RUTGERS

Rutgers Business School
Newark and New Brunswick

User Funding Prediction

Pratik Prashant Guram
RU ID: 196004669

Course No: 22:548:688

MITA Capstone Project

Under the guidance of

Professor. Michail Xyntarakis

Table of Contents

1. INTRODUCTION

With the future of data practically leading towards accuracy of predicting human behavior and buying patterns, predictive analytics is, according to me, the ‘numero uno’ strategy that companies, big or small, should absolutely focus on. Data Analysis, Visualization and Prediction are concepts that need no introduction, with the importance of each growing exponentially every minute. Marketing, Sales, Finances, and practically every node of the company structure needs a strong foundation of data analytics.

Analytics has come a long way in a relatively short period of time. It can aid in multiple aspects of operations and be a real game-changer for many businesses. But to get maximum results, companies need to know how to properly utilize this technology, improve the quality of their data, and effectively manage it.

In this capstone project, I have attempted to intertwine predictive analytics through machine learning models, data visualization through visualization tools, and user design with quick access to model deployment through web application framework.

As an individual with a knack for delving deeper into trends and patterns expecting to arrive at answers that are exclusive to the observant eye, this was an enlightening experience that allowed me the freedom to express and understand each and every concept used in the project better than I previously knew.

2. DATASET

I stumbled upon this dataset on Kaggle whilst looking for datasets that would accurately allow me to display my skillset to the best of my abilities. This dataset was part of a data visualization challenge in 2015, put forth by Localytics.

BootLoader is a fictional mobile app that helps people crowdfund their creative projects. Anyone with an idea, dream, and 2-minute video can post their project on BootLoader. Others then donate money to the project to help bring it to fruition.

The dataset consists of 50,000 rows and 14 columns/attributes.

"View Project" and "Fund Project," for when a user views details about a project and for when they fund a project are the two attributes that are the focal point of this project. The events also have the following attributes:

- category: the categories that a certain project belongs to ‘Sports’, ‘Fashion’, ‘Games’, etc
- amount: how much the user donated (for "Fund Project" only)

Next up is the data supplied about the users, better called as, customer demographic data, crucial to any firms’ customer database, with more the data, better the data quality and even better predictive analysis capability.

- session_id: unique identifier for each user
- age range: one of ['18-24', '25-34', '35-44', '45-54', '55+']
- gender: one of ['M', 'F', 'U']
- city: a city in the United States
- state: a state in the United States
- latitude
- longitude
- marital status: ‘Single’ or ‘Married’
- device: one of ['iOS', 'android']

3. PROBLEM STATEMENT

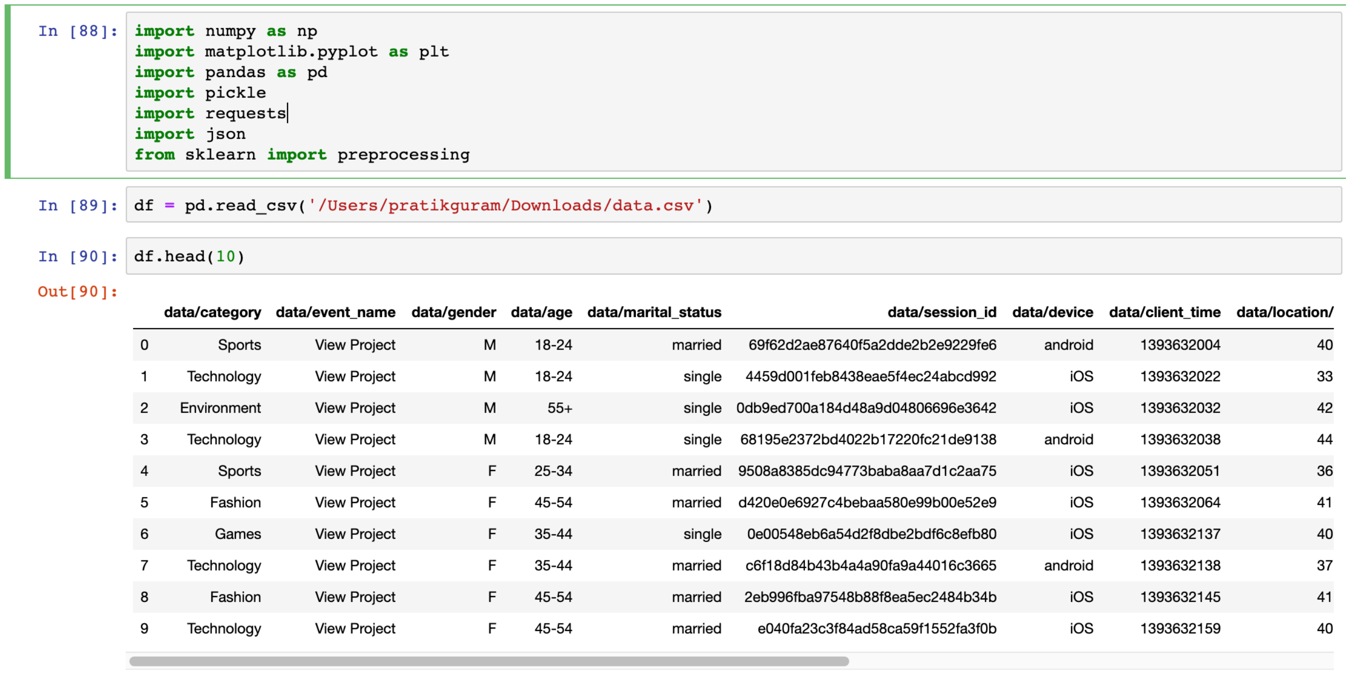
To create a machine learning model that learns from the dataset already provided and predicts if a new user will fund the project or not, based on the input provided by the user. This new user data is to be input through a webpage, which will further be provided to the ML model through web application framework and returns the output on the next page. Along with the aforementioned tasks, performing Exploratory Data Analysis and creating visualizations for better understanding of the various demographics in the dataset is also done.

4. METHODOLGY

The scripting languages, software and frameworks used in this project are Python, Jupyter notebook, Flask WSGI, Tableau, HTML.
Initially I will be explaining the ML model and structure, followed by FLASK scripting and getting the entire application framework to work as one. This will be followed by dashboards created on Tableau that help understand the dataset better and give crucial insights, allowing stakeholders and decision makers to make data driven business solutions.

(a) The ML model.

We begin by importing necessary modules, loading the dataset and having a look at the data provided.



```
In [88]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         import pickle
         import requests
         import json
         from sklearn import preprocessing
```

```
In [89]: df = pd.read_csv('/Users/pratikguram/Downloads/data.csv')
```

```
In [90]: df.head(10)
```

Out[90]:

	data/category	data/event_name	data/gender	data/age	data/marital_status	data/session_id	data/device	data/client_time	data/location/
0	Sports	View Project	M	18-24	married	69f62d2ae87640f5a2dde2b2e9229fe6	android	1393632004	40
1	Technology	View Project	M	18-24	single	4459d001feb8438eae5f4ec24abcd992	iOS	1393632022	33
2	Environment	View Project	M	55+	single	0db9ed700a184d48a9d04806696e3642	iOS	1393632032	42
3	Technology	View Project	M	18-24	single	68195e2372bd4022b17220fc21de9138	android	1393632038	44
4	Sports	View Project	F	25-34	married	9508a8385dc94773baba8aa7d1c2aa75	iOS	1393632051	36
5	Fashion	View Project	F	45-54	married	d420e0e6927c4bebaa580e99b00e52e9	iOS	1393632064	41
6	Games	View Project	F	35-44	single	0e00548eb6a54d2f8dbe2bdf6c8efb80	iOS	1393632137	40
7	Technology	View Project	F	35-44	married	c6f18d84b43b4a4a90fa9a44016c3665	android	1393632138	37
8	Fashion	View Project	F	45-54	married	2eb996fba97548b88f8ea5ec2484b34b	iOS	1393632145	41
9	Technology	View Project	F	45-54	married	e040fa23c3f84ad58ca59f1552fa3f0b	iOS	1393632159	40

Next, we check the shape of the dataset, and then the attributes to remove. Since I wanted to make a model that doesn't take too long to process, I dropped quite a number of attributes that I found to be taking a lot of time during the test processes.

```
In [91]: df.shape
Out[91]: (50000, 14)
```

```
In [92]: df.drop(["data/location/latitude", "data/location/longitude","data/location/zip_code", "data/session_id","data/amount",
```

```
In [93]: df.shape
Out[93]: (50000, 6)
```

```
In [94]: df.head(5)
Out[94]:
```

	data/category	data/event_name	data/gender	data/age	data/marital_status	data/device
0	Sports	View Project	M	18-24	married	android
1	Technology	View Project	M	18-24	single	iOS
2	Environment	View Project	M	55+	single	iOS
3	Technology	View Project	M	18-24	single	android
4	Sports	View Project	F	25-34	married	iOS

