## Regression Problem

- Explore feature correlation
- Simple Linear Regression
- Evaluate Model with MSE, MAE, R-square

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
columns = ['Overall Qual', 'Overall Cond', 'Gr Liv Area',
           'Central Air', 'Total Bsmt SF', 'SalePrice']

df = pd.read_csv('http://jse.amstat.org/v19n3/decock/AmesHousing.txt',
                 sep='\t',
                 usecols=columns)

df.head()
```

| | Overall Qual | Overall Cond | Total Bsmt SF | Central Air | Gr Liv Area | SalePrice |
|---|---|---|---|---|---|---|
| 0 | 6 | 5 | 1080.0 | Y | 1656 | 215000 |
| 1 | 5 | 6 | 882.0 | Y | 896 | 105000 |
| 2 | 6 | 6 | 1329.0 | Y | 1329 | 172000 |
| 3 | 7 | 5 | 2110.0 | Y | 2110 | 244000 |
| 4 | 5 | 5 | 928.0 | Y | 1629 | 189900 |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.describe()
```

| | Overall Qual | Overall Cond | Total Bsmt SF | Gr Liv Area | SalePrice |
|---|---|---|---|---|---|
| count | 2930.000000 | 2930.000000 | 2929.000000 | 2930.000000 | 2930.000000 |
| mean | 6.094881 | 5.563140 | 1051.614544 | 1499.690444 | 180796.060068 |
| std | 1.411026 | 1.111537 | 440.615067 | 505.508887 | 79886.692357 |
| min | 1.000000 | 1.000000 | 0.000000 | 334.000000 | 12789.000000 |
| 25% | 5.000000 | 5.000000 | 793.000000 | 1126.000000 | 129500.000000 |
| 50% | 6.000000 | 5.000000 | 990.000000 | 1442.000000 | 160000.000000 |
| 75% | 7.000000 | 6.000000 | 1302.000000 | 1742.750000 | 213500.000000 |
| max | 10.000000 | 9.000000 | 6110.000000 | 5642.000000 | 755000.000000 |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Overall Qual   2930 non-null   int64
 1   Overall Cond   2930 non-null   int64
 2   Total Bsmt SF  2929 non-null   float64
 3   Central Air    2930 non-null   object
 4   Gr Liv Area    2930 non-null   int64
```

```
 5   SalePrice      2930 non-null   int64
dtypes: float64(1), int64(4), object(1)
memory usage: 137.5+ KB
```

```
# Encoding categorical feature
df['Central Air'] = df['Central Air'].map({'N': 0, 'Y': 1})
```

```
# Remove NaN
df = df.dropna(axis=0)
df.isnull().sum()
```

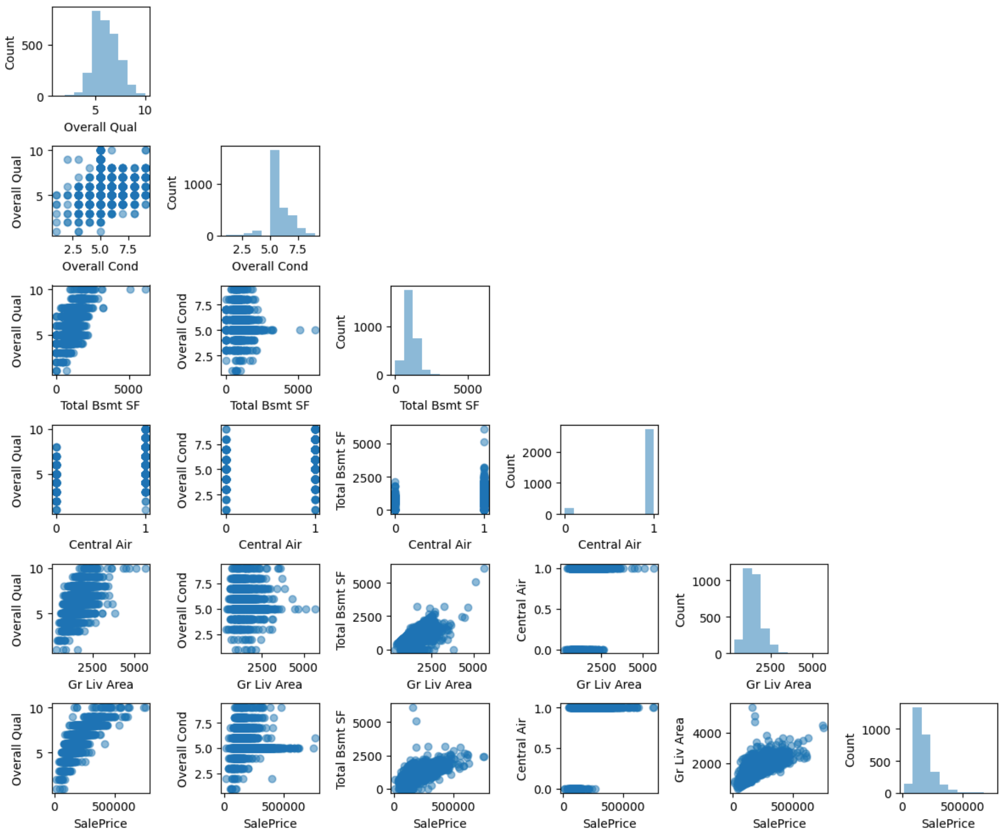
| | 0 |
|---|---|
| **Overall Qual** | 0 |
| **Overall Cond** | 0 |
| **Total Bsmt SF** | 0 |
| **Central Air** | 0 |
| **Gr Liv Area** | 0 |
| **SalePrice** | 0 |

**dtype:** int64

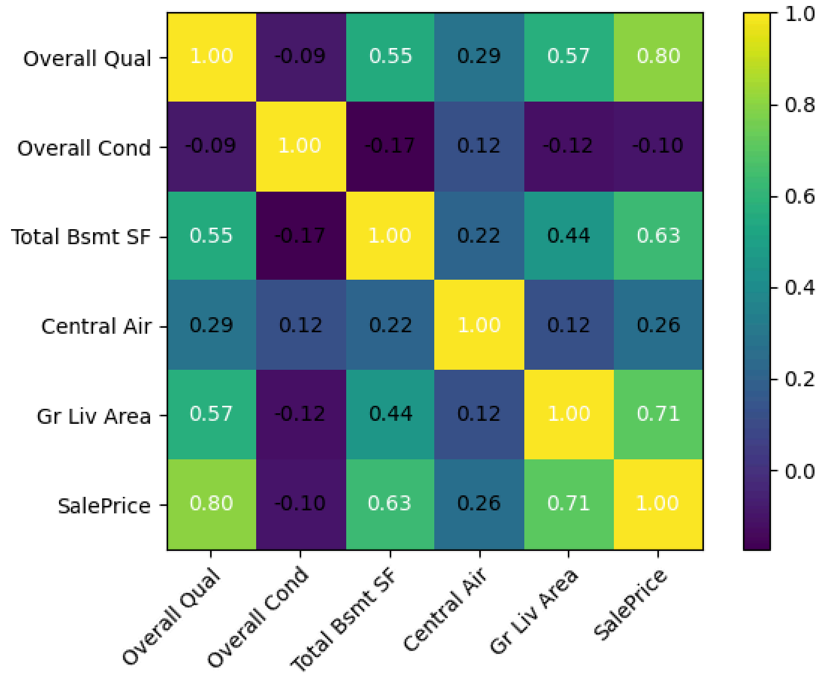
## Visualizing Feature Correlation

```
import matplotlib.pyplot as plt
from mlxtend.plotting import scatterplotmatrix

# Use pairplot as an alternative
```

```
scatterplotmatrix(df.values, figsize=(12, 10),
                  names=df.columns, alpha=0.5)
plt.tight_layout()
```




```
import numpy as np
from mlxtend.plotting import heatmap

cm = np.corrcoef(df.values.T)
hm = heatmap(cm, row_names=df.columns, column_names=df.columns)

