Ford Car Price Prediction

Done By: Toh Kien Yu (2222291)

Table Of Content

Dataset:
https://www.kaggle.com/datasets/adhurimquku/ford-car-price-prediction

Scrum Board

Train an AI model for result prediction

#1

Entering Prediction Parameter through UI

#2

Storing of Prediction

#3

Responsive Web Application Integration

#4

User Authentication and Login Credentials

#5

Unit Testing on Web Application

#7

Deployment to Render

#6

Scrum Board

Train an AI model for result prediction

#1

As a user I want to provide a set of data of my past result so as to train an AI model that can predict result in the future

- [ ] EDA of user data
- [ ] Data Engineering
- [ ] Training and Testing
- [ ] Exporting the model

Entering Prediction Parameter through UI

#2

As a user, I want to enter values of the ford car model, year, type of transmission, mileage, tax, miles per gallon and engine size to submit the parameters for prediction

- [ ] Wire Frame for UI
- [ ] UI
- [ ] Backend Processing

Scrum Board

Storing of Prediction

#3

As a user I want be able to see the history of all the past prediction in a prediction table so as to know what values has been submitted before.

User Authentication and Login Credentials

#5

As a user I want a secure system for authentication so as to access the web application.

- [ ] Login Page
- [ ] Registration Page

Responsive Web Application Integration

#4

As a user I want the trained AI model to be integrated into a web applications that is user-friendly and intuitive for easy access.

Scrum Board

Unit Testing on Web Application

#7

As a developer I want to do unit testing and automate test suites using Pytest so as to ensure reliability of web application

Deployment to Render

#6

As a developer I want to deploy the web application to Render so as to be accessible through internet access

Branches

```
kieny@kien MINGW64 ~/Documents/Y2S2 - DAAA/DOAA Tester/doaadraft (main)
$ git branch
  MLAppHistoryTable_branch
  MLAppLogin_branch
  MLAppNewestPyTest_branch
  MLApp_branch
  fordApplication
  improveUI
* main
```

Branch 1: fordApplication
- Training regression model for web application

Branch 2: MLAppDB_branch
- Integrate regression model to web application
- Entering prediction parameter through UI

Branch 3: MLAppHistoryTable_branch
- Storing of prediction through a history table

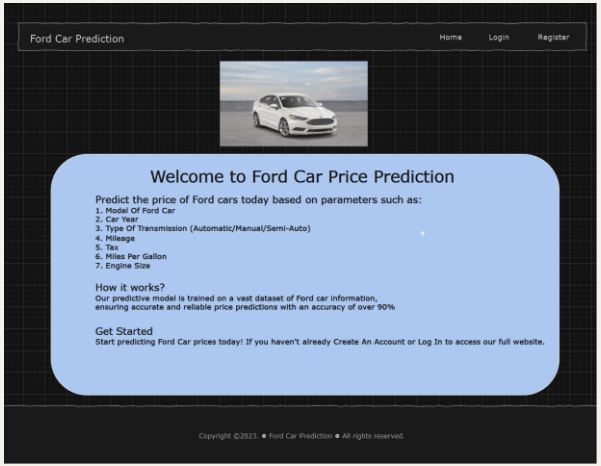
Branch 4: MLAppLogin_branch
- User Authentication and Login Credentials

Branch 5: improveUI
- Responsive Web Application

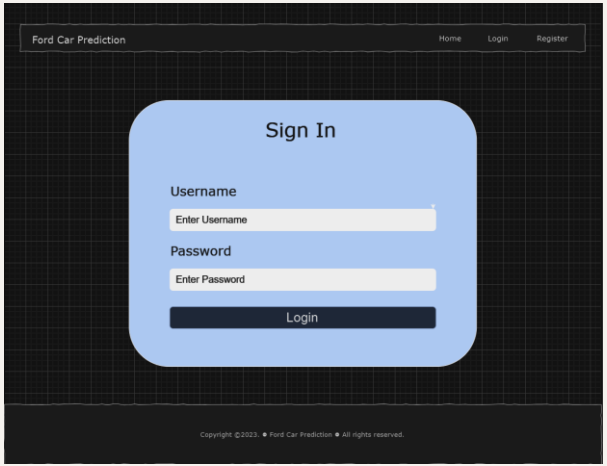
Branch 6 MLAppNewestPyTest_branch
- Unit Testing on Web Application

WireFrame

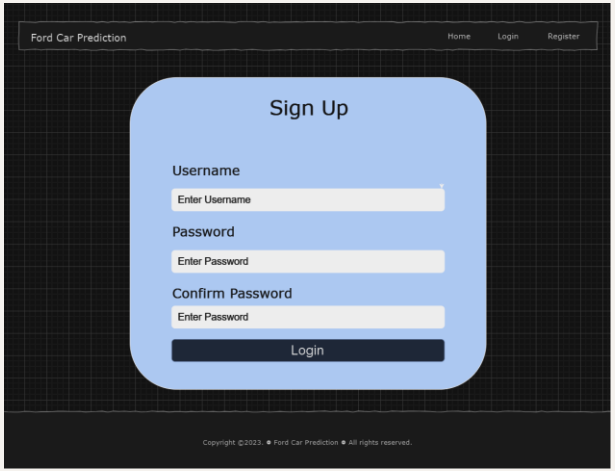
index.html



login.html

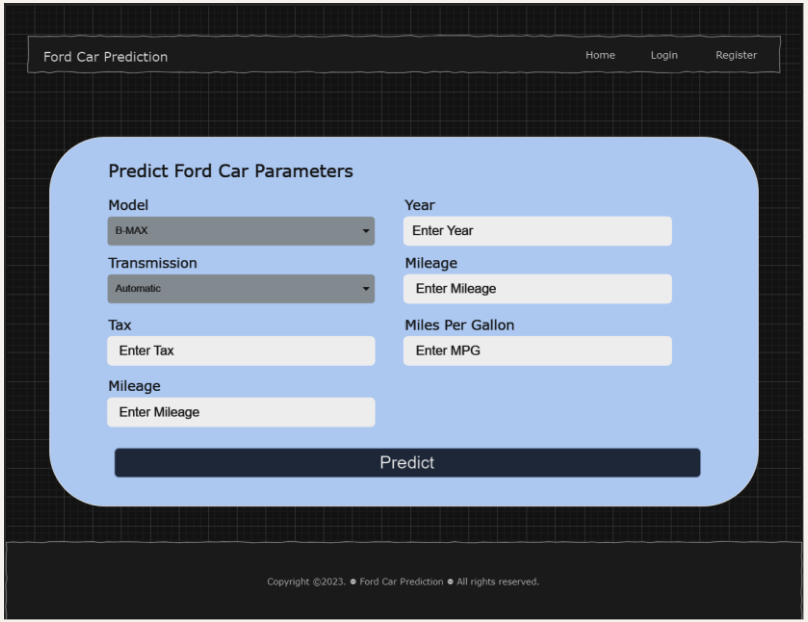


register.html

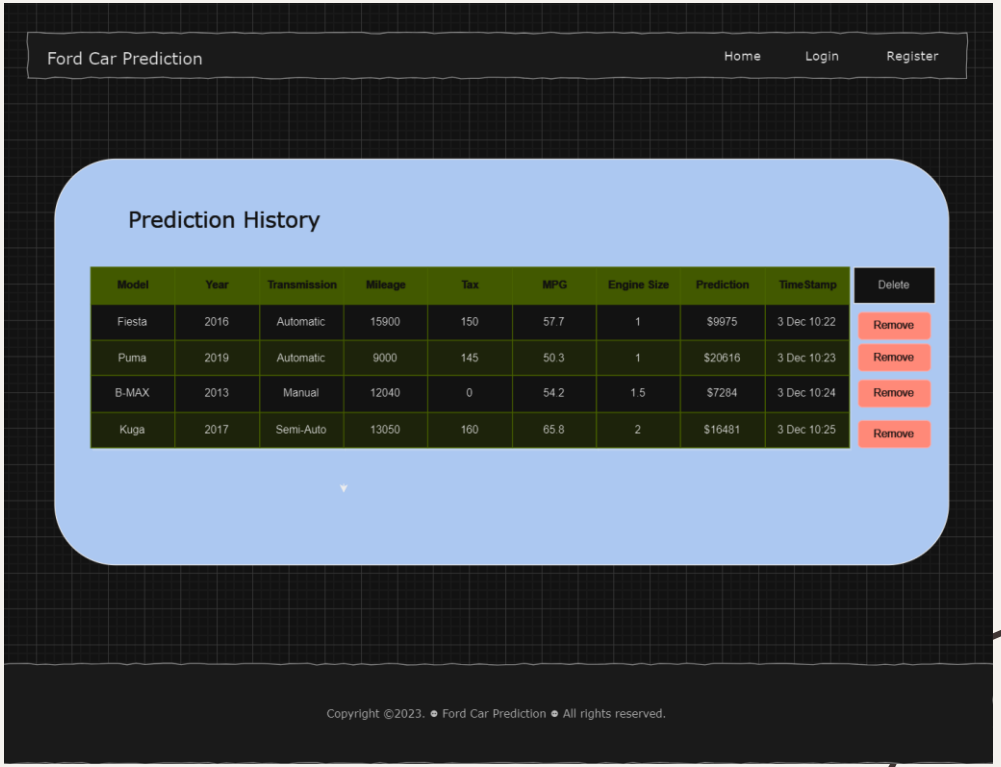


WireFrame

predictionForm.html



history.html



Branch 1 (fordApplication)

Nature of Dataset

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	Fiesta	2017	12000	Automatic	15944	Petrol	150	57.7	1.0
1	Focus	2018	14000	Manual	9083	Petrol	150	57.7	1.0
2	Focus	2017	13000	Manual	12456	Petrol	150	57.7	1.0

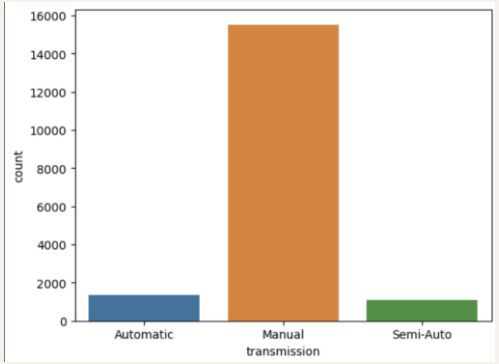
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17966 entries, 0 to 17965
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   model         17966 non-null  object
 1   year          17966 non-null  int64
 2   price         17966 non-null  int64
 3   transmission  17966 non-null  object
 4   mileage       17966 non-null  int64
 5   fuelType      17966 non-null  object
 6   tax           17966 non-null  int64
 7   mpg           17966 non-null  float64
 8   engineSize    17966 non-null  float64
dtypes: float64(2), int64(4), object(3)
memory usage: 1.2+ MB
```

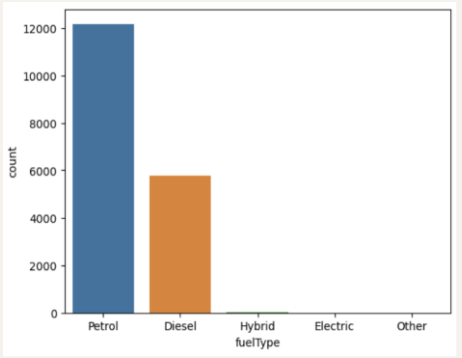
```
df.isnull().sum() # No Null Values
```

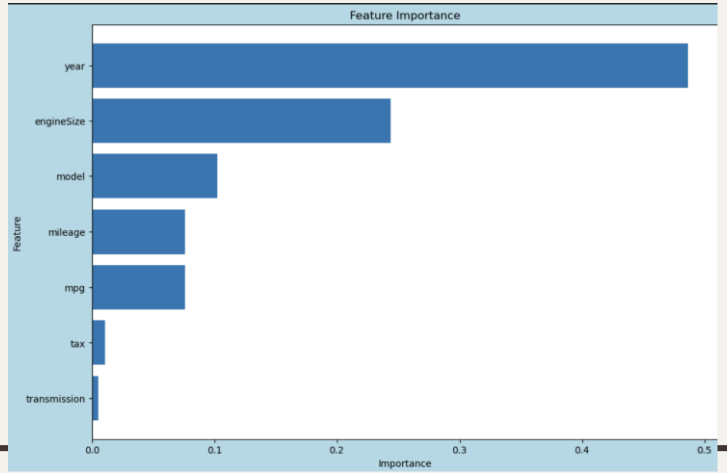
```
model           0
year            0
price           0
transmission    0
mileage         0
fuelType        0
tax             0
mpg             0
engineSize      0
dtype: int64
```

```
Database Shape
(17966, 9)
```

Exploratory Data Analysis







Branch 1 (fordApplication)

Modelling

```
clfs =[('Linear_Regression',LinearRegression()),
       ('Random_Forest',RandomForestRegressor()),
       ('Gradient_Boosting',GradientBoostingRegressor()),
       ('HGB',HistGradientBoostingRegressor()),
       ('Decision_Tree', DecisionTreeRegressor()),]

for name,clf in clfs:
    pipeline = Pipeline(steps = [('scaler',MinMaxScaler()),('regressor',clf)])
    pipeline.fit(X_train,y_train)
    y_pred = pipeline.predict(X_test)
    r2 = r2_score(y_test, y_pred)
```

Results

```
The R Square score for Linear_Regression: 0.698
The R Square score for Random_Forest: 0.925
The R Square score for Gradient_Boosting: 0.918
The R Square score for HGB: 0.938
The R Square score for Decision_Tree: 0.895
```

```
The cv score for Linear_Regression: 0.735
The cv score for Random_Forest: 0.927
The cv score for Gradient_Boosting: 0.915
The cv score for HGB: 0.934
The cv score for Decision_Tree: 0.880
```

	Model	MSE	RMSE	MAE	MAPE	R2
0	Linear_Regression	6.904514e+06	2627.644244	1745.716159	0.168050	0.697972
1	Random_Forest	1.705234e+06	1305.846103	876.869934	0.073595	0.925407
2	Gradient_Boosting	1.873921e+06	1368.912445	964.613858	0.081761	0.918028
3	HGB	1.418014e+06	1190.803800	825.727133	0.069864	0.937971
4	Decision_Tree	2.535460e+06	1592.312752	1068.504127	0.090163	0.889090

Branch 1 (fordApplication)

Hyper-parameter tuning

```
#Hypertuning Random Forest
from sklearn.model_selection import GridSearchCV, KFold
randomForestReg = RandomForestRegressor()
parameters = {'n_estimators': [50,100,150],'max_depth':[None,3,5],'max_features':[None,'sqrt','log2']}
grid = GridSearchCV(randomForestReg, param_grid=parameters, cv=3)
best_clf = grid.fit(X_train,y_train)
print("score = %3.3f" %(grid.score(X_test,y_test)))
print(grid.best_params_)
```

```
score = 0.932
{'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 150}
```

```
#Hypertuning HistGradientBoosting
from sklearn.model_selection import GridSearchCV, KFold
hgb = HistGradientBoostingRegressor()
parameters = {'learning_rate': [0.01,0.1,0.2],'max_depth':[None,3,5],'max_leaf_nodes':[None,5,10]}
grid = GridSearchCV(hgb, param_grid=parameters, cv=3)
best_clf = grid.fit(X_train,y_train)
print("score = %3.3f" %(grid.score(X_test,y_test)))
print(grid.best_params_)
```

```
score = 0.938
{'learning_rate': 0.2, 'max_depth': 5, 'max_leaf_nodes': None}
```

I Picked out 2 of the most accurate model to hyperparameter tune

1. Random Forest
2. HistGradientBoosting

Now, I will hyperparameter tune the model to further improve the accuracy by customizing the parameters for the algorithm.

1. Define a param grid for each model, and parameters I would like to test
2. GridSearchCV will loop through all the different types of combinations
3. Upon looping through all the combinations, it will provide me with the best parameter combinations and highest R Square Score obtained

Observations

- Random Forest R Square increased from 0.925 to 0.932
- HistGradientBoosting R Square remained stagnent.

Branch 1 (fordApplication)

We will choose HistGradientBoosting as our final model.

Final Model

```
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
HGB_model = HistGradientBoostingRegressor(learning_rate=0.2,max_depth=5,max_leaf_nodes=None)
scaler = MinMaxScaler()
X_trainScaled = scaler.fit_transform(X_train)
X_testScaled = scaler.transform(X_test)

HGB_model.fit(X_trainScaled,y_train)
expected = y_test
predicted = HGB_model.predict(X_testScaled)
r2 = r2_score(expected, predicted)
print(f'The R Square score: {r2:.3f}')
```

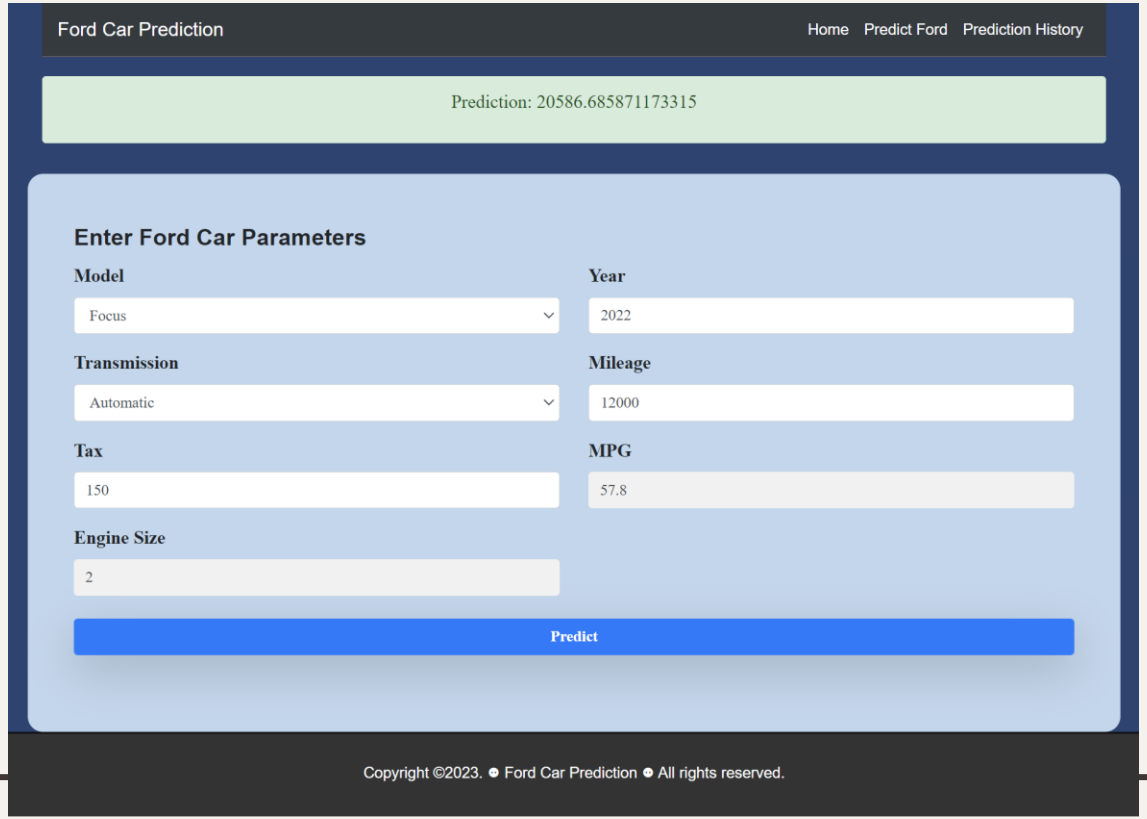
```
The R Square score: 0.938
```

Export the model

```
import pickle
with open('../application/static/ford_Model.pkl', 'wb') as f:
    pickle.dump(HGB_model, f)
```

Branch 2 (MLAppDB_branch)

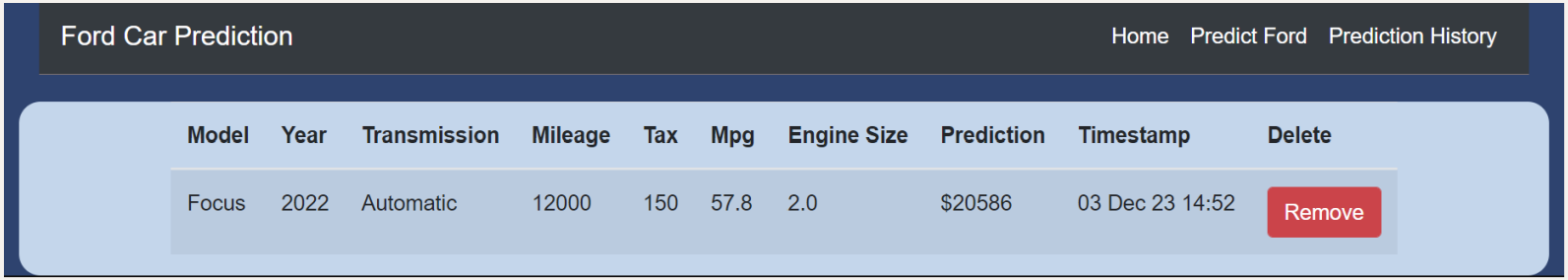
- Integrate regression model to web application
- Entering prediction parameter through UI



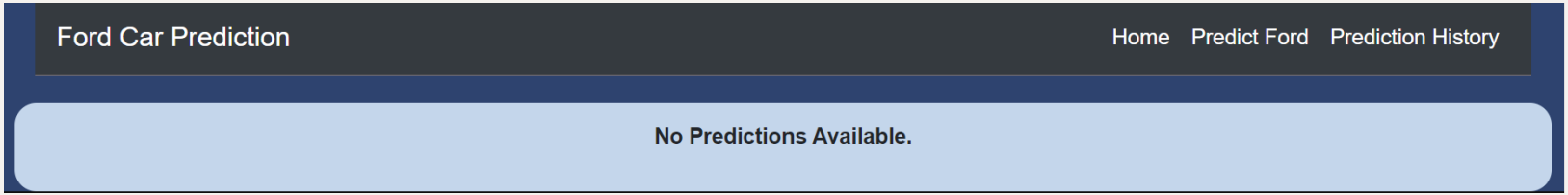
Branch 3 (MLAppHistoryTable_branch)

- Storing of prediction through a history table

Predictions are stored in the prediction history table



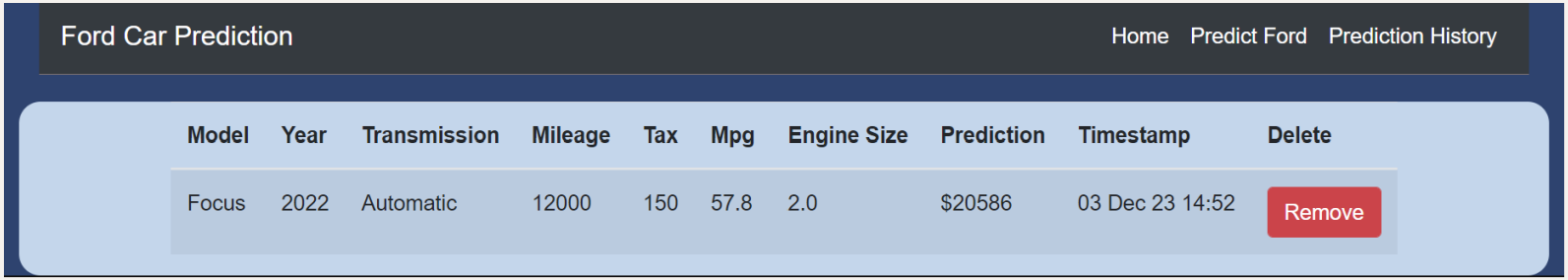
If there are no predictions made, user will see no predictions available



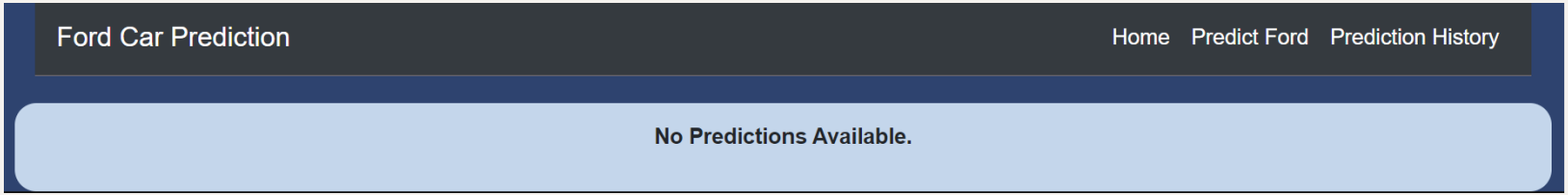
Branch 3 (MLAppHistoryTable_branch)

- Storing of prediction through a history table

Predictions are stored in the prediction history table



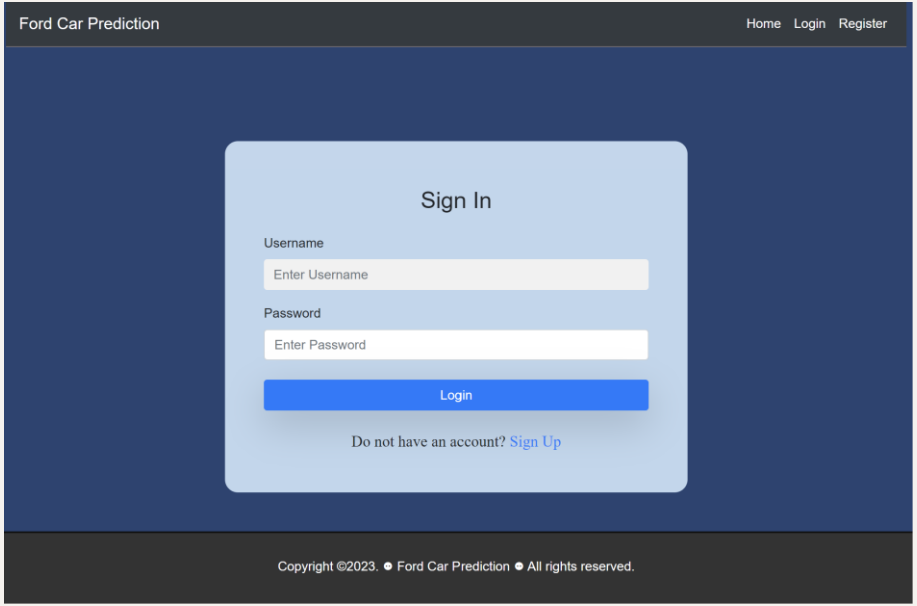
If there are no predictions made, user will see no predictions available



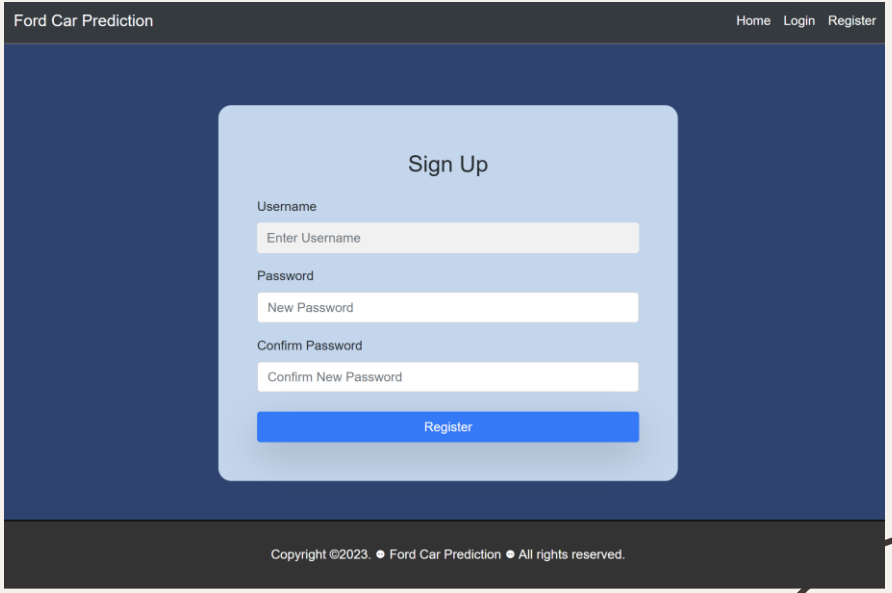
Branch 4 (MLAppLogin_branch)

- User Authentication and Login Credentials

login.html



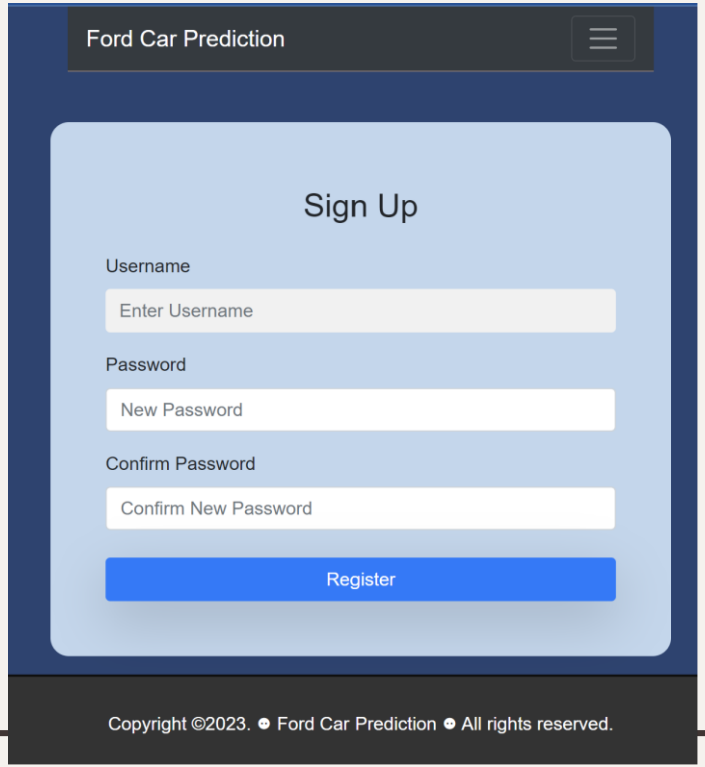
register.html

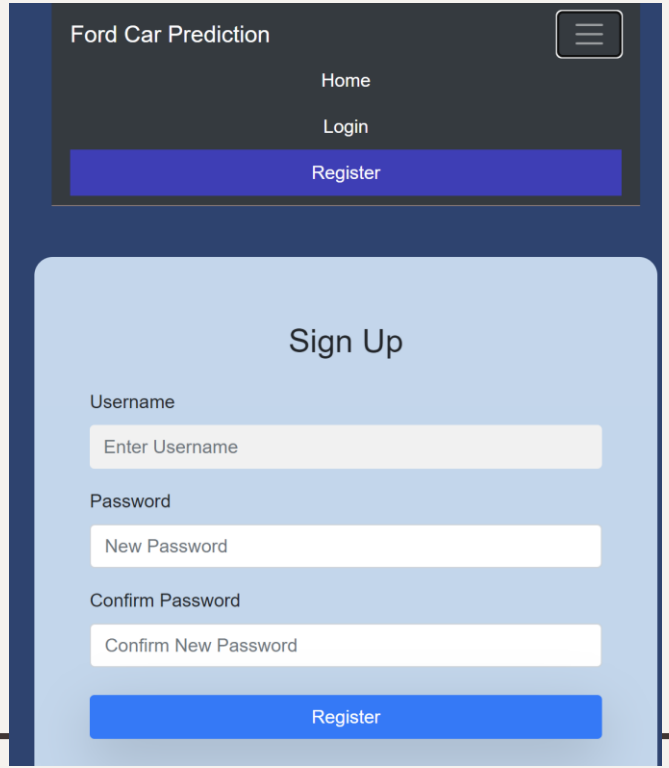


Branch 5 (improveUI)

- Responsive Web Application (Catered for smaller and larger devices)

For example: Dropdown menu was added for smaller devices and hover animations





Branch 6 (MLAppPyTest_branch)

• Unit Testing on Web Application

Expected Failure and Consistency testing

Registration API Testing
- Passes when unique username is inserted into the database
- Expected failure when username is not unique

```python
@pytest.mark.parametrize("entrylist",[
    ['newUser10','123'],
    ['duplicateUser','12345'],
    ['duplicateUser','12345']
])

def test_addUser(client,entrylist,capsys):
    with capsys.disabled():
        #prepare the data into a dictionary
        data1 = { 'username': entrylist[0],
        'password' : entrylist[1],
        }
        #use client object to post
        #data is converted to json
        #posting content is specified
        try:
            response = client.post('/api/register',
                                   data=json.dumps(data1),
                                   content_type="application/json",)
            #check the outcome of the action
            assert response.status_code == 200
            assert response.headers["Content-Type"] == "application/json"
            response_body = json.loads(response.get_data(as_text=True))
            assert response_body["id"] is not None
            print(f"Added Entry: {response_body['id']}")
        except TypeError as e:
            print("Username already exists. Please choose another username.")
            pytest.xfail("Username must be unique.")
```

```python
@app.route("/api/register", methods=['POST'])
def api_register():
    #retrieve the json file posted from client
    data = request.get_json()
    #retrieve each field from the data
    username = data['username']
    password = data['password']
    print(data)
    #create an Entry object store all data for db action
    new_entry = User( username=username, password=password)
    #invoke the add entry function to add entry
    result = add_entry(new_entry)
    #return the result of the db action
    return jsonify({'id':result})
```

```
tests/test_application.py::test_addUser[entrylist0] {'username': 'newUser15', 'password': '123'}
Added Entry: 20
PASSED                                                                   [ 33%]
tests/test_application.py::test_addUser[entrylist1] {'username': 'duplicatedUser', 'password': '12345'}
Added Entry: 21
PASSED                                                                   [ 66%]
tests/test_application.py::test_addUser[entrylist2] {'username': 'duplicatedUser', 'password': '12345'}
Username already exists. Please choose another username.
XFAIL (Username must be unique.)                                         [100%]

========================================================== 2 passed, 1 xfailed in 0.55s ==========
```

Branch 6 (MLAppPyTest_branch)

- Unit Testing on Web Application

<u>Consistency Testing</u>

Testing adding prediction API in prediction history

```python
@pytest.mark.parametrize("entrylist",[
 ['Puma', 2017, 'Automatic', 20000, 150,57.7,1,26000,1]
])
def test_deletePredictionAPI(client, entrylist, capsys):
    with capsys.disabled():
        data1 = {
            'model': entrylist[0],
            'year': entrylist[1],
            'transmission': entrylist[2],
            'mileage': entrylist[3],
            'tax': entrylist[4],
            'mpg': entrylist[5],
            'engineSize': entrylist[6],
            'prediction':entrylist[7],
            'userID':entrylist[8]}

        response = client.post('/api/predict',data=json.dumps(data1),content_type="application/json")
        response_body = json.loads(response.get_data(as_text=True))

        assert response_body["id"]
        id = response_body["id"]
        print(f"Added Entry: {response_body['id']}")

        response2 =  client.get(f'/api/delete/{id}')
        ret = json.loads(response2.get_data(as_text=True))
        assert response2.status_code == 200
        assert response2.headers["Content-Type"] == "application/json"
        response2_body = json.loads(response2.get_data(as_text=True))
        assert response2_body["result"] == "ok"
        print(f"Deleted Entry: {response_body['id']}")
```

```python
#API: add predict entry
@app.route("/api/predict", methods=['POST'])
def api_addPredict():
    #retrieve the json file posted from client
    data = request.get_json()
    #retrieve each field from the data
    model = data['model']
    year = data['year']
    transmission = data['transmission']
    mileage = data['mileage']
    tax = data['tax']
    mpg = data['mpg']
    engineSize = data['engineSize']
    prediction = data['prediction']
    userID = data['userID']
    print(data)
    #create an Entry object store all data for db action
    new_entry = Entry(model=model, year=year,transmission=transmission,mileage=mileage,
                    tax=tax,mpg=mpg,engineSize=engineSize,prediction=prediction,predicted_on=datetime.utcnow(),userID=userID)
    #invoke the add entry function to add entry
    result = add_entry(new_entry)
    #return the result of the db action
    return jsonify({'id':result})
```

```python
#API delete entry
@app.route("/api/delete/<id>", methods=['GET'])
def api_delete(id):
    entry = remove_entry(int(id))
    return jsonify({'result':'ok'})
```

```
tests/test_application.py::test_deletePredictionAPI[entrylist0] {'model': 'Puma', 'year': 2017, 'transmission': 'Automatic', 'mileage': 20000, 'tax': 150, 'mpg': 57.7, 'engineSize': 1, 'prediction': 2600
0, 'userID': 1}
Added Entry: 46
Deleted Entry: 46
PASSED                                                                                   [100%]

=============================== 1 passed in 0.11s ===============================
```

Branch 6 (MLAppPyTest_branch)

- Unit Testing on Web Application

Validity Testing

Testing to ensure Entry class can be instantiated with the given attributes. The test is deemed successful when all assertion are passed.

```python
# Validity Testing
@pytest.mark.parametrize("entrylist",[
 ['Puma', 2017, 'Automatic', 20000, 150,57,1,26000,1],
 ['B-MAX', 2014.2, 'Manual', 20000.0, 145.3,58.7,1,23000,1]
])
#3: Write the test function pass in the arguments
def test_EntryClass(entrylist,capsys):
    with capsys.disabled():
        print(entrylist)
        now = datetime.datetime.utcnow()
        new_entry = Entry( model= entrylist[0],
                           year = entrylist[1],
                           transmission= entrylist[2],
                           mileage = entrylist[3],
                           tax = entrylist[4],
                           mpg = entrylist[5],
                           engineSize = entrylist[6],
                           prediction = entrylist[7],
                           userID = entrylist[8],
                           predicted_on= now)

        assert new_entry.model == entrylist[0]
        assert new_entry.year == entrylist[1]
        assert new_entry.transmission == entrylist[2]
        assert new_entry.mileage == entrylist[3]
        assert new_entry.tax == entrylist[4]
        assert new_entry.mpg == entrylist[5]
        assert new_entry.engineSize == entrylist[6]
        assert new_entry.prediction == entrylist[7]
        assert new_entry.userID == entrylist[8]
        assert new_entry.predicted_on == now
```

```
tests/test_application.py::test_EntryClass[entrylist0] ['Puma', 2017, 'Automatic', 20000, 150, 57, 1, 26000, 1]
PASSED                                                                                                    [ 50%]
tests/test_application.py::test_EntryClass[entrylist1] ['B-MAX', 2014.2, 'Manual', 20000.0, 145.3, 58.7, 1, 23000, 1]
PASSED                                                                                                    [100%]

============================================================================== 2 passed in 0.07s ===============================================
(env)
```

Branch 6 (MLAppPyTest_branch)

- Unit Testing on Web Application

Expected Failure Testing

Testing to ensure that the form is able to handle negative values.

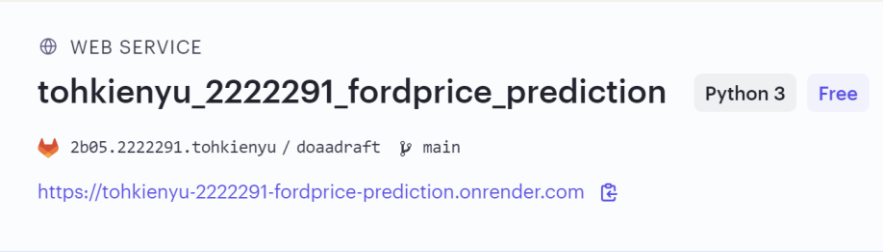
```python
#4: Expected Failure Testing
@pytest.mark.xfail(reason="arguments < 0")
@pytest.mark.parametrize("entrylist",[
 ['Puma', -2017, 'Automatic', 20000, 150,57,1,26000,1],
 ['B-MAX', 2014.2, 'Manual', -20000.0, 145.3,58.7,1,-23000,-1],
 ['Galaxy', 0, 'Automatic', -30000, -150,57,1,26000,1],
 ['Puma', 2014.5, 'Manual', 20000.0, 145.3,58.7,-1,23000,1],

])
def test_EntryValidation(entrylist, capsys):
    test_EntryClass(entrylist, capsys)
```

```
tests/test_application.py::test_EntryValidation[entrylist0] ['Puma', -2017, 'Automatic', 20000, 150, 57, 1, 26000, 1]
XPASS (arguments < 0)                                                                    [ 50%]
tests/test_application.py::test_EntryValidation[entrylist1] ['B-MAX', 2014.2, 'Manual', -20000.0, 145.3, 58.7, 1, -23000, -1]
XPASS (arguments < 0)                                                                    [ 66%]
tests/test_application.py::test_EntryValidation[entrylist2] ['Galaxy', 0, 'Automatic', -30000, -150, 57, 1, 26000, 1]
XPASS (arguments < 0)                                                                    [ 83%]
tests/test_application.py::test_EntryValidation[entrylist3] ['Puma', 2014.5, 'Manual', 20000.0, 145.3, 58.7, -1, 23000, 1]
XPASS (arguments < 0)                                                                    [100%]
```

Internet Deployment

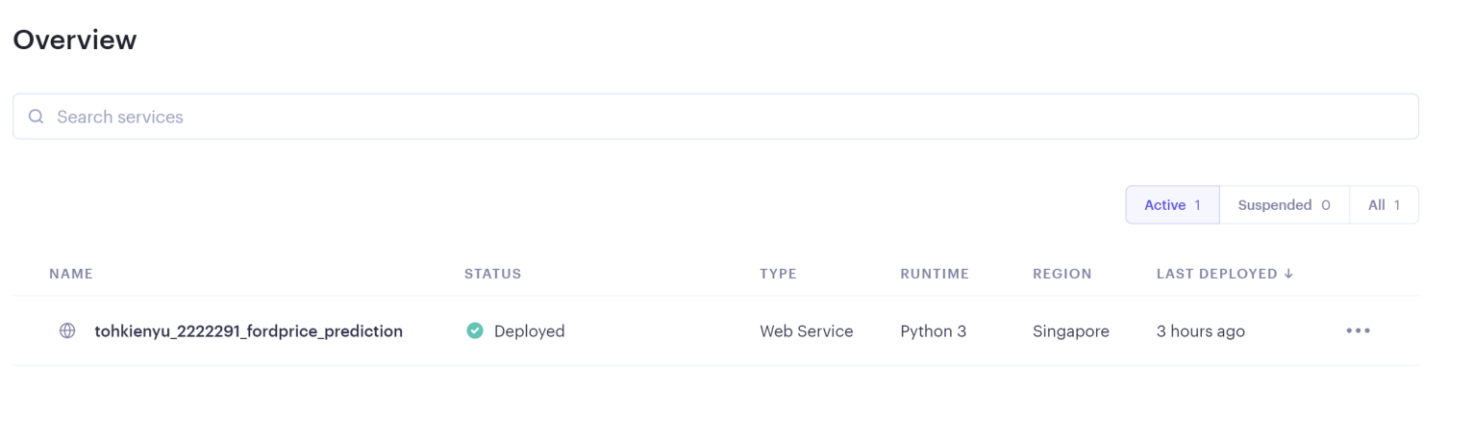
Render was used to deploy the web application



app.py

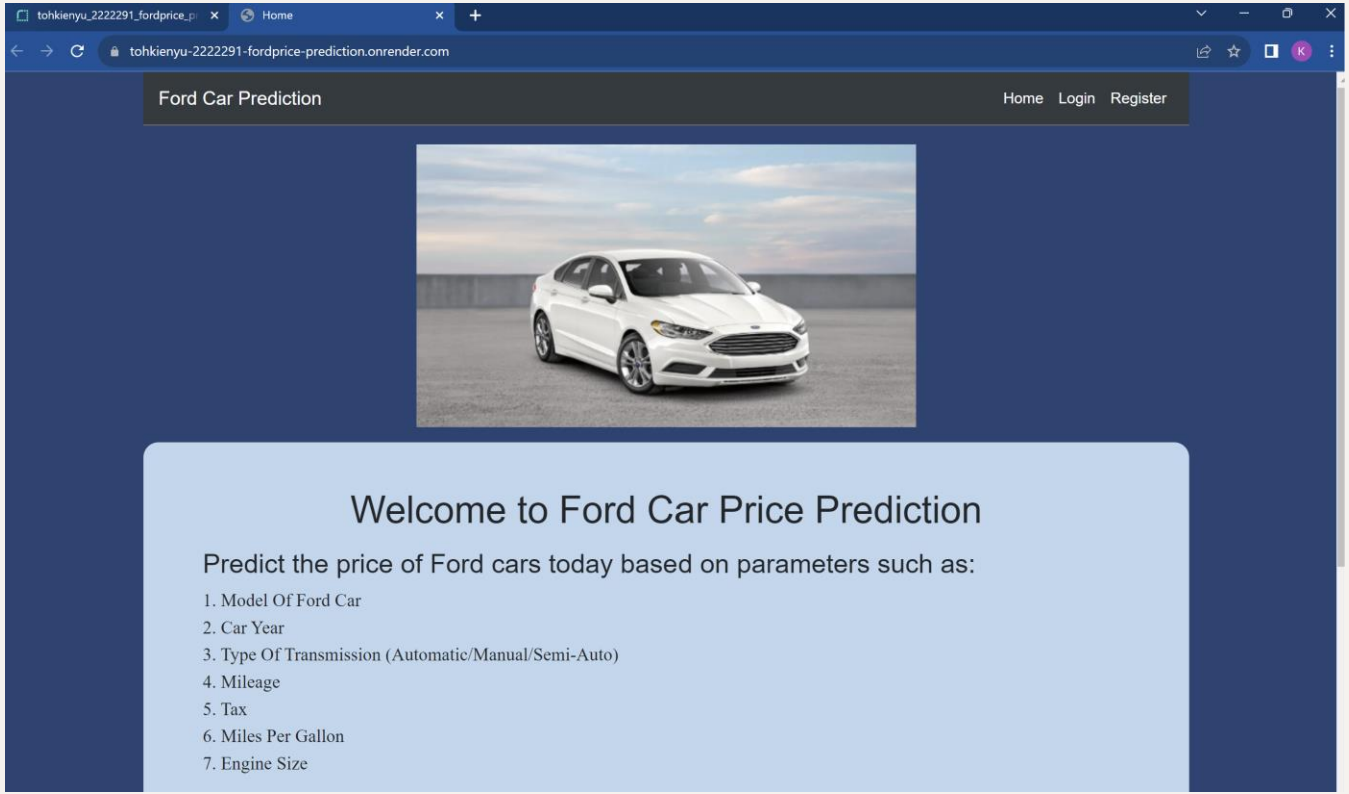
```python
from application import app

if __name__ == "__main__":
    app.run(debug=True,host='0.0.0.0',port=5000)
```



Internet Deployment

Webpage is now accessible on the internet



Thank You