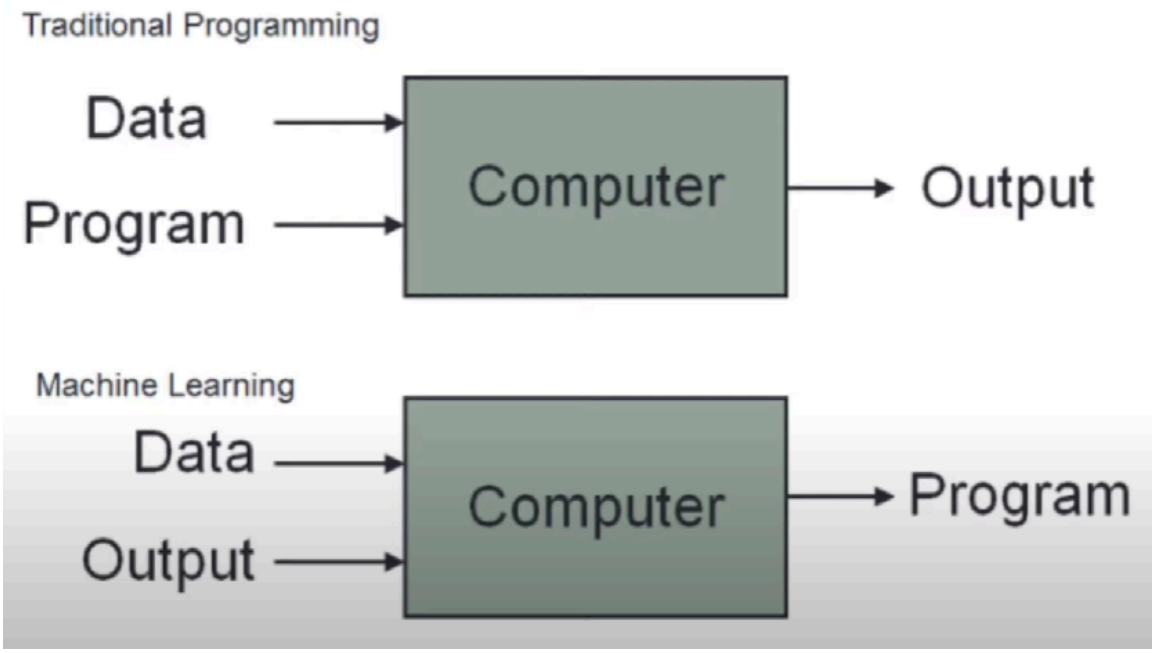
Machine Learning

- Machine learning is a field of computer science that uses statical techniques to give computer system the ability to learn with data ,without being explicity programmed
- Machine Learning is the science (and art) of programming computers so they can learn from data.
- [Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.



Types of Machine Learning Systems

- There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:
- Whether or not they are trained with human supervision (supervised, unsupervised, semisupervised, and Reinforcement Learning)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do (instance-based versus model-based learning)

-Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning,

semisupervised learning, and Reinforcement Learning.Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning.

Supervised learning

- In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels
- some of the most important supervised learning algorithms
- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks2

Unsupervised learning

- In unsupervised learning, as you might guess, the training data is unlabeled . The system tries to learn without a teacher.
- most important unsupervised learning algorithms

• Clustering

- K-Means
- DBSCAN
- Hierarchical Cluster Analysis (HCA)

• Anomaly detection and novelty detection

- One-class SVM
- Isolation Forest

• Visualization and dimensionality reduction

- Principal Component Analysis (PCA)
- Kernel PCA
- Locally-Linear Embedding (LLE)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

• Association rule learning

- Apriori
- Eclat

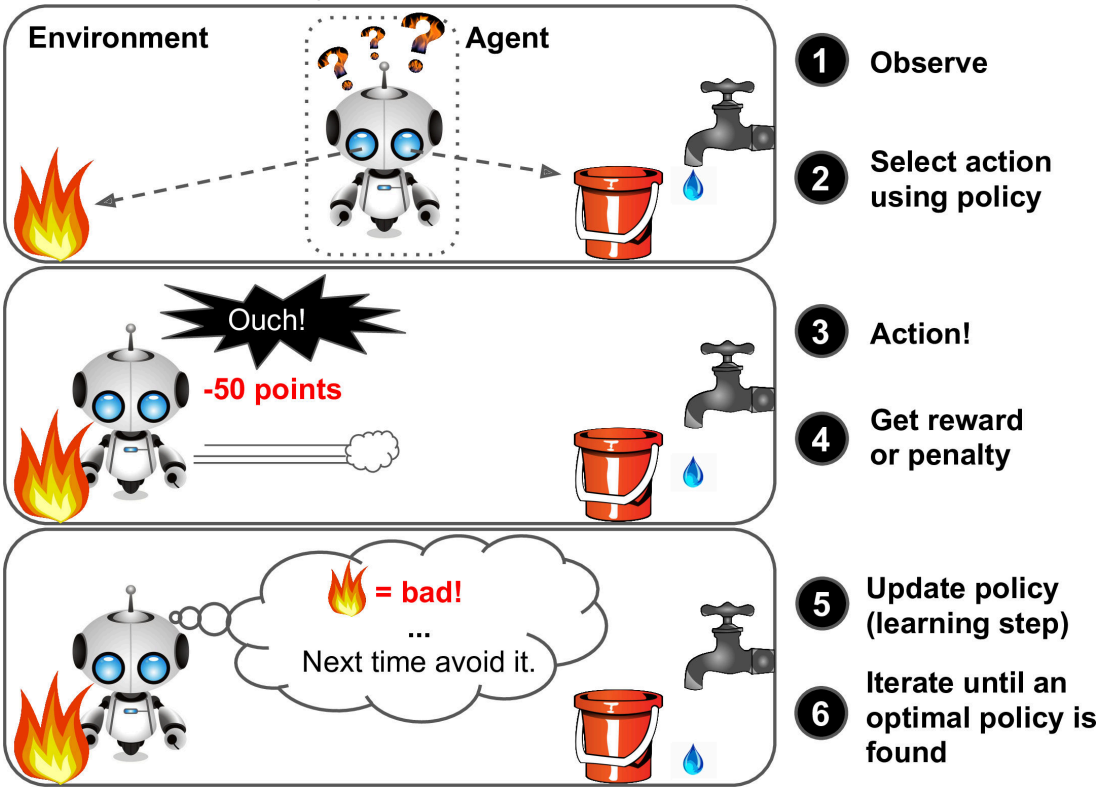
- Some neural network architectures can be unsupervised, such as autoencoders and restricted Boltzmann machines. They can also be semisupervised, such as in deep

Semisupervised learning

- Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called semisupervised learning
- Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms. For example, deep belief networks (DBNs) are based on unsupervised components called restricted Boltzmann machines (RBMs) stacked on top of one another. RBMs are trained sequentially in an unsupervised manner, and then the whole system is fine-tuned using supervised learning techniques.

Reinforcement Learning

- Reinforcement Learning is a very different beast. The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards, It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.



- For example, many robots implement Reinforcement Learning algorithms to learn how to walk. DeepMind's AlphaGo program is also a good example of Reinforcement Learning: it made the headlines in May 2017 when it beat the world champion Ke Jie at the game of Go. It learned its winning policy by analyzing millions of games, and then playing many games against itself. Note that learning was turned off during the games against the champion; AlphaGo was just applying the policy it had learned.

Batch and Online Learning

Another criterion used to classify Machine Learning systems is whether or not the system can learn incrementally from a stream of incoming data.

Batch learning

- In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning.

Online learning

- In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini-batches. Each learning step is fast and cheap, so the system can learn about new data on the fly

Instance-Based Versus Model-Based Learning

- INSTANCE BASED (also known as memory-based learning or lazy learning) involves memorizing training data in order to make predictions about future data points. This approach doesn't require any prior knowledge or assumptions about the data, which makes it easy to implement and understand. However, it can be computationally expensive since all of the training data needs to be stored in memory before making a prediction. Additionally, this approach doesn't generalize well to unseen data sets because its predictions are based on memorized examples rather than learned models.
- Model-based learning (also known as structure-based or eager learning) takes a different approach by constructing models from the training data that can generalize better than instance-based methods. This involves using algorithms like linear regression, logistic regression, random forest, etc. trees to create an underlying model from which predictions can be made for new data points. The picture below represents how the prediction about the class is decided based on boundary learned from training data rather than comparing with learned data set based on similarity measures.

Feature_Construction

A feature is an attribute of a data set that is used in a machine learning process.

•Feature engineering refers to the process of translating a data set into features such that these features are able to represent the data set more effectively and result in a better learning performance.

•It has two major elements:

•Feature Transformation

•Feature Subset Selection

In [ ]:
```
#Adding a new feature
```

In [2]:
```
import pandas as pd

#defining features
room_length = [18, 20, 10, 12, 18, 11]
room_breadth = [20, 20, 10, 11, 19, 10]
room_type = ['Big', 'Big', 'Normal', 'Normal', 'Big', 'Normal']
#creating a data frame
data = pd.DataFrame({'Length': room_length, 'Breadth': room_breadth, 'Type':
data
```

Out[2]:

	Length	Breadth	Type
0	18	20	Big
1	20	20	Big
2	10	10	Normal
3	12	11	Normal
4	18	19	Big
5	11	10	Normal

In [3]:
```
#Adding a feature called area from length and breadth

data['Area'] = data['Length'] * data['Breadth']
data
```

Out[3]:

	Length	Breadth	Type	Area
0	18	20	Big	360
1	20	20	Big	400
2	10	10	Normal	100
3	12	11	Normal	132
4	18	19	Big	342
5	11	10	Normal	110

In [6]:
```
##Encoding Nominal Variables
import pandas as pd
#Creating features
age = [18, 20, 23, 19, 18, 22]
city = ['City A', 'City B', 'City B', 'City A', 'City C', 'City B']

#Creating a data frame
data1 = pd.DataFrame({'age': age, 'city': city})
data1
```

Out[6]:

	age	city
0	18	City A
1	20	City B
2	23	City B
3	19	City A
4	18	City C
5	22	City B

In [7]:
```
#get_dummies function of pandas library can be used to dummy code categorica
df=pd.get_dummies(data=data1, drop_first=True)
df
```

Out[7]:

	age	city_City B	city_City C
0	18	0	0
1	20	1	0
2	23	1	0
3	19	0	0
4	18	0	1
5	22	1	0

In [8]:
```
##Transforming numeric (continuous) features to categorical features
#Defining features
apartment_area = [4720, 2430, 4368, 3969, 6142, 7912]
apartment_price = [2360000,1215000,2184000,1984500,3071000,3956000]

#Creating a data frame
data4 = pd.DataFrame({'Area':apartment_area, 'Price': apartment_price})
data4
```

Out[8]:

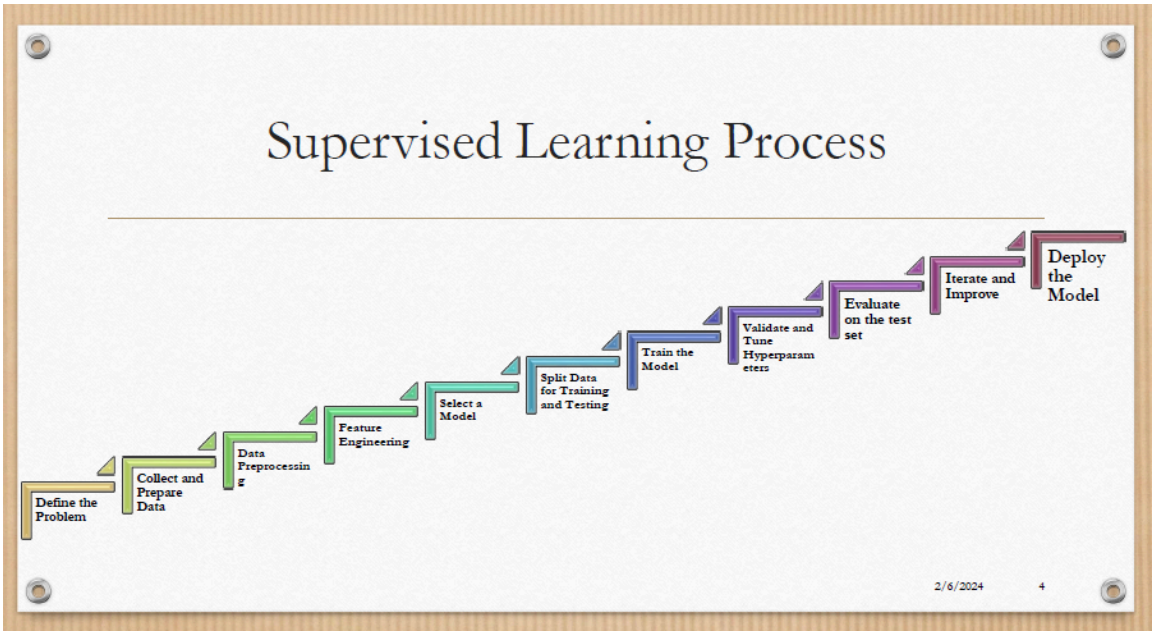
	Area	Price
0	4720	2360000
1	2430	1215000
2	4368	2184000
3	3969	1984500
4	6142	3071000
5	7912	3956000

In [9]:
```
import numpy as np
data4['Price'] = np.where(data4['Price'] > 3000000, 'High', np.where(data4['
data4
```

Out[9]:

	Area	Price
0	4720	Medium
1	2430	Low
2	4368	Medium
3	3969	Low
4	6142	High
5	7912	High

Regression



- Regression searches for relationships among variables. For example, you can observe several employees of some company and try to understand how their salaries depend on their features, such as experience, education level, role, city of employment, and so on.
- This is a regression problem where data related to each employee represents one observation. The presumption is that the experience, education, role, and city are the independent features, while the salary depends on them.
- In other words, you need to find a function that maps some features or variables to others sufficiently well.
- The dependent features are called the dependent variables, outputs, or responses. The independent features are called the independent variables, inputs, regressors, or predictors.

When Do You Need Regression?

- Typically, you need regression to answer whether and how some phenomenon influences the other or how several variables are related. For example, you can use it to determine if and to what extent experience or gender impacts salaries.
- Regression is also useful when you want to forecast a response using a new set of predictors. For example, you could try to predict electricity consumption of a household for the next hour given the outdoor temperature, time of day, and number of residents in that household.
- Regression is used in many different fields, including economics, computer science, and the social sciences. Its importance rises every day with the availability of large amounts of data and increased awareness of the practical value of data.

LINEAR REGRESSION

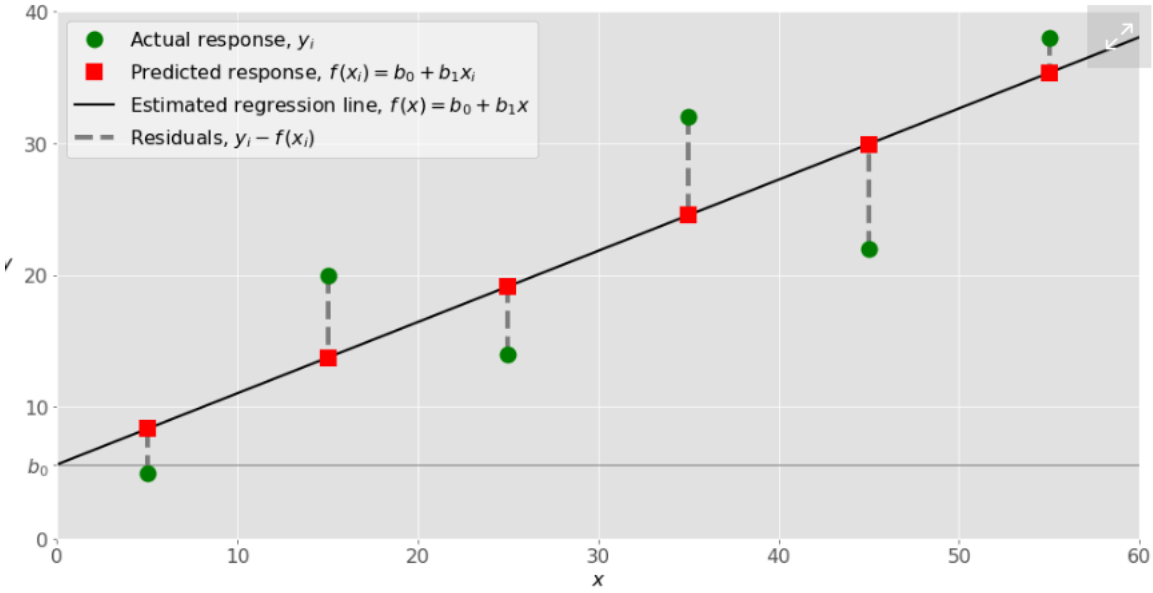
- When implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and $\mathbf{x}$: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$. This equation is the regression equation. $\beta_0, \beta_1, \ldots, \beta_r$ are the regression coefficients, and ε is the random error.
- Linear regression calculates the estimators of the regression coefficients or simply the predicted weights, denoted with $b_0, b_1, \ldots, b_r$. These estimators define the estimated regression function $f(\mathbf{x}) = b_0 + b_1 x_1 + \cdots + b_r x_r$. This function should capture the dependencies between the inputs and output sufficiently well.
- The estimated or predicted response, $f(\mathbf{x}_i)$, for each observation $i = 1, \ldots, n$, should be as close as possible to the corresponding actual response y_i. The differences $y_i - f(\mathbf{x}_i)$ for all observations $i = 1, \ldots, n$, are called the residuals. Regression is about determining the best predicted weights—that is, the weights corresponding to the smallest residuals.
- To get the best weights, you usually minimize the sum of squared residuals (SSR) for all observations $i = 1, \ldots, n$: $\text{SSR} = \Sigma_i (y_i - f(\mathbf{x}_i))^2$. This approach is called the method of ordinary least squares.

Regression Performance

- The variation of actual responses y_i, $i = 1, \ldots, n$, occurs partly due to the dependence on the predictors $\mathbf{x}_i$. However, there's also an additional inherent variance of the output.
- The coefficient of determination, denoted as R^2, tells you which amount of variation in y can be explained by the dependence on $\mathbf{x}$, using the particular regression model. A larger R^2 indicates a better fit and means that the model can better explain the variation of the output with different inputs.
- The value $R^2 = 1$ corresponds to SSR = 0. That's the perfect fit, since the values of predicted and actual responses fit completely to each other.

Simple Linear Regression

- Simple or single-variate linear regression is the simplest case of linear regression, as it has a single independent variable, $\mathbf{x} = x$



Example of simple linear regression

- The estimated regression function, represented by the black line, has the equation $f(x)$ = b_0 + b_1x. Your goal is to calculate the optimal values of the predicted weights b_0 and b_1 that minimize SSR and determine the estimated regression function.
- The value of b_0, also called the intercept, shows the point where the estimated regression line crosses the y axis. It's the value of the estimated response $f(x)$ for x = 0. The value of b_1 determines the slope of the estimated regression line

In [11]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [12]:
```
df=pd.read_csv("book1.csv")
```

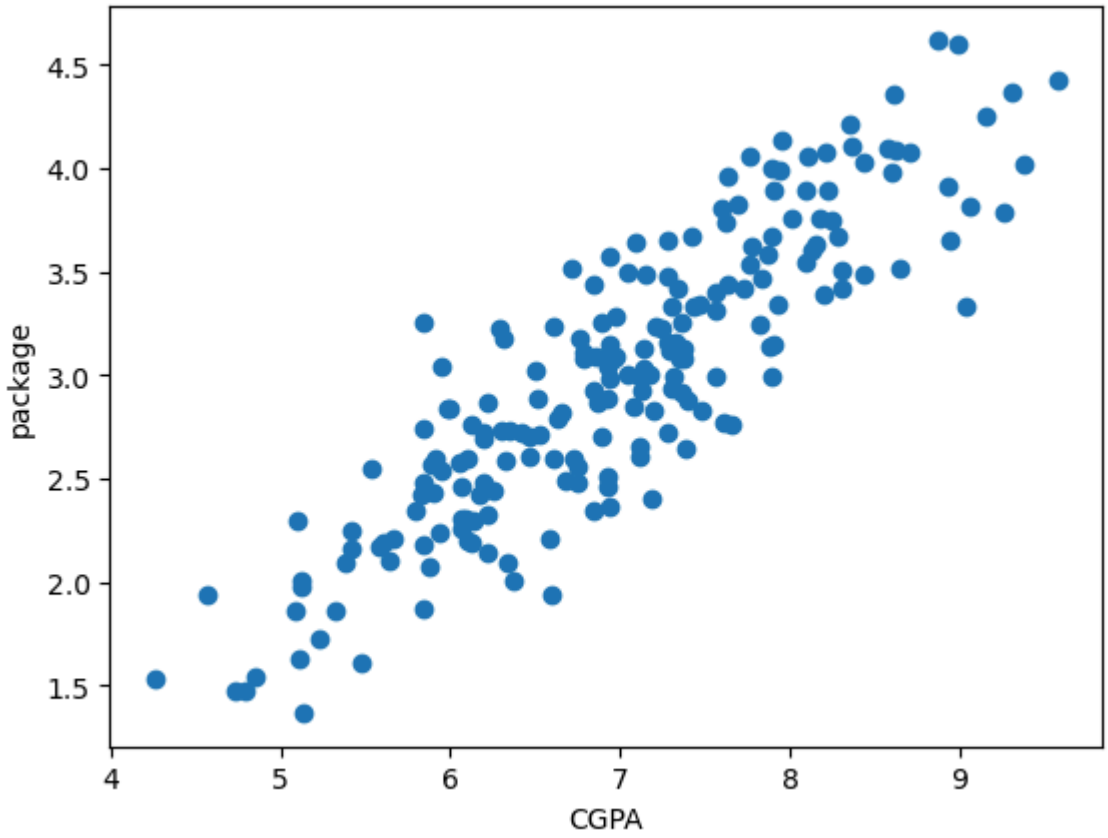
In [13]:
```
df.head()
```

Out[13]:

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57

In [14]:
```
plt.scatter(df["cgpa"],df["package"])
plt.xlabel("CGPA")
plt.ylabel("package")
```

