Data Glacier

Data Science Intern at Data Glacier

Week4: Deployment on Flask

Customer Annual Income Prediction Using Logistic Regression

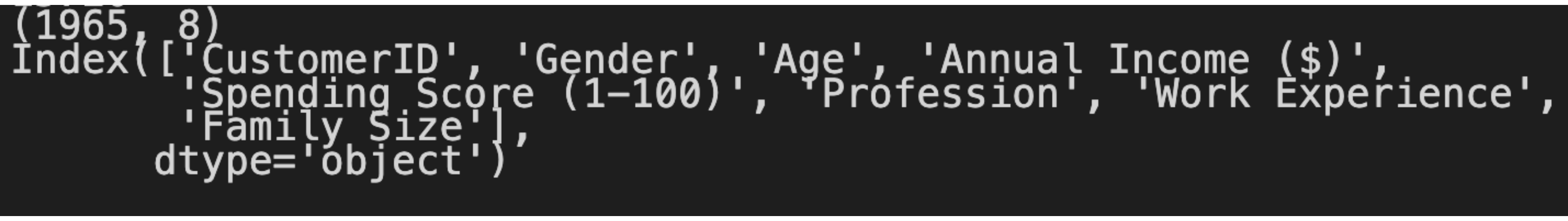
Name: Nilsu Bozan
Batch Code: LISUM20
Date: 1 May 2023
Submitted to: Data Glacier

Introduction

In this project, we've created a machine learning model that can predict a customer's annual income category by analyzing various demographic and behavioral attributes. We chose the Logistic Regression algorithm to build our predictive model, as it is a widely used method for tackling classification problems.

Dataset

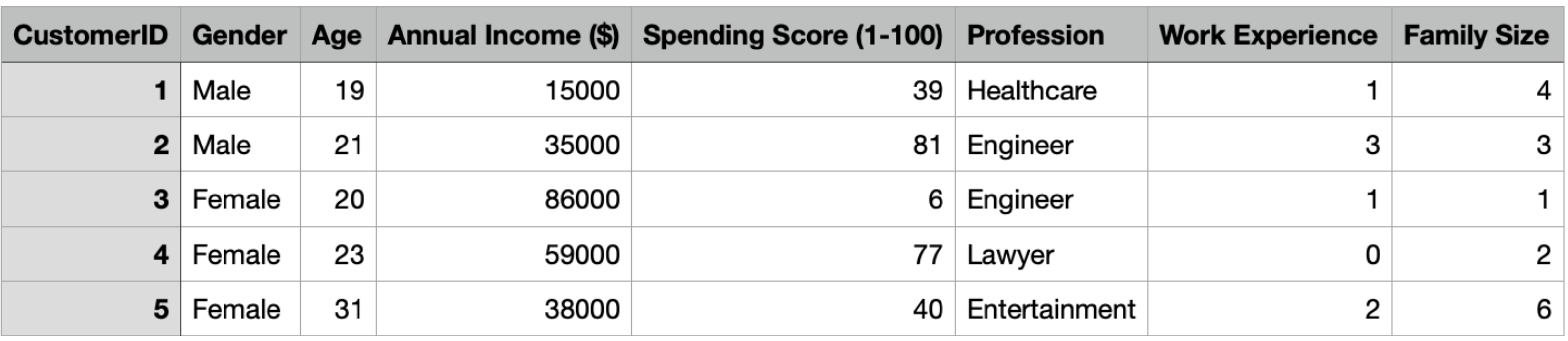
We used Customer.csv dataset which consist of 8 columns and 1965 rows. The dataset we used contains information about customers, including their age, gender, profession, spending score, work experience, and family size.



```
(1965, 8)
Index(['CustomerID', 'Gender', 'Age', 'Annual Income ($)',
       'Spending Score (1-100)', 'Profession', 'Work Experience',
       'Family Size'],
      dtype='object')
```

Attributes

This table shows the first 5 lines of our dataset.



CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
1	Male	19	15000	39	Healthcare	1	4
2	Male	21	35000	81	Engineer	3	3
3	Female	20	86000	6	Engineer	1	1
4	Female	23	59000	77	Lawyer	0	2
5	Female	31	38000	40	Entertainment	2	6

Data Preparation

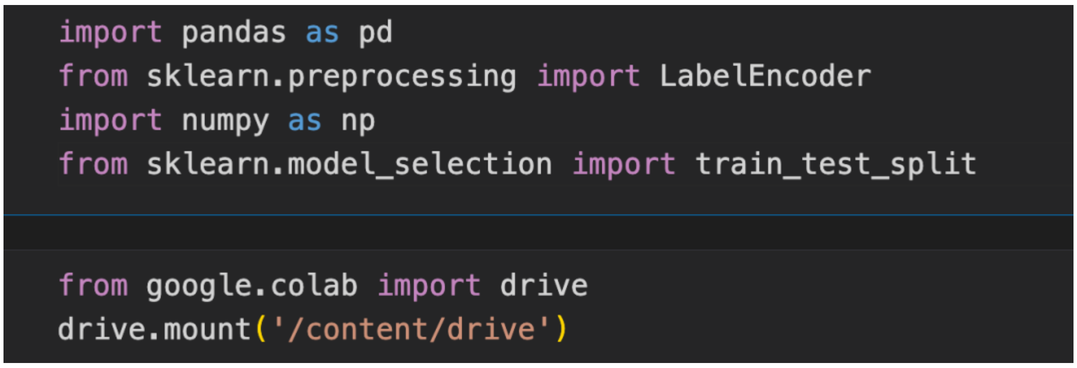
To prepare the data for our model, we encoded categorical variables, like gender and profession, and divided the annual income variable into five distinct categories: 'Very Low', 'Low', 'Moderate', 'High', and 'Very High'.

In addition to categorizing annual income, we've also grouped customers into different age ranges by creating bins for the age column. This approach enables us to capture the diverse income patterns that emerge across various age groups, ultimately leading to a more precise understanding of a customer's financial situation.

Building the Model

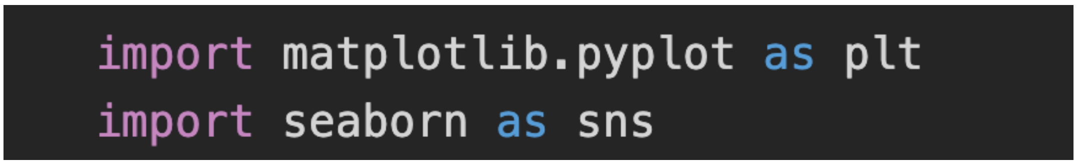
1)Importing Libraries

We imported pandas for data manipulation, numpy for data statistical analysis, matplot.lib.pyplot ans seaborn to visualize the data.
We also imported several modules from Scikit-learn library.

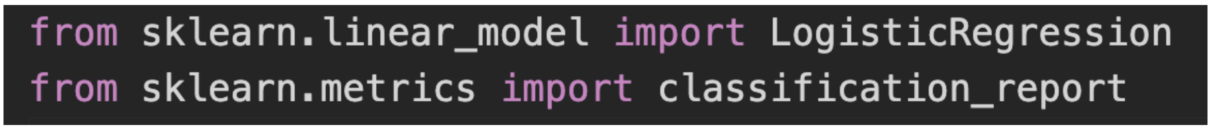


```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.model_selection import train_test_split
```

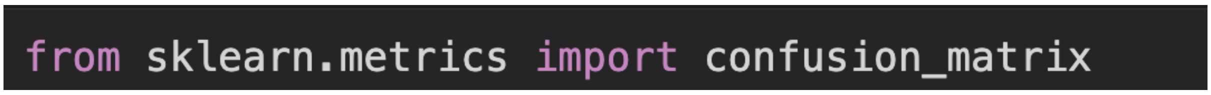
```
from google.colab import drive
drive.mount('/content/drive')
```



```
import matplotlib.pyplot as plt
import seaborn as sns
```

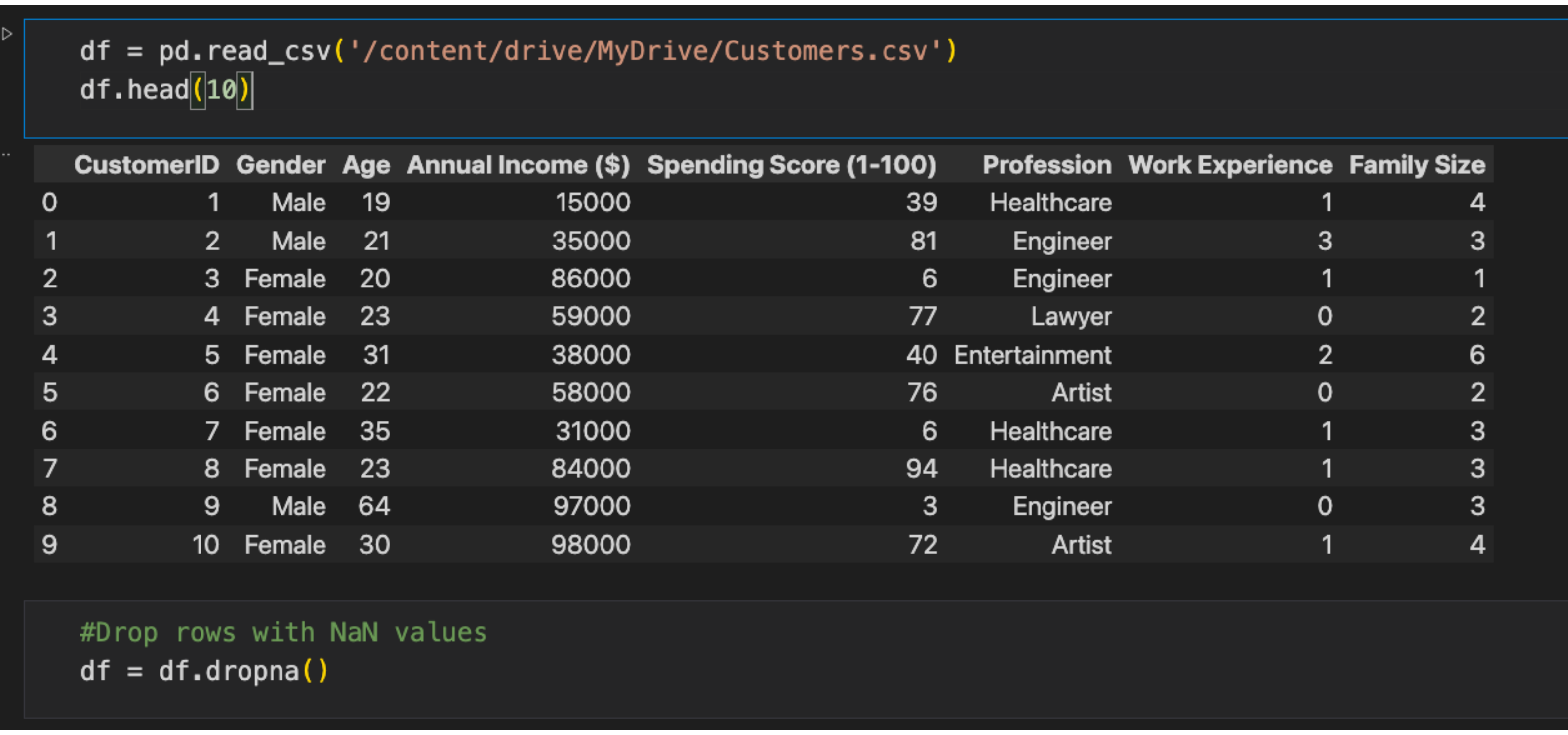


```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```



```
from sklearn.metrics import confusion_matrix
```

2)Reading and Preparing the Dataset

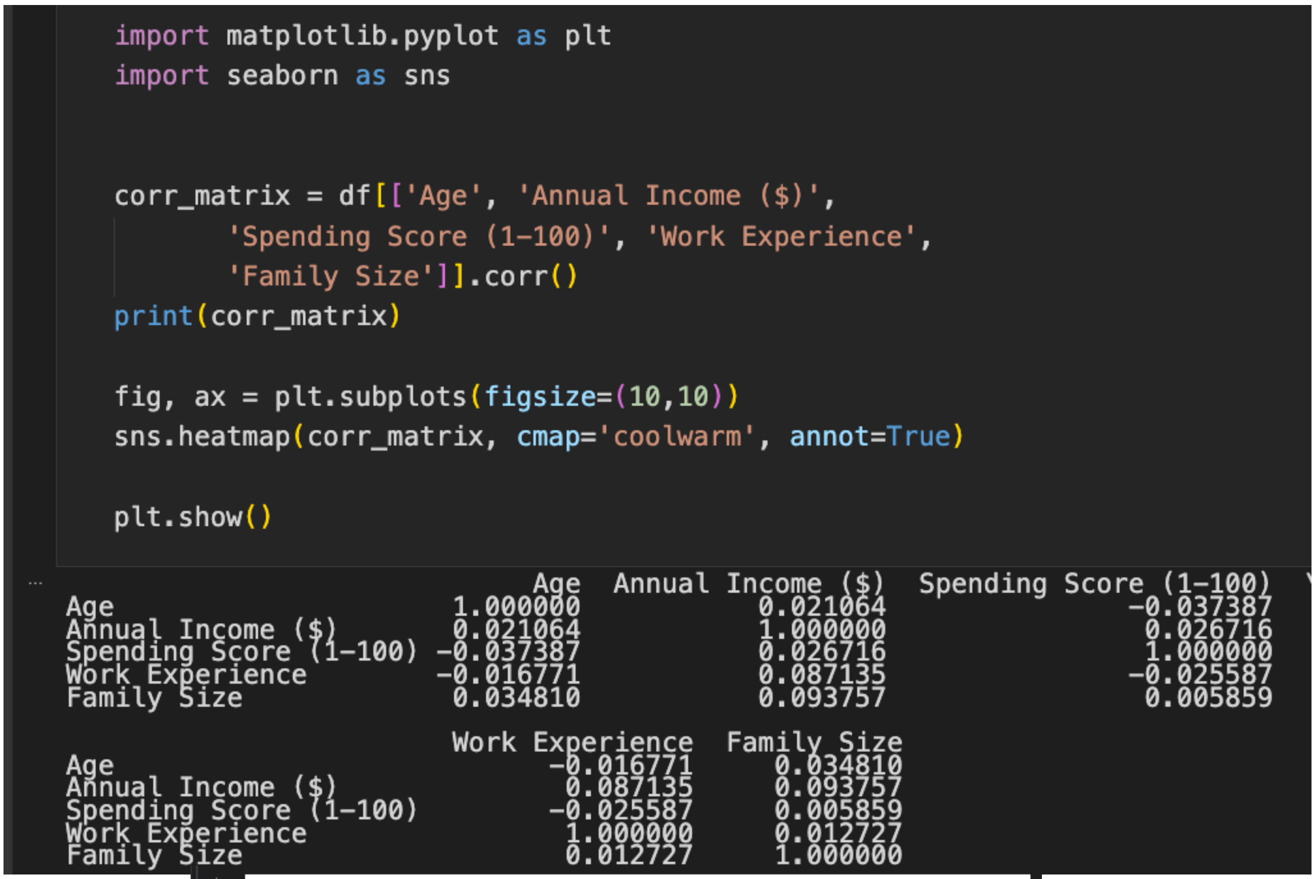


```
df = pd.read_csv('/content/drive/MyDrive/Customers.csv')
df.head(10)
```

	CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
0	1	Male	19	15000	39	Healthcare	1	4
1	2	Male	21	35000	81	Engineer	3	3
2	3	Female	20	86000	6	Engineer	1	1
3	4	Female	23	59000	77	Lawyer	0	2
4	5	Female	31	38000	40	Entertainment	2	6
5	6	Female	22	58000	76	Artist	0	2
6	7	Female	35	31000	6	Healthcare	1	3
7	8	Female	23	84000	94	Healthcare	1	3
8	9	Male	64	97000	3	Engineer	0	3
9	10	Female	30	98000	72	Artist	1	4

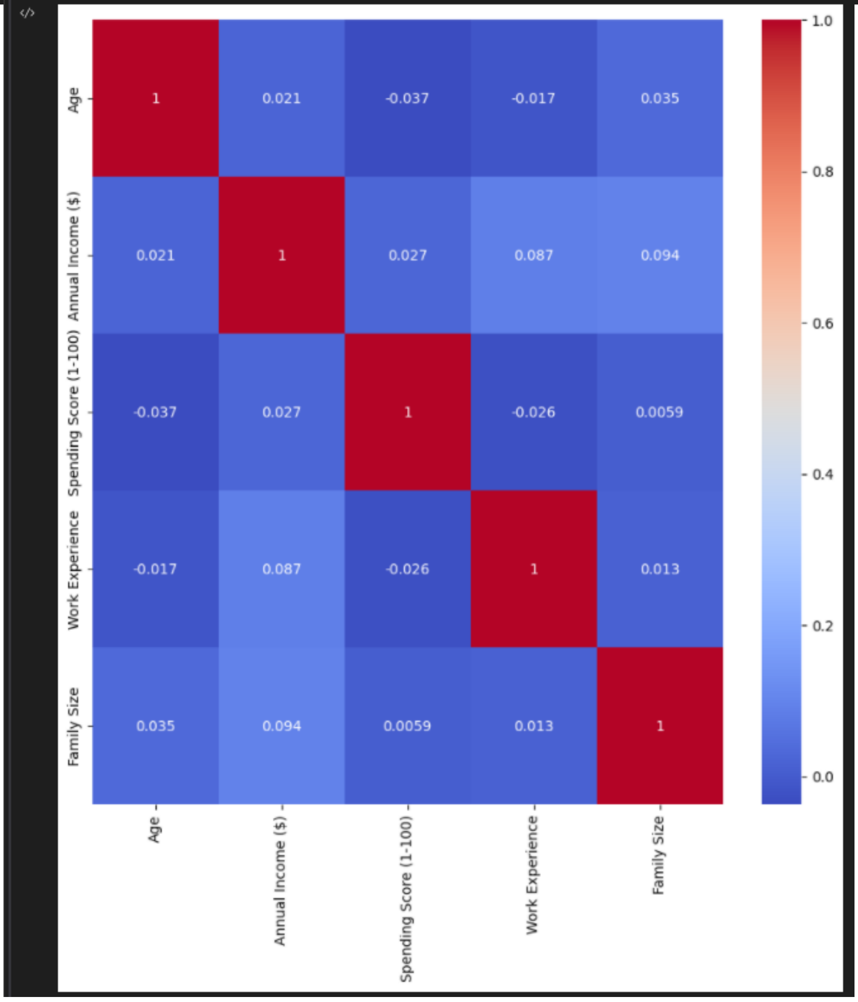
```
#Drop rows with NaN values
df = df.dropna()
```

We used correlation matrix to observe the relationship between attributes. It proved us that family size and work experience is positively related.

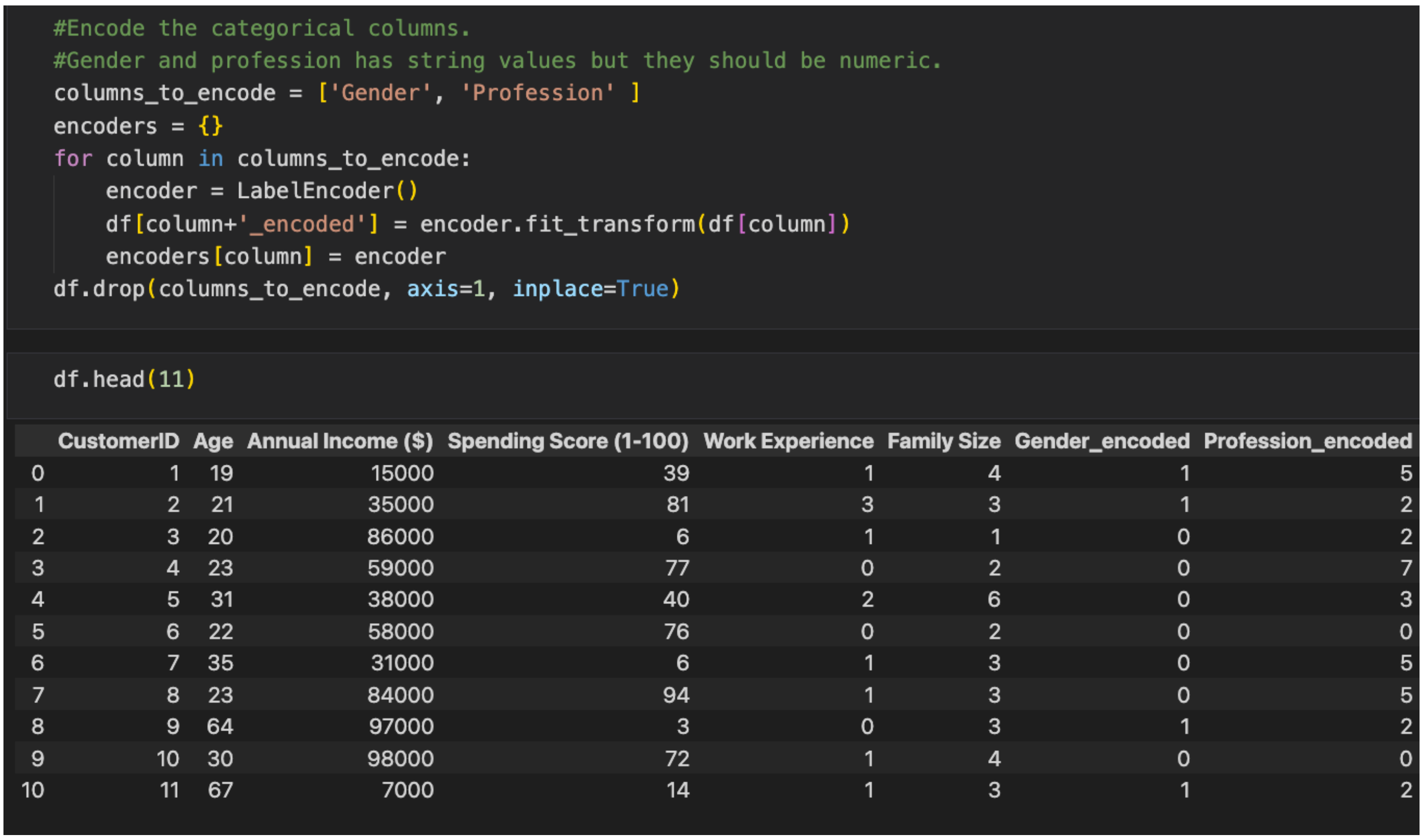


```python
import matplotlib.pyplot as plt
import seaborn as sns


corr_matrix = df[['Age', 'Annual Income ($)',
        'Spending Score (1-100)', 'Work Experience',
        'Family Size']].corr()
print(corr_matrix)

fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)

plt.show()
```

```
                             Age  Annual Income ($)  Spending Score (1-100)
Age                     1.000000           0.021064               -0.037387
Annual Income ($)       0.021064           1.000000                0.026716
Spending Score (1-100) -0.037387           0.026716                1.000000
Work Experience        -0.016771           0.087135               -0.025587
Family Size             0.034810           0.093757                0.005859

                        Work Experience  Family Size
Age                           -0.016771     0.034810
Annual Income ($)              0.087135     0.093757
Spending Score (1-100)        -0.025587     0.005859
Work Experience                1.000000     0.012727
Family Size                    0.012727     1.000000
```



Since many of the Machine Learning algorithms require numerical input, We encoded categorical columns ‘Gender’ and ‘Profession’.



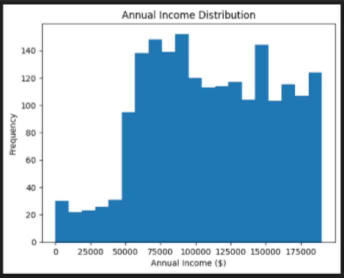
```python
#Encode the categorical columns.
#Gender and profession has string values but they should be numeric.
columns_to_encode = ['Gender', 'Profession' ]
encoders = {}
for column in columns_to_encode:
    encoder = LabelEncoder()
    df[column+'_encoded'] = encoder.fit_transform(df[column])
    encoders[column] = encoder
df.drop(columns_to_encode, axis=1, inplace=True)
```

```python
df.head(11)
```

	CustomerID	Age	Annual Income ($)	Spending Score (1-100)	Work Experience	Family Size	Gender_encoded	Profession_encoded
0	1	19	15000	39	1	4	1	5
1	2	21	35000	81	3	3	1	2
2	3	20	86000	6	1	1	0	2
3	4	23	59000	77	0	2	0	7
4	5	31	38000	40	2	6	0	3
5	6	22	58000	76	0	2	0	0
6	7	35	31000	6	1	3	0	5
7	8	23	84000	94	1	3	0	5
8	9	64	97000	3	0	3	1	2
9	10	30	98000	72	1	4	0	0
10	11	67	7000	14	1	3	1	2

We observed that ‘Male’ is represented with 1 and ‘0’ is represented with 0 after encoded. Additionally, we observed different professions are represented with different numeric values.

```python
plt.hist(df['Annual Income ($)'], bins=20)


plt.xlabel('Annual Income ($)')
plt.ylabel('Frequency')
plt.title('Annual Income Distribution')

#I used qcut to bin variable into 5 equal frequency bins
df['income_bins'] = pd.qcut(df['Annual Income ($)'], q=5, duplicates='drop')
labels = ['Very Low', 'Low', 'Moderate', 'High', 'Very High']
df['income_bin_labels'] = pd.qcut(df['Annual Income ($)'], q=5, duplicates='drop', labels=labels)
```



```python
#Bin edges for age column
bins_age = [0, 25, 30, 35, 40, 50, max(df['Age'])]
labels_age = ['<25', '25-30', '31-35', '36-40', '41-50', '>50']

#Created a new column called age group
df['Age Group'] = pd.cut(df['Age'], bins=bins_age, labels=labels_age)
```

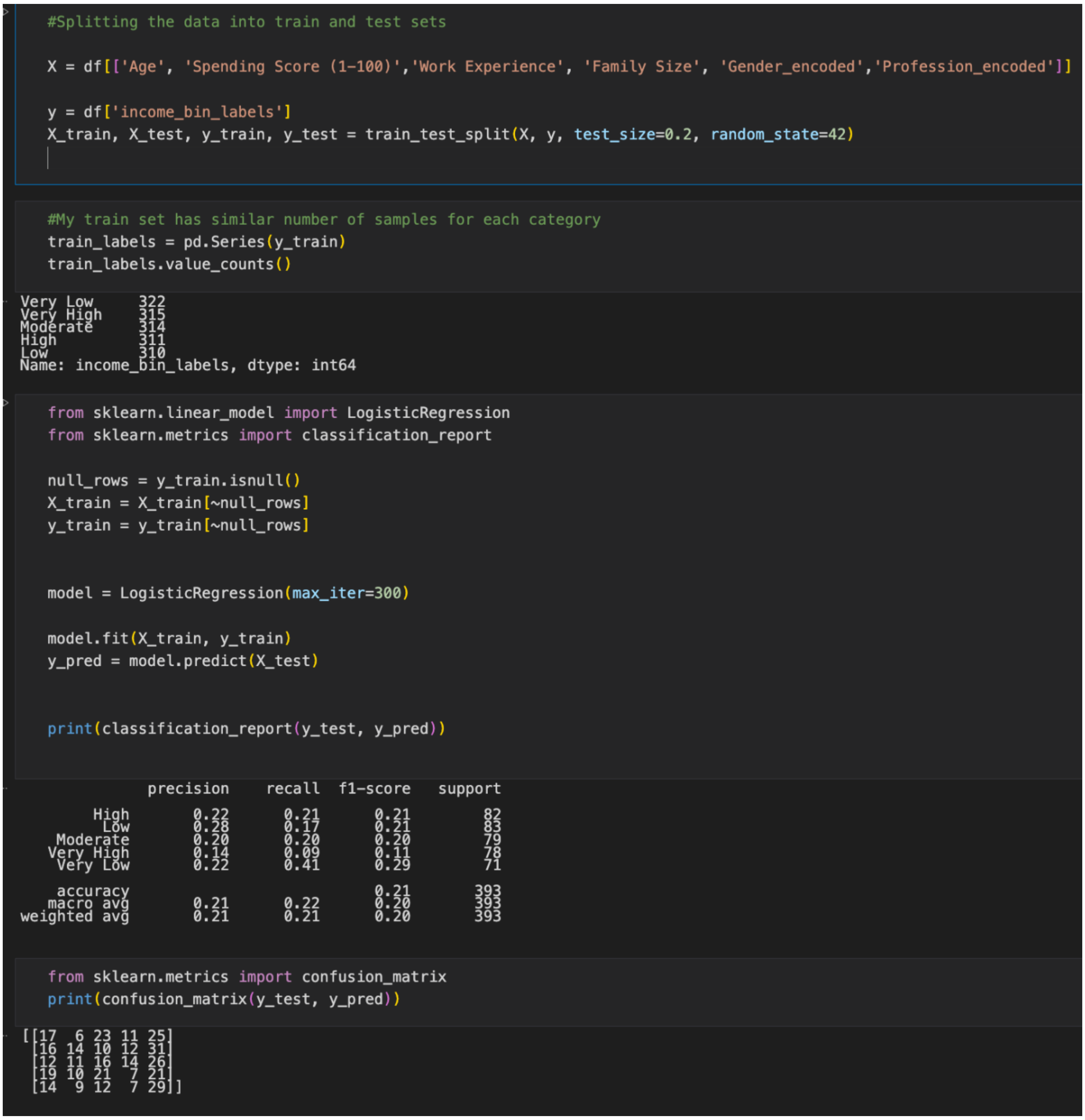
```python
df.head(10)
```

	CustomerID	Age	Annual Income ($)	Spending Score (1-100)	Work Experience	Family Size	Gender_encoded	Profession_encoded	income_bins	income_bin_labels	Age Group
0	1	19	15000	39	1	4	1	5	(-0.001, 67983.2]	Very Low	<25
1	2	21	35000	81	3	3	1	2	(-0.001, 67983.2]	Very Low	<25
2	3	20	86000	6	1	1	0	2	(67983.2, 93848.8]	Low	<25
3	4	23	59000	77	0	2	0	7	(-0.001, 67983.2]	Very Low	<25
4	5	31	38000	40	2	6	0	3	(-0.001, 67983.2]	Very Low	31-35
5	6	22	58000	76	0	2	0	0	(-0.001, 67983.2]	Very Low	<25
6	7	35	31000	6	1	3	0	5	(-0.001, 67983.2]	Very Low	31-35
7	8	23	84000	94	1	3	0	5	(67983.2, 93848.8]	Low	<25
8	9	64	97000	3	0	3	1	2	(93848.8, 125666.8]	Moderate	>50
9	10	30	98000	72	1	4	0	0	(93848.8, 125666.8]	Moderate	25-30

We created histogram to create income bins and decide their labels. After that, we also created bin edges for 'Age' column. The reason why we perfom these steps were improving the accuracy and performance of our model.

3)Model Building

4)Saving and Testing our model



```
#Splitting the data into train and test sets

X = df[['Age', 'Spending Score (1-100)','Work Experience', 'Family Size', 'Gender_encoded','Profession_encoded']]

y = df['income_bin_labels']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#My train set has similar number of samples for each category
train_labels = pd.Series(y_train)
train_labels.value_counts()
```

```
Very Low     322
Very High    315
Moderate     314
High         311
Low          310
Name: income_bin_labels, dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

null_rows = y_train.isnull()
X_train = X_train[~null_rows]
y_train = y_train[~null_rows]


model = LogisticRegression(max_iter=300)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)


print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

        High       0.22      0.21      0.21        82
         Low       0.28      0.17      0.21        83
    Moderate       0.20      0.20      0.20        79
   Very High       0.14      0.09      0.11        78
    Very Low       0.22      0.41      0.29        71

    accuracy                           0.21       393
   macro avg       0.21      0.22      0.20       393
weighted avg       0.21      0.21      0.20       393
```

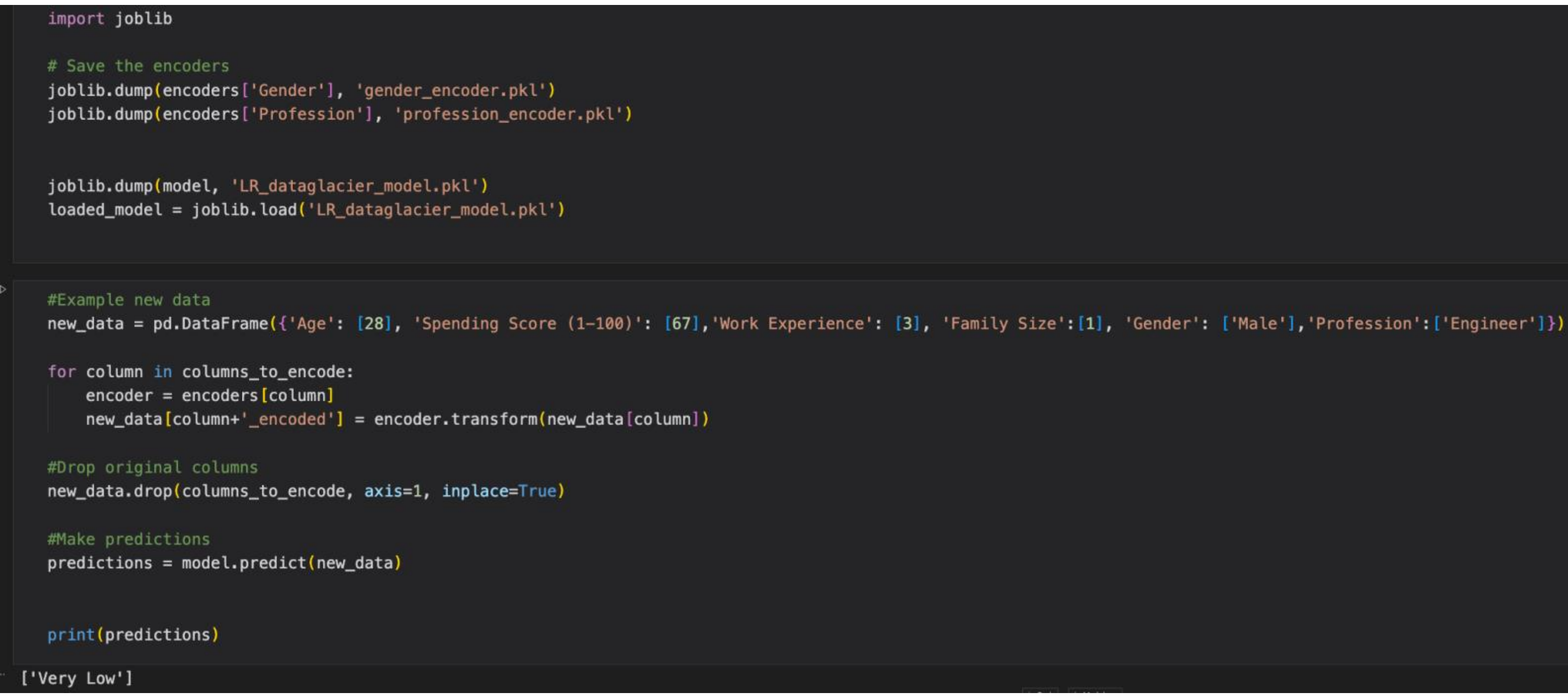
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[17  6 23 11 25]
 [16 14 10 12 31]
 [12 11 16 14 26]
 [19 10 21  7 21]
 [14  9 12  7 29]]
```

We splitted the dataset into training and testing sets. 0.2 of the data is used for testing and 0.8 of the data is used for training our model.

Then, we illustrate that for each category of Income we have similar number of samples.

After we made sure that we have no null rows in our train sets, we created our Logistic Regression Model. We printed classification report and confusion matrix for our model to see it’s performance. The results were not as good as we were expecting. However, we still believe that we can improve the performance of our model in the future.



```
import joblib

# Save the encoders
joblib.dump(encoders['Gender'], 'gender_encoder.pkl')
joblib.dump(encoders['Profession'], 'profession_encoder.pkl')


joblib.dump(model, 'LR_dataglacier_model.pkl')
loaded_model = joblib.load('LR_dataglacier_model.pkl')
```

```
#Example new data
new_data = pd.DataFrame({'Age': [28], 'Spending Score (1-100)': [67],'Work Experience': [3], 'Family Size':[1], 'Gender': ['Male'],'Profession':['Engineer']})

for column in columns_to_encode:
    encoder = encoders[column]
    new_data[column+'_encoded'] = encoder.transform(new_data[column])

#Drop original columns
new_data.drop(columns_to_encode, axis=1, inplace=True)

#Make predictions
predictions = model.predict(new_data)


print(predictions)
```

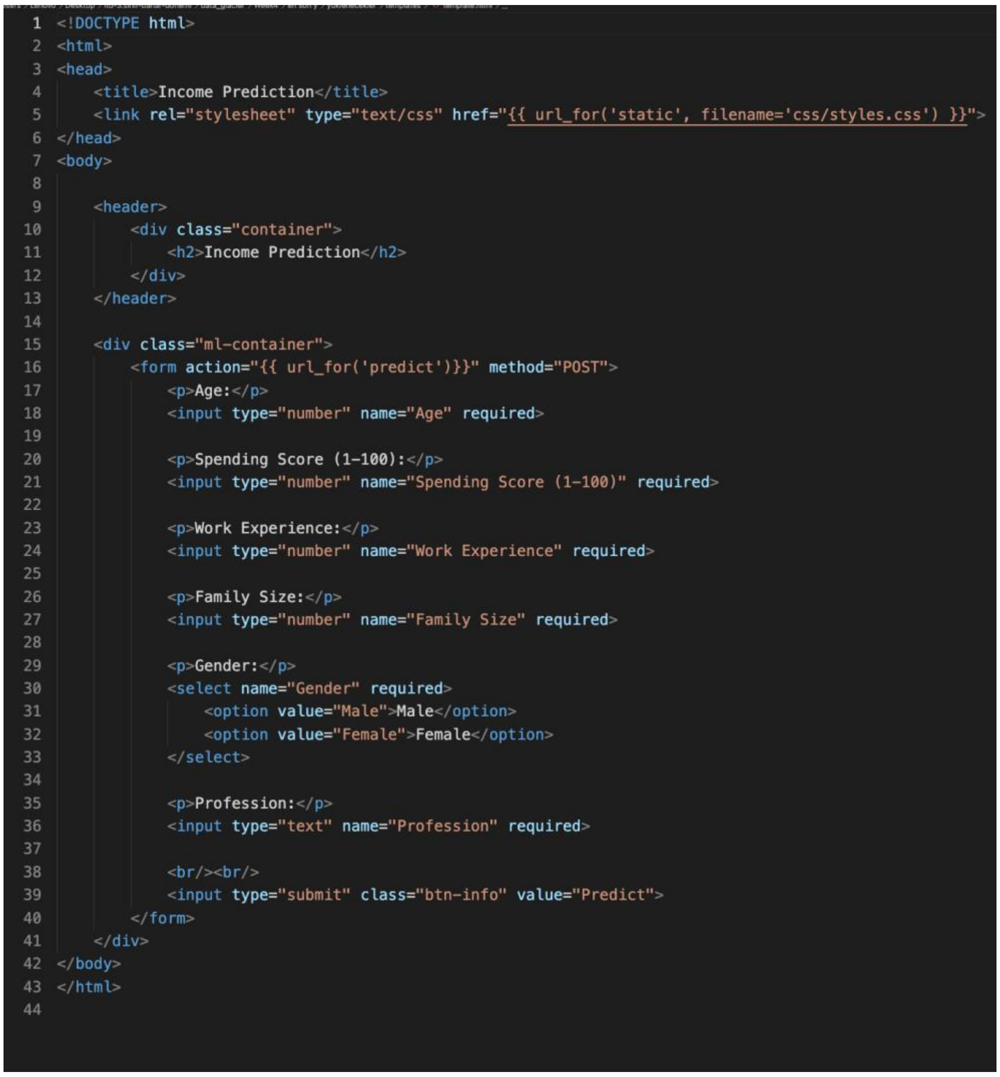
```
['Very Low']
```

Finally, we saved our model to be used in the future. And we tried our model with an example. Our model predicted that, according to the new_data(example data) this customer has Very Low annual income.

Deploying the Model on Flask

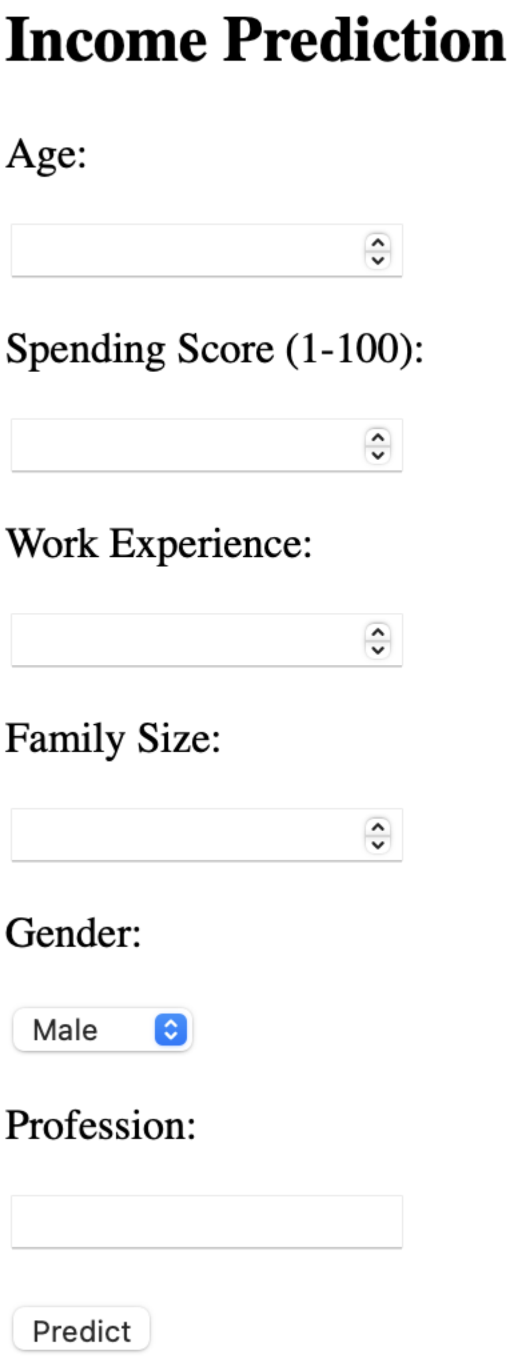
After we were done with the implementation of our model, we started to deploy our model on Flask.

Our web application designed to predict a customer's annual income category based on their demographic and behavioral attributes. By inputting relevant customer data such as age, gender, profession, spending score, work experience, and family size, users can quickly receive predictions about the customer's annual income category. These categories include 'Very Low', 'Low', 'Moderate', 'High', and 'Very High'.



```html
<!DOCTYPE html>
<html>
<head>
    <title>Income Prediction</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
<body>

    <header>
        <div class="container">
            <h2>Income Prediction</h2>
        </div>
    </header>

    <div class="ml-container">
        <form action="{{ url_for('predict')}}" method="POST">
            <p>Age:</p>
            <input type="number" name="Age" required>

            <p>Spending Score (1-100):</p>
            <input type="number" name="Spending Score (1-100)" required>

            <p>Work Experience:</p>
            <input type="number" name="Work Experience" required>

            <p>Family Size:</p>
            <input type="number" name="Family Size" required>

            <p>Gender:</p>
            <select name="Gender" required>
                <option value="Male">Male</option>
                <option value="Female">Female</option>
            </select>

            <p>Profession:</p>
            <input type="text" name="Profession" required>

            <br/><br/>
            <input type="submit" class="btn-info" value="Predict">
        </form>
    </div>
</body>
</html>
```

Initially we created a HTML template for a web page that serves as the user interface for the income prediction web application.


Income Prediction

Age:

Spending Score (1-100):

Work Experience:

Family Size:

Gender:

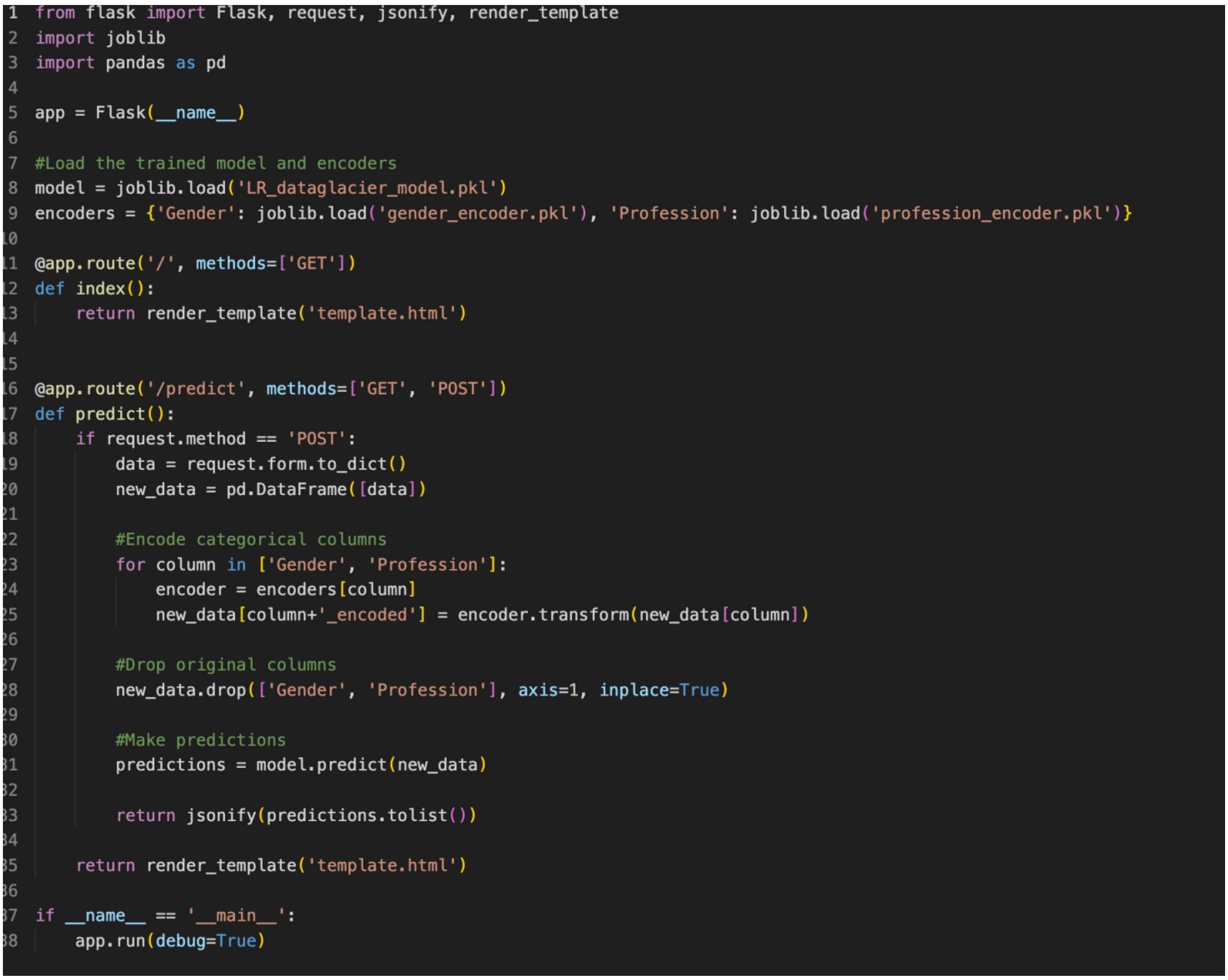
Male

Profession:

Predict

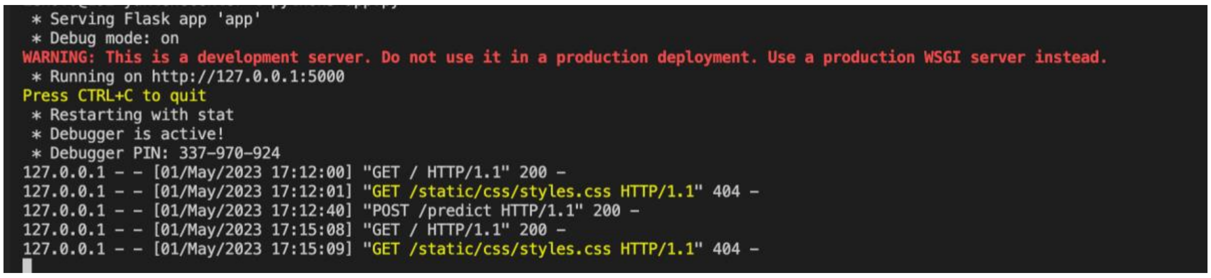
Flask looks for HTML template under templates folder. It renders a text form where a user can enter age, spending score, work experience, family size, gender, profession. Then it provides us with web page looking like this.

In our application, we initialized a Flask instance with the argument 'name'. It enables Flask to find the HTML template folder(templates) in the same directory. We used route decorators to specify URLs for the index function, which renders the template.html file, and the predict function, which handles GET and POST requests.
The predict function preprocesses user input data by encoding categorical variables and uses our Logistic Regression model to predict the annual income category. We employ the POST method to transport form data to the server and activate Flask's debugger with debug=True in the app.run method.

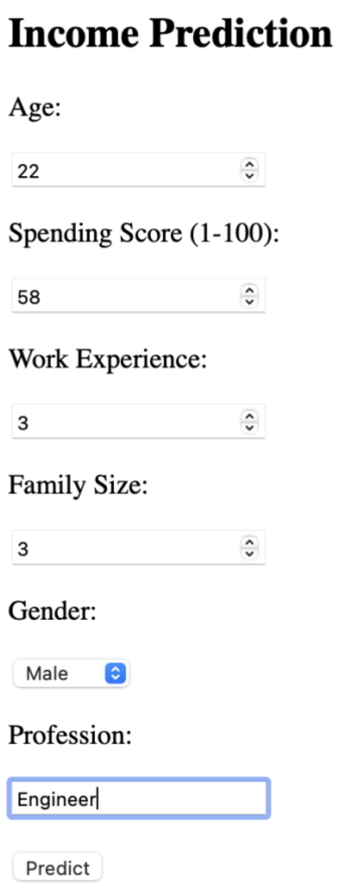


```python
from flask import Flask, request, jsonify, render_template
import joblib
import pandas as pd

app = Flask(__name__)

#Load the trained model and encoders
model = joblib.load('LR_dataglacier_model.pkl')
encoders = {'Gender': joblib.load('gender_encoder.pkl'), 'Profession': joblib.load('profession_encoder.pkl')}

@app.route('/', methods=['GET'])
def index():
    return render_template('template.html')


@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        data = request.form.to_dict()
        new_data = pd.DataFrame([data])

        #Encode categorical columns
        for column in ['Gender', 'Profession']:
            encoder = encoders[column]
            new_data[column+'_encoded'] = encoder.transform(new_data[column])

        #Drop original columns
        new_data.drop(['Gender', 'Profession'], axis=1, inplace=True)

        #Make predictions
        predictions = model.predict(new_data)

        return jsonify(predictions.tolist())

    return render_template('template.html')

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

We ensure the application runs on the server only when the script is executed by the Python interpreter using the if statement with name == 'main'. This setup creates an interface for predicting a customer's annual income category based on their demographic and behavioral features.



```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 337-970-924
127.0.0.1 - - [01/May/2023 17:12:00] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/May/2023 17:12:01] "GET /static/css/styles.css HTTP/1.1" 404 -
127.0.0.1 - - [01/May/2023 17:12:40] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [01/May/2023 17:15:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/May/2023 17:15:09] "GET /static/css/styles.css HTTP/1.1" 404 -
```



Income Prediction

Age:

22

Spending Score (1-100):

58

Work Experience:

3

Family Size:

3

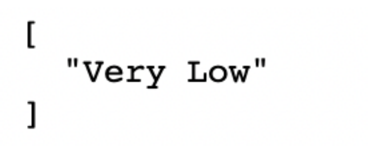
Gender:

Male

Profession:

Engineer

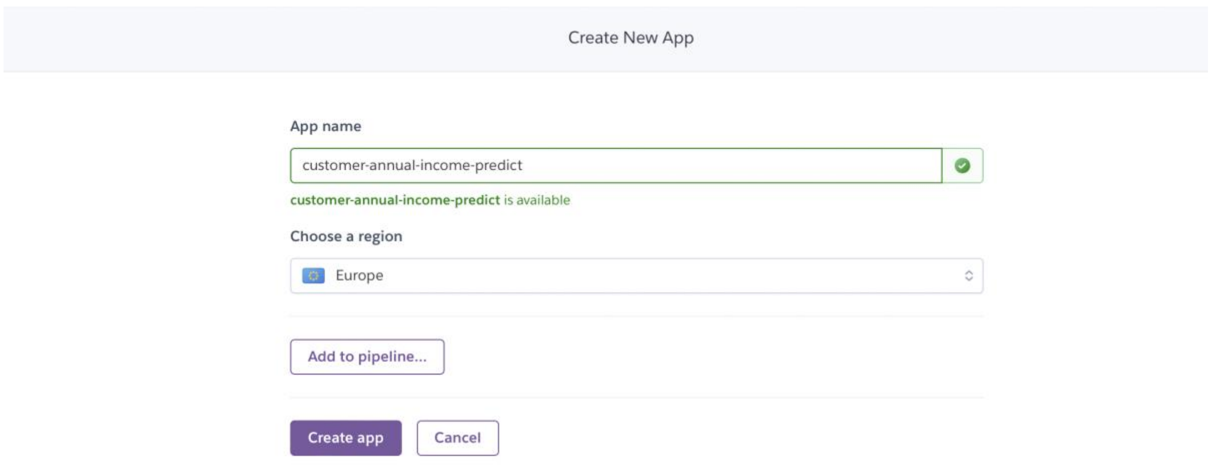
We copy the url and paste it to our web browser. Users can enter values as an input. Finally, after clicking ‘Predict’ the category of the annual income will be displayed on the screen.



```
[
  "Very Low"
]
```

Model Deployment Using Heroku

Firstly, we created a new account on Heroku. After that we clicked ‘Create New App’ button. We chose ‘customer-annual-income-predict’ as a name and ‘Europe’ as a region.



Create New App

App name

customer-annual-income-predict

customer-annual-income-predict is available

Choose a region

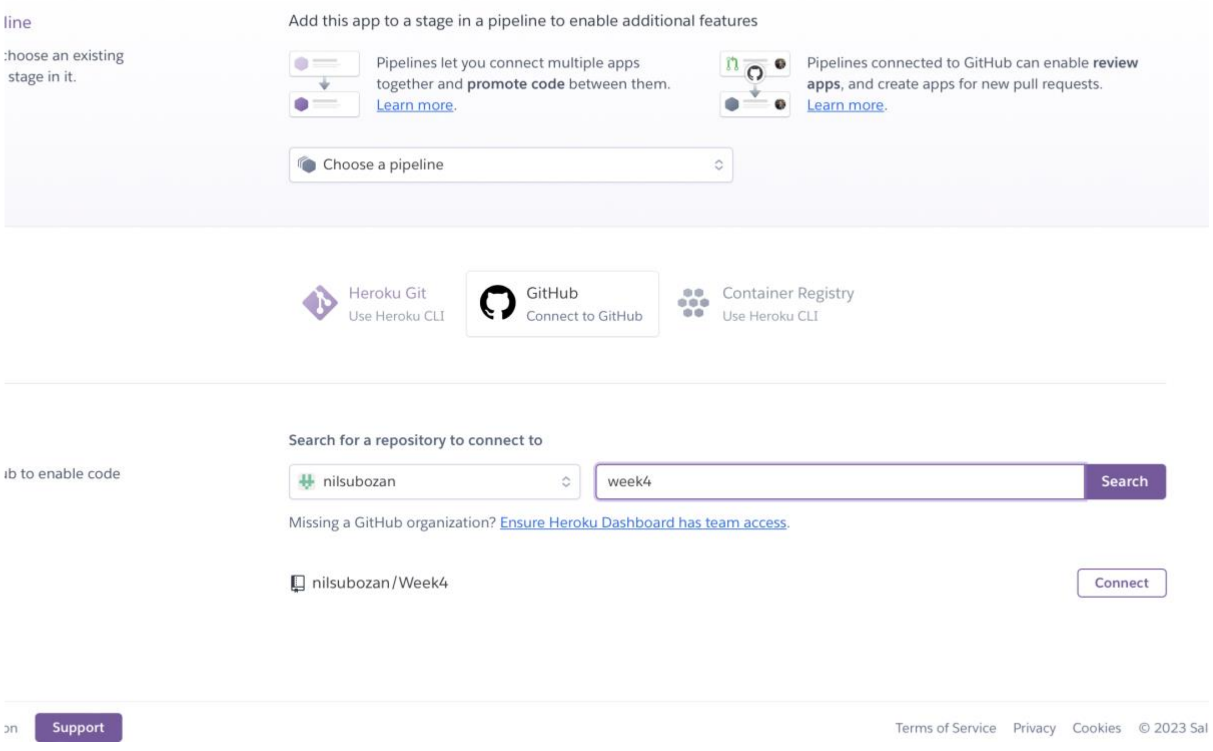
Europe

We chose to deploy our model by using Heroku. We linked our Github repository called 'Week4' to our Heroku account to be able to efficiently deploy our model.

1)We created a file called 'requirements.txt'. This file contains necessary python packages to run our app.

2) We created a file called 'Procfile'. This tells Heroku to start a web process using gunicorn with the app object defined in the app.py file as the WSGI callable.

We added both of these files to our repository on Github called 'Week4'.


line

:hoose an existing stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. Learn more.

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. Learn more.

Choose a pipeline

Heroku Git
Use Heroku CLI

GitHub
Connect to GitHub

Container Registry
Use Heroku CLI

ıb to enable code

Search for a repository to connect to

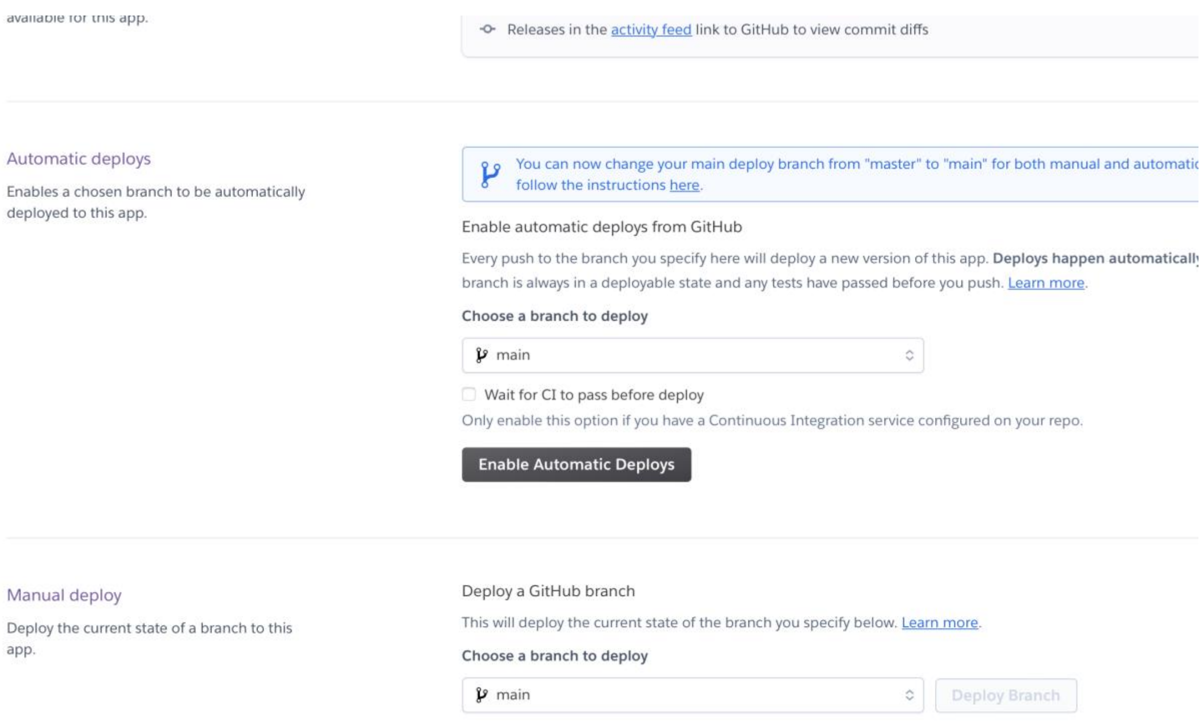
nilsubozan | week4 | Search

Missing a GitHub organization? Ensure Heroku Dashboard has team access.

nilsubozan/Week4 | Connect

ɔn | Support | Terms of Service | Privacy | Cookies | © 2023 Sal

Main branch is choosen for deploying our model.


available for this app.

Releases in the activity feed link to GitHub to view commit diffs

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic follow the instructions here.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically** branch is always in a deployable state and any tests have passed before you push. Learn more.

Choose a branch to deploy

main

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

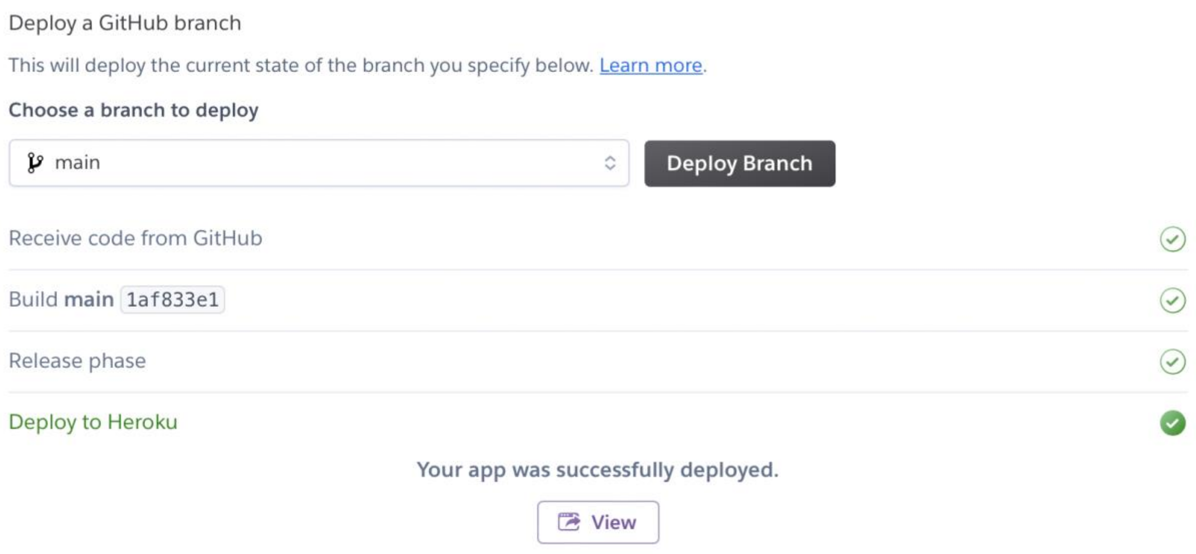
This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

main

Deploy Branch

Our app is successfully deployed on https://customer-annual-income-predict.herokuapp.com


Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

main

Deploy Branch

Receive code from GitHub ✓

Build **main** `1af833e1` ✓

Release phase ✓

Deploy to Heroku ✓

Your app was successfully deployed.

View