Name: CALEB ANTHONY

Batch code: LISUM04

Submission date: 26/10/2021

Submitted to: DATA GLACIER VIRTUAL INTERNSHIP

Week4: Deployment on Flask

The necessary elements are:

Static folder — Exists in the root directory. This contains all your static assets like css files, images, fonts, and compressed models.

Template folder — Exists in the root directory. This is the default location that template HTML files MUST be in for Flask to render them properly. Any page that interacts with your models will be in here.

Model folder — Exists in the root directory. This contains the saved deep learning model in h5 format and the tokenizer dictionary in json format.

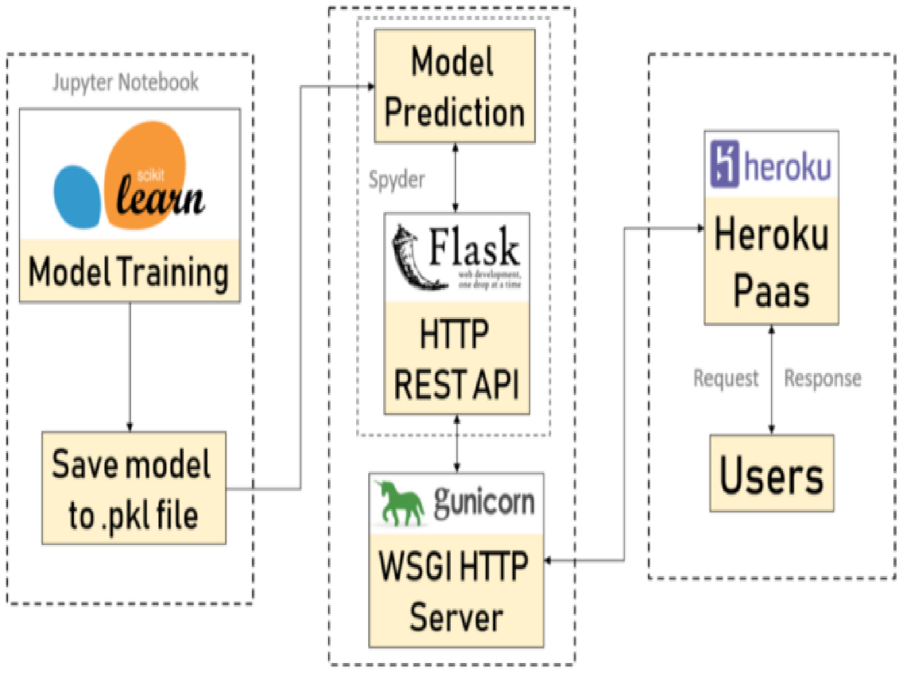
Home.html — This is the front-facing HTML file that users can interact with and that your model will take the input text.

Result.html — For visualizing the output with nice progress bars
App.py — Exists in root directory. This file contains the functions that run your model and data preprocessing.

Requirments.txt — Exists in root directory. This file contains necessary libraries to install it in cloud platform.

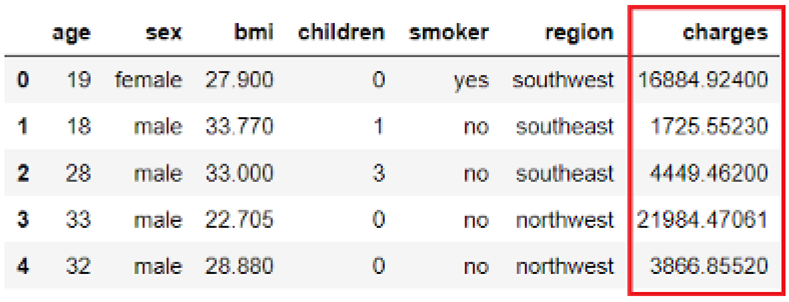
Procfile — Exists in root directory. This file is necessary for Heroku platform. It contains information about the programming language and the main file.

Runtime.txt — Exists in root directory. This file is necessary for Heroku platform. It contains programming language version.



Business Problem

An insurance company wants to improve its cash flow forecasting by better predicting patient charges using demographic and basic patient health risk metrics at the time of hospitalization.



	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

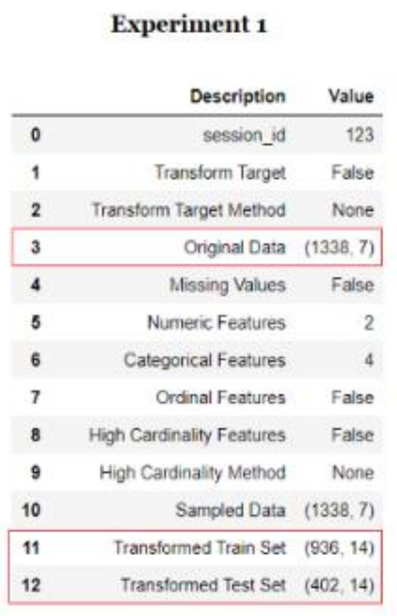
Objective

To build a web application where demographic and health information of a patient is entered in a web form to predict charges.

Task 1 — Model Training and Validation

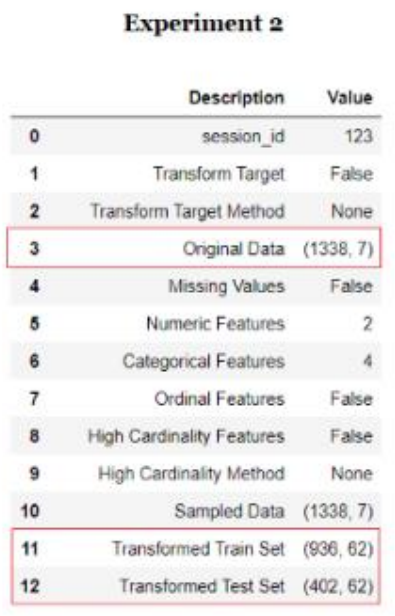
Here we will use PyCaret in Jupyter Notebook to develop machine learning pipeline and train regression models. We have performed two experiments. The first experiment is performed with default preprocessing settings in PyCaret (missing value imputation, categorical encoding etc). The second experiment has some additional preprocessing tasks such as scaling and normalization, automatic feature engineering and binning continuous data into intervals. See the setup example for the second experiment:

```
# Experiment No. 2from pycaret.regression import *r2 =
setup(data, target = 'charges', session_id = 123,
           normalize = True,
           polynomial_features = True,
trigonometry_features = True,
           feature_interaction=True,
           bin_numeric_features= ['age', 'bmi'])
```



Experiment 1

	Description	Value
0	session_id	123
1	Transform Target	False
2	Transform Target Method	None
3	Original Data	(1338, 7)
4	Missing Values	False
5	Numeric Features	2
6	Categorical Features	4
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(1338, 7)
11	Transformed Train Set	(936, 14)
12	Transformed Test Set	(402, 14)

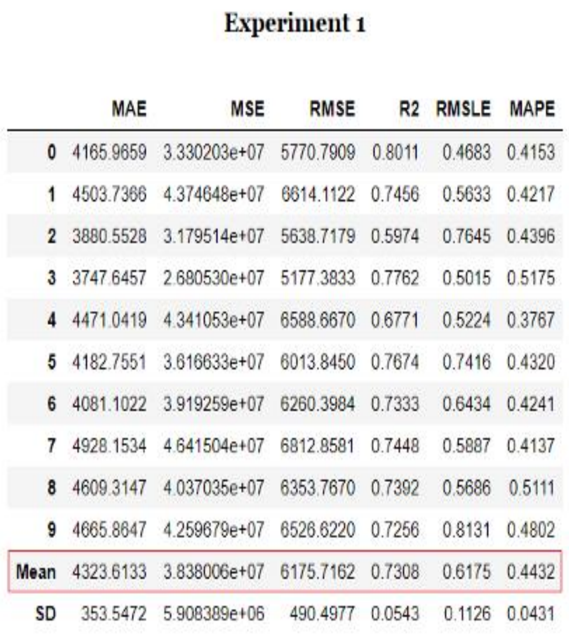


Experiment 2

	Description	Value
0	session_id	123
1	Transform Target	False
2	Transform Target Method	None
3	Original Data	(1338, 7)
4	Missing Values	False
5	Numeric Features	2
6	Categorical Features	4
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(1338, 7)
11	Transformed Train Set	(936, 62)
12	Transformed Test Set	(402, 62)

