Cloud and API Deployment

Name: **Rana Fakeeh**
Batch code: **LISUM28**
Submission date: **12/29/2023**
Submitted to:

I. Summary

The created web API 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* endpoint will receive the future *days* to forecast and the *company* name and responses with JSON data needed to render the 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 APIs 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 a JSON data for rendering the interactive *Plotly* charts visualizing the prediction results. Check Figures 3, 4.

```
from flask import Flask, request, jsonify, render_template
import pickle
import re
import json
import plotly_express as px

app = Flask(__name__)

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

@app.route('/predict', methods=['POST'])
def predict():

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

    days = int(request.form['days'])
    company = request.form['company']

    company = re.sub(r'[\s]+', '_', company)

    uids = [uid for uid in unique_ids if company in uid]

    ts_profit_pred = profit_model.predict(days, ids=uids)
    ts_profit_pred['profit'] = ts_profit_pred[['XGBRegressor','RandomForestRegressor']].mean(axis=1)
    ts_profit_pred = ts_profit_pred.copy()[['unique_id','ds','profit']]
    ts_profit_pred['unique_id'] = ts_profit_pred['unique_id'].str.replace(company+'_','')
    ts_profit_pred = ts_profit_pred.rename(columns={'unique_id':'city'})

```

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

```
    ts_profit_pred_summary = ts_profit_pred_summary.melt(id_vars='ds')
    ts_profit_pred_summary = ts_profit_pred_summary.copy().rename(columns={'value':'profit'})

    fig1 = px.line(ts_profit_pred_summary,
                   x='ds',
                   y='profit',
                   color='variable',
                   height=600,
                   title=company+'\'s Total Profit For Next '+str(days)+' Day(s)')

    fig2 = px.line(ts_profit_pred,
                   x='ds',
                   y='profit',
                   color='city',
                   height=600,
                   title=company+'\'s Total Profit For Next '+str(days)+' Day(s) By City')

    fig3 = px.pie(ts_profit_pred,
             names='city',
             values='profit',
             height=600,
             title=company+'\'s Total Profit For Next '+str(days)+' Day(s) By City')



    return jsonify({'profit_line_plot':json.loads(fig1.to_json()),
                    'profit_per_city_line_plot':json.loads(fig2.to_json()),
                    'profit_per_city_pie_plot':json.loads(fig3.to_json())})


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

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

4. The endpoint */predict* was tested locally before deployment on the cloud using *Postman*. Check Figure 5.

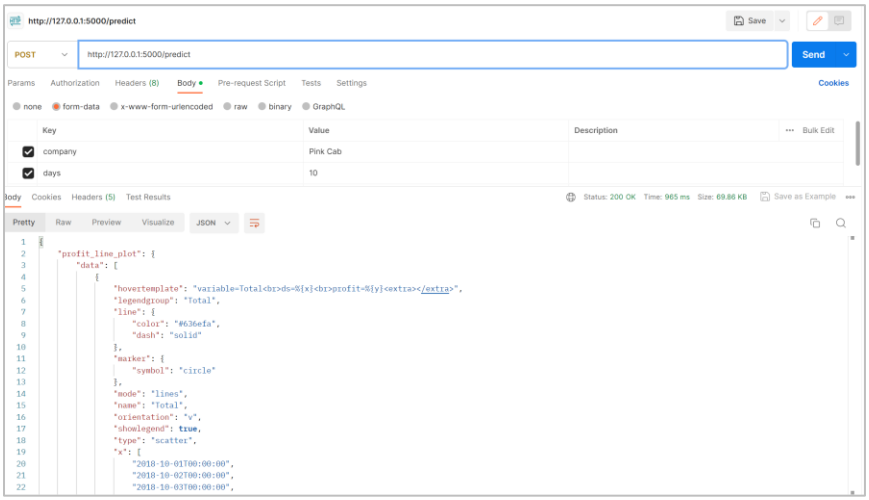

Figure 5: development step 4 Postman

5. The frontend was created using *HTML5*, *CSS3*, *Bootstrap v5*, and *JQuery*. The request to the backend's API was developed using *Ajax*. Check Figures 6, 7, 8.
6. Finally, the backend code (*app.py*), model code (*model.py*) and frontend files (in *static* and *templates* folders) were pushed onto *GitHub* repository. Check Figure 9.

```
<div class="row" id="form">
    <div class="col">
        <form action="/predict" method="POST" id="forecastform">
          <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="{{ days }}" />
            </div>
          </div>
          <br/>
          <button type="submit" class="btn btn-primary" id="submit">Forecast</button>
        </form>
    </div>
</div>
```

Figure 6: development step 5 part 1 (index.html)

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

        <div class="row">
            <div class="col" id="profit_line"></div>
        </div>

        <div class="row">
            <div class="col" id="profit_per_city_line"></div>
        </div>

        <div class="row">
            <div class="col" id="profit_per_city_pie"></div>
        </div>
    </div>
</div>
```

Figure 7: development step 5 part 2 (index.html)

```
$("#submit").click(function(e) {
    e.preventDefault();
    $("#submit").html("Loading...");
    var form = $("#forecastform");
    var url = form.attr("action");
    $.ajax({
            type: "POST",
            url: url,
            data: form.serialize(),
            success: function(data) {

                $("#submit").html("Forecast");

                // Ajax call completed successfully
                console.log("Form Submited Successfully");
                console.log(data);

                Plotly.newPlot( document.getElementById("profit_line"),
                                data.profit_line_plot.data,
                                data.profit_line_plot.layout);

                Plotly.newPlot( document.getElementById("profit_per_city_line"),
                                data.profit_per_city_line_plot.data,
                                data.profit_per_city_line_plot.layout);

                Plotly.newPlot( document.getElementById("profit_per_city_pie"),
                                data.profit_per_city_pie_plot.data,
                                data.profit_per_city_pie_plot.layout);

            },
            error: function(err) {

                // Some error in ajax call
                console.log("Some Error");
                console.log(err);
            }
        });
});
```

Figure 8: development step 5 part 3 (index.html)

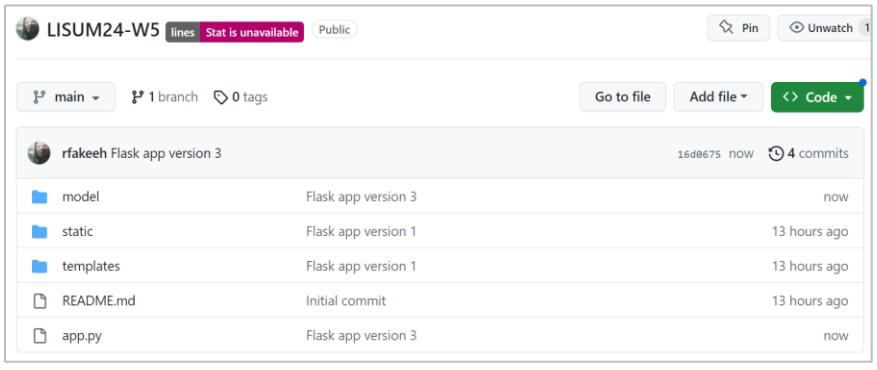

Figure 9: development step 6 GitHub

III. Deployment Steps

1. Go to *https://www.pythonanywhere.com/* and click on *Create Beginner account*. This kind of account is free for three months. Follow the steps for creating the account and sign in. Check Figure 10.

