

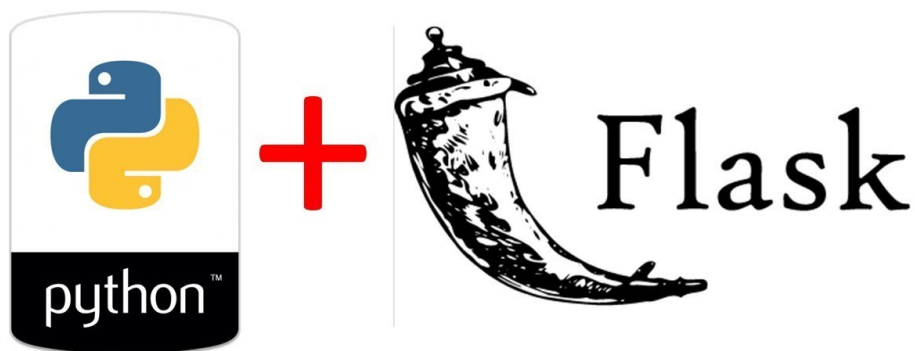
Deploy Machine Learning models in Production as APIs (using Flask)

Breakdown of this Notebook:

- 1) Install **flask & gunicorn**
- 2) Create Machine Learning Model
- 3) Saving Machine Learning Model
- 4) Creating an API using Flask
- 5) Create an front-end application
- 6) Hosting on personal server

Environment setup

REST API using Flask

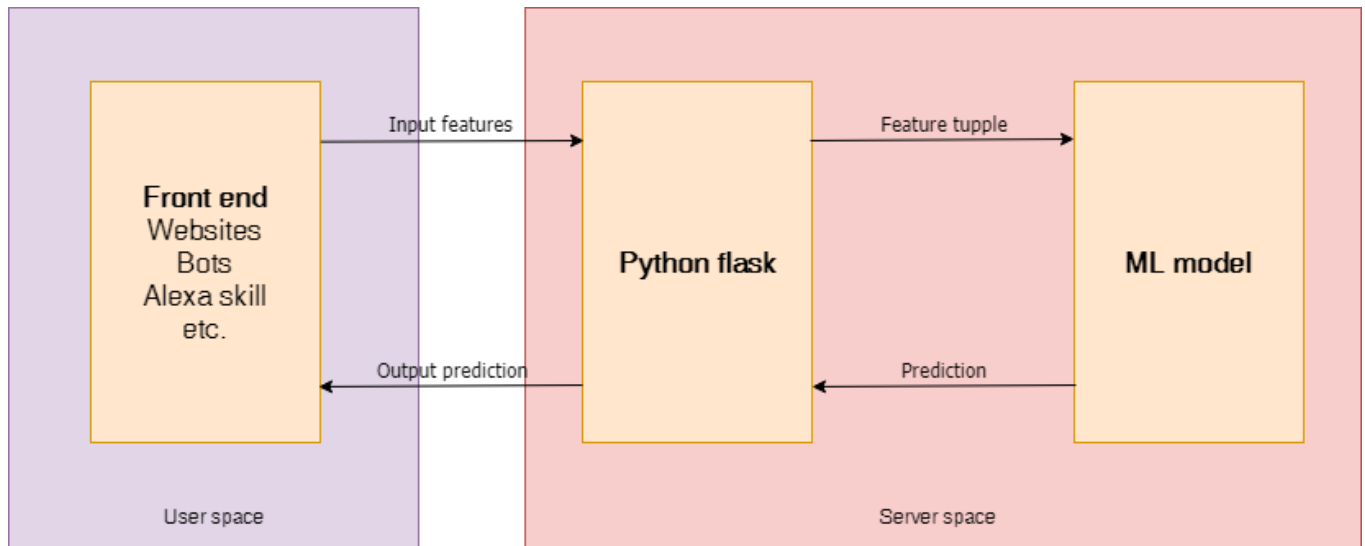


Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

In Anaconda Distribution install Flask:

- 1) pip install flask
- 2) pip install gunicorn

Explanation of Data Flow using Flask



Steps to be taken:

- 1) Train the Machine Learning Model
- 2) Save the trained model object as a pickle file (serialization)
- 3) Create a flask environment that will have an API endpoint which would encapsulate our trained model and enable it to receive inputs (features) through GET requests over HTTP/HTTPS and then return the output after de-serializing the earlier serialized model
- 4) Upload the flask script along with the trained Machine Learning Model
- 5) Make requests to the hosted flask script through a website or any other application capable of sending HTTP/HTTPS requests

Machine Learning Model

Train our Machine Learning Model

Create a Python file model.py

In [3]:

```
# Multiple Linear Regression

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle #A module called pickle helps perform serialization and deserialization in python.

# Importing the dataset
dataset = pd.read_csv('50_Startup.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values

# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X, y)

# Saving model to disk
pickle.dump(regressor, open('model.pkl', 'wb'))

# Loading model to compare the results
model = pickle.load(open('model.pkl', 'rb'))
print(model.predict([[16000, 135000, 450000]]))
```

[71646.10721559]

Here after running the above code, a file called “model.pkl” is created using dump which is the trained model that can be transferred anywhere and used after de-serialization as given using load.

Flask setup

Let's first set up the flask server on the local host and later deploy it on python

The following script starts the flask server on localhost and default port (5000) making the URL:

<http://127.0.0.1:5000/> (<http://127.0.0.1:5000/>)

In []:

```

import numpy as np
from flask import Flask, request, jsonify, render_template
# Flask - To import flask,request - Getting the request data (for which predictions are
to be made),
# jsonify - jsonify our predictions and send the response back
import pickle

app = Flask(__name__) #create an instance of flask.
model = pickle.load(open('model1.pkl', 'rb')) #Load our model pickle file

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

# @app.route('/') is used to tell flask what url should trigger the function index()
# and in the function index we use render_template('index.html') to display the script
index.html in the browser.
@app.route('/predict',methods=['POST'])
""" Let's write a function predict() which will do:

1) Load the persisted model into memory when the application starts,
2) Create an API endpoint that takes input variables, transforms them into the appropri
ate format, and returns predictions."""

def predict():
    """
    For rendering results on HTML GUI
    """
    int_features = [int(x) for x in request.form.values()] #Take Input as integer value
s
    final_features = [np.array(int_features)] #convert it into array
    prediction = model.predict(final_features) #PRedict

    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text='Profit should be $ {}'.format
(output))

@app.route('/predict_api',methods=['POST'])
def predict_api():
    """
    For direct API calls through request
    """
    data = request.get_json(force=True)
    prediction = model.predict([np.array(list(data.values()))])

    output = prediction[0]
    return jsonify(output)

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

```

In **App Route** `app.route` decorator is used for specifying the flask app route over the web.

1) `"/` simply means the home that is [“http://127.0.0.1:5000/”](http://127.0.0.1:5000/) (`http://127.0.0.1:5000/`)

2) `/predict/` means <http://127.0.0.1:5000/predict/> (`http://127.0.0.1:5000/predict/`).

Routing

After initializing the app, we have to tell Flask what we want to do when the web page loads. The line `@app.route("/", methods = ["GET", "POST"])` tells Flask what to do when we load the home page of our website. The GET method is the type of request your web browser will send the website when it accesses the URL of the web page. Don't worry about the POST method because it's a request conventionally used when a user want to change a website, and it doesn't have much relevance for us in this deployment process.

Setting up the index.html

In []:

```
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
  <meta charset="UTF-8">
  <title>ML API</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
  <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
  <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
  <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
  <div class="login">
    <h1>Predict Profit</h1>

    <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict') }}"method="post">
      <input type="text" name="R&D Spend" placeholder="R&D Spend" required="required" /> <!--Taking Input of Marketing -->
      <input type="text" name="Administration" placeholder="Administration" required="required" /><!--Input of Administration -->
      <input type="text" name="Marketing" placeholder="Marketing" required="required" /> <!-- Input for Marketing -->

      <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>

    <br>
    <br>
    {{ prediction_text }} <!-- Predicted Value -->

  </div>

</body>
</html>
```

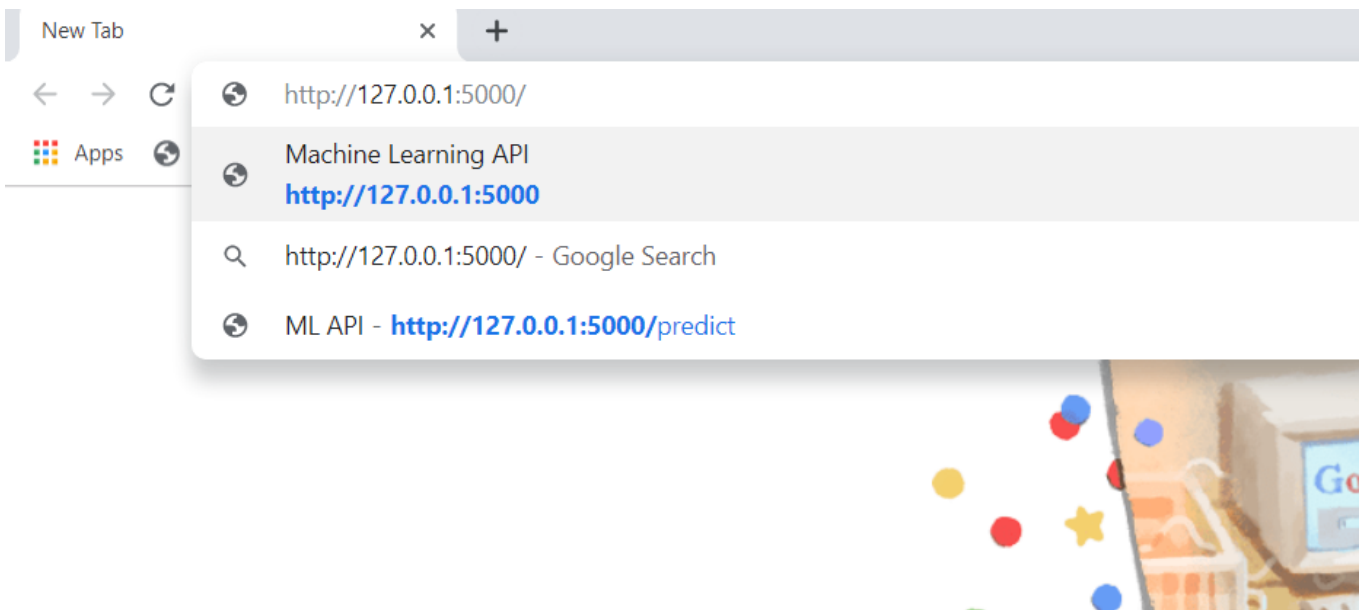
Save model.py, app.py and index.html

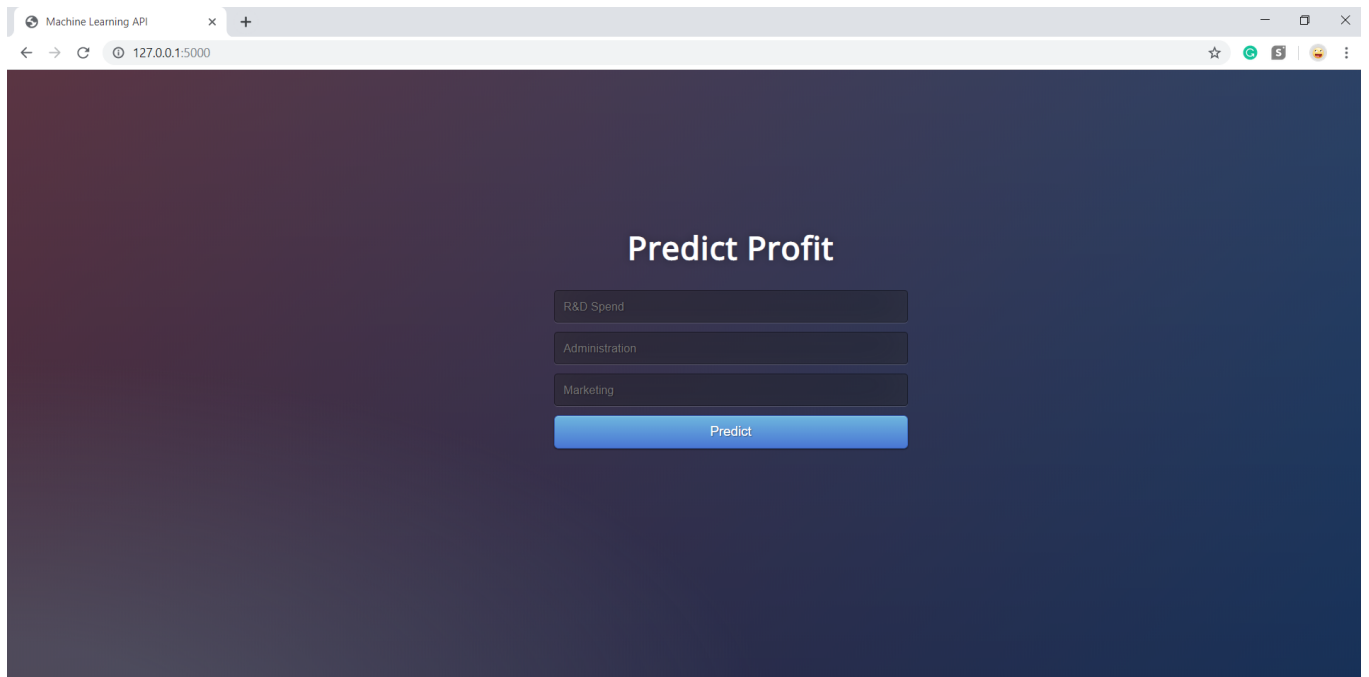
Run app.py in anaconda prompt

Select Anaconda Prompt (Anaconda3) - python app.py

```
(base) C:\Users\Chirag>cd C:\Users\Chirag\Desktop\Flask  
  
(base) C:\Users\Chirag\Desktop\Flask>python app.py  
* 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 stat  
* Debugger is active!  
* Debugger PIN: 196-854-268  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

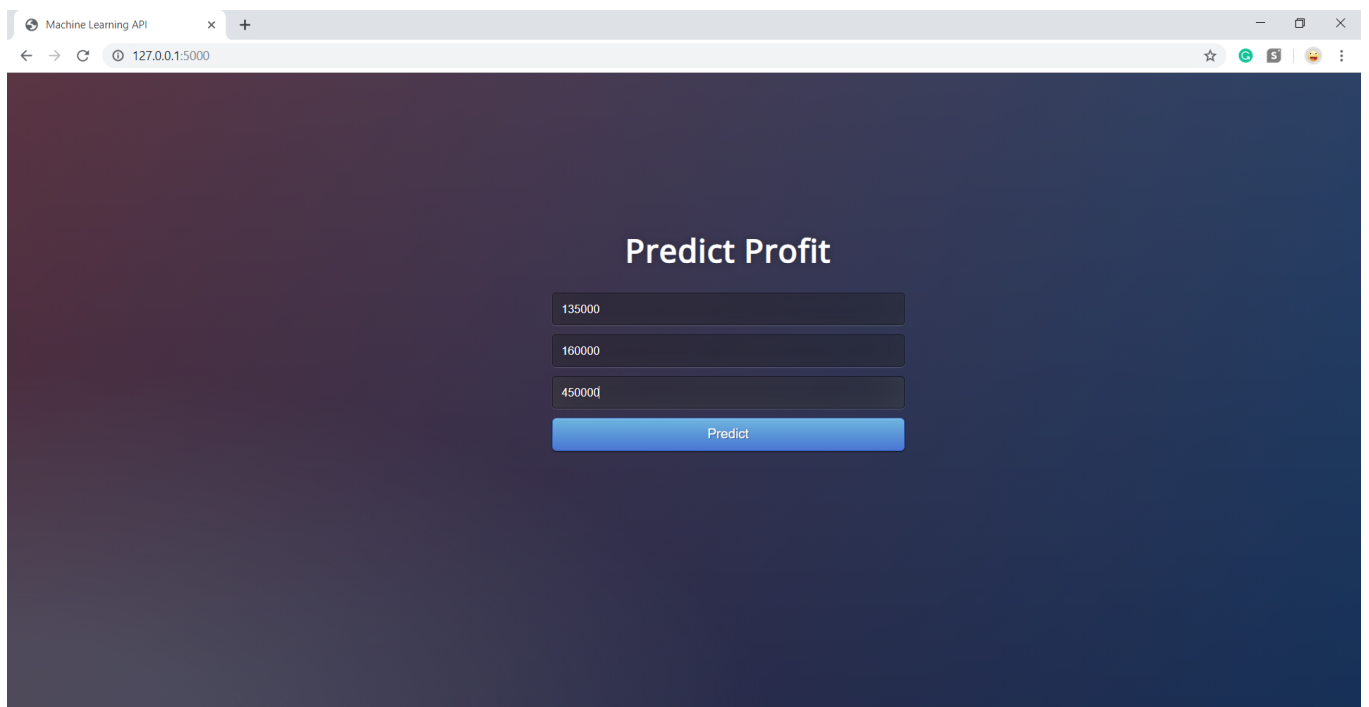
Copy the url <http://127.0.0.1:5000/> and paste it into your browser





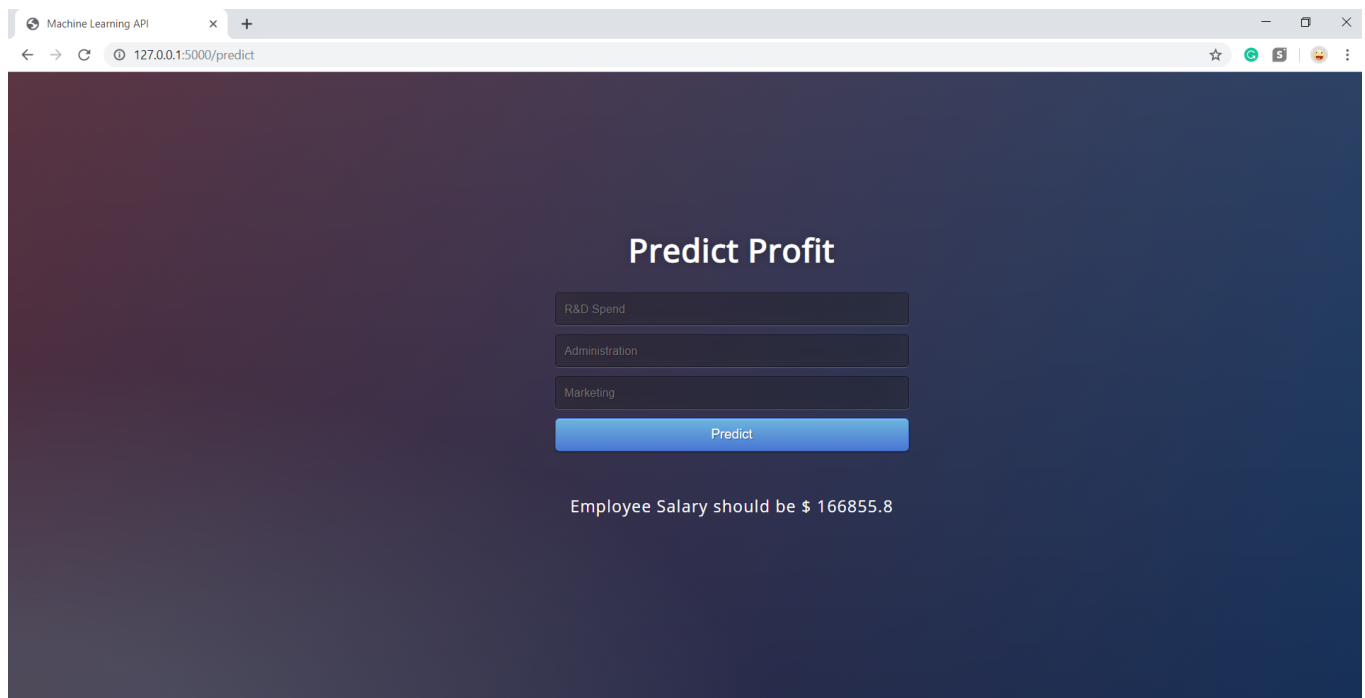
A screenshot of a web browser window titled "Machine Learning API" with the address bar showing "127.0.0.1:5000". The page has a dark blue gradient background and is titled "Predict Profit" in white text. Below the title, there are three empty input fields labeled "R&D Spend", "Administration", and "Marketing". At the bottom of these fields is a blue button labeled "Predict".

Input Data into different Categories



A screenshot of the same web browser window, but now the input fields contain numerical values: "135000" for "R&D Spend", "160000" for "Administration", and "450000" for "Marketing". The "Predict" button remains at the bottom.

Then Press Predict Button



The screenshot shows a web browser window with the title 'Machine Learning API'. The address bar displays '127.0.0.1:5000/predict'. The main content area has a dark blue gradient background. In the center, there is a white box containing the title 'Predict Profit'. Below the title are three text input fields labeled 'R&D Spend', 'Administration', and 'Marketing'. A blue button labeled 'Predict' is positioned below the input fields. At the bottom of the white box, the text 'Employee Salary should be \$ 166855.8' is displayed.

You can see that we Predicted our profit and notice that url of the page changed to <http://127.0.0.1:5000/predict> (<http://127.0.0.1:5000/predict>)

The Writer of this notebook is **Chirag Samal**. GitHub Link: https://github.com/chiragsamal/flask_tutorial (https://github.com/chiragsamal/flask_tutorial)

For any other help contact via LinkedIn: <https://www.linkedin.com/in/chiragsamal/> (<https://www.linkedin.com/in/chiragsamal/>)

In []: