Machine Learning with Statology

Introduction to Machine Learning

The field of machine learning contains a massive set of algorithms that can be used for understanding data. These algorithms can be classified into one of two categories:

1. **Supervised Learning Algorithms :** Involves building a model to estimate or predict an output based on one or more inputs.
2. **Unsupervised Learning Algorithms :** Involves finding structure and relationships from inputs. There is no "supervising" output.

Supervised Learning Algorithms

A **Supervised Learning Algorithms** can be used when we have one or more *explanatory variables* $(X_1, X_2, X_3, \cdots, X_p)$ and a *response variable* (Y) and we would like to find some function that describes the relationship between the explanatory variables and the response variable:

$$Y = f(X) + \epsilon$$

where, **f** represents systematic information that **X** provides about Y and where **ε** is a random error term independent of X with a mean of zero.

Thee are two main types of *Supervised Learning algoruths :*

1. **Regression :** The output variable is continuous.
   (e.g. weight, height, time, etc.)

2. **Classification :** The output variable is categorical.
   (e.g. male or female, pass or fail, benign or malignant, etc.)

There are two main reasons that we use supervised learning algorithms:

A. **Prediction :** We often use a set of explanatory variables to predict the value of some response variable .
(e.g. using square footage and number of bedrooms to predict home price)

B. **Inference :** We may be interested in understanding the way that a response variable is affected as the value of the explanatory variables change .
(e.g. how much does home price increase, on average, when the number of bedrooms increases by one ?)

Depending on whether our goal is inference or prediction (or a mix of both), we may use different methods for estimating the function f.

For example, linear models offer easier interpretation but non-linear models that are difficult to interpret may offer more accurate prediction.

Hee is list of most commonly used supervised learning algorithms :

- **Linear Regression**
- **Logistic Regression**
- **Linear Discriminant Analysis**
- **Quadratic Discriminant Analysis**
- **Decision Trees**
- **Naive Bayes**
- **Support Vector Machines**
- **Neural Networks**

Unsupervised Learning Algorithms

An **Unsupervised Learning Algorithm** can be used when we have a list of variables $(X_1, X_2, X_3, \cdots, X_p)$ and we would simply like to find underlying structure or patterns within the data.

There are two main types of unsupervised learning algorithms:

1. **Clustering :** Using these types of algorithms, we attempt to find "clusters" of observations in a dataset that are similar to each other. This is often used in retail when a company would like to identify clusters of customers who have similar shopping habits so that they can create specific marketing strategies that target certain clusters of customers.

2. **Association :** Using these types of algorithms, we attempt to find "rules" that can be used to draw associations. For example, retailers may develop an association algorithm that says "if a customer buys product X they are highly likely to also buy product Y."

Here is a list of the most commonly used unsupervised learning algorithms:

- **Principal Component Analysis**
- **K-Means Clustering**
- **K-Medoids Clustering**
- **Hierarchical Clustering**
- **Apriori Algorithm**

Summary : Supervised Vs. Unsupervised Learning

In [2]:
```
from IPython.display import Image
Image(filename='C:\\Users\\ACER\\Pictures\\Screenshots\dif.png')
```

Out[2]:

Summary: Supervised vs. Unsupervised Learning

The following table summarizes the differences between supervised and unsupervised learning algorithms:

	Supervised Learning	Unsupervised Learning
Description	Involves building a model to estimate or predict an output based on one or more inputs.	Involves finding structure and relationships from inputs. There is no "supervising" output.
Variables	Explanatory and Response variables	Explanatory variables only
End goal	Develop model to **(1)** predict new values or **(2)** understand existing relationship between explanatory and response variables	Develop model to **(1)** place observations from a dataset into a specific cluster or to **(2)** create rules to identify associations between variables.
Types of algorithms	**(1)** Regression and **(2)** Classification	**(1)** Clustering and **(2)** Association

Regression

Regression : The response variable is continuous.
For example, the response variable could be:

- Weight
- Height
- Price
- Time
- Total units
  In each case, a regression model seeks to predict a continuous quantity.

Regression Example :
i. Suppose we have a dataset that contains three variables for 100 different houses: square footage, number of bathrooms, and selling price.

ii. We could fit a regression model that uses square footage and number of bathrooms as explanatory variables and selling price as the response variable.

iii. We could then use this model to predict the selling price of a house, based on its square footage and number of bathrooms.

iv. This is an example of a regression model because the response variable (selling price) is continuous.

The most common way to measure the **accuracy** of a **regression model** is by calculating the **Root Mean Square Error (RMSE)** , a metric that tells us how far apart our predicted values are from our observed values in a model, on average.

It is calculated as:

$$RMSE = \sqrt{\frac{\sum(P_i - O_i)}{n}}$$

where, $\sum$ is a fancy symbol that means "sum"
P_i is the predicted value for the ith observation
O_i is the observed value for the ith observation
n is the sample size

The *smaller* the *RMSE* , the better a regression model is able to fit the data .

Classification

Classification : The response variable is categorical.

For example, the response variable could take on the following values :

- Male or Female
- Pass or Fail
- Low, Medium, or High
  In each case, a classification model seeks to predict some class label.

Classification Example :

1. Suppose we have a dataset that contains three variables for 100 different college basketball players: average points per game, division level, and whether or not they got drafted into the NBA.
2. We could fit a classification model that uses average points per game and division level as explanatory variables and "drafted" as the response variable.
3. We could then use this model to predict whether or not a given player will get drafted into the NBA based on their average points per game and division level.
4. This is an example of a classification model because the response variable ("drafted") is categorical. That is, it can only take on values in two different categories: "Drafted" or "Not drafted."

The most common way to measure the **accuracy** of a *classification model* is by simply calculating the *percentage of correct classifications* the model makes :

$$Accuracy = \frac{Correction\ \ Classifications}{Total\ \ Attempted\ \ Classifications} \times 100$$

For example, if a model correctly identifies whether or not a player will get drafted into the NBA 88 times out of 100 possible times then the accuracy of the model is :
$Accuracy = \frac{88}{100} \times 100$

The **higher** the accuracy, the better a classification model is able to predict outcomes.

Similarities Between Regression and Classification :
Regression and classification algorithms are similar in the following ways:

- Both are supervised learning algorithms, i.e. they both involve a response variable.
- Both use one or more explanatory variables to build models to predict some response.
- Both can be used to understand how changes in the values of explanatory variables affect the values of a response variable.

Differences Between Regression and Classification :
Regression and classification algorithms are different in the following ways:

- Regression algorithms seek to predict a continuous quantity and classification algorithms seek to predict a class label.
- The way we measure the accuracy of regression and classification models differs.

Converting Regression into Classification :
It's worth noting that a regression problem can be converted into a classification problem by simply *discretizing* the response variable into buckets.

For example, suppose we have a dataset that contains three variables: square footage, number of bathrooms, and selling price.
We could build a regression model using square footage and number of bathrooms to predict selling price.
However, we could discretize selling price into three different classes:
*$*80k – *$*160k: "Low selling price"
*$*161k – *$*240k: "Medium selling price"
*$*241k – *$*320k: "High selling price"
We could then use square footage and number of bathrooms as explanatory variables to predict which class (low, medium or high) that a given house selling price will fall in.
This would be an example of a classification model since we're attempting to place each house in a class.

Bias-Variance Tradeoff in Machine Learning

To evaluate the performance of a model on a dataset, we need to measure how well the model predictions match the observed data.

For regression models, the most commonly used metric is the *mean squared error (MSE)* , which is calculated as :

$$MSE = \frac{1}{n}\sum(y_i - f(X_i))^2$$

where, n : Total number of observations
y_i : The response value of the ith observation
$f(X_i)$: The predicted response value of the ith observation
The closer the model predictions are to the observations, the smaller the MSE will be.

However, we only care about **test MSE** – the MSE when our model is applied to unseen data. This is because we only care about how the model will perform on unseen data, not existing data.
For example, it's nice if a model that predicts stock market prices has a low MSE on historical data, but we really want to be able to use the model to accurately forecast future data.

It turns out that the test MSE can always be decomposed into two parts:

1. ***The Variance :*** Refers to the amount by which our function f would change if we estimated it using a different training set.
2. ***The Bias :*** Refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.

Written in mathematical terms:
$Test\ MSE = Var(\hat{f}\,(x_0)) + [Bias(\hat{f}\,(x_0))]^2 + Var(\epsilon)$
$Test\ MSE = Variance + Bias^2 + Irreducibleerror$

The third term, the irreducible error, is the error that cannot be reduced by any model simply because there always exists some noise in the relationship between the set of explanatory variables and the *response variable*.

Models that have **high bias** tend to have **low variance**.
For example, linear regression models tend to have high bias (assumes a simple linear relationship between explanatory variables and response variable) and low variance (model estimates won't change much from one sample to the next).

However, models that have **low bias** tend to have **high variance**.
For example, complex non-linear models tend to have low bias (does not assume a certain relationship between explanatory variables and response variable) with high variance (model estimates can change a lot from one training sample to the next).

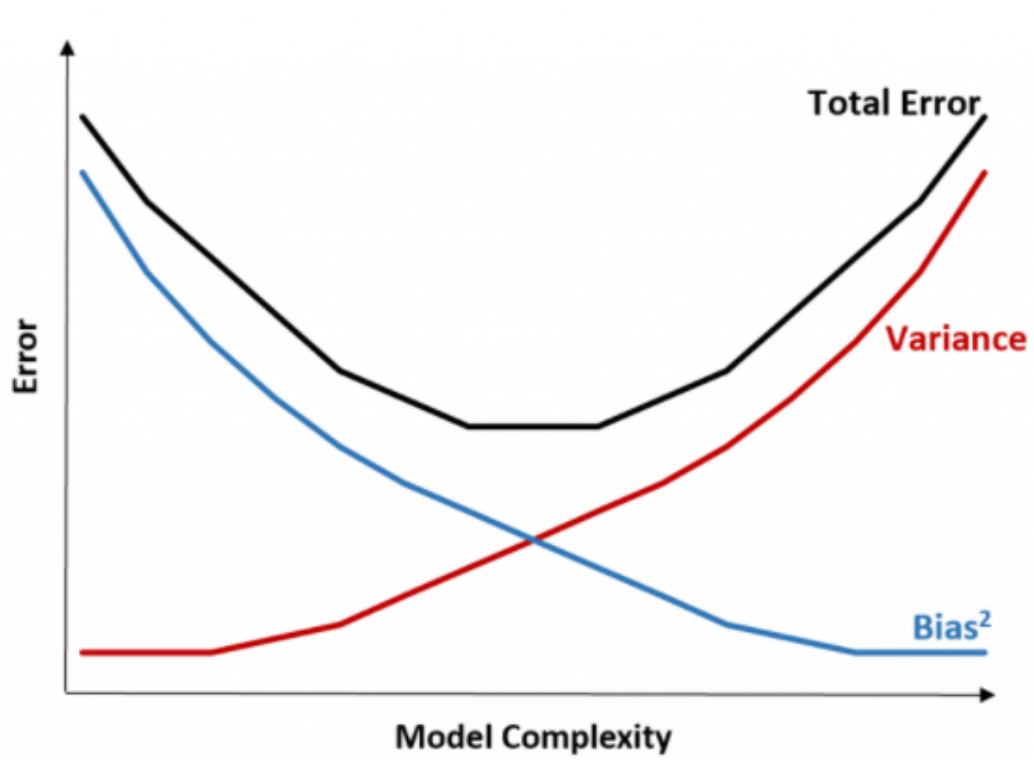
The Bias-Variance Tradeoff

The **Bias-Variance Tradeoff** refers to the tradeoff that takes place when we choose to lower bias which typically increases variance, or lower variance which typically increases bias.

In [9]:
```
from IPython.display import Image
Image(filename='E:\\fig.png')
```

Out[9]:

The following chart offers a way to visualize this tradeoff:



The total error decreases as the complexity of a model increases but only up to a certain point. Past a certain point, variance begins to increase and total error also begins to increase.

In practice, we only care about minimizing the total error of a model, not necessarily minimizing the variance or bias. It turns out that the way to minimize the total error is to strike the right balance between variance and bias.

In other words, we want a model that is complex enough to capture the true relationship between the explanatory variables and the response variable, but not overly complex such that it finds patterns that don't really exist.

When a model is too complex, it **overfits** the data. This happens because it works too hard to find patterns in the training data that are just caused by random chance. This type of model is likely to perform poorly on unseen data.

But when a model is too simple, it **underfits** the data. This happens because it assumes the true relationship between the explanatory variables and the response variable is more simple than it actually is.

The way to pick optimal models in machine learning is to strike the balance between bias and variance such that we can minimize the test error of the model on future unseen data.

In practice, the most common way to minimize test MSE is to use cross-validation.

Linear Regression

Simple Linear Regression

Simple linear regression is a statistical method you can use to understand the relationship between two variables, x and y.
One variable, **x**, is known as the predictor variable.
The other variable, **y**, is known as the response variable.

For example, suppose we have the following dataset with the weight and height of seven individuals :

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

In [7]:
```
df = pd.read_csv("E:\\AMU M.Sc     (Data Sciene)\\D - 1003 RA & PM\\Source Files\\Adver
df.head()
```

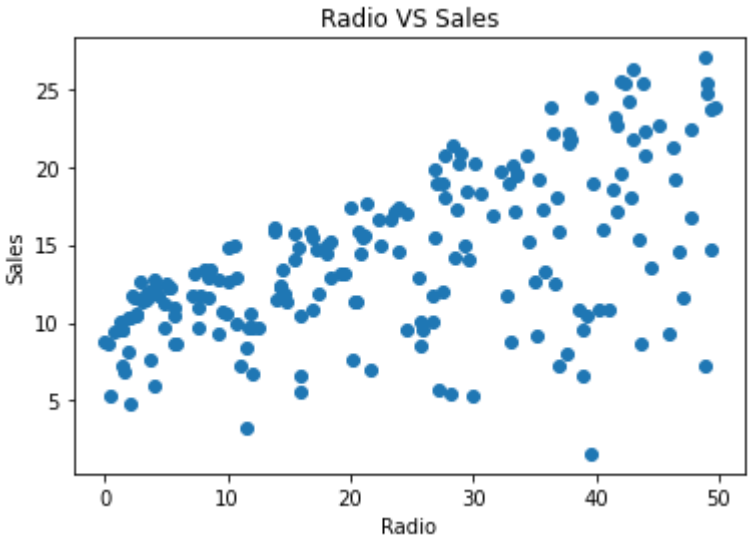
Out[7]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

Let weight be the predictor variable and let height be the response variable.

If we graph these two variables using a *scatterplot* , with weight on the x-axis and height on the y-axis, here's what it would look like :

In [10]:
```
plt.scatter(df['radio'], df['sales'])
plt.xlabel("Radio")
plt.ylabel("Sales")
plt.title("Radio VS Sales")
plt.show()
```



Suppose we're interested in understanding the relationship between weight and height.
From the scatterplot we can clearly see that as weight increases, height tends to increase as well, but to actually *quantify* this relationship between weight and height, we need to use linear regression.

Using linear regression, we can find the line that best "fits" our data. This line is known as the **least squares regression line** and it can be used to help us understand the relationships between weight and height.
Usually you would use software like Microsoft Excel, SPSS, or a graphing calculator to actually find the equation for this line.

The formula for the line of best fit is written as:

$$y = \beta_0 + \beta_1 x + \epsilon \quad \Rightarrow \quad \hat{y} = \beta_0 + \beta_1 x$$

where $\hat{y}$ is the predicted value of the response variable, β_0 is the y-intercept, β_1 is the regression coefficient, and x is the value of the predictor variable.

Model : $Sales = \beta_0 + \beta_1 \times Radio + \epsilon$

In [6]:
```
# Import library
import statsmodels.api as sm

#define response variable
y = df['sales']

#define predictor variables
x = df[['radio']]

#add constant to predictor variables
x = sm.add_constant(x)

#fit linear regression model
model = sm.OLS(y, x).fit()

#view model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  sales   R-squared:                       0.332
Model:                            OLS   Adj. R-squared:                  0.329
Method:                 Least Squares   F-statistic:                     98.42
Date:                Sat, 12 Mar 2022   Prob (F-statistic):           4.35e-19
Time:                        19:00:34   Log-Likelihood:                -573.34
No. Observations:                 200   AIC:                             1151.
Df Residuals:                     198   BIC:                             1157.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          9.3116      0.563     16.542      0.000       8.202      10.422
radio          0.2025      0.020      9.921      0.000       0.162       0.243
==============================================================================
Omnibus:                       19.358   Durbin-Watson:                   1.946
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               21.910
Skew:                          -0.764   Prob(JB):                     1.75e-05
Kurtosis:                       3.544   Cond. No.                         51.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

1. **Fittted Model :** $\widehat{Sales} = 9.31 + 0.2025 \times Radio$
2. R^2 : 0.33

Multiple Linear Regression

If we have **p** predictor variables, then a Multiple Linear Regression Model takes the form :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

where, Y : The Response Variable
X_j : The j^{th} Predictor Variable
β_j : The average effect on Y of a one unit increase in X_j, holding all other predictors fixed
ϵ : The Error term

The values for $\beta_0, \beta_1, \beta_2, \cdots, \beta_p$ are chosen using the least square method, which minimizes the **Sum of Squared Residuals (RSS)** :

$$RSS = \sum (Y_i - \hat{Y}_i)^2$$

where, $\sum$: A greek symbol that means sum
Y_i : The actual response value for the ith observation
$\hat{Y}_i$: The predicted response value based on the multiple linear regression model

Our Model : $Sales = \beta_0 + \beta_1 \times radio + \beta_2 \times newspaper + \beta_3 \times TV + \epsilon$

In [11]:
```
# Import library
```

```
import statsmodels.api as sm

#define response variable
y = df['sales']

#define predictor variables
x = df[['radio', 'newspaper', 'TV']]

#add constant to predictor variables
x = sm.add_constant(x)

#fit linear regression model
model = sm.OLS(y, x).fit()

#view model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  sales   R-squared:                       0.897
Model:                            OLS   Adj. R-squared:                  0.896
Method:                 Least Squares   F-statistic:                     570.3
Date:                Sat, 12 Mar 2022   Prob (F-statistic):           1.58e-96
Time:                        21:45:31   Log-Likelihood:                -386.18
No. Observations:                 200   AIC:                             780.4
Df Residuals:                     196   BIC:                             793.6
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.9389      0.312      9.422      0.000       2.324       3.554
radio          0.1885      0.009     21.893      0.000       0.172       0.206
newspaper     -0.0010      0.006     -0.177      0.860      -0.013       0.011
TV             0.0458      0.001     32.809      0.000       0.043       0.049
==============================================================================
Omnibus:                       60.414   Durbin-Watson:                   2.084
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              151.241
Skew:                          -1.327   Prob(JB):                     1.44e-33
Kurtosis:                       6.332   Cond. No.                         454.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

Our Fitted Model :

$$\widehat{Sales} = 2.939 + 0.1885 \times radio + (-)0.0010 \times newspaper + 0.0458 \times TV$$

R^2 : 0.896

Classification

Logistic Regression

When we want to understand the relationship between one or more predictor variables and a continuous response variable, we often use **linear regression**.

However, when the *response variable* is *categorical* we can instead use *logistic regression*.

Logistic Regression is a type of classification algorithm because it attempts to "classify" observations from a dataset into distinct categories.

Assumptions of Logistic Regression
Logistic regression uses the following assumptions :

1. **The response variable is binary :** It is assumed that the response variable can only take on two possible outcomes.
2. **The observations are independent :** It is assumed that the observations in the dataset are independent of each other. That is, the observations should not come from repeated measurements of the same individual or be related to each other in any way.
3. **There is no severe multicollinearity among predictor variables :** It is assumed that none of the predictor variables are highly correlated with each other.
4. **There are no extreme outliers :** It is assumed that there are no extreme outliers or influential observations in the dataset.
5. **There is a linear relationship between the predictor variables and the logit of the response variable :** This assumption can be tested using a Box-Tidwell test.
6. **The sample size is sufficiently large :** As a rule of thumb, you should have a minimum of 10 cases with the least frequent outcome for each explanatory variable.
   For example, if you have 3 explanatory variables and the expected probability of the least frequent outcome is 0.20, then you should have a sample size of at least (10*3) / 0.20 = 150.

Here are a few examples of when we might use logistic regression:

- We want to use credit score and bank balance to predict whether or not a given customer will default on a loan. (Response variable = "Default" or "No default")
- We want to use average rebounds per game and average points per game to predict whether or not a given basketball player will get drafted into the NBA (Response variable = "Drafted" or "Not Drafted")
- We want to use square footage and number of bathrooms to predict whether or not a house in a certain city will be listed at a selling price of 200k or more. (Response variable = "Yes" or "No")

Notice that the response variable in each of these examples can only take on one of two values. Contrast this with linear regression in which the response variable takes on some continuous value.

The Logistic Regression Equation :
Logistic regression uses a method known as maximum likelihood estimation (details will not be covered here) to find an equation of the following form:

$$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

where, X_j : The j^{th} predictor variable
β_j : The coefficient estimate for the jth predictor variable
The formula on the right side of the equation predicts the **log odds** of the response variable taking on a value of 1.
Thus, when we fit a logistic regression model we can use the following equation to calculate the probability that a given observation takes on a value of 1:

$$p(x) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_p X_p}}$$

We then use some probability threshold to classify the observation as either 1 or 0.

For example, we might say that observations with a probability greater than or equal to 0.5 will be classified as "1" and all other observations will be classified as "0."

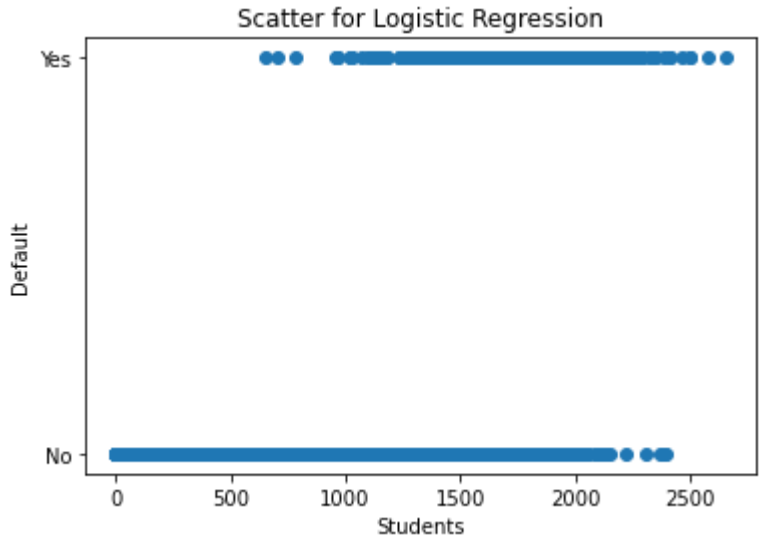
In [1]:
```
import pandas as  pd
import matplotlib.pyplot as plt
```

In [16]:
```
# Load the data
df = pd.read_csv("D:\\AMU M.Sc     (Data Sciene)\\default.csv")
df.head()
```

Out[16]:

	Unnamed: 0	default	student	balance	income
0	1	No	No	729.526495	44361.625074
1	2	No	Yes	817.180407	12106.134700
2	3	No	No	1073.549164	31767.138947
3	4	No	No	529.250605	35704.493935
4	5	No	No	785.655883	38463.495879

In [3]:
```
# Scatter plot
plt.scatter(df['balance'], df['default'])
plt.xlabel('Students')
plt.ylabel("Default")
plt.title("Scatter for Logistic Regression")
plt.show()
```



Our Model : $default = \frac{e^{\beta_0+\beta_1 \times balance}}{1+e^{\beta_0+\beta_1 \times balance}}$

In [4]:
```
# Perform Logistic Regression
import numpy as np
from sklearn.linear_model import LogisticRegression as logit
x = df.balance
x = np.array(x).reshape(-1, 1)
y = df.default
log_reg = logit().fit(x, y)
log_reg.coef_
```

Out[4]:
```
array([[0.00549892]])
```

In [6]:
```
# Here we convert our categorical (defaul) variable into dummy variable
df = pd.get_dummies(df, columns=['default'], drop_first=False)
df.head()
```

Out[6]:

	Unnamed: 0	student	balance	income	default_No	default_Yes
0	1	No	729.526495	44361.625074	1	0
1	2	Yes	817.180407	12106.134700	1	0
2	3	No	1073.549164	31767.138947	1	0
3	4	No	529.250605	35704.493935	1	0
4	5	No	785.655883	38463.495879	1	0

In [7]:
```
# importing libraries
import statsmodels.api as sm
# defining the dependent and independent variables
Xtrain = df.balance
ytrain = df.default_Yes

# building the model and fitting the data
log_reg = sm.Logit(ytrain, Xtrain).fit()
```

```
# printing the summary table
print(log_reg.summary())
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\discrete\discrete_model.py:1810:
RuntimeWarning: overflow encountered in exp
  return 1/(1+np.exp(-X))
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\discrete\discrete_model.py:1863:
RuntimeWarning: divide by zero encountered in log
  return np.sum(np.log(self.cdf(q*np.dot(X,params))))
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\base\model.py:547: HessianInversi
onWarning: Inverting hessian failed, no bse or cov_params available
  warnings.warn('Inverting hessian failed, no bse or cov_params '
Optimization terminated successfully.
         Current function value: inf
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:            default_Yes   No. Observations:                10000
Model:                          Logit   Df Residuals:                    9999
Method:                           MLE   Df Model:                           0
Date:                Tue, 03 May 2022   Pseudo R-squ.:                    inf
Time:                        23:01:11   Log-Likelihood:                   -inf
converged:                       True   LL-Null:                        0.0000
Covariance Type:            nonrobust   LLR p-value:                      nan
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
balance       -0.0028   5.16e-05    -54.773      0.000      -0.003      -0.003
==============================================================================
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\base\model.py:547: HessianInversi
onWarning: Inverting hessian failed, no bse or cov_params available
  warnings.warn('Inverting hessian failed, no bse or cov_params '
```

Fitted Model : $\hat{p}(default) = \frac{e^{-10.65+0.0055(balance)}}{1+e^{-10.65+0.0055(balance)}}$

Interpretation : A one-unit increase in **balance** is associated with an increase in the log odds of **default** by **0.0055** units. **OR**

$\hat{p}(default) = \frac{e^{-10.65+0.0055\times1000}}{1+e^{-10.65+0.0055\times1000}} = 0.00576$

1000 unit increase in **balance** is associated with an increase in the log odds of **default** by **5** units with probability **0.00576** .

Multiple Logistic Regression Using R

$$p(X) = \frac{e^{\beta_0+\beta_1X_1+...+\beta_pX_p}}{1+e^{\beta_0+\beta_1X_1+...+\beta_pX_p}}$$

Model : $p(default) = \frac{e^{\beta_0+\beta_1(balance)+\beta_2(income)+\beta_3(student)}}{1+e^{\beta_0+\beta_1(balance)+\beta_2(income)+\beta_3(student)}}$

In [18]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn import metrics
import matplotlib.pyplot as plt
```

In [19]:

```
#import dataset from CSV file on Github
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/default.csv"
df = pd.read_csv(url)

#view first six rows of dataset
df.head()
```

Out[19]:

	default	student	balance	income
0	0	0	729.526495	44361.625074
1	0	1	817.180407	12106.134700
2	0	0	1073.549164	31767.138947
3	0	0	529.250605	35704.493935
4	0	0	785.655883	38463.495879

In [21]:

```
#define the predictor variables and the response variable
X = df[['student', 'balance', 'income']]
y = df['default']

#split the dataset into training (70%) and testing (30%) sets
# X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)

#instantiate the model
log_regression = LogisticRegression()

#fit the model using the training data
log_regression.fit(X , y)
```

Out[21]:

```
LogisticRegression()
```

In [22]:

```
log_regression.coef_
```

Out[22]:

```
array([[-3.89009045e+00,  4.08201022e-03, -1.33893466e-04]])
```

In [24]:

```
y_pred = log_regression.predict(X)
y_pred
```

Out[24]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [28]:

```
# Model Diagnostics
# Confusion Matrix
cnf_matrix = metrics.confusion_matrix(y , y_pred)
cnf_matrix
```

Out[28]:

```
array([[9609,   58],
       [ 271,   62]], dtype=int64)
```

From the confusion matrix we can see that:

- True positive predictions: 9609
- True negative predictions: 62
- False positive predictions: 271
- False negative predictions: 58

In [29]:
```
print("Accuracy:",metrics.accuracy_score(y, y_pred))
```

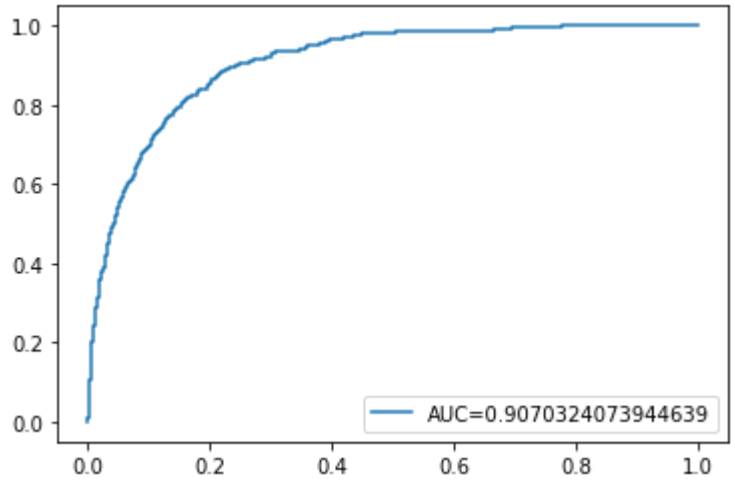
```
Accuracy: 0.9671
```

This tells us that the model made the correct prediction for whether or not an individual would default **96.2\%** of the time.

Lastly, we can plot the ROC (Receiver Operating Characteristic) Curve which displays the percentage of true positives predicted by the model as the prediction probability cutoff is lowered from 1 to 0.

The higher the AUC (area under the curve), the more accurately our model is able to predict outcomes:

In [30]:
```
#define metrics
y_pred_proba = log_regression.predict_proba(X)[::,1]
fpr, tpr, _ = metrics.roc_curve(y,  y_pred_proba)
auc = metrics.roc_auc_score(y, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.legend(loc=4)
plt.show()
```



Linear Discriminant Analysis (LDA)

When we have a set of predictor variables and we'd like to classify a *response variable* into one of two classes, we typically use *logistic regression*.

For example, we may use logistic regression in the following scenario :

- We want to use credit score and bank balance to predict whether or not a given customer will default on a loan. (Response variable = "Default" or "No default")
  However, when a response variable has more than two possible classes then we typically prefer to use a method known as *linear discriminant analysis*, often referred to as LDA.

For example, we may use LDA in the following scenario:
We want to use points per game and rebounds per game to predict whether a given high school basketball player will get accepted into one of three schools: Division 1, Division 2, or Division 3.
Although LDA and logistic regression models are both used for classification, it turns out that LDA is far more stable than logistic regression when it comes to making predictions for multiple classes and is therefore the preferred algorithm to use when the response variable can take on more than two classes.

LDA also performs better when sample sizes are small compared to logistic regression, which makes it a preferred method to use when you're unable to gather large samples.

How to Build LDA Models :
LDA makes the following assumptions about a given dataset:

1. The values of each predictor variable are *normally distributed*. That is, if we made a histogram to visualize the distribution of values for a given predictor, it would roughly have a "bell shape."

2. Each predictor variable has the *same variance*. This is almost never the case in real-world data, so we typically scale each variable to have the same mean and variance before actually fitting a LDA model.

Once these assumptions are met, LDA then estimates the following values :

- μ_l : The meaan of all training observations from the k^{th} class .
- σ^2 : The weighted average of the samples variances for each of the k classes .
- π_k : The proportion of the training oservations that belong to the k^{th} class.

LDA then plugs thses numbers into the following formula and assigns each observation *X=x* to the class for which the formula produces the largest value :

$$D_k(x) = x \times \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{\sigma_2^2} + log\left(\pi_k\right)$$

Note that *LDA* has *linear* in its name because the value produced by the function above comes from a result of *linear functions of x*.

How to Prepare Data for LDA :

Make sure your data meets the following requirements before applying a LDA model to it :

1.**The response variable is categorical :** LDA models are designed to be used for classification problems, i.e. when the response variable can be placed into classes or categories.

1. **The predictor variables follow a normal distribution :** First, check that each predictor variable is roughly normally distributed. If this is not the case, you may choose to first *transform the data* to make the distribution more normal.

1. **Each predictor variable has the same variance :** As mentioned earlier, LDA assumes that each predictor variable has the same variance. Since this is rarely the case in practice, it's a good idea to scale each variable in the dataset such that it has a mean of 0 and a standard deviation of 1.

2. **Account for extreme outliers :** Be sure to check for extreme outliers in the dataset before applying LDA. Typically you can check for outliers visually by simply using *boxplots* or *scatterplots*.

Examples of Using Linear Discriminant Analysis :
LDA models are applied in a wide variety of fields in real life. Some examples include :

1. **Marketing :** Retail companies often use LDA to classify shoppers into one of several categories. For example, they may build an LDA model to predict whether or not a given shopper will be a low spender, medium spender, or high spender using predictor variables like income, total annual spending, and household size.

2. **Medical :** Hospitals and medical research teams often use LDA to predict whether or not a given group of abnormal cells is likely to lead to a mild, moderate, or severe illness.

3. **Product Development :** Companies may build LDA models to predict whether a certain consumer will use their product daily, weekly, monthly, or yearly based on a variety of predictor variables like gender, annual income, and frequency of similar product usage.

4. **Ecology :** Researchers may build LDA models to predict whether or not a given coral reef will have an overall health of good, moderate, bad, or endangered based on a variety of predictor variables like size, yearly contamination, and age.

Step - 1. Load the require libraries to perform a *LDA*

In [1]:
```
# Import the require libraries to perform LDA
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

Step - 2. Load the Dataset and Convert it for use

In [2]:
```
#Load iris dataset
iris = datasets.load_iris()

#convert dataset to pandas DataFrame
df = pd.DataFrame(data = np.c_[iris['data'], iris['target']],
```

```
                    columns = iris['feature_names'] + ['target'])

df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

df.columns = ['s_length', 's_width', 'p_length', 'p_width', 'target', 'species']

#view first six rows of DataFrame
df.head()
```

Out[2]:

	s_length	s_width	p_length	p_width	target	species
0	5.1	3.5	1.4	0.2	0.0	setosa
1	4.9	3.0	1.4	0.2	0.0	setosa
2	4.7	3.2	1.3	0.2	0.0	setosa
3	4.6	3.1	1.5	0.2	0.0	setosa
4	5.0	3.6	1.4	0.2	0.0	setosa

Step - 3. Fit the LDA Model

In [3]:

```
#define predictor and response variables
X = df[['s_length', 's_width', 'p_length', 'p_width']]
y = df['species']

#Fit the LDA model
model = LinearDiscriminantAnalysis()
model.fit(X, y)
```

Out[3]:

```
LinearDiscriminantAnalysis()
```

Step - 4. Use the Model to Make Predictions Once we've fit the model using our data, we can evaluate how well the model performed by using repeated stratified k-fold cross validation. For this example, we'll use 10 folds and 3 repeats:

In [4]:

```
# Define method to evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

print(np.mean(scores))
```

```
0.9800000000000001
```

We can see that the model performed a mean accuracy of **98.00\%**.

We can also use the model to predict which class a new flower belongs to, based on input values :

In [5]:

```
#define new observation
new = [5, 3, 1, .4]

#predict which class the new observation belongs to
model.predict([new])
```

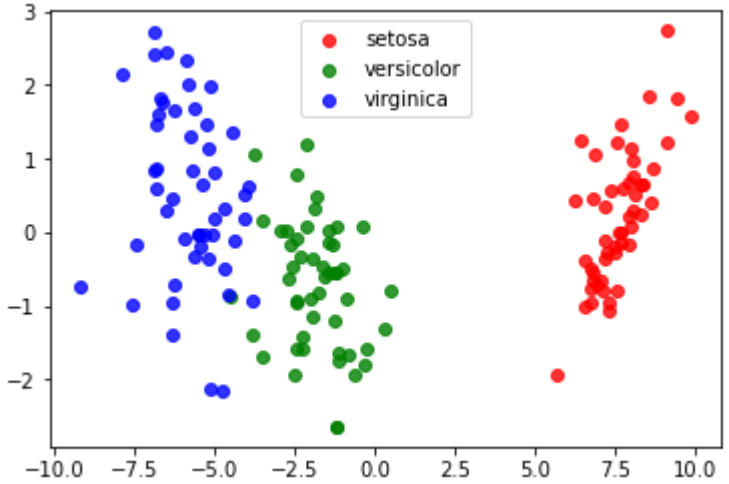
Out[5]:
```
array(['setosa'], dtype='<U10')
```

We can see that the model predicts this new observation to belong to the species called *setosa*

Step 5. Visualize the Results

Lastly, we can create an LDA plot to view the linear discriminants of the model and visualize how well it separated the three different species in our dataset:

In [6]:
```
#define data to plot
X = iris.data
y = iris.target

model = LinearDiscriminantAnalysis()

data_plot = model.fit(X, y).transform(X)

target_names = iris.target_names

#create LDA plot
plt.figure()
colors = ['red', 'green', 'blue']
lw = 2
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
    plt.scatter(data_plot[y == i, 0], data_plot[y == i, 1], alpha=.8, color=color,
                label=target_name)

#add legend to plot
plt.legend(loc = 'best', shadow = False, scatterpoints = 1)

#display LDA plot
plt.show()
```



Quadratic Discriminant Analysis (QDA)

When we have a set of predictor variables and we'd like to classify a *response variable* into one of *two classes*, we typically use *logistic regression*.

However, when *a response variable* has *more than two possible classes* then we typically use *linear discriminant analysis*, often referred to as LDA.

LDA assumes that
(1) observations from each class are normally distributed
(2) observations from each class share the same covariance matrix. Using these assumptions, LDA then finds the following values:
μ_k : The mean of all training observations from the kth class.
σ^2 : The weighted average of the sample variances for each of the k classes. π_k : The proportion of the training observations that belong to the k^{th} class.
LDA then plugs these numbers into the following formula and assigns each observation *X = x* to the class for which the formula produces the largest value :

$$D_k(x) = x \times \frac{\mu_k}{\sigma^2} + \frac{\mu_k^2}{2\sigma^2} + log(\pi_k)$$

LDA has linear in its name because the value produced by the function above comes from a result of linear functions of x.

An *extension* of linear discriminant analysis *(LDA)* is **Quadratic Discriminant Analysis**, often referred to as QDA.

This method is similar to LDA and also assumes that the observations from each class are normally distributed, but it does not assume that each class shares the same covariance matrix. Instead, QDA assumes that each class has its own covariance matrix.

That is, it assumes that an observation from the kth class is of the form $X \sim N(\mu_k\ ,\ \Sigma_k)$

Using this *assumption, QDA* then finds the following values:
μ_k : The mean of all training observations from the k^{th} class.
Σ_k : The covariance matrix of the k^{th} class.
π_k : The proportion of the training observations that belong to the k^{th} class.

QDA then plugs these numbers into the following formula and assigns each observation *X = x* to the class for which the formula produces the largest value :

$$D_k(x) = -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}log|\Sigma_k| + log(\pi_k)$$

Note that QDA has *quadratic* in its name because the value produced by the function above comes from a result of quadratic functions of x.

LDA vs. QDA : When to Use One vs. the Other
The main difference between LDA and QDA is that LDA assumes each class shares a covariance matrix, which makes it a much less flexible classifier than QDA.

This inherently means it has low variance – that is, it will perform similarly on different training datasets. The drawback is that if the assumption that the K classes have the same covariance is

untrue, then LDA can suffer from *high bias*.

QDA is generally preferred to LDA in the following situations:
(1) The training set is large.
(2) It's unlikely that the *K* classes share a common covariance matrix.

When these conditions hold, QDA tends to perform better since it is more flexible and can provide a better fit to the data.

How to Prepare Data for QDA :
Make sure your data meets the following requirements before applying a QDA model to it :

1. **The response variable is categorical :** QDA models are designed to be used for classification problems, i.e. when the response variable can be placed into classes or categories.
2. **The observations in each class follow a normal distribution :** First, check that each the distribution of values in each class is roughly normally distributed. If this is not the case, you may choose to first transform the data to make the distribution more normal.
3. **Account for extreme outlier :**. Be sure to check for extreme outliers in the dataset before applying LDA. Typically you can check for outliers visually by simply using *boxplots* or *scatterplots*.

QDA Using Python

Quadratic Discriminant Analysis

Step-1 : Load Necessary Libraries

In [7]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

Step 2. Load the Data

For this example, we'll use the iris dataset from the sklearn library. The following code shows how to load this dataset and convert it to a pandas DataFrame to make it easy to work with:

In [8]:
```
# load iris dataset
iris = datasets.load_iris()

# convert dataset to pandas DataFrame
df = pd.DataFrame(data = np.c_[iris['data'], iris['target']],
                  columns = iris['feature_names'] + ['target'])

df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

```
df.columns = ['s_length', 's_width', 'p_length', 'p_width', 'target', 'species']

# view first six rows of DataFrame
df.head()
```

Out[8]:

	s_length	s_width	p_length	p_width	target	species
0	5.1	3.5	1.4	0.2	0.0	setosa
1	4.9	3.0	1.4	0.2	0.0	setosa
2	4.7	3.2	1.3	0.2	0.0	setosa
3	4.6	3.1	1.5	0.2	0.0	setosa
4	5.0	3.6	1.4	0.2	0.0	setosa

Step 3. Fit the QDA Model

Next, we'll fit the QDA model to our data using the QuadraticDiscriminantAnalsyis function from sklearn:

In [ ]:

```
# define predictor and response variables
X = df[['s_length', 's_width', 'p_length', 'p_width']]
y = df['species']

# Fit the QDA model
model = QuadraticDiscriminantAnalysis()
model.fit(X, y)
```

Step 4. Use the Model to Make Predictions

Once we've fit the model using our data, we can evaluate how well the model performed by using repeated stratified k-fold cross validation.
For this example, we'll use 10 folds and 3 repeats:

In [9]:

```
# Define method to evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print(np.mean(scores))
```

```
0.9800000000000001
```

We can see that the model performed a mean accuracy of **98.00\%**.

We can also use the model to predict which class a new flower belongs to, based on input values:

In [10]:

```
# define new observation
new = [5, 3, 1, .4]

# predict which class the new observation belongs to
model.predict([new])
```

Out[10]: array([0])

We can see that the model predicts this new observation to belong to the species called setosa.

How to Assess Model Fit

Overfitting

In the field of machine learning, we often build models so that we can make accurate predictions about some phenomenon.

For example, suppose we want to build a *regression model* that uses the predictor variable hours spent studying to predict the response variable ACT score for students in high school.

To build this model, we'll collect data about hours spent studying and the corresponding ACT Score for hundreds of students in a certain school district.

Then we'll use this data to train a model that can make predictions about the score a given student will receive based on their total hours studied.

To assess how useful the model is, we can measure how well the model predictions match the observed data. One of the most commonly used metrics for doing so is the **Mean Squared Error** (MSE), which is calculated as :

$$MSE = \frac{1}{n}\sum(y_i - f(x_i))^2$$

where, n : Total number of observations
y_i : The response value of the i^{th} observation
$f(x_i)$: The predicted response value of the i^{th} observation

The closer the model predictions are to the observations, the smaller the MSE will be.

However, one of the biggest mistakes made in machine learning is optimizing models to reduce **training MSE** – i.e. how closely the model predictions match up with the data that we used to train the model.

When a model focuses too much on reducing training MSE, it often works too hard to find patterns in the training data that are just caused by random chance. Then when the model is applied to unseen data, it performs poorly.

This phenomenon is known as overfitting. It occurs when we "fit" a model too closely to the training data and we thus end up building a model that isn't useful for making predictions about new data.

Leave-One-Out Cross-Validation (LOOCV)

To evaluate the performance of a model on a dataset, we need to measure how well the predictions made by the model match the observed data.

The most common way to measure this is by using the mean squared error (MSE), which is calculated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

where, n : Total number of observations
y_i : The response value of the ith observation
$f(x_i)$: The predicted response value of the ith observation

The closer the model predictions are to the observations, the smaller the MSE will be.

In practice, we use the following process to calculate the MSE of a given model :

- Split a dataset into a training set and a testing set.
- Build the model using only data from the training set.
- Use the model to make predictions on the testing set and measure the MSE – this is know as the **test MSE**.

The test MSE gives us an idea of how well a model will perform on data it hasn't previously seen, i.e. data that wasn't used to "train" the model.

However, the drawback of using only one testing set is that the test MSE can vary greatly depending on which observations were used in the training and testing sets.

It's possible that if we use a different set of observations for the training set and the testing set that our test MSE could turn out to be much larger or smaller.

One way to avoid this problem is to fit a model several times using a different training and testing set each time, then calculating the test MSE to be the average of all of the test MSE's.

This general method is known as cross-validation and a specific form of it is known as **leave-one-out cross-validation**.

LOOCV
Leave-one-out cross-validation uses the following approach to evaluate a model:

- Split a dataset into a training set and a testing set, using all but one observation as part of the training set.
  Note that we only leave one observation "out" from the training set. This is where the method gets the name "leave-one-out" cross-validation.
- Build the model using only data from the training set.
- Use the model to predict the response value of the one observation left out of the model and calculate the MSE.
- Repeat the process n times.

Lastly, we repeat this process n times (where n is the total number of observations in the dataset), leaving out a different observation from the training set each time.

We then calculate the test MSE to be the average of all of the test MSE's:

$$Test\ MSE = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

where:, n : The total number of observations in the dataset
MSE_i : The test MSE during the ith time of fitting the model.

Pros & Cons of LOOCV
Leave-one-out cross-validation offers the following pros:

- It provides a much less biased measure of test MSE compared to using a single test set because we repeatedly fit a model to a dataset that contains n-1 observations.
- It tends not to overestimate the test MSE compared to using a single test set.

However, leave-one-out cross-validation comes with the following cons:

- It can be a time-consuming process to use when n is large.
- It can also be time-consuming if a model is particularly complex and takes a long time to fit to a dataset.
- It can be computationally expensive.

Fortunately, modern computing has become so efficient in most fields that LOOCV is a much more reasonable method to use compared to many years ago.

Note that **LOOCV** can be *used* in both **regression** and **classification settings** as well. For regression problems, it calculates the test MSE to be the mean squared difference between predictions and observations while in classification problems it calculates the test MSE to be the percentage of observations correctly classified during the n repeated model fittings.

LOOCV Using Python

Leave-One-Out-Cross-Validation

One commonly used method for doing this is known as leave-one-out cross-validation (LOOCV), which uses the following approach:

1. Split a dataset into a training set and a testing set, using all but one observation as part of the training set.

2. Build a model using only data from the training set.

3. Use the model to predict the response value of the one observation left out of the model and calculate the mean squared error (MSE).

4. Repeat this process n times. Calculate the test MSE to be the average of all of the test MSE's.

Step-1. Load Necessary Libraries

In [11]:
```
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
```

Step 2. Create the Data

Next, we'll create a pandas DataFrame that contains two predictor variables, x_1 and x_2 and a single response variable y.

In [12]:
```
df = pd.DataFrame({'y': [6, 8, 12, 14, 14, 15, 17, 22, 24, 23],
                   'x1': [2, 5, 4, 3, 4, 6, 7, 5, 8, 9],
                   'x2': [14, 12, 12, 13, 7, 8, 7, 4, 6, 5]})
```

Step 3. Perform Leave-One-Out Cross-Validation

Next, we'll then fit a multiple linear regression model to the dataset and perform LOOCV to evaluate the model performance.

In [13]:
```
# define predictor and response variables
X = df[['x1', 'x2']]
y = df['y']

# define cross-validation method to use
cv = LeaveOneOut()

# build multiple linear regression model
model = LinearRegression()

# use LOOCV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',
                         cv=cv, n_jobs=-1)

# view mean absolute error
mean(absolute(scores))
```

Out[13]:
```
3.1461548083469744
```

From the output we can see that the mean absolute error (MAE) was **3.146**. That is, the average absolute error between the model prediction and the actual observed data is 3.146.

In general, the lower the MAE, the more closely a model is able to predict the actual observations.

Another commonly used metric to evaluate model performance is the root mean squared error (RMSE). The following code shows how to calculate this metric using LOOCV:

In [14]:
```
# define predictor and response variables
```

```
X = df[['x1', 'x2']]
y = df['y']

# define cross-validation method to use
cv = LeaveOneOut()

# build multiple linear regression model
model = LinearRegression()

# use LOOCV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',
                         cv=cv, n_jobs=-1)

# view RMSE
sqrt(mean(absolute(scores)))
```

Out[14]: 3.6194564763855688

From the output we can see that the root mean squared error (RMSE) was **3.619**. The lower the RMSE, the more closely a model is able to predict the actual observations.

In practice we typically fit several different models and compare the RMSE or MAE of each model to decide which model produces the lowest test error rates and is therefore the best model to use.

K-Fold Validation

To evaluate the performance of some model on a dataset, we need to measure how well the predictions made by the model match the observed data.

The most common way to measure this is by using the mean squared error (MSE), which is calculated as :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

where, n : Total number of observations
y_i : The response value of the ith observation
$f(x_i)$: The predicted response value of the ith observation
The closer the model predictions are to the observations, the smaller the MSE will be.

In practice, we use the following process to calculate the MSE of a given model :
1. Split a dataset into a training set and a testing set.
2. Build the model using only data from the training set.
3. Use the model to make predictions on the testing set and measure the test MSE.

The test MSE gives us an idea of how well a model will perform on data it hasn't previously seen. However, the drawback of using only one testing set is that the test MSE can vary greatly depending on which observations were used in the training and testing sets.

One way to avoid this problem is to fit a model several times using a different training and testing set each time, then calculating the test MSE to be the average of all of the test MSE's.

This general method is known as cross-validation and a specific form of it is known as k-fold cross-validation.

K-Fold Cross-Validation
K-fold cross-validation uses the following approach to evaluate a model :
Step 1: Randomly divide a dataset into k groups, or "folds", of roughly equal size.
Step 2: Choose one of the folds to be the holdout set. Fit the model on the remaining k-1 folds. Calculate the test MSE on the observations in the fold that was held out.
Step 3: Repeat this process k times, using a different set each time as the holdout set.
Step 4: Calculate the overall test MSE to be the average of the k test MSE's.

$$Test\ MSE = \frac{1}{k}\sum_{i=1}^{n} MSE_i$$

where, $k4 : Numberoffolds < br >MSE_i : Test MSE on the ith iteration

How to Choose K*
In general, the more folds we use in k-fold cross-validation the lower the bias of the test MSE but the higher the variance. Conversely, the fewer folds we use the higher the bias but the lower the variance. This is a classic example of the bias-variance tradeoff** in machine learning.

In practice, we typically choose to use between 5 and 10 folds. As noted in An Introduction to Statistical Learning, this number of folds has been shown to offer an optimal balance between bias and variance and thus provide reliable estimates of test MSE:

- To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation.
- Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

Advantages of K-Fold Cross-Validation
When we split a dataset into just one training set and one testing set, the test MSE calculated on the observations in the testing set can vary greatly depending on which observations were used in the training and testing sets.

By using k-fold cross-validation, we're able to use calculate the test MSE using several different variations of training and testing sets. This makes it much more likely for us to obtain an unbiased estimate of the test MSE.

K-fold cross-validation also offers a computational advantage over leave-one-out cross-validation **(LOOCV)** because it only has to fit a model k times as opposed to n times.

For models that take a long time to fit, k-fold cross-validation can compute the test MSE much quicker than LOOCV and in many cases the test MSE calculated by each approach will be quite similar if you use a sufficient number of folds.

Extensions of K-Fold Cross-Validation

There are several extensions of k-fold cross-validation, including:

1. **Repeated K-fold Cross-Validation :** This is where k-fold cross-validation is simply repeated *n* times. Each time the training and testing sets are shuffled, so this further reduces the bias in the estimate of test MSE although this takes longer to perform than ordinary k-fold cross-validation.

2. **Leave-One-Out Cross-Validation :** This is a special case of k-fold cross-validation in which *k*=*n*. You can read more about this method here.

3. **Stratified K-Fold Cross-Validation :** This is a version of k-fold cross-validation in which the dataset is rearranged in such a way that each fold is representative of the whole. As noted by Kohavi, this method tends to offer a better tradeoff between bias and variance compared to ordinary k-fold cross-validation.

4. **Nested Cross-Validation :** This is where k-fold cross validation is performed within each fold of cross-validation. This is often used to perform hyperparameter tuning during model evaluation.

K-Fold Cross Validation Using Python

k-fold cross-validation , which uses the following approach :

1. Randomly divide a dataset into k groups, or “folds”, of roughly equal size.

2. Choose one of the folds to be the holdout set. Fit the model on the remaining k-1 folds. Calculate the test MSE on the observations in the fold that was held out.

3. Repeat this process k times, using a different set each time as the holdout set.

4. Calculate the overall test MSE to be the average of the k test MSE’s.

Step 1. Load Necessary Libraries First, we’ll load the necessary functions and libraries for this example:

In [15]:
```
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
```

Step 2. Create the Data

Next, we'll create a pandas DataFrame that contains two predictor variables, x_1 and x_2 and a single response variable y.

In [16]:
```
df = pd.DataFrame({'y': [6, 8, 12, 14, 14, 15, 17, 22, 24, 23],
                   'x1': [2, 5, 4, 3, 4, 6, 7, 5, 8, 9],
                   'x2': [14, 12, 12, 13, 7, 8, 7, 4, 6, 5]})
```

Step 3. Perform K-Fold Cross-Validation

Next, we'll then fit a multiple linear regression model to the dataset and perform LOOCV to evaluate the model performance.

In [17]:
```
# define predictor and response variables
X = df[['x1', 'x2']]
y = df['y']

# define cross-validation method to use
cv = KFold(n_splits=10, random_state=1, shuffle=True)

# build multiple linear regression model
model = LinearRegression()

# use k-fold CV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',
                         cv=cv, n_jobs=-1)

# view mean absolute error
mean(absolute(scores))
```

Out[17]:
```
3.1461548083469744
```

From the output we can see that the mean absolute error (MAE) was **3.14**. That is, the average absolute error between the model prediction and the actual observed data is *3.14*.

In general, the lower the MAE, the more closely a model is able to predict the actual observations.

Another commonly used metric to evaluate model performance is the root mean squared error (RMSE).
The following code shows how to calculate this metric using LOOCV:

In [18]:
```
#define predictor and response variables
X = df[['x1', 'x2']]
y = df['y']

#define cross-validation method to use
cv = KFold(n_splits=5, random_state=1, shuffle=True)

#build multiple linear regression model
model = LinearRegression()

#use LOOCV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',
                         cv=cv, n_jobs=-1)
```

```
#view RMSE
sqrt(mean(absolute(scores)))
```

Out[18]: 4.284373111711817

From the output we can see that the root mean squared error (RMSE) was **4.284**. The lower the RMSE, the more closely a model is able to predict the actual observations.

In practice we typically fit several different models and compare the RMSE or MAE of each model to decide which model produces the lowest test error rates and is therefore the best model to use.

Also note that in this example we chose to use k=5 folds, but we can choose however many folds we'd like. In practice, we typically choose between 5 and 10 folds because this turns out to be the optimal number of folds that produce reliable test error rates.

Model Selection

Best Subset Selection

In the field of machine learning, we're often interested in building models using a set of predictor variables and a *response variable*. Our goal is to build a model that can effectively use the predictor variables to predict the value of the response variable.

Given a set of p total predictor variables, there are many models that we could potentially build. One method that we can use to pick the *best model* is known as **best subset selection** and it works as follows :

1. Let M_0 denote the null model, which contains no predictor variables.
2. For $k = 1, 2, \ldots, p$

- fit all $\binom{p}{k}$ models that contain exactly k predictors.
- Pick the best among these $\binom{p}{k}$ models and call it M_k. Define "best" as the model with the highest R^2 or equivalently the lowest RSS.

1. Select a single best model from among $M_0 \ldots M_p$ using **cross-validation prediction error**, C_p, BIC, AIC or $adjusted\ R^2 \Rightarrow R^2_{adj.}$.

Note that for a set of p predictor variables, there are 2^p possible models.

Example of Best Subset Selection :
Suppose we have a dataset with $p = 3$ predictor variables and one response variable y. To perform best subset selection with this dataset, we would fit the following $2^p = 23 = 8$ models :

- A model with no predictors
- A model with predictor x_1
- A model with predictor x_2

- A model with predictor x_3
- A model with predictors x_1, x_2
- A model with predictors x_1, x_3
- A model with predictors x_2, x_3
- A model with predictors x_1, x_2, x_3

Next, we'd choose the model with the highest R^2 among each set of models with k predictors. For example, we might end up choosing:

- A model with predictor x_2
- A model with predictors x_1, x_2
- A model with predictors x_1, x_2, x_3

Next, we'd perform *cross-validation* and choose the best model to be the one that results in the lowest **prediction error**, C_p, BIC, AIC or **adjusted** $R^2 = R^2_{adj.}$.

For example, we might end up choosing the following model as the "best" model because it produced the lowest cross-validated prediction error:

- A model with predictors x_1, x_2

Criteria for Choosing the "Best" Model

The last step of best subset selection involves choosing the model with the **lowest prediction error, lowest Cp, lowest BIC, lowest AIC, or highest adjusted** $R^2 = R^2_{adj.}$.

Here are the formulas used to calculate each of these metrics :
$C_p : \frac{(RSS+2d\hat{\sigma})}{n}$
$AIC : \frac{(RSS+2d\hat{\sigma^2})}{n\hat{\sigma^2}}$
$BIC : \frac{(RSS+log(n)d\hat{\sigma^2})}{n}$
$R^2_{adj.} : 1 - \frac{\frac{RSS}{(n-d-1)}}{\frac{TSS}{(n-1)}}$

where, d : The number of predictors
n : Total observations
$\hat{\sigma}$: Estimate of the variance of the error associate with each response measurement in a regression model
RSS : Residual sum of squares of the regression model
TSS : Total sum of squares of the regression model

Pros & Cons of Best Subset Selection

Best subset selection offers the following pros :

- It's a straightforward approach to understand and interpret.

- It allows us to identify the best possible model since we consider all combinations of predictor variables.

However, this method comes with the following cons :

- It can be computationally intense. For a set of p predictor variables, there are 2^p possible models. For example, with 10 predictor variables there are $2^{10} = 1,000$ possible models to consider.
- Because it considers such a large number of models, it could potentially find a model that performs well on training data but not on future data. This could result in *overfitting*.

Conclusion

While best subset selection is straightforward to implement and understand, it can be unfeasible if you're working with a dataset that has a large number of predictors and it could potentially lead to overfitting.

An alternative to this method is known as **stepwise selection**, which is more computationally efficient.

Stepwise Selection

In the field of machine learning, our goal is to build a model that can effectively use a set of predictor variables to predict the value of some *response variable*.

Given a set of p total predictor variables, there are many models that we could potentially build. One method that we can use to pick the best model is known as **best subset selection**, which attempts to choose the best model from all possible models that could be built with the set of predictors.

Unfortunately this method suffers from two drawbacks :

- It can be computationally intense. For a set of p predictor variables, there are 2^p possible models. For example, with 10 predictor variables there are $2^{10} = 1,000$ possible models to consider.
- Because it considers such a large number of models, it could potentially find a model that performs well on training data but not on future data. This could result in *overfitting*.

An alternative to best subset selection is known as **stepwise selection**, which compares a much more restricted set of models.

There are two types of stepwise selection methods: forward stepwise selection and backward stepwise selection.

Forward Stepwise Selection

Forward stepwise selection works as follows :
1. Let M_0 denote the null model, which contains no predictor variables.

2. For $k = 0, 2, \ldots, p-1$:

- Fit all $p-k$ models that augment the predictors in Mk with one additional predictor variable.
- Pick the best among these $p-k$ models and call it M_{k+1}. Define "best" as the model with the highest R^2 or equivalently the lowest RSS.
  3. Select a single best model from among $M_0 \ldots M_p$ using *cross-validation prediction error, Cp, BIC, AIC, or adjusted* $R^2 = R^2_{adj}$.

Backward Stepwise Selection

Backward stepwise selection works as follows :
1. Let M_p denote the full model, which contains all p predictor variables.
2. For $k = p, p-1, \ldots, 1$:

- Fit all k models that contain all but one of the predictors in M_k, for a total of $k-1$ predictor variables.
- Pick the best among these k models and call it M_{k-1}. Define "best" as the model with the highest R^2 or equivalently the lowest RSS.
  3. Select a single best model from among $M_0 \ldots M_p$ using *cross-validation prediction error, Cp, BIC, AIC, or adjusted* R^2.

Criteria for Choosing the "Best" Model

The last step of both forward and backward stepwise selection involves choosing the model with the *lowest prediction error, lowest Cp, lowest BIC, lowest AIC, or highest adjusted* R^2.

Here are the formulas used to calculate each of these metrics :
$C_p : \frac{(RSS+2d\hat{\sigma})}{n}$
$AIC : \frac{(RSS+2d\hat{\sigma^2})}{n\hat{\sigma^2}}$
$BIC : \frac{(RSS+log(n)d\hat{\sigma^2})}{n}$
$R^2_{adj.} : 1 - \frac{\frac{RSS}{(n-d-1)}}{\frac{TSS}{(n-1)}}$

where, d : The number of predictors
n : Total observations
$\hat{\sigma}$: Estimate of the variance of the error associate with each response measurement in a regression model
RSS : Residual sum of squares of the regression model
TSS : Total sum of squares of the regression model

Pros & Cons of Stepwise Selection

Stepwise selection offers the following **benefit** :
It is more computationally efficient than best subset selection. Given p predictor variables, best subset selection must fit 2^p models.

Conversely, stepwise selection only has to fit $\frac{1+p(p+1)}{2}$ models. For $p = 10$ predictor variables, best subset selection must fit $1,000$ models while stepwise selection only has to fit 56 models.

However, stepwise selection has the following potential **drawback** :
It is not guaranteed to find the best possible model out of all 2^p potential models.
For example, suppose we have a dataset with $p = 3$ predictors. The best possible one-predictor model may contain x_1 and the best possible two-predictor model may instead contain x_1 and x_2.

In this case, forward stepwise selection will fail to select the best possible two-predictor model because M_1 will contain x_1, so M_2 must also contain x_1 along with some other variable.

Regularization

Ridge Regression

In ordinary multiple linear regression, we use a set of p predictor variables and a response variable to fit a model of the form :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

where, Y : The response variable
X_j : The j^{th} predictor variable
β_j : The average effect on Y of a one unit.
ϵ : The error term

The values for $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ are chosen using the **least square method** , which minimizes the sum of squared residuals *(RSS)*

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where, $\sum$: A greek symbol that means *sum*.
y_i : The actual response value for the i^{th} observation.
$\hat{y}_i$: The predicted response value based on the multiple linear regression model.

However, when the predictor variables are highly correlated then **multicollinearity** can become a problem. This can cause the coefficient estimates of the model to be unreliable and have high variance.

One way to get around this issue without completely removing some predictor variables from the model is to use a method known as ridge regression, which instead seeks to minimize the following :

$$RSS + \lambda \Sigma \beta_j^2$$

where, j ranges from $1 \; to \; p$ and $\lambda \geq 0$.

This second term in the equation is known as a **shrinkage penalty**.

When $\lambda = 0$, this penalty term has no effect and ridge regression produces the same coefficient estimates as least squares. However, as λ approaches infinity, the shrinkage penalty becomes more influential and the ridge regression coefficient estimates approach zero.

In general, the predictor variables that are least influential in the model will shrink towards zero the fastest.

Why Use Ridge Regression ?
The advantage of ridge regression compared to least squares regression lies in the *bias-variance tradeoff*.

Recall that mean squared error (MSE) is a metric we can use to measure the accuracy of a given model and it is calculated as :

$$MSE = Var(\hat{f}\,(x_0)) + [Bias(\hat{f}\,(x_0))]^2 + Var(\epsilon)$$

$$MSE = Variance + Bias^2 + Irreducible\ error$$

The basic idea of ridge regression is to introduce a little bias so that the variance can be substantially reduced, which leads to a lower overall MSE.

*Ridge regression test MSE reduction

This means the model fit by ridge regression will produce smaller test errors than the model fit by least squares regression.

**Steps to Perform Ridge Regression in Practice*
The following steps can be used to perform ridge regression :

Step 1: Calculate the correlation matrix and VIF values for the predictor variables.

First, we should produce a correlation matrix and calculate the **VIF** (variance inflation factor) values for each predictor variable.

If we detect high correlation between predictor variables and high VIF values (some texts define a "high" VIF value as 5 while others use 10) then ridge regression is likely appropriate to use.

However, if there is no multicollinearity present in the data then there may be no need to perform ridge regression in the first place. Instead, we can perform ordinary least squares regression.

Step 2: Standardize each predictor variable.

Before performing ridge regression, we should scale the data such that each predictor variable has a mean of 0 and a standard deviation of 1. This ensures that no single predictor variable is overly influential when performing ridge regression.

Step 3: Fit the ridge regression model and choose a value for λ.

There is no exact formula we can use to determine which value to use for λ. In practice, there are two common ways that we choose λ:

(1) **Create a Ridge trace plot:** This is a plot that visualizes the values of the coefficient estimates as λ increases towards infinity. Typically we choose λ as the value where most of the coefficient estimates begin to stabilize.

(2) **Calculate the test MSE for each value of λ**

Another way to choose λ is to simply calculate the test MSE of each model with different values of λ and choose λ to be the value that produces the lowest test MSE.

Pros & Cons of Ridge Regression

The biggest **benefit** of ridge regression is its ability to produce a lower test mean squared error (MSE) compared to least squares regression when multicollinearity is present.

However, the biggest **drawback** of ridge regression is its inability to perform variable selection since it includes all predictor variables in the final model. Since some predictors will get shrunken very close to zero, this can make it hard to interpret the results of the model.

In practice, ridge regression has the potential to produce a model that can make better predictions compared to a least squares model but it is often harder to interpret the results of the model.

Depending on whether model interpretation or prediction accuracy is more important to you, you may choose to use ordinary least squares or ridge regression in different scenarios.

Ridge Regression Using Python

Step 1. Import Necessary Packages

In [19]:
```
from numpy import arange
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import RepeatedKFold
```

Step 2: Load the Data For this example, we'll use a dataset called mtcars, which contains information about 33 different cars. We'll use hp as the response variable and the following variables as the predictors :

- mpg
- wt
- drat
- qsec

In [20]:
```
# define URL where data is located
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"
```

```
# read in data
data_full = pd.read_csv(url)

# select subset of data
data = data_full[["mpg", "wt", "drat", "qsec", "hp"]]

# view first six rows of data
data[0:6]
```

Out[20]:

	mpg	wt	drat	qsec	hp
0	21.0	2.620	3.90	16.46	110
1	21.0	2.875	3.90	17.02	110
2	22.8	2.320	3.85	18.61	93
3	21.4	3.215	3.08	19.44	110
4	18.7	3.440	3.15	17.02	175
5	18.1	3.460	2.76	20.22	105

Step 3. Fit the Ridge Regression Model

Next, we'll use the RidgeCV() function from sklearn to fit the ridge regression model and we'll use the RepeatedKFold() function to perform k-fold cross-validation to find the optimal alpha value to use for the penalty term.

Note: The term "alpha" α is used instead of "lambda" λ in Python.

For this example we'll choose k = 10 folds and repeat the cross-validation process 3 times.

Also note that RidgeCV() only tests alpha values .1, 1, and 10 by default. However, we can define our own alpha range from 0 to 1 by increments of 0.01 :

In [21]:

```
# define predictor and response variables
X = data[["mpg", "wt", "drat", "qsec"]]
y = data["hp"]

# define cross-validation method to evaluate model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

# define model
model = RidgeCV(alphas=arange(0, 1, 0.01), cv=cv, scoring='neg_mean_absolute_error')

# fit model
model.fit(X, y)

# display lambda that produced the lowest test MSE
print(model.alpha_)
```

```
0.99
```

The lambda value that minimizes the test MSE turns out to be **0.99**.

Step 4. Use the Model to Make Predictions

Lastly, we can use the final ridge regression model to make predictions on new observations. For example, the following code shows how to define a new car with the following attributes :

- mpg : 24
- wt : 2.5
- drat : 3.5
- qsec : 18.5

The fitted ridge regression model to predict the value for hp of this new observation:

In [22]:
```
# define new observation
new = [24, 2.5, 3.5, 18.5]

# predict hp value using ridge regression model
model.predict([new])
```

Out[22]:
```
array([104.16398018])
```

Based on the input values, the model predicts this car to have an hp value of **104.164**.

Lasso Regression

In ordinary multiple linear regression, we use a set of p predictor variables and a response variable to fit a model of the form :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

where, Y : The response Vairable
X_j : The j^{th} predictor variable
β_j : The average effect on Y of a one unit increase in X_j, holding all other predictors fixed
ϵ : The error term

The values for $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ are chosen using the **least square method**, which minimizes the sum of squared residuals (RSS) :

$$RSS = \sum_{i=1}^{p} (y_i - \hat{y}_i)^2$$

where, Σ : A greek symbol that means sum
y_i : The actual response value for the ith observation
$\hat{y}_i$: The predicted response value based on the multiple linear regression model

However, when the predictor variables are highly correlated then *multicollinearity* can become a problem. This can cause the coefficient estimates of the model to be unreliable and have high variance. That is, when the model is applied to a new set of data it hasn't seen before, it's likely to perform poorly.

One way to get around this issue is to use a method known as **lasso regression**, which instead seeks to minimize the following :

$$RSS + \lambda\Sigma|\beta_j|$$

where j ranges from $1 \, to \, p$ and $\lambda \geq 0$.

This second term in the equation is known as a *shrinkage penalty*.

When $\lambda = 0$, this penalty term has no effect and lasso regression produces the same coefficient estimates as least squares.

However, as λ approaches infinity the shrinkage penalty becomes more influential and the predictor variables that aren't importable in the model get shrunk towards zero and some even get dropped from the model.

Why Use Lasso Regression ?

The advantage of lasso regression compared to least squares regression lies in the *bias-variance tradeoff*.

Recall that mean squared error (MSE) is a metric we can use to measure the accuracy of a given model and it is calculated as :

$$MSE = Var(\hat{f}\,(x_0)) + [Bias(\hat{f}\,(x_0))]^2 + Var(\epsilon)$$

$$MSE = Variance + Bias^2 + Irreducible \ error$$

The basic idea of lasso regression is to introduce a little bias so that the variance can be substantially reduced, which leads to a lower overall MSE.

Lasso Regression vs. Ridge Regression

Lasso regression and ridge regression are both known as *regularization methods* because they both attempt to minimize the sum of squared residuals *(RSS)* along with some penalty term.

In other words, they constrain or regularize the coefficient estimates of the model.

However, the penalty terms they use are a bit different:

Lasso regression attempts to minimize $RSS + \lambda\Sigma|\beta_j|$

Ridge regression attempts to minimize $RSS + \lambda\Sigma\beta_j^2$

When we use ridge regression, the coefficients of each predictor are shrunken towards zero but none of them can go completely to zero.

Conversely, when we use lasso regression it's possible that some of the coefficients could go completely to zero when λ gets sufficiently large.

In technical terms, lasso regression is capable of producing "sparse" models – models that only include a subset of the predictor variables.

This begs the question: Is ridge regression or lasso regression better ?
The answer: It depends!

In cases where only a small number of predictor variables are significant, lasso regression tends to perform better because it's able to shrink insignificant variables completely to zero and remove them from the model.

However, when many predictor variables are significant in the model and their coefficients are roughly equal then ridge regression tends to perform better because it keeps all of the predictors in the model.

To determine which model is better at making predictions, we perform *k-fold cross-validation*. Whichever model produces the lowest test mean squared error (MSE) is the preferred model to use.

Steps to Perform Lasso Regression in Practice

The following steps can be used to perform lasso regression:

Step 1: Calculate the correlation matrix and VIF values for the predictor variables.
First, we should produce a *correlation matrix* and calculate the *VIF* (variance inflation factor) values for each predictor variable.
If we detect high correlation between predictor variables and high VIF values (some texts define a "high" VIF value as 5 while others use 10) then lasso regression is likely appropriate to use.

However, if there is no multicollinearity present in the data then there may be no need to perform lasso regression in the first place. Instead, we can perform ordinary least squares regression.

Step 2: Fit the lasso regression model and choose a value for λ.
Once we determine that lasso regression is appropriate to use, we can fit the model (using popular programming languages like R or Python) using the optimal value for λ.
To determine the optimal value for λ, we can fit several models using different values for λ and choose λ to be the value that produces the lowest test MSE.

Step 3: Compare lasso regression to ridge regression and ordinary least squares regression.
Lastly, we can compare our lasso regression model to a ridge regression model and least squares regression model to determine which model produces the lowest test MSE by using k-fold cross-validation.
Depending on the relationship between the predictor variables and the response variable, it's entirely possible for one of these three models to outperform the others in different scenarios.

Lasso Regression Using Python

Step 1. Load the Data
We'll use *hp* as the response variable and the variables as the predictors : *mpg, wt, drat, qsec*

In [23]:
```
import pandas as pd
from numpy import arange
from sklearn.linear_model import LassoCV
from sklearn.model_selection import RepeatedKFold
```

In [24]:
```
# define URL where data is located
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"

# read in data
data_full = pd.read_csv(url)

# select subset of data
data = data_full[["mpg", "wt", "drat", "qsec", "hp"]]

# view first six rows of data
data[0:6]
```

Out[24]:

	mpg	wt	drat	qsec	hp
0	21.0	2.620	3.90	16.46	110
1	21.0	2.875	3.90	17.02	110
2	22.8	2.320	3.85	18.61	93
3	21.4	3.215	3.08	19.44	110
4	18.7	3.440	3.15	17.02	175
5	18.1	3.460	2.76	20.22	105

Step 3. Fit the Lasso Regression Model

Next, we'll use the LassoCV() function from sklearn to fit the lasso regression model and we'll use the RepeatedKFold() function to perform k-fold cross-validation to find the optimal alpha value to use for the penalty term.

Note: The term "alpha" is used instead of "lambda" in Python.

For this example we'll choose k = 10 folds and repeat the cross-validation process 3 times.

Also note that LassoCV() only tests alpha values 0.1, 1, and 10 by default. However, we can define our own alpha range from 0 to 1 by increments of 0.01 :

In [28]:
```
#define predictor and response variables
X = data[["mpg", "wt", "drat", "qsec"]]

y = data["hp"]

#define cross-validation method to evaluate model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

#define model
model = LassoCV(alphas=arange(0, 1, 0.01), cv=cv, n_jobs=-1)

#fit model
```

```
model.fit(X, y)

#display lambda that produced the lowest test MSE
print(model.alpha_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: ConvergenceWarning: Objective did not converge. You might want to increase the numbe
r of iterations. Duality gap: 12262.264733628224, tolerance: 11.509525
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
```

```
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
0.99
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
```

```
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:5
26: UserWarning: Coordinate descent with alpha=0 may lead to unexpected results and is d
iscouraged.
  model = cd_fast.enet_coordinate_descent_gram(
```

The lambda value that minimizes the test MSE turns out to be **0.99**.

Step 4. Use the Model to Make Predictions

Lastly, we can use the final lasso regression model to make predictions on new observations. For example, the following code shows how to define a new car with the following attributes :

- mpg: 24
- wt: 2.5
- drat: 3.5
- qsec: 18.5

The fitted lasso regression model to predict the value for hp of this new observation:

In [29]:
```
#define new observation
new = [24, 2.5, 3.5, 18.5]

#predict hp value using lasso regression model
model.predict([new])
```

Out[29]:
```
array([105.63442071])
```

Based on the input values, the model predicts this car to have an hp value of **105.634**.

Dimension Reduction

Principal Components Regression(PCR)

One of the most common problems that you'll encounter when building models is *multicollinearity*. This occurs when two or more predictor variables in a dataset are highly correlated.

When this occurs, a given model may be able to fit a training dataset well but it will likely perform poorly on a new dataset it has never seen because it *overfit* the training set.

One way to avoid overfitting is to use some type of **subset selection** method like:
1. Best subset selection
2. Stepwise selection

These methods attempt to remove irrelevant predictors from the model so that only the most important predictors that are capable of predicting the variation in the response variable are left in the final model.

Another way to avoid overfitting is to use some type of **regularization** method like :
1. Ridge Regression
2. Lasso Regression

These methods attempt to constrain or regularize the coefficients of a model to reduce the variance and thus produce models that are able to generalize well to new data.

An entirely different approach to dealing with multicollinearity is known as dimension reduction.

A common method of dimension reduction is know as principal components regression, which works as follows :

1. Suppose a given dataset contains p predictors : $X_1, X_2, \ldots, X_p$
2. Calculate $Z_1, \ldots, Z_M$ to be the M linear combinations of the original p predictors.

- $Z_m = \sum \phi_{jm} X_j$ for some constants $\phi_{1m}, \phi_{2m}, \ldots, \phi_{pm} \quad m = 1, \ldots, M.$
- Z_1 is the linear combination of the predictors that captures the most variance possible.
- Z_2 is the next linear combination of the predictors that captures the most variance while being *orthogonal* (i.e. uncorrelated) to Z_1.
- Z_3 is then the next linear combination of the predictors that captures the most variance while being orthogonal to Z_2.
- And so on.

1. Use the method of least squares to fit a linear regression model using the first M principal components Z_1,..., Z_M$ as predictors.

The phrase **dimension reduction** comes from the fact that this method only has to estimate $M + 1$ coefficients instead of $p + 1$ coefficients, where $M < p$.

In other words, the dimension of the problem has been reduced from $p + 1 \; to \; M + 1$.

In many cases where multicollinearity is present in a dataset, principal components regression is able to produce a model that can generalize to new data better than conventional *multiple linear regression*.

Steps to Perform Principal Components Regression
In practice, the following steps are used to perform principal components regression:

Pros & Cons of Principal Components Regression

Principal Components Regression (PCR) offers the following *pros* :

- PCR tends to perform well when the first few principal components are able to capture most of the variation in the predictors along with the relationship with the response variable.
- PCR can perform well even when the predictor variables are highly correlated because it produces principal components that are orthogonal (i.e. uncorrelated) to each other.
- PCR doesn't require you to choose which predictor variables to remove from the model since each principal component uses a linear combination of all of the predictor variables.
- PCR can be used when there are more predictor variables than observations, unlike multiple linear regression.

However, PCR comes with one *con* :

- PCR does not consider the response variable when deciding which principal components to keep or drop. Instead, it only considers the magnitude of the variance among the predictor variables captured by the principal components. It's possible that in some cases the principal components with the largest variances aren't actually able to predict the response variable well.

In practice, we fit many different types of models (PCR, Ridge, Lasso, Multiple Linear Regression, etc.) and use k-fold cross-validation to identify the model that produces the lowest test MSE on new data.

In cases where multicollinearity is present in the original dataset (which is often), PCR tends to perform better than ordinary least squares regression. However, it's a good idea to fit several different models so that you can identify the one that generalizes best to unseen data.

PCR Using Python

Principal Components Regression

Step 1. Load Necessary Packages

In [30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Step 2. Load the Data

For this example, we'll use a dataset called mtcars, which contains information about 33 different cars. We'll use hp as the response variable and the following variables as the predictors :

- mpg
- disp
- drat

- wt
- qsec

In [31]:
```
#define URL where data is located
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"

#read in data
data_full = pd.read_csv(url)

#select subset of data
data = data_full[["mpg", "disp", "drat", "wt", "qsec", "hp"]]

#view first six rows of data
data[0:6]
```

Out[31]:

	mpg	disp	drat	wt	qsec	hp
0	21.0	160.0	3.90	2.620	16.46	110
1	21.0	160.0	3.90	2.875	17.02	110
2	22.8	108.0	3.85	2.320	18.61	93
3	21.4	258.0	3.08	3.215	19.44	110
4	18.7	360.0	3.15	3.440	17.02	175
5	18.1	225.0	2.76	3.460	20.22	105

Step 3. Fit the PCR Model

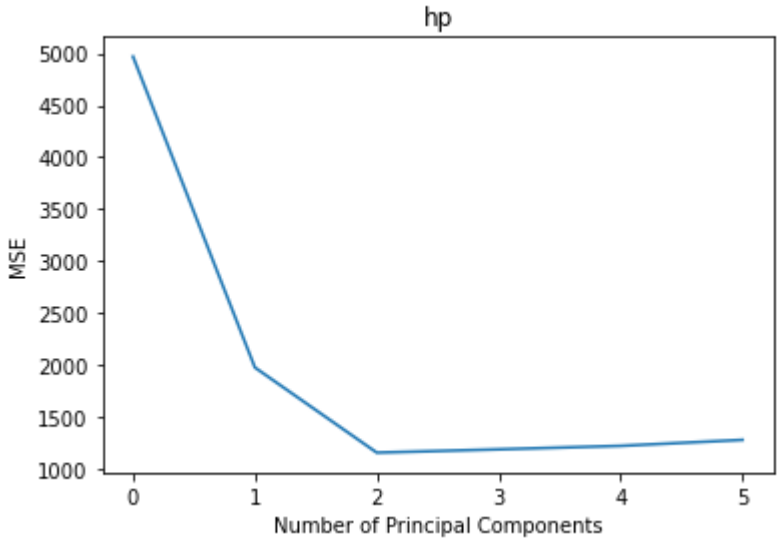
The following code shows how to fit the PCR model to this data. Note the following:

- **pca.fit_transform(scale(X))** : This tells Python that each of the predictor variables should be scaled to have a mean of 0 and a standard deviation of 1. This ensures that no predictor variable is overly influential in the model if it happens to be measured in different units.
- **cv = RepeatedKFold()** : This tells Python to use k-fold cross-validation to evaluate the performance of the model. For this example we choose k = 10 folds, repeated 3 times.

In [32]:
```
#define predictor and response variables
X = data[["mpg", "disp", "drat", "wt", "qsec"]]
y = data[["hp"]]

#scale predictor variables
pca = PCA()
X_reduced = pca.fit_transform(scale(X))

#define cross validation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

regr = LinearRegression()
mse = []

# Calculate MSE with only the intercept
score = -1*model_selection.cross_val_score(regr,
```

```
            np.ones((len(X_reduced),1)), y, cv=cv,
            scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using cross-validation, adding one component at a time
for i in np.arange(1, 6):
    score = -1*model_selection.cross_val_score(regr,
               X_reduced[:,:i], y, cv=cv, scoring='neg_mean_squared_error').mean()
    mse.append(score)

# Plot cross-validation results
plt.plot(mse)
plt.xlabel('Number of Principal Components')
plt.ylabel('MSE')
plt.title('hp')
```

Out[32]:
```
Text(0.5, 1.0, 'hp')
```



The plot displays the number of principal components along the x-axis and the test MSE (mean squared error) along the y-axis.

From the plot we can see that the test MSE decreases by adding in two principal components, yet it begins to increase as we add more than two principal components.

Thus, the optimal model includes just the first two principal components.

We can also use the following code to calculate the percentage of variance in the response variable explained by adding in each principal component to the model :

In [33]:
```
np.cumsum(np.round(pca.explained_variance_ratio_ ,
                   decimals=4)*100)
```

Out[33]:
```
array([69.83, 89.35, 95.88, 98.95, 99.99])
```

We can see the following:

- By using just the first principal component, we can explain **69.83\%** of the variation in the response variable.

- By adding in the second principal component, we can explain **89.35\%** of the variation in the response variable.

Note that we'll always be able to explain more variance by using more principal components, but we can see that adding in more than two principal components doesn't actually increase the percentage of explained variance by much.

Step 4. Use the Final Model to Make Predictions

We can use the final PCR model with two principal components to make predictions on new observations.

The following code shows how to split the original dataset into a training and testing set and use the PCR model with two principal components to make predictions on the testing set.

In [34]:
```
#split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)

#scale the training and testing data
X_reduced_train = pca.fit_transform(scale(X_train))
X_reduced_test = pca.transform(scale(X_test))[:,:1]

#train PCR model on training data
regr = LinearRegression()
regr.fit(X_reduced_train[:,:1], y_train)

#calculate RMSE
pred = regr.predict(X_reduced_test)
np.sqrt(mean_squared_error(y_test, pred))
```

Out[34]:
```
40.209642107611515
```

We can see that the test RMSE turns out to be **40.2096**. This is the average deviation between the predicted value for hp and the observed value for hp for the observations in the testing set.

Partial Least Squares (PLS)

One of the most common problems that you'll encounter in machine learning is *multicollinearity*. This occurs when two or more predictor variables in a dataset are highly correlated.

When this occurs, a model may be able to fit a training dataset well but it may perform poorly on a new dataset it has never seen because it *overfits* the training set.

One way to get around the problem of multicollinearity is to use *principal components regression (PCR)*, which calculates M linear combinations (known as "principal components") of the original p predictor variables and then uses the method of least squares to fit a linear regression model using the principal components as predictors.

The drawback of principal components regression (PCR) is that it does not consider the response variable when calculating the principal components.

Instead, it only considers the magnitude of the variance among the predictor variables captured by the principal components. Because of this, it's possible that in some cases the principal components with the largest variances aren't actually able to predict the response variable well.

A technique that is related to PCR is known as partial least squares. Similar to PCR, partial least squares calculates M linear combinations (known as "PLS components") of the original p predictor variables and uses the method of least squares to fit a linear regression model using the PLS components as predictors.

But unlike PCR, partial least squares attempts to find linear combinations that explain the variation in both the response variable and the predictor variables.

Steps to Perform Partial Least Squares

In practice, the following steps are used to perform partial least squares.

1. Standardize the data such that all of the predictor variables and the response variable have a mean of 0 and a standard deviation of 1. This ensures that each variable is measured on the same scale.

2. Calculate $Z_1, \ldots, Z_M$ to be the M linear combinations of the original p predictors.

- $Z_m = \sum \phi_{jm} X_j$ for some constants $\phi_{1m}, \phi_{2m}, \ldots, \phi_{pm} \quad m = 1, 2, \ldots, M$
- To calculate Z_1, set ϕ_{j1} equal to the coefficient from the simple linear regression of Y onto X_j is the linear combination of the predictors that captures the most variance possible.
- To calculate Z_2, regression each variable on Z_1 and take the residuals. Then calculate Z_2 using this orthogonalized data in exactly the same manner that Z_1 was calculated.
- Repeat this process M times to obtain the M PLS components.

1. Use the method of least squares to fit a linear regression model using the PLS components $Z_1, \ldots, Z_M$ as predictors.

2. Lastly, use *k-fold cross-validation* to find the optimal number of PLS components to keep in the model. The "optimal" number of PLS components to keep is typically the number that produces the lowest test mean-squared error (MSE).

Conclusion

In cases where multicollinearity is present in a dataset, partial least squares tends to perform better than ordinary least squares regression. However, it's a good idea to fit several different models so that we can identify the one that generalizes best to unseen data.

In practice, we fit many different types of models **(PLS, PCR, Ridge, Lasso, Multiple Linear Regression, etc.)** to a dataset and use k-fold cross-validation to identify the model that produces the lowest test MSE on new data.

PLS Using Python

Partial Least Squares

Step 1 : Step Necessary Package

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.cross_decomposition import PLSRegression
from sklearn.metrics import mean_squared_error
```

Step 2. Load the Data

For this example, we'll use a dataset called mtcars, which contains information about 33 different cars. We'll use hp as the response variable and the following variables as the predictors :

- mpg
- disp
- drat
- wt
- qsec

The following code shows how to load and view this dataset :

In [3]:
```
# define URL where data is located
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"

# read in data
data_full = pd.read_csv(url)

# select subset of data
data = data_full[["mpg", "disp", "drat", "wt", "qsec", "hp"]]

# view first six rows of data
data[0:6]
```

Out[3]:

	mpg	disp	drat	wt	qsec	hp
0	21.0	160.0	3.90	2.620	16.46	110
1	21.0	160.0	3.90	2.875	17.02	110
2	22.8	108.0	3.85	2.320	18.61	93
3	21.4	258.0	3.08	3.215	19.44	110
4	18.7	360.0	3.15	3.440	17.02	175
5	18.1	225.0	2.76	3.460	20.22	105

Step 3. Fit the Partial Least Squares Model

The following code shows how to fit the PLS model to this data.

Note that **cv = RepeatedKFold()** tells Python to use k-fold cross-validation to evaluate the performance of the model. For this example we choose k = 10 folds, repeated 3 times.

In [4]:
```
# define predictor and response variables
X = data[["mpg", "disp", "drat", "wt", "qsec"]]
y = data[["hp"]]

# define cross-validation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

mse = []
n = len(X)

# Calculate MSE with only the intercept
score = -1*model_selection.cross_val_score(PLSRegression(n_components=1),
           np.ones((n,1)), y, cv=cv, scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using cross-validation, adding one component at a time
for i in np.arange(1, 6):
    pls = PLSRegression(n_components=i)
    score = -1*model_selection.cross_val_score(pls, scale(X), y, cv=cv,
               scoring='neg_mean_squared_error').mean()
    mse.append(score)

# plot test MSE vs. number of components
plt.plot(mse)
plt.xlabel('Number of PLS Components')
plt.ylabel('MSE')
plt.title('hp')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
```

```
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

```
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
```

```
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
```

```
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
```

```
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
```

```
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

```
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
```

```
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
```

```
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
```

```
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
```

```
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
```

```
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
```

```
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
```

```
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
```

```
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
```

```
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
```

```
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
```

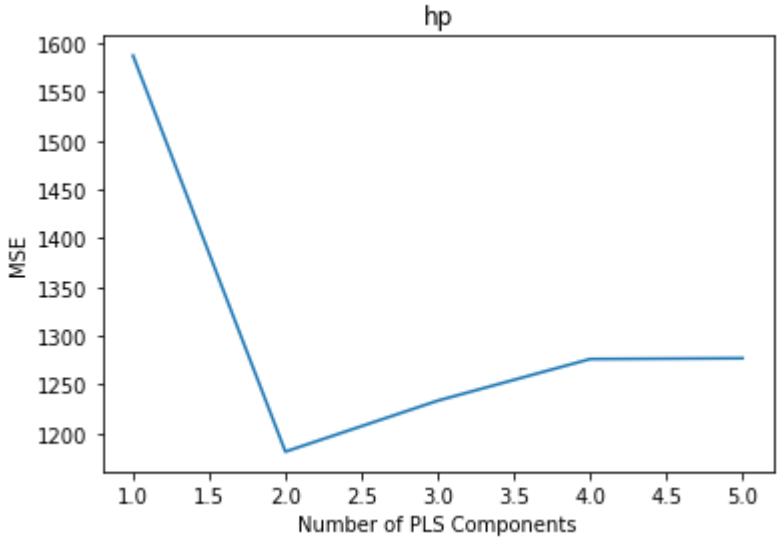
```
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
```

```
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
 will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
```

```
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:83: Runti
meWarning: invalid value encountered in true_divide
  y_weights = np.dot(Y.T, x_score) / np.dot(x_score.T, x_score)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:291: Runt
imeWarning: invalid value encountered in true_divide
  x_loadings = np.dot(x_scores, Xk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_decomposition\_pls.py:300: Runt
imeWarning: invalid value encountered in true_divide
  y_loadings = np.dot(x_scores, Yk) / np.dot(x_scores, x_scores)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:696: U
serWarning: Scoring failed. The score on this train-test partition for these parameters
 will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.p
y", line 687, in _score
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 87,
in __call__
    score = scorer._score(cached_call, estimator,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 24
2, in _score
    return self._sign * self._score_func(y_true, y_pred,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
335, in mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_regression.py", line
90, in _check_reg_targets
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 6
3, in inner_f
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 72
0, in check_array
    _assert_all_finite(array,
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 10
```

```
3, in _assert_all_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
Text(0.5, 1.0, 'hp')
```

Out[4]:



The plot displays the number of PLS components along the x-axis and the test MSE (mean squared error) along the y-axis.

From the plot we can see that the test MSE decreases by adding in two PLS components, yet it begins to increase as we add more than two PLS components.

Thus, the optimal model includes just the first two PLS components.

Step 4. Use the Final Model to Make Predictions

We can use the final PLS model with two PLS components to make predictions on new observations.

The following code shows how to split the original dataset into a training and testing set and use the PLS model with two PLS components to make predictions on the testing set.

In [5]:

```
# split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)

# calculate RMSE
pls = PLSRegression(n_components=2)
pls.fit(scale(X_train), y_train)

np.sqrt(mean_squared_error(y_test, pls.predict(scale(X_test))))
```

Out[5]:

```
29.909431977844196
```

We can see that the test RMSE turns out to be **29.9094**. This is the average deviation between the predicted value for hp and the observed value for hp for the observations in the testing set.

Advanced Regression Models

Polynomial Regression

When we have a dataset with *one predictor variable and one response variable*, we often use *simple linear regression* to quantify the relationship between the two variables.

However, simple linear regression *(SLR)* assumes that the relationship between the predictor and response variable is linear. Written in mathematical notation, SLR assumes that the relationship takes the form :

$$Y = \beta_0 + \beta_1 X + \epsilon$$

But in practice the relationship between the two variables can actually be nonlinear and attempting to use linear regression can result in a poorly fit model.

One way to account for a nonlinear relationship between the predictor and response variable is to use **polynomial regression**, which takes the form :

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_h X^h + \epsilon$$

In this equation, h is referred to as the degree of the polynomial.

As we increase the value for h, the model is able to fit nonlinear relationships better, but in practice we rarely choose h to be greater than 3 or 4. Beyond this point, the model becomes too flexible and *overfits the data*.

Technical Notes

- Although polynomial regression can fit nonlinear data, it is still considered to be a form of linear regression because it is linear in the coefficients $\beta_1, \beta_2, \cdots, \beta_h$
- Polynomial regression can be used for multiple predictor variables as well but this creates interaction terms in the model, which can make the model extremely complex if more than a few predictor variables are used.

When to Use Polynomial Regression

We use polynomial regression when the relationship between a predictor and response variable is nonlinear.

There are *three* common ways to detect a nonlinear relationship:

If the residuals of the plot are roughly evenly distributed around zero with no clear pattern, then simple linear regression is likely sufficient.

However, if the residuals display a nonlinear pattern in the plot then this is a sign that the relationship between the predictor and the response is likely nonlinear.

If we fit a simple linear regression model to a dataset and the R^2 value of the model is quite low, this could be an indication that the relationship between the predictor and response variable is more complex than just a simple linear relationship.

This could be a sign that you may need to try polynomial regression instead.

How to Choose the Degree of the Polynomial

A polynomial regression model takes the following form:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_h X^h + \epsilon$$

In this equation, h is the degree of the polynomial.

But how do we choose a value for h?

In practice, we fit several different models with different values of h and perform *k-fold cross-validation* to determine which model produces the lowest test mean squared error (MSE).

For example, we may fit the following models to a given dataset :

- $Y = \beta_0 + \beta_1 X$
- $Y = \beta_0 + \beta_1 X + \beta_2 X^2$
- $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$
- $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4$

We can then use k-fold cross-validation to calculate the test MSE of each model, which will tell us how well each model performs on data it hasn't seen before.

The Bias-Variance Tradeoff of Polynomial Regression

There exists a *bias-variance tradeoff* when using polynomial regression. As we increase the degree of the polynomial, the bias decreases (as the model becomes more flexible) but the variance increases.

As with all machine learning models, we must find an optimal tradeoff between bias and variance.

In most cases it helps to increase the degree of the polynomial to an extent, but beyond a certain value the model begins to fit the noise of the data and the test MSE begins to decrease.

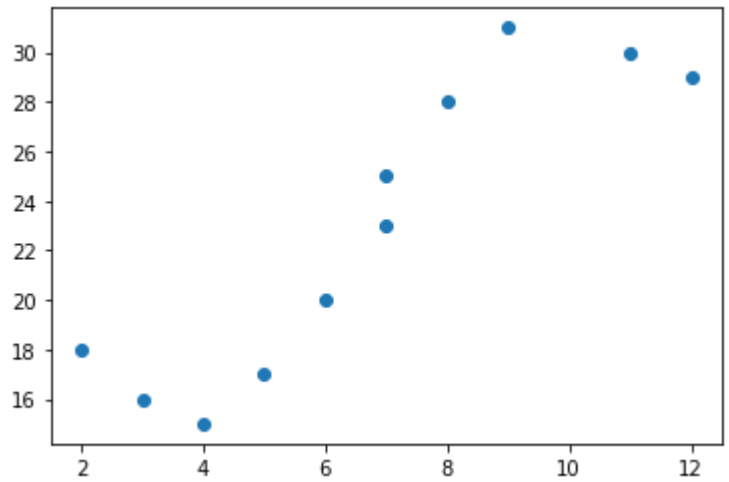
To ensure that we fit a model that is flexible but not too flexible, we use k-fold cross-validation to find the model that produces the lowest test MSE.

Polynomial Regression Using Python

In [6]:

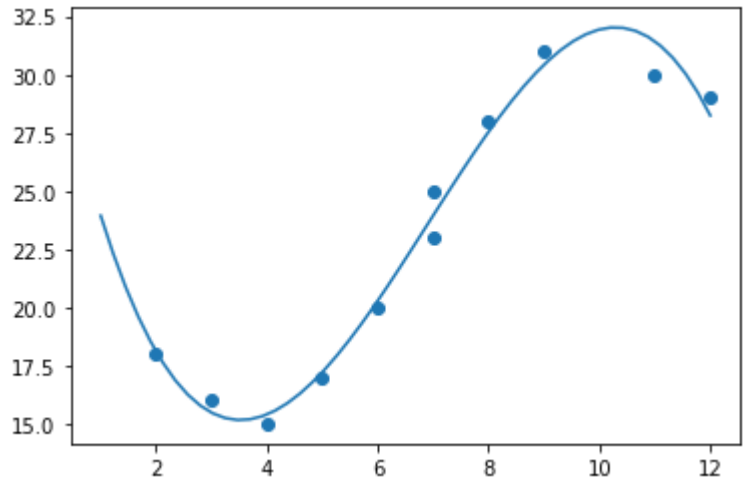
```
x = [2, 3, 4, 5, 6, 7, 7, 8, 9, 11, 12]
y = [18, 16, 15, 17, 20, 23, 25, 28, 31, 30, 29]

import matplotlib.pyplot as plt

#create scatterplot
plt.scatter(x, y)
```

Out[6]:
```
<matplotlib.collections.PathCollection at 0x240027563d0>
```



Thus, it wouldn't make sense to fit a linear regression model to this data. Instead, we can attempt to fit a polynomial regression model with a degree of 3 using the *numpy.polyfit()* function :

In [7]:
```
import numpy as np

#polynomial fit with degree = 3
model = np.poly1d(np.polyfit(x, y, 3))

#add fitted polynomial line to scatterplot
polyline = np.linspace(1, 12, 50)
plt.scatter(x, y)
plt.plot(polyline, model(polyline))
plt.show()
```



We can obtain the fitted polynomial regression equation by printing the model coefficients :

In [8]:
```
print(model)
```

```
         3         2
-0.1089 x + 2.256 x - 11.84 x + 33.63
```

The fitted ploynomial regression equation is :

$$y = -0.109x^3 + 2.256x^2 - 11.839x + 33.626$$

This equation can be used to find the expected value for the response variable based on a given value for the explanatory variable. For example, suppose x = 4. The expected value for the response variable, y, would be :

$y = -0.109(4)^3 + 2.256(4)^2 - 11.839(4) + 33.626 =$ **15.39** .

We can also write a short function to obtain the R-squared of the model, which is the proportion of the variance in the response variable that can be explained by the predictor variables.

In [10]:
```
# define function to calculate r-squared
def polyfit(x, y, degree):
    results = {}
    coeffs = np.polyfit(x, y, degree)
    p = np.poly1d(coeffs)
    #calculate r-squared
    yhat = p(x)
    ybar = np.sum(y)/len(y)
    ssreg = np.sum((yhat-ybar)**2)
    sstot = np.sum((y - ybar)**2)
    results['r_squared'] = ssreg / sstot

    return results

# find r-squared of polynomial model with degree = 3
polyfit(x, y, 3)
```

Out[10]:
```
{'r_squared': 0.9841113454244934}
```

In this example, the R-squared of the model is **0.9841**. This means that **98.41\%** of the variation in the response variable can be explained by the predictor variables.

Multivariate Adaptive Regression Splines (MARS)

When the relationship between a set of predictor variables and a *response variable* is linear, we can often use *linear regression*, which assumes that the relationship between a given predictor variable and a response variable takes the form :

$$Y = \beta_0 + \beta_1 X + \epsilon$$

But in practice the relationship between the variables can actually be nonlinear and attempting to use linear regression can result in a poorly fit model.

One way to account for a nonlinear relationship between the predictor and response variable is to use *polynomial regression*, which takes the form :

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_h X^h + \epsilon$$

In this equation, h is referred to as the “degree” of the polynomial. As we increase the value for h, the model becomes more flexible and is able to fit nonlinear data.

However, polynomial regression has a couple drawbacks:

1. Polynomial regression can easily *overfit* a dataset if the degree, h, is chosen to be too large. In practice, h is rarely larger than 3 or 4 because beyond this point it simply fits the noise of a training set and does not generalize well to unseen data.

2. Polynomial regression imposes a global function on the entire dataset, which is not always accurate.

An alternative to polynomial regression is multivariate adaptive regression splines.

The Basic Idea

Multivariate adaptive regression splines work as follows:

1. Divide a dataset into k pieces
First, we divide a dataset into *k* different pieces. The points where we divide the dataset are known as *knots*.

We identify the knots by assessing each point for each predictor as a potential knot and creating a linear regression model using the candidate features. The point that is able to reduce the most error in the model is deemed to be the knot.

Once we've identified the first knot, we then repeat the process to find additional knots. You can find as many knots as you think is reasonable to start.

2. Fit a regression function to each piece to form a hinge function
Once we've chosen the knots and fit a regression model to each piece of the dataset, we're left with something known as a *hinge function*, denoted as $h(x-a)$, where *a* is the *cutpoint value(s)*.

For example, the hinge function for a model with one knot may be as follows :

- $y = \beta_0 + \beta_1(4.3 - x)\ ;\ if x < 4.3$
- $y = \beta_0 + \beta_1(x - 4.3)\ ;\ if x > 4.$

In this case, it was determined that choosing 4.3 to be the cutpoint value was able to reduce the error the most out of all possible cutpoints values. We then fit a different regression model to the values less than *4.3* compared to values greater than *4.3*.

A hinge function with two knots may be as follows:

- $y = \beta_0 + \beta_1(4.3 - x)\ ;\ if x < 4.3$
- y = \beta_0 + \beta_1(x-4.3)\,\ ;\ if x > 4.3 \,\& \,\ x < 6.7
- $y = \beta_0 + \beta_1(6.7 - x)\ ;\ if x > 6.7$

In this case, it was determined that choosing 4.3 and 6.7 as the cutpoint values was able to reduce the error the most out of all possible cutpoint values. We then fit one regression model to the values less than *4.3*, another regression model to values between *4.3* and *6.7*, and another regression model to the values greater than *4.3*.

3. Choose k based on k-fold cross-validation

Lastly, once we've fit several different models using a different number of knots for each model, we can perform k-fold cross-validation to identify the model that produces the lowest test mean squared error (MSE).

The model with the lowest test MSE is chosen to be the model that generalizes best to new data.

Pros & Cons

Multivariate adaptive regression splines come with the following pros and cons:

Pros :

-It can be used for both regression and classification p roblems.

- It works well on large datasets.
- It offers quick computation.
- It does not require you to standardize the predictor variables.

Cons :

- It tends to not perform as well as non-linear methods like random forests and gradient boosting machines.

MARS Using Python

Multivariate Adaptive Regression Splines

Multivariate adaptive regression splines (MARS) can be used to model nonlinear relationships between a set of *predictor variables and a response variable*.

This method works as follows:
1. Divide a dataset into *k* pieces.
2. Fit a regression model to each piece.
3. Use k-fold cross-validation to choose a value for *k*.

Step 1 Import Necessary Packages

To fit a MARS model in Python, we'll use the Earth() function from sklearn-contrib-py-earth. We'll start by installing this package:

In [16]:
```
pip install sklearn-contrib-py-earth
```

```
Collecting sklearn-contrib-py-earth
  Using cached sklearn-contrib-py-earth-0.1.0.tar.gz (1.0 MB)
Requirement already satisfied: scipy>=0.16 in c:\programdata\anaconda3\lib\site-packages
(from sklearn-contrib-py-earth) (1.7.1)
Requirement already satisfied: scikit-learn>=0.16 in c:\programdata\anaconda3\lib\site-p
ackages (from sklearn-contrib-py-earth) (0.24.2)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from s
```

```
klearn-contrib-py-earth) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
WARNING: Ignoring invalid distribution -andas (c:\programdata\anaconda3\lib\site-package
s)
WARNING: Ignoring invalid distribution -andas (c:\programdata\anaconda3\lib\site-package
s)
  ERROR: Command errored out with exit status 1:
   command: 'C:\ProgramData\Anaconda3\python.exe' -u -c 'import io, os, sys, setuptools,
tokenize; sys.argv[0] = '"'"'C:\\Users\\HP\\AppData\\Local\\Temp\\pip-install-_c253w4a
\\sklearn-contrib-py-earth_85c8d0614f8d45168f6b3c37c3ba637d\\setup.py'"'"'; __file__
='"'"'C:\\Users\\HP\\AppData\\Local\\Temp\\pip-install-_c253w4a\\sklearn-contrib-py-eart
h_85c8d0614f8d45168f6b3c37c3ba637d\\setup.py'"'"';f = getattr(tokenize, '"'"'open'"'"',
open)(__file__) if os.path.exists(__file__) else io.StringIO('"'"'from setuptools import
setup; setup()'"'"');code = f.read().replace('"'"'\r\n'"'"', '"'"'\n'"'"');f.close();exe
c(compile(code, __file__, '"'"'exec'"'"'))' bdist_wheel -d 'C:\Users\HP\AppData\Local\Te
mp\pip-wheel-ni_q_cvn'
       cwd: C:\Users\HP\AppData\Local\Temp\pip-install-_c253w4a\sklearn-contrib-py-earth
_85c8d0614f8d45168f6b3c37c3ba637d\
  Complete output (72 lines):
  C:\ProgramData\Anaconda3\lib\site-packages\setuptools\dist.py:717: UserWarning: Usage
of dash-separated 'description-file' will not be supported in future versions. Please us
e the underscore name 'description_file' instead
    warnings.warn(
  running bdist_wheel
  running build
  running build_py
  creating build
  creating build\lib.win-amd64-3.9
  creating build\lib.win-amd64-3.9\pyearth
  copying pyearth\earth.py -> build\lib.win-amd64-3.9\pyearth
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site
-packages (from scikit-learn>=0.16->sklearn-contrib-py-earth) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\programdata\anaconda3\lib\site-package
s (from scikit-learn>=0.16->sklearn-contrib-py-earth) (1.1.0)
Requirement already satisfied: numpy>=1.13.3 in c:\programdata\anaconda3\lib\site-packag
es (from scikit-learn>=0.16->sklearn-contrib-py-earth) (1.20.3)
Building wheels for collected packages: sklearn-contrib-py-earth
  Building wheel for sklearn-contrib-py-earth (setup.py): started
  Building wheel for sklearn-contrib-py-earth (setup.py): finished with status 'error'
  Running setup.py clean for sklearn-contrib-py-earth
Failed to build sklearn-contrib-py-earth
Installing collected packages: sklearn-contrib-py-earth
    Running setup.py install for sklearn-contrib-py-earth: started
    Running setup.py install for sklearn-contrib-py-earth: finished with status 'error'
  copying pyearth\export.py -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_version.py -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\__init__.py -> build\lib.win-amd64-3.9\pyearth
  creating build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\testing_utils.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_earth.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_export.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_forward.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_knot_search.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_pruning.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_qr.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_util.py -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\__init__.py -> build\lib.win-amd64-3.9\pyearth\test
  creating build\lib.win-amd64-3.9\pyearth\test\basis
  copying pyearth\test\basis\base.py -> build\lib.win-amd64-3.9\pyearth\test\basis
```

```
  copying pyearth\test\basis\test_basis.py -> build\lib.win-amd64-3.9\pyearth\test\basis
  copying pyearth\test\basis\test_constant.py -> build\lib.win-amd64-3.9\pyearth\test\ba
sis
  copying pyearth\test\basis\test_hinge.py -> build\lib.win-amd64-3.9\pyearth\test\basis
  copying pyearth\test\basis\test_linear.py -> build\lib.win-amd64-3.9\pyearth\test\basi
s
  copying pyearth\test\basis\test_missingness.py -> build\lib.win-amd64-3.9\pyearth\test
\basis
  copying pyearth\test\basis\test_smoothed_hinge.py -> build\lib.win-amd64-3.9\pyearth\t
est\basis
  copying pyearth\test\basis\__init__.py -> build\lib.win-amd64-3.9\pyearth\test\basis
  creating build\lib.win-amd64-3.9\pyearth\test\record
  copying pyearth\test\record\test_forward_pass.py -> build\lib.win-amd64-3.9\pyearth\te
st\record
  copying pyearth\test\record\test_pruning_pass.py -> build\lib.win-amd64-3.9\pyearth\te
st\record
  copying pyearth\test\record\__init__.py -> build\lib.win-amd64-3.9\pyearth\test\record
  running egg_info
  writing sklearn_contrib_py_earth.egg-info\PKG-INFO
  writing dependency_links to sklearn_contrib_py_earth.egg-info\dependency_links.txt
  writing requirements to sklearn_contrib_py_earth.egg-info\requires.txt
  writing top-level names to sklearn_contrib_py_earth.egg-info\top_level.txt
  reading manifest file 'sklearn_contrib_py_earth.egg-info\SOURCES.txt'
  reading manifest template 'MANIFEST.in'
  warning: no files found matching 'pyearth\test\pathological_data'
  adding license file 'LICENSE.txt'
  writing manifest file 'sklearn_contrib_py_earth.egg-info\SOURCES.txt'
  copying pyearth\_basis.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_basis.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_forward.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_forward.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_knot_search.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_knot_search.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_pruning.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_pruning.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_qr.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_qr.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_record.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_record.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_types.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_types.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_util.c -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\_util.pxd -> build\lib.win-amd64-3.9\pyearth
  copying pyearth\test\earth_linvars_regress.txt -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\earth_regress.txt -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\earth_regress_missing_data.txt -> build\lib.win-amd64-3.9\pyearth
\test
  copying pyearth\test\earth_regress_smooth.txt -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\forward_regress.txt -> build\lib.win-amd64-3.9\pyearth\test
  copying pyearth\test\test_data.csv -> build\lib.win-amd64-3.9\pyearth\test
  UPDATING build\lib.win-amd64-3.9\pyearth/_version.py
  set build\lib.win-amd64-3.9\pyearth/_version.py to '0.1.0'
  running build_ext
  building 'pyearth._util' extension
  error: Microsoft Visual C++ 14.0 or greater is required. Get it with "Microsoft C++ Bu
ild Tools": https://visualstudio.microsoft.com/visual-cpp-build-tools/
  ----------------------------------------
  ERROR: Failed building wheel for sklearn-contrib-py-earth
WARNING: Ignoring invalid distribution -andas (c:\programdata\anaconda3\lib\site-package
s)
```

```
    ERROR: Command errored out with exit status 1:
     command: 'C:\ProgramData\Anaconda3\python.exe' -u -c 'import io, os, sys, setuptool
s, tokenize; sys.argv[0] = '"'"'C:\\Users\\HP\\AppData\\Local\\Temp\\pip-install-_c253w4
a\\sklearn-contrib-py-earth_85c8d0614f8d45168f6b3c37c3ba637d\\setup.py'"'"'; __file__
='"'"'C:\\Users\\HP\\AppData\\Local\\Temp\\pip-install-_c253w4a\\sklearn-contrib-py-eart
h_85c8d0614f8d45168f6b3c37c3ba637d\\setup.py'"'"';f = getattr(tokenize, '"'"'open'"'"',
open)(__file__) if os.path.exists(__file__) else io.StringIO('"'"'from setuptools import
setup; setup()'"'"');code = f.read().replace('"'"'\r\n'"'"', '"'"'\n'"'"');f.close();exe
c(compile(code, __file__, '"'"'exec'"'"'))' install --record 'C:\Users\HP\AppData\Local
\Temp\pip-record-pagzvs9i\install-record.txt' --single-version-externally-managed --comp
ile --install-headers 'C:\ProgramData\Anaconda3\Include\sklearn-contrib-py-earth'
         cwd: C:\Users\HP\AppData\Local\Temp\pip-install-_c253w4a\sklearn-contrib-py-ear
th_85c8d0614f8d45168f6b3c37c3ba637d\
    Complete output (20 lines):
    C:\ProgramData\Anaconda3\lib\site-packages\setuptools\dist.py:717: UserWarning: Usag
e of dash-separated 'description-file' will not be supported in future versions. Please
use the underscore name 'description_file' instead
      warnings.warn(
    running install
    running build
    running build_py
    running egg_info
    writing sklearn_contrib_py_earth.egg-info\PKG-INFO
    writing dependency_links to sklearn_contrib_py_earth.egg-info\dependency_links.txt
    writing requirements to sklearn_contrib_py_earth.egg-info\requires.txt
    writing top-level names to sklearn_contrib_py_earth.egg-info\top_level.txt
    reading manifest file 'sklearn_contrib_py_earth.egg-info\SOURCES.txt'
    reading manifest template 'MANIFEST.in'
    warning: no files found matching 'pyearth\test\pathological_data'
    adding license file 'LICENSE.txt'
    writing manifest file 'sklearn_contrib_py_earth.egg-info\SOURCES.txt'
    UPDATING build\lib.win-amd64-3.9\pyearth/_version.py
    set build\lib.win-amd64-3.9\pyearth/_version.py to '0.1.0'
    running build_ext
    building 'pyearth._util' extension
    error: Microsoft Visual C++ 14.0 or greater is required. Get it with "Microsoft C++
Build Tools": https://visualstudio.microsoft.com/visual-cpp-build-tools/
    ----------------------------------------
ERROR: Command errored out with exit status 1: 'C:\ProgramData\Anaconda3\python.exe' -u
-c 'import io, os, sys, setuptools, tokenize; sys.argv[0] = '"'"'C:\\Users\\HP\\AppData
\\Local\\Temp\\pip-install-_c253w4a\\sklearn-contrib-py-earth_85c8d0614f8d45168f6b3c37c3
ba637d\\setup.py'"'"'; __file__='"'"'C:\\Users\\HP\\AppData\\Local\\Temp\\pip-install-_c
253w4a\\sklearn-contrib-py-earth_85c8d0614f8d45168f6b3c37c3ba637d\\setup.py'"'"';f = get
attr(tokenize, '"'"'open'"'"', open)(__file__) if os.path.exists(__file__) else io.Strin
gIO('"'"'from setuptools import setup; setup()'"'"');code = f.read().replace('"'"'\r
\n'"'"', '"'"'\n'"'"');f.close();exec(compile(code, __file__, '"'"'exec'"'"'))' install
--record 'C:\Users\HP\AppData\Local\Temp\pip-record-pagzvs9i\install-record.txt' --singl
e-version-externally-managed --compile --install-headers 'C:\ProgramData\Anaconda3\Inclu
de\sklearn-contrib-py-earth' Check the logs for full command output.
WARNING: Ignoring invalid distribution -andas (c:\programdata\anaconda3\lib\site-package
s)
WARNING: Ignoring invalid distribution -andas (c:\programdata\anaconda3\lib\site-package
s)
WARNING: Ignoring invalid distribution -andas (c:\programdata\anaconda3\lib\site-package
s)
```

In [18]:

```
import pandas as pd
from numpy import mean
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import RepeatedKFold
from sklearn.datasets import make_regression
from pyearth import Earth
```

Step 2. Create a Dataset

For this example we'll use the make_regression() function to create a fake dataset with 5,000 observations and 15 predictor variables:

In [19]:
```
#create fake regression data
X, y = make_regression(n_samples=5000, n_features=15, n_informative=10,
                       noise=0.5, random_state=5)
```

Step 3. Build & Optimize the MARS Model

Next, we'll use the Earth() function to build a MARS model and the **RepeatedKFold()** function to perform k-fold cross-validation to evaluate the model performance.

For this example we'll perform 10-fold cross-validation, repeated 3 times.

In [ ]:
```
#define the model
model = Earth()

#specify cross-validation method to use to evaluate model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

#evaluate model performance
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',
                         cv=cv, n_jobs=-1)

#print results
mean(scores)
```

From the output we can see that the mean absolute error (ignore the negative sign) for this type of model is **1.7453**.

In practice we can fit a variety of different models to a given dataset (like Ridge, Lasso, Multiple Linear Regression, Partial Least Squares, Polynomial Regression, etc.) and compare the mean absolute error among all models to determine the one that produces the lowest MAE.

Note that we could also use other metrics to measure error such as adjusted R-squared or mean squared error.

In [ ]:
```

```