assignment 1

[ ] ↳ 5 cells hidden

assignment 2

```
from sklearn.datasets import fetch_california_housing
boston = fetch_california_housing()
```

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

# Load the Boston Housing dataset
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['MEDV'] = boston.target
df
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MEDV
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
...	...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09	0.781
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21	0.771
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22	0.923
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32	0.847
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24	0.894

20640 rows × 9 columns

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

```
# Check for missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
 8   MEDV        20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

```
#no null values
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max Scaling (scales to a range of 0 to 1)
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df)

# Standardization (scales to a mean of 0 and standard deviation of 1)
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

```
from sklearn.model_selection import train_test_split

# Assuming 'X' contains your features and 'y' contains your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Linear Regression model
from sklearn.linear_model import LinearRegression

model_lr = LinearRegression()

# Train the model on the training data
model_lr.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_lr = model_lr.predict(X_test)
```

```
#Decision Tree Regressor model
from sklearn.tree import DecisionTreeRegressor

model_dt = DecisionTreeRegressor(random_state=42)

# Train the model on the training data
model_dt.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_dt1 = model_dt.predict(X_test)
```

```
#Random Forest Regressor model
from sklearn.ensemble import RandomForestRegressor

model_rf = RandomForestRegressor(random_state=42)

# Train the model on the training data
model_rf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_rf2 = model_rf.predict(X_test)
```

```
#Random Forest Regressor model
from sklearn.ensemble import RandomForestRegressor

model_rf = RandomForestRegressor(random_state=42)

# Train the model on the training data
model_rf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_rf3 = model_rf.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Evaluate the models
mse_lr = mean_squared_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mse_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

# Print the evaluation metrics for each model
print("Linear Regression:")
print("MSE:", mse_lr)
print("RMSE:", rmse_lr)
print("MAE:", mae_lr)
print("R-squared:", r2_lr)
```

```
Linear Regression:
MSE: 0.12907350144188584
RMSE: 0.3592680078185168
MAE: 0.3125595409266851
R-squared: -0.4341500160209537
```

```
# Decision Tree
mse_dt = mean_squared_error(y_test, y_pred_dt)
rmse_dt = np.sqrt(mse_dt)
mae_dt = mean_absolute_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)

# Random Forest
```

```
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Print the evaluation metrics for Decision Tree and Random Forest
print("Decision Tree:")
print("MSE:", mse_dt)
print("RMSE:", rmse_dt)
print("MAE:", mae_dt)
print("R-squared:", r2_dt)

print("\nRandom Forest:")
print("MSE:", mse_rf)
print("RMSE:", rmse_rf)
print("MAE:", mae_rf)
print("R-squared:", r2_rf)
```

```
Decision Tree:
MSE: 0.25
RMSE: 0.5
MAE: 0.25
R-squared: -1.7777777777777772

Random Forest:
MSE: 0.19866999999999999
RMSE: 0.4457241299279185
MAE: 0.35
R-squared: -1.207444444444444
```