# Web App Deployment Using Flask

**Name: Omar Safwat**

**Batch Code: LISP01**

**Date: 2021-03-22**

## Introduction

This report presents my approach in deploying my web app using Flask, the micro Web app frame work on python. As a demonstration, the app asks the client to select a polynomial degree from a top down list, and subsequently plot a polynomial regression on the airquality data set.

## Importing libraries

In [ ]:

```
from flask import Flask, request, render_template #Web app framework
import numpy as np
import pandas as pd
#Scikit learn for regression model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt #To create plots
from io import BytesIO #To encode plots and pass them to render_template
import base64
```

## Home page

The home page prompts the client to choose an order for the degree polynomial to build the model. A HTTP GET request is issued (refer to layout.html), and the url is then parsed using the imported function *request()* from Flask, as seen below.

In [ ]:

```
#Create app using flask
app = Flask(__name__)

@app.route('/')
def index():
    #Parse html for "GET" request with client's input
    poly_order = request.args.get("poly_order", 1, type=int)
    #Load data
    airquality = pd.read_csv('airquality.csv')
    airquality.dropna(inplace=True)
    data = airquality[['Temp', 'Ozone']]
    #Build  model
```

```
    plot_url, r2 = build_model(poly_order, data)
    return render_template('layout.html', tables=[data.head(6).to_html(classes='data', header=True)], r2=r2, plot_url=plot_url.decode('utf8'), chosen_order=poly_or
```

## Model building

The client's selection is then passed to the function *build_model()* where the polynomial regression model is built as seen below.

In [ ]:

```
def build_model(poly_order, data):

    x = data['Ozone']
    x = x[:, np.newaxis]
    y = data['Temp']
    y = y[:, np.newaxis]
    #Create polynomial terms out of 1D array
    poly_features = PolynomialFeatures(degree = poly_order)
    x_poly = poly_features.fit_transform(x)
    #Build the model
    model = LinearRegression()
    model.fit(x_poly, y)
    y_poly_pred = model.predict(x_poly)

    #Training Error
    r2 = r2_score(y, y_poly_pred)

    #Sort axis and create plot
    plt.scatter(data['Ozone'], data['Temp'], s=10)
    sorted_zip = sorted(zip(data['Ozone'], y_poly_pred))
    x, y_poly_pred = zip(*sorted_zip)
    plt.plot(x, y_poly_pred, color='m')
    plt.xlabel('Ozone (ppb)')
    plt.ylabel('Temp (Fahrenheit)')
    #Save plot to return to html
    img = BytesIO()
    plt.savefig(img, format='png')
    plt.close()
    img.seek(0)
    #Encode figure and pass to html in index function
    plot_url = base64.b64encode(img.getvalue())
    return(plot_url, r2)

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

# Results

The image below shows the web app, built with the above code.

In [6]:

```
from IPython.display import Image
Image(filename= "webapp.jpg", width=900)
```

Out[6]: