Name: **Deployment on Flask**
Submission date: **27-June-2022**
Internship Batch: **LISUM10**
Data intake and model creation by: **Efe KARASIL**
Submitted to: **https://github.com/ekarasil/DataGlacier-Week4**

First of all, after choosing our data and deciding which feature to use to predict in the Machine Learning model, it was realized that encoding should be applied for a “Company” feature/column in the data. Since the effect of the taxi company on the estimated "Price Charged" is known, the "Company" column-feature is simply encoded. We converted the categorical value to ordinal value by giving 0 for Pink Cabs and 1 for Yellow Cabs.

```
df1 = pd.read_csv("Cab_Data.csv")
df1.head()
```

	Transaction ID	Date of Travel	Company	City	KM Travelled	Price Charged	Cost of Trip
0	10000011	42377	Pink Cab	ATLANTA GA	30.45	370.95	313.635
1	10000012	42375	Pink Cab	ATLANTA GA	28.62	358.52	334.854
2	10000013	42371	Pink Cab	ATLANTA GA	9.04	125.20	97.632
3	10000014	42376	Pink Cab	ATLANTA GA	33.17	377.40	351.602
4	10000015	42372	Pink Cab	ATLANTA GA	8.73	114.62	97.776

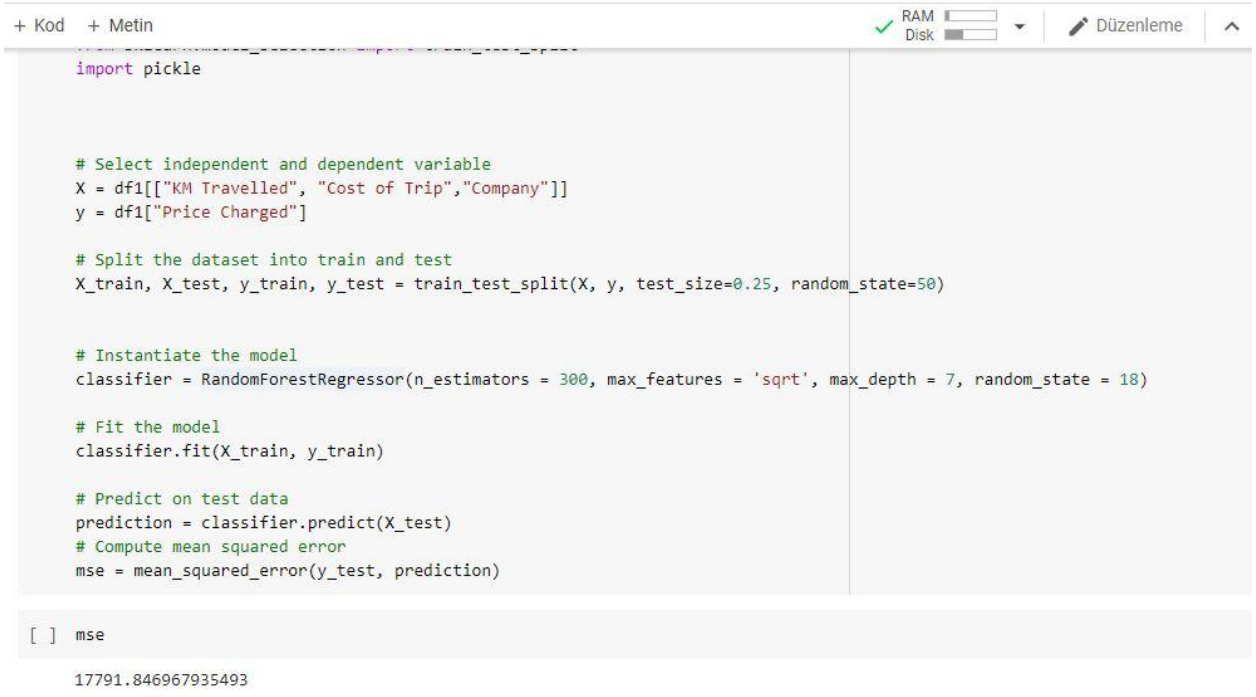
```
def encoder(x):
  if x=="Pink Cab":
    return(0)
  else:
    return(1)


df1["Company"]=df1["Company"].apply(encoder)
```

```
df1.tail()
```

	Transaction ID	Date of Travel	Company	City	KM Travelled	Price Charged	Cost of Trip
359387	10440101	43108	1	WASHINGTON DC	4.80	69.24	63.3600
359388	10440104	43104	1	WASHINGTON DC	8.40	113.75	106.8480
359389	10440105	43105	1	WASHINGTON DC	27.75	437.07	349.6500
359390	10440106	43105	1	WASHINGTON DC	8.80	146.19	114.0480
359391	10440107	43102	1	WASHINGTON DC	12.76	191.58	177.6192

Random Forest Regressor was used in the model and various parameters were tried to be selected by trial and error method in such a way that the Mean squared error between the estimation and the set to be tested would be minimized.



Afterwards, the code was transferred to the environment where all the files and the application would be kept, and finally the process was continued using the model and "pickle".

model.py > ...

```python
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import pickle

# Load the csv file
df = pd.read_csv("Cab_Data.csv")

def encoder(x):
  if x=="Pink Cab":
    return(0)
  else:
    return(1)

df["Company"]=df["Company"].apply(encoder)
print(df.head())

# Select independent and dependent variable
X = df[["KM Travelled", "Cost of Trip","Company"]]
y = df["Price Charged"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)

# Instantiate the model
classifier = RandomForestRegressor(n_estimators = 300, max_features = 'sqrt', max_depth = 7, random_state

# Fit the model
classifier.fit(X_train, y_train)

# Make pickle file of our model
pickle.dump(classifier, open("model.pkl", "wb"))
```

In the app python file, we start by reading the pickle file and using the index.html file as the home page.
Apart from that, since we send a POST request when the button is clicked on our index page, there is a code to process the data received accordingly and reflect the prediction from the machine learning model.

In index html, there are codes as we want the interface to look, and css styles are also used to make it look better.

index.html

templates > index.html > html > body > div.login > form > div.box > select#company

```
<!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='style.css') }}">
<script src="{{url_for('static', filename='javascriptfile.js')}}"></script>


</head>

<body>
 <div class="login">
  <h1>Price charged prediction for Cab Data</h1>

    <form action="{{ url_for('predict')}}"method="post">
      <div class="numericals">
        <input type="text" name="KM_Travelled" placeholder="KM Travelled" required="required" />
        <input type="text" name="Cost_of_Trip" placeholder="Cost of Trip" required="required" />
      </div>

        <label for="cars">Choose a company:</label>
        <div class="box">
            <select name="company" id="company" required>
              <option value=0>Pink Cab</option>
              <option value=1>Yellow Cab</option>
            </select>
          </div>
        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>

   <br>
   <br>
   {{ prediction_text }}

 </div>


</body>
</html>
```

In its latest form, our Flask application looks like the following, and with the sample values entered, we can also see the predicted values.