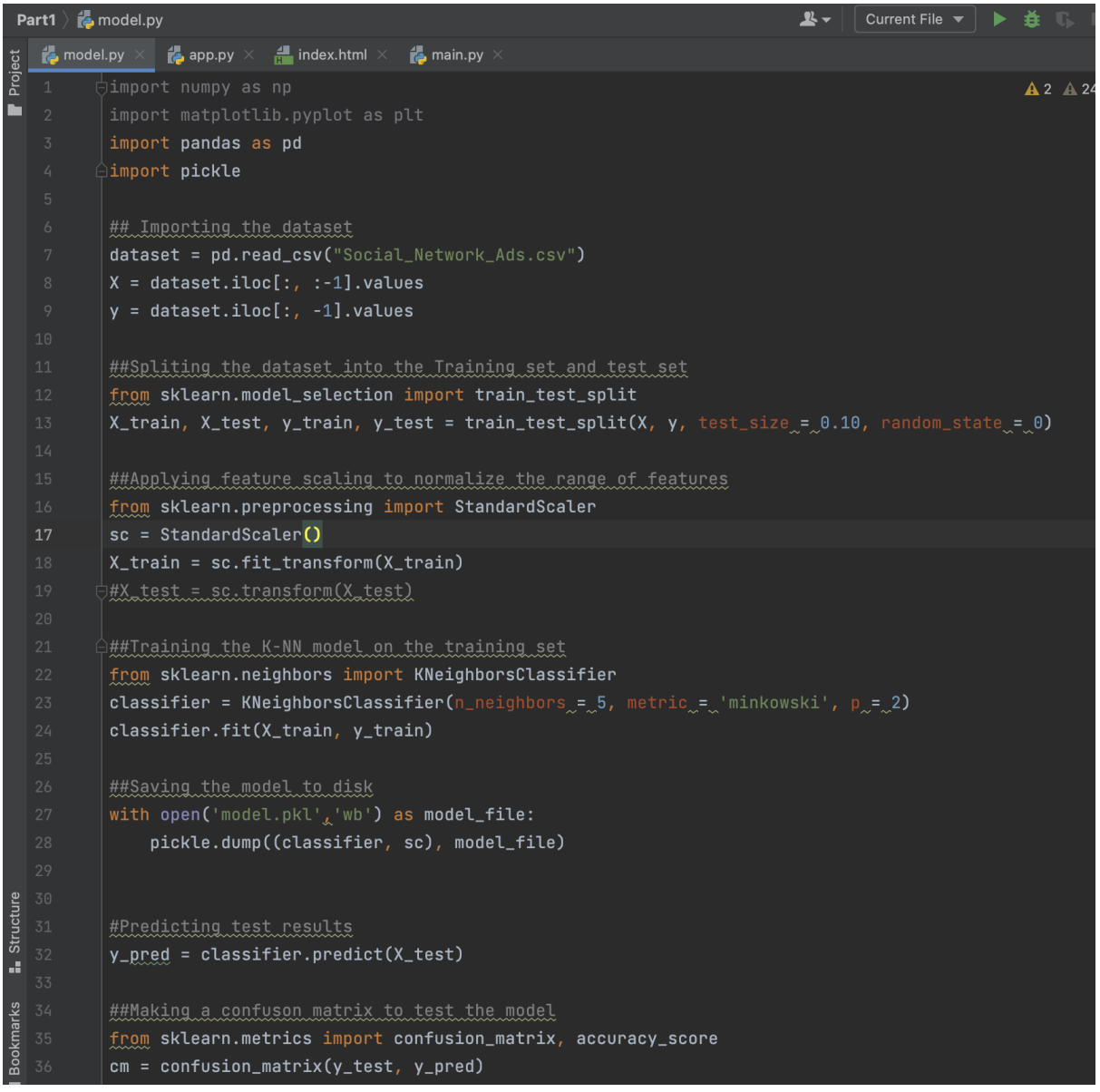
Model Deployment Project Report

Name: Predict Product Purchase Model
Report date: November 4, 2023
Internship Batch: LISUM26
Submitted by: Rupert Tawiah-Quashie
Submitted to: Github

This model takes in a person's age and yearly income and determines if they are likely to purchase the product. This product is promoted on social media platforms, and the data collected was based on the purchases made from social media platforms.

Below are the files I implemented:

Machine Learning Model: K-Nearest Neighbors



Importing necessary libraries:

The code begins by importing several Python libraries, including numpy for numerical operations, matplotlib.pyplot for data visualization, pandas for data manipulation, and pickle for serializing and saving the trained model to disk.

Importing the dataset:

- The code reads a CSV file named "Social_Network_Ads.csv" using the pandas library and stores the data in a variable called dataset. This dataset is made of 2 numerical variables
- It then separates the dataset into two parts:
- X: This variable contains the feature columns (all columns except the last one) of the dataset.
- y: This variable contains the target variable (the last column) of the dataset.

Splitting the dataset:

- The code splits the dataset into training and testing sets using the train_test_split function from sklearn.model_selection. It allocates 10% of the data for testing and uses a random seed (random_state) to ensure reproducibility.
- It results in four datasets:
- X_train: Features of the training set.
- X_test: Features of the test set.
- y_train: Target values of the training set.
- y_test: Target values of the test set.

Feature Scaling:

- Feature scaling is applied to normalize the range of feature values. This is important when using K-NN, as it relies on distance-based calculations.
- The code uses the StandardScaler from sklearn.preprocessing to standardize the feature values of the X_train dataset.

Training the K-NN model:

- The code imports the K-NN classifier (KNeighborsClassifier) from sklearn.neighbors.
- It creates an instance of the classifier with some parameters:
- n_neighbors: Number of neighbors to consider (set to 5 in this case).
- metric: The distance metric used, which is 'minkowski' in this case.
- p: The power parameter for the 'minkowski' metric, set to 2 (Euclidean distance).
- The K-NN model is then trained on the standardized training data (X_train and y_train) using the method of the classifier object.

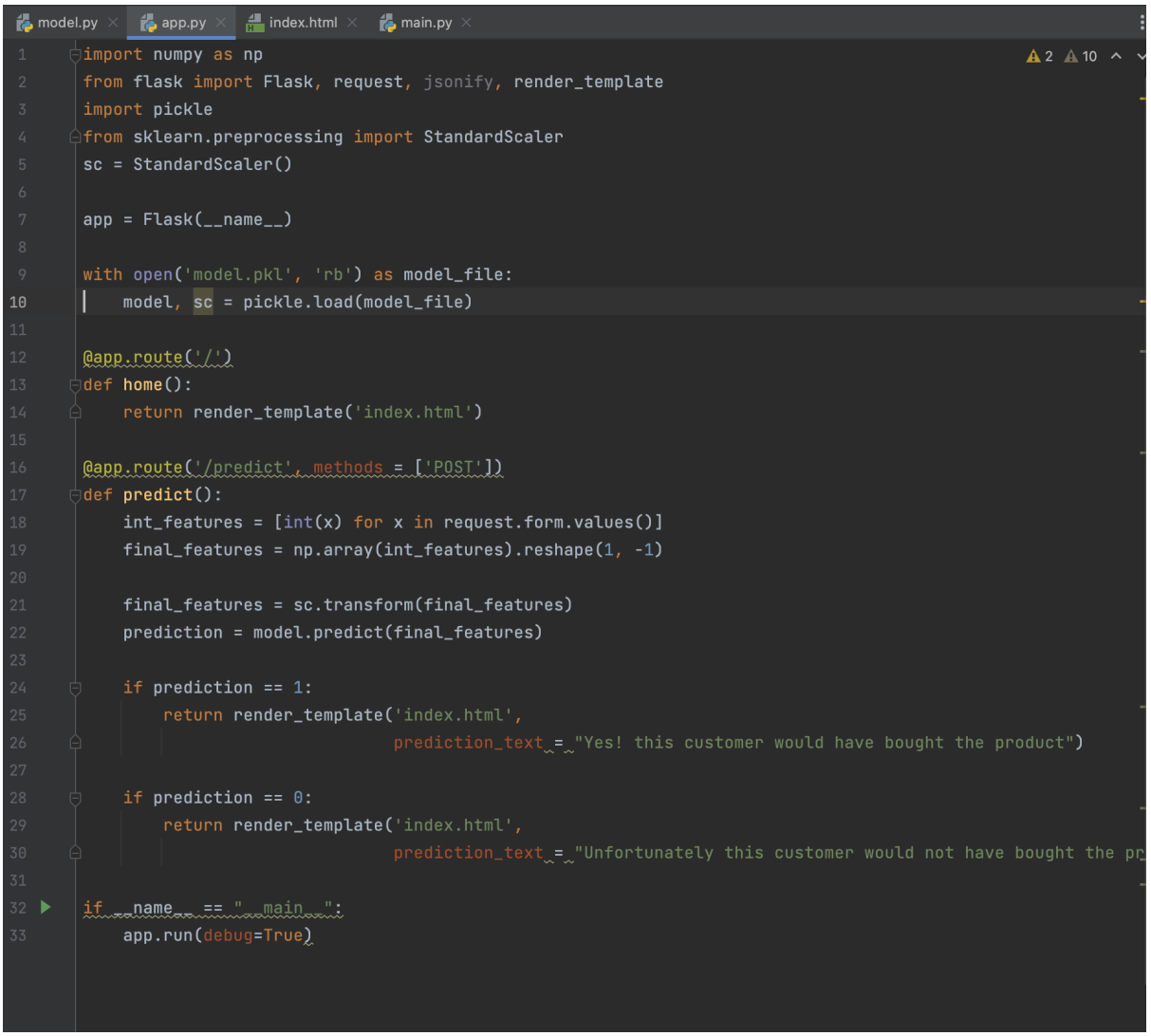
Saving the model to disk:

- The trained K-NN model and the StandardScaler object used for feature scaling are saved to disk using the pickle module.
- They are stored in a file named 'model.pkl' in binary write ('wb') mode.

Predicting test results:

- The code uses the trained K-NN classifier to make predictions on the test dataset (X_test) and stores the predictions in the y_pred variable.

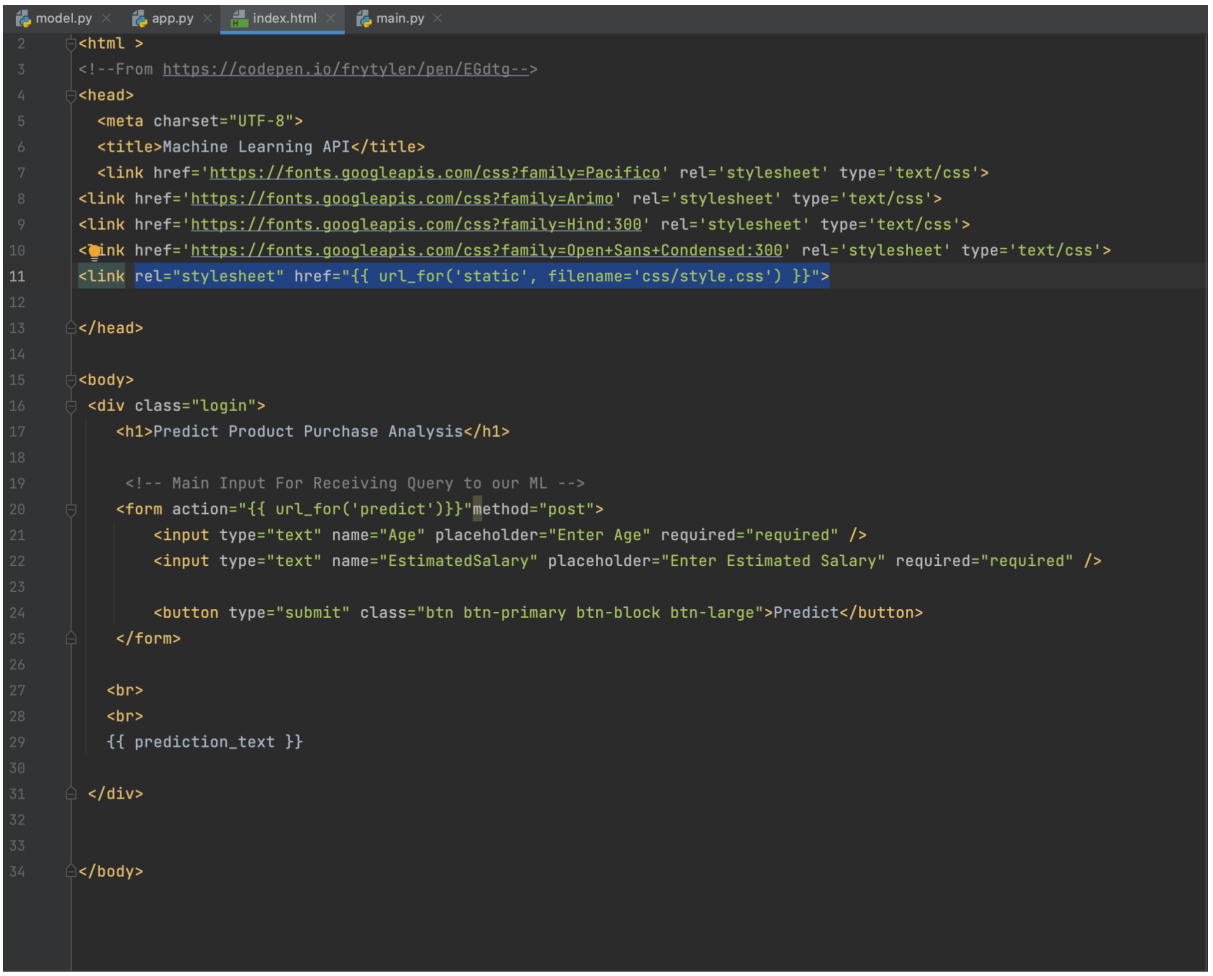
Deploying the model on Flask



This Python code is creating a simple web application for making predictions using a trained machine learning model.

- It imports the necessary libraries like Flask for creating the web app, numpy for numerical processing, pickle for loading the trained model, and StandardScaler for data preprocessing.
- It creates a Flask app instance and loads the pre-trained model and StandardScaler object from a pickle file called 'model.pkl'. This file contains the trained model object and fitted StandardScaler object we need to reuse.
- The '/' route renders a simple HTML template called 'index.html' to allow submitting input data.
- The '/predict' route handles making predictions. It takes in user input data, converts it to a numpy array, reshapes it to 2D for the model, applies the loaded StandardScaler to preprocess the data, and passes it to the model to make a prediction.
- Based on the prediction, it renders the same 'index.html' template but with an appropriate message for whether the customer would have bought the product or not.
- This allows creating a simple web interface where a user can submit data, it is preprocessed with the fitted StandardScaler, fed to the pretrained model, and a prediction is returned to the user.

HTML file for my Flask Webpage



This HTML code creates a simple web form to allow users to input data and make predictions from the Machine Learning API created in the Python Flask code.

- It defines a basic HTML layout with title, CSS styling and font imports.
- The with class "login" contains the main content.
- An <h1> tag displays the title "Predict Product Purchase Analysis".
- A HTML <form> is defined to send user input data to the "/predict" route of the Flask app on submit.
- There are two <input> fields to allow entering Age and EstimatedSalary values. These names match what the Flask code is expecting as input.
- A "Predict" button submits the form data.
- Below the form, the {{prediction_text}} variable will display the prediction result sent back from the Flask app.

So in summary, this simple HTML form allows a user to enter data, submit it to the API, and see the prediction result, creating an interactive web interface for the ML model. The Python Flask code handles the actual ML predictions based on the submitted data.

CSS File to beautify my webpage

```css
#import url(https://fonts.googleapis.com/css?family=Open+Sans);
.btn { display: inline-block; *display: inline; *zoom: 1; padding: 4px 10px 4px; margin-bottom: 0; font-size: 13px; line-height: 18px; color: #333333; text-align: center;text-shadow: 0 1px 1px rgba(255, 255, 255, 0.75); vertical-align: middle; background-color: #f5f5f5; background-image: -moz-linear-gradient(top, #ffffff, #e6e6e6); background-image: -ms-linear-gradient(top, #ffffff, #e6e6e6); background-image: -webkit-gradient(linear, 0 0, 0 100%, from(#ffffff), to(#e6e6e6)); background-image: -webkit-linear-gradient(top, #ffffff, #e6e6e6); background-image: -o-linear-gradient(top, #ffffff, #e6e6e6); background-image: linear-gradient(top, #ffffff, #e6e6e6); background-repeat: repeat-x; filter: progid:dximagetransform.microsoft.gradient(startColorstr=#ffffff, endColorstr=#e6e6e6, GradientType=0); border-color: #e6e6e6 #e6e6e6 #e6e6e6; border-color: rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.25); border: 1px solid #e6e6e6; -webkit-border-radius: 4px; -moz-border-radius: 4px; border-radius: 4px; -webkit-box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); -moz-box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); cursor: pointer; *margin-left: .3em; }
.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }
.btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px; }
.btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; -webkit-transition: background-position 0.1s linear; -moz-transition: background-position 0.1s linear; -ms-transition: background-position 0.1s linear; -o-transition: background-position 0.1s linear; transition: background-position 0.1s linear; }
.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25); color: #ffffff; }
.btn-primary.active { color: rgba(255, 255, 255, 0.75); }
.btn-primary { background-color: #4a77d4; background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4); background-image: -ms-linear-gradient(top, #6eb6de, #4a77d4); background-image: -webkit-gradient(linear, 0 0, 0 100%, from(#6eb6de), to(#4a77d4)); background-image: -webkit-linear-gradient(top, #6eb6de, #4a77d4); background-image: -o-linear-gradient(top, #6eb6de, #4a77d4); background-image: linear-gradient(top, #6eb6de, #4a77d4); background-repeat: repeat-x; filter: progid:dximagetransform.microsoft.gradient(startColorstr=#6eb6de, endColorstr=#4a77d4, GradientType=0);  border: 1px solid #3762bc; text-shadow: 1px 1px 1px rgba(0,0,0,0.4); box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.5); }
.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { filter: none; background-color: #4a77d4; }
.btn-block { width: 100%; display:block; }

* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; box-sizing:border-box; }

html { width: 100%; height:100%; overflow:hidden; }

body {
  width: 100%;
  height:100%;
  font-family: 'Open Sans', sans-serif;
  background: #092756;
  color: #fff;
  font-size: 18px;
  text-align:center;
  letter-spacing:1.2px;
  background: -moz-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%),-moz-linear-gradient(top,  rgba(57,173,219,.25) 0%, rgba(42,60,87,.4) 100%), -moz-linear-gradient(-45deg,  #670d10 0%, #092756 100%);
  background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -webkit-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -webkit-linear-gradient(-45deg,  #670d10 0%,#092756 100%);
  background: -o-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -o-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -o-linear-gradient(-45deg,  #670d10 0%,#092756 100%);
  background: -ms-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), -ms-linear-gradient(top,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), -ms-linear-gradient(-45deg,  #670d10 0%,#092756 100%);
  background: -webkit-radial-gradient(0% 100%, ellipse cover, rgba(104,128,138,.4) 10%,rgba(138,114,76,0) 40%), linear-gradient(to bottom,  rgba(57,173,219,.25) 0%,rgba(42,60,87,.4) 100%), linear-gradient(135deg,  #670d10 0%,#092756 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1 );

}
.login {
  position: absolute;
  top: 40%;
  left: 50%;
  margin: -150px 0 0 -150px;
  width:400px;
  height:400px;
}

.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing:1px; text-align:center; }

input {
  width: 100%;
  margin-bottom: 10px;
  background: rgba(0,0,0,0.3);
  border: none;
  outline: none;
  padding: 10px;
  font-size: 13px;
  color: #fff;
  text-shadow: 1px 1px 1px rgba(0,0,0,0.3);
  border: 1px solid rgba(0,0,0,0.3);
  border-radius: 4px;
  box-shadow: inset 0 -5px 45px rgba(100,100,100,0.2), 0 1px 1px rgba(255,255,255,0.2);
  -webkit-transition: box-shadow .5s ease;
  -moz-transition: box-shadow .5s ease;
  -o-transition: box-shadow .5s ease;
  -ms-transition: box-shadow .5s ease;
  transition: box-shadow .5s ease;
}
input:focus { box-shadow: inset 0 -5px 45px rgba(100,100,100,0.4), 0 1px 1px rgba(255,255,255,0.2); }
```

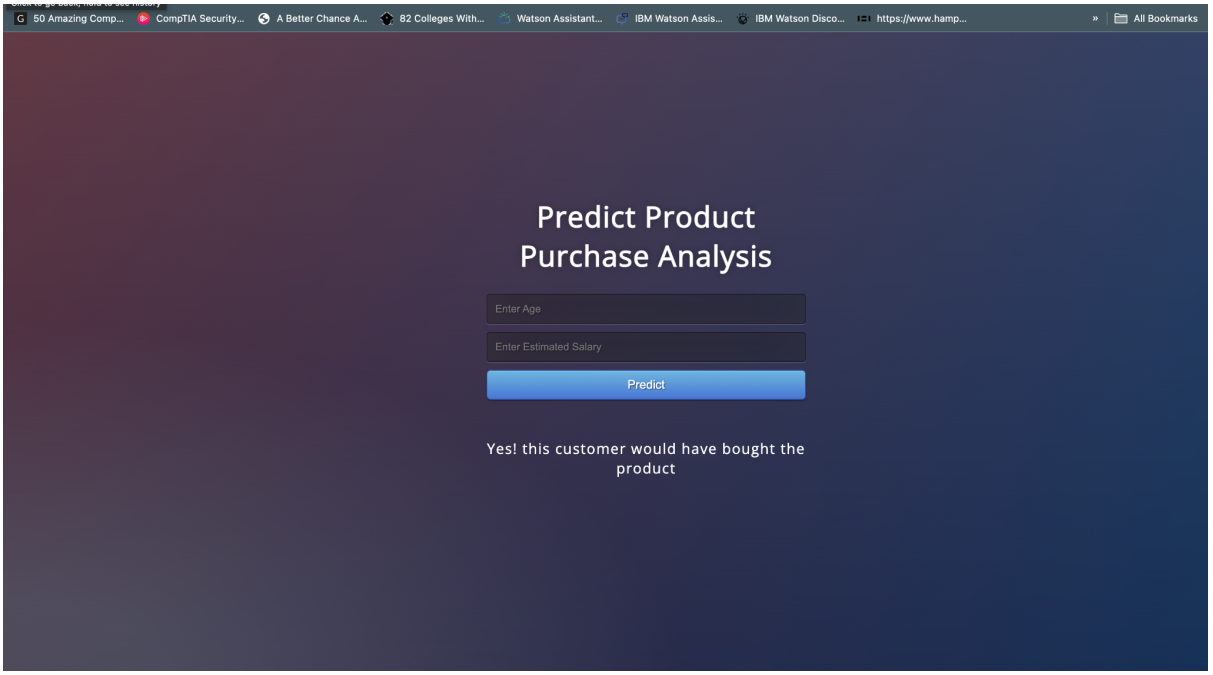
This CSS code creates the styling and layout for the web form to make predictions. The CSS imports the Open Sans font for styling the text.

- It defines styling for the submit button - colors, shadows, hover effects etc. This will style the "Predict" button.
- The .btn-primary class sets the main blue color and styling for the submit button.
- It sets box-sizing to border-box for all elements for easier styling.
- The html and body tags set up the overall page with dark background and centered text.
- The .login class centers the form container on the page using positioning and negative margins.
- The h1 tag displays the main heading "Predict Product Purchase Analysis".
- The input tags create the form fields for Age and EstimatedSalary. They have styles for width, fonts, shadows etc.

- The focus pseudo-class adds effect when focusing on an input.

In summary, the HTML constructs the basic form structure while the CSS stylizes and centers it to look nice on the page, creating the front end for users to input data and see predictions.

Outcome of UI

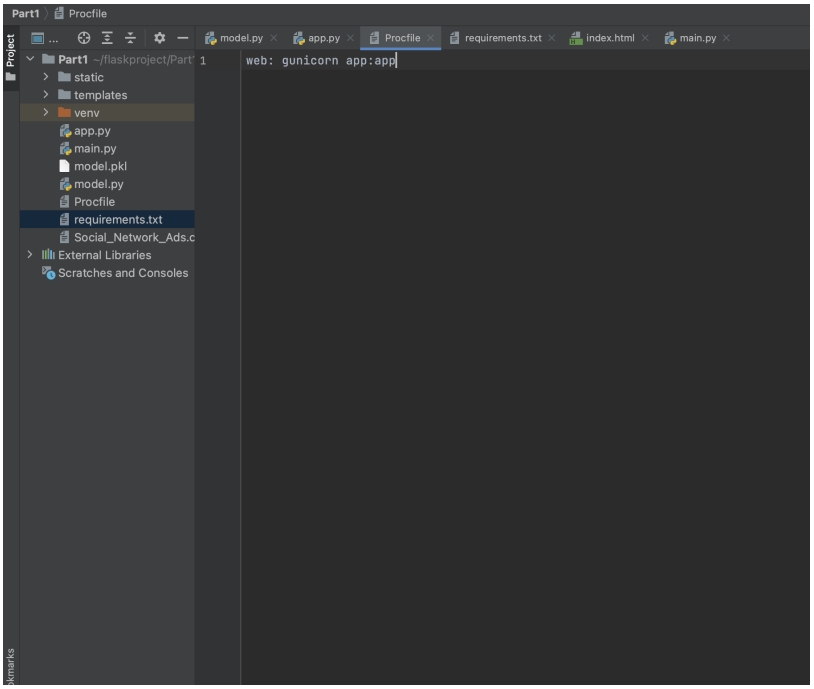


Model Deployment

After using Flask to create the UI of the model I had to deploy it using a cloud platform. In my case, I chose Heroku as my preferred cloud platform to deploy it.

1. I first had to create a Procfile.
    a. The Procfile defines the processes and commands to launch an app on Heroku
    b. It allows Heroku to know what processes like web server, workers etc need to be launched
    c. The "web" process runs the Flask app using something like gunicorn
    d. Additional processes can be defined like celery workers
    e. This allows the app to start and run properly on the Heroku dynos

So the Procfile is a key configuration file needed to deploy apps on Heroku.

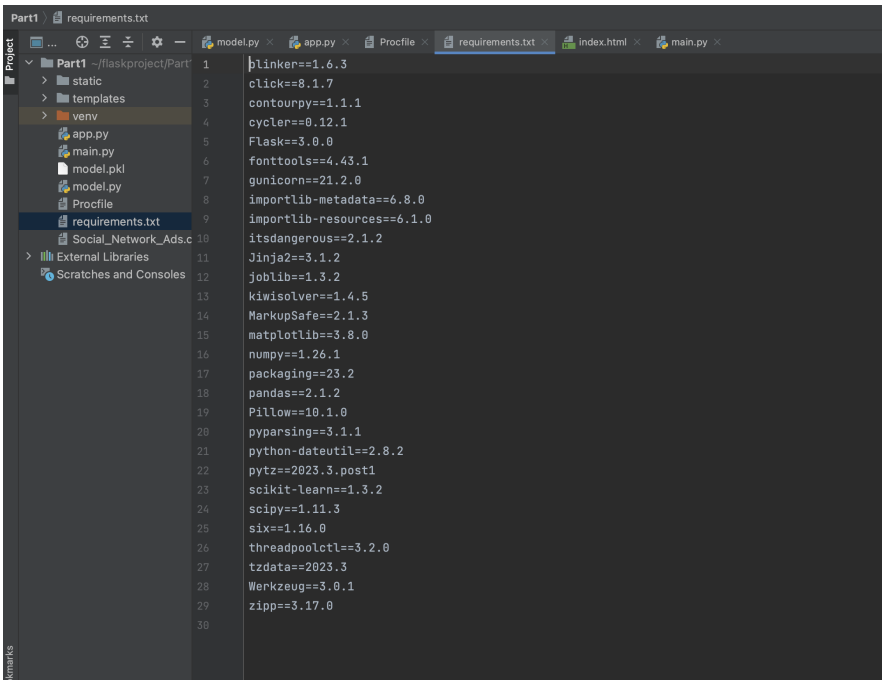


2. Then I had to create a 'requirements.txt' file
    a. It goes into the app root directory, similar to Procfile.
    b. It contains a list of package names, one on each line.
    c. It specifies all the app's dependencies that need to be installed.

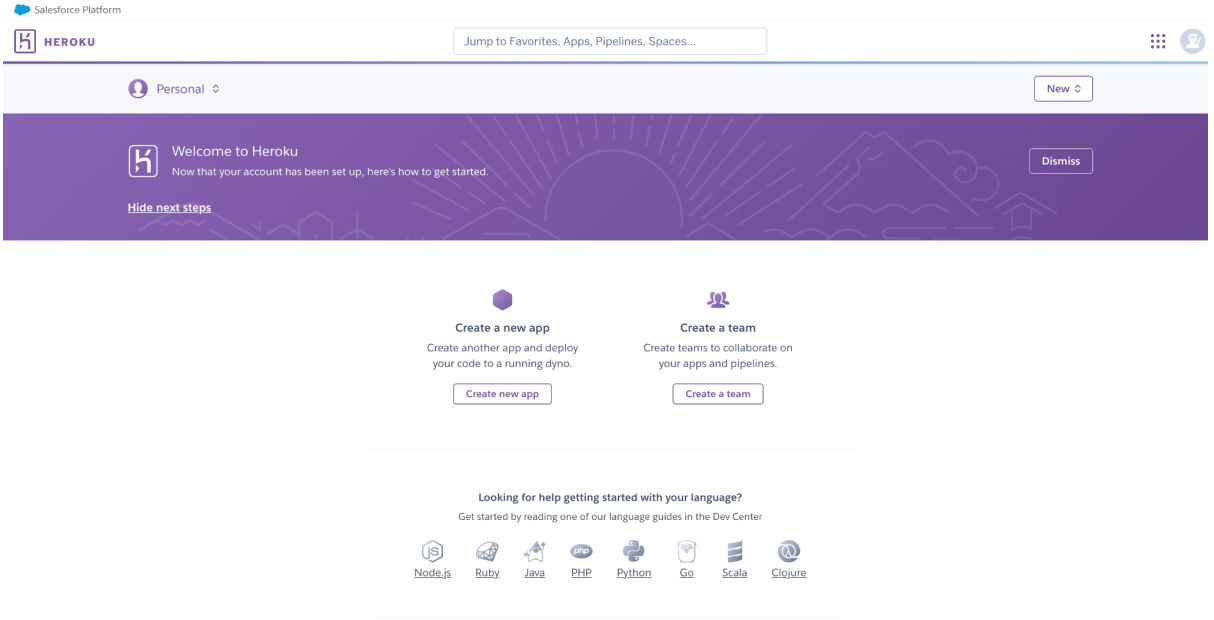
    When you deploy the app to Heroku, it will:

    A. Upload the code
    B. Install all packages listed in requirements.txt
    C. Use the Procfile to determine processes to run

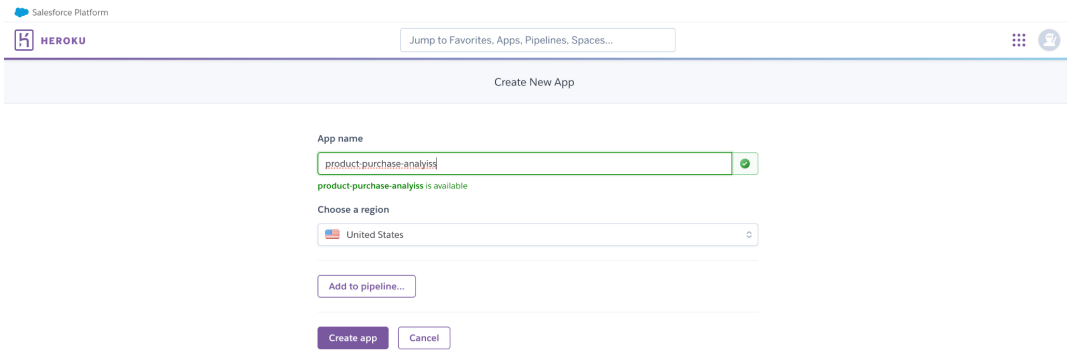
This ensures that when your app code runs on the Heroku dynos, all the required libraries are installed in the environment.



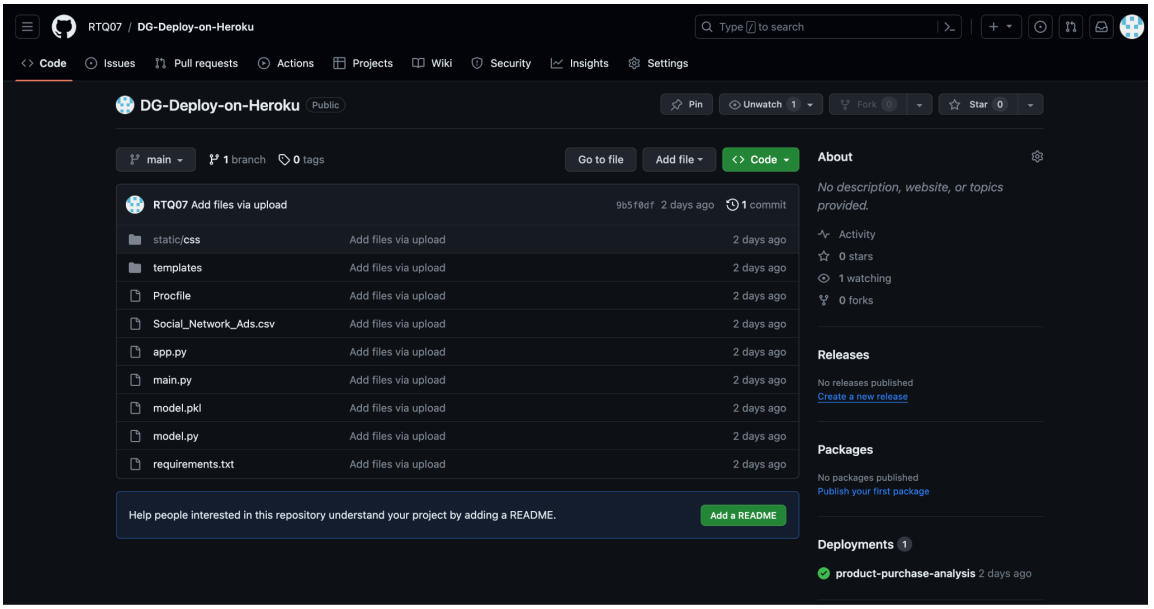
3. Then I went straight to deploying the model using Heroku.
    a. The first thing was to create the web app name
        i. We first click “create a new app”



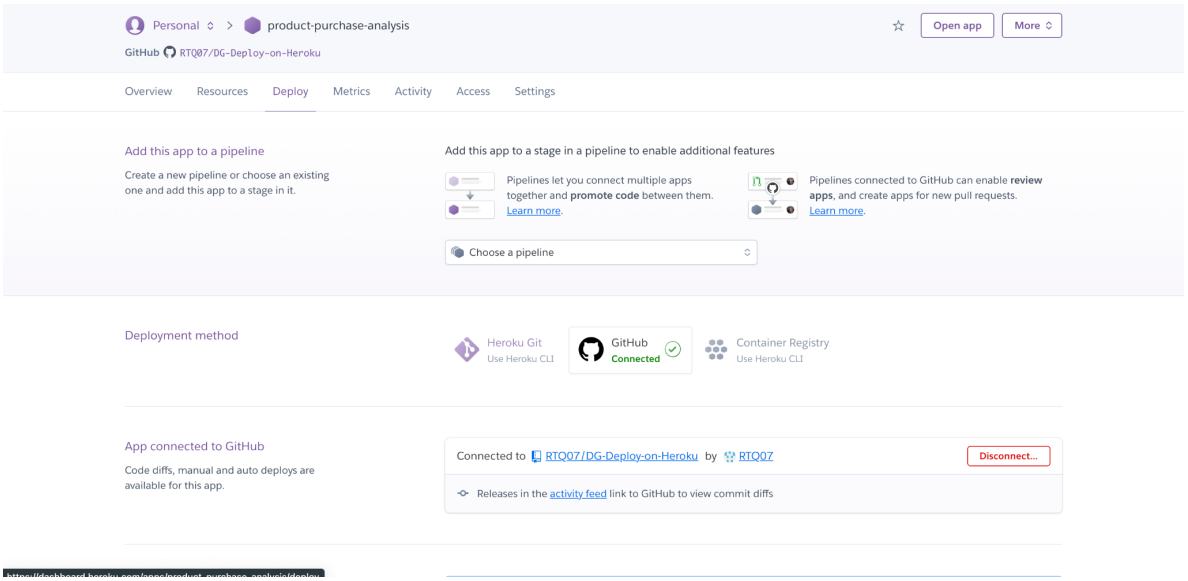
        ii. Then we name the new app and click “Create app”



    b. Then we connect Heroku to our repository on the Git hub
        i. Create repository



ii. Once the model is connected



iii. Click on “Deploy Branch”