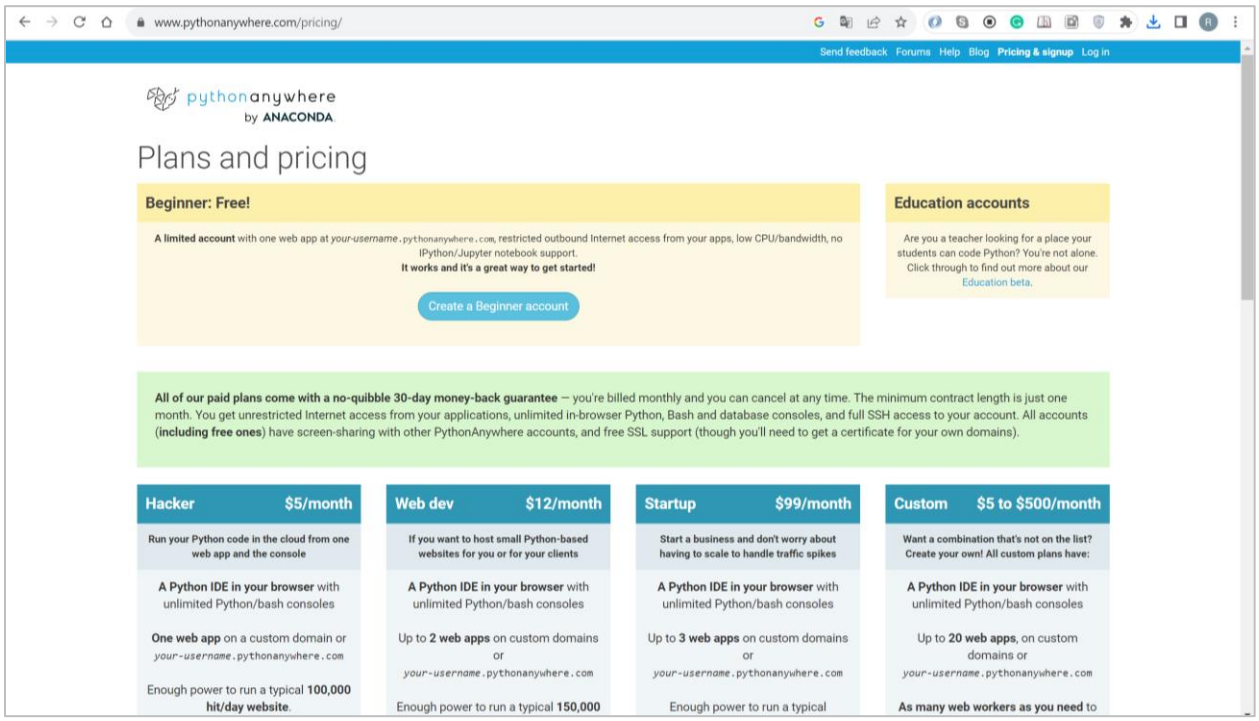

Figure 10: deployment step 1

2. On *Dashboard* page, navigate to *Web* page from top right menu. Check Figure 11.

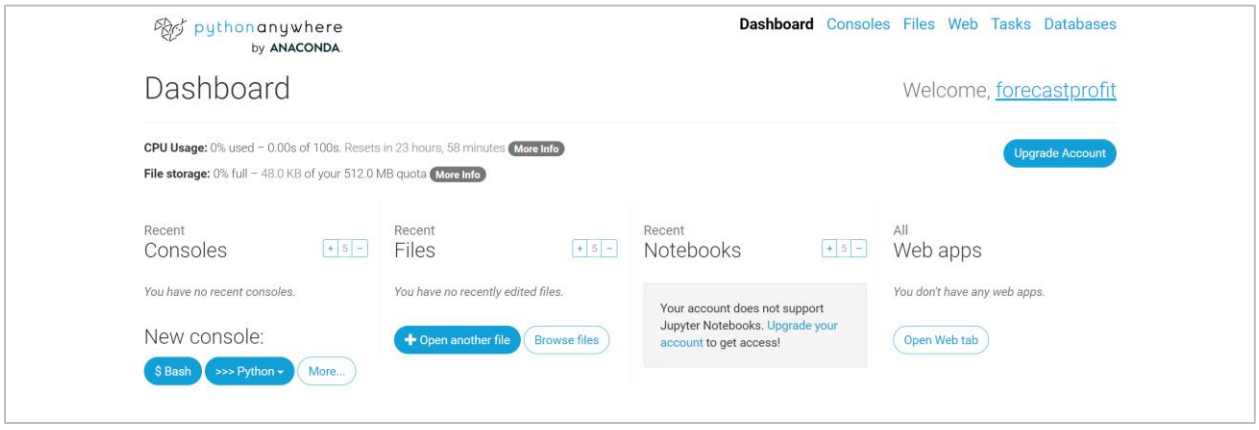

Figure 11: deployment step 2

3. On *Web* page, click on *Add a new web app* and follow the steps to launch the initial Flask web app. Check Figures 12, 13, 14, 15, 16.
4. On *Web* page, navigate to *Files* page from top right menu and click on *mysite/* directory below *Directories* panel. Notice *flask_app.py* file. Check Figures 17, 18.