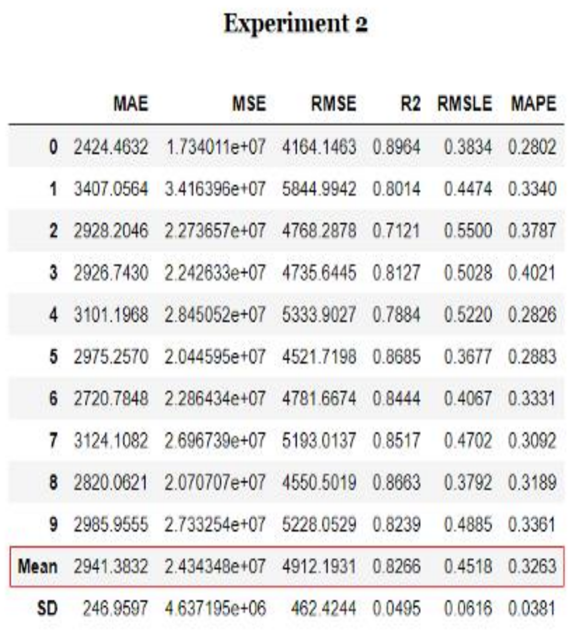
Sample code for model training and validation in PyCaret:

```
# Model Training and Validation
lr = create_model('lr')
```



Experiment 1

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	4165.9659	3.330203e+07	5770.7909	0.8011	0.4683	0.4153
1	4503.7366	4.374648e+07	6614.1122	0.7456	0.5633	0.4217
2	3880.5528	3.179514e+07	5638.7179	0.5974	0.7645	0.4396
3	3747.6457	2.680530e+07	5177.3833	0.7762	0.5015	0.5175
4	4471.0419	4.341053e+07	6588.6670	0.6771	0.5224	0.3767
5	4182.7551	3.616633e+07	6013.8450	0.7674	0.7416	0.4320
6	4081.1022	3.919259e+07	6260.3984	0.7333	0.6434	0.4241
7	4928.1534	4.641504e+07	6812.8581	0.7448	0.5887	0.4137
8	4609.3147	4.037035e+07	6353.7670	0.7392	0.5686	0.5111
9	4665.8647	4.259679e+07	6526.6220	0.7256	0.8131	0.4802
Mean	4323.6133	3.838006e+07	6175.7162	0.7308	0.6175	0.4432
SD	353.5472	5.908389e+06	490.4977	0.0543	0.1126	0.0431



Experiment 2

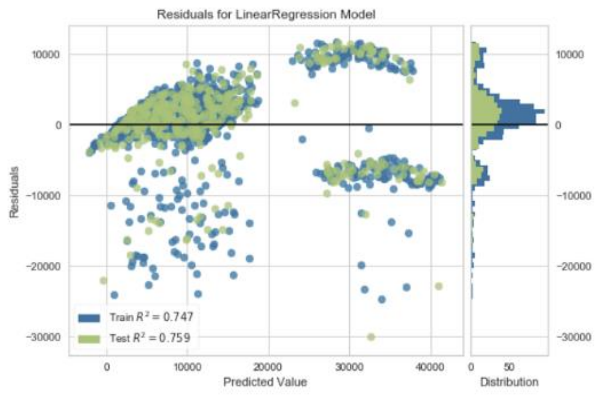
	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	2424.4632	1.734011e+07	4164.1463	0.8964	0.3834	0.2802
1	3407.0564	3.416396e+07	5844.9942	0.8014	0.4474	0.3340
2	2928.2046	2.273657e+07	4768.2878	0.7121	0.5500	0.3787
3	2926.7430	2.242633e+07	4735.6445	0.8127	0.5028	0.4021
4	3101.1968	2.845052e+07	5333.9027	0.7884	0.5220	0.2826
5	2975.2570	2.044595e+07	4521.7198	0.8685	0.3677	0.2883
6	2720.7848	2.286434e+07	4781.6674	0.8444	0.4067	0.3331
7	3124.1082	2.696739e+07	5193.0137	0.8517	0.4702	0.3092
8	2820.0621	2.070707e+07	4550.5019	0.8663	0.3792	0.3189
9	2985.9555	2.733254e+07	5228.0529	0.8239	0.4885	0.3361
Mean	2941.3832	2.434348e+07	4912.1931	0.8266	0.4518	0.3263
SD	246.9597	4.637195e+06	462.4244	0.0495	0.0616	0.0381

10 Fold cross-validation of Linear Regression Model(s)

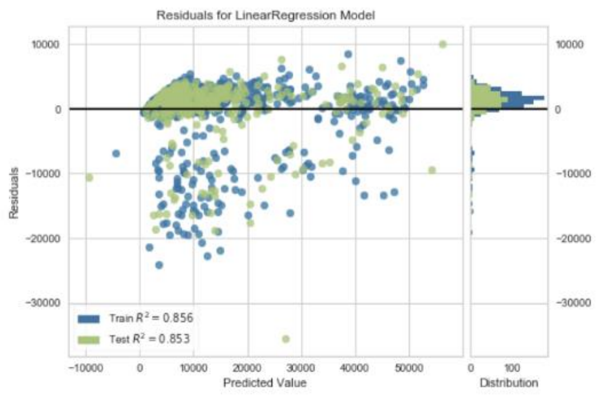
```
# plot residuals of trained model
```

```
plot_model(lr, plot = 'residuals')
```

Experiment 1

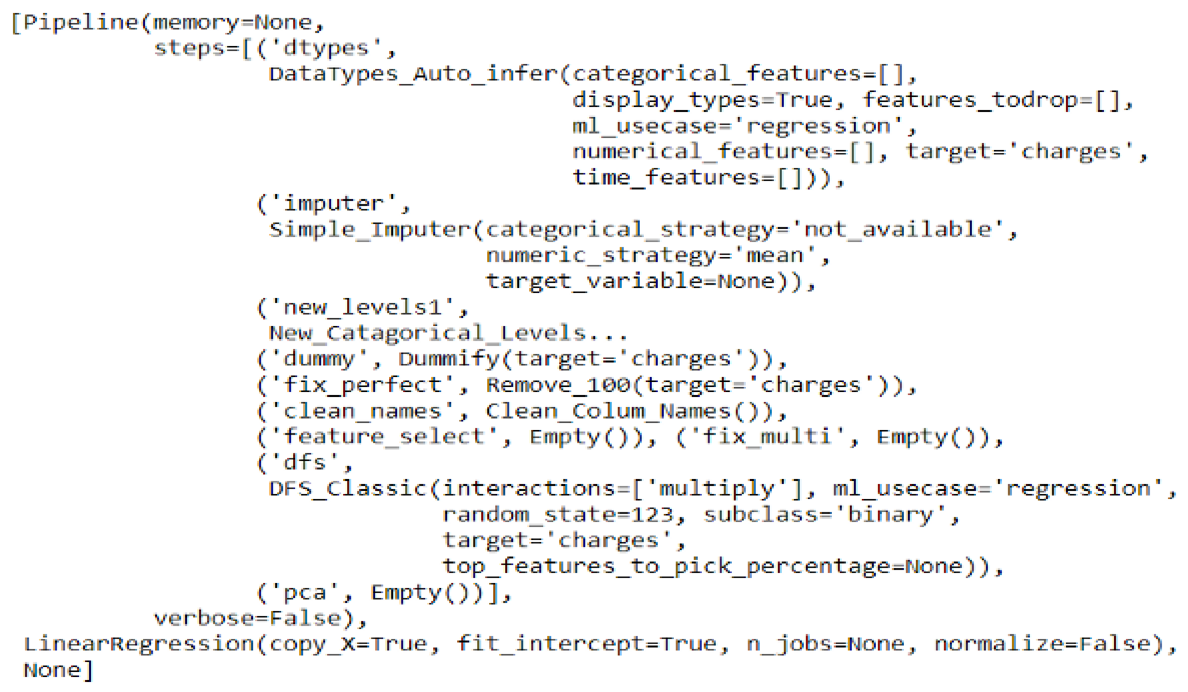


Experiment 2



To save it as a file that can be transferred to and consumed by other applications, run the following code:

```
# save transformation pipeline and model save_model(lr, model_name =
'c:/username/ins/deployment_28042020')
```



```
[Pipeline(memory=None,
          steps=[('dtypes',
                  DataTypes_Auto_infer(categorical_features=[],
                                       display_types=True, features_todrop=[],
                                       ml_usecase='regression',
                                       numerical_features=[], target='charges',
                                       time_features=[])),
                 ('imputer',
                  Simple_Imputer(categorical_strategy='not_available',
                                 numeric_strategy='mean',
                                 target_variable=None)),
                 ('new_levels1',
                  New_Catagorical_Levels...
                 ('dummy', Dummify(target='charges')),
                 ('fix_perfect', Remove_100(target='charges')),
                 ('clean_names', Clean_Colum_Names()),
                 ('feature_select', Empty()), ('fix_multi', Empty()),
                 ('dfs',
                  DFS_Classic(interactions=['multiply'], ml_usecase='regression',
                              random_state=123, subclass='binary',
                              target='charges',
                              top_features_to_pick_percentage=None)),
                 ('pca', Empty())],
          verbose=False),
 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False),
 None]
```

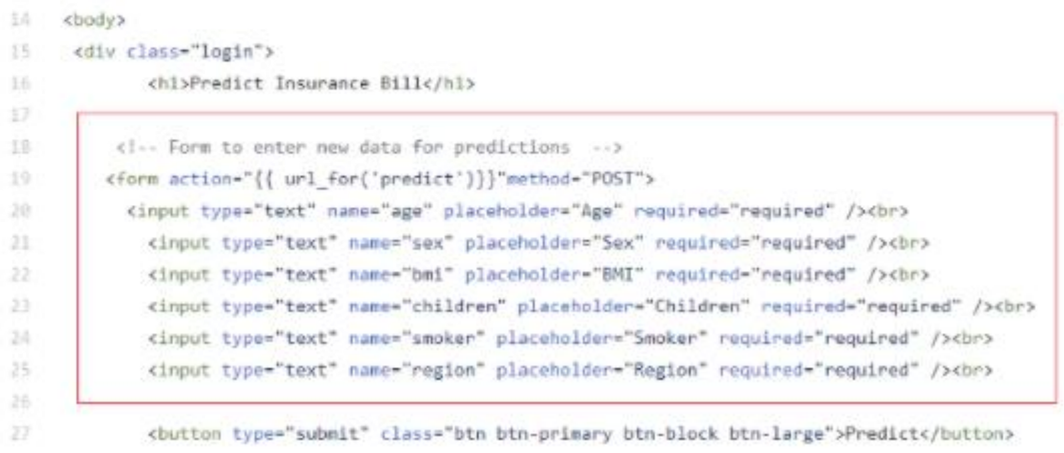
We have finished our first task of training and selecting a model for deployment. The final machine learning pipeline and linear regression model is now saved as a file in the local drive under the location defined in the **save_model()** function. (In this example: c:/*username*/ins/deployment_28042020.pkl).

