```
1 Start coding or generate with AI.
```

Assignment Supervised machine Learning - Regression

House Price Prediction Using Bolton Housing Data

This dataset provides information about house prices in Bolton. House Price Prediction

In this notebook, you'll follow the basic machine learning process to build a regression model to predict house prices using the "Boston Housing Dataset" from sklearn. The regression model will either be a Decision Tree or Random Forest regressor.

Follow the instructions and complete each TODO to complete the assessment on the essential steps in building and evaluating a regression model.

The following is a description of each column in the dataset:

Dataset Features (Bolton Housing):

- CRIM: Crime rate by town
- ZN: Proportion of residential land zoned for large lots
- INDUS: Proportion of non-retail business acres per town
- CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX: Nitric oxide concentration (parts per 10 million)
- RM: Average number of rooms per dwelling
- AGE: Proportion of owner-occupied units built before 1940
- DIS: Weighted distances to five Boston employment centers
- RAD: Index of accessibility to radial highways
- TAX: Full-value property tax rate per `$10,000`
- PTRATIO: Pupil-teacher ratio by town
- B: Proportion of Black population
- LSTAT: Percentage of lower status of the population
- MEDV (Target): Median value of owner-occupied homes in `$1,000s`

Dataset is from sklearn Datasets

```
1 from seaborn.palettes import color_palette
2
3
4 # --- Imports ---
5 # TODO: Import all the necessary libraries for data handling, visualization, and model building.
6 # Example: import pandas as pd
7 # Add your imports here:
8
9 import pandas as pd
10 import numpy as np
11 from sklearn.datasets import fetch_openml
12 from sklearn.model_selection import train_test_split
13 from sklearn.tree import DecisionTreeRegressor
14 from sklearn.ensemble import RandomForestRegressor
15 from sklearn.metrics import mean_squared_error, r2_score
16 from matplotlib import pyplot as plt
17 import seaborn as sns
18 from sklearn.model_selection import GridSearchCV
19
20 # set up plot area:
21
22 # %matplotlib inline
23 # sns.set_style(style='darkgrid', palette ='pastel')
24
25 # --- Data Collection and Loading ---
26 # TODO: Load the 'Boston Housing' dataset from sklearn and convert it into a pandas DataFrame.
27 # Hint: Use `load_boston()` from `sklearn.datasets`
28
29 # Load dataset and convert to DataFrame:
30
31 # Add your code here:
32
```

```
boston_data = fetch_openml(name='boston', version=1, as_frame=True) # load the boston dataset from sklearn
df = boston_data.frame # convert to df

# --- Quick Check of Data ---
# TODO: Display the first few rows of the dataset to understand its structure.
# Hint: Use `.head()` to inspect the first few rows.

# Add your code here:
print("Inspect the Dataset")
display(df.head())
print("-" * 30)
# --- Data Exploration ---

# TODO: Check the features and target variable. Identify which is continuous and categorical if applicable.
# Hint: Use `.info()` and `.describe()` to inspect data types and statistical properties.

# Add your code here:
print("Dataset Info:")
df.info()
print("Summary Statistics:")
print("-" * 140)
display(df.describe())
print("-" * 15)
# --- EDA and Data Preprocessing ---
# TODO: Check for missing/null values.
# Hint: Use `.isnull().sum()` to check for null values.

# Add your code here:

print("Missing Values:")
print(df.isnull().sum())
print("-" * 25)

# Elimiate Duplicates

df=df.drop_duplicates()
print("Duplicates Removed")
print("-" * 25)

# --- Step 5: Rename Columns for Clarity ---

# Rename columns to descriptive


df.rename(columns={
    'CRIM': 'crime_rate',                 # per capita crime rate by town
    'ZN': 'residential_land_zone',        # proportion of residential land zoned for lots over 25,000 sq.ft.
    'INDUS': 'non_retail_business_acres', # proportion of non-retail business acres per town
    'CHAS': 'charles_river_dummy',        # Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    'NOX': 'nitric_oxides_conc',          # nitric oxides concentration (parts per 10 million)
