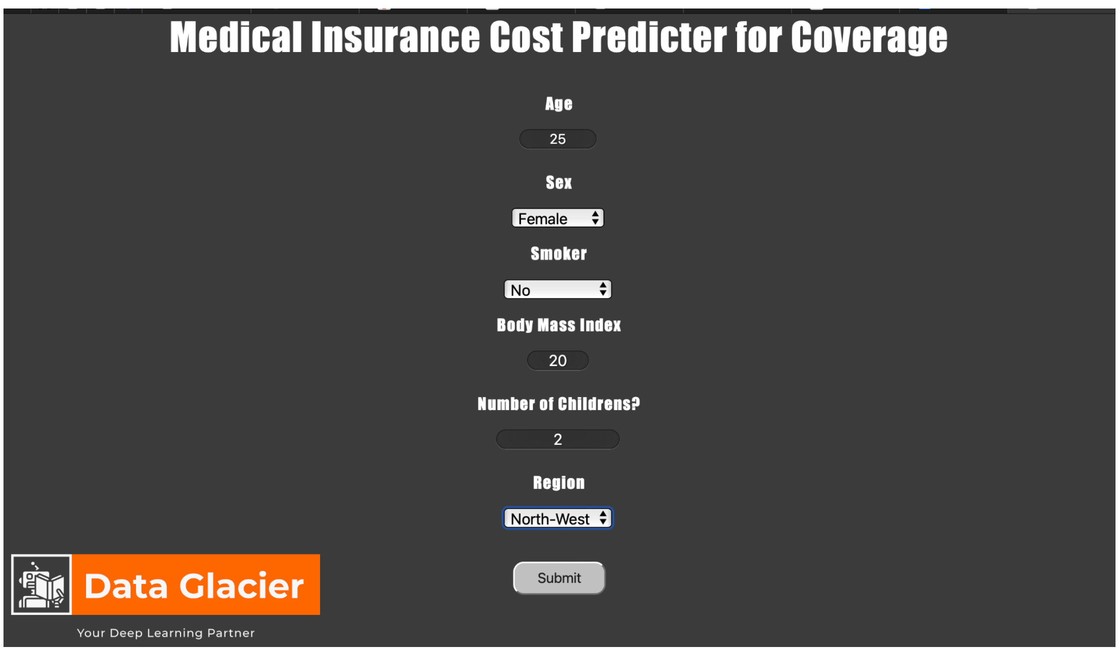
Once we have completed everything above, we can launch the API by typing the following command into the Terminal:

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[(base) Elissas-MacBook-Pro:~ elissakuon$ cd ~/data_glacier_repos/First-Flask/
[(base) Elissas-MacBook-Pro:First-Flask elissakuon$ python app-copy.py
 * Serving Flask app 'app-copy'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
```

Where we launch a web browser and navigate to http://127.0.0.1:5000, where we ought to see a basic site with the following content:



Now we enter appropriate information in the form:



After entering the data, we click the "Submit" button to see the model's interpretation of the data as a result of our input:

6. CONCLUSION

With the help of the Python micro-framework Flask Framework, we aimed to build a model using machine learning for this project that would aid in calculating the annual cost of health insurance coverage. Customers will have access to an internet browser through which they may enter their personal information to get a ballpark idea of how much a user's medical insurance will cost. Despite the fact that our model's results are not the most accurate, they nonetheless provide users a general notion of what to expect. However, further research is required to obtain a more accurate estimate of the cost of medical insurance. We advise considering additional data samples or attributes other than the ones that were utilized in this research.