

# Using Classification Model in Flask App

Github: <https://github.com/DatascienceAuthority/Classification-Model-Flask>

## Train the model

```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

Path = "Carseats.csv"
data = pd.read_csv(Path)

data.loc[data.Sales > 4, 'Sale'] = 'Yes'
data.loc[data.Sales < 4, 'Sale'] = 'No'
data = data.loc[:,data.columns != 'Sales']

data['Sale'], sales_index = pd.factorize(data['Sale'])
sales_index
print(data['Sale'].unique())

data['ShelveLoc'], shelveloc_index = pd.factorize(data['ShelveLoc'])
shelveloc_index
data['Urban'], urban_index= pd.factorize(data['Urban'])
urban_index
data['US'], us_index = pd.factorize(data['US'])
us_index

data.info()

X = data.loc[:,data.columns != 'Sale']
Y = data.Sale

feature_names = X.columns

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)

##### Random Forest #####
model = RandomForestClassifier(n_estimators=100)

model.fit(X_train,y_train)

predicted = model.predict(X_test)

print(metrics.confusion_matrix(y_test, predicted))
```

```
print(metrics.classification_report(y_test, predicted))

RF_accuracy = metrics.accuracy_score(y_test, predicted)
print('RF_Accuracy: {:.2f}'.format(RF_accuracy))
```

## Save the model

Save the trained random forest model as a pickle file

```
# save the model to disk
import pickle
filename = 'model/model.pkl'
pickle.dump(model, open(filename, 'wb'))

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
```

## Flask Application

### Create a Virtual Environment (Optional)

From the command line, go to a desired directory and create a virtual environment for an isolated environment for the project and activate it ( C:\> python -m venv <any name>)

```
C:\flask-app>python -m venv rfvenv
C:\flask-app>rfvenv\Scripts\activate
(rfvenv) C:\flask-app>
```

### requirements.txt

```
gunicorn==20.0.4
Click==7.0
dominate==2.4.0
Flask==1.1.1
Flask-Bootstrap==3.3.7.1
Flask-WTF==0.14.3
itsdangerous==1.1.0
Jinja2==2.11.1
```

```
joblib==0.14.1
MarkupSafe==1.1.1
numpy==1.18.1
pandas==1.0.1
python-dateutil==2.8.1
pytz==2019.3
scikit-learn==0.22.1
scipy==1.4.1
six==1.14.0
sklearn==0.0
visitor==0.1.3
Werkzeug==1.0.0
WTForms==2.2.1
```

Install the above required packages from requirements.txt using pip

```
(rfvenv) C:\flask-app> pip install -r requirements.txt
```

## app.py

```
from flask import Flask, render_template
import pickle
from flask_bootstrap import Bootstrap
from flask_wtf import FlaskForm
from wtforms import SubmitField, validators, FloatField, SelectField
import pandas as pd
#pd.set_option('display.max_colwidth', -1)

# code which helps initialize our server
app = Flask(__name__)
app.config['SECRET_KEY'] = 'any secret key'

bootstrap = Bootstrap(app)

sales_index = ['Yes', 'No']
shelveLoc_index = ['Bad', 'Good', 'Medium']
urban_index = ['Yes', 'No']
us_index = ['Yes', 'No']

# load the model from disk
model = pickle.load(open('model/model.pkl', 'rb'))
feature_names= ['CompPrice', 'Income', 'Advertising', 'Population', 'Price', 'ShelveLoc',
'Age', 'Education', 'Urban', 'US']

class FeaturesForm(FlaskForm):
    CompPrice = FloatField('CompPrice', [validators.DataRequired(),
validators.NumberRange(min=0, max=1000)])
```

```

    Income = FloatField('Income', [validators.DataRequired(), validators.NumberRange(min=0,
max=10000)])
    Advertising = FloatField('Advertising', [validators.DataRequired(),
validators.NumberRange(min=0, max=1000)])
    Population = FloatField('Population', [validators.DataRequired(),
validators.NumberRange(min=0, max=10000)])
    Price = FloatField('Price', [validators.DataRequired(), validators.NumberRange(min=0,
max=10000)])
    ShelfLoc = SelectField('ShelveLoc', [validators.DataRequired()], choices=[('Bad', 'Bad'),
('Good', 'Good'), ('Medium', 'Medium')])
    Age = FloatField('Age', [validators.DataRequired(), validators.NumberRange(min=1,
max=100)])
    Education = FloatField('Education', [validators.DataRequired(),
validators.NumberRange(min=0, max=30)])
    Urban = SelectField('Urban', [validators.DataRequired()], choices=[('Yes', 'Yes'),
('No', 'No')])
    US = SelectField('US', [validators.DataRequired()], choices=[('Yes', 'Yes'), ('No', 'No')])
    submit = SubmitField('Submit')

@app.route('/', methods=['GET', 'POST'])
def predict():
    form = FeaturesForm()
    if form.validate_on_submit():
        CompPrice = form.CompPrice.data
        Income = form.Income.data
        Advertising = form.Advertising.data
        Population = form.Population.data
        Price = form.Price.data
        ShelfLoc = form.ShelveLoc.data
        ShelfLoc_val = shelveloc_index.index(ShelveLoc)
        Age = form.Age.data
        Education = form.Education.data
        Urban = form.Urban.data
        Urban_val = urban_index.index(Urban)
        US = form.US.data
        US_val = us_index.index(US)
        features = [CompPrice, Income, Advertising, Population, Price, ShelfLoc, Age,
Education, Urban, US]
        features_val = [CompPrice, Income, Advertising, Population, Price, ShelfLoc_val, Age,
Education, Urban_val, US_val]
        df = pd.DataFrame([features], columns=feature_names)
        prediction = model.predict([features_val])
        result = sales_index[prediction[0]]
        return render_template('result.html', df = df, result=result)

    return render_template('index.html', form=form)

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

```

Create a folder named model and place the saved random forest model 'model.pkl' in it

Instead of creating the html files from scratch, we can use the flask jinja template engine and flask\_bootstrap to create UI easily. Create a folder named templates and create the html files base.html, index.html, result.html

### base.html

```
{% extends 'bootstrap/base.html' %}

{% block title %}
    Child Car Seats
{% endblock %}

{% block navbar %}
    <nav class="navbar navbar-default">
        <div class="container">
            <div class="navbar-header">
                <ul class="nav navbar-nav">
                    <li class="nav-item active"><a href="{{ url_for('predict') }}">Home
</a></li>
                </ul>
            </div>
        </div>
    </nav>
{% endblock %}
```

### index.html

```
{% extends "base.html" %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block content %}

    {% if form %}
        <div class="col-md-4 col-lg-offset-4">
            {{ wtf.quick_form(form) }}
        </div>
    {% endif %}

{% endblock %}
```

### result.html

```
{% extends "base.html" %}

{% block content %}
```

```

    {{ df.to_html(render_links=True, index=False, classes=["text-center", "table", "table-
bordered", "table-striped", "table-hover"]).replace('<th>', '<th style = "text-align:
center">')|safe }}

<div class="col-md-4 col-lg-offset-2">
    <h2>Is the Sales High?</h2>
    <h1 style="color:green;">{{ result }}</h1>
</div>

{% endblock %}

```

Run the application from command prompt using 'flask run' or 'python app.py'

```
(rfvenv) C:\flask-app>flask run
```

## Output

```

(rfvenv) C:\flask-app>flask run
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```

The app can now be accessed at <http://127.0.0.1:5000/>

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000'. The page has a 'Home' button in the top left. The main content area contains a form with the following fields:

- CompPrice: A text input field with a small icon on the right.
- Income: A text input field.
- Advertising: A text input field.
- Population: A text input field.
- Price: A text input field.
- ShelveLoc: A dropdown menu with 'Bad' selected.
- Age: A text input field.
- Education: A text input field.
- Urban: A text input field.

A 'Window Title' button is visible on the right side of the form.

## Deploying Flask App to Heroku

Install git : <https://git-scm.com/downloads>

Install heroku CLI: <https://devcenter.heroku.com/articles/heroku-cli>

In the root directory of the flask app, create a file named 'Procfile' and add following text

```
web: gunicorn app:app
```

Heroku uses the above file to create a gunicorn web server interface and app:<app-name> indicates the app name. Here our app name is 'app' ( app = Flask(\_\_name\_\_) )

Create a heroku account - <https://www.heroku.com/>

From the command prompt, go to the directory where the flask app is located

Initiate git

```
C:\flask-app>git init .
```

Add all the required files and folders to git

```
C:\flask-app>git add app.py Procfile requirements.txt model/* templates/*
```

(Optional) If an error "Tell me who you are" is displayed while executing git commands, use the following commands

```
git config user.name "someone"  
git config user.email "someone@someplace.com"
```

Commit git with any message

```
C:\flask-app>git commit -m "first commit"
```

Provide heroku account credentials

```
C:\flask-app>heroku login -i
```

Login to heroku in a browser and “Create New App” with any name {your-app-name}. Add the app name you created on heroku website from the command prompt

```
C:\flask-app>heroku git:remote -a {your-app-name}
```

Deploy the app to heroku

```
C:\flask-app>git push heroku master
```

Now, a message will be shown saying the app is successfully deployed and can be accessed at the URL <https://{your-app-name}.herokuapp.com/>