Week 4

Flask report and file details

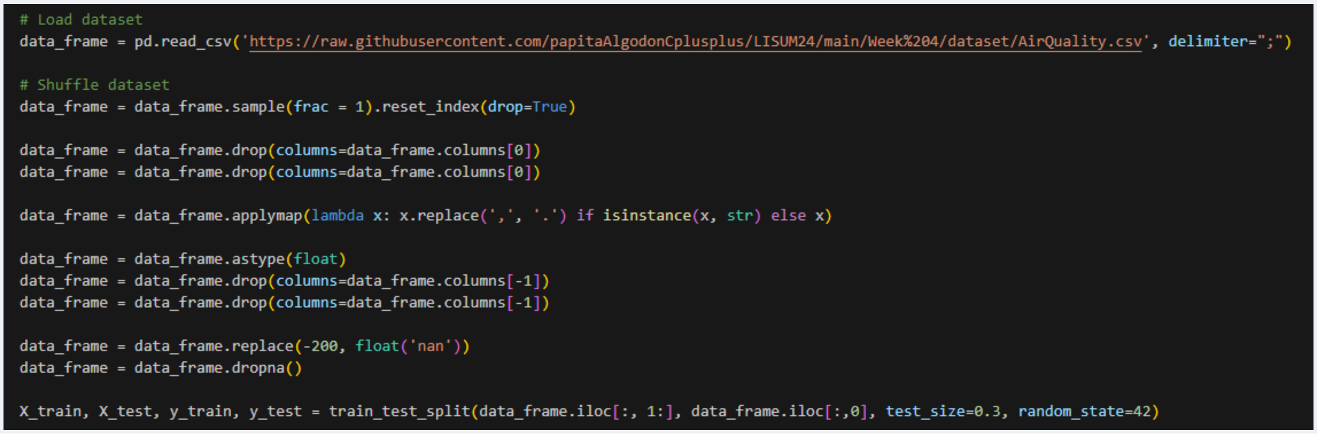
4-8-2023
Virtual Internship

Alexander Quesada Q
LISUM24

Step 1 – CSV Data extraction

***AirQuality.csv** dataset used for this project was provided by FEDESORIANO (Saverio De Vito (saverio.devito '@' enea.it), ENEA - National Agency for New Technologies, Energy and Sustainable Economic Development), details of data collection and meaning of each value are described by FEDESORIANO online at: Kaggle*

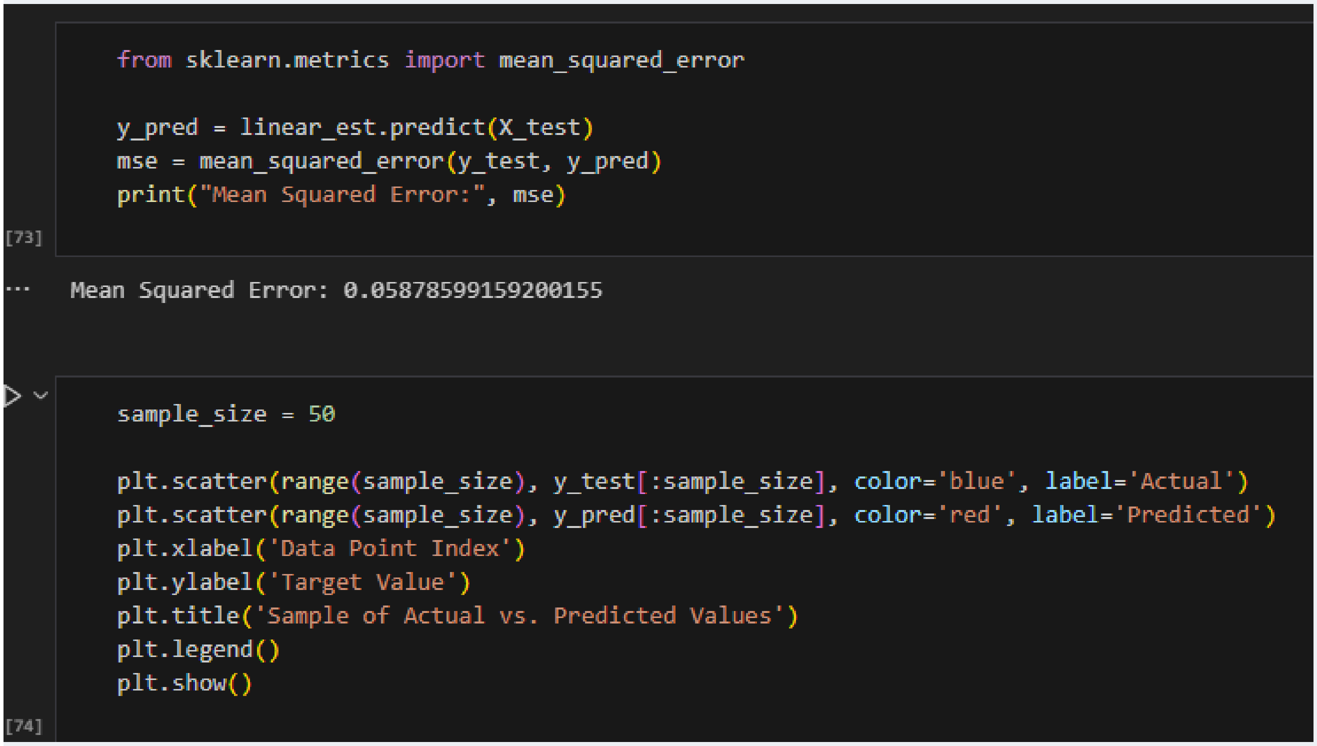
1. Load the dataset from raw storage on Github.
2. Drop unused columns (date and hour)
3. Replace invalid characters for numeric values
4. Convert to float all data
5. Drop last columns (Noise created after conversion)
6. Replace -200 with nan and drop nan values (See Kaggle description)
7. Split the dataset into 70% training, 30% validation



```
# Load dataset
data_frame = pd.read_csv('https://raw.githubusercontent.com/papitaAlgodonCplusplus/LISUM24/main/Week%204/dataset/AirQuality.csv', delimiter=";")

# Shuffle dataset
data_frame = data_frame.sample(frac = 1).reset_index(drop=True)

data_frame = data_frame.drop(columns=data_frame.columns[0])
data_frame = data_frame.drop(columns=data_frame.columns[0])

data_frame = data_frame.applymap(lambda x: x.replace(',', '.') if isinstance(x, str) else x)

data_frame = data_frame.astype(float)
data_frame = data_frame.drop(columns=data_frame.columns[-1])
data_frame = data_frame.drop(columns=data_frame.columns[-1])

data_frame = data_frame.replace(-200, float('nan'))
data_frame = data_frame.dropna()

X_train, X_test, y_train, y_test = train_test_split(data_frame.iloc[:, 1:], data_frame.iloc[:,0], test_size=0.3, random_state=42)
```

Step 2 – Model Training and Plotting

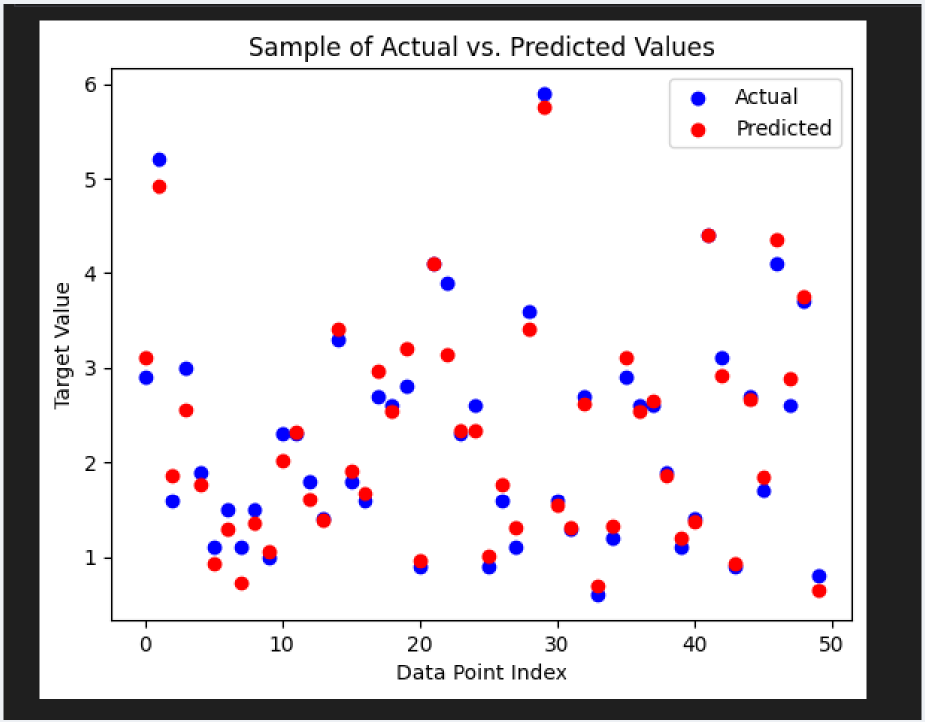
By using scikit-learn library, I could easily implement this model so it allows plk saving (for flask deployment)



```
from sklearn.metrics import mean_squared_error

y_pred = linear_est.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.05878599159200155
```

