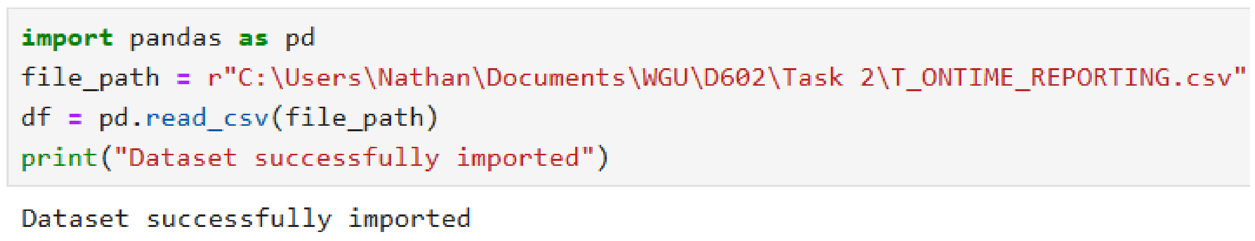
Deployment

Task 2

Nathan Hefner

B. Write a script in either Python or R to import the data you downloaded and format it according to the criteria required by the model script.

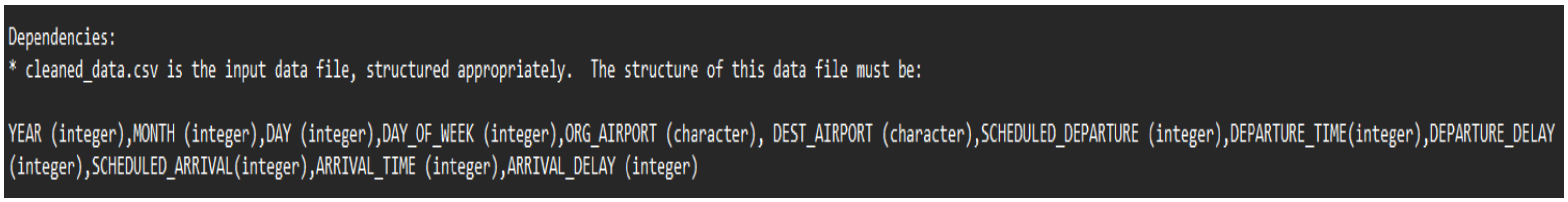
I first downloaded the data from the Bureau of Transportation, only selecting the required columns. I opted for Florida in April 2019. I then imported my dataset.



```
import pandas as pd
file_path = r"C:\Users\Nathan\Documents\WGU\D602\Task 2\T_ONTIME_REPORTING.csv"
df = pd.read_csv(file_path)
print("Dataset successfully imported")
```

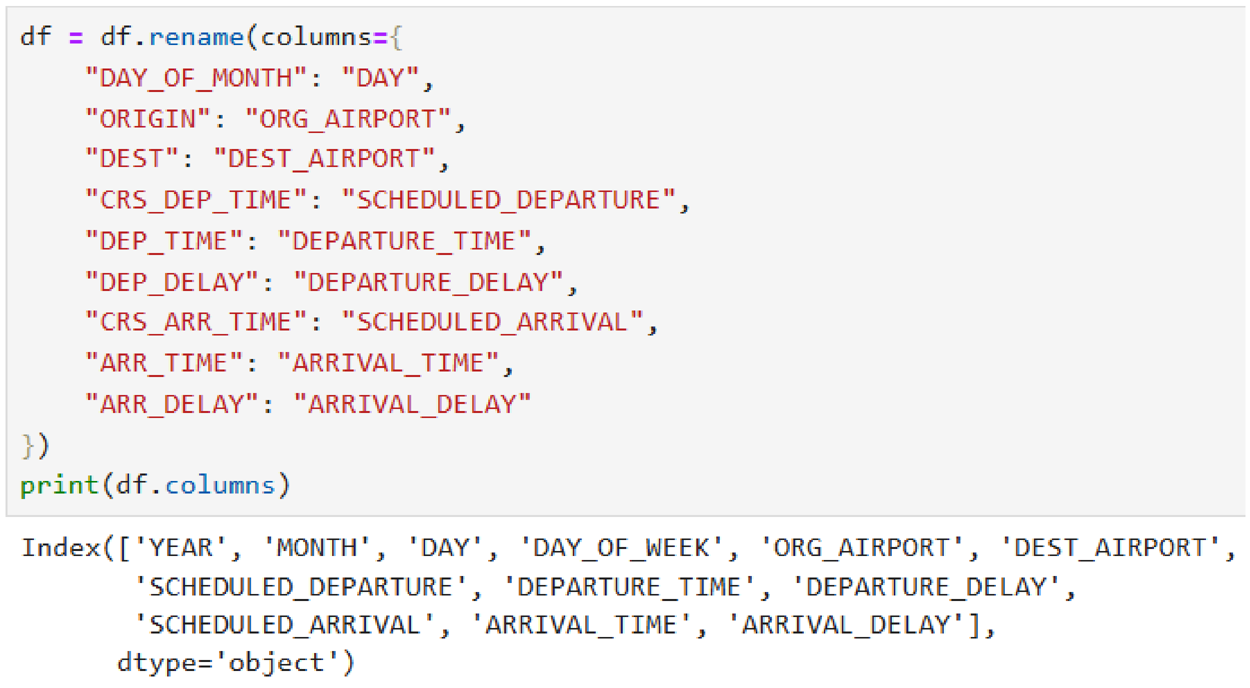
```
Dataset successfully imported
```

I then needed to format the dataset so that it would match the criteria from the model script. Below is what the model requires the columns to be named.



```
Dependencies:
* cleaned_data.csv is the input data file, structured appropriately.  The structure of this data file must be:

YEAR (integer),MONTH (integer),DAY (integer),DAY_OF_WEEK (integer),ORG_AIRPORT (character), DEST_AIRPORT (character),SCHEDULED_DEPARTURE (integer),DEPARTURE_TIME(integer),DEPARTURE_DELAY
(integer),SCHEDULED_ARRIVAL(integer),ARRIVAL_TIME (integer),ARRIVAL_DELAY (integer)
```

Below is the code I used to rename the columns, as there were some discrepancies.

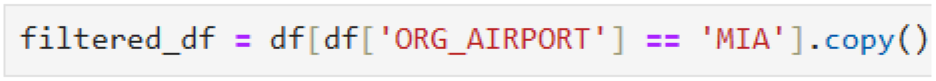


```
df = df.rename(columns={
    "DAY_OF_MONTH": "DAY",
    "ORIGIN": "ORG_AIRPORT",
    "DEST": "DEST_AIRPORT",
    "CRS_DEP_TIME": "SCHEDULED_DEPARTURE",
    "DEP_TIME": "DEPARTURE_TIME",
    "DEP_DELAY": "DEPARTURE_DELAY",
    "CRS_ARR_TIME": "SCHEDULED_ARRIVAL",
    "ARR_TIME": "ARRIVAL_TIME",
    "ARR_DELAY": "ARRIVAL_DELAY"
})
print(df.columns)
```

```
Index(['YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'ORG_AIRPORT', 'DEST_AIRPORT',
       'SCHEDULED_DEPARTURE', 'DEPARTURE_TIME', 'DEPARTURE_DELAY',
       'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME', 'ARRIVAL_DELAY'],
      dtype='object')
```

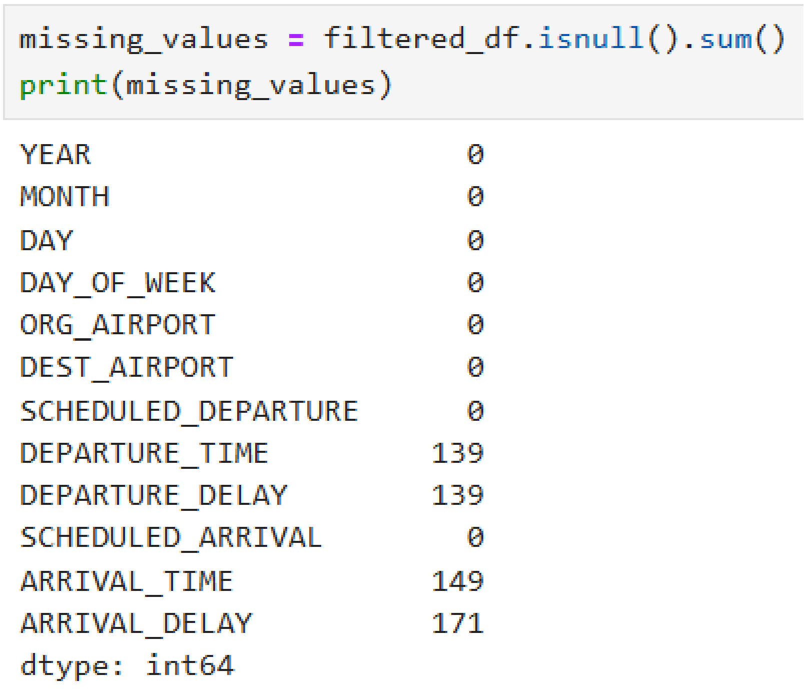
C. Write a script in either Python or R to filter data to only departures from the chosen airport, then implement at least two other data cleaning steps.

My airport in Florida I chose, was Miami.



```python
filtered_df = df[df['ORG_AIRPORT'] == 'MIA'].copy()
```

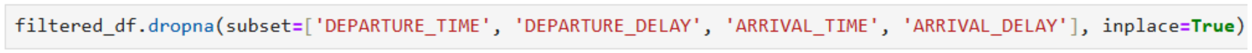
I then checked for null values in each column.



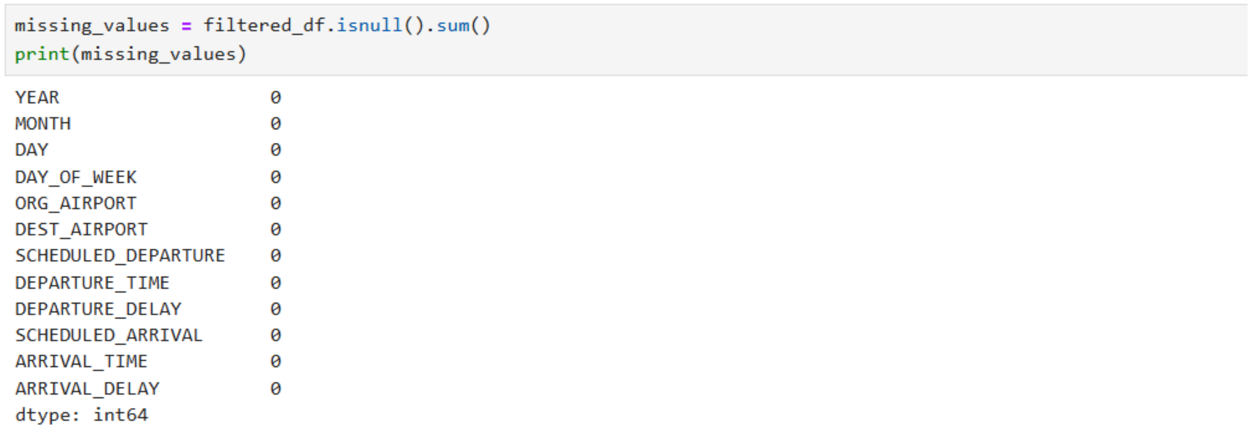
```python
missing_values = filtered_df.isnull().sum()
print(missing_values)
```

```
YEAR                   0
MONTH                  0
DAY                    0
DAY_OF_WEEK            0
ORG_AIRPORT            0
DEST_AIRPORT           0
SCHEDULED_DEPARTURE    0
DEPARTURE_TIME       139
DEPARTURE_DELAY      139
SCHEDULED_ARRIVAL      0
ARRIVAL_TIME         149
ARRIVAL_DELAY        171
dtype: int64
```

Seeing that we had some, I dropped all null values in these columns.



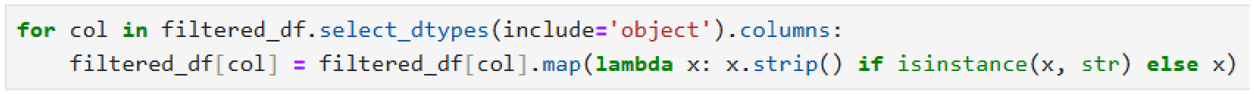
```python
filtered_df.dropna(subset=['DEPARTURE_TIME', 'DEPARTURE_DELAY', 'ARRIVAL_TIME', 'ARRIVAL_DELAY'], inplace=True)
```



```python
missing_values = filtered_df.isnull().sum()
print(missing_values)
```

```
YEAR                   0
MONTH                  0
DAY                    0
DAY_OF_WEEK            0
ORG_AIRPORT            0
DEST_AIRPORT           0
SCHEDULED_DEPARTURE    0
DEPARTURE_TIME         0
DEPARTURE_DELAY        0
SCHEDULED_ARRIVAL      0
ARRIVAL_TIME           0
ARRIVAL_DELAY          0
dtype: int64
```

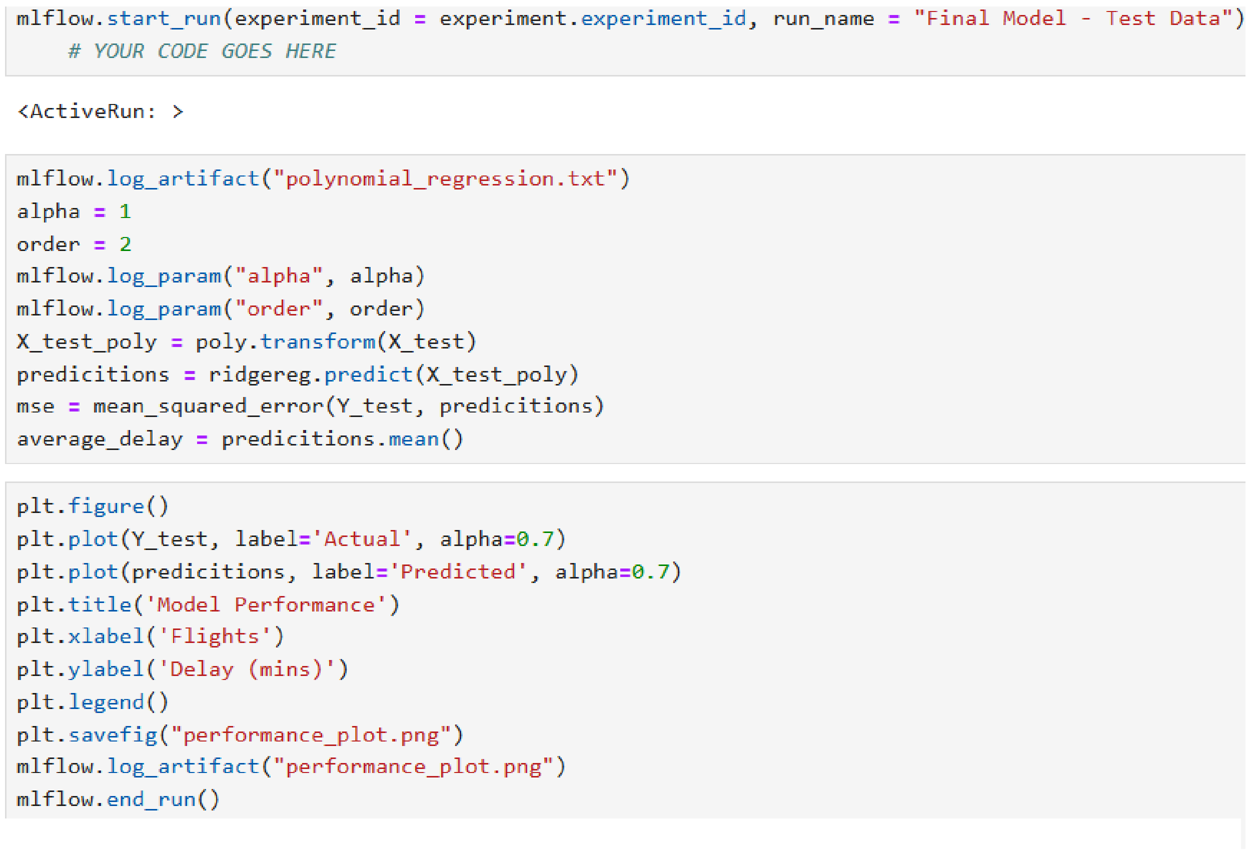
For my final cleaning step, I eliminated all extra and unneeded trailing spaces in every cell that had them. This helps keep the data consistent.



```python
for col in filtered_df.select_dtypes(include='object').columns:
    filtered_df[col] = filtered_df[col].map(lambda x: x.strip() if isinstance(x, str) else x)
```

D. Using the code template provided in the GitLab repository, implement an MLFlow experiment that captures the features listed in the comments within the poly_regressor file for either Python or R.

Working with someone else's code was challenging, as I needed to run multiple tests to finally debug the script fully. I did so by downloading the Jupyter notebook version and going through it step by step. Below is my code added near the end.

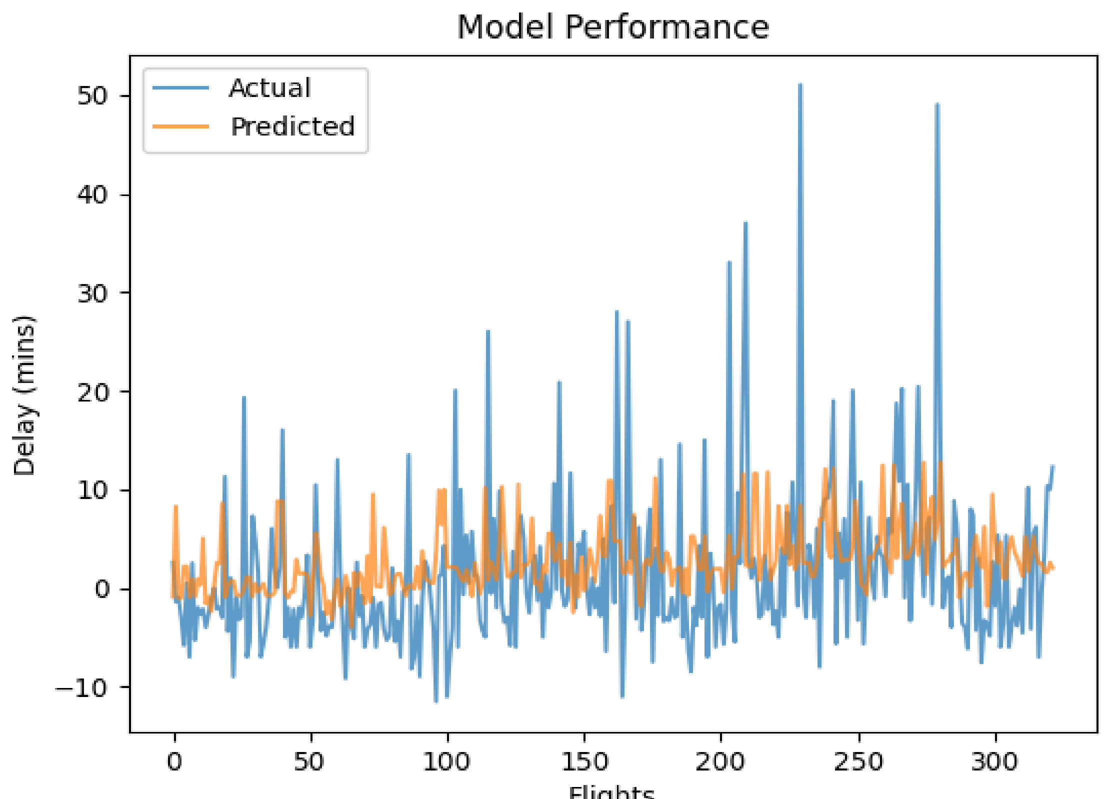


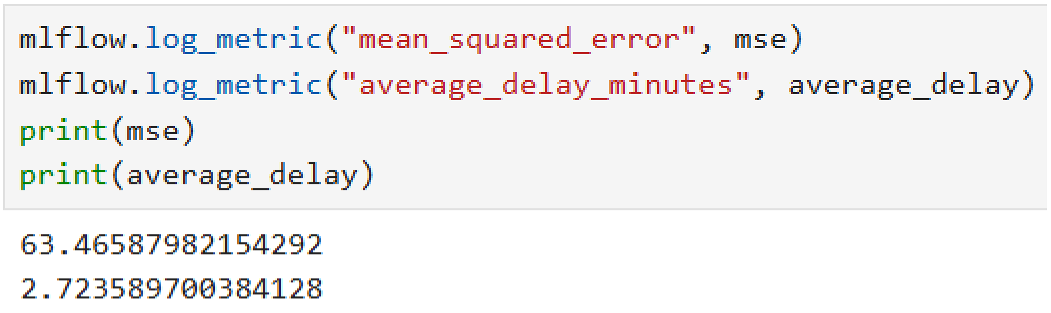
```
mlflow.start_run(experiment_id = experiment.experiment_id, run_name = "Final Model - Test Data")
    # YOUR CODE GOES HERE
```

```
<ActiveRun: >
```

```
mlflow.log_artifact("polynomial_regression.txt")
alpha = 1
order = 2
mlflow.log_param("alpha", alpha)
mlflow.log_param("order", order)
X_test_poly = poly.transform(X_test)
predicitions = ridgereg.predict(X_test_poly)
mse = mean_squared_error(Y_test, predicitions)
average_delay = predicitions.mean()
```

```
plt.figure()
plt.plot(Y_test, label='Actual', alpha=0.7)
plt.plot(predicitions, label='Predicted', alpha=0.7)
plt.title('Model Performance')
plt.xlabel('Flights')
plt.ylabel('Delay (mins)')
plt.legend()
plt.savefig("performance_plot.png")
mlflow.log_artifact("performance_plot.png")
mlflow.end_run()
```





```python
mlflow.log_metric("mean_squared_error", mse)
mlflow.log_metric("average_delay_minutes", average_delay)
print(mse)
print(average_delay)
```

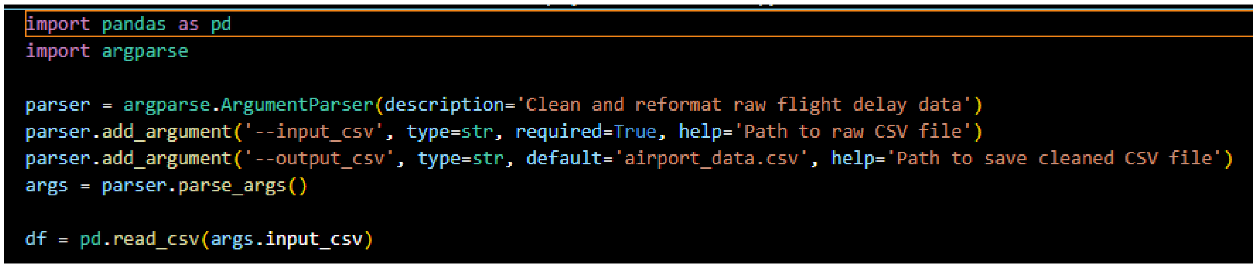
```
63.46587982154292
2.723589700384128
```

E. Using the provided YAML file for either Python or R, write an MLProject file that links the two scripts you wrote in parts B and C with the modified poly_regressor script from part D.

This part was the most challenging for me as I had gotten more errors than I ever had before, and at one point wasn't even sure I was starting in the right place. The task seemed simple: use all three scripts in tandem, and I knew that each one worked separately, so what could possibly go wrong? After a whole day of spinning my gears and getting nowhere, I reset and went back into my research to essentially start from scratch.

First, I installed MLflow onto my machine using pip. This is the package that's essential and responsible for what we are trying to achieve. Next, I downloaded miniconda, this was the key I was missing. Conda creates a virtual environment for the scripts to run in, which ensures all the parameters of the code language and packages installed are the same. Allowing whatever project you are working on can run on any machine.

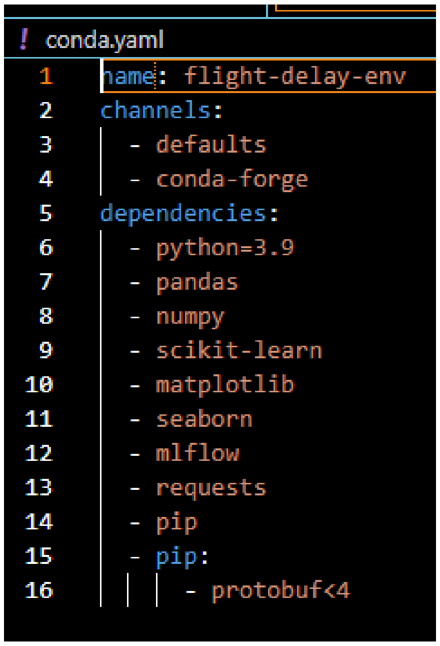
After this, I had to go back and modify my original code to allow MLflow to link all scripts together and convert them to .py files and not notebook files.



```python
import pandas as pd
import argparse

parser = argparse.ArgumentParser(description='Clean and reformat raw flight delay data')
parser.add_argument('--input_csv', type=str, required=True, help='Path to raw CSV file')
parser.add_argument('--output_csv', type=str, default='airport_data.csv', help='Path to save cleaned CSV file')
args = parser.parse_args()

df = pd.read_csv(args.input_csv)
```

The parser argument is what allowed MLflow to read each script.

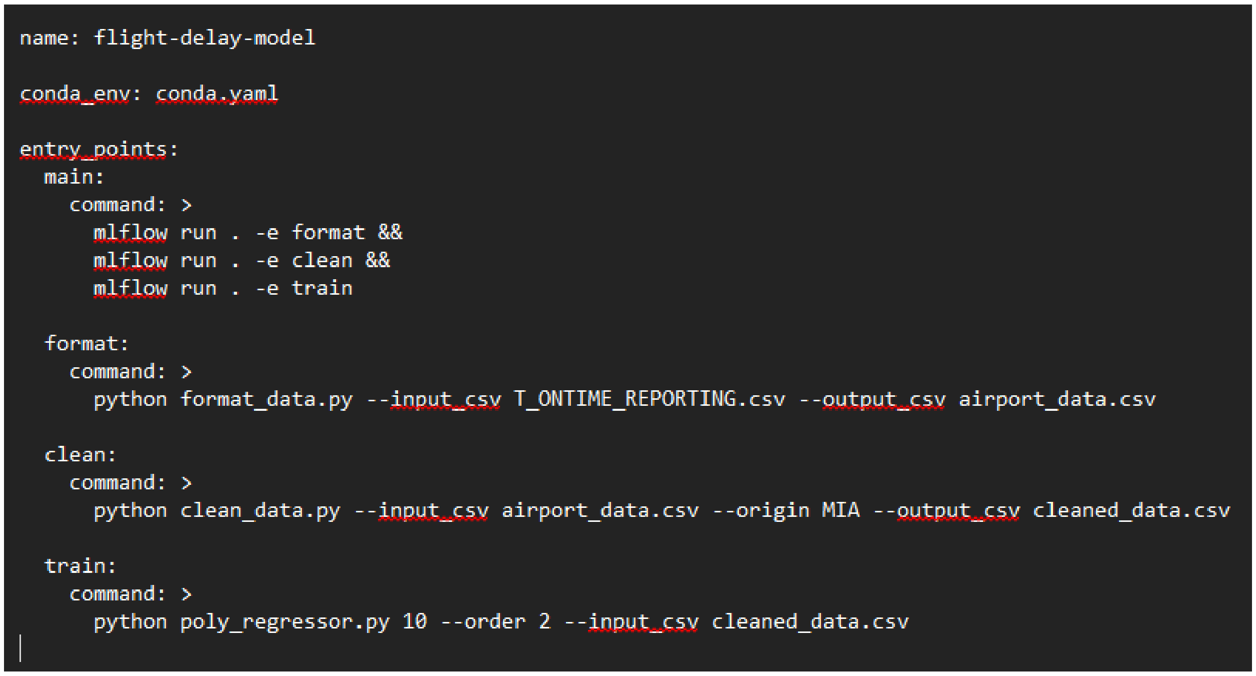
I then changed the given YAML file to fit my environment



conda.yaml

```yaml
name: flight-delay-env
channels:
  - defaults
  - conda-forge
dependencies:
  - python=3.9
  - pandas
  - numpy
  - scikit-learn
  - matplotlib
  - seaborn
  - mlflow
  - requests
  - pip
  - pip:
    - protobuf<4
```

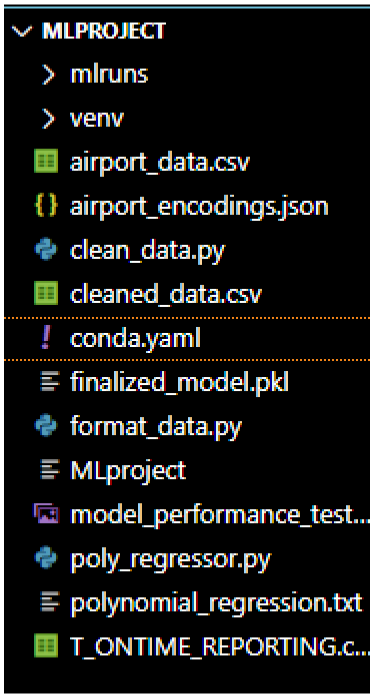
Finally, the file that's vital is the MLproject file, which is a script on its own.



```
name: flight-delay-model

conda_env: conda.yaml

entry_points:
  main:
    command: >
      mlflow run . -e format &&
      mlflow run . -e clean &&
      mlflow run . -e train

  format:
    command: >
      python format_data.py --input_csv T_ONTIME_REPORTING.csv --output_csv airport_data.csv

  clean:
    command: >
      python clean_data.py --input_csv airport_data.csv --origin MIA --output_csv cleaned_data.csv

  train:
    command: >
      python poly_regressor.py 10 --order 2 --input_csv cleaned_data.csv
```

Essentially, it is stating what environment to use, what scripts to run and in what order, and what CSV files to use for each.

Now that we have all our files, it's time to run MLflow. To do this, I first initialized the project file, which was done earlier for the DVC file as well. Then we create a pipeline from the folder in the terminal. Also in the terminal, I executed the "code ." command, which opened VS Studio Code. To show I'm in the right spot, I could see all my files.



```
MLPROJECT
  > mlruns
  > venv
  airport_data.csv
  airport_encodings.json
  clean_data.py
  cleaned_data.csv
  conda.yaml
  finalized_model.pkl
  format_data.py
  MLproject
  model_performance_test...
  poly_regressor.py
  polynomial_regression.txt
  T_ONTIME_REPORTING.c...
```

*This was taken after MLflow succeeded

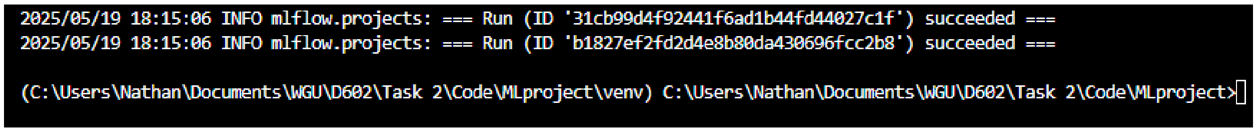
I then created the virtual environment and activated it within the terminal with the following commands:

"Create -p venv python==3.10 -y"

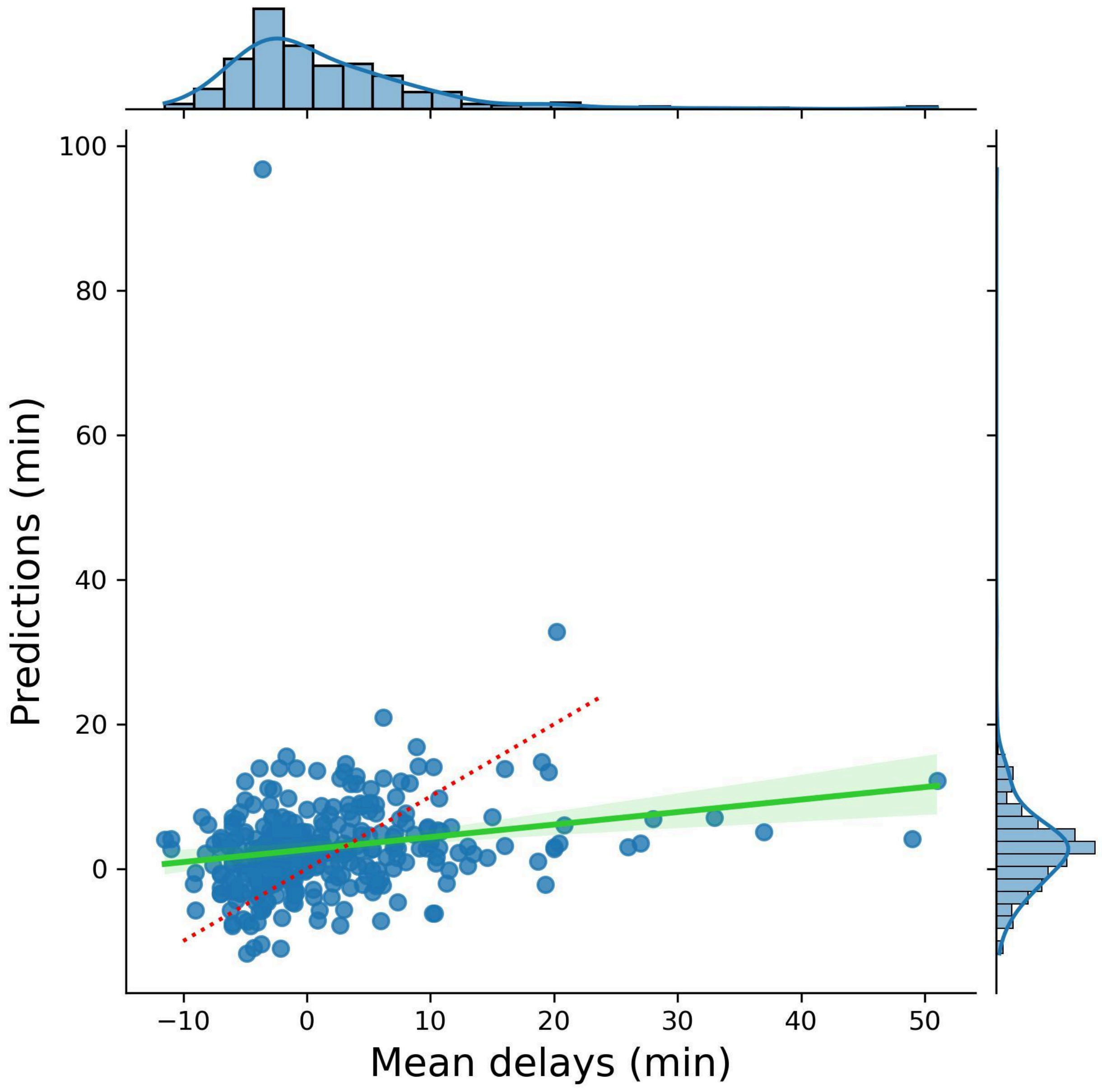
"Conda activate venv"

The stage is finally set, we have all our files, and the virtual environment is ready to go. Finally, all we need to do is run the final command "mlflow run ."

After some time, I got my first error; luckily, it did make it past the first two scripts, the formatting and cleaning data scripts, but the problem was with the given poly_regressor script. So after more debugging and more errors, I was finally able to get the succeeded alert.



```
2025/05/19 18:15:06 INFO mlflow.projects: === Run (ID '31cb99d4f92441f6ad1b44fd44027c1f') succeeded ===
2025/05/19 18:15:06 INFO mlflow.projects: === Run (ID 'b1827ef2fd2d4e8b80da430696fcc2b8') succeeded ===

(C:\Users\Nathan\Documents\WGU\D602\Task 2\Code\MLproject\venv) C:\Users\Nathan\Documents\WGU\D602\Task 2\Code\MLproject>
```



Sources

No sources were used except WGU Material