Indian Institute of Technology Jodhpur

MLOps

Assignment 1

ML Ops

Submitted by: Jojo Joseph (G23AI2100)

Google Colab :
https://colab.research.google.com/drive/16uM0n5-4MNq7Ct540Lkxc2pq_mExswpn?usp=sharing

gitHub : https://github.com/TheJojoJoseph/IITJ_MLOpsAssignment1.git

MLOps Assignment

- **Q1. For each of the following scenarios explain the best matching algorithm among :**

  a. Regression
  b. Classification
  c. Clustering
  d. Association Rule Mining
  e. Unsupervised Feature Extraction

- **If we form segments among customers with different financial backgrounds, poverty levels, spending habits and income levels**

Ans

CLUSTERING - The goal is to group customers into segments based on their similarities across multiple features (financial background, poverty level, spending habits, income levels).This is an unsupervised task where you don't have labels and want to discover natural groupings.

- **Suppose the camera has to automatically recognize bus from car**

Ans

Classification.Algorithm - This is a supervised learning issue that requires classifying items (bus vs automobile) based on shape, colour, and other attributes.

The system assigns a label (bus or automobile) based on input data.

- **Suppose you create a feature in facebook to automatically recognize your friends without labeling**

Ans

UNSUPERVISED FEATURE EXTRACTION

This is an example of unsupervised learning, where the system must learn to recognise patterns in data (friend recognition) without pre-labeled data.

● Extract relevant features from user profiles and behaviours without knowing the labels beforehand.

- **If we are to identify the accomplices of a thief in a stealing incident based on the call records**

Ans

ASSOCIATION RULE MINING

Discovering hidden linkages in a dataset involves identifying patterns or connections between a thief and probable accomplices using call records.

● Association rule mining can find relationships between individuals who communicate often.

- **If we are trying to predict future prices of stock based on volume of sales and difference between high and low**

Ans
REGRESSION

This is an example of discovering hidden relationships in a dataset, where you want to identify frequent patterns or connections between the thief and potential accomplices based on call records.
● Association rule mining can help identify links between individuals who frequently contact each other.

- **Q2. For the given data set boston.csv (for each measure MSE and R Sq) (Submit .ipynb files as attachment and screenshots)**

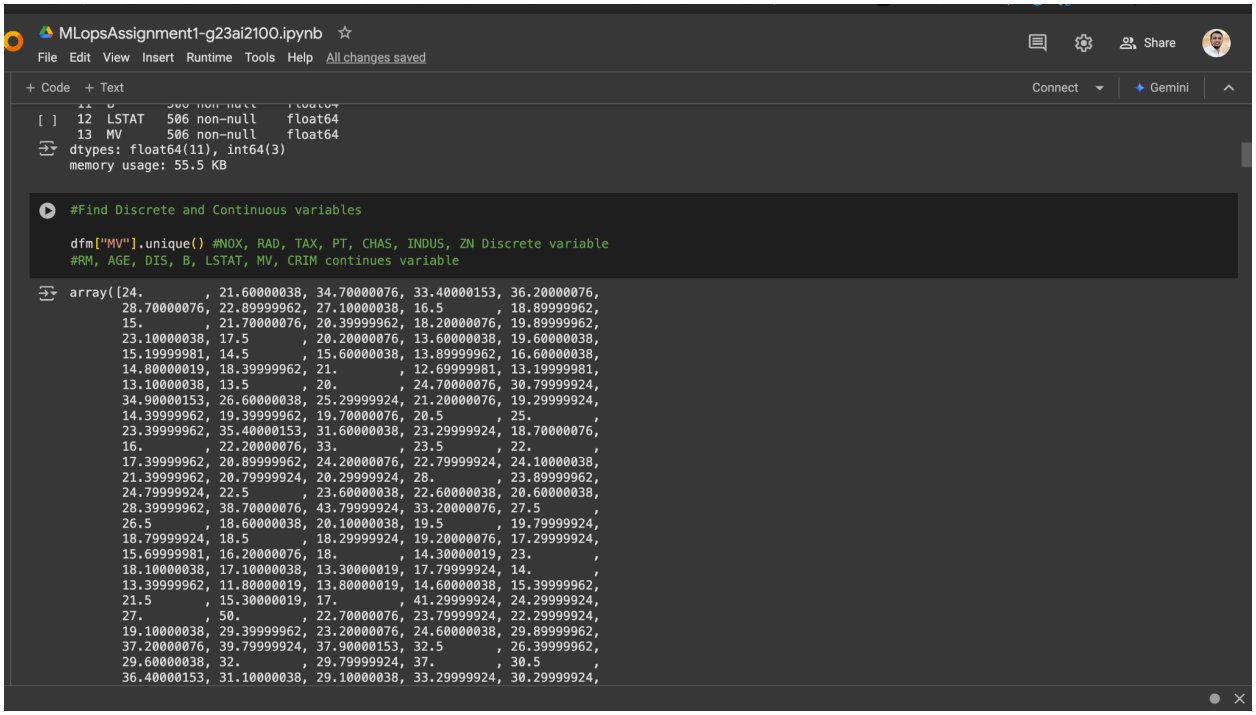
Ans

Google Colab :
https://colab.research.google.com/drive/16uM0n5-4MNq7Ct540Lkxc2pq_mExswpn?usp=sharing

gitHub : https://github.com/TheJojoJoseph/IITJ_MLOpsAssignment1.git

- **Find Discrete and Continuous variables**

Ans



- **Find descriptive analytics of all the variables**

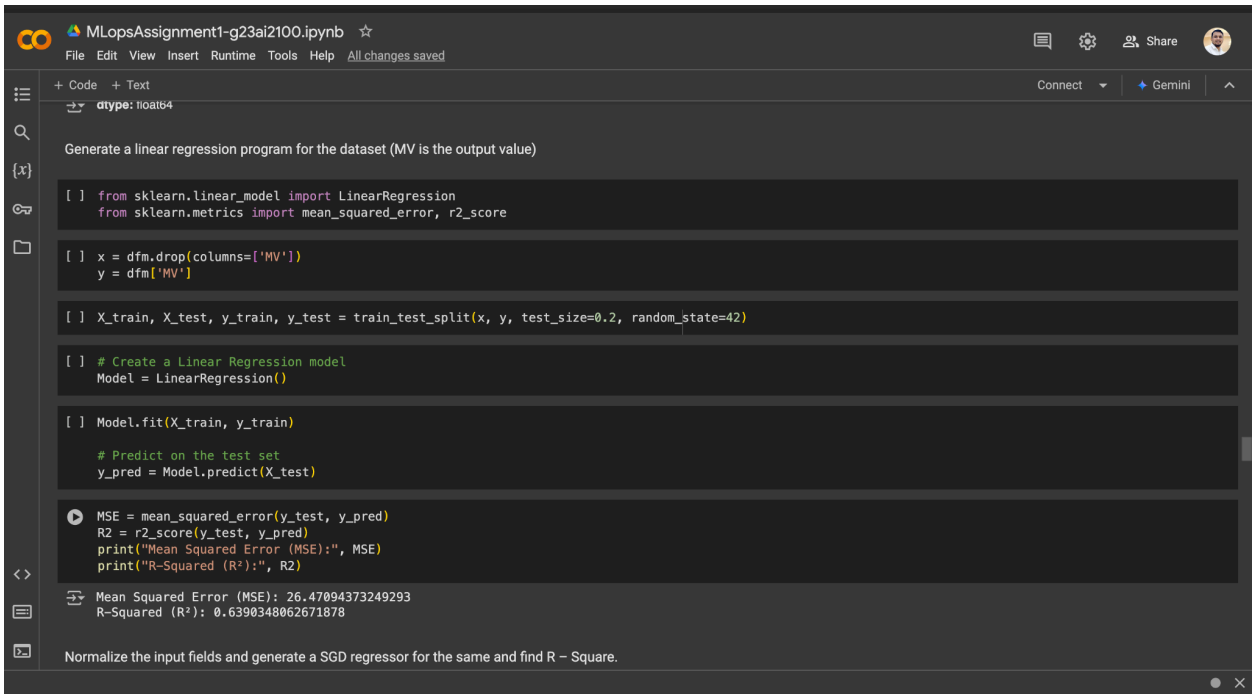
Ans

```
for clm in dfm.columns:
    print(f"Descriptive Analytics for Column: {clm}")
    print(dfm[clm].describe())  # Descriptive statistics
    print("\n" + "-"*50 + "\n")  # Separator for readability
```

```
Descriptive Analytics for Column: CRIM
count    506.000000
mean       3.613524
std        8.601545
min        0.006320
25%        0.082045
50%        0.256510
75%        3.677083
max       88.976196
Name: CRIM, dtype: float64

--------------------------------------------------

Descriptive Analytics for Column: ZN
count    506.000000
mean      11.363636
std       23.322453
min        0.000000
25%        0.000000
50%        0.000000
75%       12.500000
max      100.000000
Name: ZN, dtype: float64

--------------------------------------------------
```

- **Generate a linear regression program for the dataset (MV is the output value)**

Ans



- **Normalize the input fields and generate a SGD regressor for the same and find R – Square.**

Normalize the input fields and generate a SGD regressor for the same and find R – Square.

```
[ ] from sklearn.preprocessing import StandardScaler
    from sklearn.linear_model import SGDRegressor
    from sklearn.metrics import mean_squared_error, r2_score
```

```
[ ] x = dfm.drop(columns=['MV'])
    y = dfm['MV']
    # Normalize the input features
    Scaler = StandardScaler()
    X_scaled = Scaler.fit_transform(x)
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
[ ] sgd_regressor = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)

    sgd_regressor.fit(X_train, y_train)

    y_pred = sgd_regressor.predict(X_test)
```

```
[ ] MSE = mean_squared_error(y_test, y_pred)
    R2 = r2_score(y_test, y_pred)

    # Display results
    print("Mean Squared Error (MSE):", MSE)
    print("R-Squared (R²):", R2)
```

```
Mean Squared Error (MSE): 26.68205970643862
R-Squared (R²): 0.6361559697887642
```

- **Do feature selection by dropping pairwise high correlation fields and run the linear regression and SGD regression after removing irrelevant fields and dropping high corelated fields**

Ans

```python
    # Linear Regression
    linreg = LinearRegression()
    linreg.fit(X_train_pca, y_train)
    y_pred_linreg = linreg.predict(X_test_pca)
    linreg_mse.append(mean_squared_error(y_test, y_pred_linreg))
    linreg_r2.append(r2_score(y_test, y_pred_linreg))

    # SGD Regressor
    sgd = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)
    sgd.fit(X_train_pca, y_train)
    y_pred_sgd = sgd.predict(X_test_pca)
    sgd_mse.append(mean_squared_error(y_test, y_pred_sgd))
    sgd_r2.append(r2_score(y_test, y_pred_sgd))

# Print the results
print("\nResults with Varying Number of PCA Components:\n")
print("Components\tLinear Regression MSE\tLinear Regression R2\tSGD MSE\tSGD R2")
for i, n in enumerate(Components_List):
    print(f"{n}\t\t{linreg_mse[i]:.4f}\t\t{linreg_r2[i]:.4f}\t\t{sgd_mse[i]:.4f}\t\t{sgd_r2[i]:.4f}")
```

```
Results with Varying Number of PCA Components:

Components      Linear Regression MSE   Linear Regression R2    SGD MSE SGD R2
1               39.9893         0.4547          39.9882         0.4547
2               34.1245         0.5347          33.9756         0.5367
3               29.5241         0.5974          29.4339         0.5986
4               28.2844         0.6143          28.3277         0.6137
5               30.0843         0.5898          30.1588         0.5887
6               29.8188         0.5934          29.8853         0.5925
7               29.8093         0.5935          29.8798         0.5926
8               29.7089         0.5949          29.7422         0.5944
9               28.5572         0.6106          28.6777         0.6089
10              28.7325         0.6082          28.8298         0.6069
11              27.0729         0.6308          27.2941         0.6278
```

- **Run PCA on the inputs above and perform both SGD and Linear Regression with varying different no of components**

Ans

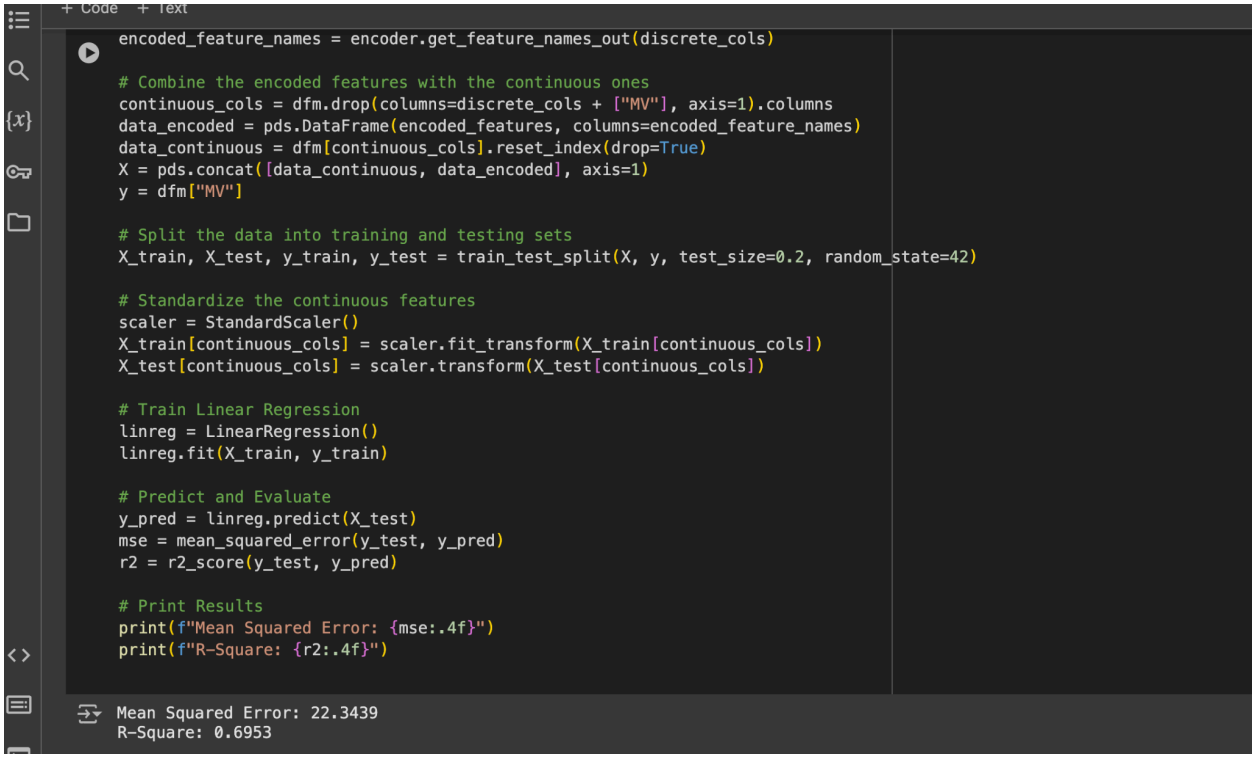
```
[ ] from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error, r2_score

[ ] x = dfm.drop(columns=['MV'])
    y = dfm['MV']

[ ] X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

[ ] # Create a Linear Regression model
    Model = LinearRegression()

[ ] Model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = Model.predict(X_test)

[ ] MSE = mean_squared_error(y_test, y_pred)
    R2 = r2_score(y_test, y_pred)
    print("Mean Squared Error (MSE):", MSE)
    print("R-Squared (R²):", R2)

    Mean Squared Error (MSE): 26.47094373249293
    R-Squared (R²): 0.6390348062671878
```

Normalize the input fields and generate a SGD regressor for the same and find R – Square.

```
+ Code + Text                                                    Connect   Gemini

    linreg_mse.append(mean_squared_error(y_test, y_pred_linreg))
    linreg_r2.append(r2_score(y_test, y_pred_linreg))

    # SGD Regressor
    sgd = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)
    sgd.fit(X_train_pca, y_train)
    y_pred_sgd = sgd.predict(X_test_pca)
    sgd_mse.append(mean_squared_error(y_test, y_pred_sgd))
    sgd_r2.append(r2_score(y_test, y_pred_sgd))

# Print the results
print("\nResults with Varying Number of PCA Components:\n")
print("Components\tLinear Regression MSE\tLinear Regression R2\tSGD MSE\tSGD R2")
for i, n in enumerate(Components_List):
    print(f"{n}\t\t{linreg_mse[i]:.4f}\t\t{linreg_r2[i]:.4f}\t\t{sgd_mse[i]:.4f}\t\t{sgd_r2[i]:.4f}")

Results with Varying Number of PCA Components:

Components      Linear Regression MSE   Linear Regression R2    SGD MSE SGD R2
1               39.9893         0.4547          39.9882         0.4547
2               34.1245         0.5347          33.9756         0.5367
3               29.5241         0.5974          29.4339         0.5986
4               28.2844         0.6143          28.3277         0.6137
5               30.0843         0.5898          30.1588         0.5887
6               29.8188         0.5934          29.8853         0.5925
7               29.8093         0.5935          29.8798         0.5926
8               29.7089         0.5949          29.7422         0.5944
9               28.5572         0.6106          28.6777         0.6089
10              28.7325         0.6082          28.8298         0.6069
11              27.0729         0.6308          27.2941         0.6278
12              26.4709         0.6390          26.7404         0.6354

[ ] import pandas as pds
```

- **Perform one hot encoding of discrete variables and perform linear regression**

Ans



```
encoded_feature_names = encoder.get_feature_names_out(discrete_cols)

# Combine the encoded features with the continuous ones
continuous_cols = dfm.drop(columns=discrete_cols + ["MV"], axis=1).columns
data_encoded = pds.DataFrame(encoded_features, columns=encoded_feature_names)
data_continuous = dfm[continuous_cols].reset_index(drop=True)
X = pds.concat([data_continuous, data_encoded], axis=1)
y = dfm["MV"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the continuous features
scaler = StandardScaler()
X_train[continuous_cols] = scaler.fit_transform(X_train[continuous_cols])
X_test[continuous_cols] = scaler.transform(X_test[continuous_cols])

# Train Linear Regression
linreg = LinearRegression()
linreg.fit(X_train, y_train)

# Predict and Evaluate
y_pred = linreg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print Results
print(f"Mean Squared Error: {mse:.4f}")
print(f"R-Square: {r2:.4f}")
```

```
Mean Squared Error: 22.3439
R-Square: 0.6953
```

- **Do a optuna based hyperparameter search for KNN regression, Random Forest Regression and Ridge regression for the same data and check the best hyperaparameter values**

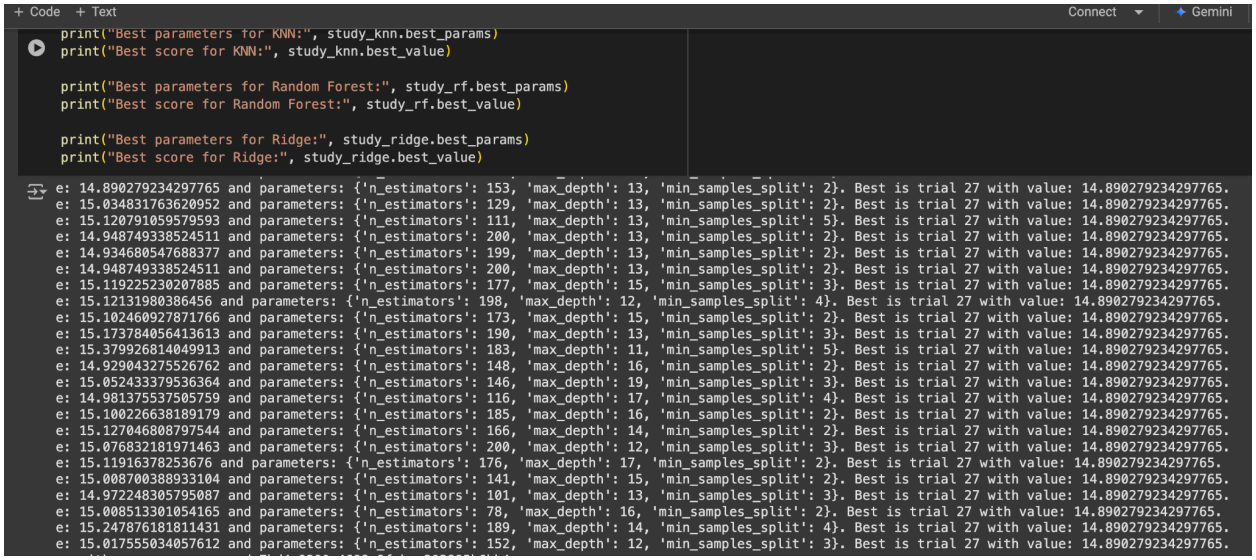
Ans

```
import optuna

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear_model import Ridge

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.metrics import mean_squared_error

import pandas as pds

import numpy as npy
```

```
X = dfm.drop(columns=["MV"], axis=1)

y = dfm["MV"]



# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)



# Standardization pipeline

Scaler = StandardScaler()

X_train = Scaler.fit_transform(X_train)

X_test = Scaler.transform(X_test)



# Define objective functions for each model



def knn_objective(trial):

   n_neighbors = trial.suggest_int("n_neighbors", 1, 20)

   weights = trial.suggest_categorical("weights", ["uniform", "distance"])

   p = trial.suggest_int("p", 1, 2)



   model = KNeighborsRegressor(n_neighbors=n_neighbors, weights=weights,
p=p)
```

```
    score = cross_val_score(model, X_train, y_train,
scoring="neg_mean_squared_error", cv=5).mean()

    return -score



def rf_objective(trial):

    n_estimators = trial.suggest_int("n_estimators", 10, 200)

    max_depth = trial.suggest_int("max_depth", 3, 20)

    min_samples_split = trial.suggest_int("min_samples_split", 2, 10)



    model = RandomForestRegressor(n_estimators=n_estimators,
max_depth=max_depth, min_samples_split=min_samples_split, random_state=42)

    score = cross_val_score(model, X_train, y_train,
scoring="neg_mean_squared_error", cv=5).mean()

    return -score



def ridge_objective(trial):

    alpha = trial.suggest_loguniform("alpha", 1e-3, 1e2)



    model = Ridge(alpha=alpha)

    score = cross_val_score(model, X_train, y_train,
scoring="neg_mean_squared_error", cv=5).mean()

    return -score



# Run Optuna for each model

print("Optimizing KNN...")
```

```
study_knn = optuna.create_study(direction="minimize")
study_knn.optimize(knn_objective, n_trials=50)

print("Optimizing Random Forest...")
study_rf = optuna.create_study(direction="minimize")
study_rf.optimize(rf_objective, n_trials=50)

print("Optimizing Ridge...")
study_ridge = optuna.create_study(direction="minimize")
study_ridge.optimize(ridge_objective, n_trials=50)

# Print the best parameters and scores
print("Best parameters for KNN:", study_knn.best_params)
print("Best score for KNN:", study_knn.best_value)

print("Best parameters for Random Forest:", study_rf.best_params)
print("Best score for Random Forest:", study_rf.best_value)

print("Best parameters for Ridge:", study_ridge.best_params)
print("Best score for Ridge:", study_ridge.best_value)
```



- **Develop a small web page in flask based on input data and predicting MV output with different input data given**

app.py

TechStack : flask

```
from flask import Flask, request, render_template
import joblib
import pandas as pd
app = Flask(__name__)
model = joblib.load('linear_regression_model.pkl')
scaler = joblib.load('scaler.pkl')

feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
                 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']



@app.route('/', methods=['GET'])
def home():
    return render_template('index.html', feature_names=feature_names)



# prediction
@app.route('/predict', methods=['POST'])
def predict():
    try:
```

```
        # Collect input features from the form
        input_features = []
        for feature in feature_names:
            value = request.form.get(feature)
            if not value:
                raise ValueError(f"Value for '{feature}' is missing.")
            try:
                input_features.append(float(value))
            except ValueError:
                raise ValueError(f"Invalid value for '{
                                feature}'. Please enter a numeric value.")

        input_data = pd.DataFrame(
            [input_features], columns=feature_names)  # Convert  DataFrame
        input_scaled = scaler.transform(input_data)   # Scale  input features
        prediction = model.predict(input_scaled)         # Make  prediction
        prediction = round(prediction[0], 2)
        return render_template('result.html', prediction=prediction)

    except Exception as e:
        return render_template('index.html', error=str(e), feature_names=feature_names)


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

Dataset

boston.csv > data

```
CRIM,ZN,INDUS,CHAS,NOX,RM,AGE,DIS,RAD,TAX,PT,B,LSTAT,MV
0.00632,18,2.309999943,0,0.537999988,6.574999809,65.19999695,4.090000153,1,296,15.30000019,396.8999939,4.980000019,24
0.027310001,0,7.070000172,0,0.469000012,6.421000004,78.90000153,4.967100143,2,242,17.79999924,396.8999939,9.140000343,21.60000038
0.02729,0,7.070000172,0,0.469000012,7.184999943,61.09999847,4.967100143,2,242,17.79999924,392.8299866,4.03000021,34.70000076
0.032370001,0,2.180000067,0,0.458000004,6.998000145,45.79999924,6.062200069,3,222,18.70000076,394.6300049,2.940000057,33.40000153
0.069049999,0,2.180000067,0,0.458000004,7.146999836,54.20000076,6.062200069,3,222,18.70000076,396.8999939,5.329999924,36.20000076
0.029850001,0,2.180000067,0,0.458000004,6.429999828,58.70000076,6.062200069,3,222,18.70000076,394.1199951,5.210000038,28.70000076
0.088289998,12.5,7.869999886,0,0.523999989,6.012000084,66.59999847,5.560500145,5,311,15.19999981,395.6000061,12.43000031,22.89999962
0.144549996,12.5,7.869999886,0,0.523999989,6.171999931,96.09999847,5.950500011,5,311,15.19999981,396.8999939,19.14999962,27.10000038
0.211239994,12.5,7.869999886,0,0.523999989,5.631000042,100,6.082099915,5,311,15.19999981,386.6300049,29.93000031,16.5
0.170039997,12.5,7.869999886,0,0.523999989,6.004000187,85.90000153,6.592100143,5,311,15.19999981,386.7099915,17.10000038,18.89999962
0.224889994,12.5,7.869999886,0,[illegible],6.376999855,94.30000305,6.346700191,5,311,15.19999981,392.519989,20.45000076,15
0.117470004,12.5,7.869999886,0,[illegible],6.008999825,82.90000153,6.226699829,5,311,15.19999981,396.8999939,13.27000046,18.89999962
0.093780003,12.5,7.869999886,0,0.523999989,5.888999939,39,5.450900078,5,311,15.19999981,390.5,15.71000004,21.70000076
0.629760027,0,8.140000343,0,0.537999988,5.948999882,61.79999924,4.707499981,4,307,21,396.8999939,8.260000229,20.39999962
0.637960017,0,8.140000343,0,0.537999988,6.096000195,84.5,4.461900234,4,307,21,380.019989,10.26000023,18.20000076
0.627390027,0,8.140000343,0,0.537999988,5.834000111,56.5,4.498600006,4,307,21,395.6199951,8.470000267,19.89999962
1.053930044,0,8.140000343,0,0.537999988,5.934999943,29.29999924,4.498600006,4,307,21,386.8500061,6.579999924,23.10000038
0.784200013,0,8.140000343,0,0.537999988,5.989999771,81.69999695,4.257900238,4,307,21,386.75,14.67000008,17.5
0.802709997,0,8.140000343,0,0.537999988,5.455999851,36.59999847,3.796499968,4,307,21,288.9899902,11.68999958,20.20000076
0.725799978,0,8.140000343,0,0.537999988,5.727000237,69.5,3.796499968,4,307,21,390.9500122,11.27999973,18.20000076
1.251790047,0,8.140000343,0,0.537999988,5.570000172,98.09999847,3.797899961,4,307,21,376.5700073,21.02000046,13.60000038
0.852039993,0,8.140000343,0,0.537999988,5.965000153,89.19999695,4.012300014,4,307,21,392.5299988,13.82999992,19.60000038
1.232470036,0,8.140000343,0,0.537999988,6.142000198,91.69999695,3.976900101,4,307,21,396.8999939,18.71999931,15.19999981
0.988430023,0,8.140000343,0,0.537999988,5.813000202,100,4.095200062,4,307,21,394.5400085,19.87999916,14.5
0.750259995,0,8.140000343,0,0.537999988,5.923999786,94.09999847,4.399600029,4,307,21,394.3299866,16.29999924,15.60000038
0.840539992,0,8.140000343,0,0.537999988,5.598999977,85.69999695,4.454599857,4,307,21,303.4200134,16.51000023,13.89999962
0.671909988,0,8.140000343,0,0.537999988,5.813000202,90.30000305,4.68200016,4,307,21,376.8800049,14.81000042,16.60000038
```

Col 5: NOX

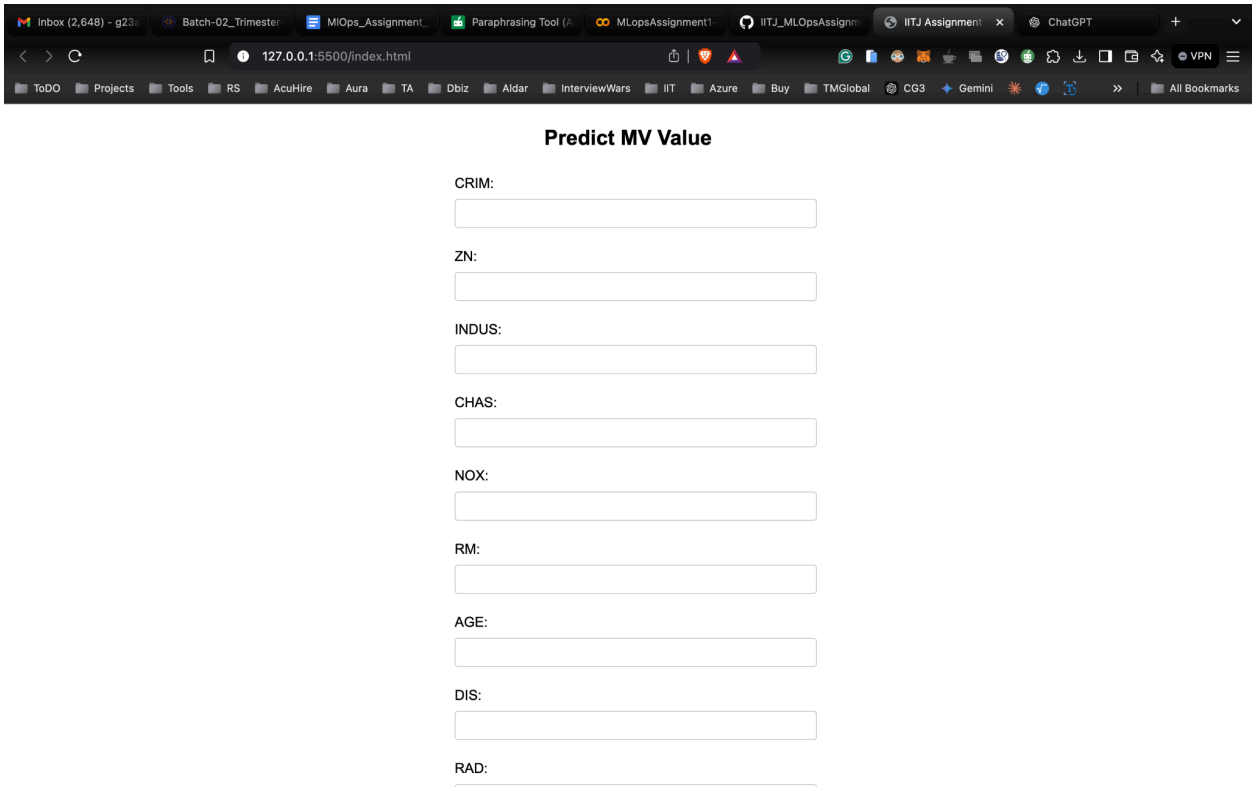
PROBLEMS 43 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE SPELL CHECKER 4 COMMENTS GITLENS Veracode Scan for VS Cod

training.py

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
import joblib


data = pd.read_csv('boston.csv')

X = data.drop('MV', axis=1)  # Separate  (X) and target (y)
y = data['MV']
X = X[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
       'AGE', 'DIS', 'RAD', 'TAX', 'PT', 'B', 'LSTAT']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)  # Split data into
training and testing sets

scaler = StandardScaler()  # Scale data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Training
model = LinearRegression()
model.fit(X_train_scaled, y_train)
# Save the scaler and model
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(model, 'linear_regression_model.pkl')


print("Model and scaler  complete")
```

UI and index.html




```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>IITJ Assignment 1 MLops - g23ai2100</title>
   <style>
     body {
       font-family: Arial, sans-serif;
       line-height: 1.6;
```

```
      margin: 20px;
    }
    form {
      max-width: 400px;
      margin: 0 auto;
    }
    label {
      display: block;
      margin: 10px 0 5px;
    }
    input {
      width: 100%;
      padding: 8px;
      margin-bottom: 10px;
      border: 1px solid #ccc;
      border-radius: 4px;
    }
    button {
      background-color: #007bff;
      color: white;
      border: none;
      padding: 10px 15px;
      border-radius: 4px;
      cursor: pointer;
    }
    button:hover {
      background-color: #0056b3;
    }
    h2 {
      text-align: center;
    }
    h3 {
      text-align: center;
      color: green;
    }
  </style>
</head>
<body>
  <h2>Predict MV Value</h2>
  <form method="post" action="/">
    <label for="CRIM">CRIM:</label>
    <input type="number" name="CRIM" id="CRIM" required step="any" />
```

```
    <label for="ZN">ZN:</label>
    <input type="number" name="ZN" id="ZN" required step="any" />

    <label for="INDUS">INDUS:</label>
    <input type="number" name="INDUS" id="INDUS" required step="any" />

    <label for="CHAS">CHAS:</label>
    <input type="number" name="CHAS" id="CHAS" required step="any" />

    <label for="NOX">NOX:</label>
    <input type="number" name="NOX" id="NOX" required step="any" />

    <label for="RM">RM:</label>
    <input type="number" name="RM" id="RM" required step="any" />

    <label for="AGE">AGE:</label>
    <input type="number" name="AGE" id="AGE" required step="any" />

    <label for="DIS">DIS:</label>
    <input type="number" name="DIS" id="DIS" required step="any" />

    <label for="RAD">RAD:</label>
    <input type="number" name="RAD" id="RAD" required step="any" />

    <label for="TAX">TAX:</label>
    <input type="number" name="TAX" id="TAX" required step="any" />

    <label for="PTRATIO">PTRATIO:</label>
    <input type="number" name="PTRATIO" id="PTRATIO" required step="any" />

    <label for="B">B:</label>
    <input type="number" name="B" id="B" required step="any" />

    <label for="LSTAT">LSTAT:</label>
    <input type="number" name="LSTAT" id="LSTAT" required step="any" />

    <button type="submit">Predict</button>
  </form>
  <h3>{{ prediction_text }}</h3>
 </body>
</html>
```