plt.tight_layout()
```




## Estimating the coefficient of a regression model

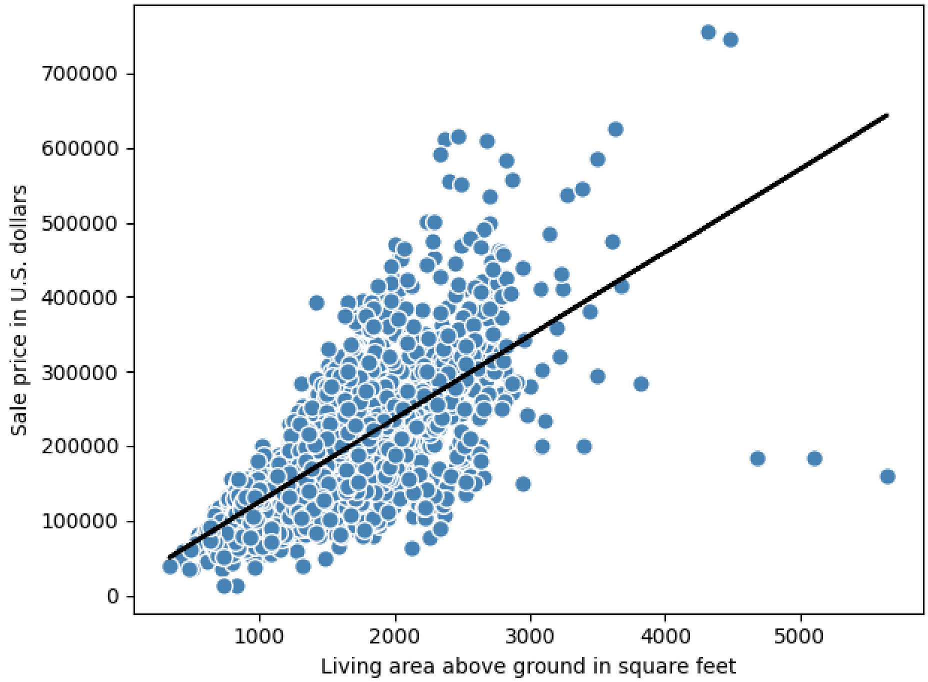
```
X = df[['Gr Liv Area']].values
y = df['SalePrice'].values
```

```
from sklearn.preprocessing import StandardScaler

sc_x = StandardScaler()
sc_y = StandardScaler()
X_std = sc_x.fit_transform(X)
y_std = sc_y.fit_transform(y[:, np.newaxis]).flatten()
```

```
from sklearn.linear_model import LinearRegression

slr = LinearRegression()
slr.fit(X, y)
y_pred = slr.predict(X)
print(f'Slope: {slr.coef_[0]:.3f}')
print(f'Intercept: {slr.intercept_:.3f}')
```

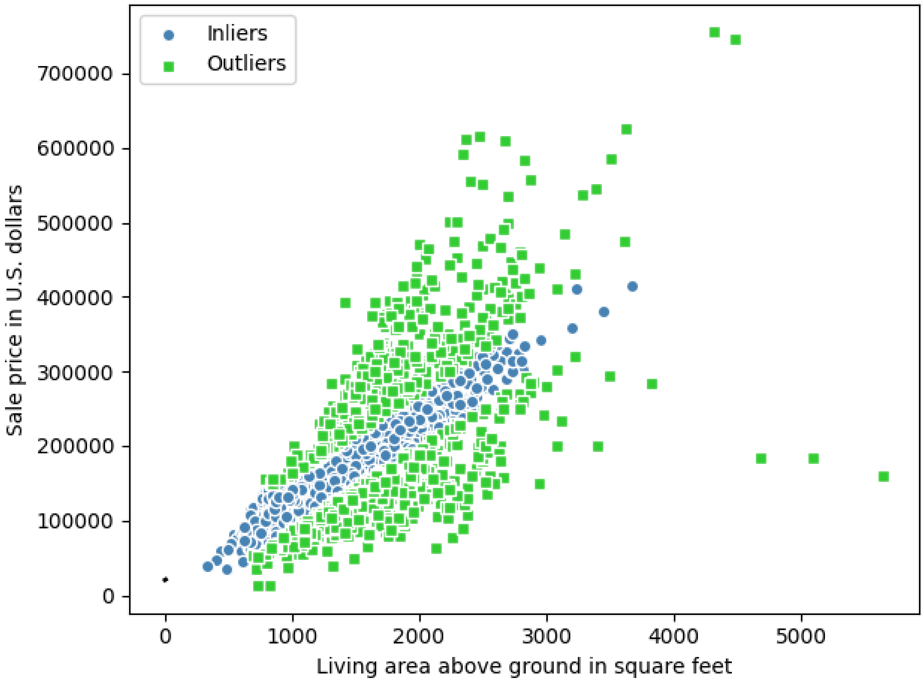
```
Slope: 111.666
Intercept: 13342.979
```

```
def lin_regplot(X, y, model):
    plt.scatter(X, y, c='steelblue', edgecolor='white', s=70)
    plt.plot(X, model.predict(X), color='black', lw=2)
    return

lin_regplot(X, y, slr)
plt.xlabel('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')

plt.tight_layout()
```




## Fitting a robust regression model using RANSAC

```
from sklearn.linear_model import RANSACRegressor

ransac = RANSACRegressor(LinearRegression(),
                         max_trials=100, # default
                         min_samples=0.95,
                         loss='absolute_error', # default
                         residual_threshold=None, # default
                         random_state=123)


ransac.fit(X, y)

inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

line_X = np.arange(3, 10, 1)
line_y_ransac = ransac.predict(line_X[:, np.newaxis])
plt.scatter(X[inlier_mask], y[inlier_mask],
            c='steelblue', edgecolor='white',
            marker='o', label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask],
            c='limegreen', edgecolor='white',
            marker='s', label='Outliers')
plt.plot(line_X, line_y_ransac, color='black', lw=2)
plt.xlabel('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')
plt.legend(loc='upper left')

plt.tight_layout()
```




```
print(f'Slope: {ransac.estimator_.coef_[0]:.3f}')
print(f'Intercept: {ransac.estimator_.intercept_:.3f}')
```

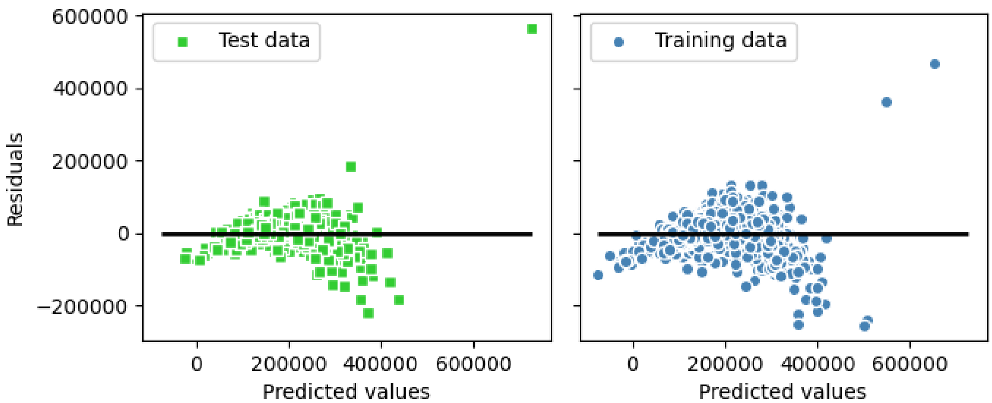
```
Slope: 106.348
Intercept: 20190.093
```

## Evaluating the performance of linear regression models

```
from sklearn.model_selection import train_test_split


target = 'SalePrice'
features = df.columns[df.columns != target]

X = df[features].values
y = df[target].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=123)
```

```
slr = LinearRegression()

slr.fit(X_train, y_train)
y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)
```

```
# Residual plot

x_max = np.max([np.max(y_train_pred), np.max(y_test_pred)])
x_min = np.min([np.min(y_train_pred), np.min(y_test_pred)])

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3), sharey=True)

ax1.scatter(y_test_pred, y_test_pred - y_test,
            c='limegreen', marker='s', edgecolor='white',
            label='Test data')
ax2.scatter(y_train_pred, y_train_pred - y_train,
            c='steelblue', marker='o', edgecolor='white',
            label='Training data')
ax1.set_ylabel('Residuals')

for ax in (ax1, ax2):
    ax.set_xlabel('Predicted values')
    ax.legend(loc='upper left')
    ax.hlines(y=0, xmin=x_min-100, xmax=x_max+100, color='black', lw=2)

plt.tight_layout()
```




```
from sklearn.metrics import mean_squared_error

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
print(f'MSE train: {mse_train:.2f}')
print(f'MSE test: {mse_test:.2f}')
```

```
MSE train: 1497216245.85
MSE test: 1516565821.00
```

```
from sklearn.metrics import mean_absolute_error

mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)
print(f'MAE train: {mae_train:.2f}')
print(f'MAE test: {mae_test:.2f}')
```

```
MAE train: 25983.03
MAE test: 24921.29
```

```
from sklearn.metrics import r2_score

r2_train = r2_score(y_train, y_train_pred)
r2_test =r2_score(y_test, y_test_pred)
print(f'R^2 train: {r2_train:.2f}')
print(f'R^2 test: {r2_test:.2f}')
```

```
R^2 train: 0.77
R^2 test: 0.75
```

## Apply regularization to regression

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=1.0)
lasso.fit(X_train, y_train)
y_train_pred = lasso.predict(X_train)
y_test_pred = lasso.predict(X_test)
print(lasso.coef_)
```

```
[26251.38276394   804.70816337    41.94651964 11364.80761309
    55.67855548]
```

```
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
print(f'MSE train: {train_mse:.3f}, test: {test_mse:.3f}')

train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
print(f'R^2 train: {train_r2:.3f}, {test_r2:.3f}')
```

```
MSE train: 1497216262.014, test: 1516576825.348
R^2 train: 0.769, 0.752
```

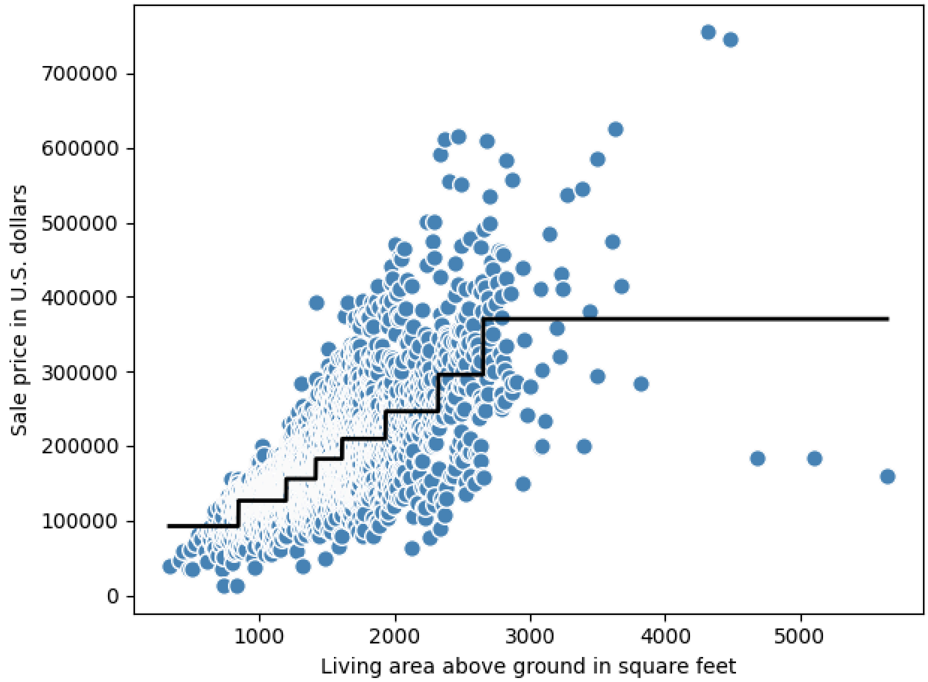
```
from sklearn.linear_model import Lasso # L1

lasso = Lasso(alpha=1.0)
```

```
from sklearn.linear_model import Ridge # L2

ridge = Ridge(alpha=1.0)
```

```
from sklearn.linear_model import ElasticNet # L3

elanet = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

## Dealing with nonlinear relationships using decision tree

```
from sklearn.tree import DecisionTreeRegressor

X = df[['Gr Liv Area']].values
y = df['SalePrice'].values

tree = DecisionTreeRegressor(max_depth=3)
tree.fit(X, y)
sort_idx = X.flatten().argsort()

lin_regplot(X[sort_idx], y[sort_idx], tree)
plt.xlabel('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')

plt.tight_layout()
```




## Your work

Use your own dataset or the **Boston Housing Dataset** below.

- Preprocess the dataset, e.g. drop null
- Explore the correlation of features with scatterplotmatrix or pairplot. Find if there is any highly correlated feature.
- Select the independent (low correlated) features as the input of your multiple regression model.
- Split dataset, standardize features and run simple linear regression model.
- Evaluate model with MSE, MAE and R-square.

**Data Dictionary**. http://lib.stat.cmu.edu/datasets/boston

**Submit your notebook in PDF to BrightSpace by 4/20 11:59 pm**.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
data_url = "http://lib.stat.cmu.edu/datasets/boston"
df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
```