Out[14]:
```
Text(0, 0.5, 'package')
```



In [15]:
```
x=df.iloc[ : ,0:1]
y=df.iloc[ : ,-1]
```

In [16]:
```
x
```

Out[16]:

	cgpa
0	6.89
1	5.12
2	7.82
3	7.42
4	6.94
...	...
195	6.93
196	5.89
197	7.21
198	7.63
199	6.22

200 rows × 1 columns

In [17]:
```
y
```

Out[17]:
```
0      3.26
1      1.98
2      3.25
3      3.67
4      3.57
       ...
195    2.46
196    2.57
197    3.24
198    3.96
199    2.33
Name: package, Length: 200, dtype: float64
```

In [19]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_stat
```

In [20]:
```
from sklearn.linear_model import LinearRegression
```

In [21]:
```
lr=LinearRegression()
```

In [22]:
```
lr.fit(x_train,y_train)
```

Out[22]:
```
LinearRegression()
```

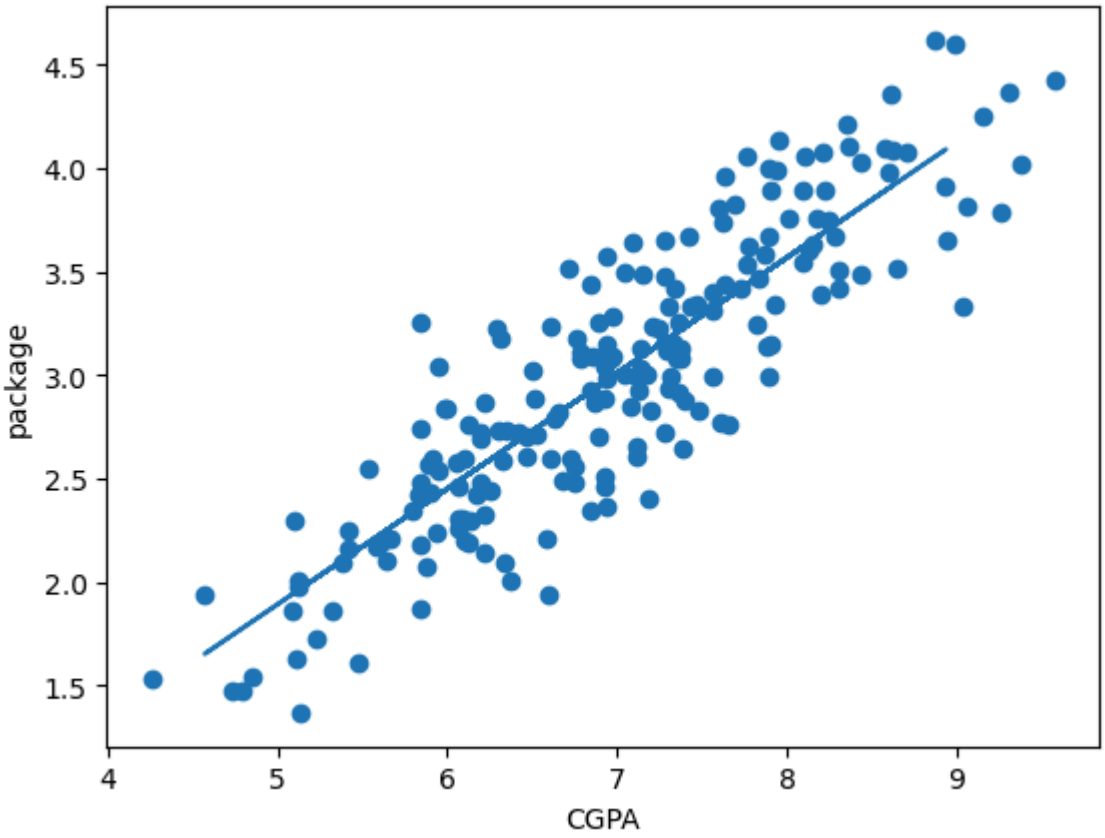
In [24]:
```
lr.predict(x_test.iloc[0].values.reshape(1,1))
```

```
C:\Users\VISHAL\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarni
ng: X does not have valid feature names, but LinearRegression was fitted w
ith feature names
  warnings.warn(
```

Out[24]:
```
array([3.89111601])
```

In [26]:
```
plt.scatter(df["cgpa"],df["package"])
plt.plot(x_test,lr.predict(x_test))
plt.xlabel("CGPA")
plt.ylabel("package")
```

Out[26]:
```
Text(0, 0.5, 'package')
```



In [28]:
```
m=lr.coef_
m
```

Out[28]:
```
array([0.55795197])
```

In [30]:
```
b= lr.intercept_
b
```

Out[30]:
```
-0.8961119222429144
```

In [31]:
```
# y=mx+b
m*8.58+b
```

Out[31]:
```
array([3.89111601])
```

In [32]:
```
y_pred = lr.predict(x_test)
```

In [33]:
```
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

```
0.12129235313495527
```

Regression Metrics

(1) MAE

(2) MSE

(3) RMSE

(4)R2 SCORE

(5)ADJUSTED R2 SCORE

Mean Absolute Error (MAE)

- Mean absolute error, also known as L1 loss is one of the simplest loss functions and an easy-to-understand evaluation metric. It is calculated by taking the absolute difference between the predicted values and the actual values and averaging it across the dataset. Mathematically speaking, it is the arithmetic average of absolute errors. MAE measures only the magnitude of the errors and doesn't concern itself with their direction. The lower the MAE, the higher the accuracy of a model.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} \left| y_i - \hat{y}_i \right|$$

where y_i = actual value, y_hat_i = predicted value, n = sample size

- Pros of the Evaluation Metric:

It is an easy to calculate evaluation metric.

All the errors are weighted on the same scale since absolute values are taken.

It is useful if the training data has outliers as MAE does not penalize high errors caused by outliers.

It provides an even measure of how well the model is performing.

- Cons of the evaluation metric:

Sometimes the large errors coming from the outliers end up being treated as the same as low errors.

MAE follows a scale-dependent accuracy measure where it uses the same scale as the data being measured. Hence it cannot be used to compare series' using different measures.

One of the main disadvantages of MAE is that it is not differentiable at zero. Many optimization algorithms tend to use differentiation to find the optimum value for parameters in the evaluation metric.

It can be challenging to compute gradients in MAE.

In [34]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [ ]:
```
y_pred=lr.predict(x_test)
```

In [35]:
```
y_test.values
```

Out[35]:
```
array([4.1 , 3.49, 2.08, 2.33, 1.94, 1.48, 1.86, 3.09, 4.21, 2.87, 3.65,
       4.  , 2.89, 2.6 , 2.99, 3.25, 1.86, 3.67, 2.37, 3.42, 2.48, 3.65,
       2.6 , 2.83, 4.08, 2.56, 3.58, 3.81, 4.09, 2.01, 3.63, 2.92, 3.51,
       1.94, 2.21, 3.34, 3.34, 3.23, 2.01, 2.61])
```

In [36]:
```
print("MAE",mean_absolute_error(y_test,y_pred))
```

```
MAE 0.2884710931878175
```

Mean Squared Error (MSE)

- MSE is one of the most common regression loss functions. In Mean Squared Error also known as L2 loss, we calculate the error by squaring the difference between the predicted value and actual value and averaging it across the dataset. MSE is also known as Quadratic loss as the penalty is not proportional to the error but to the square of the error. Squaring the error gives higher weight to the outliers, which results in a smooth gradient for small errors. Optimization algorithms benefit from this penalization for large errors as it is helpful in finding the optimum values for parameters. MSE will never be negative since the errors are squared. The value of the error ranges from zero to infinity. MSE increases exponentially with an increase in error. A good model will have an MSE value closer to zero.

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

- Pros of the Evaluation Metric:

MSE values are expressed in quadratic equations. Hence when we plot it, we get a gradient descent with only one global minima. For small errors, it converges to the minima efficiently. There are no local minima.

MSE penalizes the model for having huge errors by squaring them.

It is particularly helpful in weeding out outliers with large errors from the model by putting more weight on them.

- Cons of the evaluation metric:

One of the advantages of MSE becomes a disadvantage when there is a bad prediction. The sensitivity to outliers magnifies the high errors by squaring them.

MSE will have the same effect for a single large error as too many smaller errors. But mostly we will be looking for a model which performs well enough on an overall level.

MSE is scale-dependent as its scale depends on the scale of the data. This makes it highly undesirable for comparing different measures. When a new outlier is introduced into the data, the model will try to take in the outlier. By doing so it will produce a different line of best fit which may cause the final results to be skewed.

In [37]:
```
print("Mse",mean_squared_error(y_test,y_pred))
```

```
Mse 0.12129235313495527
```

Root Mean Squared Error (RMSE)

- RMSE is computed by taking the square root of MSE. RMSE is also called the Root Mean Square Deviation. It measures the average magnitude of the errors and is concerned with the deviations from the actual value. RMSE value with zero indicates that the model has a perfect fit. The lower the RMSE, the better the model and its predictions. A higher RMSE indicates that there is a large deviation from the residual to the ground truth. RMSE can be used with different features as it helps in figuring out if the feature is improving the model's prediction or not.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- Pros of the Evaluation Metric:

RMSE is easy to understand.

It serves as a heuristic for training models.

It is computationally simple and easily differentiable which many optimization algorithms desire.

RMSE does not penalize the errors as much as MSE does due to the square root.

- Cons of the evaluation metric:

Like MSE, RMSE is dependent on the scale of the data. It increases in magnitude if the scale of the error increases.

One major drawback of RMSE is its sensitivity to outliers and the outliers have to be removed for it to function properly.

RMSE increases with an increase in the size of the test sample. This is an issue when we

In [39]:
```
print("RMse",(mean_squared_error(y_test,y_pred))**0.5)
```

```
RMse 0.34827051717731616
```

R2 score

- Coefficient of determination also called as R2 score is used to evaluate the performance of a linear regression model. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.

$$R^2 = 1 - SS_{res} / SS_{tot}$$

Where,

SS_{res} is the sum of squares of the residual errors.

SS_{tot} is the total sum of the errors.

r2 score for perfect model is 1.0

- Conclusion:

The best possible score is 1 which is obtained when the predicted values are the same as the actual values.

R2 score of baseline model is 0.

During the worse cases, R2 score can even be negative.

1-

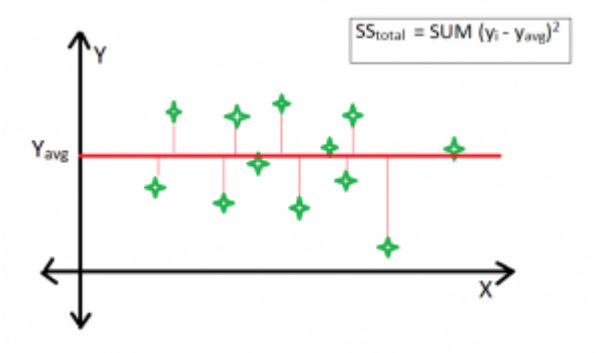
$$\sum_{i-1}^{n} (y_i - \hat{y}_i)^2$$

In [40]:
```
print("R2_score",r2_score(y_test,y_pred))
```

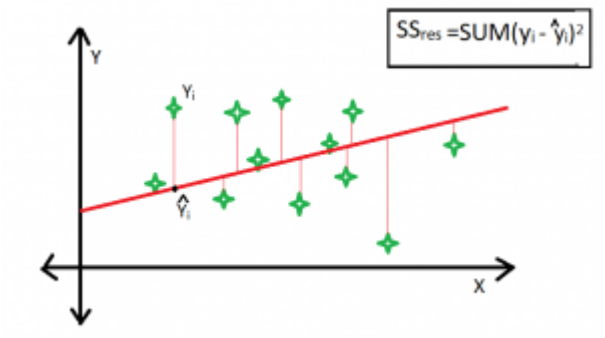
```
R2_score 0.780730147510384
```

Why Adjusted-R Square Test:

- R-square test is used to determine the goodness of fit in regression analysis. Goodness of fit implies how better regression model is fitted to the data points. More is the value of r-square near to 1, better is the model. But the problem lies in the fact that the value of r-square always increases as new variables(attributes) are added to the model, no matter that the newly added attributes have a positive impact on the model or not. also, it can lead to overfitting of the model if there are large no. of variables.
- Adjusted r-square is a modified form of r-square whose value increases if new predictors tend to improve model's performance and decreases if new predictors do not improve performance as expected.
- Average Fitted Line



- Best Fitted Line :



$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

- Clearly, SStot is always fixed for some data points if new predictors are added to the model, but value of SSres decreases as model tries to find some correlations from the added predictors. Hence, r-square's value always increases.
- Adjusted R-Square :

$$R^2_{adj} = 1 - [\frac{(1 - R^2)(n - 1)}{n - k - 1}]$$

- Here, k is the no. of regressors and n is the sample size. if the newly added variable is good enough to improve model's performance, then it will overwhelm the decrease due to k. Otherwise, an increase in k will decrease adjusted r-square value.

In [41]:
```
r2=r2_score(y_test,y_pred)
```

In [42]:
```
r2
```

Out[42]:
```
0.780730147510384
```

In [43]:
```
x_test.shape
```

Out[43]:
```
(40, 1)
```

In [44]:
```
1-((1-r2)*(40-1)/(40-1-1))
```

Out[44]:
```
0.7749598882343415
```

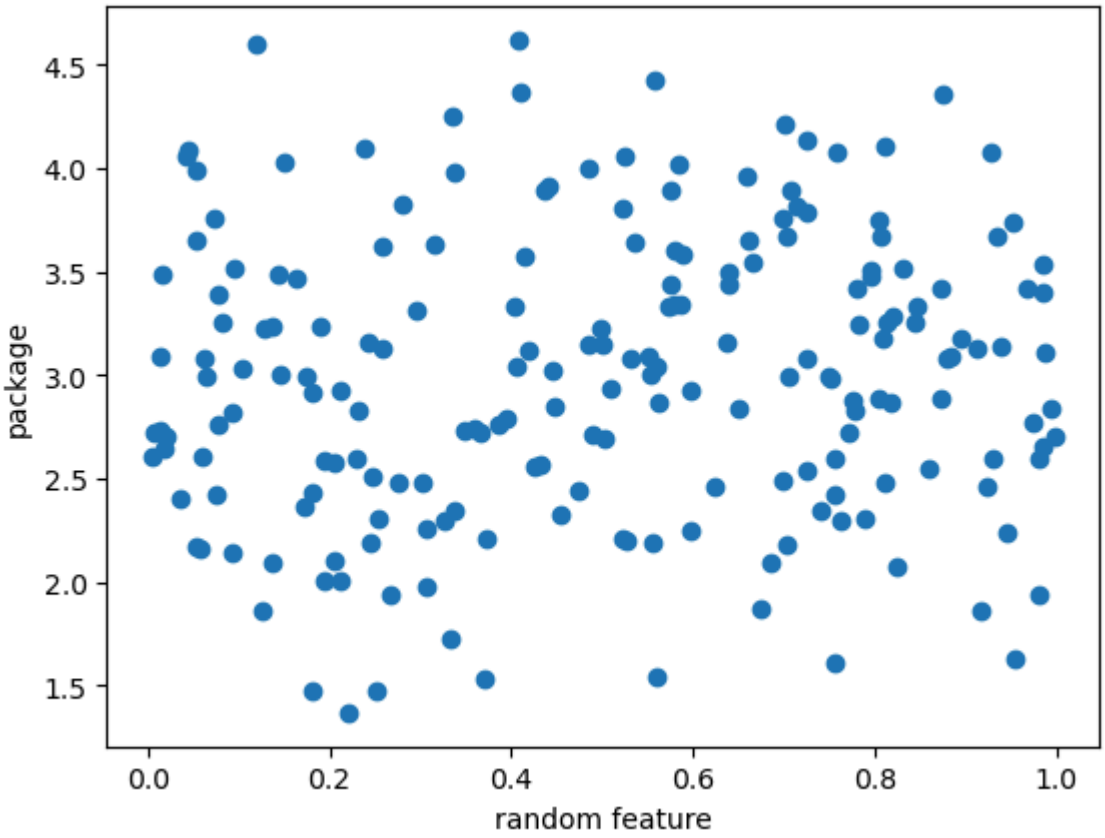
In [45]:
```
new_df1=df.copy()
new_df1["random_feature"]=np.random.random(200)
new_df1=new_df1[["cgpa","random_feature","package"]]
new_df1.head()
```

Out[45]:

	cgpa	random_feature	package
0	6.89	0.813836	3.26
1	5.12	0.306488	1.98
2	7.82	0.783204	3.25
3	7.42	0.806486	3.67
4	6.94	0.414338	3.57

In [46]:
```
plt.scatter(new_df1["random_feature"],new_df1["package"])
plt.xlabel("random feature")
plt.ylabel("package")
```

Out[46]:
```
Text(0, 0.5, 'package')
```



In [47]:
```
x=new_df1.iloc[:,0:2]
y=new_df1.iloc[:,-1]
```

In [48]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_stat
```

In [49]:
```
lr=LinearRegression()
```

In [50]:
```
lr.fit(x_train,y_train)
```

Out[50]:
```
LinearRegression()
```

In [51]:
```
y_pred=lr.predict(x_test)
```

In [52]:
```
print("r2score",r2_score(y_test,y_pred))
r2=r2_score(y_test,y_pred)
```

```
r2score 0.7812009317610142
```

In [54]:
```
1-((1-r2)*(40-1)/(40-1-2))
```

Out[54]:
```
0.7693739550994475
```

Multiple linear regression

In [84]:
```
#Importing necessary libraries and understanding the data
import pandas as pd
import numpy as np
```

In [93]:
```
df = pd.read_csv("car.csv")
```

In [94]:
```
df.head()
```

Out[94]:

	Car	Model	Volume	Weight	CO2	Unnamed: 5
0	Toyota	Aygo	1000	790	99	NaN
1	Mitsubishi	Space Star	1200	1160	95	NaN
2	Skoda	Citigo	1000	929	95	NaN
3	Fiat	500	900	865	90	NaN
4	Mini	Cooper	1500	1140	105	NaN

In [95]:
```
x = df[['Weight', 'Volume']]
y = df['CO2']
```

In [96]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_st
```

In [97]:
```
from sklearn.linear_model import LinearRegression
```

In [98]:
```

lr.fit(X_train,y_train)
```

Out[98]:
```
LinearRegression()
```

In [99]:
```
y_pred = lr.predict(X_test)
```

In [100]:
```

print("MAE",mean_absolute_error(y_test,y_pred))
print("MSE",mean_squared_error(y_test,y_pred))
print("R2 score",r2_score(y_test,y_pred))
```

```
MAE 4.802251146929541
MSE 26.172791255139416
R2 score -0.07100936082411935
```

In [102]:
```

lr.coef_
```

Out[102]:
```
array([0.00490257, 0.01026913])
```

In [103]:
```

lr.intercept_
```

Out[103]:
```
79.51799934393614
```

In [13]:
```
"""Consider variables x and y created from a pandas dataframe  “car.csv” .
Create new column named “Age_car” (Age_car=2023-year)
 For multiple linear regression problem, x contains the independent variable
 ( Age_car , Driven_kms , Fuel_Type , Selling_type , Transmission )  and y c
 the dependent (Selling_Price) variable which is to be predicted.
 Write a Python program to spilt x and y into training and testing datasets
 20% split. Then create a multiple linear regression model using
 the training data and print its coefficients ,intercept and mean squared er
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the data
data = pd.read_csv('car data.csv')
print(data.head())

# Create a new column 'Age_car'
data['Age_car'] = 2023 - data['Year']

# Select independent variables and the dependent variable
X = data[['Age_car', 'Kms_Driven', 'Fuel_Type', 'Seller_Type', 'Transmissior
y = data['Selling_Price']

# Convert categorical variables into dummy/indicator variables
X = pd.get_dummies(X, drop_first=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Create and train the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Find the coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)

# Display the results
print("Coefficients:", coefficients)
print("Intercept:", intercept)
print("Mean Squared Error:", mse)
```

```
  Car_Name  Year  Selling_Price  Present_Price  Kms_Driven Fuel_Type  \
0     ritz  2014           3.35           5.59       27000    Petrol
1      sx4  2013           4.75           9.54       43000    Diesel
2     ciaz  2017           7.25           9.85        6900    Petrol
3  wagon r  2011           2.85           4.15        5200    Petrol
4    swift  2014           4.60           6.87       42450    Diesel

  Seller_Type Transmission  Owner
0      Dealer       Manual      0
1      Dealer       Manual      0
2      Dealer       Manual      0
3      Dealer       Manual      0
4      Dealer       Manual      0
Coefficients: [-2.79387747e-01 -4.48042783e-06  6.40927579e+00  1.24649346
e+00
 -4.11102502e+00 -4.80303386e+00]
Intercept: 10.888445135748395
Mean Squared Error: 9.418108720809979
```

Polynomial Regression

It is the extension of the simple linear model by adding extra predictors obtained by raising (squaring) each of the original predictors to a power.

•For example, if there are three variables, X, X2, and X3 are used as predictors. This approach provides a simple way to yield a non-linear fit to data

$y=\beta_0+\beta_1x+\beta_2x^2+\ldots+\beta_nx^n+\epsilon$
where,

y is the dependent variable. x is the independent variable. $\beta_0,\beta_1,\ldots,\beta_n$ are the coefficients of the polynomial terms. n is the degree of the polynomial. ϵrepresents the error term

In [20]:
```
# Importing the libraries
# y = a + b_1x + b_2x^2 +....+ b_nx^n
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
file_name = "https://raw.githubusercontent.com/Jovita7/Data-Analysis-and-Vis
df = pd.read_csv(file_name)
df.head(7)
```

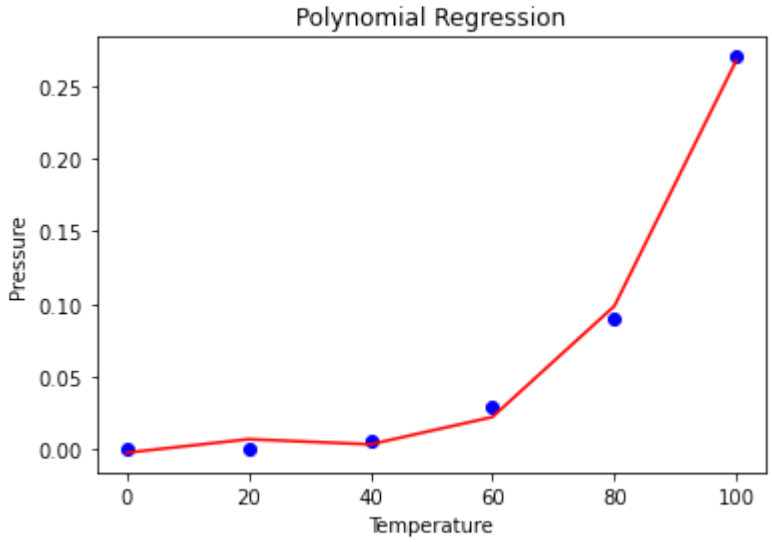
Out[20]:

	sno	Temperature	Pressure
0	1	0	0.0002
1	2	20	0.0012
2	3	40	0.0060
3	4	60	0.0300
4	5	80	0.0900
5	6	100	0.2700

In [21]:
```
# Extract our x values, the column Temperature
x = df.iloc[:, 1:2].values

# Extract our y or target variable Pressure
y = df.iloc[:, 2].values
x
y
```

Out[21]:
```
array([2.0e-04, 1.2e-03, 6.0e-03, 3.0e-02, 9.0e-02, 2.7e-01])
```

In [22]:
```
# Fitting Polynomial Regression to the dataset
# Fitting the Polynomial Regression model on two components X and y.
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)
model_Poly = poly.fit_transform(x)


lin_poly = LinearRegression()
lin_poly.fit(model_Poly, y)

# Visualising the Polynomial Regression results
plt.scatter(x, y, color = 'blue')

plt.plot(x, lin_poly.predict(model_Poly), color = 'red')
plt.title('Polynomial Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')

plt.show()
```



In [23]:
```
from sklearn.linear_model import LinearRegression
lin = LinearRegression()

lin.fit(x, y)
x_t = np.array([70]).reshape(-1, 1)
x_t.shape
```

Out[23]:
```
(1, 1)
```

In [24]:
```
lin.predict(x_t)
```

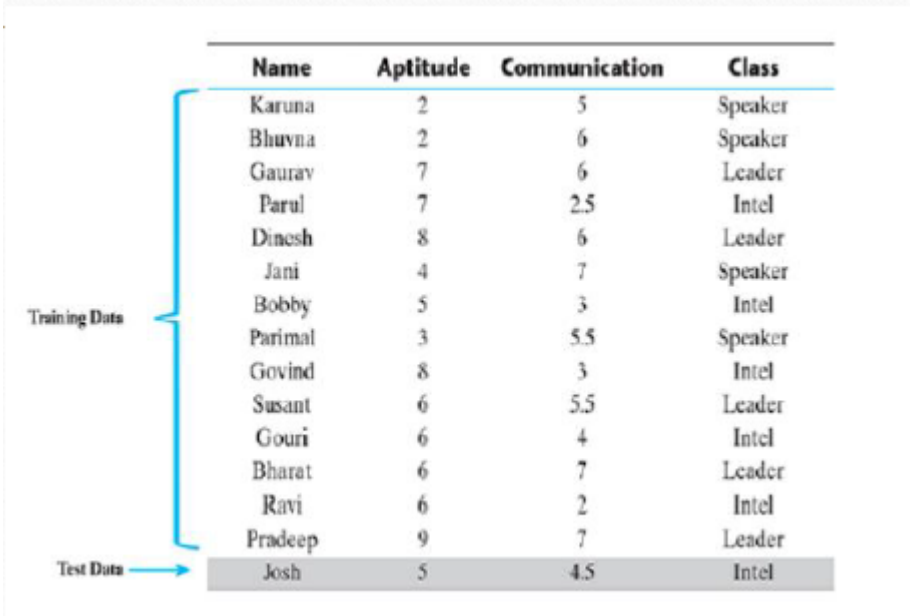
Out[24]:
```
array([0.11307333])
```

In [25]:
```
pred2array = np.array([[70]]) #predict prssure for 70 degree temperature
lin_poly.predict(poly.fit_transform(pred2array))
```

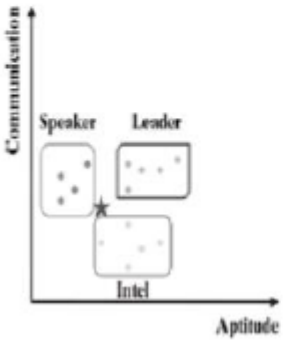
Out[25]:
```
array([0.05120625])
```

kNN–k Nearest Neighbours

Name	Aptitude	Communication	Class
Karuna	2	5	Speaker
Bhuvna	2	6	Speaker
Gaurav	7	6	Leader
Parul	7	2.5	Intel
Dinesh	8	6	Leader
Jani	4	7	Speaker
Bobby	5	3	Intel
Parimal	3	5.5	Speaker
Govind	8	3	Intel
Susant	6	5.5	Leader
Gouri	6	4	Intel
Bharat	6	7	Leader
Ravi	6	2	Intel
Pradeep	9	7	Leader
Josh	5	4.5	Intel



Name	Aptitude	Communication	Class
Karuna	2	5	Speaker
Bhuvna	2	6	Speaker
Gaurav	7	6	Leader
Parul	7	2.5	Intel
Dinesh	8	6	Leader
Jani	4	7	Speaker
Bobby	5	3	Intel
Parimal	3	5.5	Speaker
Govind	8	3	Intel
Susant	6	5.5	Leader
Gouri	6	4	Intel
Bharat	6	7	Leader
Ravi	6	2	Intel
Pradeep	9	7	Leader
Josh	5	4.5	Intel



Name	Aptitude	Communication	Class
Karuna	2	5	Speaker
Bhuvna	2	6	Speaker
Gaurav	7	6	Leader
Parul	7	2.5	Intel
Dinesh	8	6	Leader
Jani	4	7	Speaker
Bobby	5	3	Intel
Parimal	3	5.5	Speaker
Govind	8	3	Intel
Susant	6	5.5	Leader
Gouri	6	4	Intel
Bharat	6	7	Leader
Ravi	6	2	Intel
Pradeep	9	7	Leader
Josh	5	4.5	???

Name	Aptitude	Communication	Class	Distance	k = 1	k = 2	k = 3
Karuna	2	5	Speaker	3.041			
Bhuvna	2	6	Speaker	3.354			
Parimal	3	5.5	Speaker	2.236			
Jani	4	7	Speaker	2.693			
Bobby	5	3	Intel	1.500			1.500
Ravi	6	2	Intel	2.693			
Gouri	6	4	Intel	1.118	1.118	1.118	1.118
Parul	7	2.5	Intel	2.828			
Govind	8	3	Intel	3.354			
Susant	6	5.5	Leader	1.414			
Bharat	6	7	Leader	2.693			
Gaurav	7	6	Leader	2.500			
Dinesh	8	6	Leader	3.354			
Pradeep	9	7	Leader	4.717			
Josh	5	4.5	???				

)

$$\text{Euclidean distance} = \sqrt{(f_{11} - f_{12})^2 + (f_{21} - f_{22})^2}$$

where f_{11} = value of feature f_1 for data element d_1

f_{12} = value of feature f_1 for data element d_2

f_{21} = value of feature f_2 for data element d_1

f_{22} = value of feature f_2 for data element d_2

Confusion Matrix

•There are four possibilities with regards to the cricket match win/loss prediction:

•1. the model predicted win and the team won (TP)

•2. the model predicted win and the team lost (FP)

•3. the model predicted loss and the team won (FN)

•4. the model predicted loss and the team lost (TN)

Model Accuracy

$$\text{Model accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

	ACTUAL WIN	ACTUAL LOSS
Predicted Win	85	4
Predicted Loss	2	9

In context of the above confusion matrix, total count of TPs = 85, count of FPs = 4, count of FNs = 2 and count of TNs = 9.

$$\therefore \text{Model accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{85 + 9}{85 + 4 + 2 + 9} = \frac{94}{100} = 94\%$$

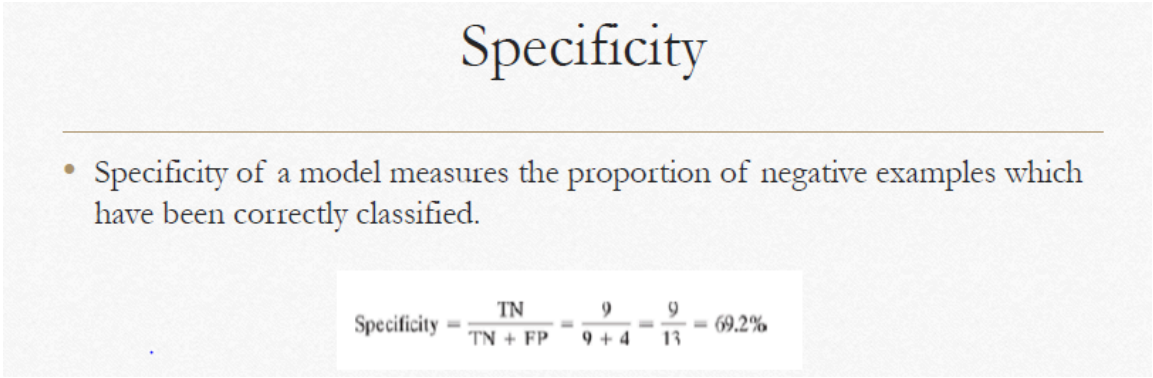
Error Rate

$$\text{Error rate} = \frac{FP + FN}{TP + FP + FN + TN} = \frac{4 + 2}{85 + 4 + 2 + 9} = \frac{6}{100} = 6\%$$
$$= 1 - \text{Model accuracy}$$

Sensitivity

- The sensitivity of a model measures the proportion of TP examples or positive cases which were correctly classified.

$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{85}{85 + 2} = \frac{85}{87} = 97.7\%$$



In [26]:
```
pip show scikit-learn
```

```
Name: scikit-learn
Version: 0.23.2
Summary: A set of python modules for machine learning and data mining
Home-page: http://scikit-learn.org (http://scikit-learn.org)
Author:
Author-email:
License: new BSD
Location: c:\programdata\anaconda3\lib\site-packages
Requires: joblib, numpy, scipy, threadpoolctl
Required-by:
Note: you may need to restart the kernel to use updated packages.

WARNING: Ignoring invalid distribution -ordcloud (c:\programdata\anaconda3
\lib\site-packages)
```

In [27]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

In [29]:
```
data = pd.read_csv("https://raw.githubusercontent.com/Jovita7/datasets/main/
data.head(100)
```

Out[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2
...	...	...	...	...	...	...	
95	6	144	72	27	228	33.9	0
96	2	92	62	28	0	31.6	0
97	1	71	48	18	76	20.4	0
98	6	93	50	30	64	28.7	0
99	1	122	90	51	220	49.7	0

100 rows × 9 columns

In [30]:
```
data.shape
```

Out[30]:
```
(768, 9)
```

In [31]:
```
#Segregating predictor variables
x = data.iloc[:, 0:8]
x
```

Out[31]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 8 columns

In [32]:
```
#Segregating the target/class variable
y = data['Outcome']
y
```

Out[32]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [33]:
```
#split into training and test datasets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

In [34]:
```
x_train.shape
```

Out[34]:
```
(614, 8)
```

In [37]:
```
x_train
```

Out[37]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
375	12	140	82	43	325	39.2	
156	2	99	52	15	94	24.6	
184	4	141	74	0	0	27.6	
419	3	129	64	29	115	26.4	
149	2	90	70	17	0	27.3	
...	...	...	...	...	...	...	
677	0	93	60	0	0	35.3	
166	3	148	66	25	0	32.5	
378	4	156	75	0	0	48.3	
596	0	67	76	0	0	45.3	
292	2	128	78	37	182	43.3	

614 rows × 8 columns

In [35]:
```
y_train
```

Out[35]:
```
375    1
156    0
184    0
419    1
149    0
      ..
677    0
166    0
378    1
596    0
292    1
Name: Outcome, Length: 614, dtype: int64
```

In [38]:
```
#kNN Classifier with k=6 means 6 closest neighbours are considered
nn = KNeighborsClassifier(n_neighbors=27)
```

In [39]:
```
#Train the classifier with the training data
model = nn.fit(x_train, y_train)
prediction = model.predict(x_test)
prediction
```

Out[39]:
```
array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [40]:
```
#Metric Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, prediction)
cm
```

Out[40]:
```
array([[87,  6],
       [34, 27]], dtype=int64)
```

In [41]:
```
TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]
print(TP, FN, TN, FP)
```

```
27 34 87 6
```

In [42]:
```
accuracy = (TP + TN) / (TP+FP+FN+TN)
accuracy
```

Out[42]:
```
0.7402597402597403
```

In [43]:
```
print("error rate= ",1-accuracy)
```

```
error rate=  0.2597402597402597
```

In [44]:
```
sensitivity = TP / (TP + FN)
sensitivity
```

Out[44]:
```
0.4426229508196721
```

In [45]:
```
specificity = TN / (TN + FP)
specificity
```

Out[45]:
```
0.9354838709677419
```

In [46]:
```
x.columns
```

Out[46]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

In [47]:
```
a = [[2, 94, 90, 21, 0, 22.3, 0.75,45]]
y = model.predict(a)
y
```

Out[47]:
```
array([0], dtype=int64)
```

In [58]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

#print(irisData)

# Create feature and target arrays
X = irisData.data
y = irisData.target
print(y)

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
0.9666666666666667
```

In [61]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
p=knn.predict(X_test)
print(p)
```

```
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```

In [62]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,p)
cm
```

Out[62]:
```
array([[10,  0,  0],
       [ 0,  8,  1],
       [ 0,  0, 11]], dtype=int64)
```

Let us calculate the TP, TN, FP, and FN values for the class Setosa using the Above tricks:

TP: The actual value and predicted value should be the same. So concerning Setosa class, the value of cell 1 is the TP value.

FN: The sum of values of corresponding rows except for the TP value

FN = (cell 2 + cell3)

= (0 + 0)

= 0

FP: The sum of values of the corresponding column except for the TP value.

FP = (cell 4 + cell 7)

= (0 + 0)

= 0

TN: The sum of values of all columns and rows except the values of that class that we are calculating the values for.

TN = (cell 5 + cell 6 + cell 8 + cell 9)

= 8 + 1 +0 + 11

= 20

Similarly, for the Versicolor class, the values/metrics are calculated as below:

TP: 8 (cell 5)

FN : 0 + 1 = 1 (cell 4 +cell 6)

FP : 0 + 0 = 0 (cell 2 + cell 8)

TN : 10 +0 +0 + 11 =21 (cell 1 + cell 3 + cell 7 + cell 9).

In [ ]: