```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Importing sklearn Libraries
from sklearn import datasets
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
X = np.arange(1, 25).reshape(12, 2)
y = np.array([0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=4, random_state=4)
X_train
X_test
y_train
y_test
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x - 5 + rng.randn(50)
plt.scatter(x, y, c='b');
model = LinearRegression(fit_intercept=True)
model.fit(x[:, np.newaxis], y)
xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])
plt.scatter(x, y, c='b')
plt.plot(xfit, yfit, 'k');
def PolynomialRegression(degree=2, **kwargs):
return make_pipeline(PolynomialFeatures(degree), LinearRegression(**kwargs))
def make_data(N, err=1.0, rseed=1):
# randomly sample the data
rng = np.random.RandomState(rseed)
X = rng.rand(N, 1) ** 2
y = 10 - 1. / (X.ravel() + 0.1)
if err > 0:
y += err * rng.randn(N)
```

```
return X, y
X, y = make_data(40)
fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)
for i, degree in enumerate([2, 9]):
N, train_lc, val_lc = learning_curve(PolynomialRegression(degree),
X, y, cv=7,
train_sizes=np.linspace(0.3, 1, 25))
ax[i].plot(N, np.mean(train_lc, 1), color='blue', label='training score')
ax[i].plot(N, np.mean(val_lc, 1), color='red', label='validation score')
ax[i].hlines(np.mean([train_lc[-1], val_lc[-1]]), N[0], N[-1],
color='gray', linestyle='dashed')
ax[i].set_ylim(0, 1)
ax[i].set_xlim(N[0], N[-1])
ax[i].set_xlabel('training size')
ax[i].set_ylabel('score')
ax[i].set_title('degree = {0}'.format(degree), size=14)
ax[i].legend(loc='best')
auto = pd.read_csv("/content/auto-mpg.csv")
print(auto.columns)
auto.info()
auto.isna().sum()
plt.style.use('ggplot')
sns.pairplot(auto)
plt.figure(figsize=(8, 8))
sns.heatmap(auto.corr(), annot=True, linewidth=0.5, center=0)
plt.show()
auto.dtypes
auto['horsepower'].unique()
auto = auto[auto['horsepower'] != '?']
auto['horsepower'].unique()
auto['horsepower'] = auto['horsepower'].astype(float)
auto.dtypes
X = auto[['displacement', 'horsepower', 'acceleration', 'model-year']]
# Target feature
y = auto['mpg']
X.head()
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.33, random_state=
101)
# Pridiction features
# Changed column names to match the actual names in the DataFrame
X = auto[['displacement', 'horsepower', 'acceleration', 'model-year']]
# Target feature
y = auto['mpg']
# Handle Missing Values: Option 1 - Drop rows with NaN
X.dropna(inplace=True) # Remove rows with any missing values in X
y = y[X.index] # Update y to keep only the corresponding rows after dropping from X
X.head()
X_test = auto
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
X = auto[['displacement', 'horsepower', 'acceleration', 'model-year']]
# Target feature
y = auto['mpg']
# Handle Missing Values: Option 1 - Drop rows with NaN (You already did this but it
seems X_test still has NaN)
# X.dropna(inplace=True)
# y = y[X.index]
# Instead, use imputation to fill missing values
imputer = SimpleImputer(strategy='mean') # Replace NaN with the mean of the column
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns) # Apply imputer and
keep column names
# Split 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) #
Adjust test_size and random_state as need
# Instantiating LinearRegression() Model
lr = LinearRegression()
# Fit the model
lr.fit(X_train, y_train)
# Making Predictions
pred = lr.predict(X_test)
# Evaluating Model's Performance
print('Mean Absolute Error:', mean_absolute_error(y_test, pred))
```

```
print('Mean Squared Error:', mean_squared_error(y_test, pred))
print('Mean Root Squared Error:', np.sqrt(mean_squared_error(y_test, pred)))
print('Coefficient of Determination:', r2_score(y_test, pred))
pred = lr.predict(X_test)
print('Predicted fuel consumption(mpg):', pred[2])
print('Actual fuel consumption(mpg):', y_test.values[2])
df = pd.read_csv('/content/Real estate.csv')
#dropping columns
# YOUR CODE HERE
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# ... (Code for loading, dropping columns, checking for null values, and scatter plot as before) ...