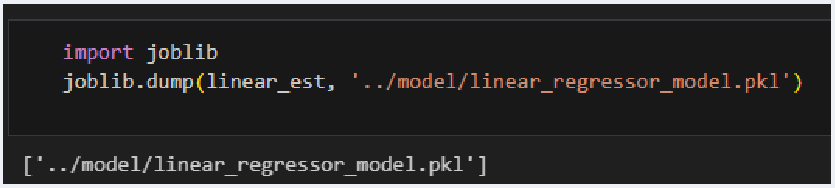
```
sample_size = 50

plt.scatter(range(sample_size), y_test[:sample_size], color='blue', label='Actual')
plt.scatter(range(sample_size), y_pred[:sample_size], color='red', label='Predicted')
plt.xlabel('Data Point Index')
plt.ylabel('Target Value')
plt.title('Sample of Actual vs. Predicted Values')
plt.legend()
plt.show()
```



Step 3 – Flask

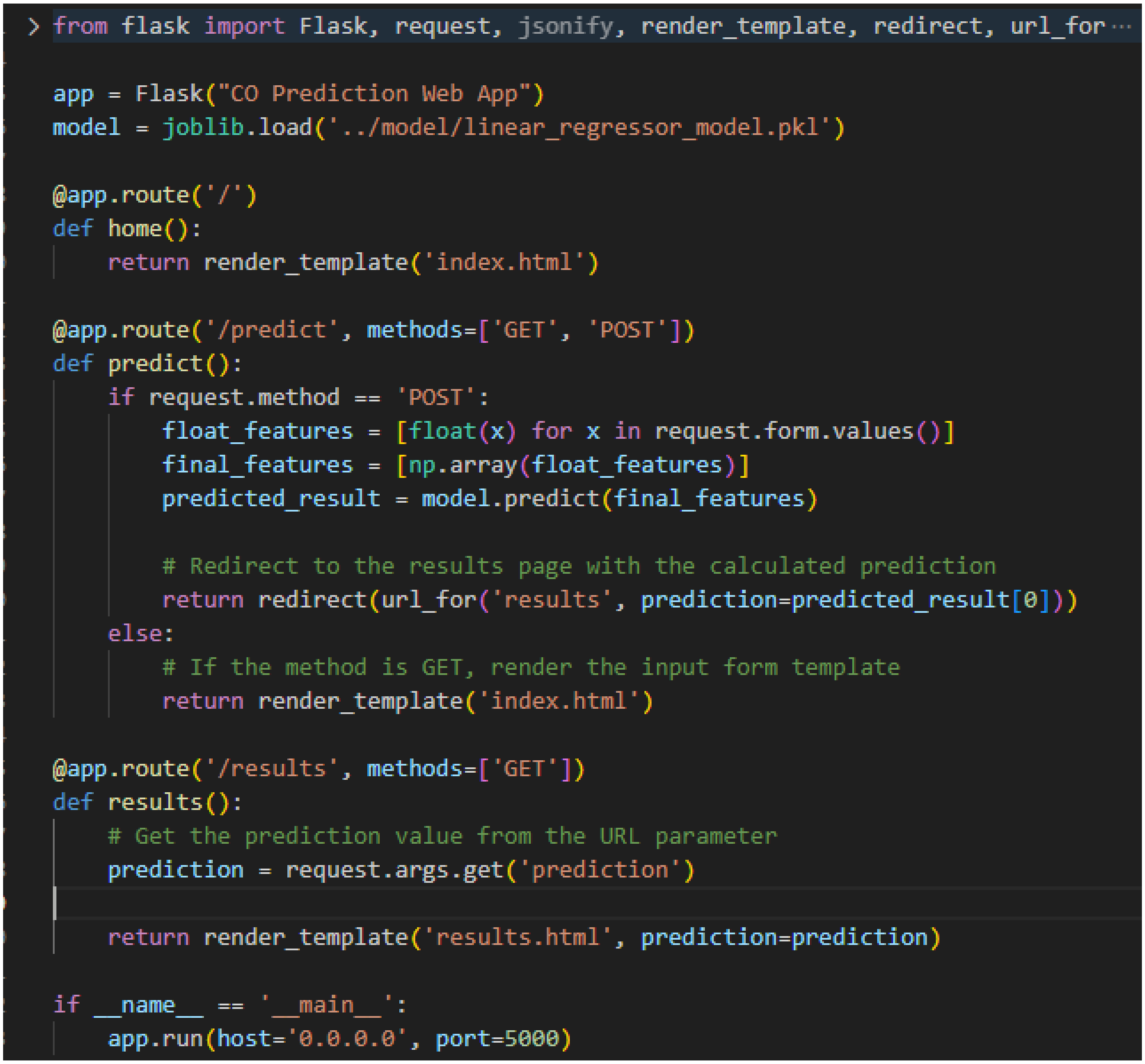
Once the model is ready, flask deployment requires:

1. Saving the model as plk
2. From web_app.py, load the plk model
3. Create home and results html files with css style (optional)
4. Declare and redirect user to each site given '/x' used
5. Declare functions and variables for each site as needed
6. Run the web_app.py from flask virtual enviorment



```
import joblib
joblib.dump(linear_est, '../model/linear_regressor_model.pkl')
```

```
['../model/linear_regressor_model.pkl']
```



```
from flask import Flask, request, jsonify, render_template, redirect, url_for ···

app = Flask("CO Prediction Web App")
model = joblib.load('../model/linear_regressor_model.pkl')

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

@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        float_features = [float(x) for x in request.form.values()]
        final_features = [np.array(float_features)]
        predicted_result = model.predict(final_features)

        # Redirect to the results page with the calculated prediction
        return redirect(url_for('results', prediction=predicted_result[0]))
    else:
        # If the method is GET, render the input form template
        return render_template('index.html')

@app.route('/results', methods=['GET'])
def results():
    # Get the prediction value from the URL parameter
    prediction = request.args.get('prediction')

    return render_template('results.html', prediction=prediction)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 4 – HTML and CSS files

HTML indexing requires 2 files: Homepage and Results

HTML Homepage:

```
<div class="container-fluid">
<div class="row">
<div class="col-md-12">
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
<h1 class="text-center">
CO Prediction Maker
</h1>
<h2>
<p class="text-center text-muted lead">
A Flask Deployed CO Predicter Model
</p>                    </h2>

<form id="prediction-form" action="{{ url_for('predict') }}" method="post">
<label for="co">PT08.S1(CO):</label>
<input type="number" name="co" step="0.01" required><br>

<label for="nmhc">NMHC(GT):</label>
<input type="number" name="nmhc" step="0.01" required><br>


...
```

HTML Results:

```
<!DOCTYPE html>
<html>
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
<head>
<title>Result Page</title>
</head>
<body>
<h1>Result:</h1>
<h3><p>The calculated result is: {{ prediction }}</p></h3>
</body>
</html>
```

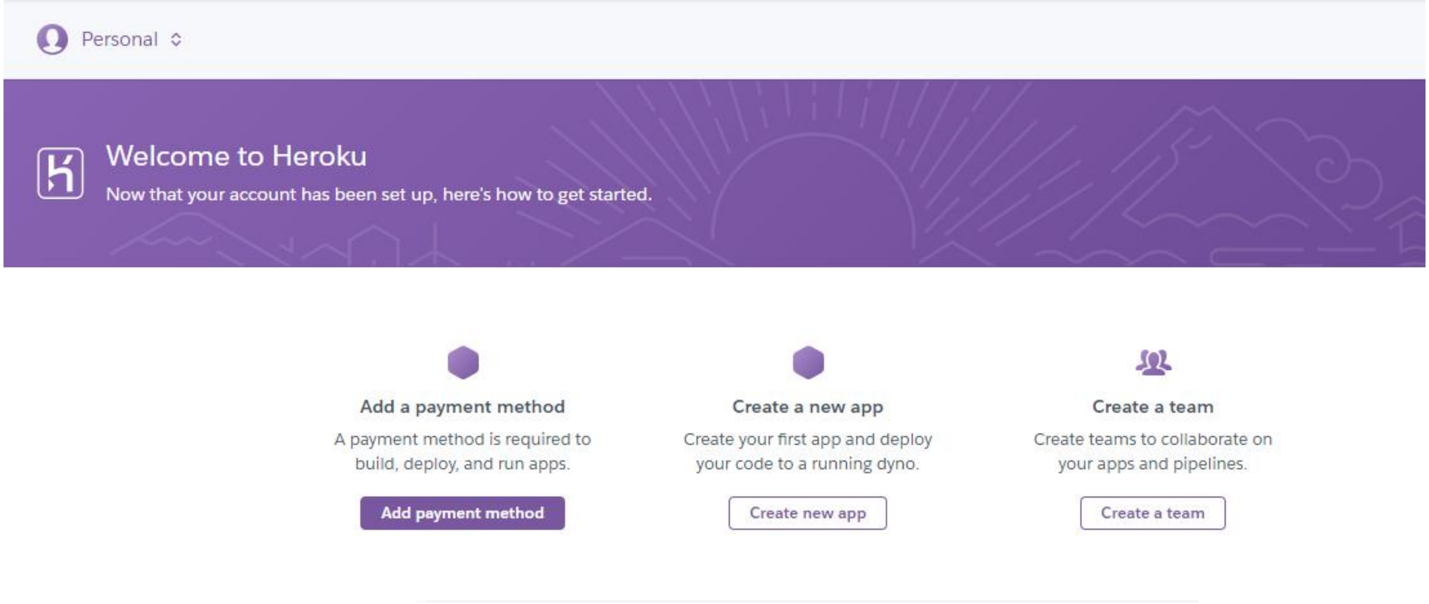
CSS Style:

```
body {
font-family: Arial, sans-serif;
background-image: url("/static/background.jpg");
background-repeat: no-repeat;
background-size: cover;
} ...
```

Step 5 – Heroku Deployment

Using Heroku Web User Interface, it was easy to upload the model using free credits for student accounts, as follows:

1. *Creating account and connect to Github*
2. *Creating requirements.txt with pip freeze*
3. *Creating Procfile with command prompt ‘web: python web_app.py’*
4. *Creating a new app from the web user interface*



Welcome to Heroku

Now that your account has been set up, here's how to get started.

Add a payment method

A payment method is required to build, deploy, and run apps.

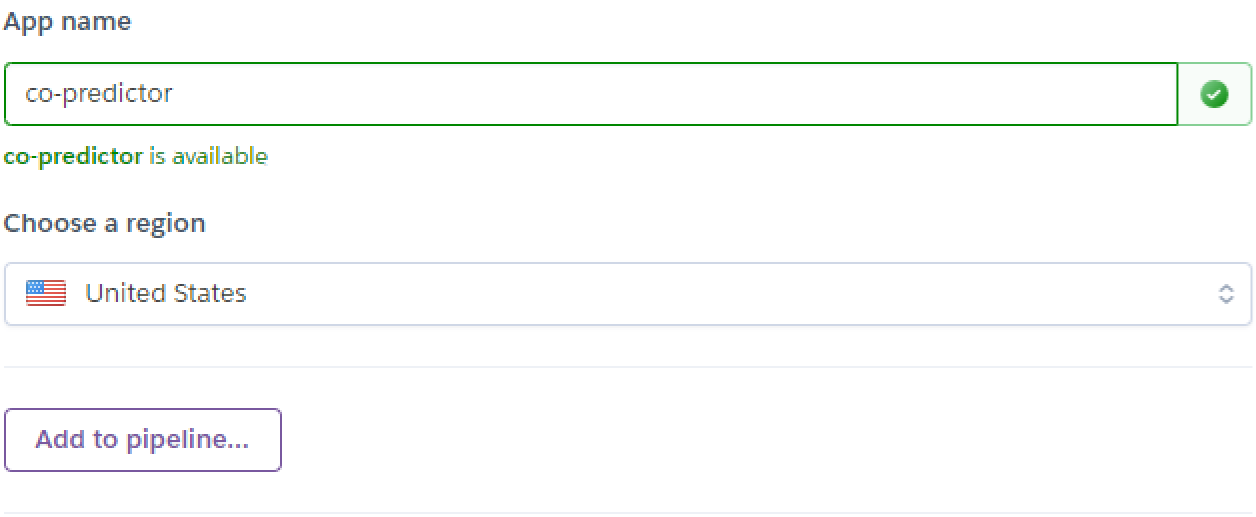
Create a new app

Create your first app and deploy your code to a running dyno.

Create a team

Create teams to collaborate on your apps and pipelines.

5. *Assing name to the new app*



App name: co-predictor

co-predictor is available

Choose a region: United States

6. *Push changes to master 'git push heroku master'*
7. *Scaling the app 'heroku ps:scale web=1'*
8. *Checking log to see if it's running properly 'heroku logs –tail'*

Thanks