I replaced View Project and Fund Project because later on, I am going to apply Label Encoding on the remaining attributes.

```
In [95]: df.replace(['View Project', 'Fund Project'],
                   ['<=1', '>1'], inplace = True)
```

```
In [96]: df.head(5)
Out[96]:
```

	data/category	data/event_name	data/gender	data/age	data/marital_status	data/device
0	Sports	<=1	M	18-24	married	android
1	Technology	<=1	M	18-24	single	iOS
2	Environment	<=1	M	55+	single	iOS
3	Technology	<=1	M	18-24	single	android
4	Sports	<=1	F	25-34	married	iOS

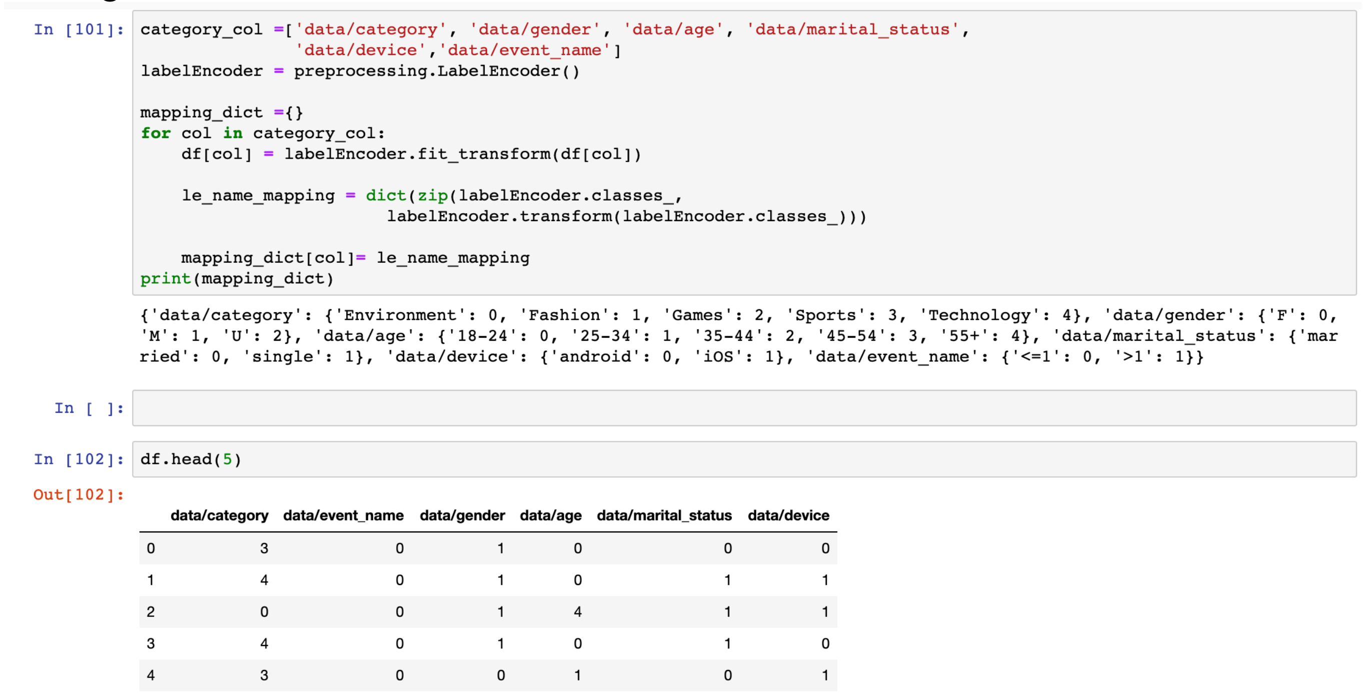
```
In [97]: print(df['data/category'].unique())
         print(df['data/event_name'].unique())
         print(df['data/gender'].unique())
         print(df['data/device'].unique())

         ['Sports' 'Technology' 'Environment' 'Fashion' 'Games']
         ['<=1' '>1']
         ['M' 'F' 'U']
         ['android' 'iOS']
```

A quick check for null/ missing values lets me know that the data is clean.

```
In [98]: df.isnull().sum()
Out[98]: data/category          0
         data/event_name        0
         data/gender            0
         data/age               0
         data/marital_status    0
         data/device            0
         dtype: int64
```

```
In [99]: df.isna().sum()
Out[99]: data/category          0
         data/event_name        0
         data/gender            0
         data/age               0
         data/marital_status    0
         data/device            0
         dtype: int64
```

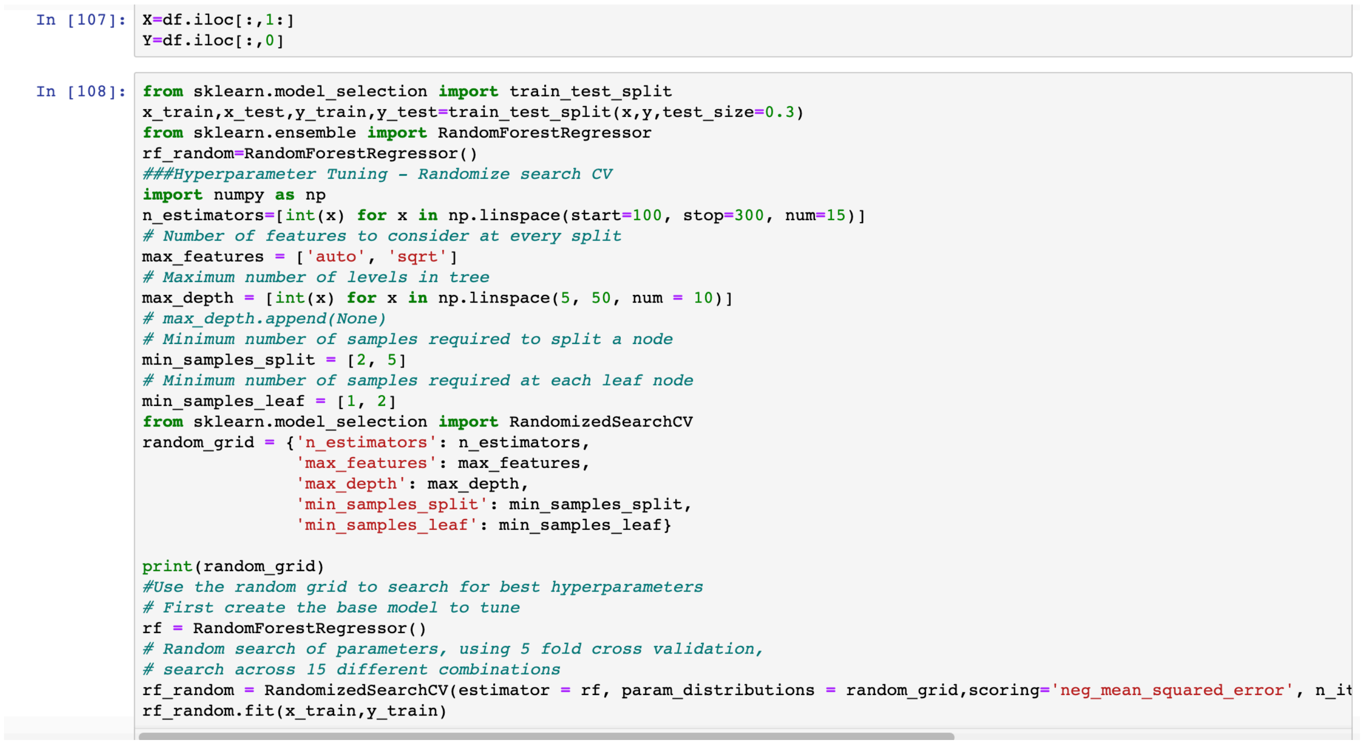
Here, I have applied Label Encoding since connecting the data input by the user would need to be matched with the data already in the dataset. Also, encoding allows the model to differentiate clearly and assign importance to certain attributes while running the model.



```
In [101]: category_col =['data/category', 'data/gender', 'data/age', 'data/marital_status',
                         'data/device','data/event_name']
          labelEncoder = preprocessing.LabelEncoder()

          mapping_dict ={}
          for col in category_col:
              df[col] = labelEncoder.fit_transform(df[col])

              le_name_mapping = dict(zip(labelEncoder.classes_,
                                  labelEncoder.transform(labelEncoder.classes_)))

              mapping_dict[col]= le_name_mapping
          print(mapping_dict)

          {'data/category': {'Environment': 0, 'Fashion': 1, 'Games': 2, 'Sports': 3, 'Technology': 4}, 'data/gender': {'F': 0,
          'M': 1, 'U': 2}, 'data/age': {'18-24': 0, '25-34': 1, '35-44': 2, '45-54': 3, '55+': 4}, 'data/marital_status': {'mar
          ried': 0, 'single': 1}, 'data/device': {'android': 0, 'iOS': 1}, 'data/event_name': {'<=1': 0, '>1': 1}}

In [ ]:

In [102]: df.head(5)

Out[102]:
```

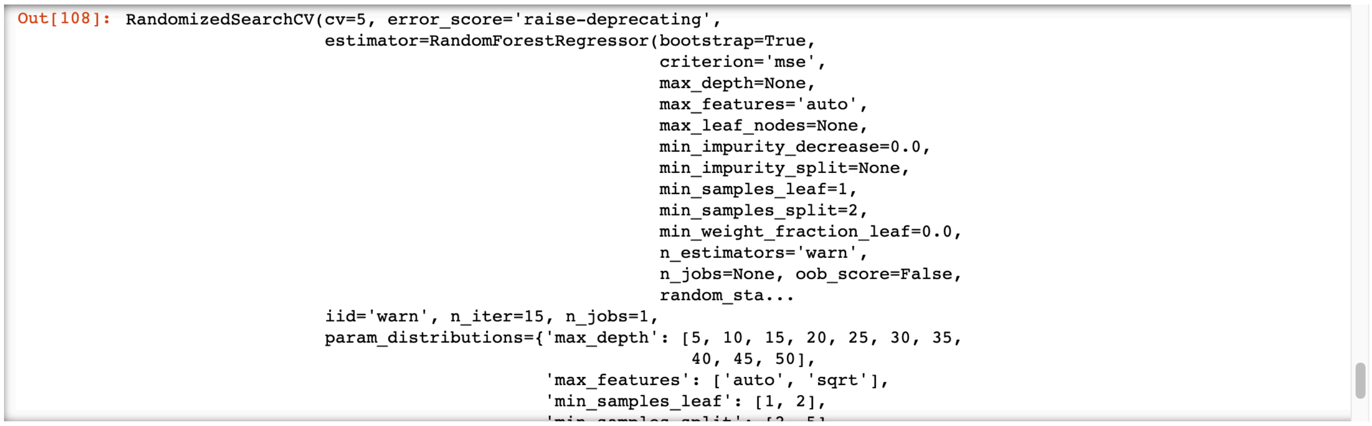
	data/category	data/event_name	data/gender	data/age	data/marital_status	data/device
0	3	0	1	0	0	0
1	4	0	1	0	1	1
2	0	0	1	4	1	1
3	4	0	1	0	1	0
4	3	0	0	1	0	1

Assigning the test and train data, and whilst applying regression and deciding on the ML model to choose. In this particular application, I have used Random Forest Regressor. A snippet of the output of regressor can be seen ahead.



```
In [107]: X=df.iloc[:,1:]
          Y=df.iloc[:,0]
```

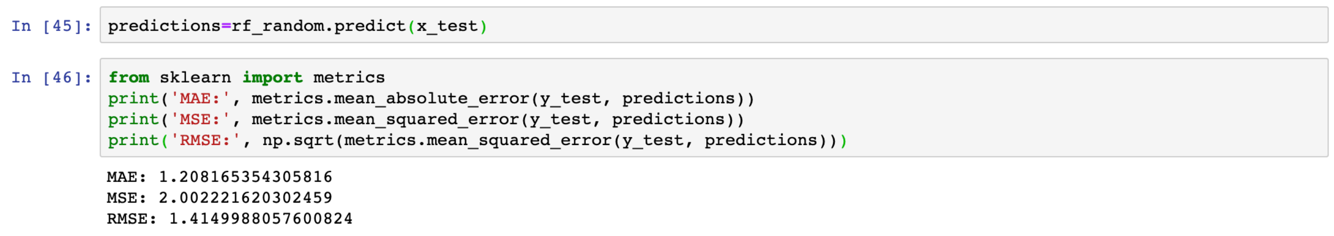
```
In [108]: from sklearn.model_selection import train_test_split
          x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
          from sklearn.ensemble import RandomForestRegressor
          rf_random=RandomForestRegressor()
          ###Hyperparameter Tuning - Randomize search CV
          import numpy as np
          n_estimators=[int(x) for x in np.linspace(start=100, stop=300, num=15)]
          # Number of features to consider at every split
          max_features = ['auto', 'sqrt']
          # Maximum number of levels in tree
          max_depth = [int(x) for x in np.linspace(5, 50, num = 10)]
          # max_depth.append(None)
          # Minimum number of samples required to split a node
          min_samples_split = [2, 5]
          # Minimum number of samples required at each leaf node
          min_samples_leaf = [1, 2]
          from sklearn.model_selection import RandomizedSearchCV
          random_grid = {'n_estimators': n_estimators,
                         'max_features': max_features,
                         'max_depth': max_depth,
                         'min_samples_split': min_samples_split,
                         'min_samples_leaf': min_samples_leaf}

          print(random_grid)
          #Use the random grid to search for best hyperparameters
          # First create the base model to tune
          rf = RandomForestRegressor()
          # Random search of parameters, using 5 fold cross validation,
          # search across 15 different combinations
          rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_it
          rf_random.fit(x_train,y_train)
```



```
Out[108]: RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                             estimator=RandomForestRegressor(bootstrap=True,
                                                             criterion='mse',
                                                             max_depth=None,
                                                             max_features='auto',
                                                             max_leaf_nodes=None,
                                                             min_impurity_decrease=0.0,
                                                             min_impurity_split=None,
                                                             min_samples_leaf=1,
                                                             min_samples_split=2,
                                                             min_weight_fraction_leaf=0.0,
                                                             n_estimators='warn',
                                                             n_jobs=None, oob_score=False,
                                                             random_sta...
                             iid='warn', n_iter=15, n_jobs=1,
                             param_distributions={'max_depth': [5, 10, 15, 20, 25, 30, 35,
                                                                40, 45, 50],
                                                  'max_features': ['auto', 'sqrt'],
                                                  'min_samples_leaf': [1, 2],
```

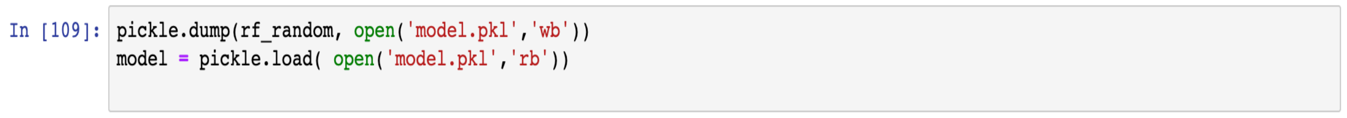
I created a few more. ipynb files with various other models, which can be substituted in place of the Random Forest, such as Logistic regression, Decision Tree Classifier, etc.

Next, we have a look at the MAE, MSE and RMSE values. The lower the values are, the better. MSE and RMSE are really useful to see if the outliers are messing with your predictions.



```
In [45]: predictions=rf_random.predict(x_test)

In [46]: from sklearn import metrics
         print('MAE:', metrics.mean_absolute_error(y_test, predictions))
         print('MSE:', metrics.mean_squared_error(y_test, predictions))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))

         MAE: 1.208165354305816
         MSE: 2.002221620302459
         RMSE: 1.4149988057600824
```

Finally, we dump the model using pickle to later use it when required while scripting flask. "*Pickling*" is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

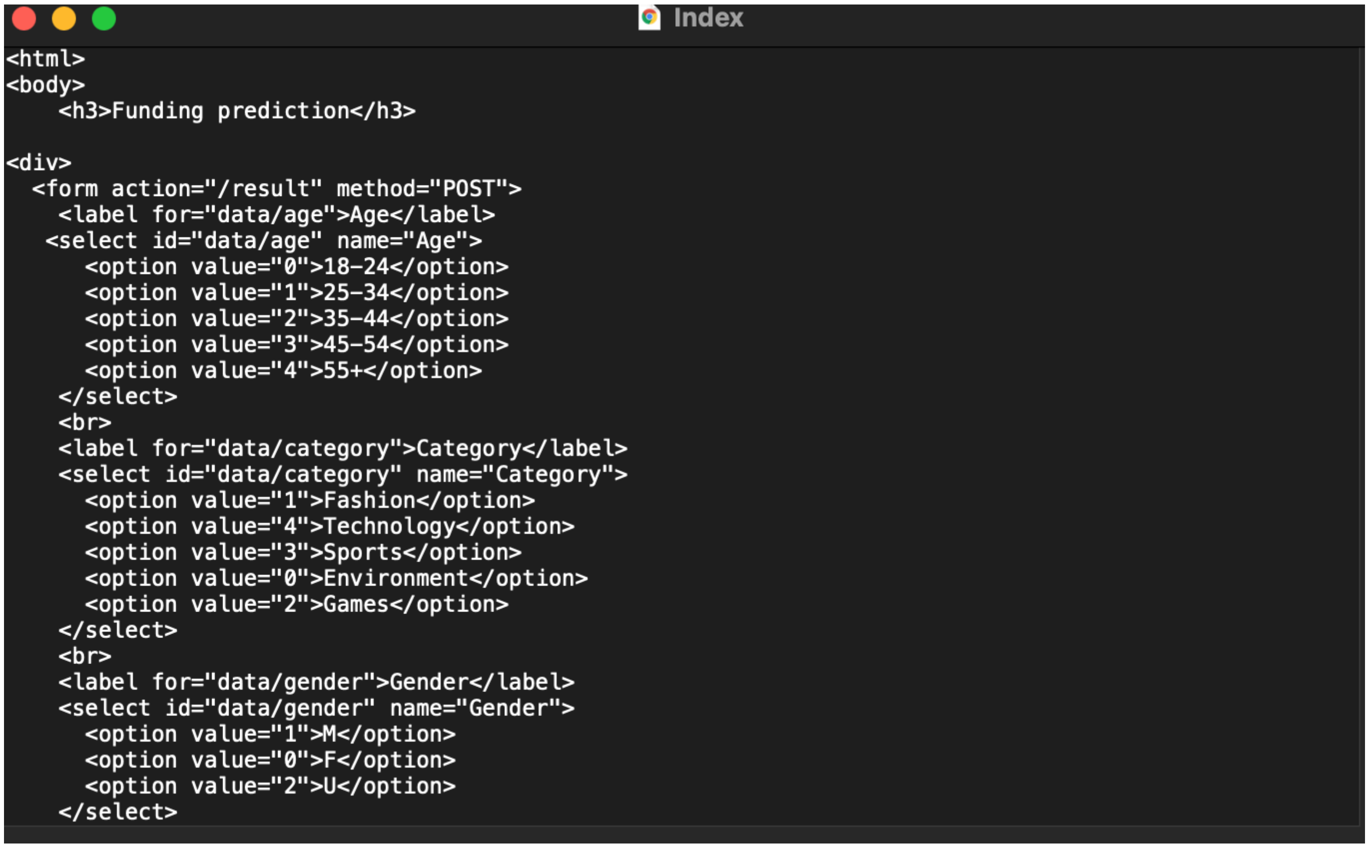


```
In [109]: pickle.dump(rf_random, open('model.pkl','wb'))
          model = pickle.load( open('model.pkl','rb'))
```

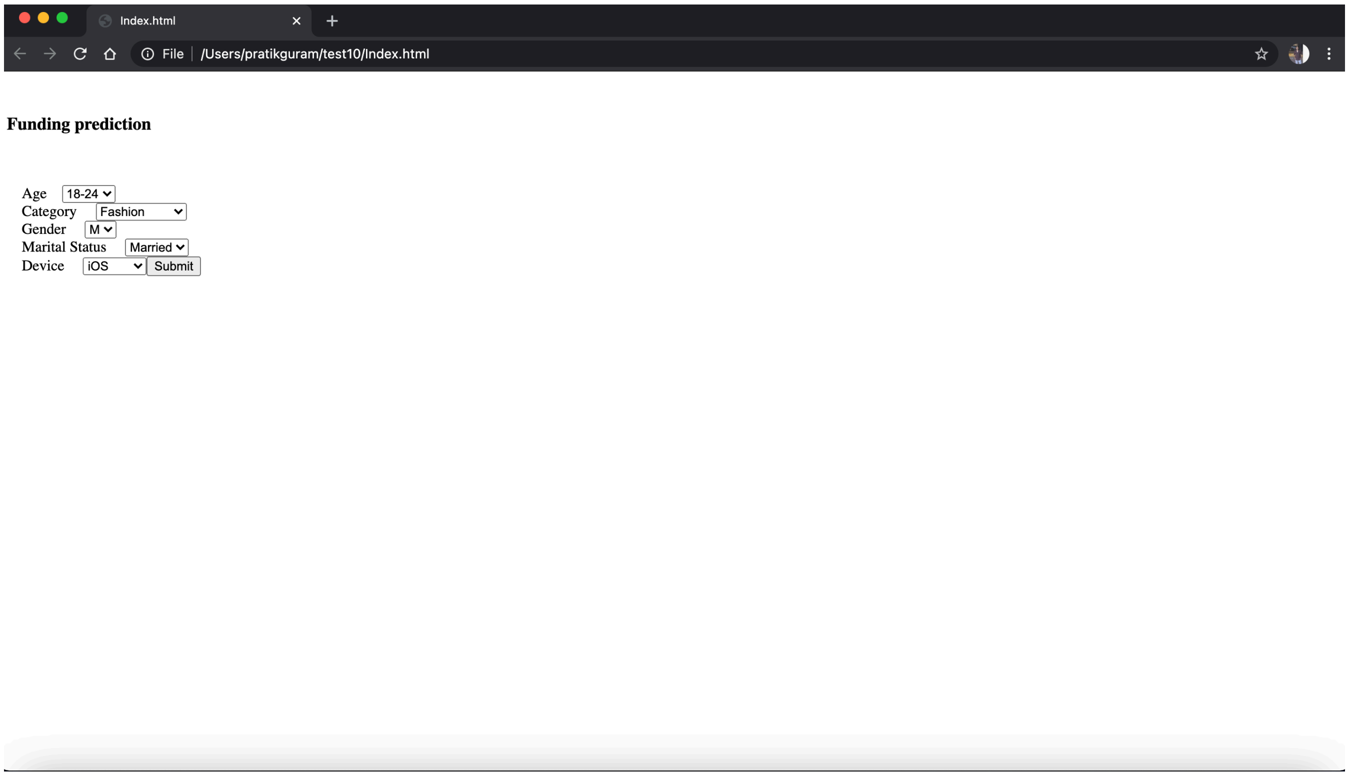
(b)The Webpages

I created two webpages, one for submitting the form with demographic data that the user inputs, and another one which will display the result, whether the user will fund the project or not. Basic HTML script has been used, wherein the code is written in TextEdit, and saved as an .html file. The two files are named as index.html and result.html.

The following is a snippet of the text form of index.html followed by the webpage.



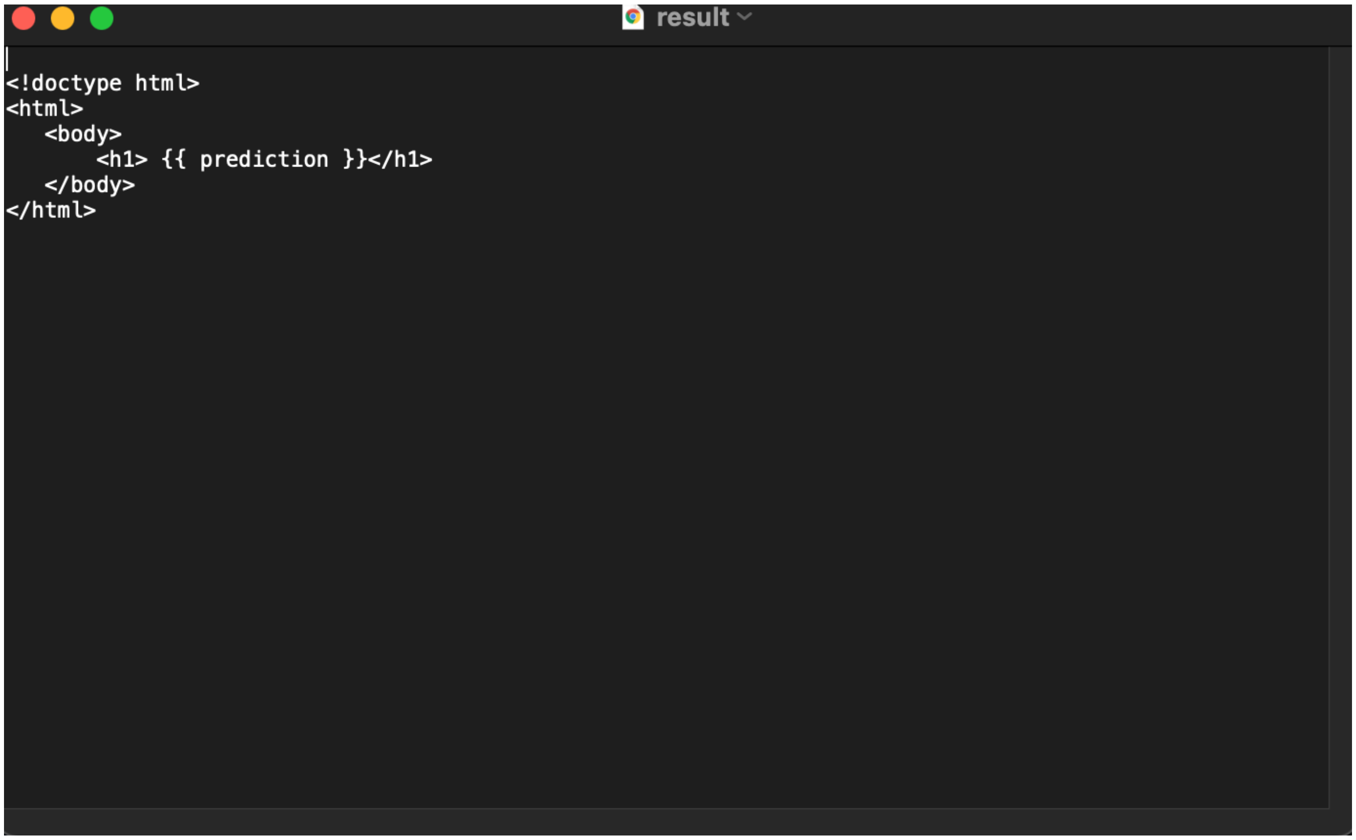
```
<html>
<body>
    <h3>Funding prediction</h3>

<div>
  <form action="/result" method="POST">
    <label for="data/age">Age</label>
   <select id="data/age" name="Age">
      <option value="0">18-24</option>
      <option value="1">25-34</option>
      <option value="2">35-44</option>
      <option value="3">45-54</option>
      <option value="4">55+</option>
    </select>
    <br>
    <label for="data/category">Category</label>
    <select id="data/category" name="Category">
      <option value="1">Fashion</option>
      <option value="4">Technology</option>
      <option value="3">Sports</option>
      <option value="0">Environment</option>
      <option value="2">Games</option>
    </select>
    <br>
    <label for="data/gender">Gender</label>
    <select id="data/gender" name="Gender">
      <option value="1">M</option>
      <option value="0">F</option>
      <option value="2">U</option>
    </select>
```



Funding prediction

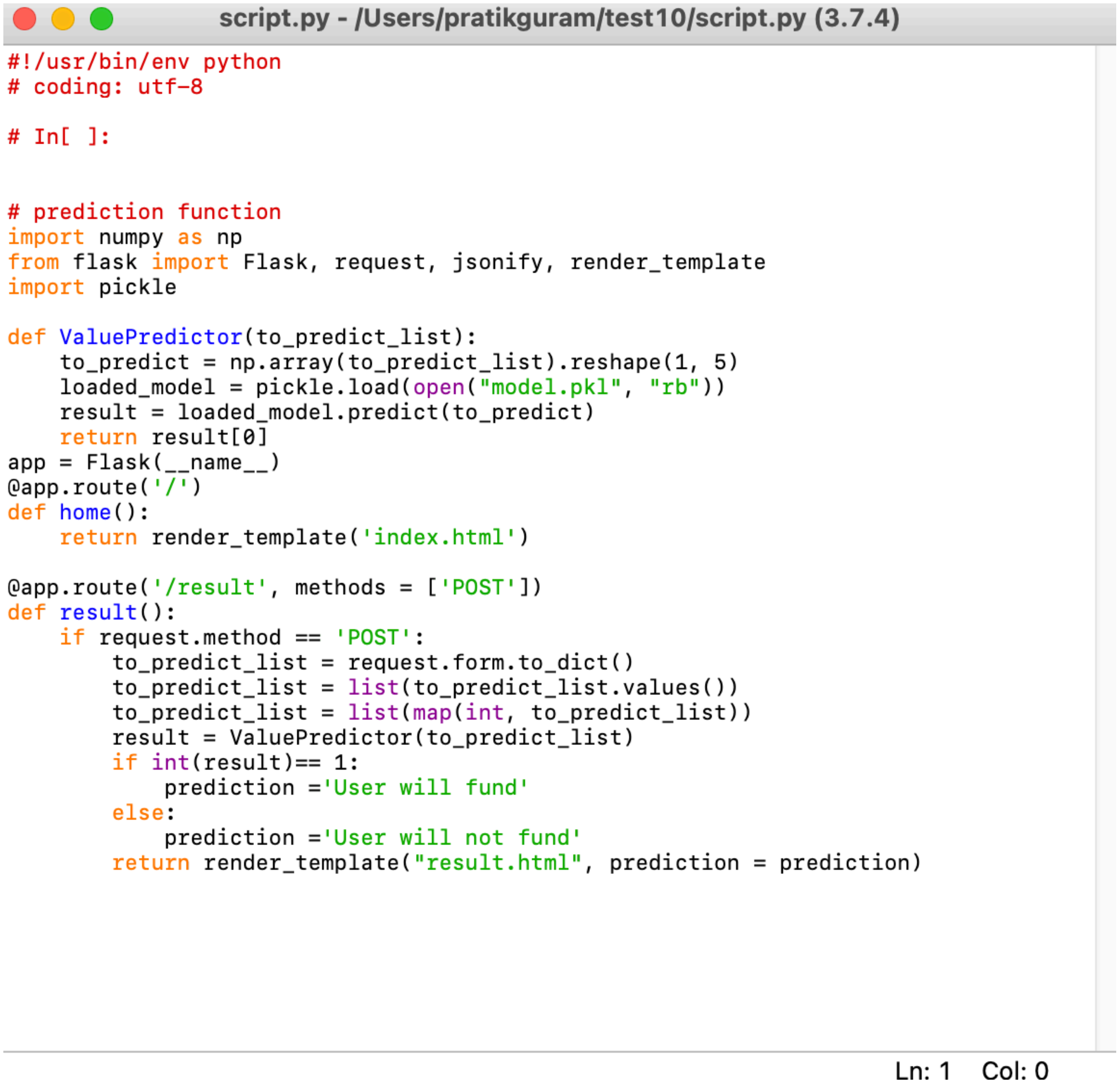
Age 18-24
Category Fashion
Gender M
Marital Status Married
Device iOS Submit

Next up is a snippet of text form of result.html. The webpage version will be displayed along with the working of the application ahead.



```
<!doctype html>
<html>
   <body>
       <h1> {{ prediction }}</h1>
   </body>
</html>
```

(c) The FLASK script



```python
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


# prediction function
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

def ValuePredictor(to_predict_list):
    to_predict = np.array(to_predict_list).reshape(1, 5)
    loaded_model = pickle.load(open("model.pkl", "rb"))
    result = loaded_model.predict(to_predict)
    return result[0]
app = Flask(__name__)
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/result', methods = ['POST'])
def result():
    if request.method == 'POST':
        to_predict_list = request.form.to_dict()
        to_predict_list = list(to_predict_list.values())
        to_predict_list = list(map(int, to_predict_list))
        result = ValuePredictor(to_predict_list)
        if int(result)== 1:
            prediction ='User will fund'
        else:
            prediction ='User will not fund'
        return render_template("result.html", prediction = prediction)
```

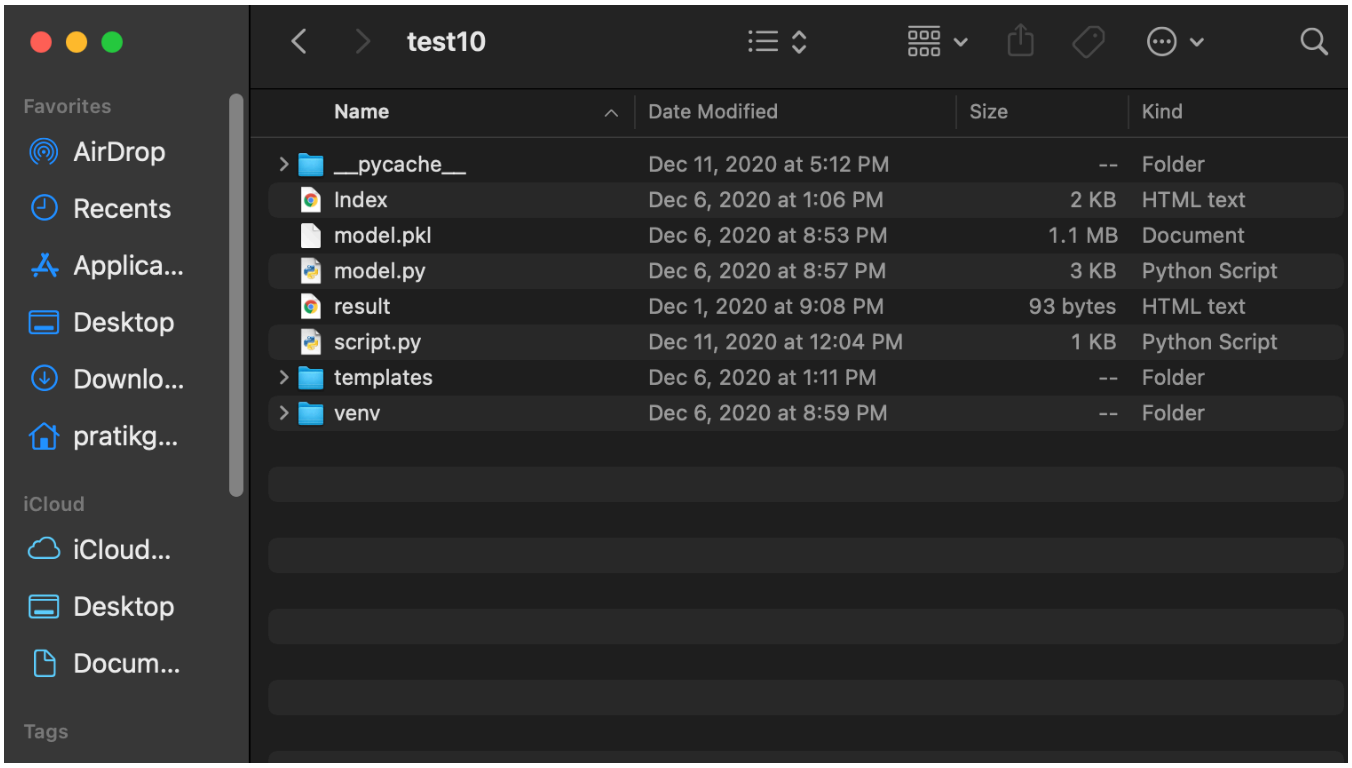
Initially we load the model that was dumped using pickle. Flask allows the data input by the user to be fed to this model, and return the output when result is called after clicking on the 'Submit' button on the webpage.
Here, after the form is submitted, the form values are stored in variable to_predict_list in the form of dictionary. We convert it into a list of the dictionary's values and pass it as an argument to ValuePredictor () function. In this function, we load the model.pkl file and predict the new values and return the result.

This result/prediction (User will fund/not fund) is then passed as an argument to the template engine with the html page to be displayed.

(d) Working application

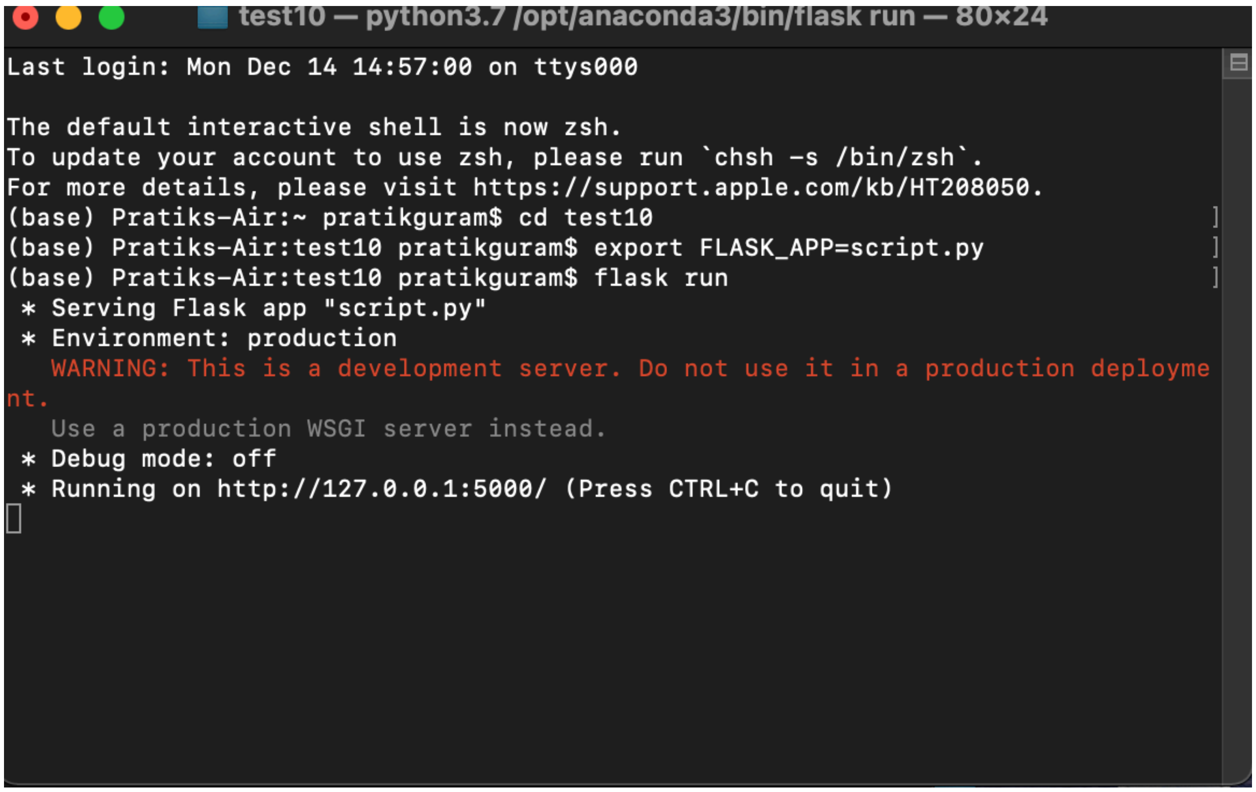
We need to make sure the necessary files are all in the same folder for the project to work. The files that need to be together are model.pkl (the pickled version of the model), model.py (the model with python extension), script.py (the FLASK script) and the webpages: index.html and result.html.

For the aforementioned steps to take place, I have created a file named test10, where these files are stored.



Name	Date Modified	Size	Kind
__pycache__	Dec 11, 2020 at 5:12 PM	--	Folder
Index	Dec 6, 2020 at 1:06 PM	2 KB	HTML text
model.pkl	Dec 6, 2020 at 8:53 PM	1.1 MB	Document
model.py	Dec 6, 2020 at 8:57 PM	3 KB	Python Script
result	Dec 1, 2020 at 9:08 PM	93 bytes	HTML text
script.py	Dec 11, 2020 at 12:04 PM	1 KB	Python Script
templates	Dec 6, 2020 at 1:11 PM	--	Folder
venv	Dec 6, 2020 at 8:59 PM	--	Folder

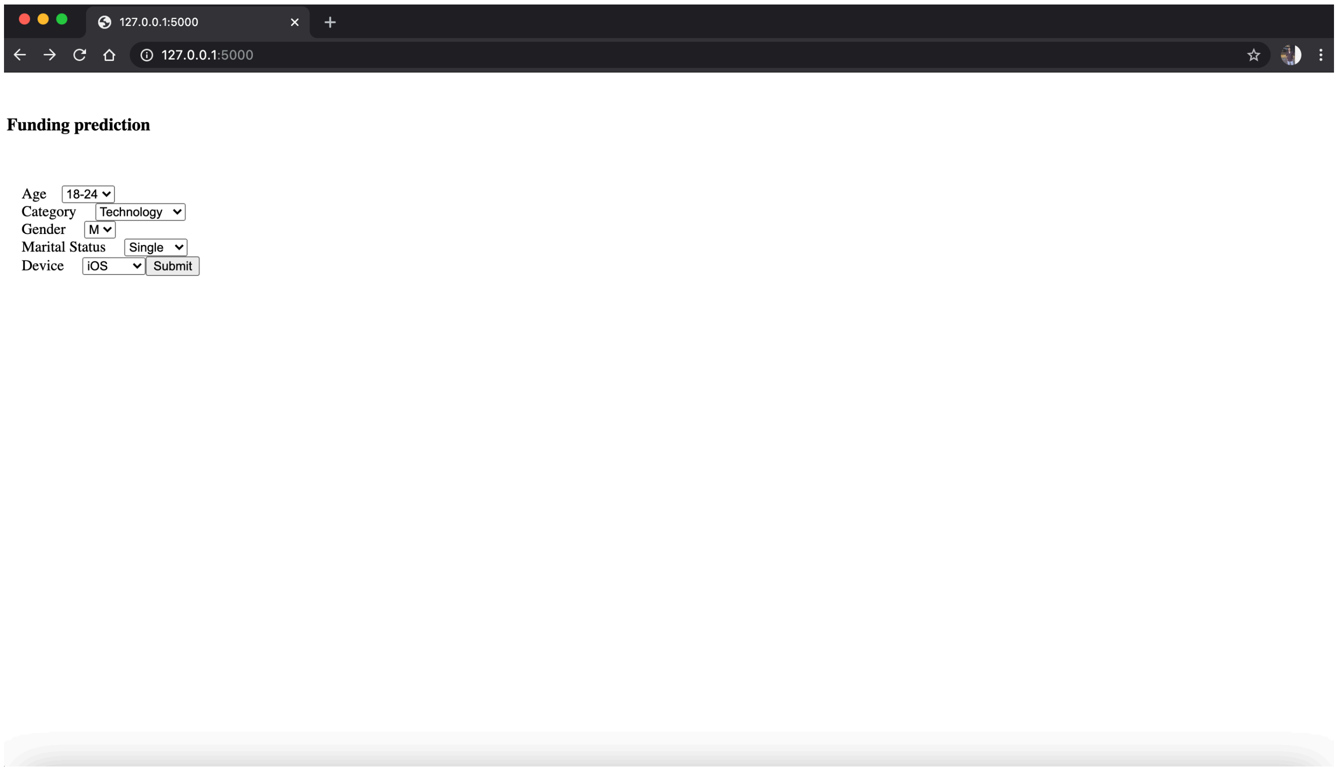
Open command window and follow the steps in the snippet below.



```
Last login: Mon Dec 14 14:57:00 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Pratiks-Air:~ pratikguram$ cd test10
(base) Pratiks-Air:test10 pratikguram$ export FLASK_APP=script.py
(base) Pratiks-Air:test10 pratikguram$ flask run
 * Serving Flask app "script.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

This will get the application running. To check the webpage, go to http://127.0.0.1:5000/

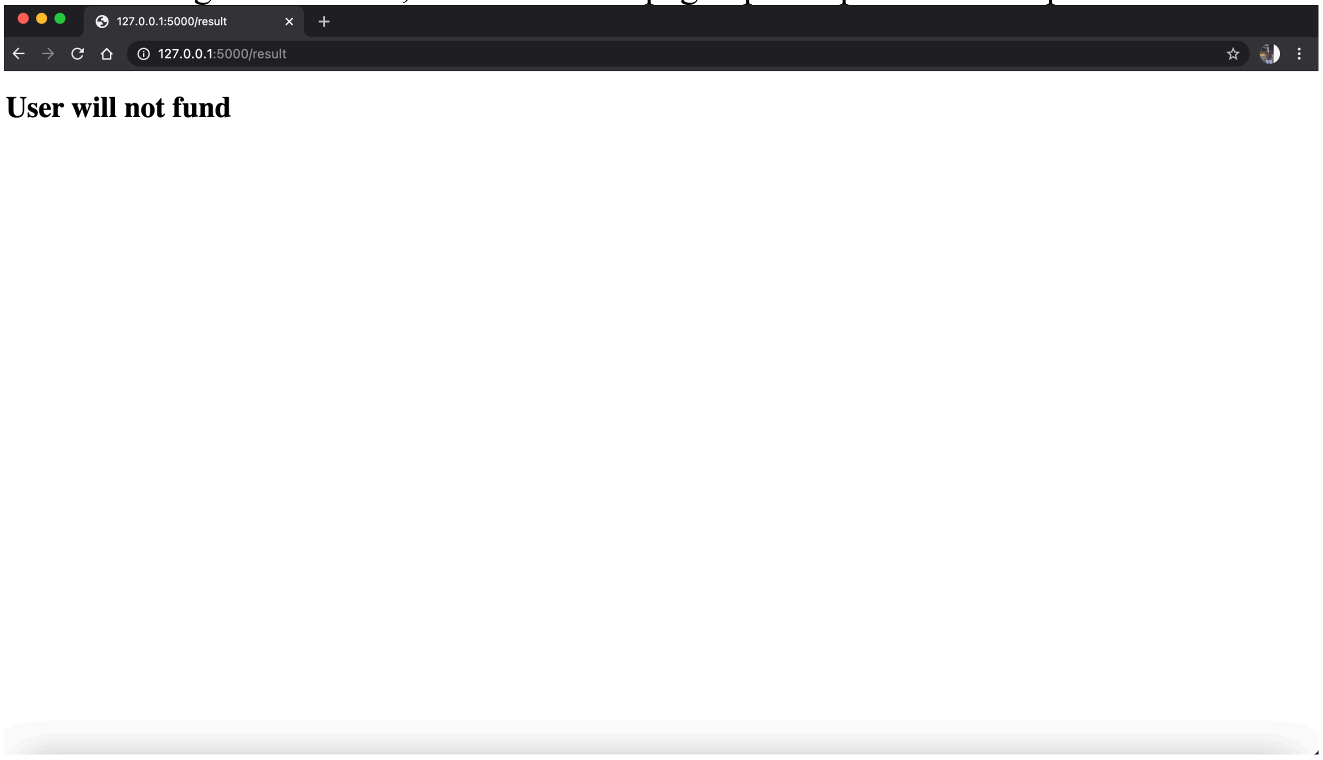
This will open up the index.html webpage we have previously seen. Input the details from the dropdown boxes and click on ‘Submit’.


127.0.0.1:5000

Funding prediction

Age 18-24
Category Technology
Gender M
Marital Status Single
Device iOS

After clicking on ‘Submit’, the result.html page opens up with the output.


127.0.0.1:5000/result

User will not fund

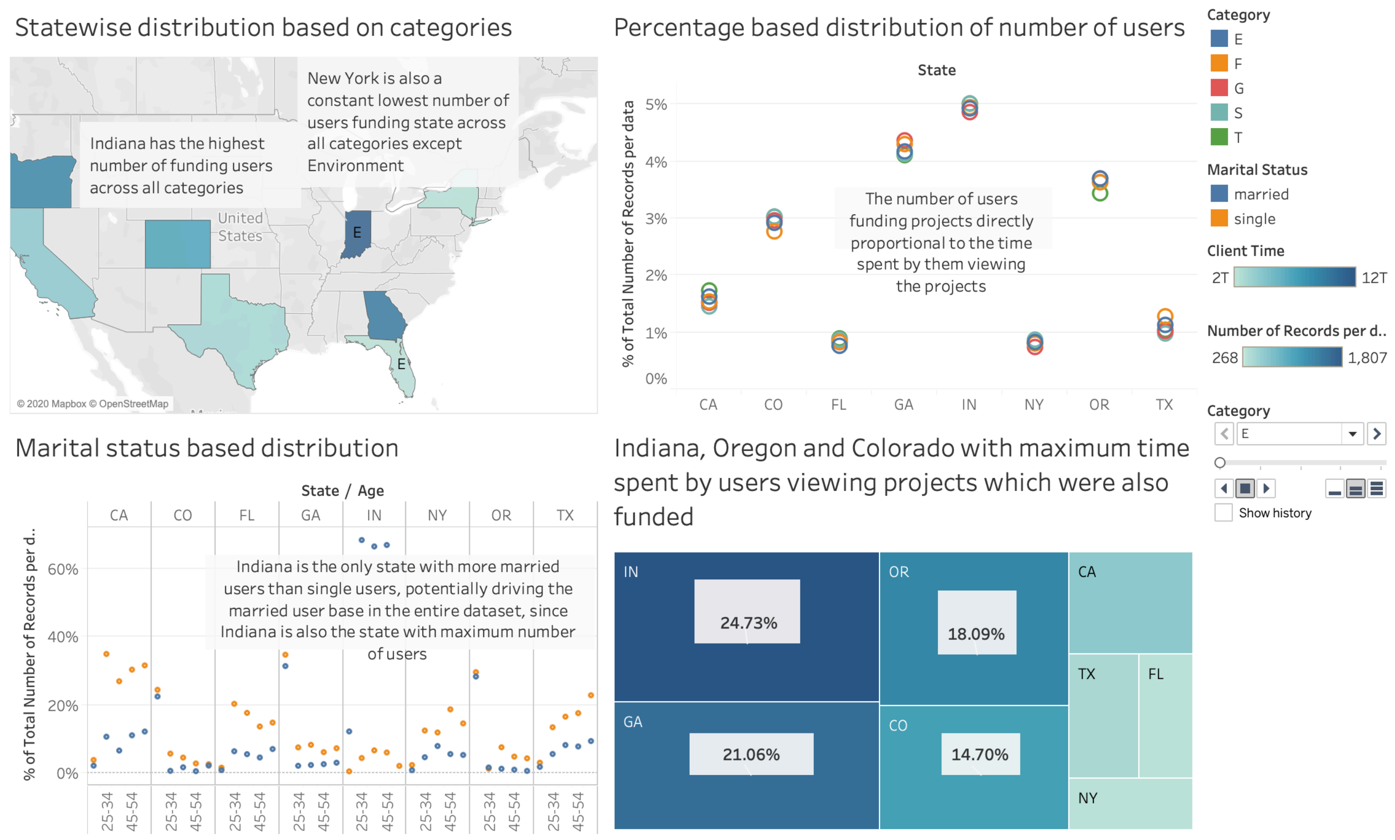
‘User will fund’ is another output that is displayed when the model arrives at a conclusion that the data input by the user predicts that the user will fund.

Such a collaboration of software and scripts is crucial learning and exploring the vast domain of data science and analysis. Furthermore, effectively portraying the insights gained creating interactive dashboards is beneficial to maximizing project capabilities.

(e) Tableau dashboards

Following are the interactive dashboards created:

I have separated the states into two parts, Above average number of users and Below average number of users. Since the number of states with Below average number of users was high, I have considered only the top three states with Below average number of users. These dashboards can be interacted with by changing Categories, since I have added Categories in Pages.


Statewise distribution based on categories

Indiana has the highest number of funding users across all categories

New York is also a constant lowest number of users funding state across all categories except Environment

Percentage based distribution of number of users

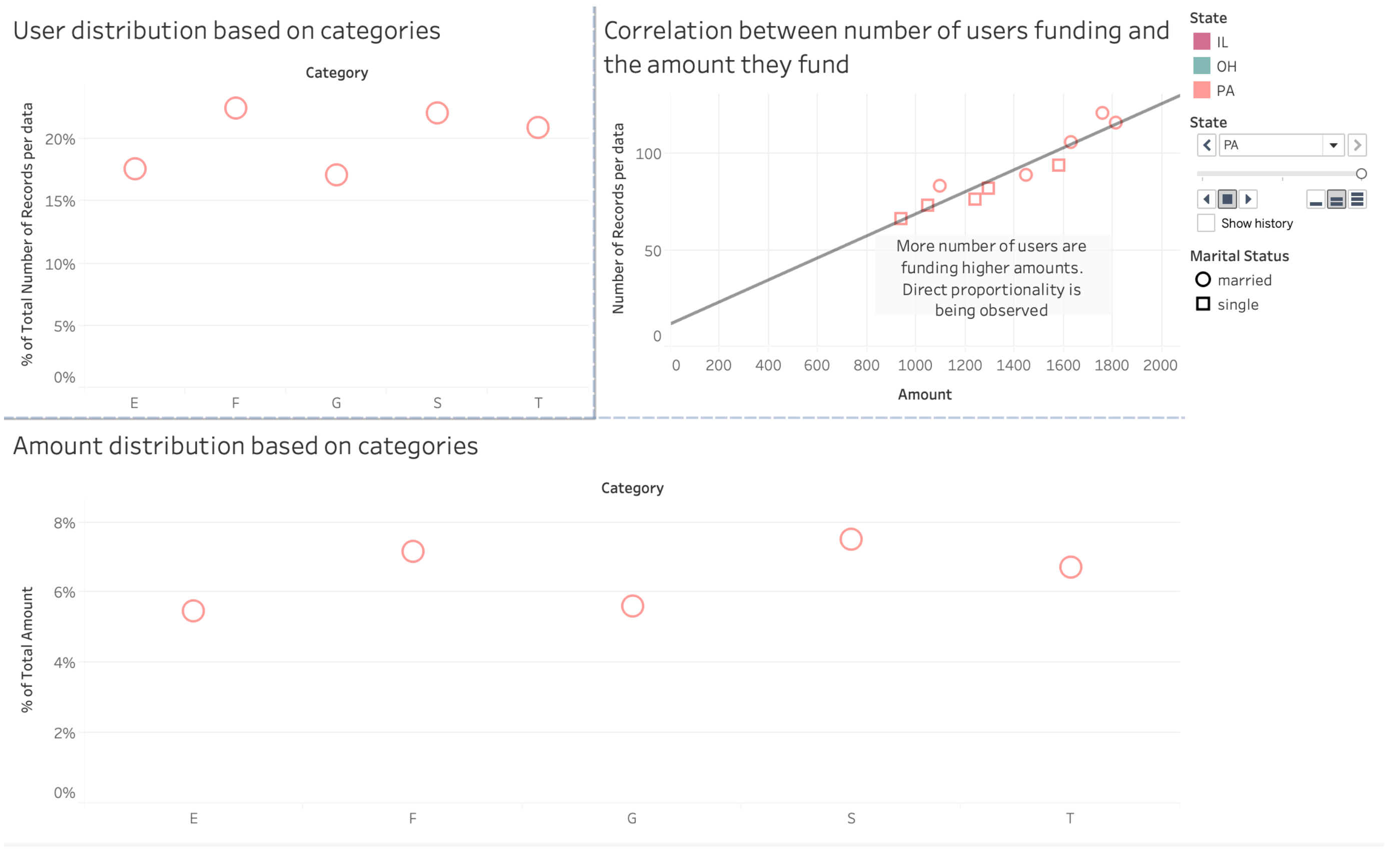
The number of users funding projects directly proportional to the time spent by them viewing the projects

Marital status based distribution

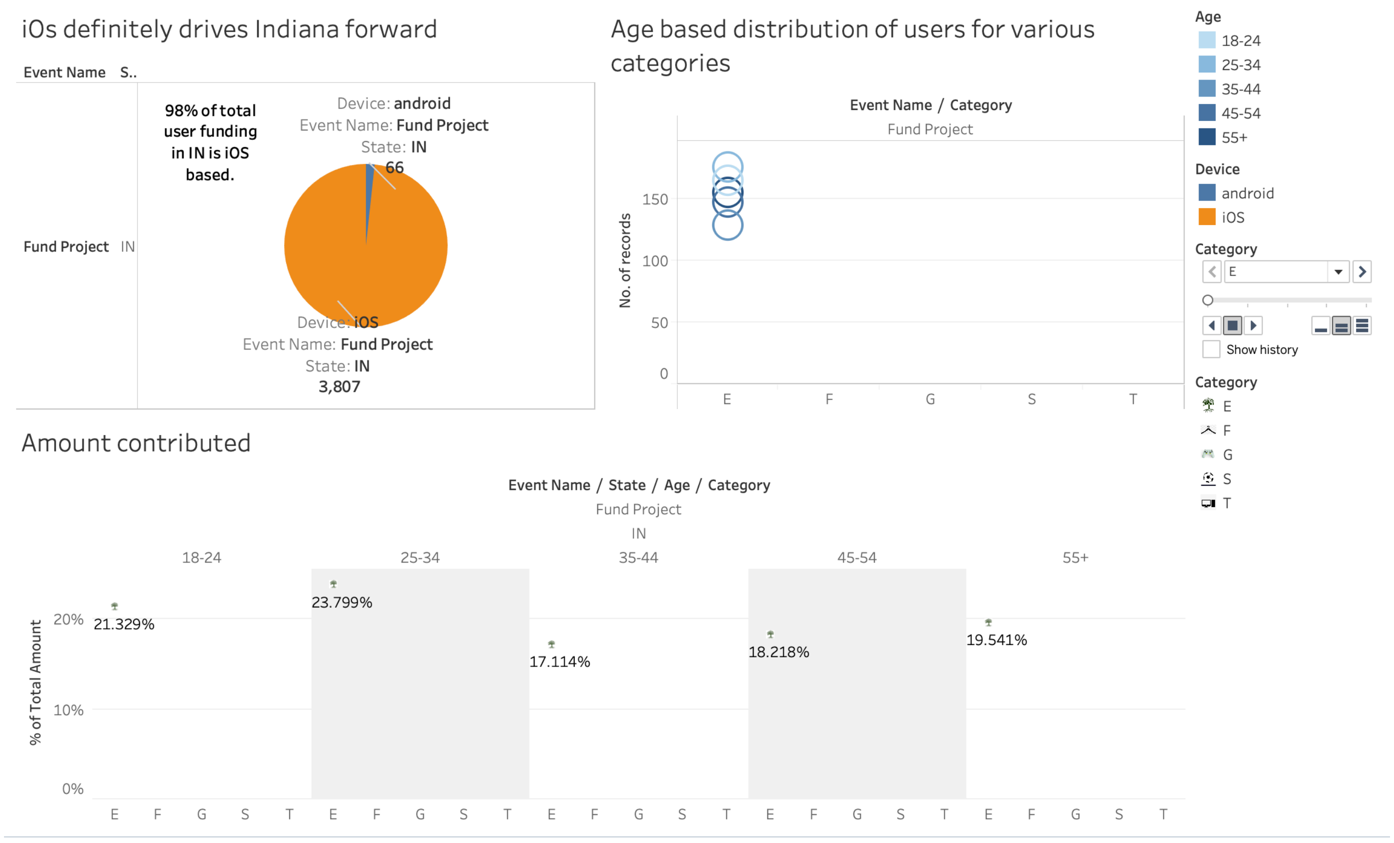
Indiana is the only state with more married users than single users, potentially driving the married user base in the entire dataset, since Indiana is also the state with maximum number of users

Indiana, Oregon and Colorado with maximum time spent by users viewing projects which were also funded

States with Above average number of users



States with Above average number of users



A brief overview of Indiana, the state that contributes the largest number of users in the database

7. CONCLUSION

I was able to create a working solution that amalgamated data analysis, visualization and prediction in the same project. Prioritizing each node of data science equally works best in a collaborative effort to get such an application running. Most of the insights are already mentioned on the dashboards itself. The rest of them are dependent on the user's preferences at receiving information. That is the prime reason for making it an interactive dashboard. This project can also be used as walkthrough on how to deploy an ML model using Flask, as I have mentioned every step required to get the application working in detail. The project can also be found at my GitHub https://github.com/pratikguram .

8. REFERENCES

https://www.geeksforgeeks.org/
https://github.com/localytics/data-viz-challenge