# Prepare the data for training
X = df[['X2 house age']] # Predictor variable
y = df['Y house price of unit area'] # Target variable
# 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)
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
# Plot the predicted values against the actual values
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('House Age')
plt.ylabel('House Price per Unit Area')
```

```
plt.title('Actual vs. Predicted House Prices')
plt.legend()
plt.grid(True)
plt.show()
import pandas as pd
import os
# ... (Code for loading and dropping columns as before) ...
# Check for null values
null_values = df.isnull().sum()
# Print the results
print("Null values in each column:")
print(null_values)
# Check if there are any null values at all
if df.isnull().values.any():
print("\nThere are null values in the dataframe.")
else:
print("\nThere are no null values in the dataframe.")
import pandas as pd
import matplotlib.pyplot as plt
import os
# ... (Code for loading, dropping columns, and checking for null values as before) ...
# Create a scatter plot
plt.figure(figsize=(8, 6)) # Adjust figure size if needed
plt.scatter(df['X2 house age'], df['Y house price of unit area'])
plt.xlabel('House Age')
plt.ylabel('House Price per Unit Area')
plt.title('Scatter Plot of House Age vs. House Price')
plt.grid(True)
# Separating the data into independent and dependent variables
# YOUR CODE HERE
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# ... (Code for loading, dropping columns, checking for null values, and scatter plot as
before) ...
```

```
# Prepare the data for training
X = df[['X2 house age']] # Predictor variable
y = df['Y house price of unit area'] # Target variable
# 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)
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
# Plot the predicted values against the actual values
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('House Age')
plt.ylabel('House Price per Unit Area')
plt.title('Actual vs. Predicted House Prices')
plt.legend()
plt.grid(True)
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# ... (Code for loading, dropping columns, checking for null values, and scatter plot as before) ...
# Prepare the data for training
X = df[['X2 house age']] # Predictor variable
y = df['Y house price of unit area'] # Target variable
# 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)
```

```
# YOUR CODE HERE
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# ... (Code for loading, dropping columns, checking for null values, splitting data, and scatter plot as before) ...
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train) # This line fits the model to the training data
# Data scatter of predicted values
# YOUR CODE HERE
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# ... (Code for loading, dropping columns, checking for null values, splitting data, and scatter plot as before) ...
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
# Plot the predicted values against the actual values
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
```

```
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('House Age')
plt.ylabel('House Price per Unit Area')
plt.title('Actual vs. Predicted House Prices')
plt.legend()
plt.grid(True)
plt.show()
n_samples, n_features = 15, 10
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)
rdg = linear_model.Ridge(alpha = 0.5) # instantiate Ridge regressor
rdg.fit(X, y)
rdg.score(X,y)
Lreg = linear_model.Lasso(alpha = 0.5)
Lreg.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
Lreg.predict([[0,1]])
#weight vectors
Lreg.coef
Lreg.intercept_
Lreg.n_iter_
ENreg = linear_model.ElasticNet(alpha = 0.5,random_state = 0)
ENreg.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
ENreg.predict([[0,1]])
#weight vectors
ENreg.coef_
ENreg.intercept_
ENreg.n_iter_
Lreg = linear_model.Lasso(alpha = 0.25)
Lreg.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
Lreg.coef_ #weight vectors
Lreg = linear_model.Lasso(alpha = 0.5)
Lreg.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
Lreg.coef_ #weight vectors
Lreg = linear_model.Lasso(alpha = 0.75)
Lreg.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
Lreg.coef_ #weight vectors
```