    'RM': 'avg_rooms_per_dwelling',       # average number of rooms per dwelling
    'AGE': 'owner_occupied_units_pct',    # proportion of owner-occupied units built prior to 1940
    'DIS': 'distance_to_employment',      # weighted distances to five Boston employment centers
    'RAD': 'highway_accessibility_index', # index of accessibility to radial highways
    'TAX': 'property_tax_rate',           # full-value property-tax rate per $10,000
    'PTRATIO': 'pupil_teacher_ratio',     # pupil-teacher ratio by town
    'B': 'black_population_index',        # 1000(Bk - 0.63)^2 where Bk is proportion of Black population by town
    'LSTAT': 'lower_status_pct',          # percentage of lower status population
    'MEDV': 'median_home_value'           # median value of owner-occupied homes in $1000s
}, inplace=True)

# --- Display the renamed columns ---
print("Renamed Columns:")
print(df.columns.tolist())


# --- Data Visualization ---

# TODO: Visualize the data. Create scatter plots to see the relationship between independent features and the target variab
# Example: Use `plt.scatter()` to visualize the relationship between features like 'RM' (average number of rooms) and the t

# Add your code here:
plt.figure(figsize=(10, 6))
sns.heatmap(
    df.corr(),
    annot=True, # indicates the corr values
    cmap='coolwarm',
    fmt=".2f"
```

```
    fmt= '.2f',
    linewidths=0.5, # add spaces between cells
    cbar_kws={'shrink': 0.8}) #smaller color bar
plt.title('Correlation Heatmap', fontsize=14, pad=12)
plt.tight_layout()
plt.xticks(rotation=45) # rotate for readiilty
plt.yticks(rotation=0)
plt.show()


# TODO: Create a function to automate scatter plots for all features vs the target variable.
# Hint: The function should loop over a list of features and plot scatter plots for each.

# Define your function here:

def plot_scatter_matrix(df, target_col, feature_cols): #ploting the target col (median_home_value) against all independent
    feature_cols = [c for c in df.select_dtypes(include=[np.number]).columns if c !=target_col] # get all the numerical col
    plt.figure(figsize=(15, 12)) # set the figure size
    plt.suptitle('Scatter Plots: Features vs Target (Median Home Value)', fontsize=16, y=1.02) # plot title- using this sup

    for i, feature in enumerate(feature_cols, 1): # Loop through each feature - this code goes through each feature in the
        plt.subplot(3, 5, i) # adjust the grid, 2 cols, 3 rows based on the features in the dataset
        sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
        plt.title(f'{feature_cols} vs {target_col}', fontsize=11)
        plt.xlabel(feature_cols, fontsize=9) # ploting the x-axis
        plt.ylabel(target_col, fontsize=9) #ploting the y-axis


plt.tight_layout()
plt.show()

# call the function

plot_scatter_matrix(df, target_col='median_home_value', feature_cols='feature')

# --- Feature Selection ---
# TODO: Use the function to visualize the relationships between multiple features and the target variable.
# Example: ['RM', 'LSTAT', 'AGE', 'CRIM']
# Target: 'MEDV'

# Add your code here:

selected_feature = ['crime_rate', 'residential_land_zone', 'non_retail_business_acres', 'nitric_oxides_conc']

def viz_selected_feat(df, features, target_col ='median_home_value'):
    plt.figure(figsize=(15, 12))
    plt.suptitle('Selected Features vs Target (Median Home Value)', fontsize=16, y=1.02)

    for j, feature in enumerate(features, 1):
        plt.subplot(2, 2, j)
        plt.scatter(x=df[feature], y=df[target_col], alpha=0.6, color='teal')
        plt.title(f'{feature} vs {target_col}', fontsize=14, pad=12)
        plt.xlabel(features, fontsize=9)
        plt.ylabel(target_col, fontsize=9)

plt.tight_layout()
plt.show()

viz_selected_feat(df, target_col='median_home_value', features=selected_feature)
print("-"  * 120)