Task 2 — Building Web Application

Now that our machine learning pipeline and model are ready we will start building a web application that can connect to them and generate predictions on new data in real-time. There are two parts of this application:

- Front-end (designed using HTML)
- Back-end (developed using Flask in Python)

Front-end of Web Application

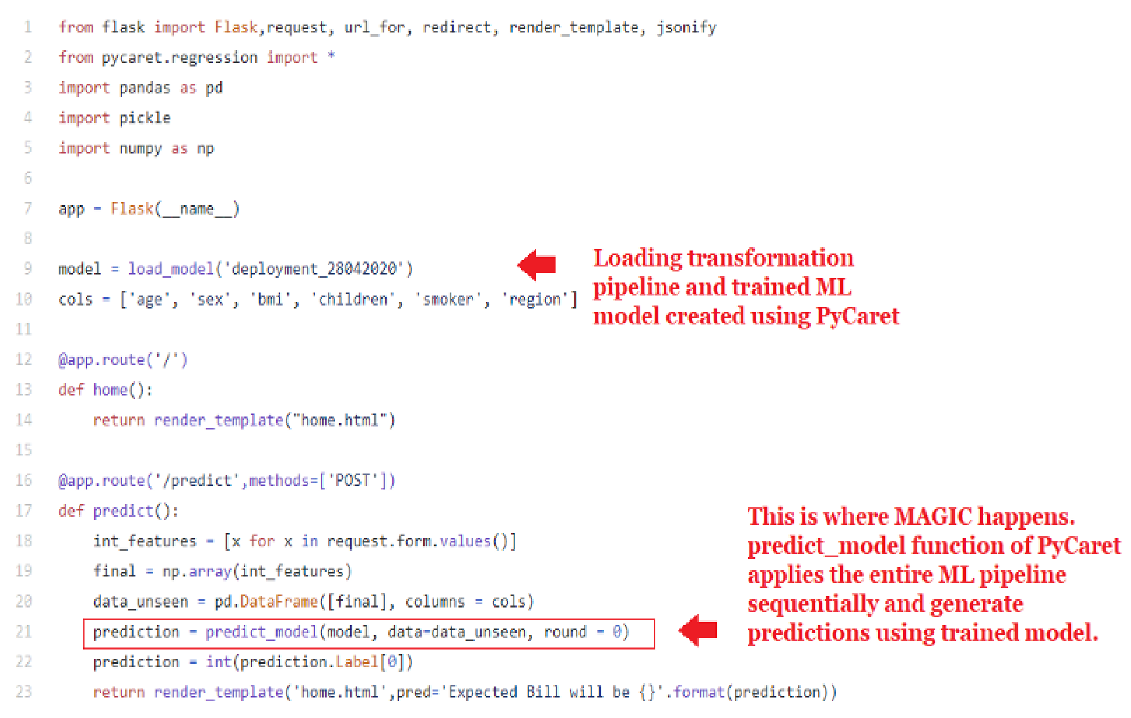


```
<body>
 <div class="login">
        <h1>Predict Insurance Bill</h1>

     <!-- Form to enter new data for predictions  -->
    <form action="{{ url_for('predict')}}"method="POST">
      <input type="text" name="age" placeholder="Age" required="required" /><br>
        <input type="text" name="sex" placeholder="Sex" required="required" /><br>
        <input type="text" name="bmi" placeholder="BMI" required="required" /><br>
        <input type="text" name="children" placeholder="Children" required="required" /><br>
        <input type="text" name="smoker" placeholder="Smoker" required="required" /><br>
        <input type="text" name="region" placeholder="Region" required="required" /><br>

        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
```

Back-end of Web Application



```
from flask import Flask,request, url_for, redirect, render_template, jsonify
from pycaret.regression import *
import pandas as pd
import pickle
import numpy as np

app = Flask(__name__)

model = load_model('deployment_28042020')
cols = ['age', 'sex', 'bmi', 'children', 'smoker', 'region']

@app.route('/')
def home():
    return render_template("home.html")

@app.route('/predict',methods=['POST'])
def predict():
    int_features = [x for x in request.form.values()]
    final = np.array(int_features)
    data_unseen = pd.DataFrame([final], columns = cols)
    prediction = predict_model(model, data=data_unseen, round = 0)
    prediction = int(prediction.Label[0])
    return render_template('home.html',pred='Expected Bill will be {}'.format(prediction))
```

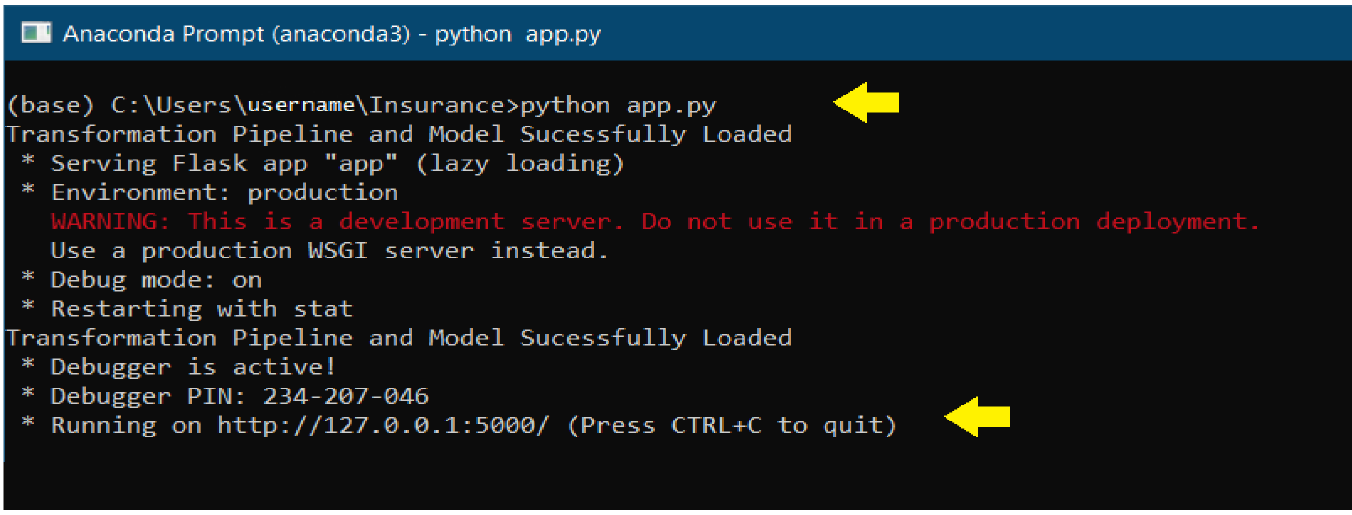
Loading transformation pipeline and trained ML model created using PyCaret

This is where MAGIC happens. predict_model function of PyCaret applies the entire ML pipeline sequentially and generate predictions using trained model.

Testing App.

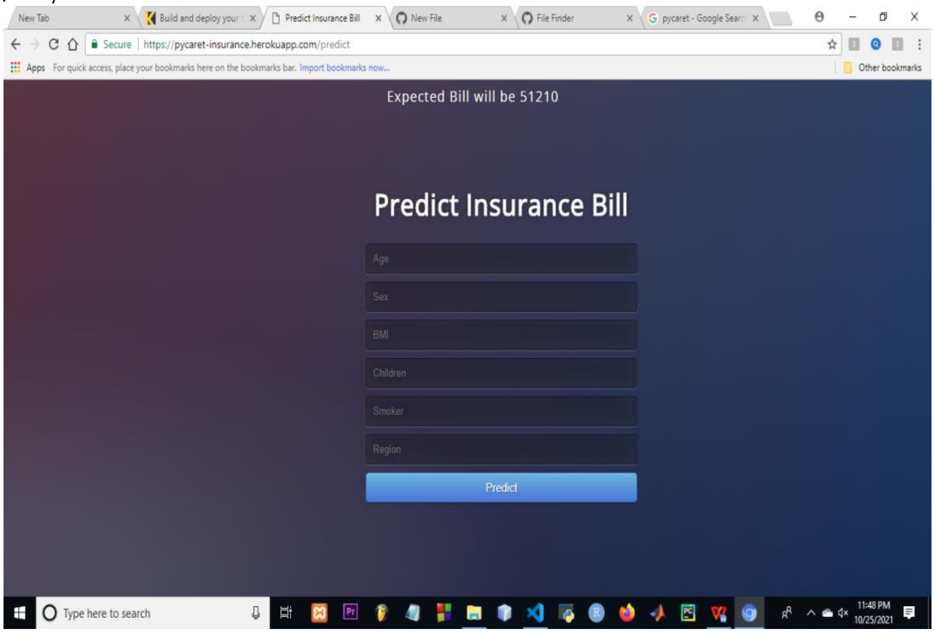
Open Anaconda Prompt and navigate to folder where **'app.py'** is saved on your computer. Run the python file with below code:

```
python app.py
```



```
(base) C:\Users\username\Insurance>python app.py
Transformation Pipeline and Model Sucessfully Loaded
 * Serving Flask app "app" (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
Transformation Pipeline and Model Sucessfully Loaded
 * Debugger is active!
 * Debugger PIN: 234-207-046
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

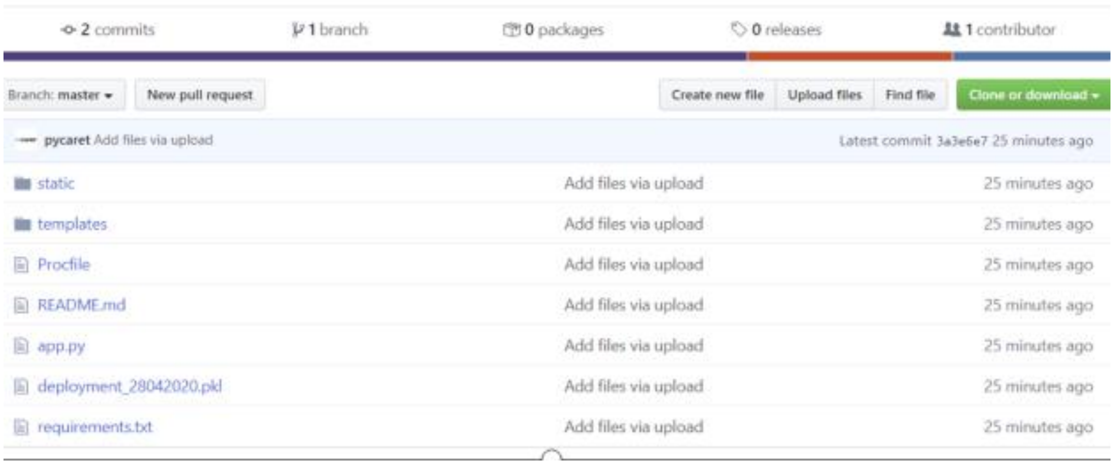
Once executed, copy the URL into a browser and it should open a web application hosted on your local machine (127.0.0.1). Try entering test values to see if the predict function is working. In the example below, the expected bill for a 19 year old female smoker with no children in the southwest is $51,210.


https://pycaret-insurance.herokuapp.com/predict
Expected Bill will be 51210
Predict Insurance Bill
Age
Sex
BMI
Children
Smoker
Region
Predict

Web application opened on local machine

Congratulations! you have now built your first machine learning app. Now it’s time to take this application from your local machine into the cloud so other people can use it with a Web URL.

Task 3 — Deploy the Web App on Flask



pycaret Add files via upload — Latest commit 3a3e6e7 25 minutes ago

static	Add files via upload	25 minutes ago
templates	Add files via upload	25 minutes ago
Procfile	Add files via upload	25 minutes ago
README.md	Add files via upload	25 minutes ago
app.py	Add files via upload	25 minutes ago
deployment_28042020.pkl	Add files via upload	25 minutes ago
requirements.txt	Add files via upload	25 minutes ago