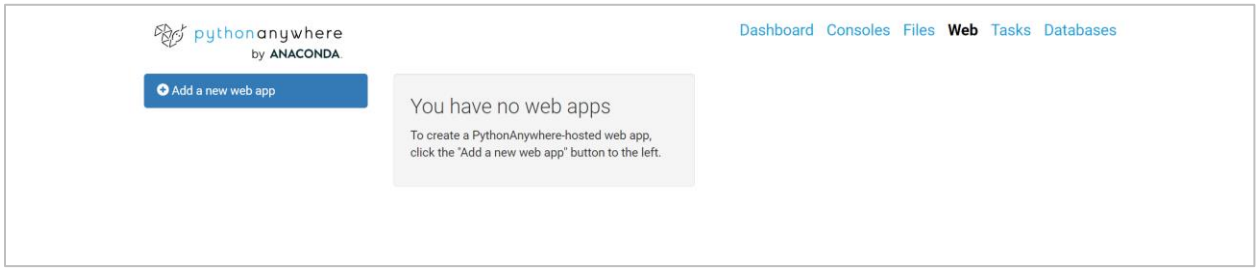

Figure 12: deployment step 3 part 1

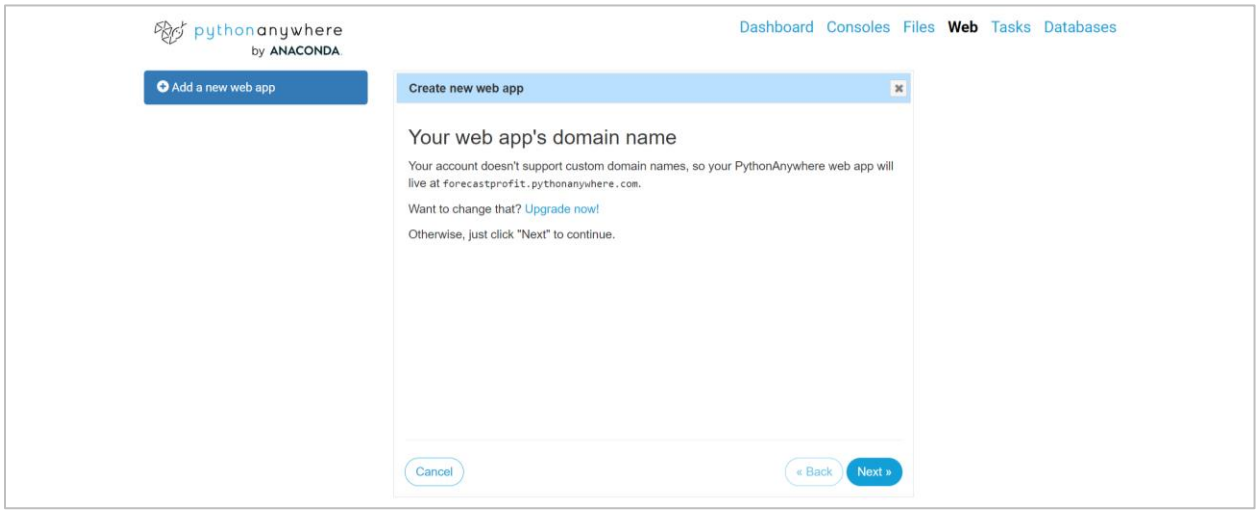

Figure 13: deployment step 3 part 2

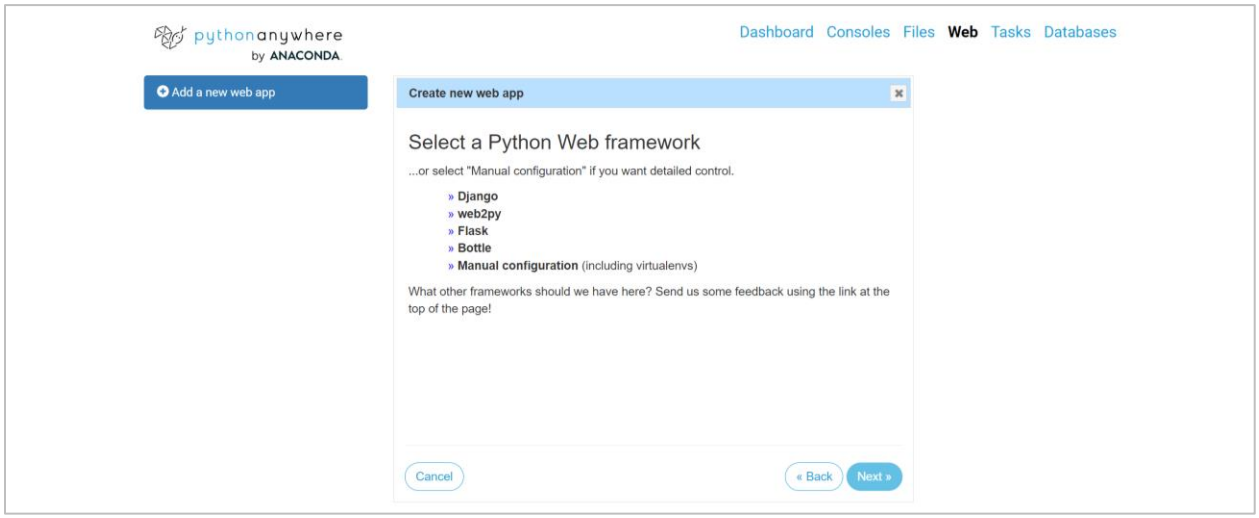

Figure 14: deployment step 3 part 3

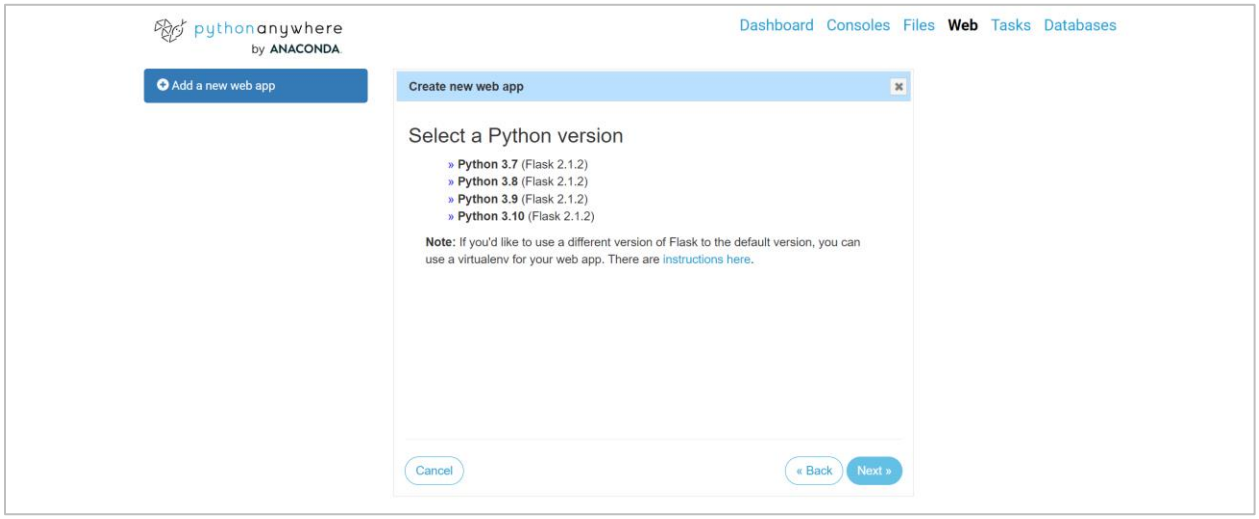

Figure 15: deployment step 3 part 4

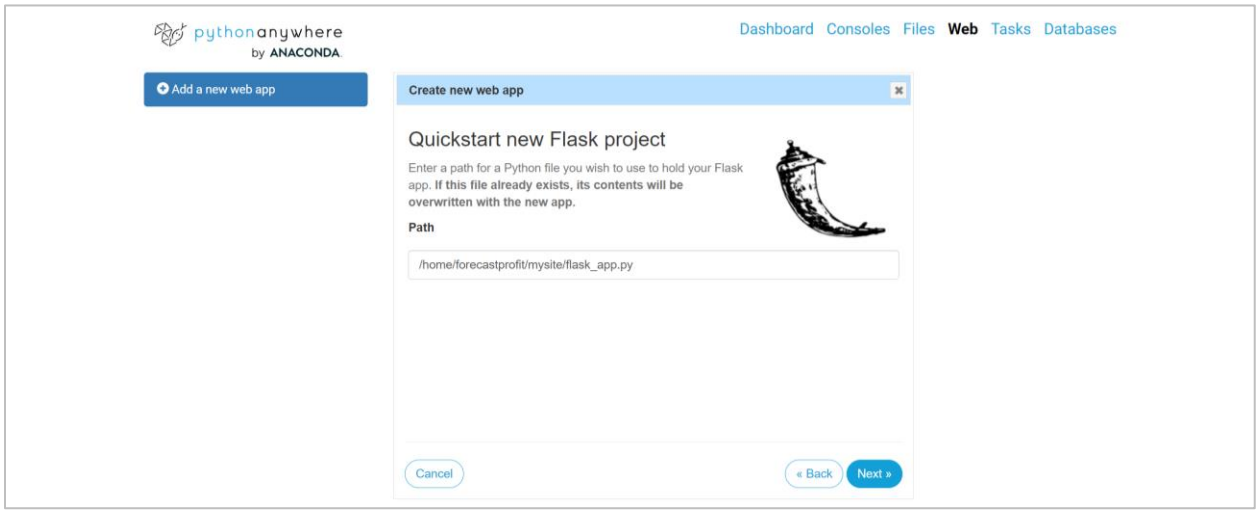

Figure 16: deployment step 3 part 5

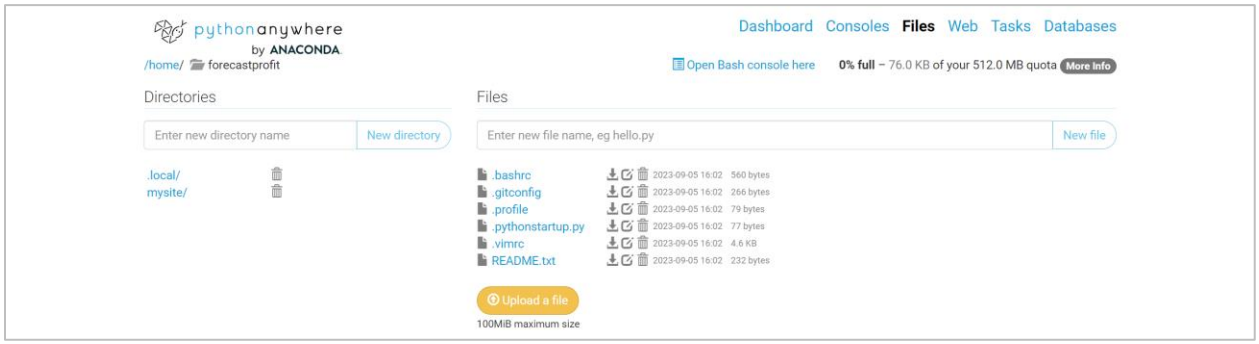

Figure 17: deployment step 4 part 1

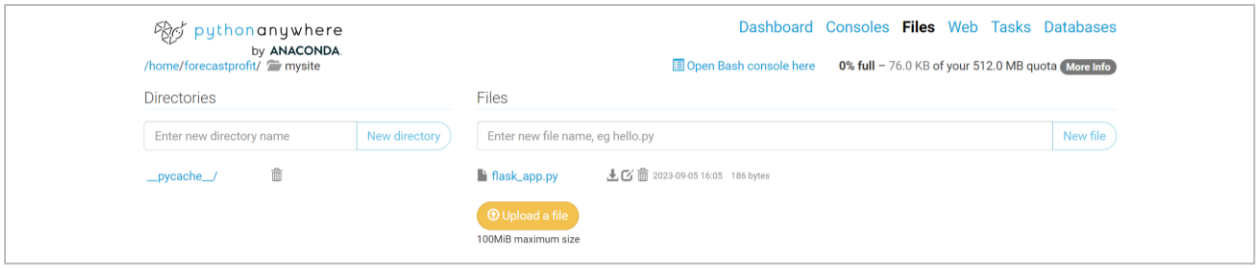

Figure 18: deployment step 4 part 2

5. On *mysite/* directory, click on *Open Bash console here* on top right to clone the application code from GitHub repository and install *mlforecast* and *plotly_express* libraries. Check Figures 19, 20.

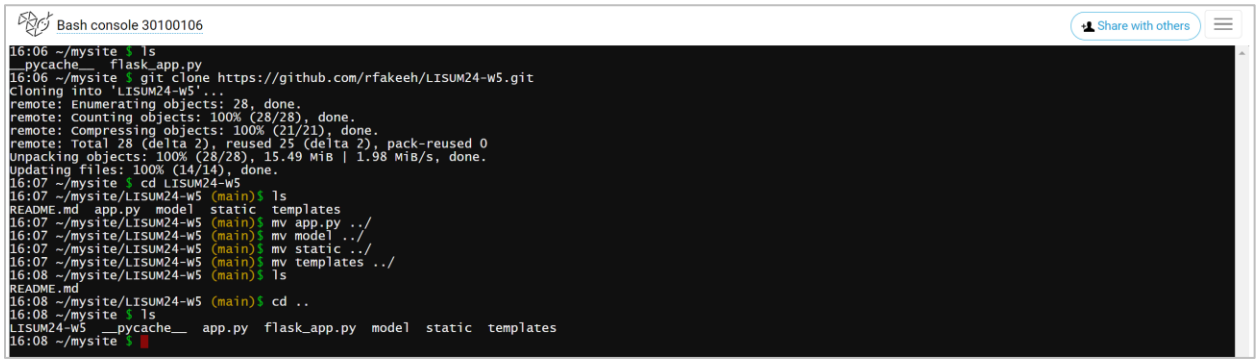

Figure 19: deployment step 5 part 1

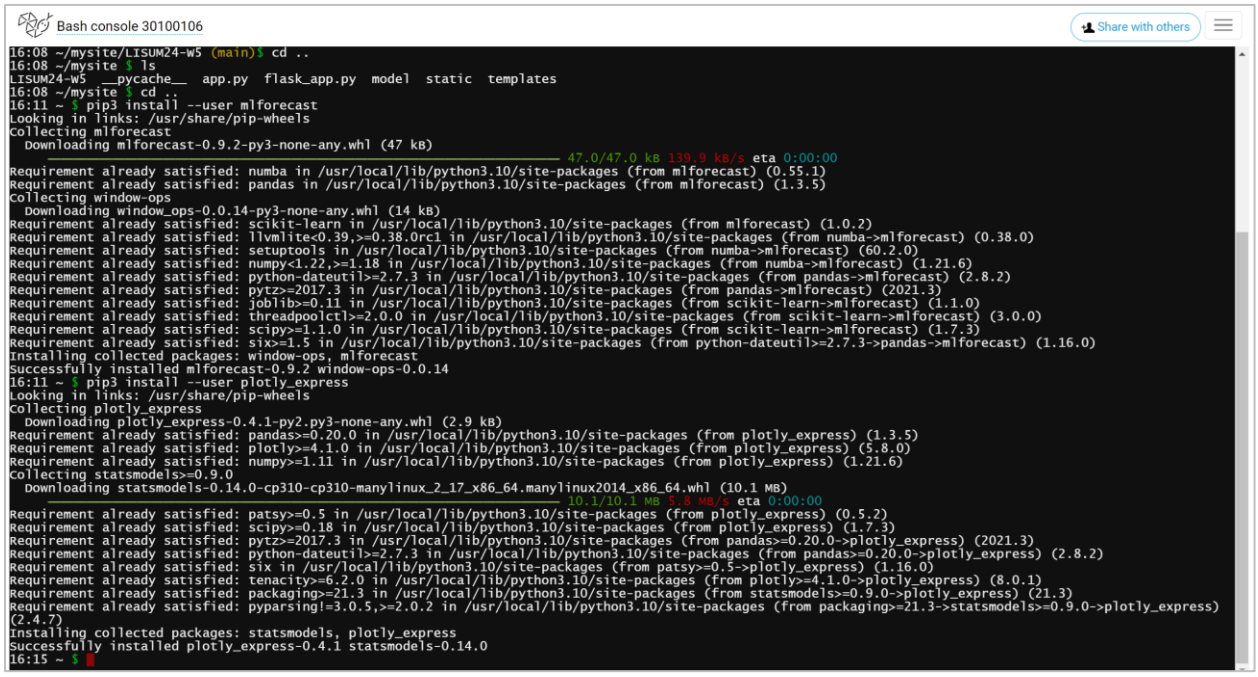

Figure 20: deployment step 5 part 2

6. On *mysite/* directory, navigate to *model/* directory, open and run *model.py* to generate the serialized model file. Check Figures 21, 22.

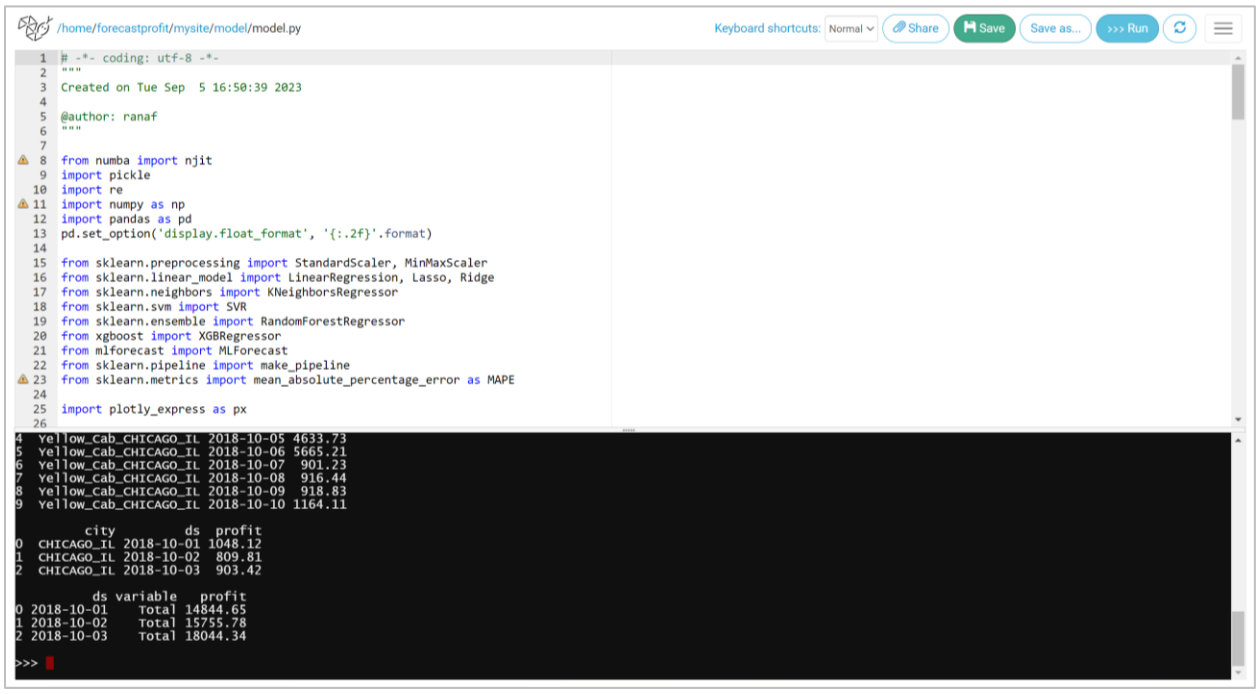

Figure 21: deployment step 6 part 1

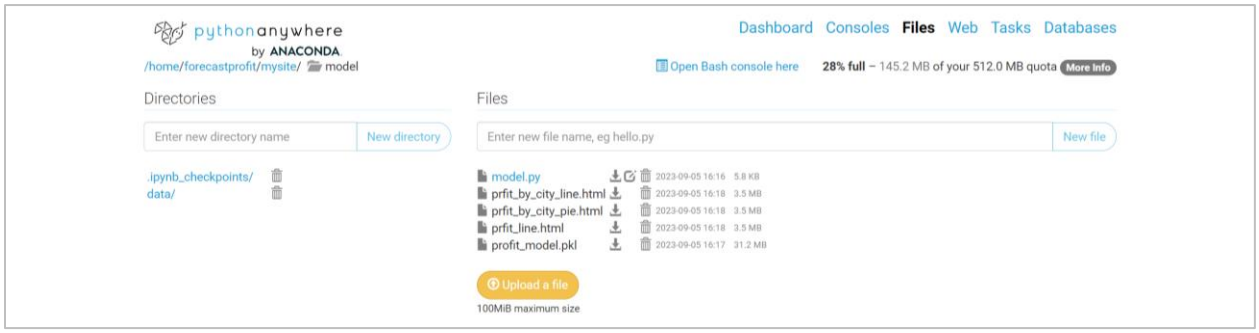

Figure 22: deployment step 6 part 2

7. Navigate to *mysite/* directory, open *app.py* file and copy the code, open *flask_app.py* and paste the code there. Save the absolute path to locate the serialized model. Finally, delete *app.py* file. Check Figure 23.

Figure 23: deployment step 7

8. Navigate back to *Web* page and click on *Reload* forecastprofit.pythonanywhere.com/ then click on the URL above it to run and view the app on the browser. Check Figures 24, 25.

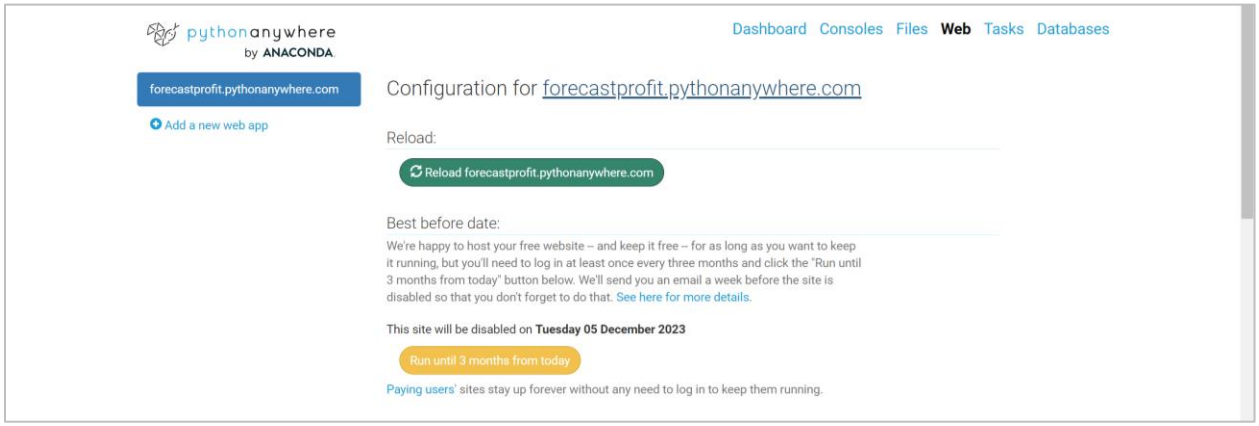

Figure 24: deployment step 8 part 1

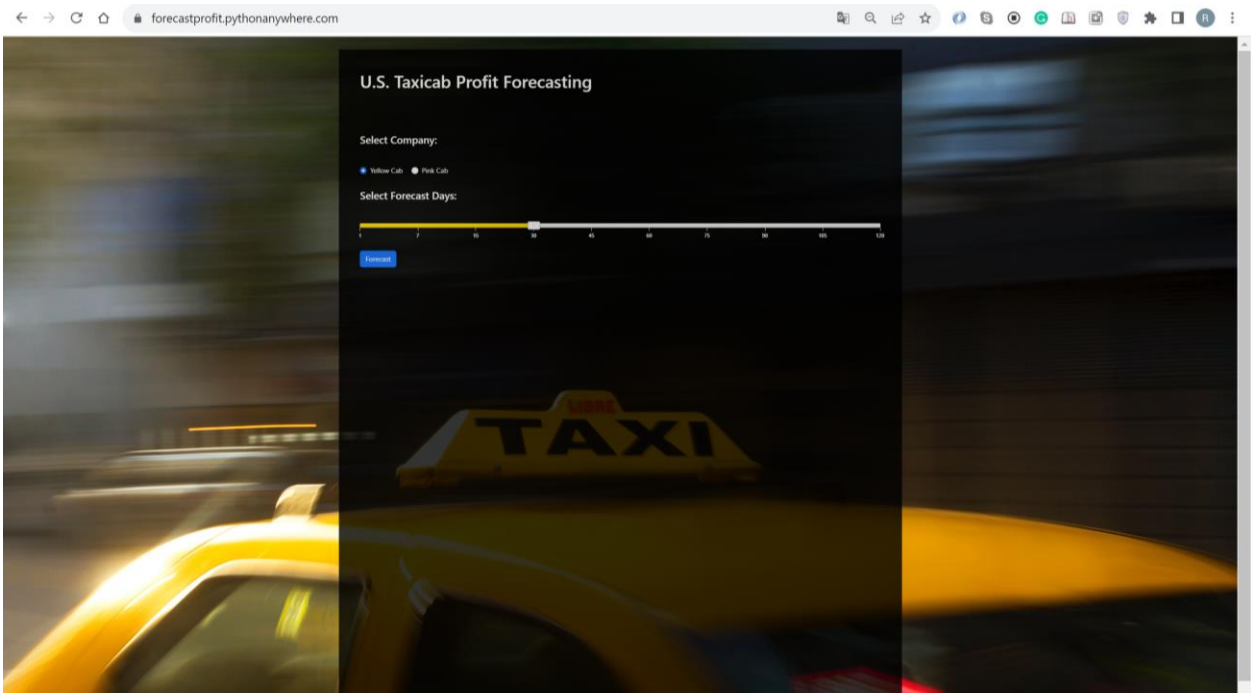

Figure 25: deployment step 8 part 2

9. Test the */predict* endpoint using *Postman*. Provide form data of *company* and *days*. The response is a JSON file with data for rendering interactive *Plotly* charts. Check Figure 26.

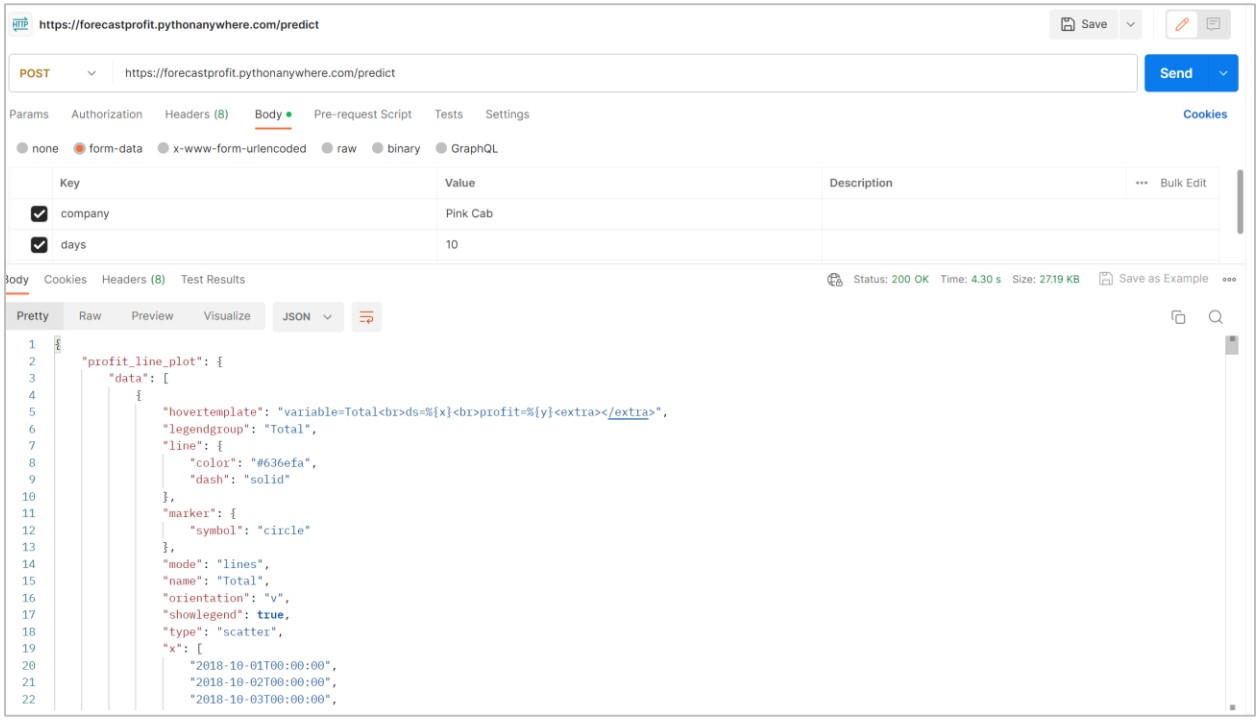

Figure 26: deployment step 9

10. Test the */predict* endpoint using the live app by selecting different company name and forecasting days. Hover on points and click on legend values to interact with the plots. Check all Figures below.

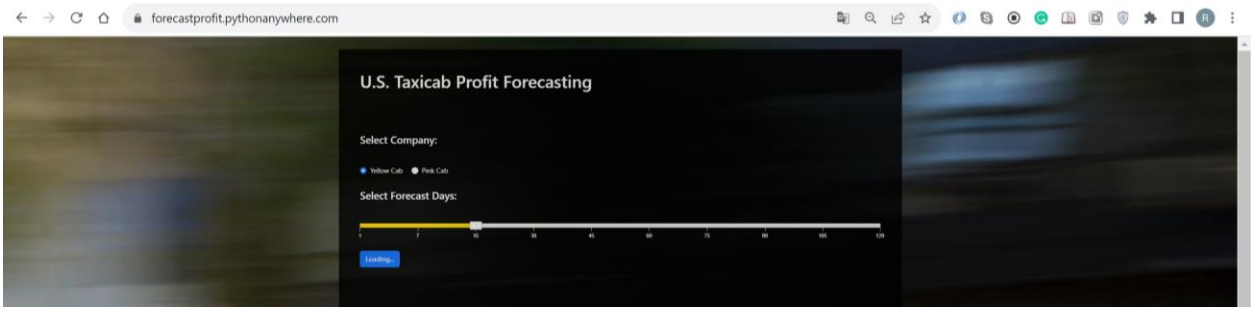

Figure 27: deployment step 10 part 1

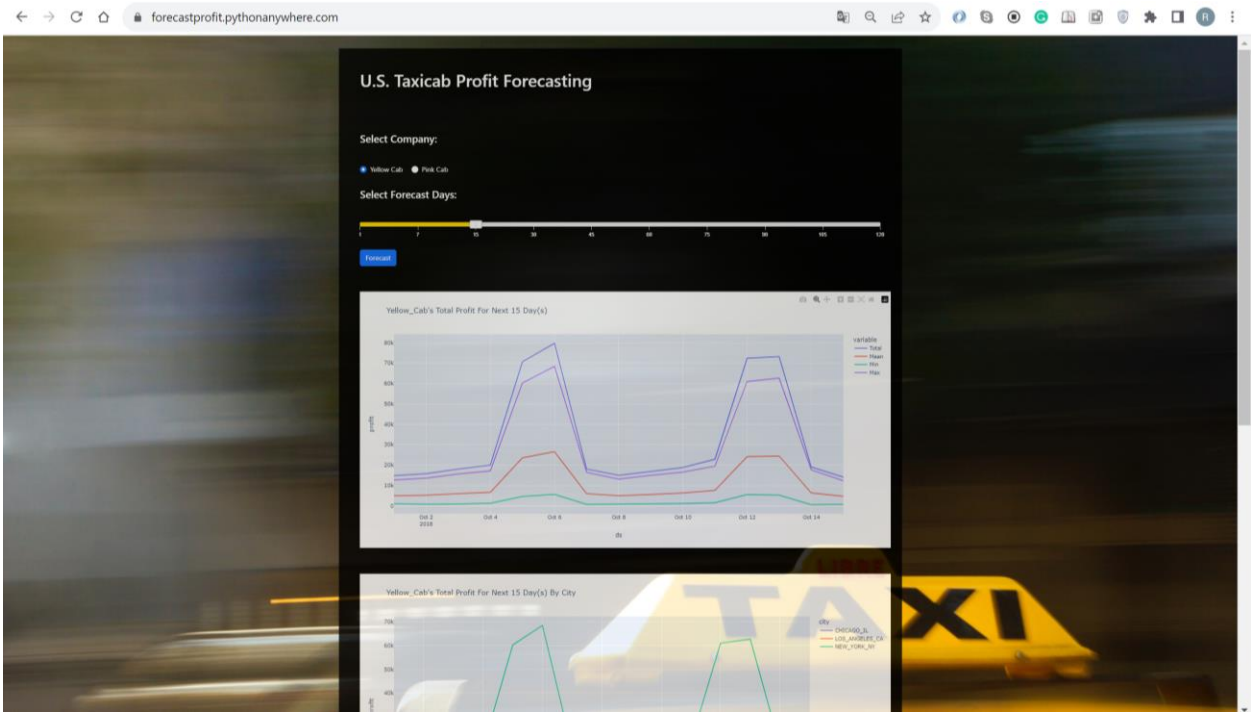

Figure 28: deployment step 10 part 2

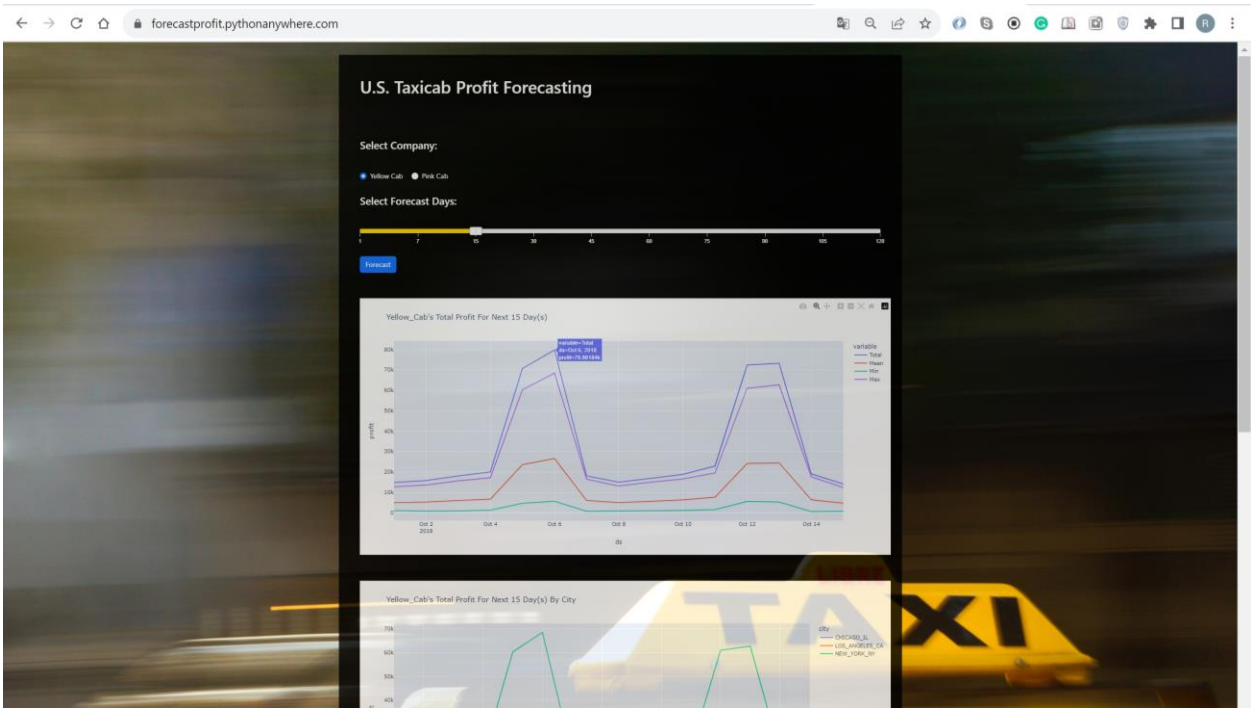

Figure 29: deployment step 10 part 3

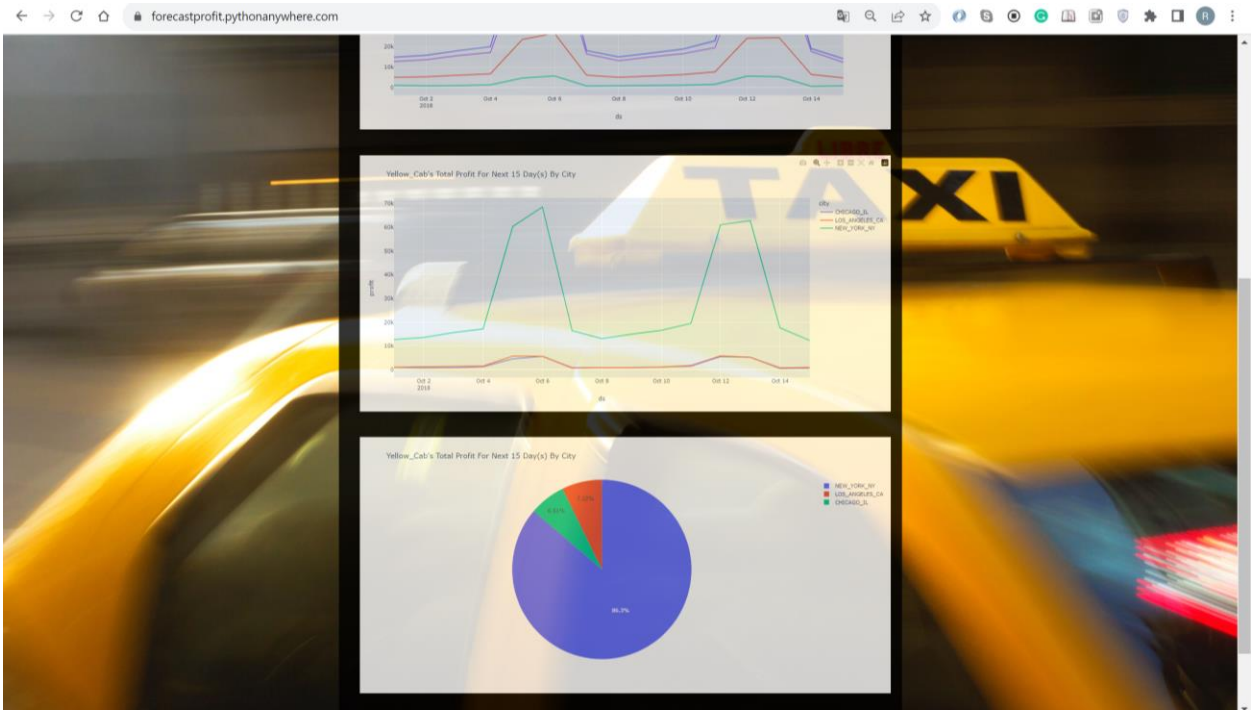

Figure 30: deployment step 10 part 4

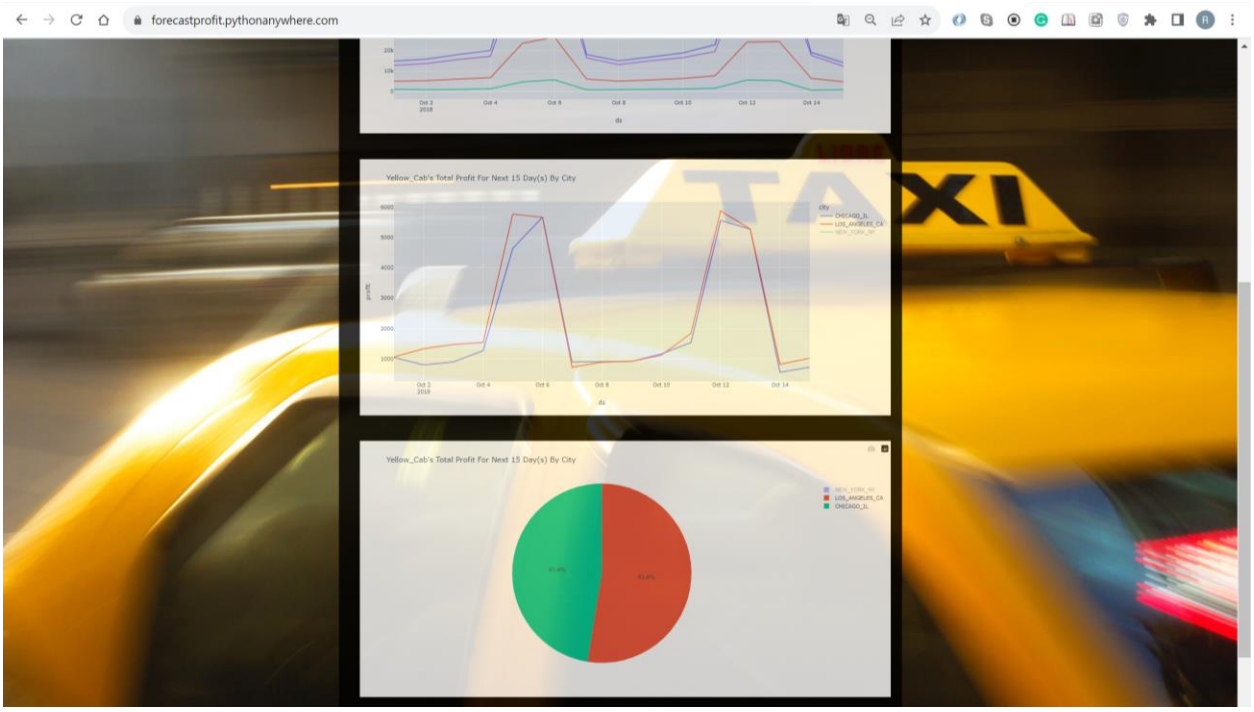

Figure 31: deployment step 10 part 5

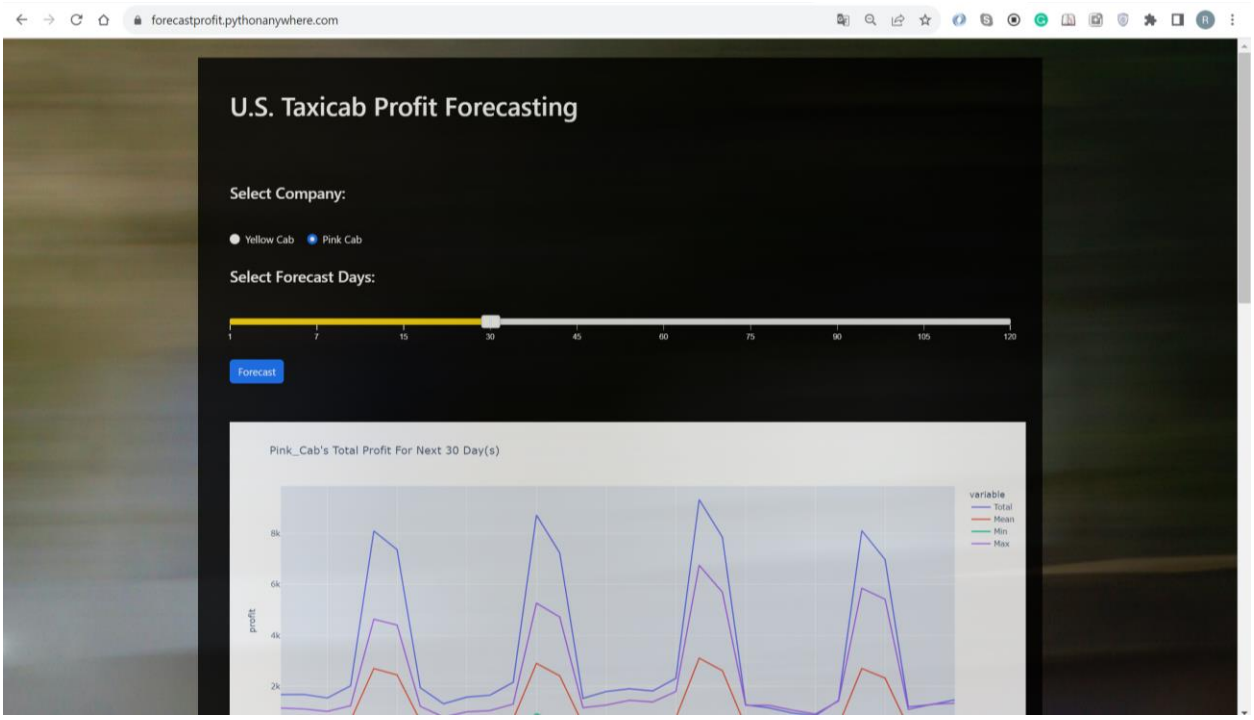

Figure 32: deployment step 10 part 6

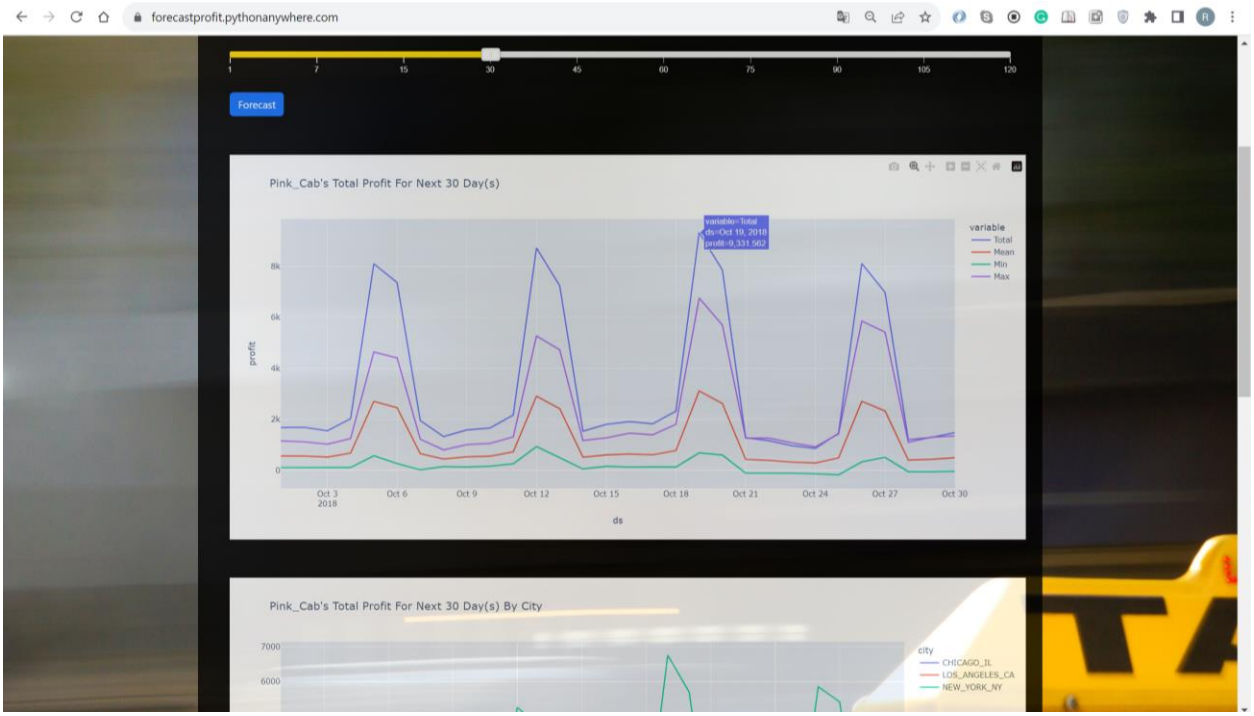

Figure 33: deployment step 10 part 7

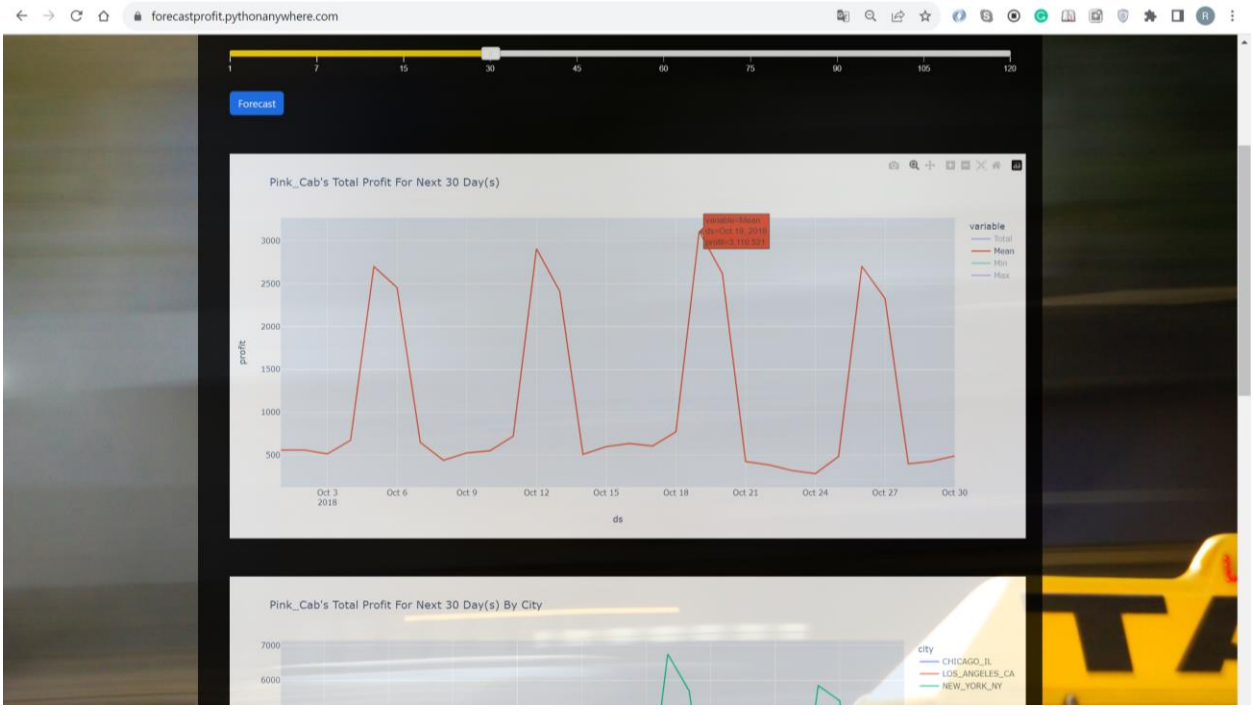

Figure 34: deployment step 10 part 8