# --- ML Model Training ---
# TODO: Split the dataset into training and testing sets.
# Hint: Use `train_test_split()` from `sklearn.model_selection` with an 80/20 split.

# Define X (features) and y (target) and perform the train-test split:

# Define feature matrix X (all independent variables)
# and target vector y (dependent variable)
X = df.drop('median_home_value', axis=1) # the target variable has been dropped, because it is the varible that I wish to p
y = df['median_home_value']

# split the dataset for training:

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

# TODO: Choose an appropriate regression model: Decision Tree or Random Forest.
# Hint: Use either `DecisionTreeRegressor` or `RandomForestRegressor` from `sklearn.tree` or `sklearn.ensemble`.
```

```
# Define your regression model here:

model = DecisionTreeRegressor(random_state=42)

# TODO: Train the model on the training data.
# Hint: Use `.fit()` to train the model.

# Add your code here:

model.fit(X_train, y_train)

# --- Model Evaluation ---
# TODO: Evaluate the performance of the model on the test set using relevant metrics (e.g., RMSE, R-squared).
# Hint: Use `mean_squared_error()` and `r2_score()` from `sklearn.metrics`.

# Predict on the test set and calculate the evaluation metrics:

y_pred = model.predict(X_test) # predict the model

eval_model = r2_score(y_test, y_pred) # this regression evlaute my model to determine if the selected model performance wel
root_mean_squared_error = np.sqrt(mean_squared_error(y_test, y_pred))

# Display evaluation metrics
print("Model Performance Metrics:")
print("---------------------------")
print(f"R-squared (R²): {eval_model:.3f}")
print(f"Root Mean Squared Error (RMSE): {root_mean_squared_error :.3f}")

# Perform parameter tuning on the model if needed to improve the performance of your model.

# Add your parameter tuning code here:

# Define the base model

tree = DecisionTreeRegressor(random_state=42)

# Define the parameter grid

param_grid = {
    'max_depth': [3, 5, 7, 9, 12, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 6],
    'max_features': [None, 'sqrt', 'log2']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=tree,
    param_grid=param_grid,
    cv=5,                 # 5-fold cross-validation
    scoring='r2',         # Evaluate based on R-squared
    n_jobs=-1,            # Use all CPU cores
    verbose=1             # Show progress
)

# Fit the grid search on training data

grid_search.fit(X_train, y_train)

# Retrieve the best model and parameters

best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

print("Best Parameters Found:")
print(best_params)

# Evaluate best model on the test data

y_pred_best = best_model.predict(X_test)
r2_best = r2_score(y_test, y_pred_best)
rmse_best = np.sqrt(mean_squared_error(y_test, y_pred_best))

print("\nTuned Model Performance:")
print("------------------------")
print(f"R-squared (R²): {r2_best:.3f}")
```

```
print(f"Root Mean Squared Error (RMSE): {rmse_best:.3f}")
print("---------------------------------")
# --- Model Prediction ---
# TODO: Predict house prices from a new set of feature inputs.
# Example new data: Use hypothetical or randomly generated values for the features.
#
# Example new data: CRIM = 0.2, ZN = 12.5, INDUS = 7.07, CHAS = 0, NOX = 0.5, RM = 6.5, AGE = 68, DIS = 4.0, RAD = 2, TAX =

# Add your prediction code here:

# The diction created below was generated based on the above text with feature names and hypothetical values given.

new_house_data = {
    'crime_rate': [0.2],
    'residential_land_zone': [12.5],
    'non_retail_business_acres': [7.07],
    'charles_river_dummy': [0],
    'nitric_oxides_conc': [0.5],
    'avg_rooms_per_dwelling': [6.5],
    'owner_occupied_units_pct': [68],
    'distance_to_employment': [4.0],
    'highway_accessibility_index': [2],
    'property_tax_rate': [250],
    'pupil_teacher_ratio': [17],
    'black_population_index': [400],
    'lower_status_pct': [12]
}

# Convert the dictionary to a DataFrame
new_data_df = pd.DataFrame(new_house_data)

# Use the best model if you tuned, otherwise your base model
predicted_value = model.predict(new_data_df) # the best model is being used here to because of the prediction that was made

print("Predicted Median Home Value:")
print(f"${predicted_value[0]:.2f}K")


```

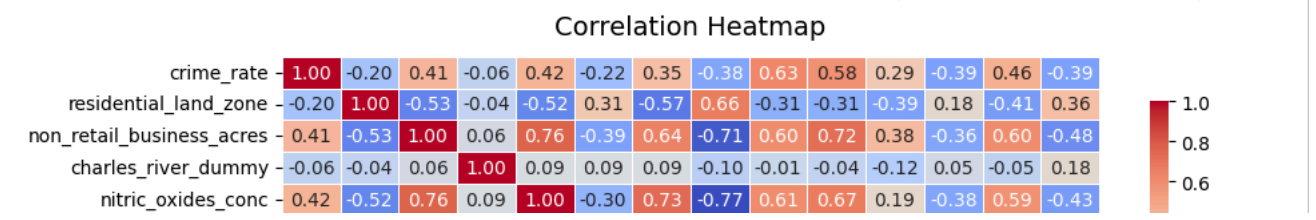
Inspect the Dataset

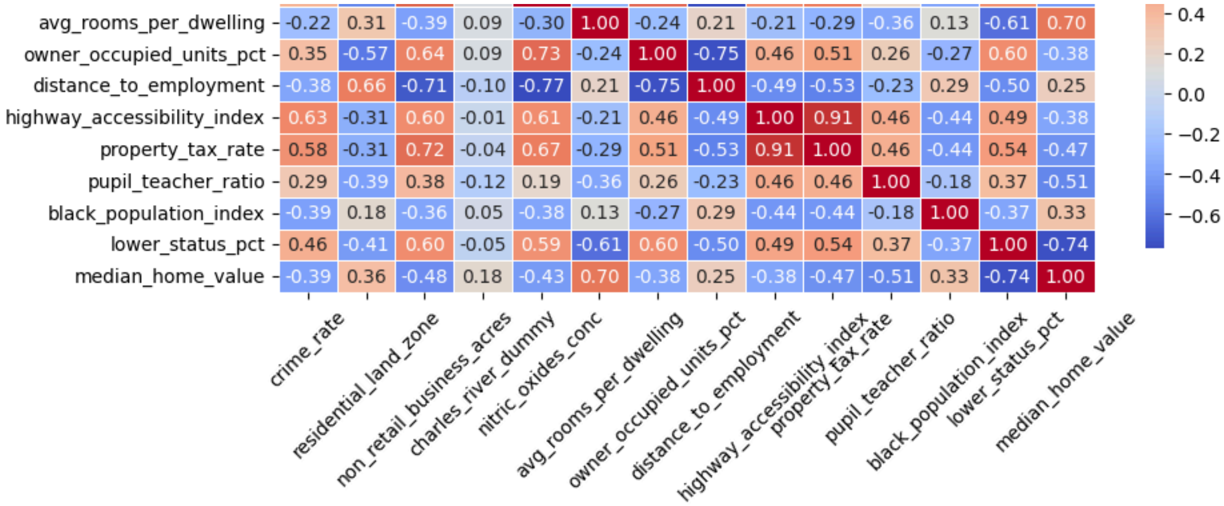
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
------------------------------
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    category
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    category
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: category(2), float64(12)
memory usage: 49.0 KB
Summary Statistics:
```

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.554695	6.284634	68.574901	3.795043	408.237154	18.455534	356.674032	12.65
std	8.601545	23.322453	6.860353	0.115878	0.702617	28.148861	2.105710	168.537116	2.164946	91.294864	7.14
min	0.006320	0.000000	0.460000	0.385000	3.561000	2.900000	1.129600	187.000000	12.600000	0.320000	1.73
25%	0.082045	0.000000	5.190000	0.449000	5.885500	45.025000	2.100175	279.000000	17.400000	375.377500	6.95
50%	0.256510	0.000000	9.690000	0.538000	6.208500	77.500000	3.207450	330.000000	19.050000	391.440000	11.36
75%	3.677083	12.500000	18.100000	0.624000	6.623500	94.075000	5.188425	666.000000	20.200000	396.225000	16.95
max	88.976200	100.000000	27.740000	0.871000	8.780000	100.000000	12.126500	711.000000	22.000000	396.900000	37.97

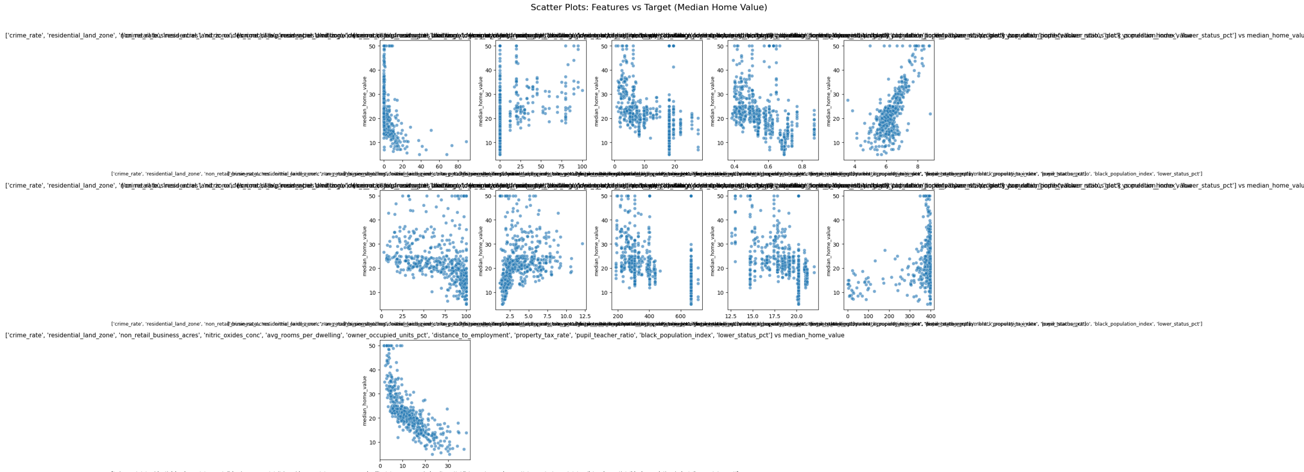
```
---------------
Missing Values:
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
-------------------------
Duplicates Removed
-------------------------
Renamed Columns:
['crime_rate', 'residential_land_zone', 'non_retail_business_acres', 'charles_river_dummy', 'nitric_oxides_conc', 'avg_rooms_per
```






```
<Figure size 640x480 with 0 Axes>
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
/tmp/ipython-input-1262737358.py:132: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.
  sns.scatterplot(x=df[feature], y=df[target_col],  data=df, alpha=0.6, palette='teal')
```

Scatter Plots: Features vs Target (Median Home Value)



```
------------------------------------------------------------------------------------------------------------------------------------------------------
Model Performance Metrics:
--------------------------
R-squared (R²): 0.858
Root Mean Squared Error (RMSE): 3.227
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
Best Parameters Found:
{'max_depth': 7, 'max_features': None, 'min_samples_leaf': 6, 'min_samples_split': 2}

Tuned Model Performance:
-----------------------
R-squared (R²): 0.794
Root Mean Squared Error (RMSE): 3.886
--------------------------------
Predicted Median Home Value:
$23.10K
```

Selected Features vs Target (Median Home Value)