Deployment on Flask

Name: **Rana Fakeeh**
Batch code: **LISUM28**
Submission date: **12/28/2023**
Submitted to:

I. Summary

The created web application utilizes the dataset used in week 2 assignment (i.e. the *G2M insight for Cab Investment firm*) and predicts the total daily profit of *Yellow Cab* and *Pink Cab* companies using *MLForecast* library. The */predict* route will receive the future *days* to forecast and the *company* name and responses with HTML of rendered interactive line charts and pie chart of the results using *Plotly* library.

II. Development Steps

1. Select and pre-process data. To reduce the size of the model, the data for the top three frequent cities were selected. Categorical features were converted into dummies. Lastly, the feature names were modified as per the conventions of the *MLForecast* model. Check Figure 1.
2. Data was split into train and test sets and the model was fitted. *MLForecast* model performs recursive fitting of multiple *Scikit-Learn* models and automatically generates the log features of the target as provided in the *fit()* method. It allows the forecasting of multiple timeseries identified by *unique_id* which was the combination of company name and city name. Therefore, the profit was forecasted by company by city level. Lastly, the fitted model was serialized using *Pickle*. Check Figure 2.

```
data_path = 'data/'

df = pd.read_csv(data_path+'Cab_Data.csv')
df['Date of Travel'] = pd.to_datetime(df['Date of Travel'], origin='1899-12-30', unit='d')
df['Profit of Trip'] = df['Price Charged'] - df['Cost of Trip']
df = df[['Transaction ID', 'Date of Travel', 'Company', 'City', 'Profit of Trip']]
df = df.round(2)

top_3_cities = df['City'].value_counts()[:3].index.tolist()
print(top_3_cities)

df = df[df['City'].isin(top_3_cities)]
print(df.head(3), end='\n\n')

df1 = df.groupby(['Date of Travel','Company','City'], as_index=False).agg(Total_Profit = ('Profit of Trip','sum'))
print(df1.head(3), end='\n\n')

pvt = pd.pivot_table(data=df1, index=['City','Company'], columns=['Date of Travel'], values=['Total_Profit'])
pvt = pvt.fillna(0)
df2 = pvt.stack().reset_index()
print(df2.head(3), end='\n\n')

df2_cat = df2.select_dtypes(include=['object'])
print(df2_cat.head(3), end='\n\n')

df2_cat = df2_cat.replace(r'[\s]+','_',regex=True)
df2_dummies = pd.get_dummies(df2_cat, drop_first=True)
print(df2_dummies.head(3), end='\n\n')

df3 = df2.join(df2_dummies)
df3 = df3.rename(columns={'Date of Travel':'ds'})
df3['unique_id'] = df3['Company']+'_'+df3['City']
df3['unique_id'] = df3['unique_id'].replace(r'[\s]+','_',regex=True)
df3 = df3.drop(['Company','City'], axis=1)
print(df3.head(3), end='\n\n')

static_features = df2_dummies.columns.tolist()
print(static_features, end='\n\n')

ts_profit = df3.copy()
ts_profit = ts_profit.rename(columns={'Total_Profit':'y'})
print(ts_profit.head(3), end='\n\n')
```

Figure 1: development step 1 (model.py)

```
ts_profit_train = ts_profit[ts_profit['ds']<'2018-10-01']
ts_profit_test = ts_profit[ts_profit['ds']>='2018-10-01']

models = {
    'LinearRegression':make_pipeline(StandardScaler(),MinMaxScaler(),LinearRegression()),
    'Lasso':make_pipeline(StandardScaler(),MinMaxScaler(),Lasso(random_state=0)),
    'Ridge':make_pipeline(StandardScaler(),MinMaxScaler(),Ridge(random_state=0)),
    'KNeighborsRegressor':make_pipeline(StandardScaler(),MinMaxScaler(),KNeighborsRegressor()),
    'SVR':make_pipeline(StandardScaler(),MinMaxScaler(),SVR()),
    'RandomForestRegressor':RandomForestRegressor(random_state=0),
    'XGBRegressor':XGBRegressor(random_state=0),
}

fcst = MLForecast(
    models=models,
    freq='D',
    lags=[1,7,30,365],
    date_features=['day','dayofweek','week','month','quarter','year'],
    num_threads=8
)

preprocessed = fcst.preprocess(ts_profit_train)
print(preprocessed.head(3), end='\n\n')

fcst.fit(ts_profit_train, time_col='ds', id_col='unique_id', target_col='y', static_features=static_features)
print(fcst, end='\n\n')

unique_ids = ts_profit_train['unique_id'].unique().tolist()
print(unique_ids, end='\n\n')

with open('profit_model.pkl', 'wb') as file:
    pickle.dump((fcst,unique_ids), file)

profit_model = None
unique_ids = None
with open('profit_model.pkl', 'rb') as file:
    (profit_model,unique_ids) = pickle.load(file)
```

Figure 2: development step 2 (model.py)

3. The backend was developed using *Flask*. Two routes were defined: the / (root) which serves the fronted *index.html* page, and the /*predict* which deserialize the model, accepts two form fields: *days* and *company*, forecasts the daily profit, and returns HTML code for rendering the interactive *Plotly* charts visualizing the prediction results. Check Figures 3, 4.

```
7
8   from flask import Flask, request, render_template
9   import pickle
10  import re
11  import plotly_express as px
12
13  app = Flask(__name__)
14
15  @app.route('/')
16  def home():
17      return render_template('index.html')
18
19  @app.route('/predict', methods=['POST'])
20  def predict():
21
22      profit_model = None
23      unique_ids = None
24      with open('model/profit_model.pkl', 'rb') as file:
25          profit_model, unique_ids = pickle.load(file)
26
27      days = int(request.form['days'])
28      company = request.form['company']
29
30      company = re.sub(r'[\s]+', '_', company)
31
32      uids = [uid for uid in unique_ids if company in uid]
33
34      ts_profit_pred = profit_model.predict(days, ids=uids)
35      ts_profit_pred['profit'] = ts_profit_pred[['XGBRegressor','RandomForestRegressor']].mean(axis=1)
36      ts_profit_pred = ts_profit_pred.copy()[['unique_id','ds','profit']]
37      ts_profit_pred['unique_id'] = ts_profit_pred['unique_id'].str.replace(company+'_','')
38      ts_profit_pred = ts_profit_pred.rename(columns={'unique_id':'city'})
39
```

Figure 3: development step 3 part 1 (app.py)

```
45      ts_profit_pred_summary = ts_profit_pred_summary.melt(id_vars='ds')
46      ts_profit_pred_summary = ts_profit_pred_summary.copy().rename(columns={'value':'profit'})
47
48      fig1 = px.line(ts_profit_pred_summary,
49                     x='ds',
50                     y='profit',
51                     color='variable',
52                     height=600,
53                     title=company+'\'s Total Profit For Next '+str(days)+' Day(s)')
54
55      fig2 = px.line(ts_profit_pred,
56                     x='ds',
57                     y='profit',
58                     color='city',
59                     height=600,
60                     title=company+'\'s Total Profit For Next '+str(days)+' Day(s) By City')
61
62      fig3 = px.pie(ts_profit_pred,
63                names='city',
64                values='profit',
65                height=600,
66                title=company+'\'s Total Profit For Next '+str(days)+' Day(s) By City')
67
68      return render_template('index.html',
69                             profit_line_html=fig1.to_html(full_html=False),
70                             profit_per_city_line_html=fig2.to_html(full_html=False),
71                             profit_per_city_pie_html=fig3.to_html(full_html=False))
72
73
74  if __name__ == "__main__":
75      app.run(debug=True)
```

Figure 4: development step 3 part 2 (app.py)

4. The frontend was created using *HTML5*, *CSS3*, *Bootstrap v5*, and *JQuery*. The request to the backend was developed using *Javascript*. Check Figures 5, 6, 7.

```
<div class="row" id="form">
    <div class="col">
        <form name="forecast-form" action="{{ url_for('predict') }}" method="POST">
          <div class="mb-3">
            <label class="form-label"><h4>Select Company:</h4></label>
            <br/>

            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio" name="company" id="yellow" value="Yellow
              Cab" checked>
              <label class="form-check-label" for="yellow">Yellow Cab</label>
            </div>

            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio" name="company" id="pink" value="Pink Cab">
              <label class="form-check-label" for="pink">Pink Cab</label>
            </div>

          </div>
          <div class="mb-3">
            <label for="days" class="form-label"><h4>Select Forecast Days:</h4></label>
            <div class="slider-container">
                <input type="text" id="days" class="slider" name="days" value="" />
            </div>
          </div>
          <br/>
          <button type="button" id="btn-submit" onclick="printLoading()" class="btn btn-primary">
          Forecast</button>
        </form>
    </div>
</div>
```

Figure 5: development step 4 part 1 (index.html)

```
<div class="row" id="result">
    <div class="col">

        <div class="row">
            <div class="col">

                {{ profit_line_html | safe }}

            </div>
        </div>

        <div class="row">
            <div class="col">

                {{ profit_per_city_line_html | safe }}

            </div>
        </div>

        <div class="row">
            <div class="col">

                {{ profit_per_city_pie_html | safe }}

            </div>
        </div>
    </div>
</div>
```

Figure 6: development step 4 part 2 (index.html)

```
        function printLoading() {
            document.getElementById('btn-submit').innerHTML = 'Loading ...';
            document.getElementById('btn-submit').setAttribute('disabled', 'disabled');
            document.forms['forecast-form'].submit();
        }

    </script>
```

Figure 7: development step 4 part 3 (index.html)

III. Deployment Steps

1. The spyder IDE is opened. The *app.py* file is opened and run using the Run file button. Check Figure 8.

```
C:\Users\ranaf\anaconda3\python.exe
 * 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 watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
```

Figure 8: deployment step 1

2. Go to *http://localhost:5000/* and view the app on *index* page. Check Figure 9.

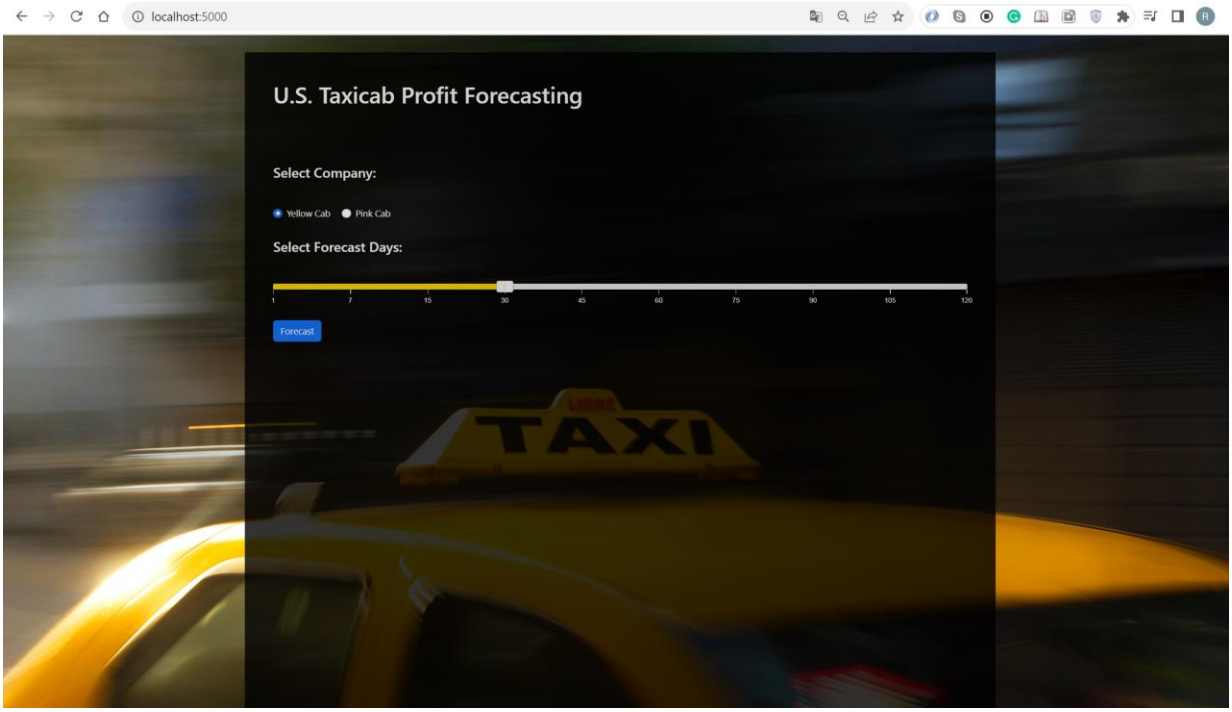

Figure 9: deployment step 2

3. Test the app using the interface by selecting different company name and forecasting days. Hover on points and click on legend values to interact with the plots. Check Figures 10, 11, 12.

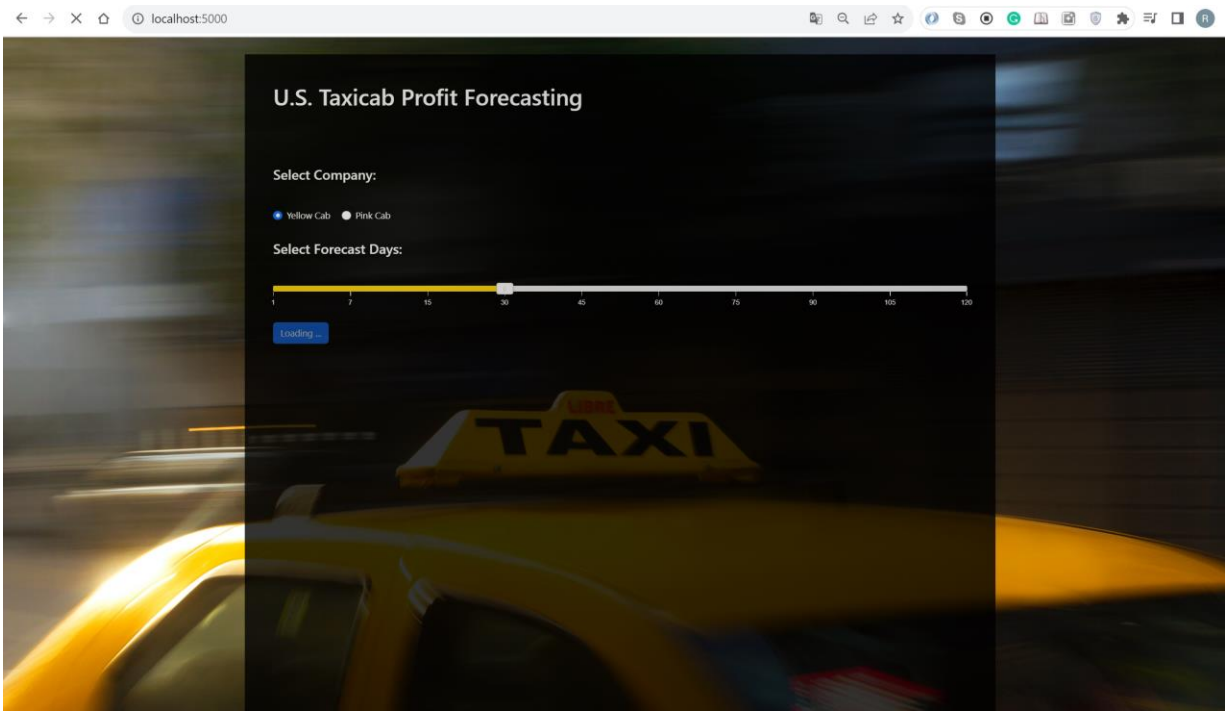

Figure 10: deployment step 3 part 1

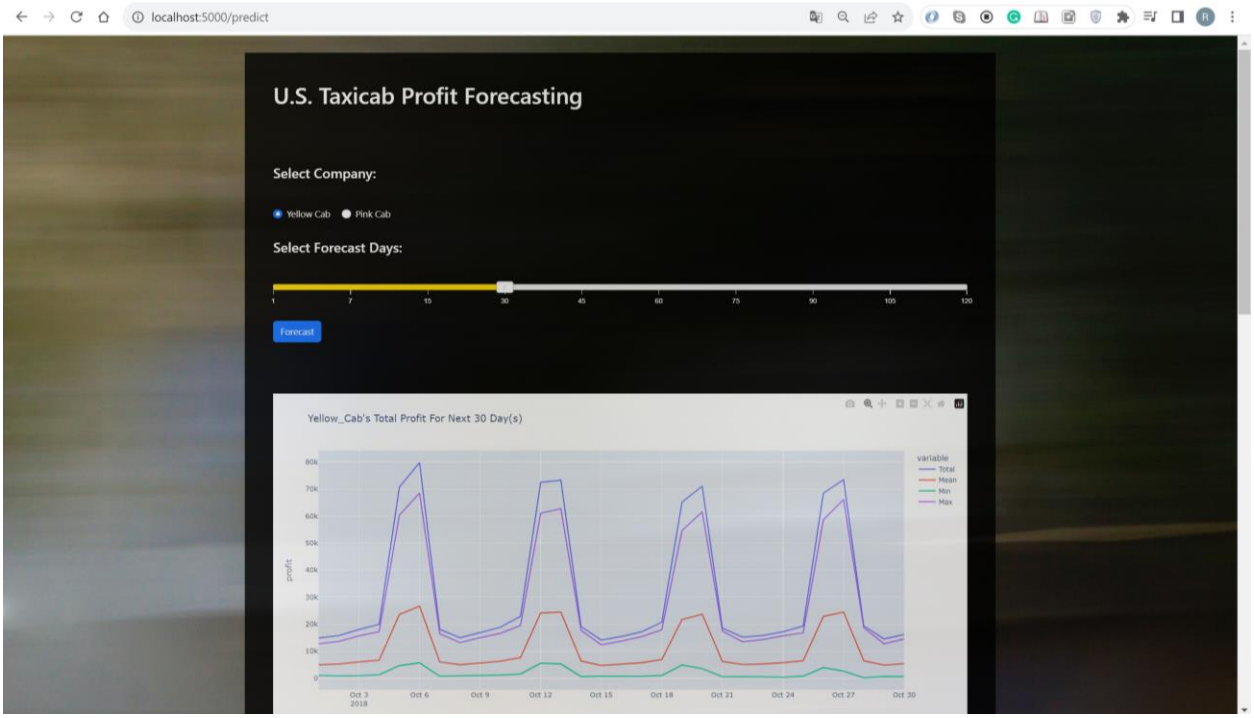

Figure 11: deployment step 3 part 2

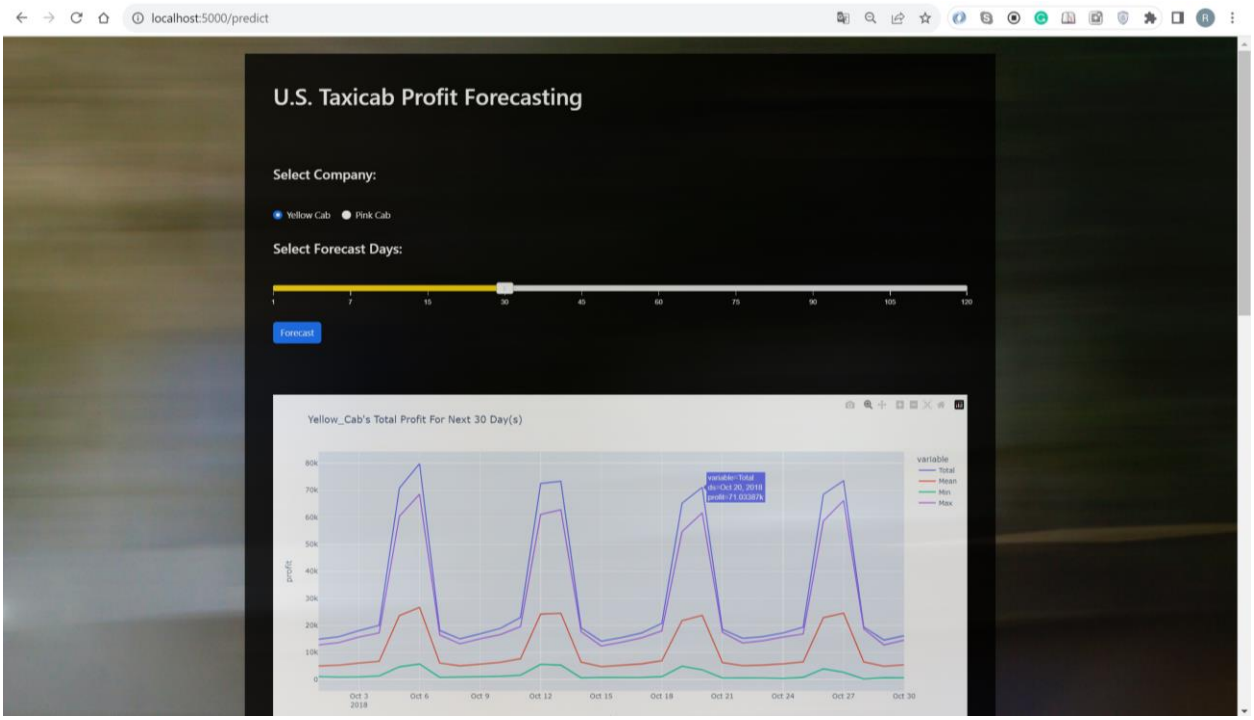

Figure 12: deployment step 3 part 3