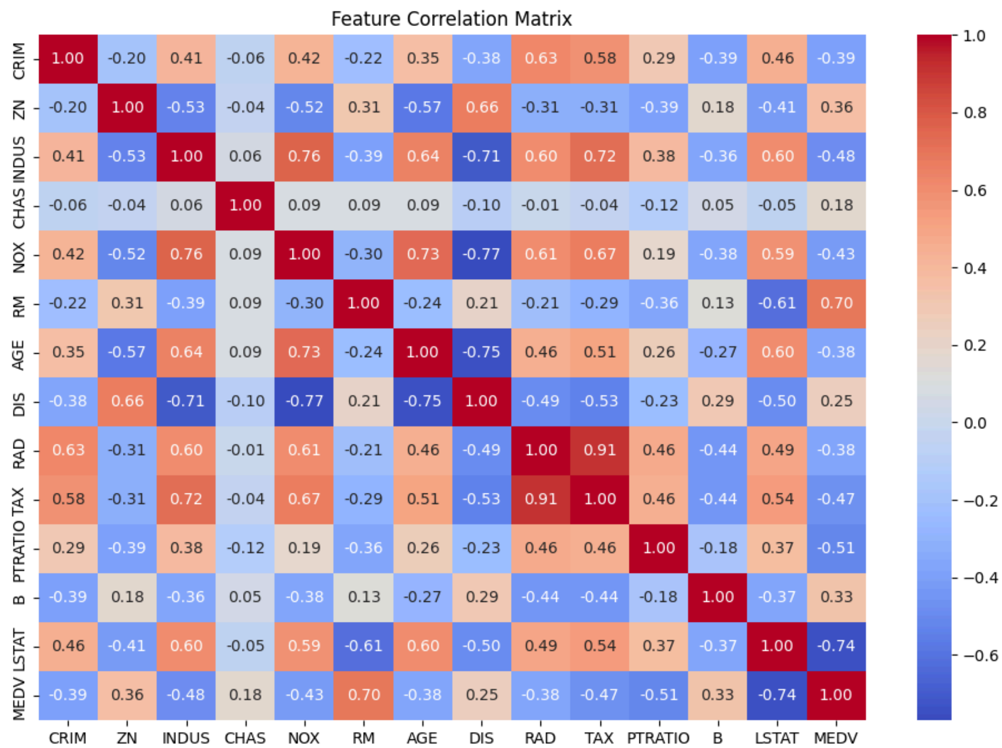
```
even_rows = df.values[::2, :]  # First part of each data point
odd_rows = df.values[1::2, :]  # Second part with target

X = np.hstack([even_rows[:, :11], odd_rows[:, :2]])  # 13 features
y = odd_rows[:, 2]  # MEDV target (3rd column of odd rows)
```

```
feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
df = pd.DataFrame(X, columns=feature_names)
df['MEDV'] = y
```

```
print("Null values:", df.isnull().sum().sum())
#df = df.dropna()
#print("Null values after:", df.isnull().sum().sum())
```

```
Null values: 0
```

```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Feature Correlation Matrix')
plt.show()
```




```
corr_matrix = df.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [column for column in upper.columns if any(upper[column] > 0.8)]
selected_features = [col for col in df.columns if col not in to_drop and col != 'MEDV']

print("Selected features:", selected_features)
```

```
Selected features: ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'PTRATIO', 'B', 'LSTAT']
```

```
X = df[selected_features].values
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)

y_pred = lr.predict(X_test_scaled)

print("\nModel Evaluation:")
print(f"MSE: {mean_squared_error(y_test, y_pred):.2f}")
print(f"MAE: {mean_absolute_error(y_test, y_pred):.2f}")
print(f"R²: {r2_score(y_test, y_pred):.2f}")
```

```
Model Evaluation:
MSE: 22.47
MAE: 3.24
R²: 0.70
```

```
coefficients = pd.Series(lr.coef_, index=selected_features)
plt.figure(figsize=(10, 6))
coefficients.sort_values().plot.barh()
plt.title('Feature Importance (Linear Regression Coefficients)')
plt.show()
```