Import common library

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
import seaborn as sns
import os

import sklearn
from sklearn.preprocessing import (
    StandardScaler, OneHotEncoder
)
from sklearn.model_selection import (
    train_test_split, cross_val_score, LeaveOneOut, KFold,
    cross_validate
)
from sklearn.decomposition import (
    PCA
)
from sklearn.cross_decomposition import (
    PLSRegression
)
from sklearn.metrics import (
    mean_squared_error, accuracy_score, r2_score, f1_score,
    confusion_matrix, classification_report
)
from sklearn.linear_model import (
    LinearRegression, LogisticRegression
)
from sklearn.tree import (
    _tree
)
from sklearn.ensemble import (
    RandomForestClassifier, RandomForestRegressor
)
from sklearn.neural_network import (
    MLPClassifier, MLPRegressor
)

import tensorflow as tf

from statsmodels.api import OLS, add_constant

from mlxtend.feature_selection import SequentialFeatureSelector
```

In [2]:
```
print(f'Numpy version: {np.__version__}')
print(f'Pandas version: {pd.__version__}')
print(f'Seaborn version: {sns.__version__}')
print(f'Tensorflow version: {tf.__version__}')
print(f'Scikit-learn version: {sklearn.__version__}')
```

```
Numpy version: 1.23.5
Pandas version: 2.3.0
Seaborn version: 0.13.2
Tensorflow version: 2.12.0
Scikit-learn version: 1.7.0
```

Data Cleaning

In [ ]:
```
df.dropna()
```

```
df.drop_duplicates()

df.fillna(value={'column1': 10, 'column2':df['column2'].mean()}) # fill missing value in colum
```

❌ **No — you should NOT use `.fit_transform()` every time, especially on test data.** Here's why:

✅ Correct Practice

Data Type	What to Use	Why
Training	`scaler.fit_transform(X_train)`	Learn mean/std **and** scale training data
Test	`scaler.transform(X_test)`	Use training mean/std to transform test data

❗ Why not `fit_transform()` on test data?

Using `.fit_transform(X_test)` **learns a new mean and std from test data**, which:

- **Leaks information** from test set (data leakage)
- **Makes model evaluation invalid**
- **Gives biased performance metrics**

📌 Best Workflow:

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Training
X_train_scaled = scaler.fit_transform(X_train)

# Test (or validation)
X_test_scaled = scaler.transform(X_test)
```

✅ Bottom line:

Use `.fit_transform()` **only on training data**. Use `.transform()` **on everything else** (validation, test, new data).

Let me know if you want a diagram or scikit-learn pipeline version!

Great question! The reason we use `train_test_split` **before** applying `StandardScaler` is to **prevent data leakage**.

🔑 Data Leakage Explained:

If you **scale the entire dataset before splitting**, then:

- The scaler "sees" the **test data** during `.fit()`
- It computes the **mean and standard deviation using all data**, including the test set
- This means your model **indirectly uses information from the test set during training**
- Your evaluation metrics will be **over-optimistic and invalid**

✅ Correct Order:

1. **Split the data first** using `train_test_split`
2. **Fit the scaler only on training data**
3. **Transform both training and test sets** using that scaler

```
x_train, x_test = train_test_split(x)
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

🧠 Analogy:

Imagine you're predicting exam scores, and you normalize scores using the **class average**. If you compute the average using **everyone's** scores (including the future test-takers), you're cheating!

✅ Summary:

Always `train_test_split` **first**, then scale. This ensures the model only learns from training data — just like it would in the real world.

In [ ]:

Introduction to Machine Learning (Week 1)

3 Linear Regression (Week 2, Chapter 3)

3.1 Simple Linear Regression

using statsmodel

```
getwd()
setwd('D:/machine-learning/data')
boston = read.csv('boston.csv')
model = lm('medv ~ .', data = boston)
summary(model)
```

In [9]:

```
from statsmodels.api import OLS, add_constant

boston = pd.read_csv('data/boston.csv')
# Y = medv (Median value of owner-occupied homes in $1000s)

# response variable is medv
y = boston['medv']

# predictor variables are all other columns (13 columns)
x = boston.drop('medv', axis=1)

# add constant for the intercept to predictor variables
```

```python
# If no constant added, it will pass through the origin (0,0)
x1 = add_constant(x)

# create and fit the OLS model
model = OLS(y, x1).fit()

# print the summary of the model
model.summary()
```

Out[9]:

OLS Regression Results

Dep. Variable:	medv	**R-squared:**	0.741
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	108.1
Date:	Sun, 06 Jul 2025	**Prob (F-statistic):**	6.72e-135
Time:	18:24:13	**Log-Likelihood:**	-1498.8
No. Observations:	506	**AIC:**	3026.
Df Residuals:	492	**BIC:**	3085.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	36.4595	5.103	7.144	0.000	26.432	46.487
crim	-0.1080	0.033	-3.287	0.001	-0.173	-0.043
zn	0.0464	0.014	3.382	0.001	0.019	0.073
indus	0.0206	0.061	0.334	0.738	-0.100	0.141
chas	2.6867	0.862	3.118	0.002	0.994	4.380
nox	-17.7666	3.820	-4.651	0.000	-25.272	-10.262
rm	3.8099	0.418	9.116	0.000	2.989	4.631
age	0.0007	0.013	0.052	0.958	-0.025	0.027
dis	-1.4756	0.199	-7.398	0.000	-1.867	-1.084
rad	0.3060	0.066	4.613	0.000	0.176	0.436
tax	-0.0123	0.004	-3.280	0.001	-0.020	-0.005
ptratio	-0.9527	0.131	-7.283	0.000	-1.210	-0.696
black	0.0093	0.003	3.467	0.001	0.004	0.015
lstat	-0.5248	0.051	-10.347	0.000	-0.624	-0.425

Omnibus:	178.041	**Durbin-Watson:**	1.078
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	783.126
Skew:	1.521	**Prob(JB):**	8.84e-171
Kurtosis:	8.281	**Cond. No.**	1.51e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.51e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Using scikit-learn

In [7]:
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 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    int64
 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    int64
 9   tax      506 non-null    int64
 10  ptratio  506 non-null    float64
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
dtypes: float64(10), int64(3)
memory usage: 51.5 KB
```

In [24]:
```
import pandas as pd

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder

boston = pd.read_csv('data/boston.csv')

y = boston['medv']
x = boston.drop('medv', axis=1)

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)

from sklearn.preprocessing import StandardScaler, OneHotEncoder
scale = StandardScaler()
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

x_train_scale = scale.fit_transform(x_train.select_dtypes(include=['float64', 'int64']))
x_test_scale = scale.fit_transform(x_test.select_dtypes(include=['float64', 'int64']))

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

modle = lr.fit(x_train_scale, y_train)

print(lr.coef_)

print(lr.intercept_)
```

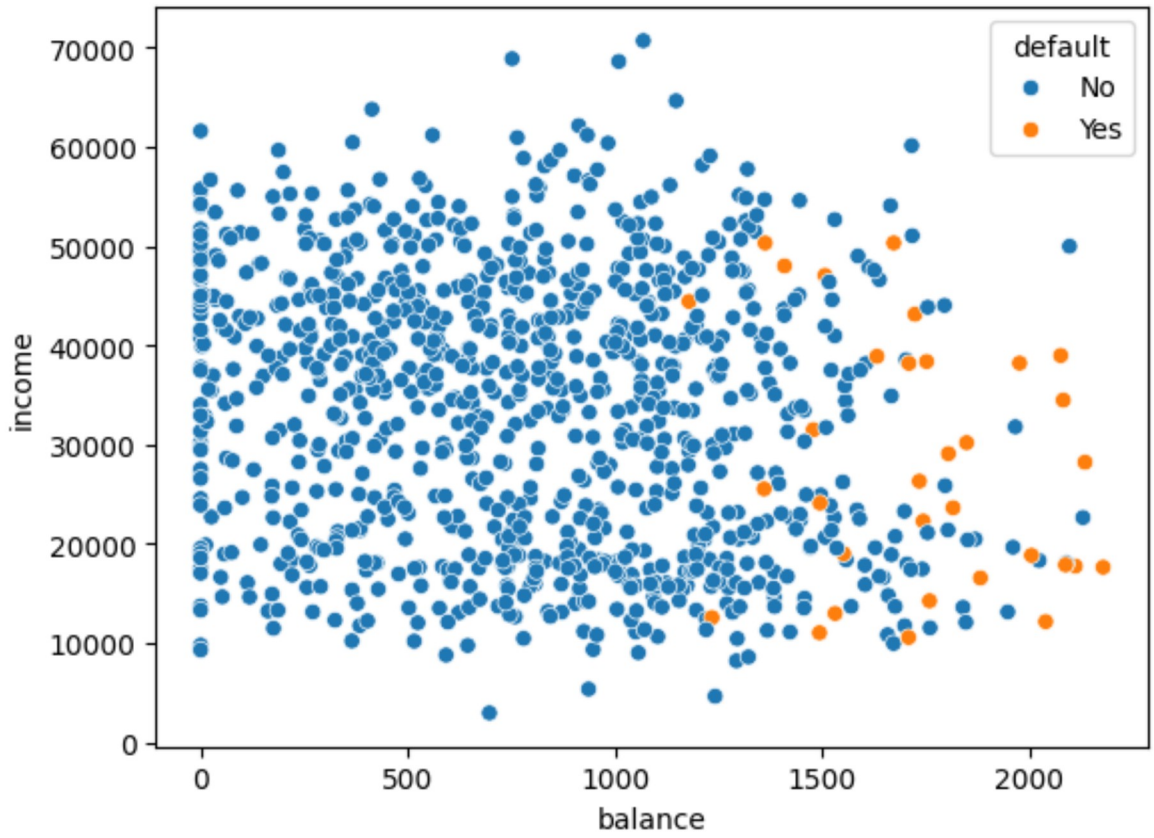
```
[-1.00213533  0.69626862  0.27806485  0.7187384  -2.0223194   3.14523956
 -0.17604788 -3.0819076   2.25140666 -1.76701378 -2.03775151  1.12956831
 -3.61165842]
22.796534653465347
```

4 Classification (Week 3, Chapter 4)

In [25]:
```
default = pd.read_csv('data/default.csv')
print(default.head(10))
print('-------------------------------------------')
print(default.shape)
```

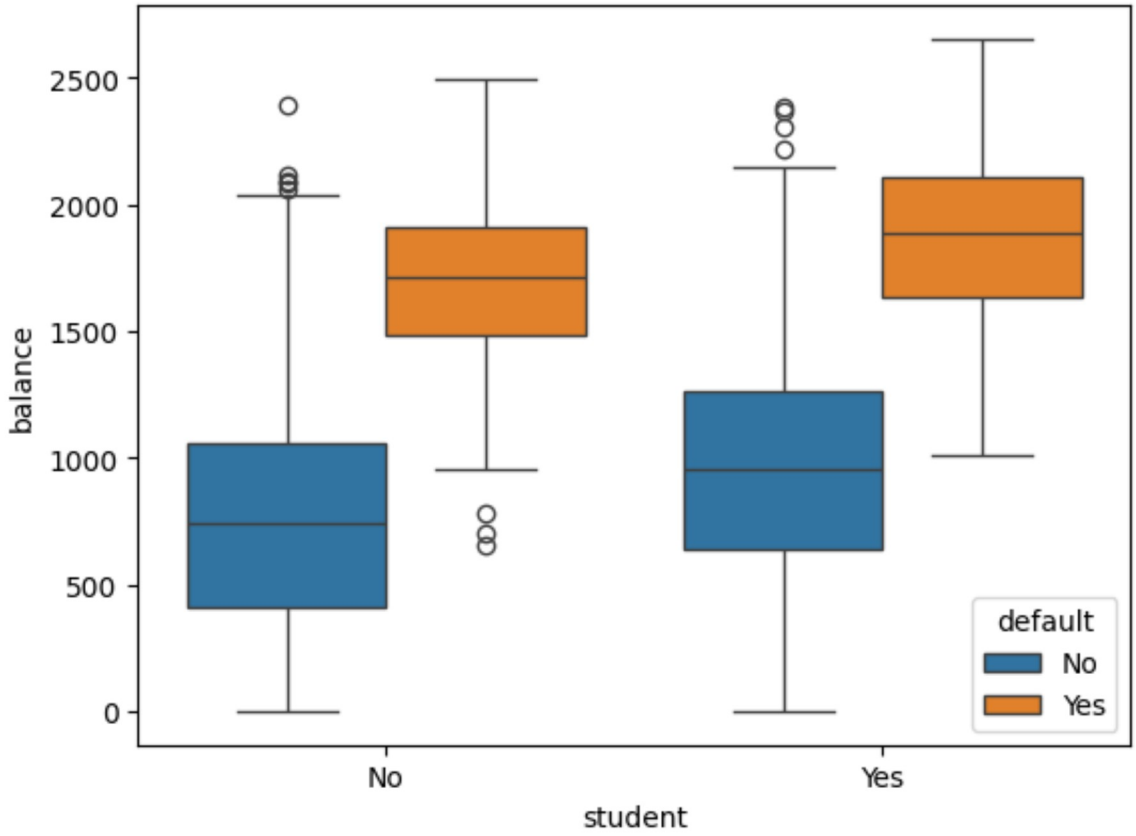
```
  default student      balance        income
0      No      No   729.526495  44361.625074
1      No     Yes   817.180407  12106.134700
2      No      No  1073.549164  31767.138947
3      No      No   529.250605  35704.493935
4      No      No   785.655883  38463.495879
5      No     Yes   919.588530   7491.558572
6      No      No   825.513331  24905.226578
7      No     Yes   808.667504  17600.451344
8      No      No  1161.057854  37468.529288
9      No      No     0.000000  29275.268293
-------------------------------------------
(10000, 4)
```

In [21]:
```
sns.scatterplot(x='balance', y='income', data=default.sample(1000), hue='default')
plt.show();
```



income does not influence whether a person default or not but balance does influence where a person with higher balance is more likely to default than a person with a lower balance.

In [22]:
```
sns.boxplot(x='student', y='balance', data=default, hue='default');
```



Student or not does not effect whether a person is a defaulter or not.

In [23]:
```
default.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   default  10000 non-null  object
 1   student  10000 non-null  object
 2   balance  10000 non-null  float64
 3   income   10000 non-null  float64
dtypes: float64(2), object(2)
memory usage: 312.6+ KB
```

In [34]:
```
x_test_cat
```

Out[34]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder


x = default.drop('default', axis=1)
y = default['default']

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=42)

scale = StandardScaler()
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
```

```
x_train_num = scale.fit_transform(x_train.select_dtypes(include=['float64']))
x_test_num = scale.fit_transform(x_test.select_dtypes(include=['float64']))

x_train_cat = ohe.fit_transform(x_train.select_dtypes(include=['object']))
x_test_cat = ohe.fit_transform(x_test.select_dtypes(include=['object']))

x_train_processed = np.hstack((x_train_num, x_train_cat))
x_test_processed = np.hstack((x_test_num, x_test_cat))

logreg = LogisticRegression()

logreg.fit(x_train_processed, y_train)

logreg.predict(x_train_processed)
```

Out[ ]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

In [ ]:
```
from sklearn.metrics import (
    confusion_matrix, accuracy_score, f1_score, classification_report
)

print("Logistic Regression performance:")
print(pd.DataFrame(confusion_matrix(y_test, logreg.predict(x_test_processed)),
                   index = ['Yes', 'No'], columns=['Yes', 'No']))

print('==========')

print(f'Accuracy: {(accuracy_score(y_test, logreg.predict(x_test_processed)) * 100):.2f}%')
```

```
Logistic Regression performance:
      Yes  No
Yes  1920  11
No     50  19
==========
Accuracy:96.95%
```

4.1 Linear Discriminant Analysis (LDA)

- Used for multiclass classification problems
- Assume all predictors come from a **gaussian distribution**

In [60]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix

lda = LinearDiscriminantAnalysis()

lda.fit(x_train_processed, y_train)

print('Linear Discrimant Analysis performance:')
print(pd.DataFrame(confusion_matrix(y_test, lda.predict(x_test_processed)),
                   index=['Yes', 'No'], columns=['Yes', 'No']))

print('==========')

print(f'Accuracy: {(accuracy_score(y_test, lda.predict(x_test_processed)) * 100):.2f}%')
```

```
Linear Discrimant Analysis performance:
      Yes  No
Yes  1925   6
No     57  12
==========
Accuracy: 96.85%
```

4.2 Quadratic Discriminant Analysis (QDA)

In [61]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

qda = QuadraticDiscriminantAnalysis()

qda.fit(x_train_processed, y_train)

print('Quadratic Discrimant Analysis performance:')
print(pd.DataFrame(confusion_matrix(y_test, qda.predict(x_test_processed)),
                   index=['Yes', 'No'], columns=['Yes', 'No']))

print('==========')

print(f'Accuracy: {(accuracy_score(y_test, qda.predict(x_test_processed)) * 100):.2f}%')
```

```
Quadratic Discrimant Analysis performance:
      Yes   No
Yes  1730  201
No     38   31
==========
Accuracy: 88.05%
```

```
c:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\discriminant_a
nalysis.py:1024: LinAlgWarning: The covariance matrix of class 0 is not full rank. Increasing
the value of parameter `reg_param` might help reducing the collinearity.
  warnings.warn(
c:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\discriminant_a
nalysis.py:1024: LinAlgWarning: The covariance matrix of class 1 is not full rank. Increasing
the value of parameter `reg_param` might help reducing the collinearity.
  warnings.warn(
```

4.3 K-Nearest Neighbor (KNN)

📘 Example Problem

You want to classify a new point using **K=3** based on the following training data:

Point	X1	X2	Class
A	1	2	0
B	2	3	0
C	3	3	1
D	6	5	1

And the **new point** to classify is:

```
New Point = (3, 2)
```

🧮 Step 1: Compute Euclidean Distance

Use the formula:

$$d = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2}$$

Point	Coordinates	Distance to (3,2)
A	(1, 2)	$\sqrt{(3-1)^2 + (2-2)^2} = \sqrt{4} =$ **2.0**
B	(2, 3)	$\sqrt{(3-2)^2 + (2-3)^2} = \sqrt{2} =$ **1.41**
C	(3, 3)	$\sqrt{(3-3)^2 + (2-3)^2} = \sqrt{1} =$ **1.0**
D	(6, 5)	$\sqrt{(3-6)^2 + (2-5)^2} = \sqrt{18} =$ **4.24**

🔍 Step 2: Find K Nearest Neighbors (K = 3)

Sorted by distance:

1. C → 1.0 → Class 1
2. B → 1.41 → Class 0
3. A → 2.0 → Class 0

📥 Step 3: Majority Vote

Among the 3 nearest neighbors:

- Class 0: 2 votes (B, A)
- Class 1: 1 vote (C)

🎯 **Predicted class = 0**

✅ Final Answer:

The new point (3, 2) is classified as Class 0.

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(x_train, y_train)

confusion_matrix(x_test, knn.predict(y_test))
```

5 Resampling Methods (Week 4, Chapter 5)

5.1 Cross Validation

5.1.1 The Validation Set Approach

The validation set approach split the data into two comparable size to validate the training data

In [62]:
```
# load data
auto = pd.read_csv('data/auto.csv')
```

```python
# split data into training and test sets
auto_train, auto_test = train_test_split(auto, test_size=0.2, random_state=42)

# define the predictor and response variables for the training set
y_train = auto_train['mpg']
x_train = auto_train['horsepower'].values.reshape(-1, 1)

# define the predictor and response variables for the test set
y_test = auto_test['mpg']
x_test = auto_test['horsepower'].values.reshape(-1, 1)

# fit a linear regression model on the training set
lr = LinearRegression()
model = lr.fit(x_train, y_train)

# make predictions on the training set
train_pred = model.predict(x_train)

# make predictions on the test set
test_pred = model.predict(x_test)

# calculate the mean squared error on the training set
train_mse = mean_squared_error(y_train, train_pred)
print(f'Training set MSE: {train_mse:.2f}')

# calculate the mean squared error on the test set
test_mse = mean_squared_error(y_test, test_pred)
print(f'Testing set MSE: {test_mse:.2f}')
```

```
Training set MSE: 24.48
Testing set MSE: 22.15
```

In [ ]:

```python
mse_train = []
mse_test = []

for i in range(10):
    # split data into training and test sets
    auto_train, auto_test = train_test_split(auto, test_size=0.2, random_state=i)

    # define the predictor and response variables for the training set
    y_train = auto_train['mpg']
    x_train = auto_train['horsepower'].values.reshape(-1, 1)

    # define the predictor and response variables for the test set
    y_test = auto_test['mpg']
    x_test = auto_test['horsepower'].values.reshape(-1, 1)

    # fit a linear regression model on the training set
    lr = LinearRegression()
    model = lr.fit(x_train, y_train)

    # make predictions on the training set
    train_pred = model.predict(x_train)

    # make predictions on the test set
    test_pred = model.predict(x_test)

    # calculate the mean squared error on the training set
    train_mse = mean_squared_error(y_train, train_pred)
    mse_train.append(train_mse)

    # calculate the mean squared error on the test set
    test_mse = mean_squared_error(y_test, test_pred)
    mse_test.append(test_mse)

print(f'Training MSE: {np.mean(mse_train):.2f}')
```

```
print(f'Test MSE: {np.mean(mse_test):.2f}')
```

```
Training MSE: 24.07
Test MSE: 23.62
```

5.1.2 Leave-One-Out-Cross-Validation

In [63]:
```
# define x and y
y = auto['mpg']
x = auto.drop(['mpg', 'name', 'origin'], axis=1).values.reshape(-1, 6)

# define loocv
loo = LeaveOneOut()

# perform loocv with negative mse
neg_mse = cross_val_score(lr, x, y, cv=loo, scoring='neg_mean_squared_error')

# convert to rmse
rmse = np.sqrt(-neg_mse)

# print the mean rmse
print(f'LOOCV RMSE: {rmse.mean():.2f}')
```

```
LOOCV RMSE: 2.67
```

5.1.3 k-Fold Cross-Validation

In [64]:
```
# define kfold
kf = KFold(n_splits=10, shuffle=True, random_state=42)

# negative mse
neg_mse = cross_val_score(lr, x, y, cv=kf, scoring='neg_mean_squared_error')

# convert to rmse
rmse = np.sqrt(-neg_mse)

# print the mean rmse
print(f'KFold RMSE: {rmse.mean():.2f}')
```

```
KFold RMSE: 3.43
```

In [67]:
```
# manual k-fold

feature = []

model = LinearRegression()

kf = KFold(n_splits=10, shuffle=True, random_state=42)

# visualize the training and testing set for each folds
pd.DataFrame(kf.split(x))

mse_scores = []

# Perform k-fold cross-validation
for train_index, test_index in kf.split(x):
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)

    mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(mse)
```

```
np.mean(mse_scores)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[67], line 16
     14 # Perform k-fold cross-validation
     15 for train_index, test_index in kf.split(x):
---> 16     x_train, x_test = x.iloc[train_index], x.iloc[test_index]
     17     y_train, y_test = y.iloc[train_index], y.iloc[test_index]
     19     model.fit(x_train, y_train)

AttributeError: 'numpy.ndarray' object has no attribute 'iloc'
```

Cuba buat manually to check output from the above function

In [ ]:
```
#  auto dataset has 392 rows and 9 columns

auto.sample(98)
```

Out[ ]:
```
(392, 9)
```

5.2 Bootstrap

Bootstrap is basically resampling with replacement

In [70]:
```
portfolio = pd.read_csv('data/portfolio.csv')
portfolio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  100 non-null    int64
 1   X           100 non-null    float64
 2   Y           100 non-null    float64
dtypes: float64(2), int64(1)
memory usage: 2.5 KB
```

In [72]:
```
x = portfolio['X']
y = portfolio['Y']

# bootstrap
dfsample = portfolio.sample(frac=1, replace=True, random_state=42)
dfsample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 100 entries, 51 to 23
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  100 non-null    int64
 1   X           100 non-null    float64
 2   Y           100 non-null    float64
dtypes: float64(2), int64(1)
memory usage: 3.1 KB
```

6 Linear Model Selection (Week 5, Chapter 6)

- This is also called **Feature Selection**
- alternative to least square

Advantage of Linear model

1. Inference

6.1 Subset Selection

6.1.1 Best Subset Selection

Fit a separate least square regression for each possible model. The number of model that we will fit is 2^p where p is the number of variable. For example, if a data has 20 predictor variable, then the number of model will be 2^20 = 1,048,576. This is a very high number of possibility and will use a lot of computational power to execute best subset selection.

Cons

1. Use high computational power if there is a lot number of variables
2. Higher search space leads to higher chance of finding model that overfit and have high variance on coefficient estimates

no 2 tu maksudnya lagi banyak option yang kita consider untuk bina model, lagi senang untuk kita dapat model yang perform bagus untuk training data tapi tak power untuk predict new/future data. Jadi model tu akan overfit dan ada variance yang tinggi.

In [ ]:

In [ ]:

6.1.2 Stepwise Selection

1. Forward Selection
2. Backward Selection
3. Forward-Backward Selection

6.1.2.1 Forward Selection

Begin with no predictor in the model. Then add one at a time until all predictors are in the model.

Criteria when choosing model for each iteration

1. Lowest RSS
2. Highest R^2

In [ ]:
```
hitters = pd.read_csv('data/hitters.csv').dropna()

y = hitters['Salary']

df = pd.get_dummies(hitters[['League', 'Division', 'NewLeague']])

lr = LinearRegression()

def process(feature):
    model = lr.fit(x[feature], y)
    pred = model.predict(x[feature])
    mse = mean_squared_error(y, pred)
    return mse
```

Out[ ]:

	League_A	League_N	Division_E	Division_W	NewLeague_A	NewLeague_N
1	False	True	False	True	False	True
2	True	False	False	True	True	False
3	False	True	True	False	False	True
4	False	True	True	False	False	True
5	True	False	False	True	True	False
...	...	...	...	...	...	...
317	False	True	True	False	False	True
318	True	False	True	False	True	False
319	True	False	False	True	True	False
320	True	False	True	False	True	False
321	True	False	False	True	True	False

263 rows × 6 columns

In [ ]:

```
# load data
hitters = pd.read_csv('data/hitters.csv').dropna()

# list out all columns with type 'object'
object = hitters.select_dtypes(include=['object']).columns

# remove all columns with type 'object'
hitter = hitters.drop(object, axis=1)

# define x and y
y = hitter['Salary']
x = hitter.drop('Salary', axis=1)

selected_features = []
remaining_features = list(x.columns)
scores_history = []
kf = KFold(n_splits=10, shuffle=True, random_state=42)

for feature in remaining_features:
    current_features = selected_features + [feature] # add list to list, maybe can use append
    subset = x[current_features]


model = LinearRegression()
mse = cross_val_score(model, subset, y, cv=kf, scoring='neg_mean_squared_error')
avg_mse = np.mean(mse)

np.argmax()
```

```
-205708.8753359605
```

Backward Selection

In [ ]:

```
x = add_constant(hitter.drop('Salary', axis=1))
y = hitter['Salary']

model = OLS(y, x).fit()

feature = list(x.columns)
```

```
while len(feature) > 0:
    x_opt = x[feature]
    model = OLS(y, x_opt).fit()
    p_val = model.pvalues
    max_p_val = p_val.max()

    if max_p_val > 0.05:
        excluded_feature = p_val.idxmax()
        feature.remove(excluded_feature)
        print(max_p_val)

    else:
        break

model
```

Out[ ]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x247d90edcd0>

Forward-Backward Selection

In [ ]:

```

```

6.2 Principal Component Regression (PCR)

In [11]:

```
import pandas as pd
import numpy as np

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
```

In [2]:

```
hitters = pd.read_csv('data/hitters.csv').drop('Unnamed: 0', axis=1).dropna()
hitters.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 263 entries, 1 to 321
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      263 non-null    int64
 1   Hits       263 non-null    int64
 2   HmRun      263 non-null    int64
 3   Runs       263 non-null    int64
 4   RBI        263 non-null    int64
 5   Walks      263 non-null    int64
 6   Years      263 non-null    int64
 7   CAtBat     263 non-null    int64
 8   CHits      263 non-null    int64
 9   CHmRun     263 non-null    int64
 10  CRuns      263 non-null    int64
 11  CRBI       263 non-null    int64
 12  CWalks     263 non-null    int64
 13  League     263 non-null    object
 14  Division   263 non-null    object
 15  PutOuts    263 non-null    int64
 16  Assists    263 non-null    int64
 17  Errors     263 non-null    int64
 18  Salary     263 non-null    float64
 19  NewLeague  263 non-null    object
dtypes: float64(1), int64(16), object(3)
memory usage: 43.1+ KB
```

In [3]:
```
hitters_num = hitters.select_dtypes(exclude=[object])
hitters_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 263 entries, 1 to 321
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   AtBat    263 non-null    int64
 1   Hits     263 non-null    int64
 2   HmRun    263 non-null    int64
 3   Runs     263 non-null    int64
 4   RBI      263 non-null    int64
 5   Walks    263 non-null    int64
 6   Years    263 non-null    int64
 7   CAtBat   263 non-null    int64
 8   CHits    263 non-null    int64
 9   CHmRun   263 non-null    int64
 10  CRuns    263 non-null    int64
 11  CRBI     263 non-null    int64
 12  CWalks   263 non-null    int64
 13  PutOuts  263 non-null    int64
 14  Assists  263 non-null    int64
 15  Errors   263 non-null    int64
 16  Salary   263 non-null    float64
dtypes: float64(1), int64(16)
memory usage: 37.0 KB
```

In [4]:
```
scaler = StandardScaler()

hitters_scale = scaler.fit_transform(hitters_num)

pca = PCA()

hitters_reduced = pca.fit_transform(hitters_scale)

pd.DataFrame(pca.components_.T)
```

Out[4]:

	0	1	2	3	4	5	6	7	8	
0	0.195064	0.384078	0.073262	-0.077338	0.092740	-0.153752	-0.253964	-0.007509	0.184109	-(
1	0.194100	0.377645	0.052328	-0.068722	0.137321	-0.298248	-0.249078	-0.065779	0.129306	-(
2	0.196905	0.228663	-0.335869	-0.233200	-0.476803	0.147880	0.181903	0.135254	-0.344985	(
3	0.194913	0.374591	-0.061105	-0.126861	0.213772	-0.053735	-0.129235	-0.158901	0.026457	(
4	0.229566	0.310265	-0.165085	-0.160202	-0.291587	-0.026079	0.035261	0.123319	-0.174084	-(
5	0.206737	0.231158	-0.064313	0.069321	0.523822	0.636282	0.183803	-0.017067	-0.114149	-(
6	0.271085	-0.268204	0.099577	-0.018776	-0.039267	-0.047403	-0.188871	0.016583	-0.488824	-(
7	0.319705	-0.196413	0.128697	-0.002258	-0.010346	-0.073074	-0.154440	-0.049502	-0.093306	(
8	0.320773	-0.185897	0.124966	0.012231	0.002129	-0.126700	-0.156650	-0.095475	-0.083113	(
9	0.308101	-0.133864	-0.114866	-0.086805	-0.220700	0.169098	0.200670	0.150085	0.660149	-(
10	0.327615	-0.176929	0.090643	-0.011355	0.050361	-0.037331	-0.095997	-0.090117	0.002594	(
11	0.329774	-0.172834	0.017574	-0.018780	-0.126116	-0.000322	0.010759	0.038848	0.281185	-(
12	0.305731	-0.196983	0.058216	0.019782	0.143498	0.309299	0.006898	0.015674	-0.059143	(
13	0.083038	0.162952	-0.154233	0.876219	-0.239479	0.115397	-0.289396	0.113799	0.016676	(
14	0.001592	0.176246	0.653316	-0.002904	-0.017801	0.027505	0.087714	0.710120	-0.047850	(
15	-0.005293	0.209783	0.570491	0.066107	-0.397898	0.230634	0.191130	-0.610444	0.009970	-(
16	0.249142	0.054526	-0.026280	0.327562	0.186292	-0.490468	0.725920	-0.049711	-0.103622	(

In [21]:

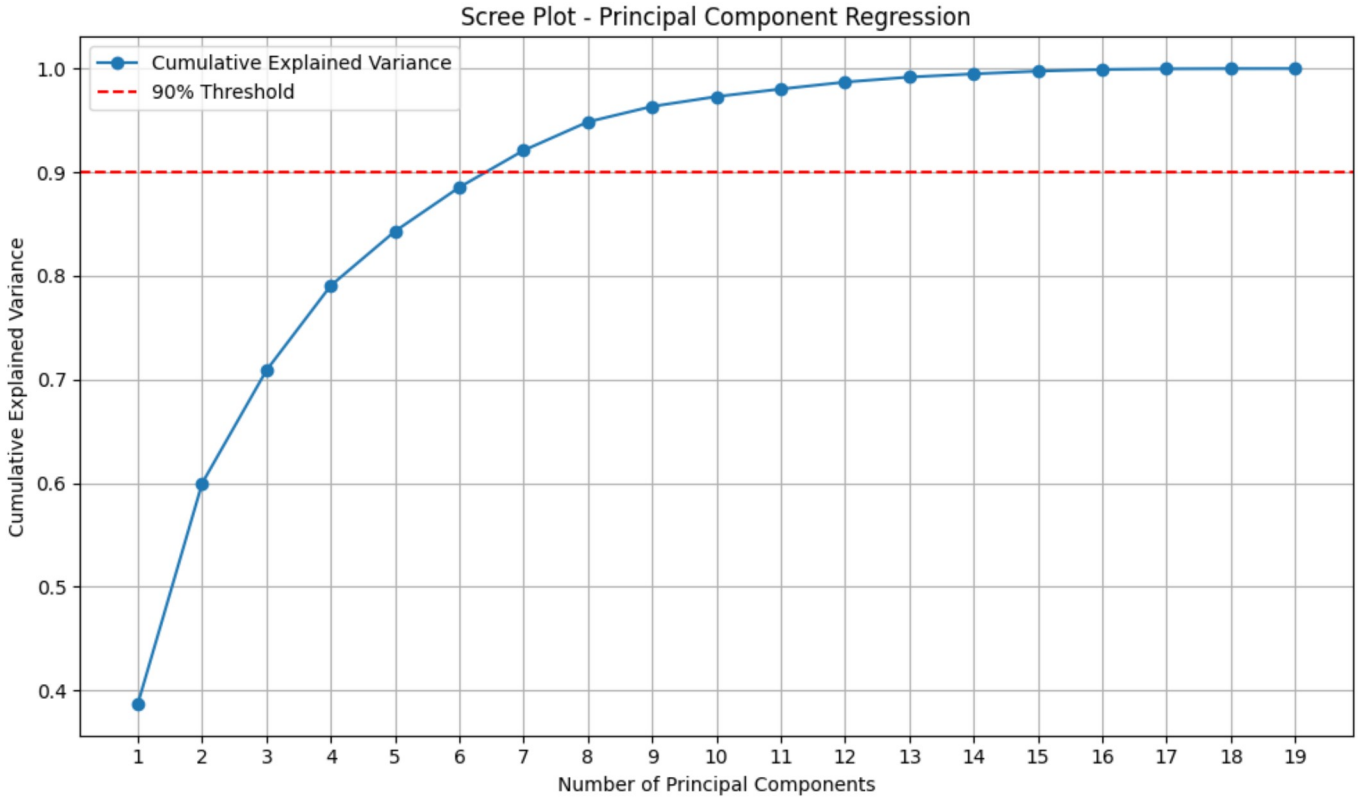
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
import numpy as np

df = pd.read_csv('data/hitters.csv')

# Step 1: Drop rows with missing Salary
df_clean = df.dropna(subset=['Salary'])

# Step 2: Drop unnecessary column and one-hot encode categoricals
df_clean = df_clean.drop(columns=['Unnamed: 0'])
df_encoded = pd.get_dummies(df_clean, drop_first=True)

# Step 3: Define X and y
X = df_encoded.drop(columns='Salary')
y = df_encoded['Salary']

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: PCR pipeline: Standardize -> PCA -> Linear Regression
pcr_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),  # keep all components first
    ('regressor', LinearRegression())
])

# Step 6: Fit PCR and predict
```

```python
pcr_pipeline.fit(X_train, y_train)
y_pred = pcr_pipeline.predict(X_test)

# Step 7: Evaluate
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# How much variance is explained by each component
explained_variance = pcr_pipeline.named_steps['pca'].explained_variance_ratio_

print(f'MSE: {mse:.4f}')
print('==========')
print(f'R2 Score: {(r2 * 100):.2f}%')
print('==========')
print(pd.DataFrame(explained_variance.cumsum()))
```

```
MSE: 128284.3455
==========
R2 Score: 29.07%
==========
           0
0   0.387252
1   0.599958
2   0.708853
3   0.790572
4   0.843038
5   0.885433
6   0.921024
7   0.948553
8   0.963480
9   0.972935
10  0.980239
11  0.986943
12  0.991763
13  0.994866
14  0.997469
15  0.999093
16  0.999725
17  0.999943
18  1.000000
```

In [19]:

```python
import matplotlib.pyplot as plt

# Extract explained variance ratio from the fitted PCA
explained_variance = pcr_pipeline.named_steps['pca'].explained_variance_ratio_
cumulative_variance = explained_variance.cumsum()
components = np.arange(1, len(explained_variance) + 1)

# Plot the scree plot
plt.figure(figsize=(10, 6))
plt.plot(components, cumulative_variance, marker='o', linestyle='-', label='Cumulative Explain
plt.axhline(y=0.9, color='red', linestyle='--', label='90% Threshold')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Scree Plot - Principal Component Regression')
plt.xticks(components)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



In [20]:
```
# Try PCR using only the top N principal components (e.g., N=6 from scree plot)
N = 6

# Redefine the pipeline with only N components
pcr_topN_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=N)),
    ('regressor', LinearRegression())
])

# Fit and evaluate
pcr_topN_pipeline.fit(X_train, y_train)
y_pred_topN = pcr_topN_pipeline.predict(X_test)

mse_topN = mean_squared_error(y_test, y_pred_topN)
r2_topN = r2_score(y_test, y_pred_topN)

mse_topN, r2_topN
```

Out[20]: (150957.42349665504, 0.16539092144953838)

6.3 Partial Least Square (PLS)

In [ ]:
```
len(hitters_scale)

kf = KFold(n_splits=10, shuffle=True, random_state=42)

pls = PLSRegression(n_components=1)

score = cross_val_score(pls, )
```

Out[ ]: 263

Method	Advantages	Disadvantages

Method	Advantages	Disadvantages
Principal Component Regression (PCR)	✅ Reduces dimensionality and multicollinearity ✅ Captures variance in predictors ✅ Simple to compute	❌ Components chosen without regard to response variable ❌ May ignore components that predict `y` well ❌ Interpretation can be difficult
Partial Least Squares (PLS)	✅ Considers both predictor and response variability ✅ Often better predictive performance than PCR ✅ Handles multicollinearity	❌ More complex to interpret than PCR ❌ May overfit if too many components chosen ❌ Slightly harder to compute and tune

🔍 Summary:

Situation	Preferred Method
You want to **reduce dimensionality** and focus on **variance in predictors**	✅ Use **PCR**
You want to **maximize prediction accuracy** and consider `y` during component selection	✅ Use **PLS**

7 Nonlinear models (Week 6, Chapter 7)

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from statsmodels.api import GLS
```

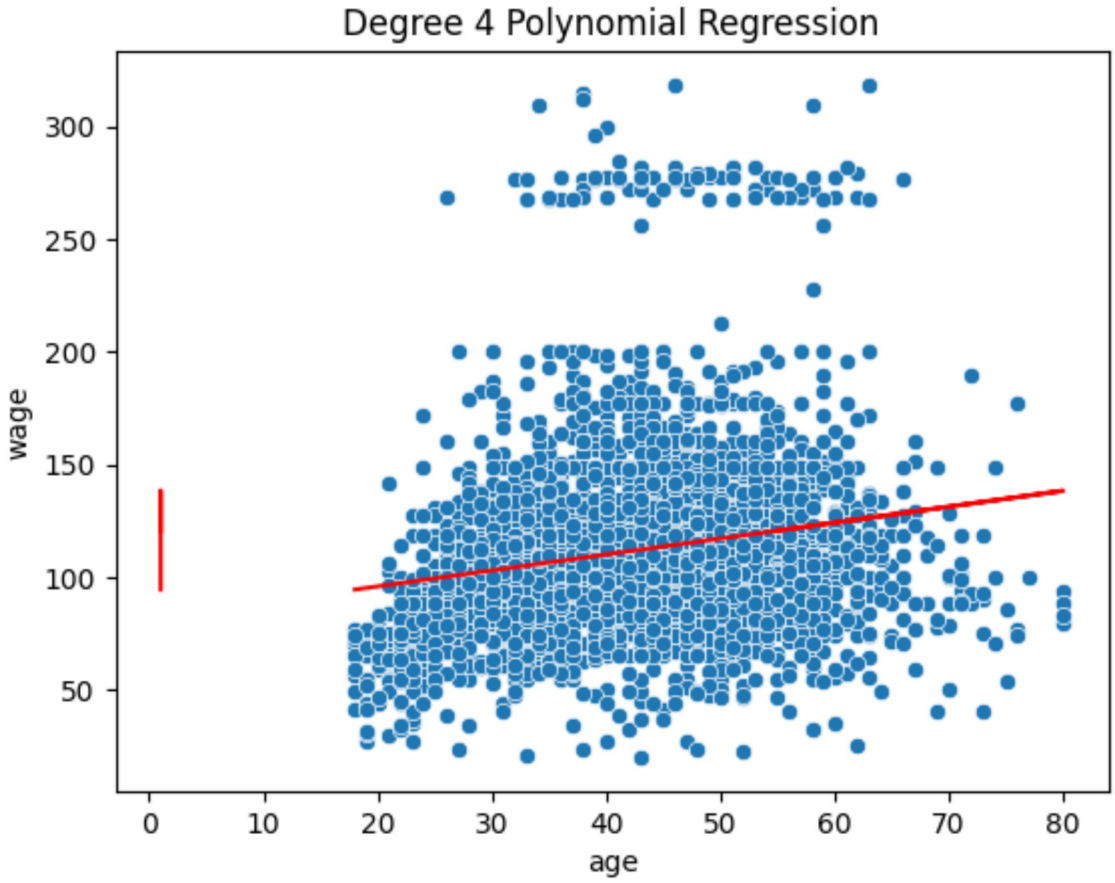
In [ ]:
```
wage = pd.read_csv('data/wage.csv')
print(wage.shape)
print('-------------------------------------------')
print(wage.head(5))
print('-------------------------------------------')
print(wage.info())
```

```
(3000, 12)
--------------------------------------------
   year  age      sex            maritl      race        education  \
0  2006   18  1. Male  1. Never Married  1. White     1. < HS Grad
1  2004   24  1. Male  1. Never Married  1. White  4. College Grad
2  2003   45  1. Male        2. Married  1. White  3. Some College
3  2003   43  1. Male        2. Married  3. Asian  4. College Grad
4  2005   50  1. Male       4. Divorced  1. White       2. HS Grad

               region        jobclass          health health_ins   logwage  \
0  2. Middle Atlantic   1. Industrial       1. <=Good      2. No  4.318063
1  2. Middle Atlantic  2. Information  2. >=Very Good      2. No  4.255273
2  2. Middle Atlantic   1. Industrial       1. <=Good     1. Yes  4.875061
3  2. Middle Atlantic  2. Information  2. >=Very Good     1. Yes  5.041393
4  2. Middle Atlantic  2. Information       1. <=Good     1. Yes  4.318063

         wage
0   75.043154
1   70.476020
2  130.982177
3  154.685293
4   75.043154
--------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   year        3000 non-null   int64
 1   age         3000 non-null   int64
 2   sex         3000 non-null   object
 3   maritl      3000 non-null   object
 4   race        3000 non-null   object
 5   education   3000 non-null   object
 6   region      3000 non-null   object
 7   jobclass    3000 non-null   object
 8   health      3000 non-null   object
 9   health_ins  3000 non-null   object
 10  logwage     3000 non-null   float64
 11  wage        3000 non-null   float64
dtypes: float64(2), int64(2), object(8)
memory usage: 281.4+ KB
None
```

In [ ]:

In [ ]:
```
x = add_constant(wage['age'])
y = wage['wage']
model = OLS(y, x).fit()
prediction = model.predict(x)
```

In [ ]:
```
sns.scatterplot(x='age', y='wage', data=wage)
plt.plot(x, prediction, color='red', label='predicted value')
plt.title('Degree 4 Polynomial Regression')
plt.show();
```

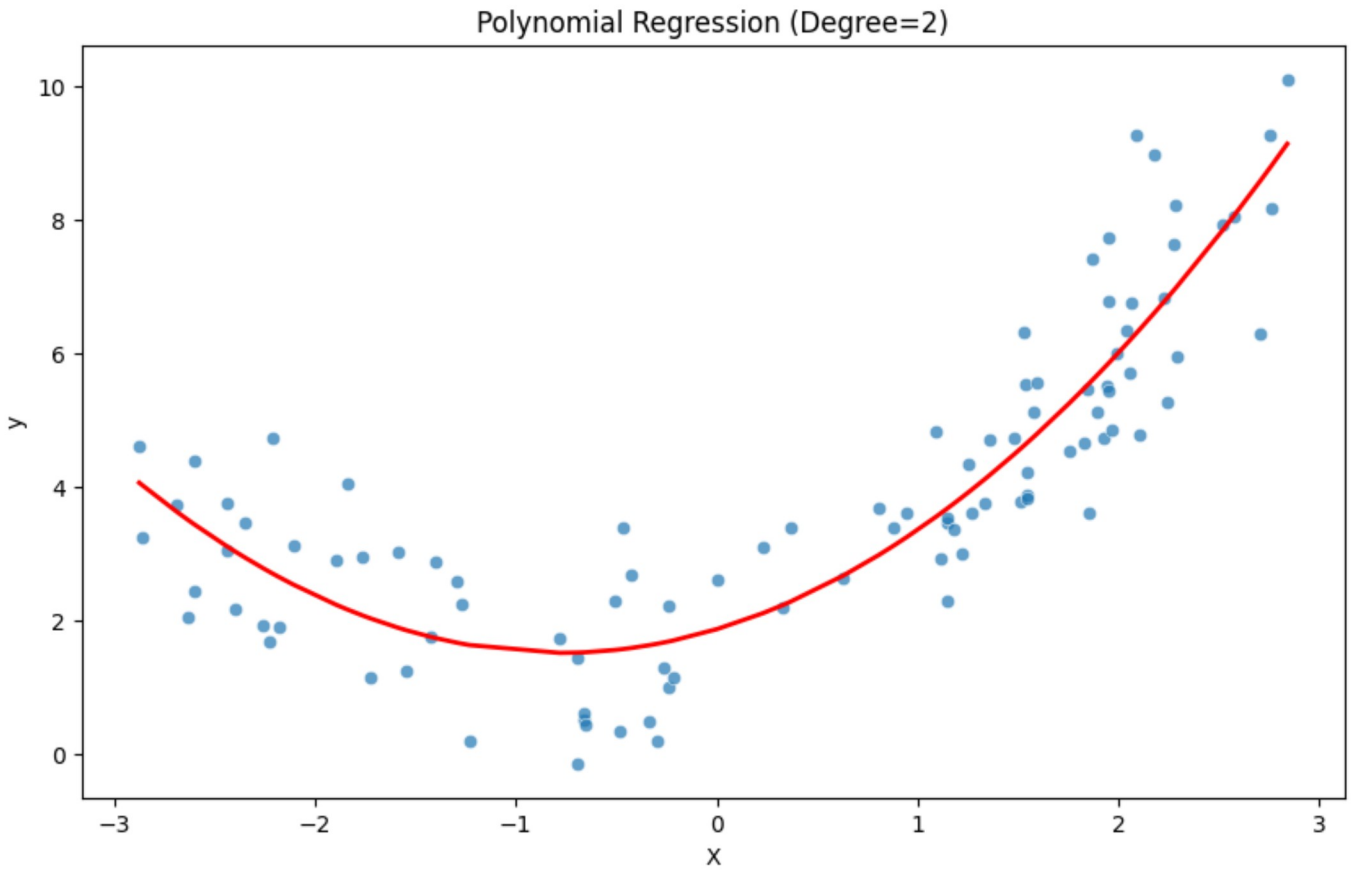


In [ ]:
```
x_poly = PolynomialFeatures(degree=2).fit_transform(wage['age'].values.reshape(-1, 1))
y = wage['wage']

print(LinearRegression().fit(x_poly, y).intercept_)
```

```
-10.425224262028578
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Generate data
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)

# Create polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Fit model and predict
model = LinearRegression().fit(X_poly, y)
predictions = model.predict(X_poly)

# Sort X and predictions for proper plotting
sorted_indices = np.argsort(X[:, 0])
X_sorted = X[sorted_indices]
predictions_sorted = predictions[sorted_indices]

# Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_sorted[:, 0], y=y[sorted_indices][:, 0], alpha=0.7)
plt.plot(X_sorted[:, 0], predictions_sorted[:, 0], 'r-', linewidth=2)
plt.title("Polynomial Regression (Degree=2)")
plt.xlabel("X")
plt.ylabel("y")
```

```
plt.show()
```



In [ ]:
```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)

from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2, include_bias=False) # try to include bias
x_poly = poly.fit_transform(X)
prediction = LinearRegression().fit(x_poly, y).predict(x_poly)

sns.scatterplot(x=x_poly, y=y)
plt.plot(X, prediction)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[68], line 10
      7 x_poly = poly.fit_transform(X)
      8 prediction = LinearRegression().fit(x_poly, y).predict(x_poly)
---> 10 sns.scatterplot(x=x_poly, y=y)
     11 plt.plot(X, prediction)

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\relationa
l.py:615, in scatterplot(data, x, y, hue, size, style, palette, hue_order, hue_norm, sizes, si
ze_order, size_norm, markers, style_order, legend, ax, **kwargs)
    606 def scatterplot(
    607     data=None, *,
    608     x=None, y=None, hue=None, size=None, style=None,
   (...)    612     **kwargs
    613 ):
--> 615     p = _ScatterPlotter(
    616         data=data,
    617         variables=dict(x=x, y=y, hue=hue, size=size, style=style),
    618         legend=legend
    619     )
    621     p.map_hue(palette=palette, order=hue_order, norm=hue_norm)
    622     p.map_size(sizes=sizes, order=size_order, norm=size_norm)

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\relationa
l.py:396, in _ScatterPlotter.__init__(self, data, variables, legend)
    387 def __init__(self, *, data=None, variables={}, legend=None):
    388
    389     # TODO this is messy, we want the mapping to be agnostic about
    390     # the kind of plot to draw, but for the time being we need to set
    391     # this information so the SizeMapping can use it
    392     self._default_size_range = (
    393         np.r_[.5, 2] * np.square(mpl.rcParams["lines.markersize"])
    394     )
--> 396     super().__init__(data=data, variables=variables)
    398     self.legend = legend

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\_base.p
y:634, in VectorPlotter.__init__(self, data, variables)
    629 # var_ordered is relevant only for categorical axis variables, and may
    630 # be better handled by an internal axis information object that tracks
    631 # such information and is set up by the scale_* methods. The analogous
    632 # information for numeric axes would be information about log scales.
    633 self._var_ordered = {"x": False, "y": False}  # alt., used DefaultDict
--> 634 self.assign_variables(data, variables)
    636 # TODO Lots of tests assume that these are called to initialize the
    637 # mappings to default values on class initialization. I'd prefer to
    638 # move away from that and only have a mapping when explicitly called.
    639 for var in ["hue", "size", "style"]:

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\_base.p
y:679, in VectorPlotter.assign_variables(self, data, variables)
    674 else:
    675     # When dealing with long-form input, use the newer PlotData
    676     # object (internal but introduced for the objects interface)
    677     # to centralize / standardize data consumption logic.
    678     self.input_format = "long"
--> 679     plot_data = PlotData(data, variables)
    680     frame = plot_data.frame
    681     names = plot_data.names

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\_core\dat
a.py:58, in PlotData.__init__(self, data, variables)
     51 def __init__(
     52     self,
```

```
     53     data: DataSource,
     54     variables: dict[str, VariableSpec],
     55 ):
     57     data = handle_data_source(data)
---> 58     frame, names, ids = self._assign_variables(data, variables)
     60     self.frame = frame
     61     self.names = names

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\seaborn\_core\dat
a.py:265, in PlotData._assign_variables(self, data, variables)
    260             ids[key] = id(val)
    262 # Construct a tidy plot DataFrame. This will convert a number of
    263 # types automatically, aligning on index in case of pandas objects
    264 # TODO Note: this fails when variable specs *only* have scalars!
--> 265 frame = pd.DataFrame(plot_data)
    267 return frame, names, ids

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core\fram
e.py:778, in DataFrame.__init__(self, data, index, columns, dtype, copy)
    772     mgr = self._init_mgr(
    773         data, axes={"index": index, "columns": columns}, dtype=dtype, copy=copy
    774     )
    776 elif isinstance(data, dict):
    777     # GH#38939 de facto copy defaults to False only in non-dict cases
--> 778     mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=copy, typ=manager)
    779 elif isinstance(data, ma.MaskedArray):
    780     from numpy.ma import mrecords

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core\inter
nals\construction.py:503, in dict_to_mgr(data, index, columns, dtype, typ, copy)
    499     else:
    500         # dtype check to exclude e.g. range objects, scalars
    501         arrays = [x.copy() if hasattr(x, "dtype") else x for x in arrays]
--> 503 return arrays_to_mgr(arrays, columns, index, dtype=dtype, typ=typ, consolidate=copy)

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core\inter
nals\construction.py:114, in arrays_to_mgr(arrays, columns, index, dtype, verify_integrity, ty
p, consolidate)
    111 if verify_integrity:
    112     # figure out the index, if necessary
    113     if index is None:
--> 114         index = _extract_index(arrays)
    115     else:
    116         index = ensure_index(index)

File c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core\inter
nals\construction.py:664, in _extract_index(data)
    662         raw_lengths.append(len(val))
    663     elif isinstance(val, np.ndarray) and val.ndim > 1:
--> 664         raise ValueError("Per-column arrays must each be 1-dimensional")
    666 if not indexes and not raw_lengths:
    667     raise ValueError("If using all scalar values, you must pass an index")

ValueError: Per-column arrays must each be 1-dimensional
```

7.5 Moving Beyond Linearity

Principal Component Analysis (PCA)

In [ ]:

8 Tree Based Method

	Decision Tree	Random Forest	Boosting
Advantage	Easy to interpret		
Disadvantage			

Advantage

1. Easy to interpret/explain
2. Closely mirror human decision-making
3. Can be displayed graphically
4. Can handle qualitative (categorical) variables without OHE

Disadvantage

1. Lower prediction accuray
2. Trees can be non-robust (A small change in the data can cause a large change in the final estimated tree)

Classification Tree

In [90]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
```

In [180…
```
carseat = pd.read_csv('data/carseats.csv')
carseat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Sales        400 non-null    float64
 1   CompPrice    400 non-null    int64
 2   Income       400 non-null    int64
 3   Advertising  400 non-null    int64
 4   Population   400 non-null    int64
 5   Price        400 non-null    int64
 6   ShelveLoc    400 non-null    object
 7   Age          400 non-null    int64
 8   Education    400 non-null    int64
 9   Urban        400 non-null    object
 10  US           400 non-null    object
dtypes: float64(1), int64(7), object(3)
memory usage: 34.5+ KB
```

In order to use classification tree, we need to convert reponse predictor variables into categorical and all categorical predictor variables into continous

- X (predictor) : categorical to continous
- Y (response) : continous to categorical (1 and 0)

In [181…
```
# convert continous variable 'Sales' into categorical 'High', 1 if Sales > 8 else 0
```

```
carseat['High'] = carseat['Sales'].apply(lambda x: 1 if x > 8 else 0)

carseat['ShelveLoc'] = carseat['ShelveLoc'].map({'Bad': 0, 'Medium': 1, 'Good': 2})
carseat['Urban'] = carseat['Urban'].map({'No': 0, 'Yes': 1})
carseat['US'] = carseat['US'].map({'No': 0, 'Yes':1})
```

In [182…

```
# define response and predictor variables
y = carseat['High']
x = carseat.drop(['Sales', 'High'], axis=1)

# split data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# fit model
model = DecisionTreeClassifier(max_depth=6, random_state=42)
model.fit(x_train, y_train)
print(f'Model Score for Training Set: {(model.score(x_train, y_train) * 100):.2f}%')
print(f'Model Score for Testing Set: {(model.score(x_test, y_test) * 100):.2f}%')

print('==========')

print(f'Accuracy score for Testing Set: {(accuracy_score(y_test, model.predict(x_test)) * 100)

print('==========')

print('Confusion Matrix: ')
print(pd.DataFrame(confusion_matrix(y_test, model.predict(x_test)),
                   index=['Yes', 'No'],
                   columns=['Yes', 'No']))
```

```
Model Score for Training Set: 91.56%
Model Score for Testing Set: 77.50%
==========
Accuracy score for Testing Set: 77.50%
==========
Confusion Matrix:
     Yes  No
Yes   37   6
No    12  25
```

In [184…

```
pred = model.predict(x_test)
print(f'Model Score for Test Set: {(model.score(x_test, y_test) * 100):.2f}%')
```

```
Model Score for Test Set: 77.50%
```

In [185…

```
print(pd.DataFrame(confusion_matrix(y_test, pred).T,
                   index = ['Yes', 'No'],
                   columns=['Yes', 'No']))
```

```
     Yes  No
Yes   37  12
No     6  25
```

Bagging

- Train a lot of decision tree at the same time
- Each decision tree are using different dataset that has been resample from the original dataset
- **Bootstrap**: resample 100 data from the original 100 data with replacement (some observation are taken multiple times, some are not taken at all)
- **Bootstrap Aggregation**

Random Forest

In [186…

```
# fit random forest regressor model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(x_train, y_train)

# make predictions on the test set
pred = rf.predict(x_test)

print(f'Model Score for Test Set: {(rf.score(x_test, y_test) * 100):.2f}%')
```

```
Model Score for Test Set: 46.39%
```

Boosting

- **n_estimators**: Number of decision trees
- **learning_rate (shrinkage)** λ:
- **max_depth**: maximum depth of each decision tree. Deeper depth can capture more complex model, but it will prone to overfitting especially with high learning rate.

In [187…

```
from sklearn.ensemble import GradientBoostingRegressor

boost = GradientBoostingRegressor(n_estimators=500, learning_rate=0.01, max_depth=4,
                                  random_state=42)

boost.fit(x_train, y_train)

print(f'MSE for Training Set: {mean_squared_error(y_train, boost.predict(x_train)):.4f}')
print(f'MSE for Testing Set: {mean_squared_error(y_test, boost.predict(x_test)):.4f}')
```

```
MSE for Training Set: 0.0211
MSE for Testing Set: 0.1385
```

In [189…

```
print(pd.DataFrame({'Variable':x.columns,
                    'Importance':boost.feature_importances_}))
```

```
      Variable  Importance
0    CompPrice    0.116175
1       Income    0.106623
2  Advertising    0.121085
3   Population    0.030032
4        Price    0.289173
5    ShelveLoc    0.186637
6          Age    0.108056
7    Education    0.030398
8        Urban    0.002758
9           US    0.009062
```

Model	Advantages	Disadvantages
Logistic Regression	✅ Simple and fast to train ✅ Interpretable coefficients ✅ Good with linearly separable data ✅ Probabilistic output	❌ Assumes linear decision boundary ❌ Poor with complex, non-linear relationships ❌ Sensitive to multicollinearity
Decision Tree	✅ Easy to interpret and visualize ✅ Handles non-linearity and interactions ✅ Works with both numerical and categorical data	❌ Prone to overfitting ❌ Unstable to small changes in data ❌ Poor generalization alone

Model	Advantages	Disadvantages
Random Forest	✅ Reduces overfitting vs single tree ✅ High accuracy ✅ Robust to noise and outliers ✅ Handles high-dimensional data well	❌ Less interpretable ❌ Slower to train and predict ❌ Requires tuning of many trees
Boosting (e.g., Gradient Boosting)	✅ Often best predictive performance ✅ Reduces both bias and variance ✅ Works well with structured/ tabular data	❌ Very sensitive to noise and overfitting if not tuned ❌ Slower training time ❌ Harder to interpret

🔍 Summary:

- Use **logistic regression** for **interpretability** and **simple problems**.
- Use **tree-based models** for **complex, non-linear data** and **high accuracy** — especially **random forest** or **boosting** for better performance.

9 Support Vector Machine (SVM)

Recommended dataset for praticing SVC

- **Linear Kernel**: Iris, Breast Cancer, Wine
- **RBF Kernel / Non-Linear**: Moons Circles, Digits, MNIST
- **High-dimensional data**: SpamBase, MNIST
- **Imbalanced data**: Breast Cancer, Titanic

Kernel

- **linear**: linear plane
- **poly**: polynomial kernel
- **rbf**: radial basis function (radial kernel)

In [78]:
```
from sklearn.svm import SVC # Support Vector Classifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
import pandas as pd
```

In [79]:
```
# Optical Recognition of Handwritten Digits dataset
train = pd.read_csv('data/optdigits.tra', header=None)
x_train = train.drop(64, axis=1)  # drop the last column (label)
y_train = train[64]  # last column is the label

test = pd.read_csv('data/optdigits.tes', header=None)
x_test = test.drop(64, axis=1)  # drop the last column (label)
y_test = test[64]  # last column is the label

print(f'Train set shape: {x_train.shape}')
print(f'Test set shape: {x_test.shape}')
```

```
Train set shape: (3823, 64)
Test set shape: (1797, 64)
```

In [ ]:
```
print(f'Null values in train set: {train.isna().sum().sum()}')
print(f'Duplicated values in train set: {train.duplicated().sum()}')
print(f'Null values in test set: {test.isna().sum().sum()}')
print(f'Duplicated values in test set: {test.duplicated().sum()}')
```

```
Null values in train set: 0
Duplicated values in train set: 0
Null values in test set: 0
Duplicated values in test set: 0
```

C : Regularization parameter

small C => wider margin and tolerate more misclassification (prevent overfit)

large C => narrow margin, tries to classify all training examples correctly (risk of overfit)

to find the best value of C, use cross validation `GridSearchCV`

In [ ]:
```
svc = SVC(kernel='linear', C=10, random_state=42)
svc.fit(x_train, y_train)

pred = svc.predict(x_train)
print(f'Train set accuracy: {(svc.score(x_train, y_train) * 100):.2f}%')
print(pd.DataFrame(confusion_matrix(y_train, pred)))
print('====================================================')
print(f'Test set accuracy: {(svc.score(x_test, y_test) * 100):.2f}%')
print(pd.DataFrame(confusion_matrix(y_test, svc.predict(x_test))))
```

```
Train set accuracy: 100.00%
     0    1    2    3    4    5    6    7    8    9
0  376    0    0    0    0    0    0    0    0    0
1    0  389    0    0    0    0    0    0    0    0
2    0    0  380    0    0    0    0    0    0    0
3    0    0    0  389    0    0    0    0    0    0
4    0    0    0    0  387    0    0    0    0    0
5    0    0    0    0    0  376    0    0    0    0
6    0    0    0    0    0    0  377    0    0    0
7    0    0    0    0    0    0    0  387    0    0
8    0    0    0    0    0    0    0    0  380    0
9    0    0    0    0    0    0    0    0    0  382
====================================================
Test set accuracy: 96.10%
     0    1    2    3    4    5    6    7    8    9
0  177    0    0    0    0    1    0    0    0    0
1    0  178    0    0    0    0    3    0    1    0
2    0    7  170    0    0    0    0    0    0    0
3    1    0    5  171    0    2    0    2    1    1
4    0    0    0    0  180    0    0    0    1    0
5    0    0    1    0    0  180    0    0    0    1
6    0    0    0    0    1    0  179    0    1    0
7    0    0    0    0    1    7    0  165    0    6
8    0    9    1    5    0    2    0    0  157    0
9    1    0    0    4    1    3    0    0    1  170
```

In [ ]:
```
# determine the optimal value of C
tuned_parameters = {'C': [0.1, 1, 10, 100, 1000]}
clf = GridSearchCV(SVC(kernel='linear', random_state=42), tuned_parameters, cv=10, scoring='ac
clf.fit(x_train, y_train)
clf.best_params_
```

Out[ ]:
```
{'C': 0.1}
```

In [ ]:
```
svc = SVC(kernel='linear', C=0.1, random_state=42)
svc.fit(x_train, y_train)
```

```
pred = svc.predict(x_train)
print(f'Train set accuracy: {(svc.score(x_train, y_train) * 100):.2f}%')
print(pd.DataFrame(confusion_matrix(y_train, pred)))
print('==================================================')
print(f'Test set accuracy: {(svc.score(x_test, y_test) * 100):.2f}%')
print(pd.DataFrame(confusion_matrix(y_test, svc.predict(x_test))))
```

```
Train set accuracy: 99.97%
     0    1    2    3    4    5    6    7    8    9
0  376    0    0    0    0    0    0    0    0    0
1    0  389    0    0    0    0    0    0    0    0
2    0    0  380    0    0    0    0    0    0    0
3    0    0    0  389    0    0    0    0    0    0
4    0    0    0    0  387    0    0    0    0    0
5    0    0    0    0    0  376    0    0    0    0
6    0    0    0    0    0    0  377    0    0    0
7    0    0    0    0    0    0    0  387    0    0
8    0    1    0    0    0    0    0    0  379    0
9    0    0    0    0    0    0    0    0    0  382
==================================================
Test set accuracy: 96.16%
     0    1    2    3    4    5    6    7    8    9
0  177    0    0    0    0    1    0    0    0    0
1    0  177    0    0    0    0    3    0    2    0
2    0    7  170    0    0    0    0    0    0    0
3    1    0    5  171    0    2    0    2    1    1
4    0    0    0    0  180    0    0    0    1    0
5    0    0    1    0    0  180    0    0    0    1
6    0    0    0    0    1    0  179    0    1    0
7    0    0    0    0    1    7    0  165    0    6
8    0    8    1    5    0    1    0    0  159    0
9    1    0    0    4    1    3    0    0    1  170
```

Radial Basis Function (RBF)

In [ ]:

```
```

10 Unsupervised Learning

- When our data does not have response (Y) value
- When we want to find interesting things in explanatory (X) variables

10.1 Principal Component Analysis (PCA)

In [9]:

```
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [29]:

```
arrest = pd.read_csv('Data/USArrests.csv', index_col='Unnamed: 0')
arrest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 50 entries, Alabama to Wyoming
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Murder    50 non-null     float64
 1   Assault   50 non-null     int64
 2   UrbanPop  50 non-null     int64
 3   Rape      50 non-null     float64
dtypes: float64(2), int64(2)
memory usage: 2.0+ KB
```

In [30]:

```
scale = StandardScaler()
arrest_scale = scale.fit_transform(arrest)
pd.DataFrame(arrest_scale, columns=arrest.columns)
```

Out[30]:

	Murder	Assault	UrbanPop	Rape
0	1.255179	0.790787	-0.526195	-0.003451
1	0.513019	1.118060	-1.224067	2.509424
2	0.072361	1.493817	1.009122	1.053466
3	0.234708	0.233212	-1.084492	-0.186794
4	0.281093	1.275635	1.776781	2.088814
5	0.025976	0.402909	0.869548	1.883901
6	-1.040880	-0.736484	0.799761	-1.092723
7	-0.437875	0.815030	0.450825	-0.585834
8	1.765415	1.990786	1.009122	1.150530
9	2.229265	0.487757	-0.386621	0.492653
10	-0.577030	-1.512241	1.218484	-0.111300
11	-1.203228	-0.615272	-0.805344	-0.758392
12	0.605789	0.948363	1.218484	0.298525
13	-0.136372	-0.700121	-0.037685	-0.025021
14	-1.295998	-1.391029	-0.595982	-1.071153
15	-0.414682	-0.675878	0.032102	-0.348567
16	0.443441	-0.748605	-0.944918	-0.531910
17	1.765415	0.948363	0.032102	0.104398
18	-1.319191	-1.063757	-1.014705	-1.448624
19	0.814521	1.566544	0.101889	0.708350
20	-0.785763	-0.263757	1.358058	-0.531910
21	1.000062	1.021090	0.590399	1.495646
22	-1.180036	-1.197090	0.032102	-0.682898
23	1.927762	1.069575	-1.503215	-0.445631
24	0.281093	0.087757	0.311251	0.751490
25	-0.414682	-0.748605	-0.875131	-0.521125
26	-0.808955	-0.833454	-0.247047	-0.510340
27	1.023254	0.984726	1.078909	2.671197
28	-1.319191	-1.378908	-0.665769	-1.265281
29	-0.089987	-0.142545	1.637207	-0.262288
30	0.837714	1.384726	0.311251	1.172100
31	0.768136	1.008969	1.427845	0.525008
32	1.208794	2.015028	-1.433428	-0.553480
33	-1.620693	-1.524362	-1.503215	-1.502548
34	-0.113180	-0.615272	0.660186	0.018119

	Murder	Assault	UrbanPop	Rape
35	-0.275527	-0.239515	0.171676	-0.132870
36	-0.669800	-0.142545	0.101889	0.870123
37	-0.345105	-0.784969	0.450825	-0.682898
38	-1.017688	0.039273	1.497632	-1.394700
39	1.533490	1.311999	-1.224067	0.136752
40	-0.924918	-1.027393	-1.433428	-0.909380
41	1.255179	0.208970	-0.456408	0.611287
42	1.139217	0.366545	1.009122	0.460298
43	-1.064073	-0.615272	1.009122	0.179892
44	-1.295998	-1.487999	-2.340661	-1.081938
45	0.165131	-0.178909	-0.177259	-0.057376
46	-0.878533	-0.312242	0.520612	0.535792
47	-0.484260	-1.087999	-1.852151	-1.286851
48	-1.203228	-1.427393	0.032102	-1.125078
49	-0.229142	-0.118303	-0.386621	-0.607404

In [33]:

```
scale = StandardScaler()
arrest_scale = scale.fit_transform(arrest)

pca = PCA()
pca.fit(arrest_scale)

pca_loadings = pd.DataFrame(pca.fit(arrest_scale),
                            index=arrest.index,
                            columns=['V1', 'V2', 'V3', 'V4']) # principal components
pca_loadings
```

Out[33]:

	V1	V2	V3	V4
Alabama	PCA()	PCA()	PCA()	PCA()
Alaska	PCA()	PCA()	PCA()	PCA()
Arizona	PCA()	PCA()	PCA()	PCA()
Arkansas	PCA()	PCA()	PCA()	PCA()
California	PCA()	PCA()	PCA()	PCA()
Colorado	PCA()	PCA()	PCA()	PCA()
Connecticut	PCA()	PCA()	PCA()	PCA()
Delaware	PCA()	PCA()	PCA()	PCA()
Florida	PCA()	PCA()	PCA()	PCA()
Georgia	PCA()	PCA()	PCA()	PCA()
Hawaii	PCA()	PCA()	PCA()	PCA()
Idaho	PCA()	PCA()	PCA()	PCA()
Illinois	PCA()	PCA()	PCA()	PCA()
Indiana	PCA()	PCA()	PCA()	PCA()
Iowa	PCA()	PCA()	PCA()	PCA()
Kansas	PCA()	PCA()	PCA()	PCA()
Kentucky	PCA()	PCA()	PCA()	PCA()
Louisiana	PCA()	PCA()	PCA()	PCA()
Maine	PCA()	PCA()	PCA()	PCA()
Maryland	PCA()	PCA()	PCA()	PCA()
Massachusetts	PCA()	PCA()	PCA()	PCA()
Michigan	PCA()	PCA()	PCA()	PCA()
Minnesota	PCA()	PCA()	PCA()	PCA()
Mississippi	PCA()	PCA()	PCA()	PCA()
Missouri	PCA()	PCA()	PCA()	PCA()
Montana	PCA()	PCA()	PCA()	PCA()
Nebraska	PCA()	PCA()	PCA()	PCA()
Nevada	PCA()	PCA()	PCA()	PCA()
New Hampshire	PCA()	PCA()	PCA()	PCA()
New Jersey	PCA()	PCA()	PCA()	PCA()
New Mexico	PCA()	PCA()	PCA()	PCA()
New York	PCA()	PCA()	PCA()	PCA()
North Carolina	PCA()	PCA()	PCA()	PCA()
North Dakota	PCA()	PCA()	PCA()	PCA()
Ohio	PCA()	PCA()	PCA()	PCA()

	V1	V2	V3	V4
Oklahoma	PCA()	PCA()	PCA()	PCA()
Oregon	PCA()	PCA()	PCA()	PCA()
Pennsylvania	PCA()	PCA()	PCA()	PCA()
Rhode Island	PCA()	PCA()	PCA()	PCA()
South Carolina	PCA()	PCA()	PCA()	PCA()
South Dakota	PCA()	PCA()	PCA()	PCA()
Tennessee	PCA()	PCA()	PCA()	PCA()
Texas	PCA()	PCA()	PCA()	PCA()
Utah	PCA()	PCA()	PCA()	PCA()
Vermont	PCA()	PCA()	PCA()	PCA()
Virginia	PCA()	PCA()	PCA()	PCA()
Washington	PCA()	PCA()	PCA()	PCA()
West Virginia	PCA()	PCA()	PCA()	PCA()
Wisconsin	PCA()	PCA()	PCA()	PCA()
Wyoming	PCA()	PCA()	PCA()	PCA()

In [25]:
```
pca = PCA()

df_plot = pd.DataFrame(pca.fit_transform(arrest_scale),
                       columns=['PC1', 'PC2', 'PC3', 'PC4'],
                       index=arrest.index)

df_plot
```

Out[25]:

	PC1	PC2	PC3	PC4
Alabama	0.985566	-1.133392	-0.444269	-0.156267
Alaska	1.950138	-1.073213	2.040003	0.438583
Arizona	1.763164	0.745957	0.054781	0.834653
Arkansas	-0.141420	-1.119797	0.114574	0.182811
California	2.523980	1.542934	0.598557	0.341996
Colorado	1.514563	0.987555	1.095007	-0.001465
Connecticut	-1.358647	1.088928	-0.643258	0.118469
Delaware	0.047709	0.325359	-0.718633	0.881978
Florida	3.013042	-0.039229	-0.576829	0.096285
Georgia	1.639283	-1.278942	-0.342460	-1.076797
Hawaii	-0.912657	1.570460	0.050782	-0.902807
Idaho	-1.639800	-0.210973	0.259801	0.499104
Illinois	1.378911	0.681841	-0.677496	0.122021
Indiana	-0.505461	0.151563	0.228055	-0.424666
Iowa	-2.253646	0.104054	0.164564	-0.017556
Kansas	-0.796881	0.270165	0.025553	-0.206496
Kentucky	-0.750859	-0.958440	-0.028369	-0.670557
Louisiana	1.564818	-0.871055	-0.783480	-0.454728
Maine	-2.396829	-0.376392	-0.065682	0.330460
Maryland	1.763369	-0.427655	-0.157250	0.559070
Massachusetts	-0.486166	1.474496	-0.609497	0.179599
Michigan	2.108441	0.155397	0.384869	-0.102372
Minnesota	-1.692682	0.632261	0.153070	-0.067317
Mississippi	0.996494	-2.393796	-0.740808	-0.215508
Missouri	0.696787	0.263355	0.377444	-0.225824
Montana	-1.185452	-0.536874	0.246889	-0.123742
Nebraska	-1.265637	0.193954	0.175574	-0.015893
Nevada	2.874395	0.775600	1.163380	-0.314515
New Hampshire	-2.383915	0.018082	0.036855	0.033137
New Jersey	0.181566	1.449506	-0.764454	-0.243383
New Mexico	1.980024	-0.142849	0.183692	0.339534
New York	1.682577	0.823184	-0.643075	0.013484
North Carolina	1.123379	-2.228003	-0.863572	0.954382
North Dakota	-2.992226	-0.599119	0.301277	0.253987
Ohio	-0.225965	0.742238	-0.031139	-0.473916

	PC1	PC2	PC3	PC4
Oklahoma	-0.311783	0.287854	-0.015310	-0.010332
Oregon	0.059122	0.541411	0.939833	0.237781
Pennsylvania	-0.888416	0.571100	-0.400629	-0.359061
Rhode Island	-0.863772	1.491978	-1.369946	0.613569
South Carolina	1.320724	-1.933405	-0.300538	0.131467
South Dakota	-1.987775	-0.823343	0.389293	0.109572
Tennessee	0.999742	-0.860251	0.188083	-0.652864
Texas	1.355138	0.412481	-0.492069	-0.643195
Utah	-0.550565	1.471505	0.293728	0.082314
Vermont	-2.801412	-1.402288	0.841263	0.144890
Virginia	-0.096335	-0.199735	0.011713	-0.211371
Washington	-0.216903	0.970124	0.624871	0.220848
West Virginia	-2.108585	-1.424847	0.104775	-0.131909
Wisconsin	-2.079714	0.611269	-0.138865	-0.184104
Wyoming	-0.629427	-0.321013	-0.240659	0.166652

10.2 Clustering

In [ ]:

10.2.1 K-Means

In [ ]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=42)

kmeans.fit(x)
```

10.2.2 K-Modes

init :

1. **Huang**
2. **Cao**

In [36]:
```
from kmodes.kmodes import KModes
```

10.2.3 K-Medoid

In [ ]:

10.2.4 K-Prototype

In [ ]:

10.2.5 Hierarchical

- Unlike K-Means, there is no need to pre-determine number of clusters

In [49]:
```
import pandas as pd
import matplotlib.pyplot as plt

from scipy.cluster.hierarchy import linkage, dendrogram
```

In [ ]:
```
nci_labs = pd.read_csv("data/NCI60_labs.csv", index_col=0)
nci_data = pd.read_csv('data/nci60_data.csv', index_col=0)

nci_data.info() # data with a huges number of features (around 6830)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 64 entries, V1 to V64
Columns: 6830 entries, 1 to 6830
dtypes: float64(6830)
memory usage: 3.3+ MB
```

In [ ]:
```
?linkage
```

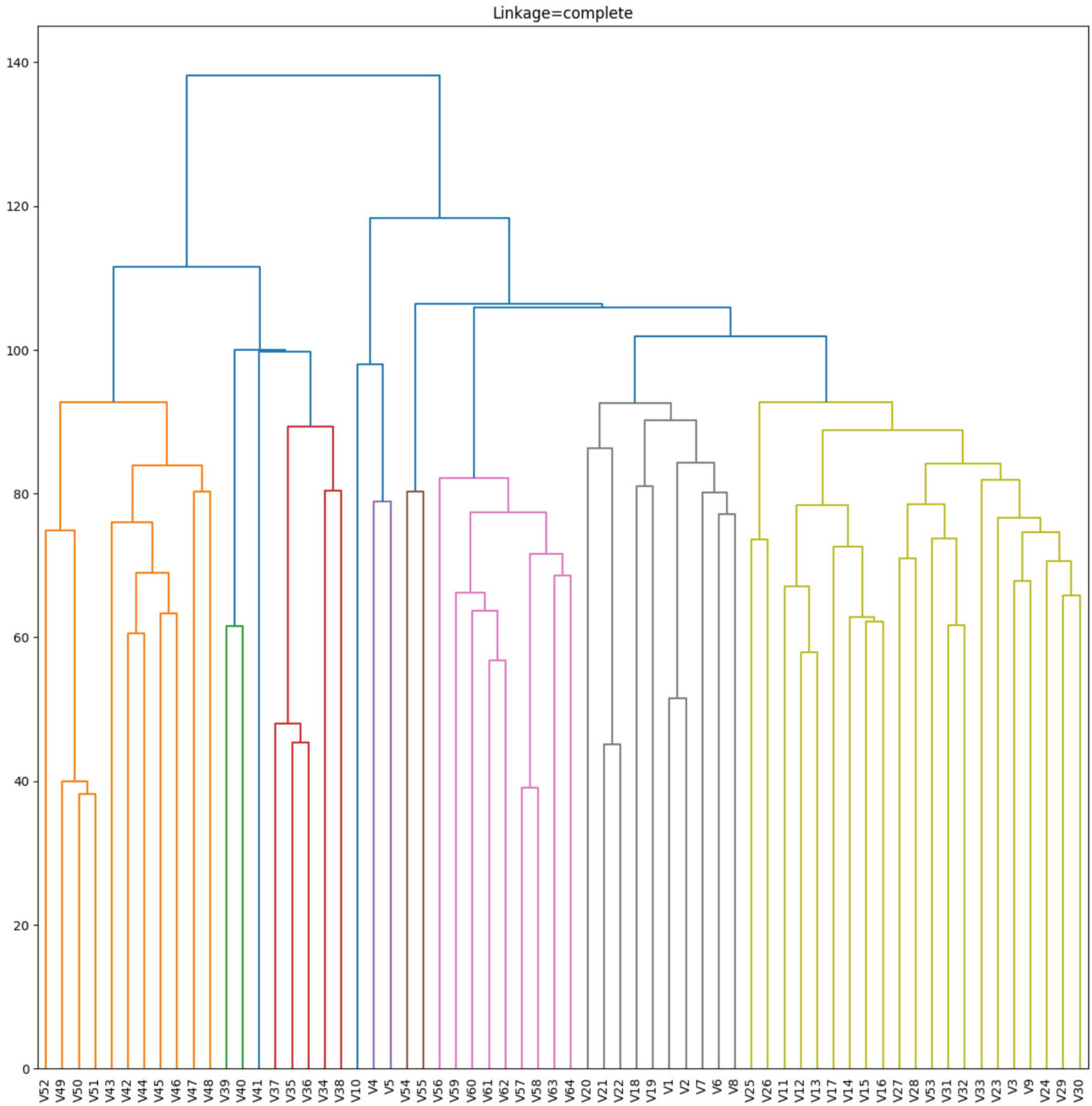
In [ ]:
```
nci_hc_complete = linkage(y=nci_data,
                          method="complete",
                          metric='euclidean')
nci_hc_complete
```

Out[ ]:
```
array([[ 49.        ,  50.        ,  38.23033267,   2.        ],
       [ 56.        ,  57.        ,  39.10562479,   2.        ],
       [ 48.        ,  64.        ,  39.99989876,   3.        ],
       [ 20.        ,  21.        ,  45.15158096,   2.        ],
       [ 34.        ,  35.        ,  45.35338129,   2.        ],
       [ 36.        ,  68.        ,  48.01162819,   3.        ],
       [  0.        ,   1.        ,  51.43823073,   2.        ],
       [ 60.        ,  61.        ,  56.78015373,   2.        ],
       [ 11.        ,  12.        ,  57.91726379,   2.        ],
       [ 41.        ,  43.        ,  60.49650657,   2.        ],
       [ 38.        ,  39.        ,  61.55425326,   2.        ],
       [ 30.        ,  31.        ,  61.63750362,   2.        ],
       [ 14.        ,  15.        ,  62.1780594 ,   2.        ],
       [ 13.        ,  76.        ,  62.78163258,   3.        ],
       [ 44.        ,  45.        ,  63.26414098,   2.        ],
       [ 59.        ,  71.        ,  63.67441112,   3.        ],
       [ 28.        ,  29.        ,  65.76413536,   2.        ],
       [ 58.        ,  79.        ,  66.23109874,   4.        ],
       [ 10.        ,  72.        ,  67.07313093,   3.        ],
       [  2.        ,   8.        ,  67.83789248,   2.        ],
       [ 62.        ,  63.        ,  68.60602282,   2.        ],
       [ 73.        ,  78.        ,  68.99669188,   4.        ],
       [ 23.        ,  80.        ,  70.62273231,   3.        ],
       [ 26.        ,  27.        ,  70.93759619,   2.        ],
       [ 65.        ,  84.        ,  71.59949094,   4.        ],
       [ 16.        ,  77.        ,  72.56828298,   4.        ],
       [ 24.        ,  25.        ,  73.56083029,   2.        ],
       [ 52.        ,  75.        ,  73.67669823,   3.        ],
       [ 83.        ,  86.        ,  74.62701247,   5.        ],
       [ 51.        ,  66.        ,  74.80702422,   4.        ],
       [ 42.        ,  85.        ,  75.92470655,   5.        ],
       [ 22.        ,  92.        ,  76.6317916 ,   6.        ],
       [  5.        ,   7.        ,  77.0683762 ,   2.        ],
       [ 81.        ,  88.        ,  77.32029088,   8.        ],
       [ 82.        ,  89.        ,  78.42221832,   7.        ],
       [ 87.        ,  91.        ,  78.55289235,   5.        ],
       [  3.        ,   4.        ,  78.90728036,   2.        ],
       [  6.        ,  96.        ,  80.11985439,   3.        ],
       [ 53.        ,  54.        ,  80.19940947,   2.        ],
       [ 46.        ,  47.        ,  80.28907807,   2.        ],
       [ 33.        ,  37.        ,  80.35099974,   2.        ],
       [ 17.        ,  18.        ,  80.97092472,   2.        ],
       [ 32.        ,  95.        ,  81.898042  ,   7.        ],
       [ 55.        ,  97.        ,  82.1622036 ,   9.        ],
       [ 94.        , 103.        ,  83.83982605,   7.        ],
       [ 99.        , 106.        ,  84.13402432,  12.        ],
       [ 70.        , 101.        ,  84.32318976,   5.        ],
       [ 19.        ,  67.        ,  86.31140874,   3.        ],
       [ 98.        , 109.        ,  88.85328369,  19.        ],
       [ 69.        , 104.        ,  89.3367509 ,   5.        ],
       [105.        , 110.        ,  90.13572614,   7.        ],
       [111.        , 114.        ,  92.58579286,  10.        ],
       [ 90.        , 112.        ,  92.65795124,  21.        ],
       [ 93.        , 108.        ,  92.70718779,  11.        ],
       [  9.        , 100.        ,  97.94830934,   3.        ],
       [ 40.        , 113.        ,  99.76405492,   6.        ],
       [ 74.        , 119.        ,  99.99018542,   8.        ],
       [115.        , 116.        , 101.82696152,  31.        ],
       [107.        , 121.        , 105.92309815,  40.        ],
       [102.        , 122.        , 106.43870527,  42.        ],
       [117.        , 120.        , 111.51306851,  19.        ],
       [118.        , 123.        , 118.25973072,  45.        ],
       [124.        , 125.        , 138.15044876,  64.        ]])
```

In [59]:
```
fig, axis = plt.subplots(1, 1, figsize=(15,15))
```

```
dendrogram(nci_hc_complete,
           ax=axis,
           labels=nci_data.index,
           leaf_rotation=90,
           leaf_font_size=10)

axis.set_title('Linkage=complete')
```

Out[59]: Text(0.5, 1.0, 'Linkage=complete')



In [62]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

scaler = StandardScaler()

nci_scale = scaler.fit_transform(nci_data)

pca = PCA(n_components=2)

nci_reduce = pca.fit_transform(nci_scale)
pd.DataFrame(nci_reduce)
```

Out[62]:

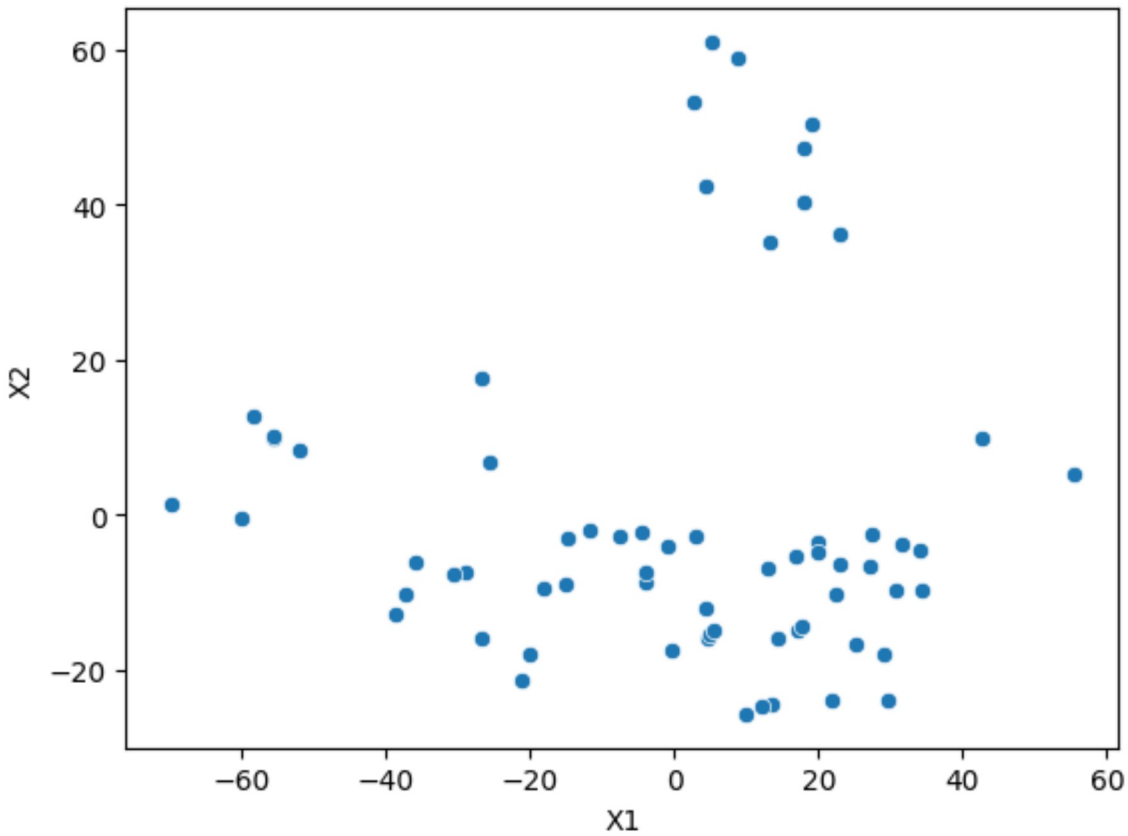
	0	1
0	19.838032	-3.556131
1	23.089217	-6.441309
2	27.456107	-2.465335
3	42.816891	9.770332
4	55.418495	5.198344
...	...	...
59	17.996259	47.242927
60	4.415491	42.309182
61	22.966980	36.101936
62	19.175928	50.396686
63	13.232880	35.125287

64 rows × 2 columns

In [ ]:

```
import seaborn as sns

df = pd.DataFrame(nci_reduce, columns=['X1', 'X2'], index=)

df.columns
sns.scatterplot(x='X1', y='X2', data=df)
```

Out[ ]: <Axes: xlabel='X1', ylabel='X2'>



11 Neural Network

11.1 Classifier

In [37]:
```
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

In [38]:
```
# Location of dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
irisdata = pd.read_csv(url, names=names)

irisdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal-length  150 non-null    float64
 1   sepal-width   150 non-null    float64
 2   petal-length  150 non-null    float64
 3   petal-width   150 non-null    float64
 4   Class         150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [39]:
```
y = irisdata['Class']
x = irisdata.drop('Class', axis=1)

le = LabelEncoder()

y = le.fit_transform(y)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

scale = StandardScaler()
x_train = scale.fit_transform(x_train)
x_test = scale.transform(x_test)

mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), # 3 hidden layer of 10 nodes each
                    max_iter=1000,
                    random_state=42)

mlp.fit(x_train, y_train)

print(f'Accuracy Score: {(accuracy_score(y_train, mlp.predict(x_train)) * 100):.2f}%')

print('==========')

print(classification_report(y_train, mlp.predict(x_train)))

print('==========')

print(pd.DataFrame(confusion_matrix(y_train, mlp.predict(x_train))))
```

```
Accuracy Score: 98.33%
==========
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       0.98      0.98      0.98        41
           2       0.97      0.97      0.97        39

    accuracy                           0.98       120
   macro avg       0.98      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120

==========
    0   1   2
0  40   0   0
1   0  40   1
2   0   1  38
```

11.2 Regressor

In [20]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

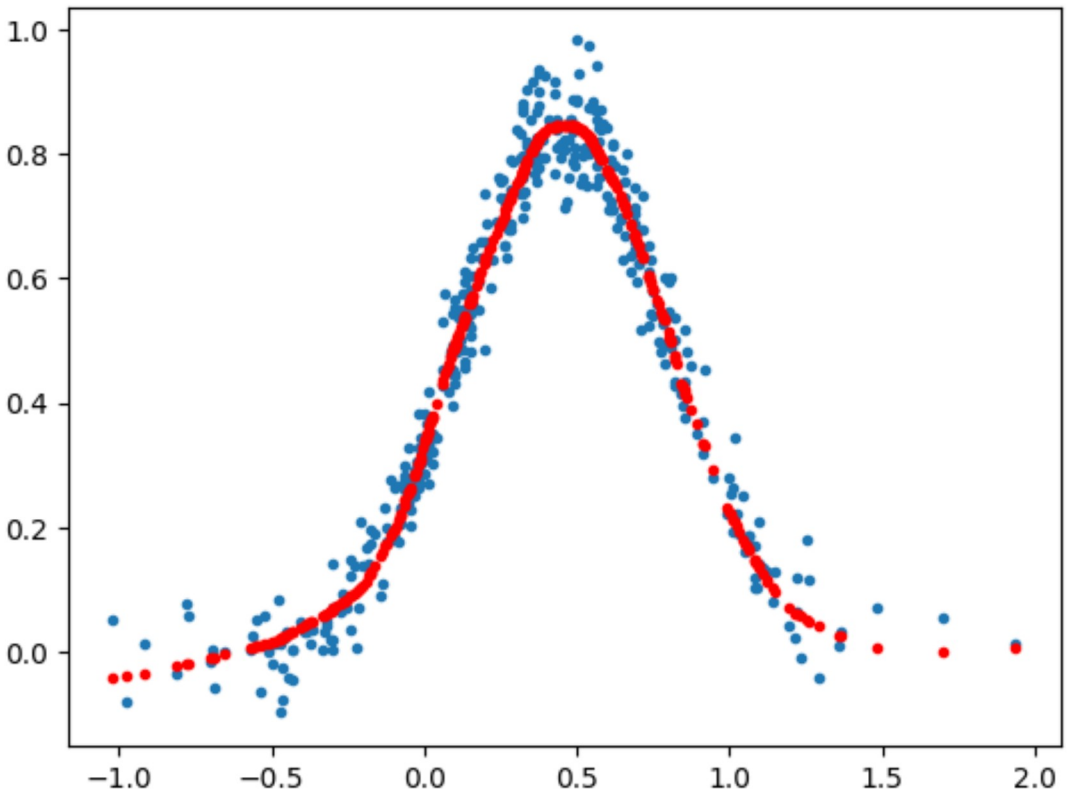
In [8]:
```
def make_data():
    N = 2000
    X = 0.5*np.random.normal(size=N)+0.35

    Xt = 0.75*X-0.35
    X = X.reshape((N,1))

    Y = -(8 * Xt**2 + 0.1*Xt + 0.1) + 0.05 * np.random.normal(size=N)
    Y = np.exp(Y) + 0.05 * np.random.normal(size=N)
    Y /= max(np.abs(Y))
    return X, Y

np.random.seed(0)
x, y = make_data()

x = pd.DataFrame(x)
y = pd.DataFrame(y)
```

In [36]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

scaler = StandardScaler()

x_train_scale = scaler.fit_transform(x_train)
x_test_scale = scaler.transform(x_test)

mlpreg = MLPRegressor(hidden_layer_sizes=(30,30,30), random_state=42)

mlpreg.fit(x_train_scale, y_train)

y_pred = mlpreg.predict(x_test_scale)

plt.plot(x_test, y_test, '.')
plt.plot(x_test, y_pred, 'r.')

print(f'MSE: {mean_squared_error(y_test, y_pred):.4f}')
```

```
print(f'R2 Score: {(r2_score(y_test, y_pred) * 100):.2f}%')
```

```
MSE: 0.0027
R2 Score: 96.90%
```

```
c:\Users\user\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\neural_network
\_multilayer_perceptron.py:1770: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



12 Deep Learning

In [ ]:
```
MLPRegressor(activation='')
```

Test

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
df = pd.read_table('Test/question_2.txt', sep=',')
df.head()
```

Out[ ]:

	No	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	SLUMP(cm)	FLOW(cm)	Compressive Strength (28-day)(Mpa)
0	1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
1	2	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	41.14
2	3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
3	4	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	42.08
4	5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82

In [ ]:

```
# define response (Y) and predictor (X) variables
y = df['Compressive Strength (28-day)(Mpa)']
x = df.drop(['Compressive Strength (28-day)(Mpa)', 'SLUMP(cm)', 'FLOW(cm)', 'No'], axis = 1)
```

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
lr = LinearRegression()

model = lr.fit(x_train, y_train)

var = x.columns.tolist()

print(pd.DataFrame({'Variable': var, 'Coefficient': model.coef_}))
print(f'Intercept: {model.intercept_:.2f}')
```

```
       Variable  Coefficient
0        Cement     4.898330
1          Slag    -1.755212
2       Fly ash     4.658713
3         Water    -4.545214
4            SP     0.423314
5  Coarse Aggr.    -4.638413
6    Fine Aggr.    -2.257704
Intercept: 35.82
```

In [ ]:

```
mse = mean_squared_error(y_train, model.predict(x_train))

print(f'MSE: {mse:.2f}')
```

```
MSE: 6.90
```

In [ ]:

```
?SequentialFeatureSelector
```

```
Init signature:
SequentialFeatureSelector(
    estimator,
    *,
    n_features_to_select='auto',
    tol=None,
    direction='forward',
    scoring=None,
    cv=5,
    n_jobs=None,
)
Docstring:
Transformer that performs Sequential Feature Selection.

This Sequential Feature Selector adds (forward selection) or
removes (backward selection) features to form a feature subset in a
greedy fashion. At each stage, this estimator chooses the best feature to
add or remove based on the cross-validation score of an estimator. In
the case of unsupervised learning, this Sequential Feature Selector
looks only at the features (X), not the desired outputs (y).

Read more in the :ref:`User Guide <sequential_feature_selection>`.

.. versionadded:: 0.24

Parameters
----------
estimator : estimator instance
    An unfitted estimator.

n_features_to_select : "auto", int or float, default="auto"
    If `"auto"`, the behaviour depends on the `tol` parameter:

    - if `tol` is not `None`, then features are selected while the score
      change does not exceed `tol`.
    - otherwise, half of the features are selected.

    If integer, the parameter is the absolute number of features to select.
    If float between 0 and 1, it is the fraction of features to select.

    .. versionadded:: 1.1
       The option `"auto"` was added in version 1.1.

    .. versionchanged:: 1.3
       The default changed from `"warn"` to `"auto"` in 1.3.

tol : float, default=None
    If the score is not incremented by at least `tol` between two
    consecutive feature additions or removals, stop adding or removing.

    `tol` can be negative when removing features using `direction="backward"`.
    `tol` is required to be strictly positive when doing forward selection.
    It can be useful to reduce the number of features at the cost of a small
    decrease in the score.

    `tol` is enabled only when `n_features_to_select` is `"auto"`.

    .. versionadded:: 1.1

direction : {'forward', 'backward'}, default='forward'
    Whether to perform forward selection or backward selection.

scoring : str or callable, default=None
    A single str (see :ref:`scoring_parameter`) or a callable
    (see :ref:`scoring_callable`) to evaluate the predictions on the test set.
```

```
    NOTE that when using a custom scorer, it should return a single
    value.

    If None, the estimator's score method is used.

cv : int, cross-validation generator or an iterable, default=None
    Determines the cross-validation splitting strategy.
    Possible inputs for cv are:

    - None, to use the default 5-fold cross validation,
    - integer, to specify the number of folds in a `(Stratified)KFold`,
    - :term:`CV splitter`,
    - An iterable yielding (train, test) splits as arrays of indices.

    For integer/None inputs, if the estimator is a classifier and ``y`` is
    either binary or multiclass,
    :class:`~sklearn.model_selection.StratifiedKFold` is used. In all other
    cases, :class:`~sklearn.model_selection.KFold` is used. These splitters
    are instantiated with `shuffle=False` so the splits will be the same
    across calls.

    Refer :ref:`User Guide <cross_validation>` for the various
    cross-validation strategies that can be used here.

n_jobs : int, default=None
    Number of jobs to run in parallel. When evaluating a new feature to
    add or remove, the cross-validation procedure is parallel over the
    folds.
    ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
    ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
    for more details.

Attributes
----------
n_features_in_ : int
    Number of features seen during :term:`fit`. Only defined if the
    underlying estimator exposes such an attribute when fit.

    .. versionadded:: 0.24

feature_names_in_ : ndarray of shape (`n_features_in_`,)
    Names of features seen during :term:`fit`. Defined only when `X`
    has feature names that are all strings.

    .. versionadded:: 1.0

n_features_to_select_ : int
    The number of features that were selected.

support_ : ndarray of shape (n_features,), dtype=bool
    The mask of selected features.

See Also
--------
GenericUnivariateSelect : Univariate feature selector with configurable
    strategy.
RFE : Recursive feature elimination based on importance weights.
RFECV : Recursive feature elimination based on importance weights, with
    automatic selection of the number of features.
SelectFromModel : Feature selection based on thresholds of importance
    weights.

Examples
--------
>>> from sklearn.feature_selection import SequentialFeatureSelector
>>> from sklearn.neighbors import KNeighborsClassifier
```

```
>>> from sklearn.datasets import load_iris
>>> X, y = load_iris(return_X_y=True)
>>> knn = KNeighborsClassifier(n_neighbors=3)
>>> sfs = SequentialFeatureSelector(knn, n_features_to_select=3)
>>> sfs.fit(X, y)
SequentialFeatureSelector(estimator=KNeighborsClassifier(n_neighbors=3),
                          n_features_to_select=3)
>>> sfs.get_support()
array([ True, False,  True,  True])
>>> sfs.transform(X).shape
(150, 3)
File:           c:\users\user\appdata\local\programs\python\python313\lib\site-packages\sklear
n\feature_selection\_sequential.py
Type:           ABCMeta
Subclasses:
```

Forward Selection

In [ ]:
```
sfs = SequentialFeatureSelector(lr,
                                direction='forward',
                                scoring='neg_mean_squared_error')

sfs.fit(x_train, y_train)

selected_features = sfs.get_feature_names_out()
print(selected_features)
```

```
['x0' 'x2' 'x3']
```

Backward Elimination

In [ ]:
```
?SequentialFeatureSelector
```

In [ ]:
```
sfs = SequentialFeatureSelector(lr,
                                direction='backward',
                                scoring='neg_mean_squared_error',)
sfs.fit(x_train, y_train)
selected_features = sfs.get_feature_names_out()
print(selected_features)
```

```
['x0' 'x2' 'x3' 'x5']
```

In [ ]:
```
best_model = LinearRegression()
best_model.fit(x_train[selected_features], y_train)

x_pred_best = best_model.predict(x_test[selected_features])
mse_best = mean_squared_error(y_test, x_pred_best)
print(f'Best Model MSE: {mse_best:.2f}')
```

```
Best Model MSE: 7.53
```

In [ ]:
```
ridge = Ridge()

model_ridge = ridge.fit(x_train, y_train)
y_pred_ridge = model_ridge.predict(x_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print(f'Ridge Model MSE: {mse_ridge:.2f}')
print(f'Ridge Coefficients: {pd.DataFrame(model_ridge.coef_)}')
```

```
Ridge Model MSE: 4.13
Ridge Coefficients:           0
0  5.200274
1 -1.297764
2  5.028981
3 -4.088444
4  0.484226
5 -3.959651
6 -1.792877
```

In [ ]:
```
lasso = Lasso()
model_lasso = lasso.fit(x_train, y_train)
y_pred_lasso = model_lasso.predict(x_test)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print(f'Lasso Model MSE: {mse_lasso:.2f}')


print(f'Lasso Coefficients: {pd.DataFrame(model_lasso.coef_)}')
```

```
Lasso Model MSE: 12.55
Lasso Coefficients:           0
0  4.536146
1  0.000000
2  4.648158
3 -1.174593
4  0.000000
5 -0.321955
6 -0.000000
```

Tree Based Method

In [150…
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

In [126…
```
boston = pd.read_csv('data/boston.csv')
boston.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    int64
 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    int64
 9   tax      506 non-null    int64
 10  ptratio  506 non-null    float64
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

In [148…
```
y = boston['medv']
x = boston.drop('medv', axis=1)

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=42)
```

```
scale = StandardScaler()

x_train_scale = scale.fit_transform(x_train)
x_test_scale = scale.fit_transform(x_test)
```

In [149…

```
x_train_scale = pd.DataFrame(x_train_scale, columns=x.columns)

dtr = DecisionTreeRegressor(max_depth=2, random_state=42)
dtr.fit(x_train_scale, y_train)

print(f'MSE: {mean_squared_error(y_train, dtr.predict(x_train))}')
```

```
MSE: 564.4473514851481
```

In [155…

```
# bagging since m = p = 13
bag = RandomForestRegressor(max_features=13, random_state=42)
bag.fit(x_train, y_train)

print(f'MSE: {mean_squared_error(y_train, bag.predict(x_train))}')
```

```
MSE: 1.9783925024752465
```

In [ ]:

```
# Random Forest since m (4) < p (13)
rfr = RandomForestRegressor(max_features=4, random_state=42)
rfr.fit(x_train_scale, y_train)

print(f'MSE: {mean_squared_error(y_train, rfr.predict(x_train))}')
```

```
MSE: 80.29666808663366
```

In [ ]:

```
boost = GradientBoostingRegressor(n_estimators=500, # number of trees
                                  learning_rate=0.01, #
                                  max_depth=4, # depth of each tree
                                  random_state=42)

boost.fit(x_train, y_train)

print(f'MSE: {mean_squared_error(y_train, boost.predict(x_train))}')
```

```
MSE: 1.6231598332847361
```

In [164…

```
importance = (rfr.feature_importances_ * 100).round(2)

imp = pd.DataFrame({'Variable':x.columns,
                    'Importance (%)':importance})

print(imp)
```

```
   Variable  Importance (%)
0      crim            7.24
1        zn            0.39
2     indus            7.48
3      chas            0.87
4       nox            5.36
5        rm           32.12
6       age            2.23
7       dis            6.70
8       rad            1.05
9       tax            2.59
10  ptratio            5.33
11    black            2.11
12    lstat           26.53
```

In [162…

```
np.sum(rfr.feature_importances_) * 100
```

Out[162…

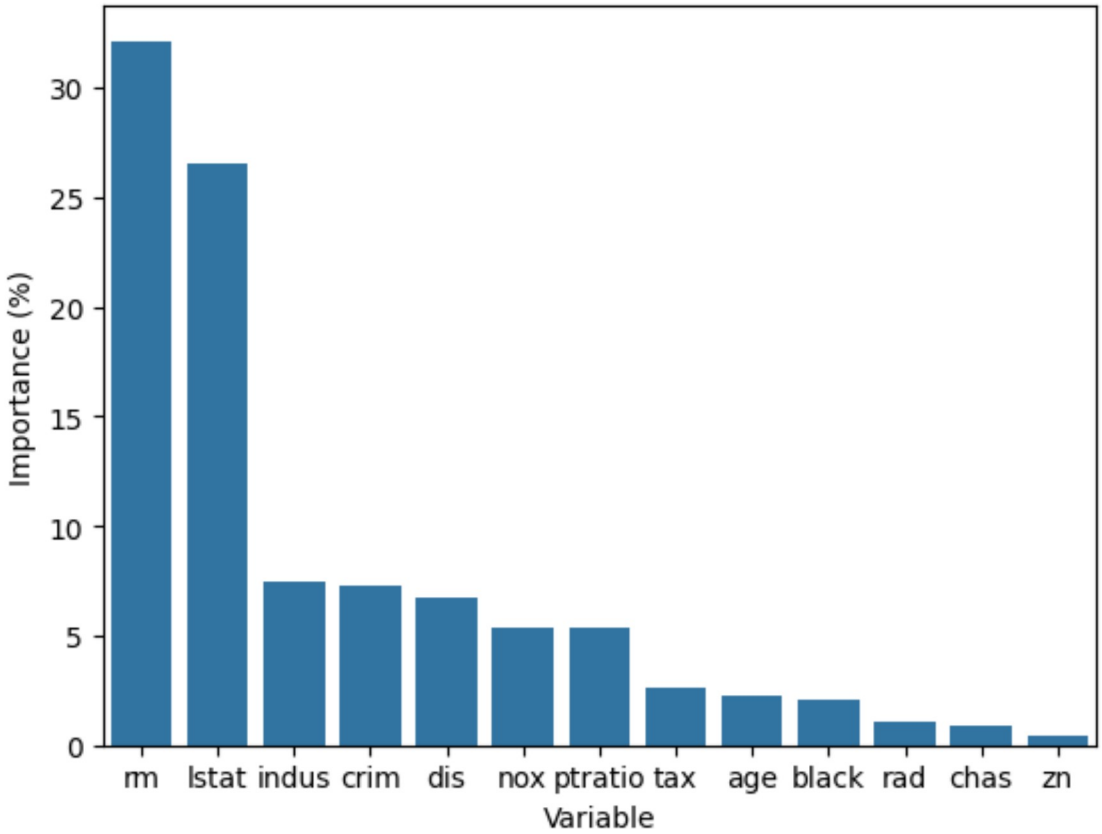
```
100.0
```

In [168…

```
imp = imp.sort_values(by='Importance (%)', ascending=False)
sns.barplot(x='Variable', y='Importance (%)', data = imp)
```

Out[168…

```
<Axes: xlabel='Variable', ylabel='Importance (%)'>
```



In [ ]:

In [ ]:

In [ ]:

In [1]:

```
import pandas as pd

# Create a sample DataFrame
data = {
    'A': [1, 2, 2, 3, 4, 4, 5],
    'B': ['apple', 'banana', 'banana', 'orange', 'grape', 'grape', 'kiwi'],
    'C': [10, 20, 20, 30, 40, 40, 50]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)

# Drop duplicate rows based on all columns
df_no_duplicates = df.drop_duplicates()
print("\nDataFrame after dropping duplicates (all columns, keeping first):")
print(df_no_duplicates)
```

```
Original DataFrame:
   A       B   C
0  1   apple  10
1  2  banana  20
2  2  banana  20
3  3  orange  30
4  4   grape  40
5  4   grape  40
6  5    kiwi  50

DataFrame after dropping duplicates (all columns, keeping first):
   A       B   C
0  1   apple  10
1  2  banana  20
3  3  orange  30
4  4   grape  40
6  5    kiwi  50
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: