# Data Preprocessing

In this notebook, let's practice the regular data preprocessing techniques.

**The regular steps for data preprocessing are:**

1. removing missing value, removing duplicates, split_train_test, imputation, encoding categorical features and labels, scaling features, feature selection or extraction
2. sometimes you may also split_train_test after imputation, encoding and scaling, but it's the best practice to split dataset beforehand, as it will avoid data leakage.

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

```
df = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/refs/heads/master/penguins.csv')
df.head()
```

| | species | island | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | sex |
|---|---|---|---|---|---|---|---|
| 0 | Adelie | Torgersen | 39.1 | 18.7 | 181.0 | 3750.0 | MALE |
| 1 | Adelie | Torgersen | 39.5 | 17.4 | 186.0 | 3800.0 | FEMALE |
| 2 | Adelie | Torgersen | 40.3 | 18.0 | 195.0 | 3250.0 | FEMALE |
| 3 | Adelie | Torgersen | NaN | NaN | NaN | NaN | NaN |
| 4 | Adelie | Torgersen | 36.7 | 19.3 | 193.0 | 3450.0 | FEMALE |

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

```
df.info() # check data type, missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

```
df.describe() # check the statistics to see any outilers and extreme scales
```

| | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g |
|---|---|---|---|---|
| count | 342.000000 | 342.000000 | 342.000000 | 342.000000 |
| mean | 43.921930 | 17.151170 | 200.915205 | 4201.754386 |
| std | 5.459584 | 1.974793 | 14.061714 | 801.954536 |
| min | 32.100000 | 13.100000 | 172.000000 | 2700.000000 |
| 25% | 39.225000 | 15.600000 | 190.000000 | 3550.000000 |
| 50% | 44.450000 | 17.300000 | 197.000000 | 4050.000000 |
| 75% | 48.500000 | 18.700000 | 213.000000 | 4750.000000 |
| max | 59.600000 | 21.500000 | 231.000000 | 6300.000000 |

# Identify missing values

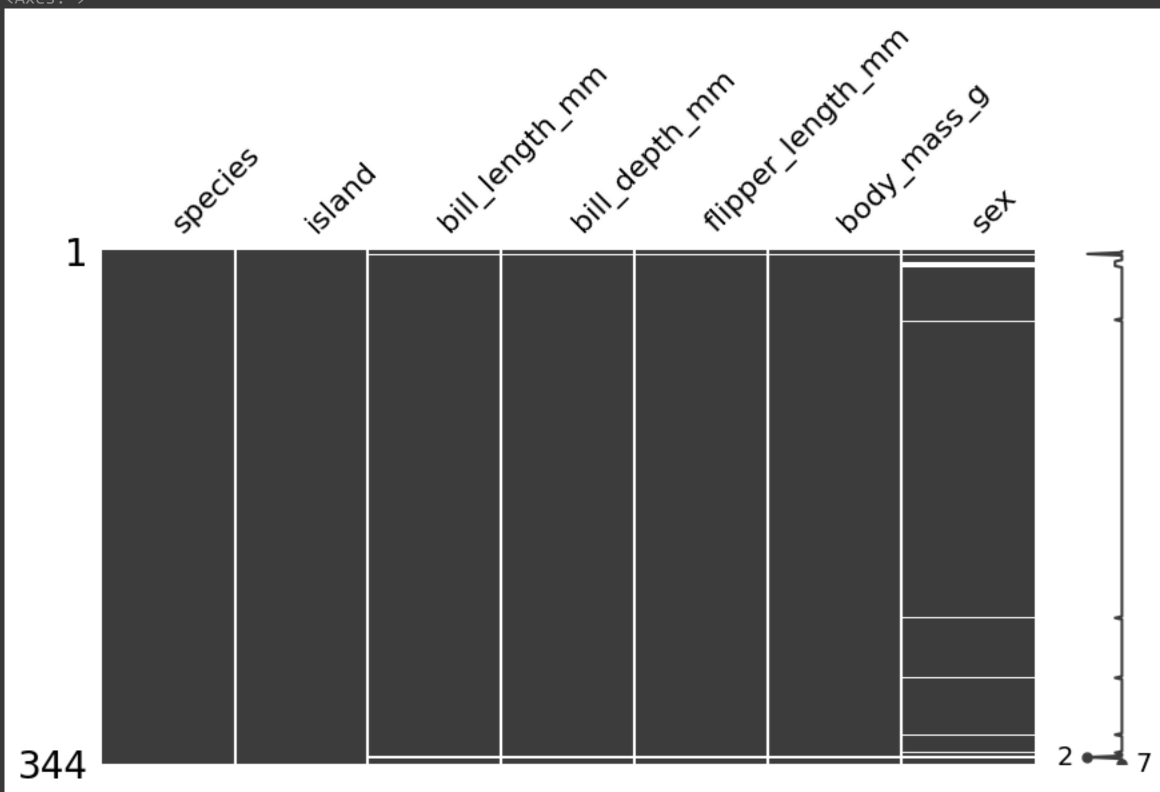
```
df.isnull().sum()
```

| | 0 |
|---|---|
| species | 0 |
| island | 0 |
| bill_length_mm | 2 |
| bill_depth_mm | 2 |
| flipper_length_mm | 2 |
| body_mass_g | 2 |
| sex | 11 |

**dtype:** int64

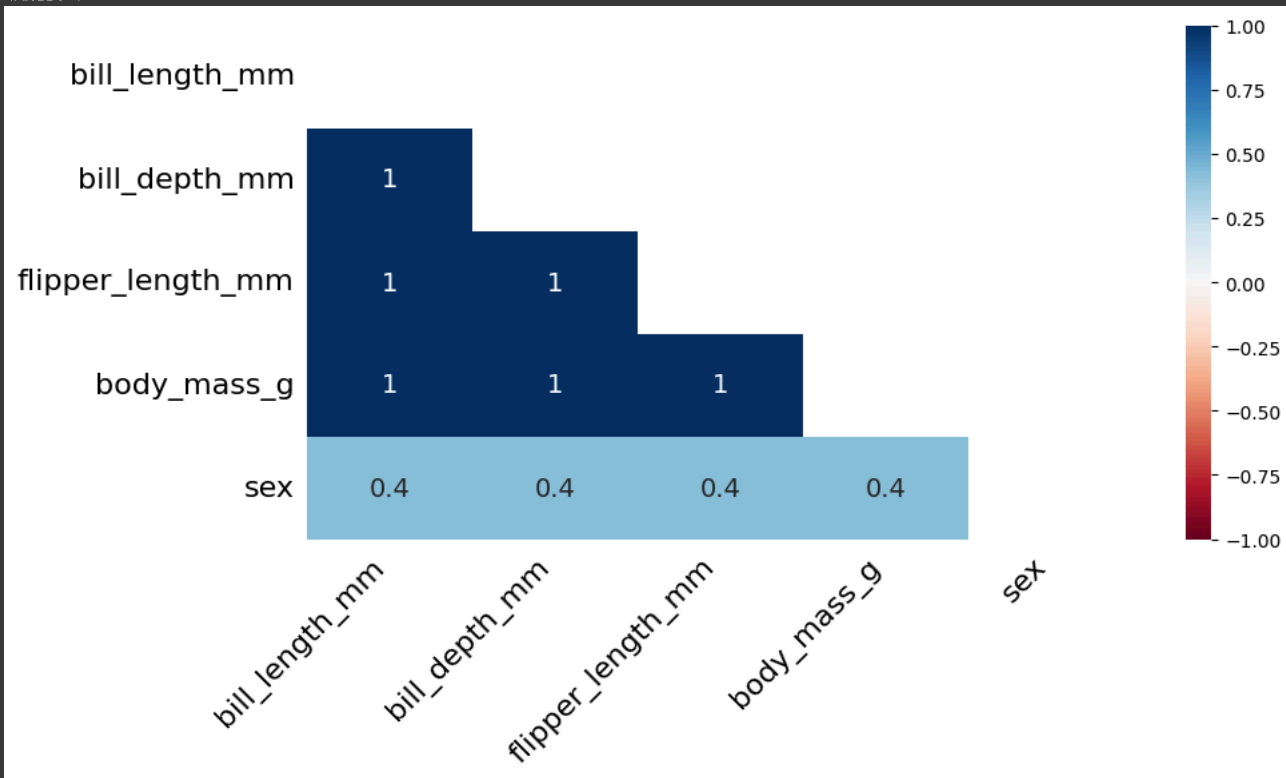
```
# Visualize the missing value with missingno
# https://github.com/ResidentMario/missingno
import missingno as msno
msno.matrix(df, figsize=(10,5))
```

<Axes: >




```
# missingno correlation heatmap measures nullity correlation:
# how strongly the presence or absence of one variable affects the presence of another
msno.heatmap(df, figsize=(10,5))
```

<Axes: >




Since the percentage of missing values is less than 5%, it's best to impute than dropping missing value.

## Split the dataset

Split the dataset after removal of missing values and duplicates. The splitting go before othe types of data preprocessing.

```
# Split the dataset into train and test set
X = df.drop('species', axis=1)
y = df['species']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

## Create Pipelines for Numeric and Categorical separately

- num: impute, standardize
- cat: impute, one-hot-encoding

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

```
# Identify numeric and categorical columns
num_cols = X_train.select_dtypes(include=np.number).columns
cat_cols = X_train.select_dtypes(include='object').columns
```

```
# Create transformers for numeric
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

```
cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False, drop = 'first'))
])
```

```
# Combine transformers using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_cols),
        ('cat', cat_transformer, cat_cols)
    ])

# Apply transformations
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

```
# Get the feature names after OneHotEncoding
cat_feature_names = preprocessor.named_transformers_['cat'].named_steps['onehot'].get_feature_names_out(cat_cols)

# Combine numerical and categorical feature names
all_feature_names = num_cols.tolist() + cat_feature_names.tolist()

# Convert back to DataFrame
X_train_processed_df = pd.DataFrame(X_train_processed, columns=all_feature_names)
X_test_processed_df = pd.DataFrame(X_test_processed, columns=all_feature_names)
```

```
X_train_processed_df.head()
```

| | bill_length_mm | bill_depth_mm | flipper_length_mm | body_mass_g | island_Dream | island_Torgersen | sex_MALE |
|---|---|---|---|---|---|---|---|
| 0 | -1.517846 | -0.437373 | -0.433589 | -1.064468 | 0.0 | 0.0 | 0.0 |
| 1 | 0.539212 | -0.842410 | 0.970707 | 1.181983 | 0.0 | 0.0 | 1.0 |
| 2 | -0.844296 | 1.284033 | -0.433589 | 0.589170 | 0.0 | 1.0 | 1.0 |
| 3 | -0.662255 | 0.018293 | -0.574019 | -1.002067 | 1.0 | 0.0 | 0.0 |
| 4 | 0.957905 | -1.044928 | 1.883499 | 1.618794 | 0.0 | 0.0 | 1.0 |

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

```
# Encode the label
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
```

```
y_train[:10]
```

```
array([0, 2, 0, 0, 2, 2, 2, 0, 0, 0])
```

## Features Scaling

## Features Selection

1.Pairplot & corr matrix

2.L1 regularization (L2 is used to prevent shrink features and avoid overfitting, not for feature selection (i.e. zero out features))

3.Sequential Backward Selection

4.Random Forest, find feature importances

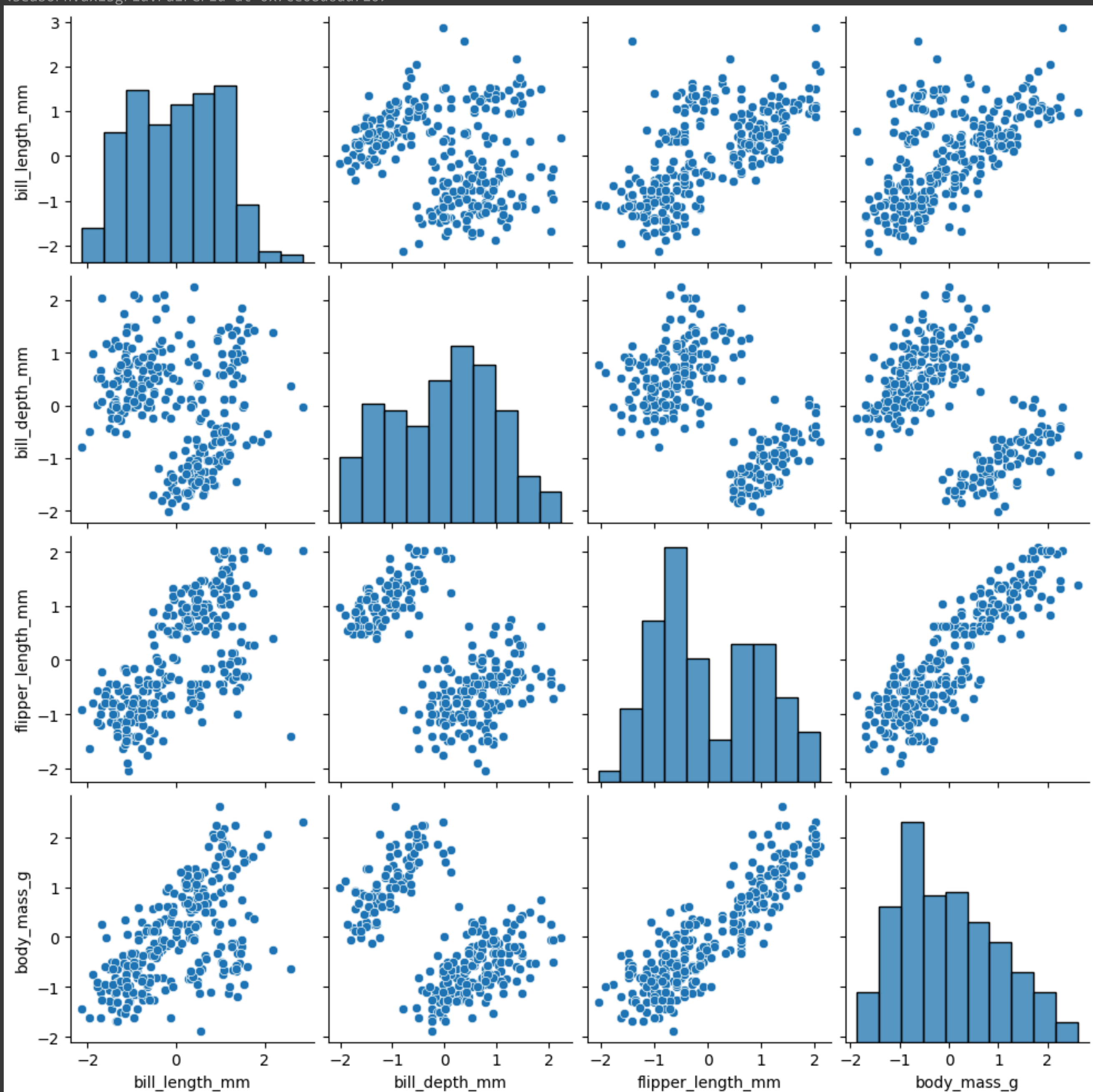
## Pairplot & Corr Matrix

```
# To make it simple, we consider only numeric features, excl. encoded features

X_train_fs = X_train_processed_df[num_cols]
X_test_fs = X_test_processed_df[num_cols]
```

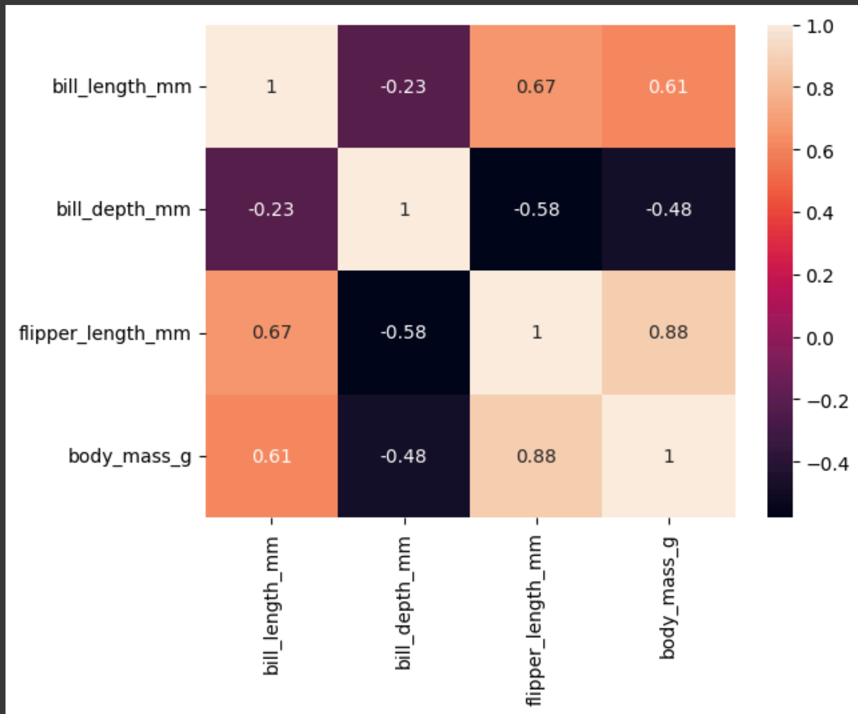
```
# Use seaborn pairplot to visualize the correlation of feature pairs
sns.pairplot(X_train_fs)
```

<seaborn.axisgrid.PairGrid at 0x7ce08d6ad710>



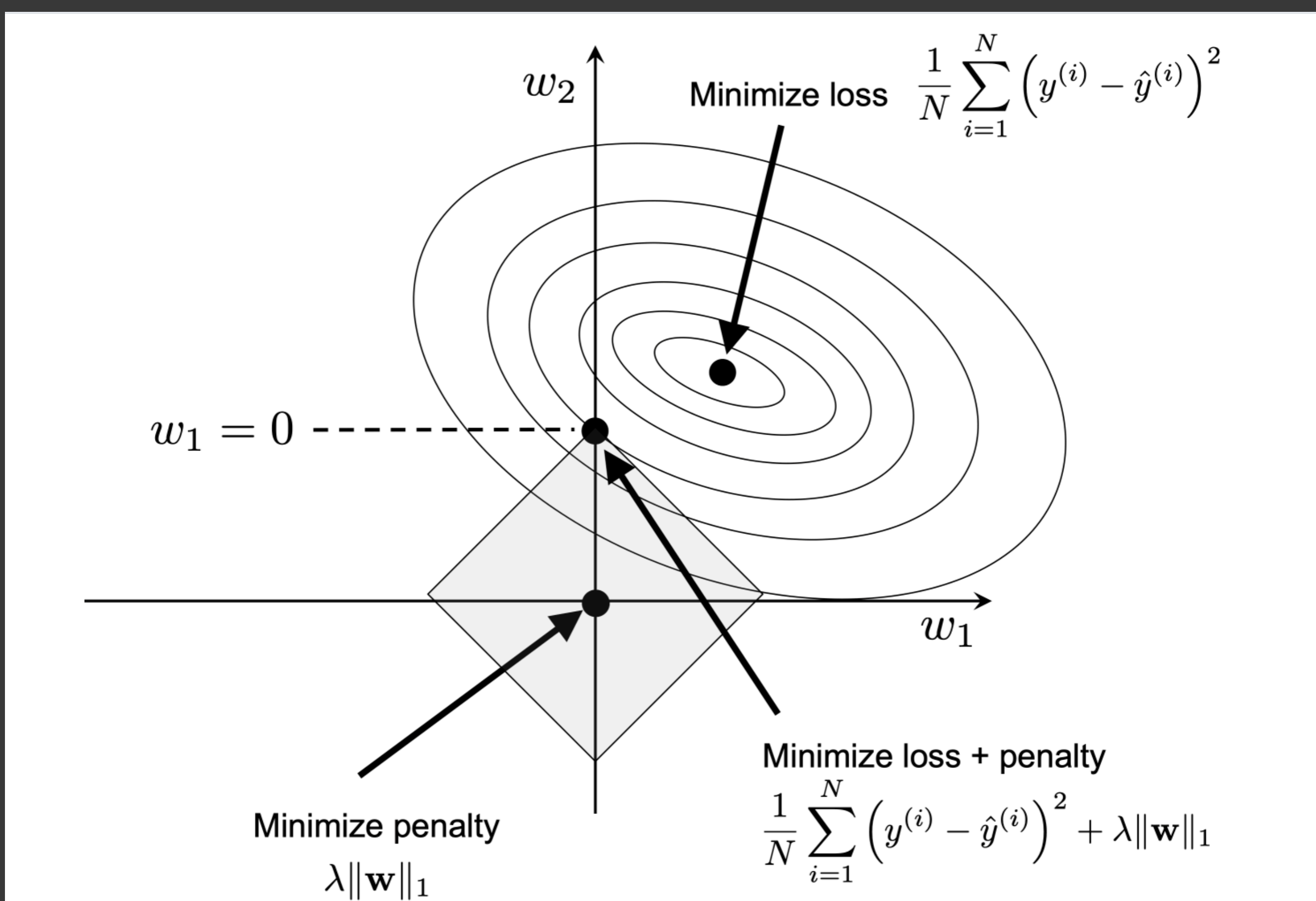

```
# Use the corr_matrix = df.corr() to calculate the corr
# Visual the corr matrix sns.heatmap(corr_matrix, annot=True)
corr_matrix = X_train_fs.corr()
sns.heatmap(corr_matrix, annot=True)
```

<Axes: >




body_mass is highly correlated with flipper_length and bill_length. body_mass can be dropped for redundancy. Let's explore further with L1 regularizatioin.

## L1 Regularization



```
# Logistic Regression with L1
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1', C=1.0, solver='liblinear')
# Note that C=1.0 is the default. You can increase or decrease it to make the regulariztion effect weaker or stronger, respectively.

lr.fit(X_train_fs, y_train)
print('Training accuracy:', lr.score(X_train_fs.iloc[:,:4], y_train))
print('Test accuracy:', lr.score(X_test_fs, y_test))
```

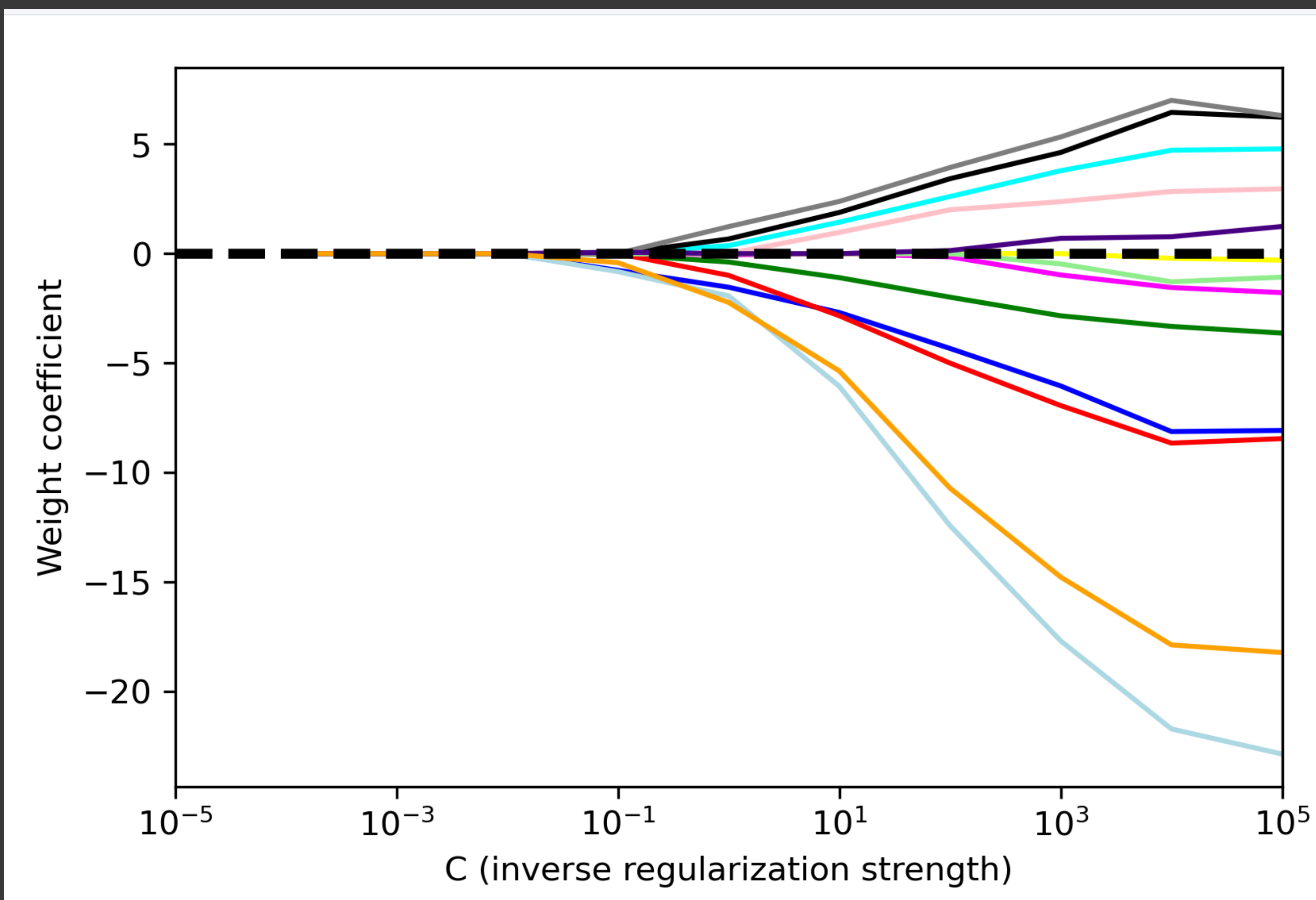
```
Training accuracy: 0.9963636363636363
Test accuracy: 0.9710144927536232
```

```
X_train_fs.columns
```

```
Index(['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'], dtype='object')
```

```
lr.coef_
```

```
array([[-6.17307023,  2.82971696,  0.        ,  0.59704262],
       [ 5.49969293,  0.37561118, -0.45586471, -4.60292283],
       [ 0.        , -3.13312682,  1.72843245,  1.98896991]])
```




## Sequential Backward Selection (SBS)

L1 is used for regularized models only. For unregularized models, such as KNN, Decision Tree, etc, use SBS as an alternative.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_fs, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
from sklearn.feature_selection import SequentialFeatureSelector

# Create a Sequential Backward Selection (SBS) object
sbs = SequentialFeatureSelector(
    knn,
    n_features_to_select='auto',  # Automatically select the best number of features
    direction='backward',
    scoring='accuracy',  # Or any other suitable scoring metric
    cv=5  # Number of cross-validation folds
)

# Fit the SBS object to the data
sbs.fit(X_train_fs, y_train)
```

```
▸ SequentialFeatureSelector
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

```
# Get the selected feature indices
selected_feature_indices = sbs.get_support(indices=True)

# Get the selected feature names
selected_feature_names = X_train_fs.columns[selected_feature_indices]
```

```
# Print the selected feature names
print("Selected Features:")
print(selected_feature_names)

# Optionally, transform the data to use only the selected features
X_train_fs_selected = sbs.transform(X_train_fs)
```

```
Selected Features:
Index(['bill_length_mm', 'bill_depth_mm'], dtype='object')
```

## Random Forest

Use random forest to rank the features by importances, then choose a subset of features for modeling training

```
from sklearn.ensemble import RandomForestClassifier

# Create a Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)  # Adjust parameters as needed

# Fit the Random Forest model to the data
rf.fit(X_train_fs, y_train)

# Get feature importances
feature_importances = rf.feature_importances_

# Create a DataFrame to display feature names and importances
feature_importance_df = pd.DataFrame({
    'Feature': X_train_fs.columns,
    'Importance': feature_importances
})

# Sort the DataFrame by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Print the feature importances
```

```
print("Feature Importances:")
print(feature_importance_df)
```

```
Feature Importances:
             Feature  Importance
0     bill_length_mm    0.415516
2  flipper_length_mm    0.300705
1      bill_depth_mm    0.198745
3        body_mass_g    0.085034
```

## Homework

- Step1: Practice Data Preprocessing on a raw dataset, which should include missing values, numeric and categorical features, so you can attemp imputation, encoding and scaling. You don't have to go over all steps. Please consider the cleaness of your dataset.
- Step2: After above process, you may perform feature selections. You may attemp L1 regularization, SBS, random forest, etc. You don't have to try all methods but just play around and find one that you feel comfortable with.

You can find medium-size raw dataset on Kaggle.com or NYCOpenData. Or you may use the Used Car dataset below.

Used Car Dataset
https://www.kaggle.com/datasets/alikalwar/uae-used-car-prices-and-features-10k-listings

**Please submit your notebook by 3/9 11:59 pm to BrightSpace.**

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Lasso, LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.ensemble import RandomForestRegressor
from google.colab import drive
```

```
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/uae_used_cars_10k.csv')
df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

| | Make | Model | Year | Price | Mileage | Body Type | Cylinders | Transmission | Fuel Type | Color | Location | Description |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | toyota | camry | 2016 | 47819 | 156500 | Sedan | 4 | Automatic Transmission | Gasoline | Black | Dubai | 2016 toyota camry with Rear camera, Leather se... |
| 1 | kia | sorento | 2013 | 61250 | 169543 | SUV | 4 | Automatic Transmission | Gasoline | Grey | Abu Dhabi | 2013 kia sorento with Sunroof, Adaptive cruise... |
| 2 | mini | cooper | 2023 | 31861 | 221583 | Soft Top Convertible | 4 | Automatic Transmission | Gasoline | Grey | Dubai | 2023 mini cooper with Adaptive cruise control,... |
| 3 | nissan | altima | 2016 | 110322 | 69754 | Sedan | 4 | Automatic Transmission | Gasoline | Red | Dubai | 2016 nissan altima with Rear camera, Adaptive ... |

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

```
df.info() # check data type, missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Make          10000 non-null  object
 1   Model         10000 non-null  object
 2   Year          10000 non-null  int64
 3   Price         10000 non-null  int64
```

```
 4   Mileage       10000 non-null  int64
 5   Body Type     10000 non-null  object
 6   Cylinders     9895 non-null   object
 7   Transmission  10000 non-null  object
 8   Fuel Type     10000 non-null  object
 9   Color         10000 non-null  object
 10  Location      10000 non-null  object
 11  Description   10000 non-null  object
dtypes: int64(3), object(9)
memory usage: 937.6+ KB
```

```
df.describe() # check the statistics to see any outilers and extreme scales
```

| | Year | Price | Mileage |
|---|---|---|---|
| count | 10000.000000 | 1.000000e+04 | 10000.000000 |
| mean | 2014.472800 | 2.452345e+05 | 155161.871700 |
| std | 5.790839 | 4.709773e+05 | 83681.858983 |
| min | 2005.000000 | 7.183000e+03 | 10006.000000 |
| 25% | 2009.000000 | 5.035250e+04 | 82904.000000 |
| 50% | 2014.000000 | 1.027660e+05 | 154370.500000 |
| 75% | 2019.000000 | 2.312480e+05 | 227551.250000 |
| max | 2024.000000 | 1.468698e+07 | 299996.000000 |

```
print("Missing values:")
df.isnull().sum()
```

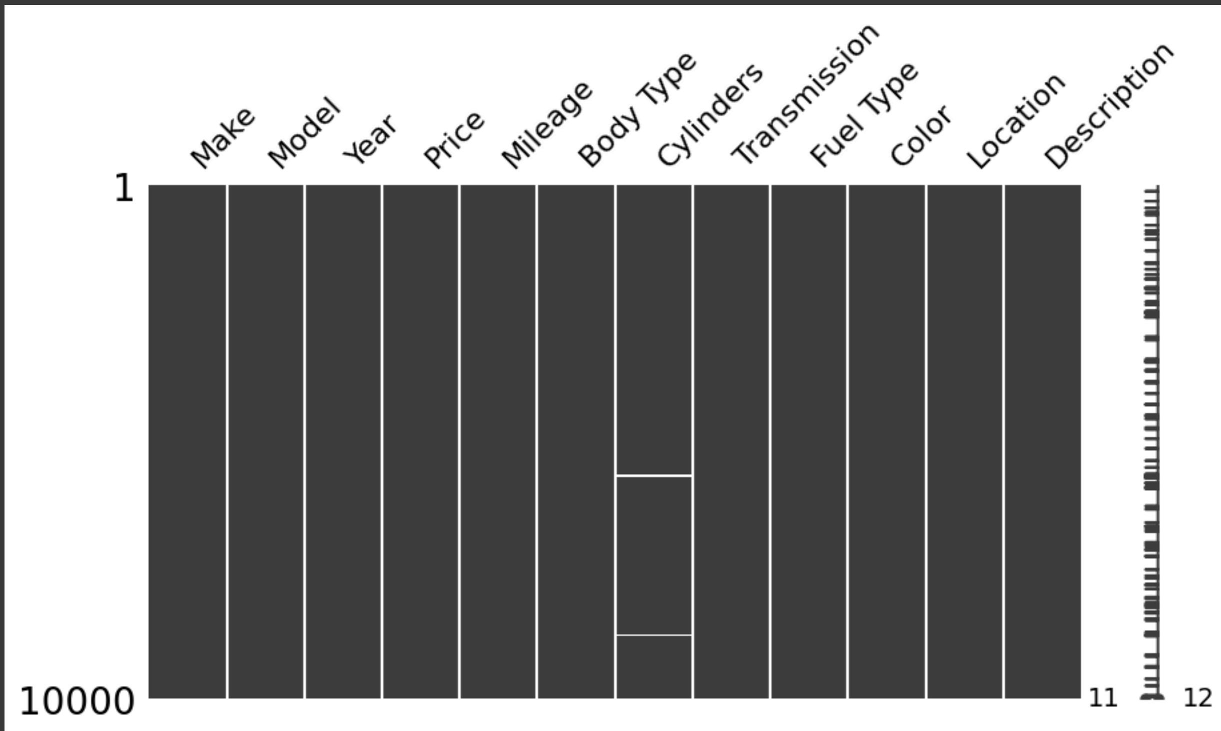
Missing values:

| | 0 |
|---|---|
| Make | 0 |
| Model | 0 |
| Year | 0 |
| Price | 0 |
| Mileage | 0 |
| Body Type | 0 |
| Cylinders | 105 |
| Transmission | 0 |
| Fuel Type | 0 |
| Color | 0 |
| Location | 0 |
| Description | 0 |

**dtype:** int64

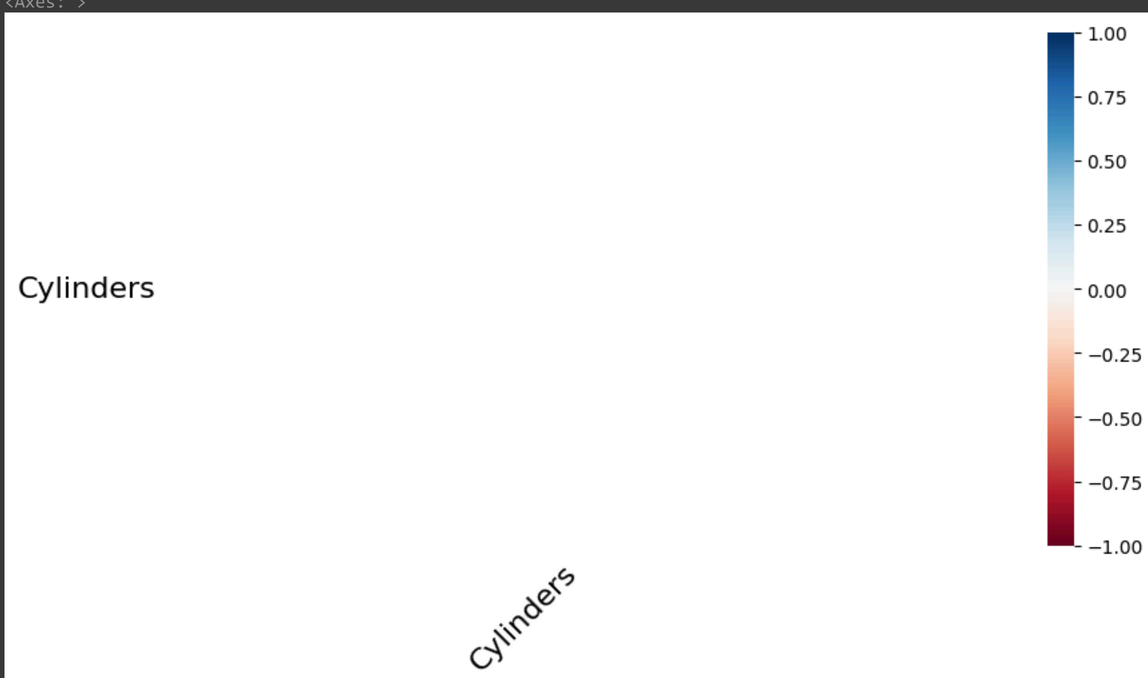
```
# Visualize the missing value with missingno
# https://github.com/ResidentMario/missingno
import missingno as msno
msno.matrix(df, figsize=(10,5))
```

<Axes: >




```
# missingno correlation heatmap measures nullity correlation:
# how strongly the presence or absence of one variable affects the presence of another
msno.heatmap(df, figsize=(10,5))
```

<Axes: >




```
X = df.drop('Price', axis=1)  # Replace 'price' with actual target column
y = df['Price']

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

```
# Identify numeric and categorical columns
num_cols = X_train.select_dtypes(include=np.number).columns
cat_cols = X_train.select_dtypes(include='object').columns
```

```
# Create transformers
num_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

```
cat_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False, drop='first'))
])
```

```
# Apply transformations
preprocessor = ColumnTransformer([
    ('num', num_transformer, num_cols),
    ('cat', cat_transformer, cat_cols)
])

X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

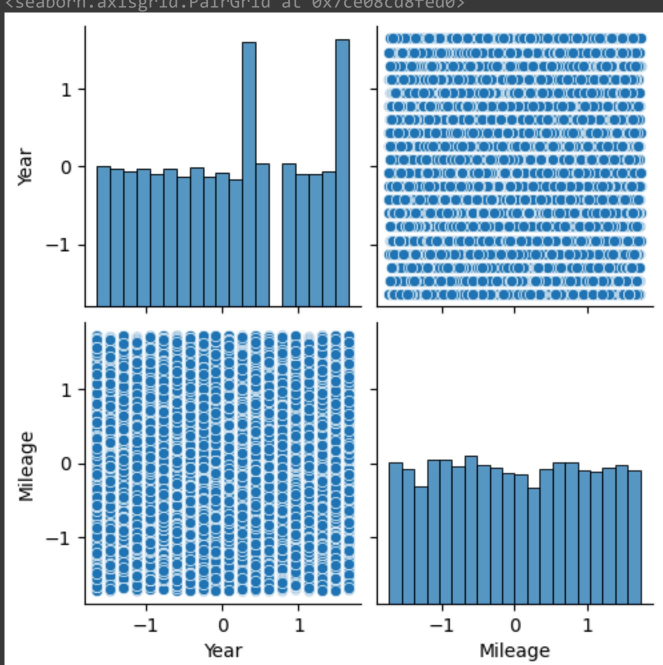
```
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarning: Found unknown categories in columns [0, 1,
  warnings.warn(
```

```
# Get feature names
cat_feature_names = preprocessor.named_transformers_['cat'].named_steps['onehot'].get_feature_names_out(cat_cols)
all_feature_names = num_cols.tolist() + cat_feature_names.tolist()
# Convert back to DataFrame
X_train_processed_df = pd.DataFrame(X_train_processed, columns=all_feature_names)
X_test_processed_df = pd.DataFrame(X_test_processed, columns=all_feature_names)
```

```
# Select only numeric features for feature selection
X_train_fs = X_train_processed_df[num_cols]
X_test_fs = X_test_processed_df[num_cols]
```

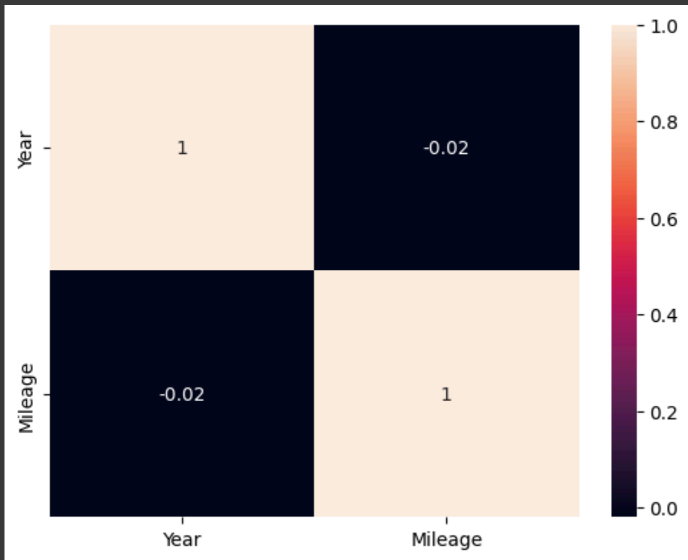
```
sns.pairplot(X_train_fs)
```

```
<seaborn.axisgrid.PairGrid at 0x7ce08cd8fed0>
```




```
# Correlation matrix and heatmap
corr_matrix = X_train_fs.corr()
sns.heatmap(corr_matrix, annot=True)
```

```
<Axes: >
```




```
lasso = Lasso(alpha=0.01)  # Adjust alpha for stronger/weaker regularization
lasso.fit(X_train_fs, y_train)
```

```
▾ Lasso
Lasso(alpha=0.01)
```

```
lasso_coef = pd.Series(lasso.coef_, index=X_train_fs.columns)
print("Lasso Selected Features:")
print(lasso_coef[lasso_coef != 0])
```

```
Lasso Selected Features:
Year      -3269.507332
Mileage    2037.622564
dtype: float64
```

```
knn = KNeighborsRegressor(n_neighbors=5)
sbs = SequentialFeatureSelector(
    knn,
    n_features_to_select="auto",  # Automatically selects best feature count
    direction="backward",
    scoring="r2",  # Adjust scoring metric for regression
    cv=5
)

sbs.fit(X_train_fs, y_train)
selected_feature_indices = sbs.get_support(indices=True)
selected_feature_names = X_train_fs.columns[selected_feature_indices]
print("SBS Selected Features:")
print(selected_feature_names)
```

```
SBS Selected Features:
Index(['Year'], dtype='object')
```

```
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train_fs, y_train)
feature_importances = rf.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X_train_fs.columns, 'Importance': feature_importances}).sort_values(by='Importance', ascenc
print("Random Forest Selected Features:")
print(feature_importance_df)
```

```
Random Forest Selected Features:
   Feature  Importance
1  Mileage    0.902282
0     Year    0.097718
```

```
# Train a Linear Regression Model
lr = LinearRegression()
lr.fit(X_train_fs, y_train)
```

```
print('Training R^2 Score:', lr.score(X_train_fs, y_train))
print('Test R^2 Score:', lr.score(X_test_fs, y_test))
```

```
Training R^2 Score: 7.135913099975966e-05
Test R^2 Score: -0.00014786921205445225
```

```
# Train a Random Forest Regressor Model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_fs, y_train)
print('Training R^2 Score (RF):', rf_model.score(X_train_fs, y_train))
print('Test R^2 Score (RF):', rf_model.score(X_test_fs, y_test))
```

```
Training R^2 Score (RF): 0.8248907204454824
Test R^2 Score (RF): -0.25943088767959677
```