SUBJECT: Data Science and Its Applications (21AD62)

MODULE-1 INTRODUCTION

Syllabus: **What is Data Science? Visualizing Data**, matplotlib, Bar Charts, Line Charts, Scatterplots, **Linear Algebra**, Vectors, Matrices, **Statistics**, Describing a Single Set of Data, Correlation, Simpson's Paradox, Some Other Correlational Caveats, Correlation and Causation, **Probability**, Dependence and Independence, Conditional Probability, Bayes's Theorem, Random Variables, Continuous Distributions, The Normal Distribution, The Central Limit Theorem.

Introduction

Data science is the study of data to extract meaningful insights for business which combines tools, methods, and technology to generate meaning from data. It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyze large amounts of data.

Data science is used to study data in four main ways:

1. **Descriptive analysis**

Descriptive analysis examines data to gain insights into what happened or what is happening in the data environment. It is characterized by data visualizations such as pie charts, bar charts, line graphs, tables, or generated narratives.

For example, a flight booking service may record data like the number of tickets booked each day. Descriptive analysis will reveal booking spikes, booking slumps, and high-performing months for this service.

2. Diagnostic analysis

Diagnostic analysis is a deep-dive or detailed data examination to understand why something happened. It is characterized by techniques such as drill-down, data discovery, data mining, and correlations. Multiple data operations and transformations may be performed on a given data set to discover unique patterns in each of these techniques.

For example, the flight service might drill down on a particularly high-performing month to better understand the booking spike. This may lead to the discovery that many customers visit a particular city to attend a monthly sporting event.

3. Predictive analysis

Predictive analysis uses historical data to make accurate forecasts about data patterns that may occur in the future. It is characterized by techniques such as machine learning, forecasting, pattern matching, and predictive modeling. In each of these techniques, computers are trained to reverse engineer causality connections in the data.

For example, the flight service team might use data science to predict flight booking patterns for the coming year at the start of each year. The computer program or algorithm may look at past data and predict booking spikes for certain destinations in May. Having anticipated their customer's future travel requirements, the company could start targeted advertising for those cities from February.

4. Prescriptive analysis

Prescriptive analytics takes predictive data to the next level. It not only predicts what is likely to happen but also suggests an optimum response to that outcome. It can analyze the potential implications of different choices and recommend the best course of action. It uses graph analysis, simulation, complex event processing, neural networks, and recommendation engines from machine learning.

Back to the flight booking example, prescriptive analysis could look at historical marketing campaigns to maximize the advantage of the upcoming booking spike. A data scientist could project booking outcomes for different levels of marketing spend on various marketing channels. These data forecasts would give the flight booking company greater confidence in their marketing decisions.

Benefits of data science

- Discover unknown transformative patterns
- Innovate new products and solution
- Real-time optimization

Data Pre processing

Data preprocessing is an important step in the data mining process. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific data mining task.

Some common steps in data preprocessing include:

Data Cleaning: This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, and duplicates. Various techniques can be used for data cleaning, such as imputation, removal, and transformation.

Data Integration: This involves combining data from multiple sources to create a unified dataset. Data integration can be challenging as it requires handling data with different formats, structures, and semantics. Techniques such as record linkage and data fusion can be used for data integration.

Data Transformation: This involves converting the data into a suitable format for analysis. Common techniques used in data transformation include normalization, standardization, and discretization. Normalization is used to scale the data to a common range, while standardization is used to transform the data to have zero mean and unit variance. Discretization is used to convert continuous data into discrete categories.

=

Data Reduction: This involves reducing the size of the dataset while preserving the important information. Data reduction can be achieved through techniques such as feature selection and feature extraction. Feature selection involves selecting a subset of relevant features from the dataset, while feature extraction involves transforming the data into a lower-dimensional space while preserving the important information.

Data Discretization: This involves dividing continuous data into discrete categories or intervals. Discretization is often used in data mining and machine learning algorithms that require categorical data. Discretization can be achieved through techniques such as equal width binning, equal frequency binning, and clustering.

Data Normalization: This involves scaling the data to a common range, such as between 0 and 1 or -1 and 1. Normalization is often used to handle data with different units and scales. Common normalization techniques include min-max normalization, z-score normalization, and decimal scaling.

Data preprocessing plays a crucial role in ensuring the quality of data and the accuracy of the analysis results. The specific steps involved in data pre processing may vary depending on the nature of the data and the analysis goals.

By performing these steps, the data mining process becomes more efficient and the results become more accurate.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that involves examining datasets to summarize their main characteristics, often using visual methods. The primary goal of EDA is to uncover patterns, spot anomalies, test hypotheses, and check assumptions through the use of summary statistics and graphical representations.

Objectives of EDA

1. **Understand Data Structure:** Determine the types of variables and their relationships.
2. **Summarize Data Characteristics:** Use summary statistics to get an overall sense of the data.
3. **Detect Outliers and Anomalies:** Identify any unusual observations that may need further investigation.
4. **Discover Patterns and Relationships:** Reveal trends, correlations, and other patterns.
5. **Check Assumptions:** Validate the assumptions of statistical models you may use later.

Steps in EDA

1. **Data Collection and Loading:**Import the data from various sources (e.g., CSV files, databases).
2. **Data Cleaning:** Handle missing values (e.g., imputation, removal),Correct errors and inconsistencies (e.g., duplicate records, incorrect data types)
3. **Descriptive Statistics:** **a) Central Tendency:** Mean, median, mode.
   b) Dispersion: Range, variance, standard deviation
   c) **Shape:** Skewness, kurtosis.

4. Data Visualization:

1. **Univariate Analysis:** Analyzing a single variable.
   1. **Histograms:** For continuous variables.
   2. **Bar Plots:** For categorical variables.
   3. **Box Plots:** For detecting outliers and understanding the distribution.
2. **Bivariate Analysis:** Analyzing relationships between two variables.
   1. **Scatter Plots:** To study correlations.
   2. **Line Graphs:** For time series data.
   3. **Heatmaps:** To visualize correlation matrices.

3. Multivariate Analysis: Analyzing relationships among more than two variables.

1. **Pair Plots:** To visualize relationships between multiple pairs of variables.
2. **3D Scatter Plots:** For understanding the interaction between three variables.

Correlation Analysis:

1. **Correlation Matrix:** To understand the pairwise correlations between variables.
2. **Heatmap:** To visualize the correlation matrix.

Hypothesis Testing:

1. **T-tests, Chi-square tests, ANOVA:** To test assumptions and hypotheses about data relationships.

Examples of EDA
Sales Data Analysis:

1. **Summary Statistics:** Calculate the average, minimum, and maximum sales.
2. **Histograms:** Show the distribution of sales amounts.
3. **Box Plots**: Identify outliers in sales data.
4. **Scatter Plots**: Analyze the relationship between advertising spend and sales.

Customer Data Analysis:

1. **Descriptive Statistics:** Understand the demographics of customers (e.g., age, income).
2. **Bar Plots:** Visualize the frequency distribution of customer segments.
3. **Heatmap:** Correlation between different customer attributes (e.g., age, income, spending score).

Data visualization

Data visualization is the graphical representation of information and data where data is representing in the form of graphs or charts. It helps to understand large and complex amounts of data very easily. It allows the decision-makers to make decisions very efficiently and also allows them in identifying new trends and patterns very easily. It is also used in high-level data analysis for Machine Learning and Exploratory Data Analysis (EDA). Data visualization can be done with various tools like Tableau, Power BI, Python.

There are two primary uses for data visualization:

- To explore data
- To communicate data

1.Purpose of Data visualization

- To simplify complex data: by bring large amount data can be bring to one graph helps to analyze the data:
- To highlight relationships, patterns, and trends:
- To support data-driven decision-making

2. **Common Visualization Techniques:**

- Charts: Bar charts, line charts, pie charts.
- Graphs: Scatter plots, histograms, box plots.
- Maps: Geographic maps to represent spatial data.
- Interactive Dashboards: Tools that allow users to interact with the data, such as filtering and zooming.

3. **Tools and Software:**

- Libraries: Matplotlib, Seaborn, Plotly (Python); ggplot2 (R).
- Software: Tableau, Power BI, D3.js for web-based visualizations.

4. **Benefits:**

- Enhances comprehension by converting data into a visual context.
- Makes it easier to spot trends and outliers.
- Facilitates communication of insights to stakeholders.

Advantages

- Data visualization is a form of visual art that grabs our interest and keeps our eyes on the message.
- Easily sharing information.
- Interactively explore opportunities.
- Visualize patterns and relationships.

Disadvantages

- When viewing visualization with many different data points, it's easy to make an inaccurate assumption.
- Biased or inaccurate information.
- Correlation doesn't always mean causation.
- Core messages can get lost in translation.

matplotlib

matplotlib is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consists of several plots like line chart, bar chart, histogram, etc. It provides a lot of flexibility but at the cost of writing more code.

Here are some key aspects of Matplotlib:

1. Core Features: 2D Plotting: Create line plots, scatter plots, bar charts, histograms, pie charts, and more.
2. Customization: Extensive options to customize plots, including colors, labels, line styles, and annotations (comment added to a text).
3. Subplots: Support for creating complex figures with multiple subplots in a single figure. 2. Usage:
4. Integration: Works seamlessly with other scientific libraries like NumPy, Pandas, and SciPy.

5. Scripting: Suitable for scripting and quick plotting from the Python shell or Jupyter notebooks.
6. Publication-Quality Figures: Capable of producing high-quality figures for publications and presentations.

To install Matplotlib type the below command in the terminal.

pip install matplotlib

For example,

```
from matplotlib import pyplot as plt
years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years, gdp, color='green', marker='o', linestyle='solid')
 # add a title
 plt.title("Nominal GDP")
 # add a label to the y-axis
plt.ylabel("Billions of $")
plt.show()
```

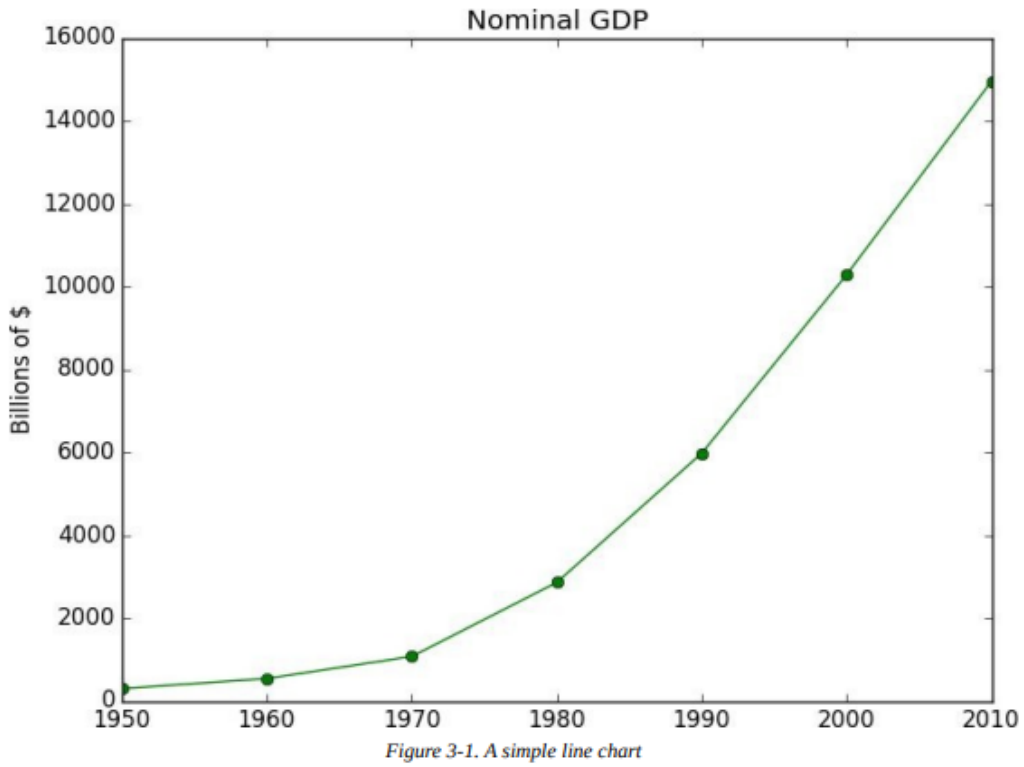


Figure 3-1. A simple line chart

Line Charts

Write Python program to plot Line chart by assuming your own data and explain the various attributes of line chart.

```
import matplotlib.pyplot as plt
# Sample data
x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]
 # Creating the line chart
plt.plot(x, y, color='blue', marker='o', linestyle='-', linewidth=2, markersize=8)
# Adding title and labels
 plt.title('Sample Line Chart') plt.xlabel('X-axis Label')
 plt.ylabel('Y-axis Label')
 # Adding grid
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
 # Adding legend
plt.legend(['Sample Data'], loc='upper left')
# Display the line chart
plt.show()
```

Explanation of Various Attributes

1. Data for Line Chart:

x: List of values for the x-axis ([1, 2, 3, 4, 5]).
y: Corresponding values for each point on the y-axis ([10, 20, 15, 25, 30]).

2. plt.plot( ) Function:

plt.plot(x, y):

This function creates the line chart with the x values on the x-axis and their corresponding y values on the y-axis.
color='blue': Sets the color of the line to blue.
marker='o': Uses a circle marker for each data point. L
inestyle='-': Sets the style of the line to solid.
linewidth=2: Sets the width of the line to 2 points.
markersize=8: Sets the size of the markers to 8 points.

3. Title and Labels:

plt.title('Sample Line Chart'): Adds a title to the line chart.
plt.xlabel('X-axis Label'): Labels the x-axis as 'X-axis Label'.
plt.ylabel('Y-axis Label'): Labels the y-axis as 'Y-axis Label'.

4.Grid:

plt.grid(True, which='both', linestyle='--', linewidth=0.5):
Adds a grid to the chart to improve readability.
True: Enables the grid. which='both': Applies the grid to both major and minor ticks. linestyle='--': Sets the style of the grid lines to dashed.
linewidth=0.5: Sets the width of the grid lines to 0.5 points.

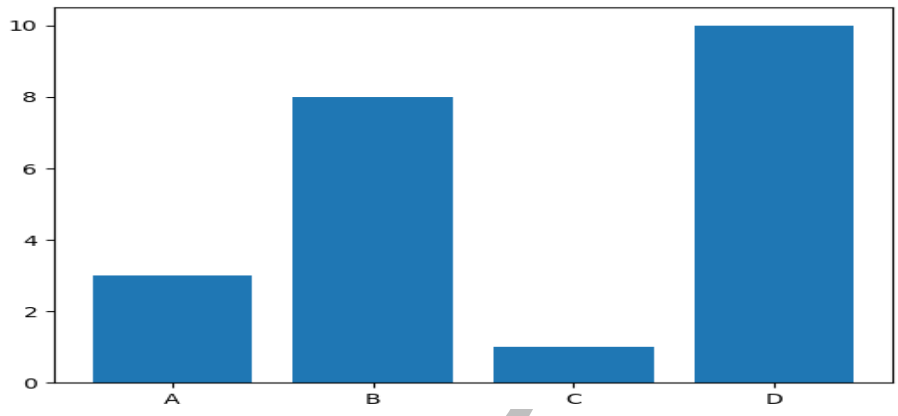
5.Legend:
plt.legend(['Sample Data'], loc='upper left'):
Adds a legend to the chart. ['Sample Data']: List of labels for the legend. loc='upper left': Positions the legend in the upper left corner of the chart.

6.plt.show():
plt.show(): Displays the line chart.

Bar Charts

A bar chart is a good choice to show how some quantity varies among some discrete set of items. For Example:

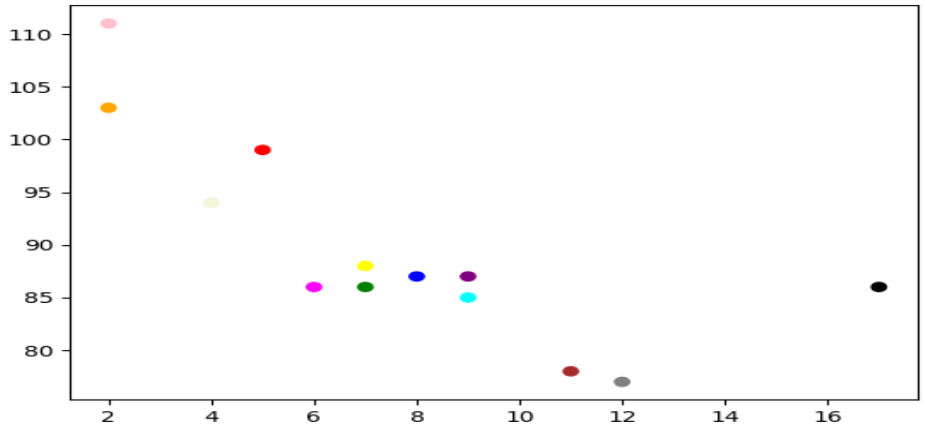
```
import matplotlib.pyplot as plt
import numpy as np
x=np.array(["A", "B", "C", "D"])
y=np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```



Scatterplots

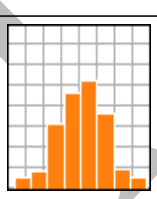
A scatterplot is the right choice for visualizing the relationship between two paired sets of data.
Example

```
import matplotlib.pyplot as plt
import numpy as np

x=np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y=np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors=np.array(["red","green","blue","yellow","pink","black","orange","purple","beige","brown",
"gray","cyan","magenta"])
plt.scatter(x,y,c=colors)

plt.show()
```



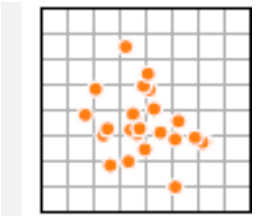
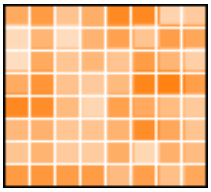
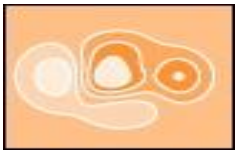
Histogram

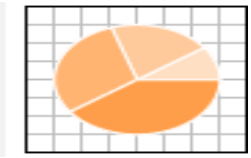
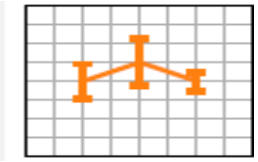
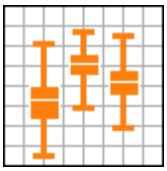
Histogram Write Python program to plot histogram by assuming your own data and explain the various attributes of histogram.

import numpy as np import matplotlib.pyplot as plt Z = np.random.normal(0, 1, 100) ax.hist(Z)	

Scatterplots

Write Python program to plot Scatterplot by assuming your own data and explain the various attributes of Scatterplot.

import numpy as np import matplotlib.pyplot as pltX = np.random.uniform(0, 1, 100) Y = np.random.uniform(0, 1, 100) ax.scatter(X, Y)	
import numpy as np import matplotlib.pyplot as pltZ = np.random.uniform(0, 1, (8,8)) ax.imshow(Z)	
Z = np.random.uniform(0, 1, (8,8)) ax.contourf(Z)	

Z = np.random.uniform(0, 1, 4) ax.pie(Z)	
X = np.arange(5) Y = np.random.uniform(0, 1, 5) ax.errorbar(X, Y, Y/4)	
Z = np.random.normal(0, 1, (100,3)) ax.boxplot(Z)	

Linear Algebra for data science

- Branch of Mathematics that deals with Vector Spaces
- Linear algebra is a foundational component of data science,underpinning many of the algorithms and techniques used for data manipulation, analysis, and machine learning.
- Linear algebra is a branch of mathematics that deals with vector spaces and linear transformations between them. It involves the study of vectors, matrices, and systems of linear equations, and explores concepts such as vector addition, scalar multiplication, matrix operations, determinants, eigenvalues, and eigenvectors.
- It forms the backbone of machine learning algorithms, enabling operations like matrix multiplication, which are essential to model training and prediction.
- Linear algebra techniques facilitate dimensionality reduction, enhancing the performance of data processing and interpretation.
- Eigenvalues and eigenvectors help understand data records variability, influencing clustering and pattern recognition.
- Solving systems of equations is crucial for optimization tasks and parameter estimation.
- Linear algebra supports image and signal processing strategies critical in data analysis.
- Proficiency in linear algebra empowers data scientists to successfully represent, control, and extract insights from data, in the end driving the development of accurate models and informed decision-making.

Applications of linear algebra in data science:
Linear Algebra Concepts

1. **Scalar**: A single number.

2. **Vectors:**
A vector is a list of numbers representing data points or features in a dataset.

Vector: An ordered array of numbers, e.g., v=[v1,v2,...,vn]. Operations: addition, subtraction, scalar multiplication, and dot product.

Vector operations

1. addition: a+b=[a1+b1,a2+b2,...,an+bn]
2. subtraction: a−b=[a1−b1,a2−b2,...,an−bn]
3. scalar multiplication: ca=[ca1,ca2,...,can]
4. dot product: a·b=∑i=1naibi
5. norm (magnitude): ‖a‖=a·a=∑i=1nai2

Write Python program to add two vectors and multiply a vector by a scalar

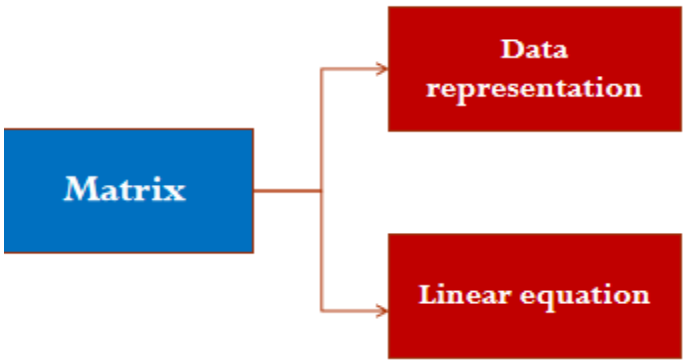
```
def vector_add(v, w):

"""Adds corresponding elements of two vectors."""

return [v_i + w_i for v_i, w_i in zip(v, w)]

def scalar_multiply(c, v):

"""Multiplies every element of vector v by the scalar c."""

return [c * v_i for v_i in v]

# Example vectors

v = [1, 2, 3]

w = [4, 5, 6]

# Scalar value

c = 2

# Adding two vectors

result_addition = vector_add(v, w)

print(f"Vector addition of {v} and {w} is {result_addition}")

# Multiplying vector by a scalar

result_scalar_multiplication = scalar_multiply(c, v)

print(f"Scalar multiplication of {v} by {c} is {result_scalar_multiplication}")
```

Output

```
Vector addition of [1, 2, 3] and [4, 5, 6] is [5, 7, 9]

Scalar multiplication of [1, 2, 3] by 2 is [2, 4, 6]
```

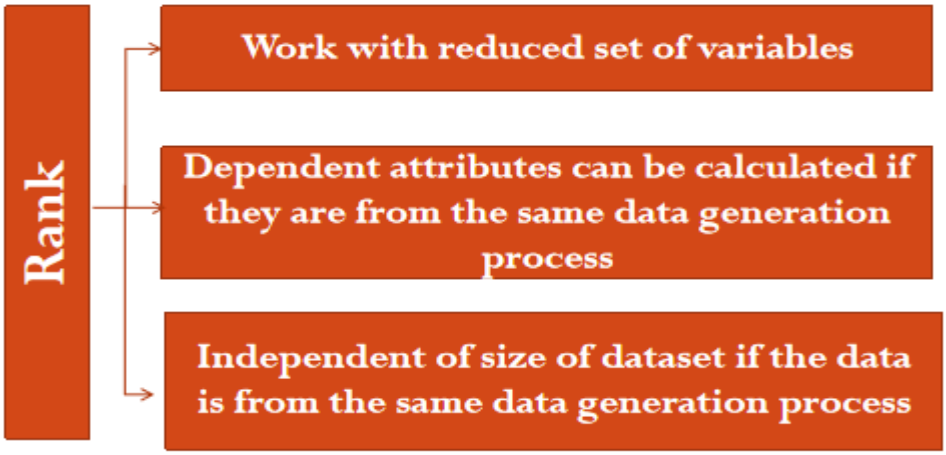
Data Representation becomes an important aspect of data science and data is represented usually in a matrix form.

To uncover the relations between variables linear algebraic tools are used.



Rank of a Matrix

It refers to the number of linearly independent rows or columns of the matrix



Null space and Nullity

- Linear relationships among attributes, is answered by the concepts of null space and nullity.
- The null space of any matrix A consists of all the vectors B such that AB = 0 and B is not zero.
- It can also be obtained from AB = 0 where A is known matrix of size m x n and B is matrix to be found of size n x k.
- Every null space vector corresponds to one linear relationship.
- Nullity can be defined as the number of vectors present in the null space of a given matrix.
- In other words, the dimension of the null space of the matrix A is called the nullity of A

Rank nullity Theorem

Consider a data matrix a with a null space and nullity

Rank nullity theorem helps us to relate the nullity of a data matrix to the rank and the number of attributes in the data

According to nullity theorem

Statistics

Statistics is the science of analyzing data and which helps to understand the data. Statistics play a crucial role in data science, enabling professionals to make sense of data, draw valid conclusions, and make informed decisions.

There are two types of statistics:

1. Descriptive Statistics: These are used to summarize and describe the main features of a dataset.
2. Inferential Statistics: These are used to draw conclusion fro that data.

The purpose of descriptive and inferential statistics is to analyze different types of data using different tools. Descriptive statistics helps to describe and organize known data using charts, bar graphs, etc., while inferential statistics aims at making inferences and generalizations about the population data**.**

Descriptive Statistics

Descriptive statistics are a part of statistics that can be used to describe data. It is used to summarize the attributes of a sample in such a way that a pattern can be drawn from the group. It enables researchers to present data in a more meaningful way such that easy interpretations can be made. Descriptive statistics uses two tools to organize and describe data. These are given as follows:

- Measures of Central Tendency - These help to describe the central position of the data by using measures such as mean, median, and mode.
- Measures of Dispersion - These measures help to see how spread out the data is in a distribution with respect to a central point. Range, standard deviation, variance, quartiles, and absolute deviation are the measures of dispersion.

Need of descriptive statistics in data analysis:

1. Summarizing Data

Descriptive statistics allow us to condense large datasets into meaningful summaries, making it easier to understand the overall characteristics of the data.

Example:

- A company surveys 1,000 customers about their satisfaction with a product. The average satisfaction score (mean) can be calculated to provide a quick summary of customer sentiment. If the mean satisfaction score is 4.2 out of 5, this indicates that customers are generally satisfied.

2. Identifying Patterns and Trends

Descriptive statistics help in detecting patterns and trends within the data, which can inform further analysis and decision-making.

Example:

- In a dataset of monthly sales figures over the past year, a time series plot can reveal trends such as seasonality (higher sales in December) or growth (increasing sales month-over-month). A histogram of sales data might show a normal distribution, indicating consistent sales performance.

3. Facilitating Comparison

Descriptive statistics enable easy comparison between different groups or datasets, helping to identify differences and similarities.

Example:

- In a clinical trial, researchers compare the average reduction in blood pressure between two groups: those receiving a new medication and those receiving a placebo. If the mean reduction in blood pressure is 15 mmHg for the medication group and 5 mmHg for the placebo group, it clearly shows the effectiveness of the medication.

4. Detecting Outliers and Anomalies

Descriptive statistics are essential for identifying outliers and anomalies, which can indicate data entry errors, exceptional cases, or areas needing further investigation.

Example:

- A box plot of students' test scores can highlight outliers, such as a few students scoring significantly lower than the rest. Identifying these outliers can lead to further investigation into whether these scores were due to errors, misunderstandings, or other factors.

5. Informing Further Analysis

Descriptive statistics provide a necessary foundation for more complex analyses, guiding the choice of appropriate statistical methods and models.

Example:

- Before performing a regression analysis, a data analyst examines the descriptive statistics of the variables involved. If the standard deviation of the dependent variable is very high, it might indicate the need for data transformation or the inclusion of additional predictor variables.

6. Communicating Results

Descriptive statistics are crucial for presenting data insights in a clear and understandable manner to stakeholders who may not have a statistical background.

Example:

- In a business report, presenting the mean and standard deviation of customer satisfaction scores, along with visual aids like bar charts and pie charts, helps executives quickly grasp the key findings and make informed decisions.

7. Ensuring Data Quality

Descriptive statistics help in assessing the quality of the data, identifying inconsistencies, and ensuring the data is suitable for analysis.

Example:

- When analyzing survey data, an analyst might calculate the mean, median, and mode of responses to check for unusual patterns. If the mean age of respondents is 150 years, it indicates a data entry error that needs correction.

Inferential Statistics

Inferential statistics is a branch of statistics that is used to make inferences about the population by analyzing a sample. When the population data is very large it becomes difficult to use it. In such cases, certain samples are taken that are representative of the entire population. Inferential statistics draws conclusions regarding the population using these samples. Sampling strategies such as simple random sampling, cluster sampling, stratified sampling, and systematic sampling, need to be used in order to choose correct samples from the population. Some methodologies used in inferential statistics are as follows:

- Hypothesis Testing - This technique involves the use of hypothesis tests such as the z test, f test, t test, etc. to make inferences about the population data. It requires setting up the null hypothesis, alternative hypothesis, and testing the decision criteria.
- Regression Analysis - Such a technique is used to check the relationship between dependent and independent variables. The most commonly used type of regression is linear regression.

Important Concepts

Mean: The average of a dataset and is calculated by summing all the numbers in the dataset and then dividing by the count of numbers.

Median: The middle value separating the higher half from the lower half of the dataset.
For an odd number of observations, it is the middle value.
For an even number of observations, it is the average of the two middle values.

Mode: The most frequently occurring value in the dataset.here can be more than one mode if multiple values have the same highest frequency.

Example: For the dataset [1, 2, 2, 3, 4], the mode is 2. For the dataset [1, 2, 2, 3, 3, 4], the modes are 2 and 3.

Range: The difference between the maximum and minimum values.

Variance and Standard Deviation: Measures of the dispersion or spread of the data. Variance is the average of the squared differences from the mean, and the standard deviation is the square root of the variance.

Probability Distributions:

Uniform Distribution: All outcomes are equally likely.
Normal Distribution: The bell curve; characterized by its mean and standard deviation.
Binomial Distribution: The number of successes in a fixed number of independent Bernoulli trials.

Correlation

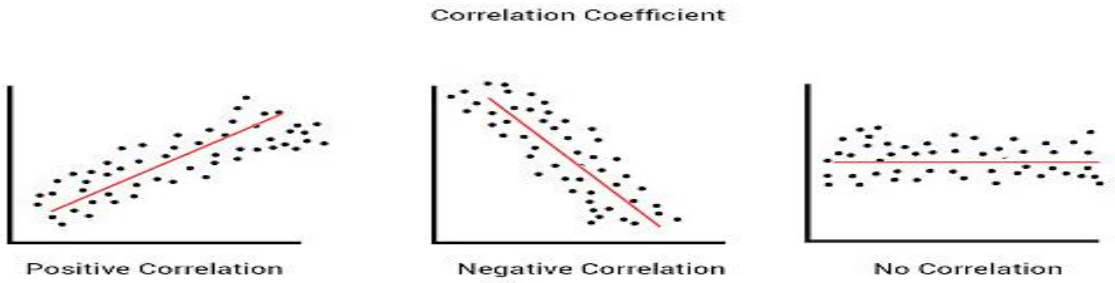
- A measure of the relationship between two variables, indicating how one variable changes with respect to another.
- Variance measures how a single variable deviates from its mean, Whereas Covariance measures how two variables vary together from their means.
- Correlation refers to the statistical relationship between the two entities.
- It measures the extent to which two variables are linearly related.
- For example, the height and weight of a person are related, and taller people tend to be heavier than shorter people.

There are three types of correlation:

Positive Correlation: A positive correlation means that this linear relationship is positive, and the two variables increase or decrease in the same direction.

Negative Correlation: A negative correlation is just the opposite. The relationship line has a negative slope, and the variables change in opposite directions, i.e., one variable decreases while the other increases.

No Correlation: No correlation simply means that the variables behave very differently and thus, have no linear relationship



Correlation and causation

Correlation depicts the degree of association between 2 random variables. In data analysis it is often used to determine the amount to which they related to one another.
A relationship where one event causes another event to occur, which is often more challenging to establish than correlation.
Causation between A & B implies that A&B have a cause & effect relationship with one another i,e A depend on B or vice versa

Example: Mobile phone- low battery leads to disconnect the video call and shut down the system, both are causation.

"**Correlation is Not Causation"**

The statement "correlation is not causation" means that just because two variables are correlated, it does not mean that one variable causes the other to change. There are three main reasons for this:

1. Coincidence: The correlation might be due to random chance.
2. Third Variable Problem: Another variable (a confounding variable) might be influencing both variables, creating a false impression of a direct relationship.

3. Reverse Causality: It might be that the supposed effect is actually the cause.

Example

Consider a scenario where there is a high correlation between ice cream sales and drowning incidents. Based on correlation alone, one might mistakenly conclude that ice cream sales cause drowning. However, the actual explanation involves a third variable: temperature.

- During hot weather, more people buy ice cream to cool off.
- During the same hot weather, more people go swimming to cool off.
- As more people swim, the likelihood of drowning incidents increases.

In this case, the hot weather (the third variable) causes both the increase in ice cream sales and the increase in drowning incidents. Therefore, the correlation between ice cream sales and drowning does not imply that one causes the other.

Quantile:
A generalization of the median is the quantile
The concept of quantiles generalizes the median.

- Quantiles are introduced to describe the values that divide a dataset into intervals with equal probabilities.
- Quantiles are points taken at regular intervals from the cumulative distribution function (CDF) of a random variable. They are used to divide the range of a dataset into contiguous intervals with equal probabilities.
- Median: The 50th percentile, which divides the dataset into two equal parts whereas Quartiles Divide the dataset into four equal parts.
  The first quartile (Q1) is the 25th percentile, the second quartile (Q2) is the median (50th percentile), and the third quartile (Q3) is the 75th percentile.Percentiles: Divide the dataset into 100 equal parts.

CODE:

```
import numpy as np data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
quantiles = np.quantile(data, [0.25, 0.5, 0.75])
 print("25th percentile:", quantiles[0])
 print("50th percentile (median):", quantiles[1])
print("75th percentile:", quantiles[2])
```

Dispersion

Dispersion in statistics refers to the extent to which a distribution is stretched or squeezed. Commonly used measures of dispersion include the range, variance, standard deviation, and interquartile range (IQR). These measures provide insights into the variability or spread of a dataset.. It indicates how much the values differ from the average (mean) value. High dispersion means the data points are spread out over a wide range of values, while low dispersion indicates that the data points are closely clustered around the mean.

1. Range: The difference between the maximum and minimum values in a dataset.
2. Variance: The average of the squared differences from the mean.
3. Standard Deviation: The square root of the variance, representing the average distance from the mean.
4. Interquartile Range (IQR): The difference between the 75th percentile (Q3) and the 25th percentile (Q1).

Code:

```
import numpy as np

# Sample data

data = [12, 15, 14, 10, 18, 20, 25, 30, 22, 23]

# Range

data_range = np.max(data) - np.min(data)

# Variance

variance = np.var(data)

# Standard Deviation

std_deviation = np.std(data)

# Interquartile Range (IQR)

Q1 = np.percentile(data, 25)

Q3 = np.percentile(data, 75)

IQR = Q3 - Q1

# Output the results

print(f"Range: {data_range}")

print(f"Variance: {variance}")

print(f"Standard Deviation: {std_deviation}")

print(f"Interquartile Range (IQR): {IQR}")
```

Variance and Covariance:

1. **Variance**:
   - Variance measures the dispersion of a single variable.
   - It calculates the average of the squared differences between each data point and the mean of the data set.
   - Variance is always non-negative.
   - It provides an idea of how much the values of a single variable differ from the mean value of that variable.
2. **Covariance**:
   - Covariance measures the degree to which two variables change together.
   - It calculates the average of the product of the deviations of each pair of data points from their respective means.
   - Covariance can be positive, negative, or zero.
   - It provides an idea of the direction of the linear relationship between two variables (i.e., whether the variables tend to increase or decrease together).

CODE:

```
import numpy as np

# Sample data

x = [12, 15, 14, 10, 18, 20, 25, 30, 22, 23]

y = [22, 25, 24, 20, 28, 30, 35, 40, 32, 33]

# Calculating covariance

covariance_matrix = np.cov(x, y, bias=True)

covariance = covariance_matrix[0, 1]

# Output the results

print(f"Covariance between x and y: {covariance}")
```

Simpson'sParadox

Simpson's paradox occurs when groups of data show one particular trend, but this trend is reversed when the groups are combined together. Understanding and identifying this paradox is important for correctly interpreting data.

Consider n groups of data such that I has A_i trails and $0 \leq a_i \leq A_i$ " Successes" . Similarly consider an analogous n groups of data such that group I has B_i trails and $0 \leq b_i \leq B_i$ " Successes". Then, Simpsons paradox occurs if

$$\frac{a_i}{A_i} < \frac{b_i}{B_i} \; for \; i = 1, 2, \dots, n \; but \; \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} A_i} \geq \frac{\sum_{i=1}^{n} b_i}{\sum_{i=1}^{n} B_i}$$

Consider an example

Let's consider an example involving a case study of two students A and B assigned to solve some problems for two days. We want to determine which treatment is more effective.

Data:

Day	Student A	Success Rate	Student B	Success Rate
Saturday	7/8	87.5%	2/2	100%
Sunday	1/2	50%	5/8	62.5%
Total	8/10	80%	7/10	70%

Thus Simpson's paradox is a phenomenon in probability and statistics in which a trend appears in several groups of data but disappears or reverses when the groups are combined.

Probability

Probability Theory is the backbone of many Data Science concepts, such as Inferential Statistics, Machine Learning, Deep Learning, etc. Probability is an estimation of how likely a certain event or outcome will occur. It is typically expressed as a number between 0 and 1.Probability can be calculated by dividing the number of favorable outcomes by the total number of outcomes of an event.

Probability Formula
P(A)=Number of favorable outcomes to A/Total number of possible outcomes

For example, if there is an 80% chance of rain tomorrow, then the probability of it raining tomorrow is 0.8.

Dependence and Independence

If two events E and F are dependent if knowing something about whether E happens gives us information about whether F happens or vice versa.

If we flip a fair coin twice, knowing whether the first flip is Heads gives us no information about whether the second flip is Heads. These events are independent.

Independent events are events that are not affected by the occurrence of other events.
The formula for the Independent Events is,

P(A and B) = P(A)×P(B)

Dependent events are events that are affected by the occurrence of other events.

Bayes' theorem

Bayes' theorem is a powerful tool in data science for updating the probability of a hypothesis based on new evidence. It's widely used in various applications such as spam filtering, medical diagnosis, and machine learning.

Bayes' Theorem Formula

Bayes' theorem relates the conditional and marginal probabilities of random events. The formula is:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where:

- P(A|B) is the posterior probability: the probability of event A occurring given that B is true.
- P(B|A)is the likelihood: the probability of event B occurring given that A is true.
- P(A)is the prior probability: the initial probability of event A before any evidence is considered.
- P(B) is the marginal probability: the total probability of event B occurring.

Probability distribution

A probability distribution describes how the values of a random variable are distributed. It provides the probabilities of occurrence of different possible outcomes in an experiment. Probability distributions can be categorized into two main types: discrete and continuous.

Discrete Probability Distribution:

This type applies to discrete random variables, which are variables that take on a finite or countable number of distinct values.

Example: The number of heads in 10 coin flips.

Expected Value and Variance:
Expected Value (Mean): For a discrete random variable X:

$$E(X) = i\sum x_i \, P(x_i)$$

Variance: Measures the spread of the random variable's values:

$$\mathbf{Var(X) = E((X - E(X))^2)}$$

Continuous Distribution

A continuous distribution is a type of probability distribution in which the variable can take an infinite number of values within a given range. It describes the probabilities of the possible values of a continuous random variable. Unlike discrete variables, which have specific, countable outcomes, continuous variables can take on any value within a given range.

In continuous distribution , there are infinitely many numbers between 0 and 1Instead of assigning probabilities to individual points, we use a Probability Density Function (PDF).PDF is that it must integrate to 1 over the entire range of possible values, ensuring the total probability is 1.

Examples:

1. **Height of Adults:** The height of adults can be measured to any level of precision and is typically described by a continuous distribution. For instance, someone might be 170.2 cm tall, while another person might be 170.25 cm tall.
2. **Temperature:** Temperature readings are continuous as they can take on any value within a range, such as 23.5°C or 23.56°C..

Normal Distribution

The normal distribution, also known as the Gaussian distribution, is a specific type of continuous distribution. It is characterized by its bell-shaped curve, which is symmetric around the mean. The properties of a normal distribution include:

- The mean, median, and mode are all equal.
- It is fully described by two parameters: the mean (μ) and the standard deviation (σ).
- Approximately 68% of the data falls within one standard deviation of the mean, 95% within two standard deviations, and 99.7% within three standard deviations (empirical rule).

Example: The distribution of heights of adult men in a specific population is often modeled as a normal distribution with a mean of around 175 cm and a standard deviation of about 7 cm. This means most men's heights will be around 175 cm, with fewer men being significantly shorter or taller.

The normal distribution is the king of distributions: determined by two parameters - its mean μ (mu) and its standard deviation σ (sigma).

Code:

```
def normal_cdf(x, mu=0,sigma=1):
 return (1 + math.erf((x - mu) / math.sqrt(2) / sigma))
 xs = [x / 10.0 for x in range(-50, 50)]

plt.plot(xs,[normal_cdf(x,sigma=1) for x in xs],'-',label='mu=0,sigma=1')
plt.plot(xs,[normal_cdf(x,sigma=2) for x in xs],'--',label='mu=0,sigma=2')
plt.plot(xs,[normal_cdf(x,sigma=0.5) for x in xs],':',label='mu=0,sigma=0.5')
plt.plot(xs,[normal_cdf(x,mu=-1) for x in xs],'-.',label='mu=-1,sigma=1')
plt.legend(loc=4) # bottom right plt.title("Various Normal cdfs")
 plt.show()
```

Common Distributions

Binomial Distribution: Describes the number of successes in a fixed number of independent Bernoulli trials.
Parameters: n (number of trials) and p (probability of success).

Conditional Probability

Conditional probability refers to the probability of an event occurring given that another event has already occurred. The conditional probability of event A given that event B has occurred is denoted as P(A/B) and is defined as:

$$P(A/B) = P(A \text{ and } B) / P(B)$$

This formula assumes that $P(B) \neq 0$. It is used to update our probability estimates based on new information.

Example: Medical Test

Consider a scenario involving a medical test for a particular disease. Consider the following information:
The probability of having the disease (D) is P (D)=0.01.

The probability of testing positive (T) given that one has the disease is

$$P(T/D)=0.99.$$

The probability of testing positive given that one does not have the disease is $P(T/\neg D)=0.05$.
We want to find the probability that a person has the disease given that they tested positive (P(D/T)).

Applying Bayes' Theorem

Bayes' Theorem provides a way to update our beliefs based on new evidence. The theorem states:

$$P(D/T) = P(T)P(T/D) \cdot P(D)$$

We need to calculate P (T), the total probability of testing positive. This can be done using the law of total probability:

$$P(T) = P(T/D) \cdot P(D) + P(T/\neg D) \cdot P(\neg D)$$

Where: $P(\neg D) = 1 - P(D)$

Substituting the given values:

$P(T) = (0.99 \cdot 0.01) + (0.05 \cdot 0.99)$

Calculating this:

$P(T) = 0.0099 + 0.0495 = 0.0594$

Now we can use Bayes' Theorem to find $P(D/T)$:

$P(D/T) = P(T/D) \cdot P(D) / P(T) = 0.99 \cdot 0.01 / 0.0594$

$P(D/T) = 0.0099 / 0.0594 \approx 0.1667$

Interpretation Despite the high sensitivity $P(T/D) = 0.99$ of the test, the probability of actually having the disease given a positive test result is only about 16.67%. This example illustrates the importance of considering the base rate (prevalence) of the disease in the population when interpreting test results. In this case, the low prevalence of the disease significantly affects the conditional probability.

Random Variables and Probability Distributions

1. Random Variables:

Definition: A random variable is a variable whose value is subject to variations due to randomness. Types: Discrete and continuous random variables. A random variable is a variable whose possible values have an associated probability distribution.

Example:
A very simple random variable equals 1 if a coin flip turns up heads and 0 if the flip turns up tails.
A more complicated one might measure the number of heads observed when flipping a coin 10 times or a value picked from range (10) where each number is equally likely. The associated distribution gives the probabilities that the variable realizes each of its possible values. The coin flip variable equals 0 with probability0.5 and 1 with probability 0.5. The range (10) variable has a distribution that assigns probability 0.1 to each of the numbers from 0 to 9.

The expected value of a random variable, which is the average of its values weighted by their probabilities. The coin flip variable has an expected value of

1/2 (= 0 * 1/2 + 1 * 1/2)

and the range(10) variable has an expected value of 4.5.Random variables can be conditioned on events just as other events .

Central Limit Theorem (CLT)

The Central Limit Theorem is a fundamental principle in statistics that describes the characteristics of the sampling distribution of the sample mean. It states that, regardless of the original distribution of the data, the distribution of the sample means approaches a normal distribution as the sample size becomes larger, provided the samples are independent and identically distributed.

Example: Suppose you want to estimate the average height of all students in a large university. If you take multiple random samples of students and calculate the average height for each sample, the distribution of these sample means will approximate a normal distribution as the sample size increases, regardless of the original distribution of student heights.

SUBJECT: Data Science and Its Applications (21AD62)

Module-2: Hypothesis and Inference

Syllabus: Statistical Hypothesis Testing, Example: Flipping a Coin, p-Values, Confidence Intervals, p-Hacking, Example: Running an A/B Test, Bayesian Inference, Gradient Descent, The Idea Behind Gradient Descent Estimating the Gradient, Using the Gradient, Choosing the Right Step Size, Using Gradient Descent to Fit Models, Minibatch and Stochastic Gradient Descent, Getting Data, stdin and stdout, Reading Files, Scraping the Web, Using AP!s, Example: Using the Twitter AP!s, Working with Data, Exploring Your Data, Using Named Tuples, Data classes, Cleaning and Munging, Manipulating Data, Rescaling, An Aside: tqdm, Dimensionality Reduction.

Statistical Hypothesis Testing

Statistical hypothesis testing is a method used to make decisions or inferences about a population based on a sample of data. It is used to estimate the relationship between 2 statistical variables. Calculating a test statistic from the sample data, and making a decision about which hypothesis is more probable based on this statistic.

Example: A doctor believes that 3D (Diet, Dose, and Discipline) is 90% effective for diabetic patients.

Null Hypothesis and Alternate Hypothesis

The Null Hypothesis is the assumption that the event will not occur. A null hypothesis has no bearing on the study's outcome unless it is rejected. It is denoted by H_0.

The Alternate Hypothesis is the logical opposite of the null hypothesis. The acceptance of the alternative hypothesis follows the rejection of the null hypothesis. It is denoted H_1 .
Let's understand this with an example.
A sanitizer manufacturer claims that its product kills 95 percent of germs on average. To put this company's claim to the test, create a null and alternate hypothesis.
H0 (Null Hypothesis): Average = 95%.
Alternative Hypothesis (H1): The average is less than 95%.

Suppose you have a coin and you want to test whether it is fair, i.e., whether it lands heads and tails with equal probability.

Steps for Hypothesis Testing

Consider

1. Null Hypothesis (H_0): The coin is fair. That is, P(Heads) = 0.5
2. Alternative Hypothesis ($H_{1):}$ The coin is not fair. That is, P(Heads) ≠ 0.5.

Choose a Significance Level (α\alphaα)

The significance level (α) is the probability of rejecting the null hypothesis when it is actually true. Common choices for α are 0.05, 0.01, and 0.10. Let's use α=0.05.

Flip the coin a certain number of times and record the results. For example, flip the coin 100 times and count the number of heads.

Suppose you get 60 heads out of 100 flips.

Since we are dealing with proportions, we can use a z-test for our hypothesis testing.

The sample proportion (p) = 60/100 = 0.6

The population proportion under the null hypothesis is 0.5.

The standard error (SE) is calculated as:

$$SE = \sqrt{\frac{p_0(1-p_0)}{n}} = \sqrt{\frac{0.5 \times 0.5}{100}} = \sqrt{\frac{0.25}{100}} = \sqrt{0.0025} = 0.05$$

The z-score is calculated as:

$$z = \frac{p - p_0}{SE} = \frac{0.6 - 0.5}{0.05} = \frac{0.1}{0.05} = 2$$

3. Determine the Critical Value and Decision Rule

   For a two-tailed test with α=0.05, the critical z-values are ±1.96. If the calculated z-value is be±1.96, we reject the null hypothesis.

   - Calculated z-value = 2
   - Critical z-value = ±1.96

   Since 2 > 1.96, we reject the null hypothesis and the coin is not fair.

P value:

The p-value is the probability of obtaining test results at least as extreme as the observed data, assuming that the null hypothesis (H_0) is true. It provides a measure of the evidence against the null hypothesis:

- A low p-value indicates strong evidence against H_0, suggesting that H_0 may not be true.
- A high p-value indicates weak evidence against H_0, suggesting that there is insufficient evidence to reject H_0.

p-value≤α Reject the null hypothesis.

p-value>α Fail to reject the null hypothesis. The result is not statistically significant.

Here, α is the chosen significance level, commonly set at 0.05.

Importance of Power and Significance:

Statistical hypothesis testing is a crucial aspect of inferential statistics, enabling researchers to make decisions about populations based on sample data. Two important concepts in hypothesis testing are **power** and **significance**:

1. **Significance Level (α)**: This is the threshold for rejecting the null hypothesis. Commonly set at 0.05, it represents a 5% risk of concluding that a difference exists when there is no actual difference (Type I error).

   - Determines the likelihood of a Type I error.
   - A lower α reduces the probability of a false positive but may increase the chance of a Type II error.
   - Ensures the results are not due to random chance.

2. **Power (1 - β)**: This is the probability of correctly rejecting the null hypothesis when it is false. Power is influenced by the significance level, sample size, effect size, and variance. Higher power reduces the risk of a Type II error (failing to detect an actual effect).

    - Indicates the sensitivity of the test.
    - High power means a greater chance of detecting a true effect.
    - Important for ensuring that meaningful effects are not overlooked.

Confidence Intervals

While testing hypotheses about the value of the heads probability p, which is a parameter of the unknown "heads" distribution. When this is the case, a third approach is to construct a confidence interval around the observed value of the parameter.

For example, To estimate the probability of the unfair coin by looking at the average value of the Bernoulli variables corresponding to each flip — 1 if heads, 0 if tails. It is observed that 525 heads out of 1,000 flips, and then estimate p equals 0.525.

How confident can we be about this estimate? if the exact value of p, the central limit theorem tells us that the average of those Bernoulli variables should be approximately normal, with mean p and standard deviation:

```
math.sqrt(p * (1 - p) / 1000)
```

Here we don't know p, so instead we use our estimate:

```
p_hat = 525 / 1000
mu = p_hat
sigma = math.sqrt(p_hat * (1 - p_hat) / 1000) # 0.0158
```

This is not entirely justified. Using the normal approximation, It is conclude that we are "95% confident" that the following interval contains the true parameter p:

normal_two_sided_bounds(0.95, mu, sigma) # [0.4940, 0.5560]

In particular, we do not conclude that the coin is unfair, since 0.5 falls within our confidence interval.

If instead we'd seen 540 heads, then we'd have:

```
p_hat = 540 / 1000
mu = p_hat
sigma = math.sqrt(p_hat * (1 - p_hat) / 1000) # 0.0158
normal_two_sided_bounds(0.95, mu, sigma) # [0.5091, 0.5709]
```

Here, "fair coin" doesn't lie in the confidence interval. (The "fair coin" hypothesis doesn't pass a test that you'd expect it to pass 95% of the time if it were true.)

P-hacking

P-hacking refers to the practice of manipulating data or analysis until statistically significant results are obtained. This can include:

- Running many different tests and only reporting those that yield significant results.
- Removing outliers that do not support the desired outcome.
- Stopping data collection once significant results are obtained.

P-hacking undermines the validity of scientific research by presenting misleading results as statistically significant. This can lead to false conclusions and reduce the replicability of studies.

Common Practices of P-Hacking

Selective Reporting:

Reporting only the experiments or results that yield significant outcomes. Ignoring or underreporting no significant findings.

Data Dredging:

Conducting multiple statistical tests and only reporting those with significant p-values. Exploring various data mining techniques without pre-specified hypotheses.

Manipulating Data:

Excluding data points or outliers selectively to achieve significance. Stopping data collection once significant results are obtained, rather than following a pre-specified data collection plan.

Consequences of P-Hacking

Misleading Conclusions:

Results may be presented as discoveries when they are actually artifacts of p-hacking Can lead to incorrect theories and hinder scientific progress.

Example of P-Hacking

A researcher conducts 20 different tests on a dataset and finds one test with a p-value < 0.05.

Reports this significant result without mentioning the other 19 tests.

Running A/B test

One of the advertisers has developed a new energy drink targeted at data scientists, and the VP of Advertisements wants help choosing between advertisement A ("tastes great!") and advertisement B ("less bias!"). Being a scientist, it has be decide to run an experiment by randomly showing site visitors one of the two advertisements and tracking how many people click on each one.

If 990 out of 1,000 A-viewers click their ad while only 10 out of 1,000 B-viewers click their ad, It can be pretty confident that A is the better ad. But

Let's say that $\mathbf{N_A}$ people see ad A, and that $\mathbf{n_A}$ of them click it. Let's take each ad view as a Bernoulli trial where is the probability $\mathbf{P_A}$ that someone clicks ad A.

Then $\mathbf{n_A}$ / $\mathbf{N_A}$ approximately a normal random variable with mean $\mathbf{P_A}$and standard deviation. $\sigma_A = \sqrt{p_A(1-p_A)/N_A}$.:

Similarly $\mathbf{n_B}$ / $\mathbf{N_B}$ approximately a normal random variable with mean $\mathbf{P_B}$ and standard deviation.

$\sigma_B = \sqrt{p_B(1-p_B)/N_B}$:

If two normals are independent), then their difference should also be normal with mean PA –PB and standard deviation . $\sqrt{\sigma_A^2 + \sigma_B^2}$.

This means the null hypothesis that PA and PB are the same using the statistic:

```
def a_b_test_statistic(N_A, n_A, N_B, n_B):
p_A, sigma_A = estimated_parameters(N_A, n_A)
p_B, sigma_B = estimated_parameters(N_B, n_B)
 return (p_B - p_A) / math.sqrt(sigma_A ** 2 + sigma_B ** 2)
```

which should approximately be a standard normal.

For example, if "tastes great" gets 200 clicks out of 1,000 views and "less bias" gets 180 clicks out of 1,000 views, the statistic equals:

z = a_b_test_statistic(1000, 200, 1000, 180) # -1.14

The probability of seeing such a large difference if the means were actually equal would be:

two_sided_p_value(z) # 0.254

which is large enough to conclude there's much of a difference. On the other hand, if "less bias" only got 150 clicks, we'd have

z = a_b_test_statistic(1000, 200, 1000, 150) # -2.94
two_sided_p_value(z) # 0.003

which means there's only a 0.003 probability you'd see such a large difference if the ads were equally effective.

Bayesian Inference

Bayesian inference involves updating the beliefs about unknown parameters based on observed data. This approach treats the parameters themselves as random variables and uses probability distributions to describe our beliefs about these parameters.

An alternative approach to inference involves treating the unknown parameters themselves as random variables. The analyst starts with a prior distribution for the parameters and then uses the observed data and Bayes's Theorem to get an updated posterior distribution for the parameters. Rather than making probability judgments about the tests, the probability judgments about the parameters themselves. For example, when the unknown parameter is a probability (as in our coin-flipping example), it is often use a prior from the Beta distribution, which puts all its probability between 0 and 1:

Code:

```
import math

def B(alpha, beta):

    """A normalizing constant so that the total probability is 1"""

    return math.gamma(alpha) * math.gamma(beta) / math.gamma(alpha + beta)

def beta_pdf(x, alpha, beta):

    if x < 0 or x > 1:  # no weight outside of [0, 1]

        return 0

    return x ** (alpha - 1) * (1 - x) ** (beta - 1) / B(alpha, beta)
```

Generally speaking, this distribution centers its weight at:

alpha / (alpha + beta)

and the larger alpha and beta are, the "tighter" the distribution is.

For example, if alpha and beta are both 1, it's just the uniform distribution (centered at 0.5, very dispersed). If alpha is much larger than beta, most of the weight is near 1. And if alpha is much smaller than beta, most of the weight is near zero.

Figure 7-1 shows several different Beta distributions.

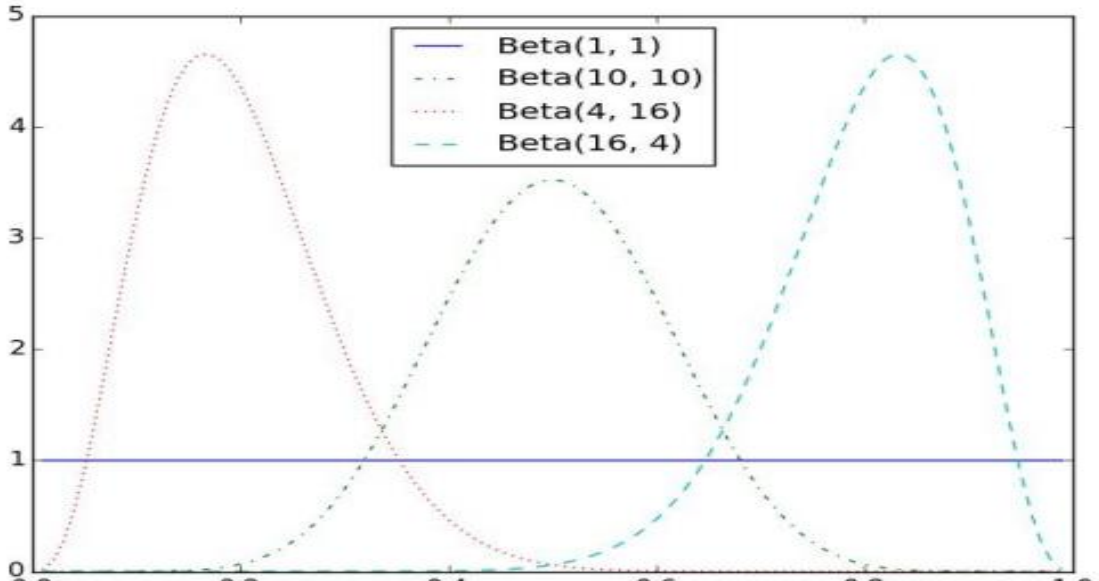


Figure 7-1. Example Beta distributions

So let's us assume a prior distribution on p. Maybe it will take a stand on whether the coin is fair, and choose alpha and beta to both equal 1. Or maybe it is strong belief that it lands heads 55% of the time, and alpha equals 55, beta equals 45. Then flip our coin a bunch of times and see h heads and t tails. Bayes's Theorem tells us that the posterior distribution for p is again a Beta distribution but with parameters alpha + h and beta + t.

Let's flip the coin 10 times and see only 3 heads. If it started with the uniform prior posterior distribution would be a Beta(4, 8), centered around 0.33. Since it is considered all probabilities equally likely, your best guess is something pretty close to the observed probability. If it is started with a Beta(20, 20) your posterior distribution would be a Beta(23, 27), centered around 0.46, indicating a revised belief that maybe the coin is slightly biased toward tails. And if you started with a Beta(30, 10) ,your posterior distribution would be a Beta(33, 17), centered around 0.66. In that case you'd still believe in a heads bias, but less strongly than you did initially. These three different posteriors are plotted in

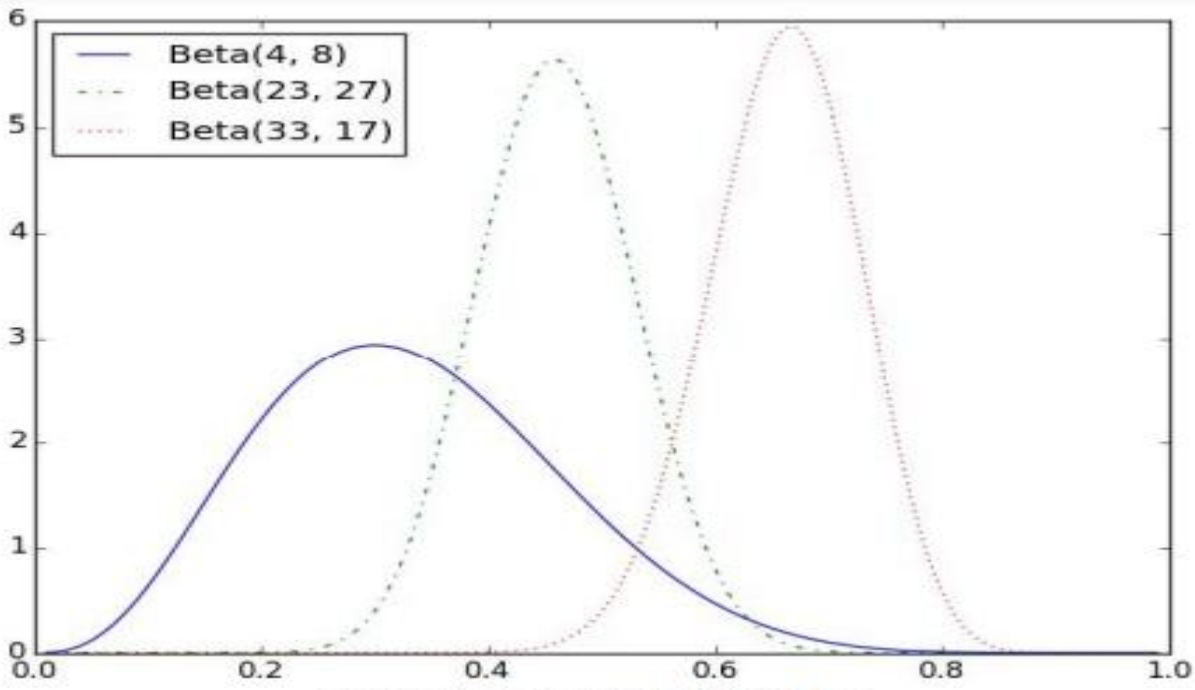


Figure 7-2. Posteriors arising from different priors

If the coin is flipped more and more times, the prior would matter less and less until eventually it have the same posterior distribution no matter which prior it started with. For example, no matter how biased you initially thought the coin was, it would be hard to maintain that belief after seeing 1,000 heads out of 2,000 flips (unless you are a lunatic who picks something like a Beta(1000000,1) prior). What's interesting is that this allows us to make probability statements about hypotheses: "Based on the prior and the observed data, there is only a 5% likelihood the coin's heads probability is between 49% and 51%." This is philosophically very different from a statement like "if the coin were fair we would expect to observe data so extreme only 5% of the time."

Gradient descent

Gradient descent is a fundamental optimization algorithm used in machine learning to minimize a cost function . Gradient descent aims to find the parameters of a model that minimize the cost function, which measures how well the model fits the data.

The steps are as follows:

1. Start with random initial values for the parameters (weights).
2. Calculate the gradient of the cost function with respect to each parameter. The gradient is a vector of partial derivatives, indicating the direction and rate of the steepest increase the cost function.
3. Adjust the parameters in the opposite direction of the gradient by a small amount, which is determined by the learning rate.

This step is repeated iteratively**:**

$$\theta_i = \theta_i - \alpha \ \partial J(\theta)/\partial \theta_i$$

where,

θ_i - i^{th} parameter, α is the learning rate, and
$J(\theta)/\partial\theta_i$ - is the cost function.

4. Repeat steps 2 and 3 until the change in the cost function is smaller than a predefined threshold or a maximum number of iterations is reached.

Gradient Descent for Parameterized Models

When fitting parameterized models, the cost function depends on the difference between the predicted and actual values. For example, in linear regression, the cost function J(θ) is typically the mean squared error:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

where:

- m is the number of training examples.
- hθ(x(i)) is the predicted value for the i^{th} training example.
- y(i) is the actual value for the i^{th} training example.

The gradient descent algorithm for linear regression would involve computing the partial derivatives of J(θ) with respect to each parameter θj:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

If the learning rate is too high, the algorithm might overshoot the minimum. If the learning rate is too low, the algorithm might take too long to converge. Adjusting the learning rate and ensuring proper data scaling are essential for effective gradient descent optimization.

Gradient descent is a powerful optimization algorithm used to minimize cost functions in machine learning and deep learning. The three main variants of gradient descent are:

1. **Batch Gradient Descent**: Uses the entire dataset to compute the gradient of the cost function.
2. **Stochastic Gradient Descent (SGD)**: Uses a single data point to compute the gradient.
3. **Mini-batch Gradient Descent**: Uses a small random subset (mini-batch) of the dataset to compute the gradient.

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent updates the model parameters using only one data point at a time. This introduces some randomness in the gradient calculation but can lead to faster convergence on large datasets.

Steps of Stochastic Gradient Descent:

1. Start with initial guesses for the parameters.
2. For each data point $(x^{(i)}, y^{(i)})$
    - Compute the prediction using the current parameters.
    - Calculate the cost (loss) for this prediction.
    - Compute the gradient of the cost with respect to each parameter using this single data point.
    - Update the parameters in the opposite direction of the gradient.
3. Continue iterating over the data points for a specified number of epochs or until convergence.

Advantages of SGD:

- Faster iterations, especially on large datasets.
- Can escape local minima due to its noisy updates.

Disadvantages of SGD:

- The parameter updates can be noisy, leading to fluctuations in the cost function.
- Convergence can be slower and more erratic compared to batch gradient descent.

Mini-batch Gradient Descent

Mini-batch Gradient Descent is a compromise between batch gradient descent and stochastic gradient descent. It splits the dataset into small random subsets called mini-batches and computes the gradient using these mini-batches.

Steps of Mini-batch Gradient Descent:

1. Start with initial guesses for the parameters.
2. For each mini-batch B:
    1. Compute the predictions for all data points in the mini-batch.
    2. Calculate the average cost (loss) for these predictions.
    3. Compute the average gradient of the cost with respect to each parameter using the mini-batch.
    4. Update the parameters in the opposite direction of the average gradient.
3. Continue iterating over mini-batches for a specified number of epochs or until convergence.

Advantages of Mini-batch Gradient Descent:

- Faster and more efficient than batch gradient descent, especially on large datasets.
- Reduces the noise in parameter updates compared to SGD, leading to more stable convergence.
- Allows for parallel computation, leveraging modern hardware like GPUs.

Disadvantages of Mini-batch Gradient Descent:

- Requires tuning of the mini-batch size.
- Still requires more memory compared to SGD, as it needs to store the mini-batch in memory.

CODE:

```
import numpy as np

# Generate some sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Initialize parameters
theta = np.random.randn(2, 1)
learning_rate = 0.01
n_iterations = 1000
batch_size = 20
```

```
# Add bias term to X
X_b = np.c_[np.ones((100, 1)), X]

# Gradient descent
for epoch in range(n_iterations):
    shuffled_indices = np.random.permutation(len(X_b))
    X_b_shuffled = X_b[shuffled_indices]
    y_shuffled = y[shuffled_indices]

    for i in range(0, len(X_b), batch_size):
        xi = X_b_shuffled[i:i + batch_size]
        yi = y_shuffled[i:i + batch_size]
        gradients = 2 / batch_size * xi.T.dot(xi.dot(theta) - yi)
        theta = theta - learning_rate * gradients

print(f"Parameters: {theta}")
```

Getting Data

stdin and stdout

In Python, stdin and stdout are standard streams used for input and output operations. They are part of the sys module, which provides access to some variables used or maintained by the Python interpreter and functions that interact with the interpreter.

stdin (Standard Input)

- stdin represents the standard input stream, which is typically used for receiving input from the user or from other programs. By default, stdin reads input from the keyboard.
- It is commonly used with functions like input() to receive input from the user interactively.
- In scripts or programs that run in a command-line environment, stdin can also be redirected to read input from files or other programs' output.

```
import sys
name = sys.stdin.readline()
print(name)
num = sys.stdin.readline(2)
print(num)
```

stdout (Standard Output)

- stdout represents the standard output stream, which is typically used for writing output to the console or terminal.
- It is commonly used with functions like print() to display information to the user.
- Similar to stdin, stdout can also be redirected to write output to files or other programs' input.

```
# egrep.py
import sys, re
# sys.argv is the list of command-line arguments
# sys.argv[0] is the name of the program itself
# sys.argv[1] will be the regex specified at the command line
regex = sys.argv[1]
# for every line passed into the script
for line in sys.stdin:
```

```
# if it matches the regex, write it to stdout
if re.search(regex, line):
sys.stdout.write(line)
```

Reading Files

- Reading files in Python is a common task, and Python provides several ways to accomplish it.

Using open() Function:

- The open() function is used to open a file.
- It returns a file object, which can then be used to read, write, or manipulate the file's contents.

Example

```
with open('file.txt', 'r') as f:
# Read the entire file
content = f.read()
 print(content)
with open('image.jpg', 'rb') as f:
 data = f.read()
```

file_for_reading = open('reading_file.txt', 'r')

- This line opens a file named 'reading_file.txt' in read-only mode ('r'). It creates a file object file_for_reading that can be used to read the contents of the file.

file_for_writing = open('writing_file.txt', 'w')

- This line opens a file named 'writing_file.txt' in write mode ('w'). If the file already exists, it will be overwritten. If the file does not exist, a new file will be created. It creates a file object file_for_writing that can be used to write data to the file.

file_for_appending = open('appending_file.txt', 'a')

- This line opens a file named 'appending_file.txt' in append mode ('a'). If the file already exists, new data will be appended to the end of the file. If the file does not exist, a new file will be created. It creates a file object file_for_appending that can be used to append data to the file.

file_for_writing.close()

- This line closes the file file_for_writing after writing to it. It's important to close files when you're done with them to free up system resources and ensure that all data is written to the file.

Delimited files

More frequently we work with files with lots of data on each line.These files are very often either comma-separated or tab-separated. Each line has several fields, with a comma (or a tab) indicating where one field ends and the next field startThis starts to get complicated when you have fields with commas and tabs and newlines in them. Instead, we should use Python's csv module.

For example, if we had a tab-delimited file of stock prices:

```
6/20/2014       AAPL  90.91
6/20/2014       MSFT  41.68
6/20/2014       FB      64.5
6/19/2014       AAPL  91.86
 6/19/2014       MSFT          41.51
6/19/2014       FB      64.34
```

```
import csv

with open('tab_delimited_stock_prices.txt', 'rb') as f:
    reader = csv.reader(f, delimiter='\t')
    for row in reader:
        date = row[0]
        symbol = row[1]
        closing_price = float(row[2])
        process(date, symbol, closing_price)
```

If file has headers:
date:symbol:closing_price
6/20/2014:AAPL:90.91
6/20/2014:MSFT:41.68
6/20/2014:FB:64

```
with open('colon_delimited_stock_prices.txt', 'rb') as f:
    reader = csv.DictReader(f, delimiter=':')
    for row in reader:
        date = row["date"]
        symbol = row["symbol"]
        closing_price = float(row["closing_price"])
        process(date, symbol, closing_price)
```

Assume we have a CSV file named example_data.csv with the following content:

name,age,height(cm),weight(kg),city
Alice,29,165,68,New York
Bob,,175,85,San Francisco
Charlie,35,NaN,78,Los Angeles
David,NaN,180,90,New York
Eve,22,NaN,55,NaN

Steps for Cleaning and Munging

1. Load Data:

import pandas as pd

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

2. Inspect Data:

```
# Display the first few rows of the data
print(data.head())

# Display summary statistics
print(data.describe())

# Display information about data types and missing values
print(data.info())
```

3. Handle Missing Values:

```
# Fill missing age values with the mean age
data['age'].fillna(data['age'].mean(), inplace=True)

# Drop rows where height or weight is missing
data.dropna(subset=['height(cm)', 'weight(kg)'], inplace=True)

# Fill missing city values with 'Unknown';
data['city'].fillna('Unknown', inplace=True)
print(data)
```

4. Convert Data Types:

```
# Ensure the 'age'; column is of integer type

data['age'] = data['age'].astype(int)
print(data.info())
```

5. Standardize Formats:

```
# Standardize city names to title case
data['city'] = data['city'].str.title()
print(data)
```

6. Remove Duplicates:

```
# Remove duplicate rows, if any
data.drop_duplicates(inplace=True)
print(data)
```

7. Transform Data:

```
# Create a new column for BMI (Body Mass Index)
data['BMI'] = data['weight(kg)'] / (data['height(cm)'] / 100) ** 2
print(data)
```

Scraping the Web

Web scraping is the automated process of extracting data from websites. It involves fetching web pages, parsing the content, and extracting specific information for various purposes such as data analysis, research, or integration into applications

Web scraping is the process of extracting data from websites.Using Python with the BeautifulSoup and requests libraries Requests for making HTTP requests. Beautiful Soup for parsing HTML and XML documents.

pip install beautifulsoup4
pip install html5lib

```
import requests
from bs4 import BeautifulSoup
# Send an HTTP GET request to the URL
req = requests.get("https://www.anaconda.com/")
# Parse the HTML content of the webpage
soup = BeautifulSoup(req.content, "html.parser")
# Print the parsed HTML content
print(soup)
```

Data Cleaning and Munging

Data cleaning and munging refer to the process of preparing raw data for analysis. Real-world data often comes with various issues such as missing values, incorrect formats, or outliers that can affect analysis. Therefore, cleaning the data is a crucial step before performing any analysis.

- Data Cleaning removes duplicates, handles missing values, corrects errors, and standardizes formats.
- Data Munging transforms data types, creates new features, normalizes/scales data, and merges datasets.

Converting Data Types
While working with data, especially from sources like CSV files.It is often need to convert data types. For instance, a column that represents numerical values might be read as strings,
so we need to convert them to floats or integers for analysis

Example:

closing_price = float(row[2])

This line converts the string value in row[2] to a float.

Parsing Data

Parsing is an essential skill for data cleaning and preparation, ensuring that raw data is accurately converted into a format suitable for analysis. Instead of converting data right before using it, we can convert it as we read it from the source. This can help avoid errors and ensure consistency. We can achieve this by creating a function that wraps csv.reader and applies the necessary conversions (parsers) to each column.
parse_row Function This function takes a list of values (input_row) and a list of parsers and applies the appropriate parser to each value.

Example
Suppose you have a CSV file with rows of data that include a date string, a company symbol, and a price. You want to parse these rows such that:

- The date string is parsed into a datetime object.
- The company symbol remains unchanged.
- The price string is parsed into a float.
-

input_row = ["2024-06-27", "AAPL", "150.23"]

```
import dateutil.parser
 parsers = [dateutil.parser.parse, None, float]
parsed_row = parse_row(input_row, parsers) print(parsed_row)
```

Output:
[**datetime.datetime(2024, 6, 27, 0, 0), "AAPL", 150.23]**

Manipulating Data

One of the essential skills for a data scientist is the ability to manipulate data. This involves transforming and organizing data to make it useful for analysis. We'll explore some examples and develop tools to make data manipulation easier.

Common Steps in Data Munging:

1. **Reshaping Data**:
   - Transforming data into a desired structure, such as pivoting tables or flattening nested data.
   - Example: Converting wide-format data to long-format for time series analysis.
2. **Merging Data**:
   - Combining multiple datasets into a single, unified dataset.
   - Example: Merging sales data from different regions to create a comprehensive dataset.
3. **Filtering and Subsetting**:
   - Selecting specific subsets of data based on criteria or conditions.
   - Example: Extracting data for a particular year or filtering rows where sales exceed a certain threshold.
4. **Transforming Variables**:
   - Creating new variables or modifying existing ones to better suit analysis needs.
   - Example: Creating a new variable for profit margin by subtracting costs from revenue.
5. **Enriching Data**:
   - Adding new information to the dataset, often from external sources, to provide additional context.
   - Example: Adding demographic information to customer data based on zip codes.

Example:
Imagine we have a list of dictionaries representing stock prices:
data = [{'closing_price': 102.06, 'date': datetime.datetime(2014, 8, 29, 0, 0), 'symbol': 'AAPL'}

To Find the Highest-Ever Closing Prices for Each Stock

- Group rows by the symbol.
- For each group, find the maximum closing_price.

```
from collections import defaultdict

# Group rows by symbol
by_symbol = defaultdict(list)
for row in data:
    by_symbol[row["symbol"]].append(row)

# Find max closing price for each symbol
max_price_by_symbol = {symbol: max(row["closing_price"]
                                   for row in grouped_rows)
                       for symbol, grouped_rows in by_symbol.items()}
```

Program

Consider an html file. Write python program to Scrap the page extract values associated with tags and properties.

To scrape an HTML page and extract values associated with tags and properties, we can use libraries such as requests to fetch the HTML content and BeautifulSoup from bs4 to parse the HTML and extract the desired information. Here's an example program that demonstrates how to do this:

Steps

- Install Required Libraries:
- Install the requests and beautifulsoup4 libraries if you haven't already.
- pip install requests beautifulsoup4
- Write the Python Program:
- Create a Python script to fetch the HTML content of a webpage and parse it using BeautifulSoup.

Code:

```
import requests
from bs4 import BeautifulSoup

# URL of the webpage to be scraped
url = 'http://example.com'

# Fetch the content of the webpage
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.content,'html.parser')

# Extract data associated with specific tags and properties
# Example: Extract all text within <h1> tags
h1_tags = soup.find_all('h1')
```

```
for tag in h1_tags:
print('H1 tag text:', tag.get_text())

# Example: Extract all links (<a> tags) and their URLs (href attributes)
a_tags = soup.find_all('a')

for tag in a_tags:
print('Link text:', tag.get_text())
print('URL:', tag.get('href'))

# Example: Extract all images (&lt;img&gt; tags) and their sources (src attributes)

img_tags = soup.find_all('img')

for tag in img_tags:
print('Image alt text:', tag.get('alt'))
print('Image URL:', tag.get('src'))

# Extract data from a specific class or id
# Example: Extract text from a div with a specific class name

div_tags = soup.find_all('div' class_='example-class')

for tag in div_tags:
print('Div class text:' tag.get_text())
else:
print('Failed to retrieve the webpage')
```

SUBJECT: Data Science and Its Applications (21AD62)

MODULE-1 MACHINE LEARNING

Syllabus: Modeling, What Is Machine Learning?, Over fitting and Under fitting, Correctness, The Bias-Variance Tradeoff, Feature Extraction and Selection, **k-Nearest Neighbors,** The Model, Example: The Iris Dataset, The Curse of Dimensionality, **Naive Bayes,** *A* Really Dumb Spam Filter, *A* More Sophisticated Spam Filter, Implementation, Testing Our Model, Using Our Model, **Simple Linear Regression,** The Model, Using Gradient Descent, Maximum Likelihood Estimation, **Multiple Regression,** The Model, Further Assumptions of the Least Squares Model, Fitting the Model, Interpreting the Model, Goodness of Fit, Digression: The Bootstrap, Standard Errors of Regression Coefficients, Regularization, **Logistic Regression,** The Problem, The Logistic Function, Applying the Model, Goodness of Fit, Support Vector Machines.

Modeling

A model is essentially a simplified representation of reality that helps us to understand, predict, or control some aspect of the world. It captures the key features and relationships of the phenomena. The primary goal of a machine learning model is to make predictions or decisions based on input data.

It's simply a specification of a mathematical (or probabilistic) relationship that exists between different variables.

For example if we want to raise money for one social networking site, It might build a *business model* that takes inputs like "number of users" and "ad revenue per user" and "number of employees" and outputs your annual profit for the next several years. A cookbook recipe entails a model that relates inputs like "number of eaters" and "hungriness" to quantities of ingredients needed.

The business model is probably based on simple mathematical relationships: profit is revenue minus expenses, revenue is units sold times average price, and so on. The recipe model is probably based on trial and error — someone went in a kitchen and tried differentcombinations of ingredients until they found one they liked. And the poker model is based on probability theory, the rules of poker, and some reasonably innocuous assumptions about the random process by which cards are dealt.

What Is Machine Learning?

The machine learning refer to creating and using models that are learned from data. In other contexts this might be called predictive modeling or data mining. Typically, the goal is to use existing data to develop models that can use to predict various outcomes for new data, such as:

- Predicting whether an email message is spam or not
- Predicting whether a credit card transaction is fraudulent
- Predicting which advertisement a shopper is most likely to click on
- Predicting which football team is going to win the Super Bowl

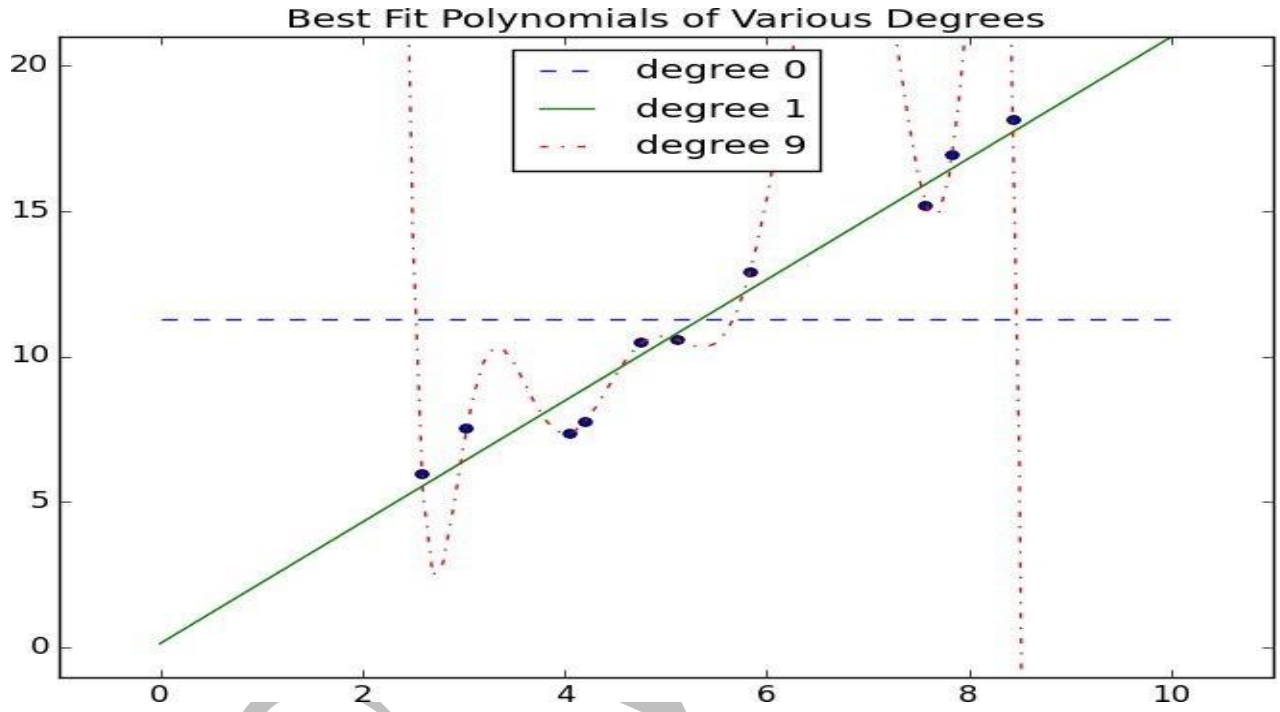
.

Overfitting and Underfitting

A **Over fitting** is a model that performs well on the training data but that generalizes poorly to any new data. This could involve learning noise in the data,or it could involve learning to identify specific inputs rather than whatever factors are actually predictive for the desired output.

Under fitting, producing a model that doesn't perform well even on the training data, or model fails to understand the relationships between the input features and outcome.

Let's us consider Figure below, to fit three polynomials to a sample of data.



The horizontal line shows the best fit degree 0 polynomial. It severely under fits the training data. The best fit degree 9 polynomial goes through every training data point exactly, but it very severely over fits — if we were to pick a few more data points it would quite likely miss them by a lot. And the degree 1 line strikes a nice balance — it's pretty close to every point, and the line will likely be close to new data points as well. Clearly models that are too complex lead to over fitting and don't generalize well beyond the data they were trained on. The most fundamental approach involves using different data to train the model and to test the model. Overfitting and Underfitting

Overfitting

Causes:

- Too many parameters relative to the number of observations.
- Model complexity is too high.
- Insufficient training data.

Symptoms:

- High accuracy on training data.
- Low accuracy on validation/test data.

Example: Consider a polynomial regression problem where we are trying to fit a polynomial to data that has aquadratic relationship.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate some data
np.random.seed(0)
X = np.random.normal(0, 1, 100)
y = 2 * X ** 2 + 3 + np.random.normal(0, 0.5, 100)

# Split the data into training and test sets
X_train = X[:80]
y_train = y[:80]
X_test = X[80:]
y_test = y[80:]

# Reshape data for sklearn
X_train = X_train[:, np.newaxis]
X_test = X_test[:, np.newaxis]

# Fit polynomial regression with a high degree
poly = PolynomialFeatures(degree=10)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)
model = LinearRegression()
model.fit(X_poly_train, y_train)
y_poly_pred_train = model.predict(X_poly_train)
y_poly_pred_test = model.predict(X_poly_test)

# Plot the data and the polynomial regression line
plt.scatter(X_train, y_train, color='blue';, label='Training data')
plt.scatter(X_test, y_test, color='red', label='Test data')
plt.plot(np.sort(X_train[:, 0]), y_poly_pred_train[np.argsort(X_train[:, 0])], color='green', label='Polynomial fit';)
plt.legend()
plt.show()
```

```
# Calculate and print training and test errors
train_error = mean_squared_error(y_train, y_poly_pred_train)
test_error = mean_squared_error(y_test, y_poly_pred_test)
print(f'Training error: {train_error}')
print(f'Test error: {test_error}')
```

In this example, using a polynomial of degree 10 leads to overfitting. The model fits the training data very well,capturing noise, but generalizes poorly to the test data.

Underfitting

Causes:

- Model complexity is too low.
- Not enough features.
- Excessive regularization.

Symptoms:

- Low accuracy on training data.
- Low accuracy on validation/test data.

Example: Continuing with the same data, consider fitting a linear regression model:

```
# Fit a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Plot the data and the linear regression line
plt.scatter(X_train, y_train, color='blue', label='Training data')
plt.scatter(X_test, y_test, color='red', label='Test data')
plt.plot(np.sort(X_train[:,0]),y_pred_train[np.argsort(X_train[:,0])],color='green',label='Linear fit')
plt.legend()
plt.show()

# Calculate and print training and test errors
train_error = mean_squared_error(y_train, y_pred_train)
test_error = mean_squared_error(y_test, y_pred_test)
print(f'Training error: {train_error}')
print(f'Test error: {test_error}')
```

Here, using a linear regression model leads to underfitting. The model is too simple to capture the quadratic relationship in the data.

Correctness

Correctness refers to how accurately a model's predictions align with the actual outcomes. It can be quantified using various evaluation metrics depending on the type of problem and the specific goals of the model.

In binary classification problems, such as determining whether an email is spam or not, the performance of the model can be evaluated using a **confusion matrix**. This matrix summarizes the outcomes of the predictions made by the model compared to the actual outcomes.

A **confusion matrix** is a table that is used to describe the performance of a classification model.

Let's consider a data for building a model to make a judgment.

	Predict "Spam"	**Predict "Not Spam"**
Actual Spam	True Positive (TP)	False Negative (FN)
Actual Not Spam	False Positive (FP)	True Negative (TN)

Given a set of labeled data and such a predictive model, every data point lies in one of four categories:

- **True Positive (TP)**: An email is actually spam, and the model correctly identifies it as spam.
- **False Positive (FP)** An email is not spam, but the model incorrectly identifies it as spam
- **False Negative (FN)** : An email is spam, but the model incorrectly identifies it as not spam.
- **True Negative (TN)**: An email is not spam, and the model correctly identifies it as not spam.

Correctness can be measured by

Accuracy: The proportion of total predictions that are correct.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Code:

```
def accuracy(tp, fp, fn, tn): correct = tp + tn
total = tp + fp + fn + tn
return correct / total
print accuracy(70, 4930, 13930, 981070)   # 0.98114
```

Precision: The proportion of positive predictions that are actually correct.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Code:

```
def precision(tp, fp, fn, tn):
return tp / (tp + fp)
print precision(70, 4930, 13930, 981070)  # 0.014
```

Recall (Sensitivity or True Positive Rate): The proportion of actual positives that are correctly identified.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Code:

```
def recall(tp, fp, fn, tn):
return tp / (tp + fn)
print recall(70, 4930, 13930, 981070)
```

F1 Score: The harmonic mean of precision and recall, providing a balance between the two.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Code:

```
def f1_score(tp, fp, fn, tn):
p = precision(tp, fp, fn, tn)
r = recall(tp, fp, fn, tn)
return 2 * p * r / (p + r)
```

Usually the choice of a model involves a trade-off between precision and recall.

- A model that predicts "yes" when it's even a little bit confident will probably have a high recall but a low precision;
- A model that predicts "yes" only when it's extremely confident is likely to have a low recall and a high precision.

The Bias-Variance Trade-off

- The bias-variance tradeoff is a key concept in understanding the performance of machine learning models.
- **Bias** is the error due to overly simplistic assumptions in the learning algorithm.
- High bias can cause an algorithm to miss relevant relations between features and target outputs (under fitting).
- **Variance** is the error due to too much complexity in the learning algorithm.
- High variance can cause an algorithm to model the random noise in the training data rather than the expected outputs (over fitting).
- The tradeoff is about finding a balance between bias and variance to minimize total error.
- Typically, by increasing model complexity will decrease bias but increase variance, while decreasing complexity will increase bias but decrease variance.
- The goal is to find the spot where the model generalizes well to new data.

The Bias and Variance both will measures what would happen if the model is retrain many times on different sets of training data.

For example, the degree 0 model in "Over fitting and Under fitting" will make a lot of mistakes for pretty much any training set (drawn from the same population), which means that it has a high bias. However, any two randomly chosen training sets should give pretty similar models (since any two randomly chosen training sets should have pretty similar average values). So we say that it has a low variance. High bias and low variance typically correspond to underfitting.

On the other hand, the degree 9 model fit the training set perfectly. It has very low bias but very high variance (since any two training sets would likely give rise to very different models). This corresponds to overfitting.

Thinking about model problems this way can help you figure out what do when your model doesn't work so well.

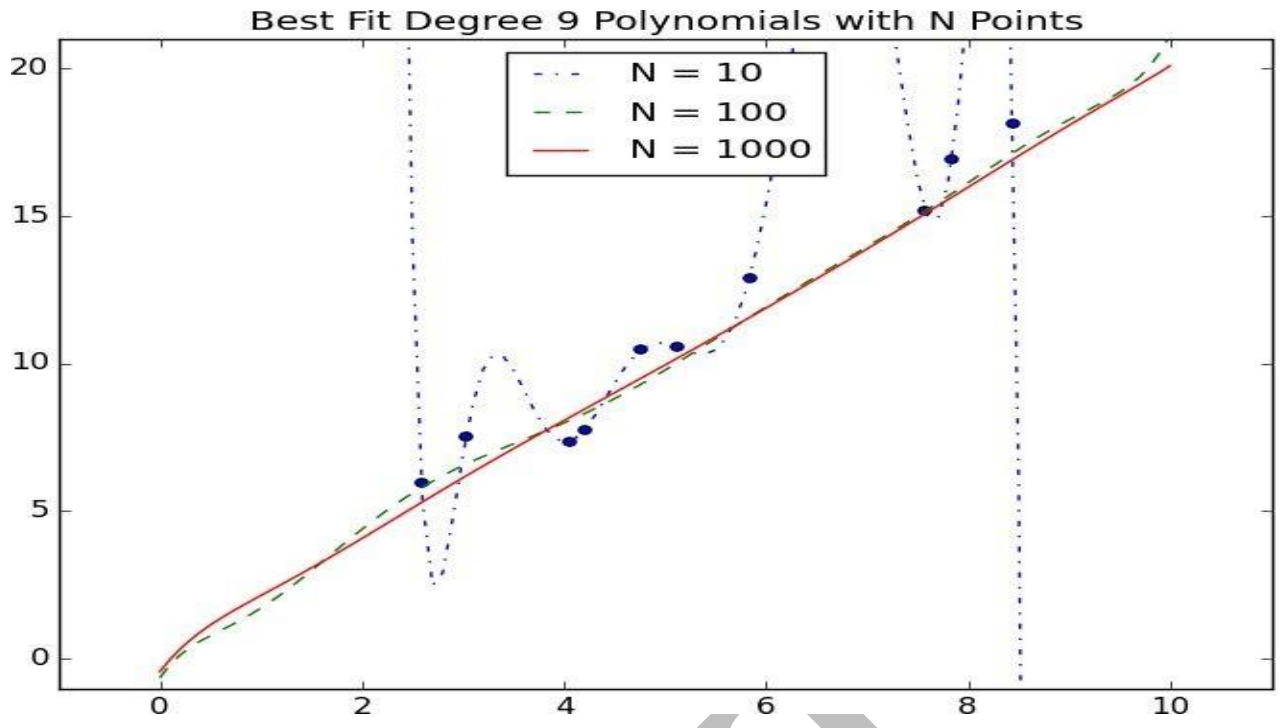
Adding More Features

If the model has high bias, it means the model is too simple to capture the underlying patterns in the data.In such cases, adding more features can help improve the model by providing it with more information. For example, in the context of polynomial regression:

- A degree 0 model (just a constant) is too simple.
- A degree 1 model (a straight line) is better because it can capture linear relationships.
- A higher-degree model can capture more complex relationships.

Reducing Features or Adding More Data

- If the model has high variance, it means the model is too complex and is over fitting the training data. Removing some features can help by simplifying the model, thus reducing the variance.
- Another effective way to take action to reduce or prevent high variance is to gather more data. More data can help the model generalize better because it provides more examples for the model to learn from, reducing the risk of over fitting.



In Figure 11-2, To fit a degree 9 polynomial to different size samples. If the model is trained on 100 datapoints, there's much less over fitting. And the model trained from 1,000 data points looks very similar to the degree 1 model. Holding model complexity constant, the more data , the harder it is to over fit.

Feature Extraction and Selection

Feature selection involves selecting a subset of the most important features for use in model construction. This can improve the model's performance by reducing over fitting, speeding up the training process, and improving the model's interpretability.

When the data doesn't have enough features, the model is likely tounder fit. And when the data has too many features, it's easy to overf it.

Features are whatever inputs that provide to our model.

In the simplest case ex: If we want to predict someone's salary based on her years of experience, then years of experience is the only feature it has.

In complicated case ex: Imagine trying to build a spam filter to predict whether an email is junk or not. Most model it is just a collection of text. To extract features.

For example:

- Does the email contain the word "lottery"? How
- many times does the letter d appear? What was
- the domain of the sender?

The first is simply a yes or no, which we typically encode as a 1 or 0.
The second is a number. And the third is a choice from a discrete set of options.

To extract features from our data that falls into one of these threecategories.

- The Naive Bayes classifier - is suited to yes-or-no features.

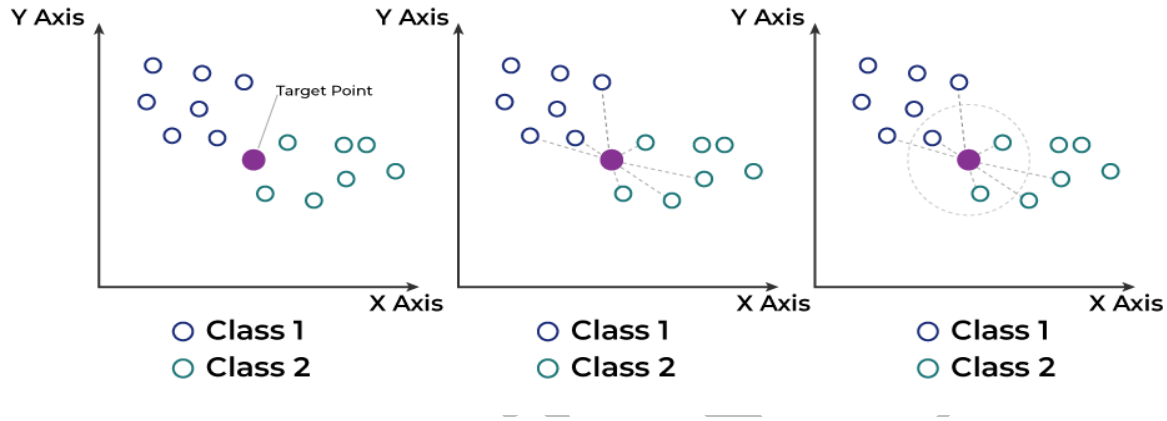
- Regression models-require numeric features
- Decision trees- can deal with numeric or categorical data.

The features are choosen by the combination of experience and domain expertise comes .

K-Nearest Neighbors (KNN) Algorithm

The **K-Nearest Neighbors (KNN) algorithm** is a supervised machine learning method employed to tackle classification and regression problems..

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity, where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Working of KNN:

:

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

- To measure the similarity between target and training data points, Euclidean distance is used. Distance is calculated between each of the data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are the nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- In the classification problem, the class labels of are determined by performing majority voting. The class with the most occurrences among the neighbors becomes the predicted class for the target data point.
- In the regression problem, the class label is calculated by taking average of the target values of K nearest neighbors. The calculated average value becomes the predicted output for the target data point.

X is the training dataset with n data points, where each data point is represented by a d-dimensional feature vector and Y be the corresponding labels or values for each data point in X. Given a new data point x, the algorithm calculates the distance between x and each data point in X using a distance metric, such as Euclidean distance:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Distance Metrics

- Euclidean Distance: $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
- Manhattan Distance: $\sum_{i=1}^{n}|x_i - y_i|$
- Minkowski Distance: $(\sum_{i=1}^{n}|x_i - y_i|^p)^{1/p}$

The algorithm selects the K data points from X that have the shortest distances to x. For classification tasks, the algorithm assigns the label y that is most frequent among the K nearest neighbors to x. For regression tasks, the algorithm calculates the average or weighted average of the values y of the K nearest neighbors and assigns it as the predicted value for x.

Python program to build a nearest neighbor model that can predict the class from the IRIS dataset

```
import numpy as np
from collections import Counter
# Sample dataset (Iris data: sepal length, sepal width, petal length, petal width, species)
data = [
(5.1, 3.5, 1.4, 0.2, 'setosa'),
(4.9, 3.0, 1.4, 0.2, 'setosa'),
(5.0, 3.6, 1.4, 0.2, 'setosa'),
(6.7, 3.0, 5.0, 1.7,'versicolor'),
(6.3, 3.3, 6.0, 2.5, 'virginica'),
(5.8, 2.7, 5.1, 1.9, 'virginica')
]
# New data point (sepal length, sepal width, petal length, petal width)
new_point = (5.5, 3.4, 1.5, 0.2)

# Function to calculate Euclidean distance
def euclidean_distance(point1, point2):
return np.sqrt(sum((x - y) ** 2 for x, y in zip(point1, point2)))

# Function to get the nearest neighbors
def get_nearest_neighbors(data, new_point, k):
distances = [(euclidean_distance(point[:-1], new_point), point[-1]) for point in data]
distances.sort(key=lambda x: x[0])
return [label for _, label in distances[:k]]
```

```
# Function to predict the class
def predict(data, new_point, k):
nearest_neighbors = get_nearest_neighbors(data, new_point, k)
most_common = Counter(nearest_neighbors).most_common(1)
return most_common[0][0]

# Predict the class for the new data point
k = 3
predicted_class = predict(data, new_point, k)
print(f'The predicted class for the new point is: {predicted_class}')
```

The Curse of Dimensionality

The Curse of Dimensionality is a concept that describes the challenges and issues that arise when working with high-dimensional data.As the number of dimensions increases, the volume of the space increases exponentially. This means that data pointsbecome sparse, and the distances between them grow, making it difficult to find meaningful patterns.

The Curse of Dimensionality impacts various aspects of data analysis, including distance calculations, data sparsity, and overfitting.

1. Distance Measures Become Less Meaningful:

In high-dimensional spaces, the distances between points tend to become similar, making it harder to distinguish between near and far points. This is problematic for algorithms that rely on distance measures, such as k-Nearest Neighbors (k-NN) and clustering algorithms.

2. Data Sparsity:

With more dimensions, the data points spread out more, leading to sparsity. In a high-dimensional space, even a large dataset may have very few data points in any given region. This sparsity makes it hard to find reliable patterns and can reduce the effectiveness of algorithms.

3. Overfitting:

High-dimensional datasets often contain many irrelevant or noisy features, which can lead to overfitting. The model may capture noise instead of the underlying pattern, performing well on training data but poorly on unseen data.

Example

Consider a simple example using a dataset with points uniformly distributed in a unit cube. We can observe how the volume and distances change as the number of dimensions increases.

1. Volume of a Hypercube:

In a 1-dimensional space , the unit hypercube is simply a line segment of length 1.

In a 2-dimensional space (a square), the unit hypercube has an area of 1.

In a 3-dimensional space (a cube), the unit hypercube has a volume of 1.

However, in higher dimensions, the volume of the unit hypercube becomes negligible compared to the space it occupies. For instance:

- In a 10-dimensional space, the unit hypercube has a volume of 1 10 =1.

- In a 100-dimensional space, the unit hypercube still has a volume of 1 100 =1, but the space it occupies is vast.

2. Distance Calculations:

In a 1-dimensional space, consider two points at 0 and 1. The distance between them is 1.

In a 2-dimensional space, consider two points (0, 0) and (1, 1). The Euclidean distance is sqrt{2}.

In a 3-dimensional space, consider two points (0, 0, 0) and (1, 1, 1). The Euclidean distance is sqrt{3}.

As dimensions increase, the distances between points increase as well. However, the difference between the maximum and minimum distances decreases proportionally, making distances less discriminative.

Dimensionality Reduction

Dimensionality reduction is a process used to reduce the number of features (dimensions) in a dataset while retaining as much information as possible. This technique helps in simplifying models, reducing computational costs, and mitigating issues related to the curse of dimensionality.

Principal Component Analysis (PCA)

PCA is a widely used technique for dimensionality reduction that transforms the original features into a new set of uncorrelated features called principal components. The first principal component captures the most variance in the data,and each subsequent component captures the remaining variance under the constraint of being orthogonal to the previous components.

Example: Dimensionality Reduction Using PCA

1. **Standardize the Data**: Standardization ensures that each feature contributes equally to the analysis by scaling the data to have a mean of 0 and a standard deviation of 1.

2. **Compute the Covariance Matrix:** The covariance matrix describes the variance and the covariance between the features.

3. **Compute the Eigenvalues and Eigenvectors**: The eigenvectors determine the directions of the new feature space, while the eigenvalues determine their magnitude (i.e., the amount of variance captured by each principal component).

4**. Sort Eigenvalues and Select Principal Components:** The eigenvalues are sorted in descending order, and the top k eigenvalues are selected. The corresponding eigenvectors form the new feature space.

5. **Transform the Data**: The original data is projected onto the new feature space to obtain the reduced dataset.

CODE:

Example using Python to illustrate PCA for dimensionality reduction:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Generate a synthetic dataset
np.random.seed(42)
X = np.random.rand(100, 3) # 100 samples, 3 features
```

```
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Print the explained variance ratios
Print(f'Explained variance ratios:&amp;quot;, pca.explained_variance_ratio')

# Visualize the original and reduced data
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

# Original data
ax[0].scatter(X[:, 0], X[:, 1], c='blue', label='Original Data')
ax[0].set_xlabel('Feature 1';)
ax[0].set_ylabel('Feature 2')
ax[0].set_title('Original Data')

# Reduced data

ax[1].scatter(X_pca[:, 0], X_pca[:, 1], c='red', label='PCA Reduced Data')
ax[1].set_xlabel('Principal Component 1')
ax[1].set_ylabel('Principal Component 2')
ax[1].set_title('PCA Reduced Data')
plt.legend()
plt.show()
```


Using Gradient Descent

Gradient descent is a fundamental optimization algorithm used in machine learning to minimize a cost function. Gradient Descent Basics Gradient descent aims to find the parameters (weights) of a model that minimize the cost function, which measures how well the model fits the data.

The steps are as follows:

1. Initialize Parameters: Start with random initial values for the parameters (weights).

2. Compute the Gradient: Calculate the gradient of the cost function with respect to each parameter. The gradient is a vector of partial derivatives, indicating the direction and rate of the steepest increase in the cost function.

3. Update Parameters: Adjust the parameters in the opposite direction of the gradient by a small amount, which is determined by the learning rate. This step is repeated iteratively:

$$\theta_i = \theta_i - \alpha \partial J(\theta)/\partial \theta_i$$

where θ_i is the i-th parameter, α is the learning rate, and $\mathbf{J(\theta)/\partial\theta_i}$ is the cost function.

4. Convergence Check: Repeat steps 2 and 3 until the change in the cost function is smaller than a predefined threshold or a maximum number of iterations is reached.

Gradient Descent for Parameterized Models

When fitting parameterized models, the cost function depends on the difference between the predicted and actual values. For example, in linear regression, the cost function J(θ) is typically the mean squared error:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

where:

- m is the number of training examples.
- hθ(x(i)) is the predicted value for the i^{th} training example.
- y(i) is the actual value for the i^{th} training example.

The gradient descent algorithm for linear regression would involve computing the partial derivatives of J(θ) with respect to each parameter θ_j:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Python program to illustrate gradient descent for a simple linear regression model

```
import numpy as np

# Example data: y = 2x + 3 with some noise
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Add x0 = 1 to each instance
X_b = np.c_[np.ones((100, 1)), X]

# Parameters
learning_rate = 0.1
n_iterations = 1000
m = len(X_b)
theta = np.random.randn(2, 1)
```

```
# Gradient Descent
for iteration in range(n_iterations):
gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
theta = theta - learning_rate * gradients
print('Fitted parameters:', theta)
```

Naive Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' Theorem, with the assumption that the features are independent given the class label.

This model predicts the probability of an instance belongs to a class with a given set of feature value. It is a probabilistic classifier. It is because it assumes that one feature in the model is independent of existence of another feature. In other words, each feature contributes to the predictions with no relation between each other.. It uses Bayes theorem in the algorithm for training and prediction

The core assumption of Naive Bayes is **conditional independence**.

Mathematically, if Xi & Xj represents the event that the i^{th} word is present in the message, then the assumption says:

$$\mathbf{P(Xi \text{ and } Xj/spam)=P(Xi/spam)\cdot P(Xj/spam)}$$

Sophisticated Spam Filter

Imagine now that a vocabulary of many words $\mathbf{W_1, W_2, W_3 \ldots\ldots W_N}$. To move this into the probability theory, let's write $\mathbf{X_i}$ let the event "a message contains the word "W_i" Also imagine that with an estimate $\mathbf{P(X_i/S)}$ for the probability that a spam message contains the *i* th word,and a similar estimate $\mathbf{P(X_i/N)}$ for the probability that a nonspam message containsthe *i*th word.

The key to Naive Bayes is making the assumption that the presences of each word are independent of one another, conditional on a message being spam or not.

Intuitively, this assumption means that knowing whether a certain spam message contains the word "lottery" gives the no information about whether that same message contains theword "rolex." In math terms, this means that:

$$P(X_1= x_1 \ldots\ldots X_n= x_n/S) = P(X1= x_1/S) \text{ x} \ldots\ldots\ldots\ldots\ldots \text{ x } P(X_n= x_n/S)$$

Imagine that a vocabulary consists *only* of the words "lottery" and "rolex," and that half of all spam messages are for "cheap rolex" and that the other half are for "authentic lottery." In this case, the

Naive Bayes estimate that a spam message contains both "lottery" and "rolex" is:

$$P(X_1 = 1, X_2 = 1 \mid S) = P(X_1 = 1 \mid S)P(X_2 = 1 \mid S) = .5 \times .5 = .25$$

since the "lottery" and "rolex" words actually never occur together. Despite the unrealisticness of this assumption, this model often performs well and is used in actual spam filters.

The same Bayes's Theorem reasoning used for the word "lottery-only" spam filter tells that we can calculate the probability a message is spam using the equation:

$$\mathbf{P(S/X=x) = P(X=x/S) / [P(X=x/S) + p(X=x/N)]}$$

The Naive Bayes assumption allows us to compute each of the probabilities on the rightsimply by multiplying together the individual probability estimates for each vocabulary word.

Let us consider three words: "rolex" "lottery," and "meeting." the following probabilities based on historical data.

For Training Data

Assume Likelihoods as

- Rolex : P(R/N) = 0.1 & P(R/S) = 0.8
- Lottery : P(L/N)=0.05 & P(L/S) = 0.7
- Meeting : P(M/N)=0.9 & P(M/S) = 0.2
- Project: P(P/N)=0.7 & P(P/S) = 0.25

Prior Probabilities:

- P(spam) = 0.3 (30% of messages are spam)
- P(Normal) = 0.7 (70% of messages are non-spam)

New Message containing the words "rolex" and "lottery."

Let us classify it as spam or not spam using Naive Bayes.

- **Calculate the probability of the message being spam:**

P(spam/message)∝P(S)·P(R/S)·P(L/S) = 0.3X0.8X0.7= 0.168

- **Calculate the probability of the message being ham:**

P(normal/message)∝P(N)·P(R/N)·P(L/N) = 0.7x0.1x0.05= 0.0035

- **Normalize the Probabilities**
- To get the actual probabilities, we need to normalize these values so they sum to 1.
- P(message)=P(S/ message)+P(N| message)
- P(message)=0.168 + 0.0035= 0.1715
- So, the normalized probabilities are:
- P(S/message)=0.1680/1715 ≈ 0.98
- P(N/message)=0.00350/1715 ≈ 0.02

Given the message contains the words "rolex" and "lottery," there is a 98% chance it is spam and a 2% chance it is not spam (normal).

Python Code:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
# Sample data
data = {
```

```
    'message': [
        'Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121
to
       receive entry question(std txt rate)',
        'Nah I don\'t think he goes to usf, he lives around here though',
        'WINNER!! As a valued network customer you have been selected to receivea £900
prize
        reward!',
        'I HAVE A work ON SUNDAY !!',
        'Had your mobile 11 months or more? U R entitled to update to the latest colour
mobiles with camera for free! Call The Mobile Update Co FREE on 08002986030'
    ],
    'label': ['spam', 'ham', 'spam', 'ham', 'spam']
}
# Convert data to DataFrame
df = pd.DataFrame(data)

# Feature extraction
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['message'])
y = df['label']

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

# Train the Naive Bayes classifier
nb = MultinomialNB()
nb.fit(X_train, y_train)

# Make predictions
y_pred = nb.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy}')
print('Classification Report:')
print(report)

# Function to classify a new message
def classify_message(message):
    message_transformed = vectorizer.transform([message])
    prediction = nb.predict(message_transformed)
    return prediction[0]

# Test the classifier
new_message = 'Congratulations! You have won a free ticket to Bahamas. Call now!'
print(f'The message: "{new_message}" is classified as {classify_message(new_message)}')
```

Simple linear Regression

- Regression is a statistical technique used to model and analyze the relationships between variables.
- It helps in understanding how the dependent variable (Y)changes when any one of the independent variables (X)is varied.
- The primary goal of regression is to predict or estimate the value of the dependent variable based on the values of one or more independent variables.
- Simple Linear Regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:
- **Independent Variable (X)**: Also known as the predictor or explanatory variable.
- **Dependent Variable (Y)**: Also known as the response or outcome variable.
- The goal of Simple Linear Regression is to model the relationship between these two variables by fitting a linear equation to the observed data.
- The linear equation for a Simple Linear Regression model is:

$$Y_i = \alpha + \beta X_i + \epsilon_i$$

Y - is the dependent variable.

X -is the independent variable.

α - is the intercept of the regression line. It is the value of Y when X=0.

β -is the slope of the regression line. It represents the change in Y for a one-unit change in X.

ϵ - error term, which accounts for the variability in Y that cannot be explained by the linear relationship with X.

- To fit the Simple Linear Regression model.
- Estimate the parameters α and β using **Ordinary Least Squares (OLS)**
- It minimizes the sum of the squared differences between the observed values and the values

predicted by the linear model.

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_i))^2$$

Where n is the number of observations, Y_i is the observed value of Y, and X_i is the observed value of X for the i-th observation.

After fitting the mode

- **Mean Squared Error (MSE)**: The average of the squared differences between the observed and predicted values.

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

Where $\hat{Y}_i$ is the predicted value of Y

R-squared (R^2): The proportion of the variance in the dependent variable that is predictable from the independent variable.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

Where $\bar{Y}$ is the mean of the observed values of Y.

Code

```
import numpy as np
import matplotlib.pyplot as plt
# Sample data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3, 5, 7, 11]


# Calculate means
x_mean = np.mean(x)
y_mean = np.mean(y
# Calculate the parameters
beta_1 = np.sum((x - x_mean) * (y - y_mean)) / np.sum((x - x_mean) ** 2)
beta_0 = y_mean - beta_1 * x_mean
```

```
print(f'Estimated parameters: beta_0 = {beta_0}, beta_1 = {beta_1}')

# Make predictions
y_pred = beta_0 + beta_1 * x

# Plot the data and the regression line
plt.scatter(x, y, color=&#39;blue&#39;, label=&#39;Data points&#39;)
plt.plot(x, y_pred, color=&#39;red&#39;, label=&#39;Regression line&#39;)
plt.xlabel(&#39;x&#39;)
plt.ylabel(&#39;y&#39;)
plt.legend()
plt.show()
```

Multiple Regressions

Multiple regressions are a statistical technique used to understand the relationship between one dependent variable and two or more independent variables. It extends simple linear regression, which involves only one independent variable, by incorporating multiple predictors to better capture the complexity of real-world phenomena.

The Multiple Regression Equation

The general form of the multiple regression equation is:

$$Y=\beta_0+\beta_1X_1+\beta_2X_2+\ldots+\beta nXn$$

- Y: Dependent variable.
- β_0: Intercept, the expected value of Y when all Xs are zero.
- $\beta_1,\beta_2,\ldots,\beta n$: Coefficients representing the change in Y for a one-unit change in the corresponding X, holding other variables constant.
- $X_1,X_2,\ldots,X_n$: Independent variables.
- ϵ: Error term, representing the deviation of observed values from the predicted values.

Steps in Multiple Regression Analysis

1. **Model Specification:** Define the dependent variable and select the independent variables based on theoretical understanding or empirical evidence.
2. **Data Collection:** Gather data for the dependent and independent variables. Ensure the data is clean and suitable for analysis.
3. **Estimation of Coefficients:** Use statistical software to estimate the coefficients (β\betaβ) of the regression equation. This is typically done using the Ordinary Least Squares (OLS) method, which minimizes the sum of the squared differences between observed and predicted values.
4. **Model Evaluation:** Assess the model's performance using various metrics:
    - **R-squared (R^2):** Proportion of variance in the dependent variable explained by the independent variables.
    - **Adjusted R-squared:** Adjusts R^2 for the number of predictors in the model.
    - **F-test:** Tests the overall significance of the model.

   - **t-tests:** Assess the significance of individual coefficients.
5. **Assumption Checking:** Ensure that the model meets the assumptions of multiple regression:
   - Linearity: The relationship between the dependent and independent variables is linear.
   - Independence: Observations are independent of each other.
   - Homoscedasticity: Constant variance of errors across all levels of the independent variables.
   - Normality: Errors are normally distributed.
6. **Diagnostics and Refinement:** Perform residual analysis to check for any patterns in the residuals that might indicate model misspecification. Address issues like multicollinearity (high correlation among predictors) if they arise.
7. **Interpretation:** Interpret the coefficients to understand the impact of each independent variable on the dependent variable. For example, a coefficient of 2 for X_1 means that a one-unit increase in X_1 results in an average increase of 2 units in Y, holding other variables constant.
8. **Prediction:** Use the fitted model to make predictions for new data points by plugging in values for the independent variables into the regression equation.

Need for fitting the model in multiple regressions

Fitting a model in multiple regressions is essential for several reasons, each contributing to the robustness, accuracy, and interpretability of the analysis. The reasons are

1. Understanding Relationships between Variables

Multiple regression allows for the examination of the relationship between a dependent variable and multiple independent variables simultaneously. This helps in understanding how various factors collectively influence the outcome.

2. Controlling for Confounding Variables

In many real-world scenarios, the effect of one independent variable on the dependent variable might be influenced by the presence of other variables. Multiple regression helps to isolate the effect of each independent variable by controlling for others, reducing potential confounding effects.

3. Improved Prediction Accuracy

By incorporating multiple predictors, the model can capture more information about the dependent variable, leading to better predictive accuracy compared to simple regression models with a single predictor.

4. Identifying Significant Predictors

Multiple regression helps in identifying which independent variables have a significant impact on the dependent variable. This is particularly useful in fields like economics, medicine, and social sciences, where understanding the importance of various factors is crucial.

5. Quantifying the Impact of Variables

The coefficients in a multiple regression model quantify the impact of each independent variable on the dependent variable, providing valuable insights into the strength and direction of these relationships.

6. Handling Multicollinearity

In multiple regression, it's important to assess and handle multicollinearity (when independent variables are highly correlated). Properly fitting the model involves diagnosing and addressing multicollinearity to ensure reliable and interpretable results.

7. Generalizability of Findings

A well-fitted multiple regression model that accounts for multiple factors is more likely to generalize to new data, making the findings more robust and applicable in various contexts.

8. Model Diagnostics and Validation

Fitting the model involves checking for assumptions (linearity, independence, homoscedasticity, normality) and performing diagnostics (residual analysis, influence analysis) to ensure the validity of the model. This step is crucial for the reliability of the regression results.

9. Enabling Complex Analyses

Multiple regression serves as a foundation for more complex analyses like interaction effects, polynomial regression, and hierarchical regression, expanding the analytical capabilities for research and decision-making.

10. Policy and Decision-Making

In applied fields, such as business, public policy, and healthcare, multiple regression models provide evidence-based insights that inform strategic decisions and policy-making by highlighting key factors and their relative importance

SUBJECT: Data Science and Its Applications (21AD62)

MODULE-5 Natural Language Processing

Syllabus: *Word Clouds, n-Gram Language Models, Grammars, An Aside: Gibbs Sampling, Topic Modeling, Word Vectors, Recurrent Neural Networks, Example: Using a Character-Level RNN,* ***Network Analysis,*** *Betweenness Centrality, Eigenvector Centrality, Directed Graphs and PageRank,* ***Recommender Systems,*** *Manual Cu ration, Recommending What's Popular, User-Based Collaborative Filtering, Item-Based Collaborative Filtering, Matrix Factorization.*

Introduction

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans through natural language. The goal of NLP is to enable computers to understand, interpret, and generate human language in a way that is both meaningful and useful.

Applications of NLP

1. Chabot's and Virtual Assistants.
2. Text Analysis and Sentiment Analysis
3. Machine Translation
4. Speech Recognition and Voice Assistants
5. Information Retrieval and Search Engines
6. Language Modeling and Understanding
7. Recommendation Systems.
8. Healthcare and Medical Applications

Word cloud

A word cloud, also known as a tag cloud, is a visual representation of text data. It highlights the most frequently occurring words within a body of text by making them larger and bolder compared to less frequent words. Word clouds are often used to quickly identify prominent terms in large text datasets and provide an at a glance summary of the content.

Example: The word cloud is created from a list of data science-related buzzwords, where each word is associated with two numbers: how frequently it appears in job postings and how frequently it appears on resumes

Creating a Word Cloud

To create a word cloud, the typically use a tool or library (such as `word cloud` in Python) to generate the visualization. The size of each word in the cloud would be proportional to its frequency.

Steps:

1. **Text Input**:
   - A body of text is provided as input, which can be anything from a single document to a collection of articles, tweets, or reviews.
2. **Text Processing**:
   - The text is processed to remove common stop words (e.g., "and", "the", "is") that do not contribute to the main content. Words are then counted to determine their frequency.
3. **Visualization**:
   - Words are arranged in a layout, with their size proportional to their frequency. The arrangement often ensures that words fit together compactly, creating a dense and readable cloud.

Example Data

Consider the following data for creating a word cloud:

data = [("big data", 100, 15), ("Hadoop", 95, 25), ("Python", 75, 50), ("R", 50, 40), ("machine learning", 80, 20), ("statistics", 20, 60), ("data science", 60, 70), ("analytics", 90, 3), ("team player", 85, 85), ("dynamic", 2, 90), ("synergies", 70, 0), ("actionable insights", 40, 30), ("think out of the box", 45, 10), ("self-starter", 30, 50), ("customer focus", 65, 15), ("thought leadership", 35, 35)]
Where,

- X-Axis: Represents the frequency of the buzzword in job postings.
- Y-Axis: Represents the frequency of the buzzword in resumes.

By plotting the words on a scatter plot with these axes, adds more value and clarity compared to a traditional word cloud

Limitations

1. Lack of Precision: Word clouds don't provide exact counts or clear comparisons between words.
2. Random Placement: The position of words in a word cloud is arbitrary and doesn't convey any additional information.
3. Space Usage: Words are placed wherever there is space, which can make the visualization cluttered and less, readable.

Python Code

```
import  matplotlib.pyplot as plt
from word cloud import wordcloud

data = [
   ("big data", 100, 15),
   ("Hadoop", 95, 25),
   ("Python", 75, 50),
   ("R", 50, 40),
   ("machine learning", 80, 20),
   ("statistics", 20, 60),
   ("data science", 60, 70),
   ("analytics", 90, 3),
   ("team player", 85, 85),
   ("dynamic", 2, 90),
   ("synergies", 70, 0),
   ("actionable insights", 40, 30),
   ("think out of the box", 45, 10),
   ("self-starter", 30, 50),
   ("customer focus", 65, 15),
   ("thought leadership", 35, 35)
]
#Prepare text for word cloud
 word_freq = {word: job_popularity + resume_popularity for word, job_popularity, resume_popularity in data}
```

```
# Generate word cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate_from_frequencies(word_freq)

# Display the generated image
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

N- Gram Model

Language modeling is the way of determining the probability of any sequence of words. Language modeling is used in a wide variety of applications such as Speech Recognition, Spam filtering, etc.

N-gram modeling is a foundational technique in natural language processing (NLP) that leverages sequences of words to model and generate human-like text. It balances simplicity and efficiency.

N-gram

N-gram can be defined as the contiguous sequence of **n** items from a given sample of text or speech. The items can be letters, words, or base pairs according to the application. The N-grams typically are collected from a text or speech corpus.

An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. A good N-gram model can predict the next word in the sentence. The goal is to estimate the probability of a word by decomposing a sentence probability into a product of conditional probabilities using chain rule as follows:

$P(S) = P(W_1, W_2, W_3, W_4, \ldots\ldots\ldots\ldots W_n)$

$= \mathbf{P}(W_1)\, P(W_2 / W_1,)\, \mathbf{P}(W_3 / W_1\, W_2)\, \mathbf{P}\,(W_4 / W_1\, W_2\, W_3) \ldots\ldots\ldots\ldots\ldots P(W_n / W_1, W_2 \ldots\ldots\ldots W_{n-1})$

$= \sum_{i=1}^{n} \boldsymbol{P}\left(\frac{\mathbf{Wi}}{\boldsymbol{hi}}\right)$

Where **hi** is history of word W_i defined as $W_1, W_2 \ldots\ldots\ldots\ldots W_{i-1}$.

To calculate sentence probability, need to calculate the probability of the word, given the sentence of words preceding it. An n-gram model simplifies the task by approximating the probability of a word given all previous words by the conditional probability given previous n-1 words only.

$$\boldsymbol{P}\left(\frac{\mathbf{Wi}}{\boldsymbol{hi}}\right) = \boldsymbol{P}\left(\frac{\mathbf{Wi}}{W_{i-n+1} \ldots\ldots\ldots W_{i-1}}\right)$$

Thus an n- gram model calculate $P(W_i / h_i)$ by modeling languages as marker model of order n-1.

Steps in N-gram Modeling

1. Collect and pre-process data: Gather a corpus of text, Clean and tokenize the text.
2. Generate n-grams:
   Example of N-gram such as unigram ("This", "article", "is", "on", "NLP") or bi-gram ('This article', 'article is', 'is on', 'on NLP').

3. Build transitions dictionary:
    - Store frequencies of each n-gram in a dictionary.
4. Estimate probabilities:
    - Calculate the probability of each word given its preceding words using formulas

Unigram Probability	$P(w)=\frac{count(w)}{N}$
Bayes Rule	$P(A\|B)=\frac{P(A\cap B)}{P(B)}$
Bigram Probability	$P(w_i\|w_{i-1})=\frac{count(w_{i-1},w_i)}{count(w_{i-1})}$
Trigram Probability	$P(w_i\|w_{i-2},w_{i-1})=\frac{count(w_{i-2},w_{i-1},w_i)}{count(w_{i-2},w_{i-1})}$

5. **Generate text**: Use the model to predict and generate sequences of text

Consider an example

"**Corpus**"

The girl bought a chocolate
The boy ate the chocolate
The girl bought a toy
The girl played with the toy

"Vocabulary"

{the, girl, bought, a,
chocolate, boy, ate, toy,
played, with}

count (The , girl , bought)
P(bought|The , girl)= count (The , girl)
P(bought|The , girl) = 2/3 =0.67
count (The , girl, played)
P(played|The , girl) = count (The , girl)
P(played|The , girl) = 1/3 = 0.33

Limitations

- **Data sparsity**: Large number of possible n-grams, many of which may not appear in the training data.
- **Context limitation**: Limited to fixed 'n' words of context, leading to potential incoherence in longer texts.
- **Scalability**: Higher-order n-grams require significantly more data to produce accurate models.

GRAMMER

A grammar is a set of rules for generating sentences. Each rule consists of a non-terminal ex- _S, _NP, _VP and its possible expansions. Non-terminals are symbols that can be expanded further whereas Terminals are symbols that cannot be expanded further and appear in the final sentence ex- "Python", "trains". Grammars can be recursive, allowing rules to refer to them. This enables the generation of complex and varied sentences.This process involves defining rules for different parts of speech and then expanding those rules until only terminal symbols remain.

Using grammars to model language allows for the generation of diverse and potentially infinite sentences by defining a set of rules and recursively expanding them. This approach is useful in natural language processing, computational linguistics and various applications where syntactically correct sentences need to be generated.

Grammar Example

```
grammar = {
    "_S": ["_NP _VP"],
    "_NP": ["_N", "_A _NP _P _A _N"],
    "_VP": ["_V", "_V _NP"],
    "_N": ["data science", "Python", "regression"],
    "_A": ["big", "linear", "logistic"],
    "_P": ["about", "near"],
    "_V": ["learns", "trains", "tests", "is"]
}
```

In above example _S represents a sentence. _NP represents a noun phrase& _VP represents a verb phrase. _N, _A, _P, and _V represent nouns, adjectives, prepositions, and verbs respectively.

Start with the Sentence Rule: Begin with the non-terminal _S.
Expand Non-terminal _S: Replace _S with its rule, _NP _VP.
Expand _NP: Choose one of the expansions for _NP, say _N.
Replace _N: Choose a terminal for _N, say "Python".
Expand _VP: Choose one of the expansions for _VP, say _V _NP.
Replace _V: Choose a terminal for _V, say "trains".
Expand _NP: Choose one of the expansions for _NP, say _A _NP _P _A _N.
Replace Each Non-terminal: Continue replacing until all are terminals.

To implement the described approach to generating sentences using grammars, the steps are as follows:

1. Define the grammar: Create a dictionary where each key is a non-terminal and each value is a list of possible expansions.
2. Helper function to identify terminals: A terminal is a token that doesn't start with an underscore.
3. Function to expand tokens: Recursively replace non-terminal tokens with one of their possible expansions until all tokens are terminals.
4. Function to generate sentences: Start with the sentence rule and expand it using the defined functions.

CODE:

```
import random
# Define the grammar
grammar = {
   "_S"  : ["_NP _VP"],
   "_NP" : ["_N", "_A _NP _P _A _N"],
   "_VP" : ["_V", "_V _NP"],
   "_N"  : ["data science", "Python", "regression"],
   "_A"  : ["big", "linear", "logistic"],
   "_P"  : ["about", "near"],
   "_V"  : ["learns", "trains", "tests", "is"]
}
# Helper function to identify terminals
def is_terminal(token):
   return token[0] != "_"
# Function to expand tokens
def expand(grammar, tokens):
   for i, token in enumerate(tokens):
      # Skip over terminals
      if is_terminal(token):
         continue
      # Found a non-terminal token, choose a replacement at random
      replacement = random.choice(grammar[token])
      if is_terminal(replacement):
         tokens[i] = replacement
      else:
         tokens = tokens[:i] + replacement.split() + tokens[(i+1):]
      # Recursively expand the new list of tokens
      return expand(grammar, tokens)
   # If we get here, we had all terminals and are done
   return tokens

# Function to generate sentences
def generate_sentence(grammar):
   return ' '.join(expand(grammar, ["_S"]))

# Generate and print a sentence
print(generate_sentence(grammar))
```

GIBB'S SAMPLING

Gibbs sampling is a Markov Chain Monte Carlo (MCMC) technique used to generate samples from a joint distribution when the conditional distributions are known. It is a technique for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution; It's particularly useful for high-dimensional problems where direct sampling is difficult.

In many practical problems, the joint distribution of multiple variables is complex, and direct sampling is infeasible. Gibbs Sampling allows us to sample from such distributions by breaking the problem into simpler conditional distributions.

Example: Rolling Two Dice

Consider a simple example to generate samples for two dice. Let x be the value of the first die, and y be the sum of the two dice. Given the conditional distributions:

- P(y / x) is straightforward: if you know x, y can be x+1, x+2,…, x+6.
- P(x / y) is a bit more complicated but can be derived from the rules of dice.

The Gibbs sampling process involves alternating between these conditional distributions:

1. **Initialize (x, y)**: Start with any valid pair (x, y). For instance, (1, 2).or Assume initial values $X^{(0)}$ $Y^{(0)}$
2. **Update x given y**: Sample x from P(x /y).
3. **Update y given x**: Sample y from P(y/x).

I,e Sample $\boldsymbol{X^{(1)} \sim P(X/Y^{(0)})}$ and
Sample $\boldsymbol{Y^{(1)} \sim P(Y/X^{(1)})}$

- **Repeat**: Repeat steps 2 and 3 for a number of iterations.

After sufficient iterations, the values of x and y will approximate samples from the joint distribution P(x,y). This is due to the Markov property of Gibbs sampling, where each step only depends on the current state and not on the history of states.

To verify the correctness of Gibbs sampling, compare the distribution of samples obtained from Gibbs sampling with those from direct sampling. Both methods should yield similar distributions after a large number of samples, validating that Gibbs sampling is correctly approximating the joint distribution.

By comparing the counts of outcomes from Gibbs sampling and direct sampling, it is observed that Gibbs sampler converges to the same distribution as the direct sampler.

CODE:

```
import random
from collections import defaultdict

def roll_a_die():
   return random.choice([1, 2, 3, 4, 5, 6])

def direct_sample():
   d1 = roll_a_die()
   d2 = roll_a_die()
   return d1, d1 + d2

def random_y_given_x(x):
   """Equally likely to be x + 1, x + 2, ..., x + 6"""
   return x + roll_a_die()

def random_x_given_y(y):
   if y <= 7:
      # if the total is 7 or less, the first die is equally likely to be
      # 1, 2, ..., (total - 1)
      return random.randrange(1, y)
```

```
else:
    # if the total is 7 or more, the first die is equally likely to be
    # (total - 6), (total - 5), ..., 6
    return random.randrange(y - 6, 7)

def gibbs_sample(num_iters=100):
  x, y = 1, 2  # initial values, doesn't really matter
  for _ in range(num_iters):
    x = random_x_given_y(y)
    y = random_y_given_x(x)
  return x, y

def compare_distributions(num_samples=1000):
  counts = defaultdict(lambda: [0, 0])
  for _ in range(num_samples):
    counts[gibbs_sample()][0] += 1
    counts[direct_sample()][1] += 1
  return counts

# Example usage
num_samples = 1000
distribution_counts = compare_distributions(num_samples)

for outcome, (gibbs_count, direct_count) in distribution_counts.items():
  print(f"Outcome {outcome}: Gibbs = {gibbs_count}, Direct = {direct_count}")
```

Topic Modeling

Latent Dirichlet Allocation (LDA) is a powerful technique used for topic modeling, which helps us identify the underlying topics in a collection of documents. The idea is to discover the hidden particular structure in a corpus of text data. Let us consider an example based on users' stated interests.

Probabilistic Model

- **Documents and Topics**: For each document, LDA assumes that it is a mixture of topics. For each topic, it is a mixture of words.
- **Random Variables**: LDA involves two sets of random variables:
  - θd: Distribution of topics for document d.
  - ϕk: Distribution of words for topic k.

LDA Process

1. **Initialization**: Assign each word in each document a topic randomly.
2. **Gibbs Sampling**: Iterate over each word in each document to update its topic assignment based on conditional probabilities.

   This involves two steps:
   - Calculate the weight for each topic based on current topic distributions in the document and word distributions in the topic.
   - Sample a new topic for the word based on these weights.

Consider the following documents (users' interests):

```
documents = [
   ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
   ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
   ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
   ["R", "Python", "statistics", "regression", "probability"],
   ["machine learning", "regression", "decision trees", "libsvm"],
   ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
   ["statistics", "probability", "mathematics", "theory"],
   ["machine learning", "scikit-learn", "Mahout", "neural networks"],
   ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
   ["Hadoop", "Java", "MapReduce", "Big Data"],
   ["statistics", "R", "statsmodels"],
   ["C++", "deep learning", "artificial intelligence", "probability"],
   ["pandas", "R", "Python"],
   ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
   ["libsvm", "regression", "support vector machines"]

]
```

Aim to find K=4 topics.

Document-Topic Counts - Track how many times each topic is assigned to each document.
Topic-Word Counts - Track how many times each word is assigned to each topic.
Topic Counts - Track the total number of words assigned to each topic.
Document Lengths - Track the total number of words in each document.
Conditional probabilities are calculated

Probability of Topic Given Document:

$$\textbf{P(topic/document)} \approx \frac{(\textbf{count of topic in document}+\alpha)}{\textbf{total words in document}+ \textbf{K}.\alpha}$$

Probability of Word Given Topic:

$$\textbf{P(word/topic)} \approx \frac{(\textbf{count of word in topic}+\beta)}{\textbf{total words assiged to topic}+ \textbf{W}.\beta}$$

Here, α **and** β **are smoothing parameters to ensure non-zero probabilities.**

CODE:

```
import random
from collections import Counter
import numpy as np

def sample_from(weights):
   total = sum(weights)
   rnd = total * random.random()
   for i, w in enumerate(weights):
      rnd -= w
      if rnd <= 0:
         return i
```

```
def p_topic_given_document(topic, d, alpha=0.1):
    return (document_topic_counts[d][topic] + alpha) / (document_lengths[d] + K * alpha)

def p_word_given_topic(word, topic, beta=0.1):
    return (topic_word_counts[topic][word] + beta) / (topic_counts[topic] + W * beta)

def topic_weight(d, word, k):
    return p_word_given_topic(word, k) * p_topic_given_document(k, d)

def choose_new_topic(d, word):
    return sample_from([topic_weight(d, word, k) for k in range(K)])

# Initialize variables
random.seed(0)
K = 4
document_topic_counts = [Counter() for _ in documents]
topic_word_counts = [Counter() for _ in range(K)]
topic_counts = [0 for _ in range(K)]
document_lengths = list(map(len, documents))
distinct_words = set(word for document in documents for word in document)
W = len(distinct_words)
D = len(documents)

# Randomly assign topics to words
document_topics = [[random.randrange(K) for word in document] for document in documents]
for d in range(D):
    for word, topic in zip(documents[d], document_topics[d]):
        document_topic_counts[d][topic] += 1
        topic_word_counts[topic][word] += 1
        topic_counts[topic] += 1

# Perform Gibbs sampling
for iter in range(1000):
    for d in range(D):
        for i, (word, topic) in enumerate(zip(documents[d], document_topics[d])):
            document_topic_counts[d][topic] -= 1
            topic_word_counts[topic][word] -= 1
            topic_counts[topic] -= 1
            document_lengths[d] -= 1

            new_topic = choose_new_topic(d, word)
            document_topics[d][i] = new_topic

            document_topic_counts[d][new_topic] += 1
            topic_word_counts[new_topic][word] += 1
            topic_counts[new_topic] += 1
            document_lengths[d] += 1

# Output the topics and their most common words
for k, word_counts in enumerate(topic_word_counts):
    print(f"Topic {k}:")
    for word, count in word_counts.most_common():
        if count > 0:
            print(f"{word}: {count}")
```

```
    print()

# Assign topic names based on the most common words
topic_names = ["Big Data and programming languages", "Python and statistics", "databases", "machine
learning"]

# Print the topic distribution for each document
for document, topic_counts in zip(documents, document_topic_counts):
    print(f"Document: {document}")
    for topic, count in topic_counts.most_common():
        if count > 0:
            print(f"{topic_names[topic]}: {count}")
    print()
```

Network Analysis

Network analysis in data science involves examining the structure and dynamics of networks. Networks, also known as graphs, consist of nodes (or vertices) and edges (or links) that represent relationships or interactions between entities. Network analysis provides insights into the patterns, connections, and behaviors within the network, which can be applied to various fields such as social sciences, biology, computer science, and more. Network analysis provides a powerful framework for understanding and leveraging the relationships and interactions within complex data. It enables data scientists to gain deeper insights, make informed decisions, and solve problems that involve interconnected entities. Network analysis in data science is crucial because it allows us to understand and model the relationships and interactions between different entities.

Networks naturally represent complex relationships between entities, such as:

- Social Networks: Relationships between individuals on social media.
- Biological Networks: Interactions between proteins, genes, or species.
- Communication Networks: Connections between devices in a telecommunication network.

Centrality is a key concept in network theory and graph analysis, which focuses on identifying the most important or influential nodes within a network. Centrality can be measured in several ways, each capturing different aspects of a node's importance:

1. **Degree Centrality**: Measures the number of direct connections a node has. Nodes with higher degrees are considered more central.
2. **Betweenness Centrality**: Quantifies the number of times a node acts as a bridge along the shortest path between two other nodes. Nodes with high betweenness centrality play a critical role in the flow of information within the network.
3. **Eigenvector Centrality**: Measures a node's influence based on the number and quality of its connections. A node is considered central if it is connected to other nodes that are themselves central. It is often used in identifying influential nodes in social networks.
4. **PageRank**: A variant of eigenvector centrality, originally used by Google to rank web pages in search results. It measures the probability that a random walker will land on a particular node, considering both the quantity and quality of incoming links.

Each centrality measure provides different insights into the network structure and the roles of individual nodes. The choice of centrality measure depends on the specific context and the aspect of importance being investigated.

Betweenness Centrality

Betweenness centrality is a measure of centrality in a graph based on the shortest paths. It quantifies the number of times a node acts as a bridge along the shortest path between two other nodes. Nodes with high betweenness centrality are crucial for the flow of information through the network.

Definition

For a given graph G= (V, E), where V is the set of vertices and E is the set of edges:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- σst is the total number of shortest paths from node s to node t.
- σst(v) is the number of those shortest paths that pass through v.

Steps to calculate betweenness centrality for the DataSciencester network:

1. Setup the Users and Friendships:

- Define the users and their friendships.
- Add lists to each user to track their friends.

Code:

```
from collections import deque

# Define users
users = [
   { "id": 0, "name": "Hero" },
   { "id": 1, "name": "Dunn" },
   { "id": 2, "name": "Sue" },
   { "id": 3, "name": "Chi" },
   { "id": 4, "name": "Thor" },
   { "id": 5, "name": "Clive" },
   { "id": 6, "name": "Hicks" },
   { "id": 7, "name": "Devin" },
   { "id": 8, "name": "Kate" },
   { "id": 9, "name": "Klein" }
]

# Define friendships
friendships = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
          (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]

# Add friends lists to each user
for user in users:
   user["friends"] = []

for i, j in friendships:
   users[i]["friends"].append(users[j]) # add j as a friend of i
   users[j]["friends"].append(users[i]) # add i as a friend of j
```

2. Compute the Shortest Paths:

- Use Breadth-First Search (BFS) to find the shortest paths from a given user to every other user.
- Store all the shortest paths in a dictionary.

Code:

```
def shortest_paths_from(from_user):
    shortest_paths_to = { from_user["id"]: [[]] }
    frontier = deque((from_user, friend) for friend in from_user["friends"])

    while frontier:
        prev_user, user = frontier.popleft()
        user_id = user["id"]

        paths_to_prev_user = shortest_paths_to[prev_user["id"]]
        new_paths_to_user = [path + [user_id] for path in paths_to_prev_user]

        old_paths_to_user = shortest_paths_to.get(user_id, [])

        if old_paths_to_user:
            min_path_length = len(old_paths_to_user[0])
        else:
            min_path_length = float('inf')

        new_paths_to_user = [path for path in new_paths_to_user if len(path) <= min_path_length and path not
in old_paths_to_user]

        shortest_paths_to[user_id] = old_paths_to_user + new_paths_to_user

        frontier.extend((user, friend) for friend in user["friends"] if friend["id"] not in shortest_paths_to)

    return shortest_paths_to

for user in users:
    user["shortest_paths"] = shortest_paths_from(user)
```

3. Compute Betweenness Centrality:

- For each pair of nodes, find all shortest paths between them.
- For each node on these paths (excluding the source and target), update its betweenness centrality score.

Code:

```
# Initialize betweenness centrality for each user
for user in users:
    user["betweenness_centrality"] = 0.0

# Compute betweenness centrality
for source in users:
    source_id = source["id"]
    for target_id, paths in source["shortest_paths"].items():
        if source_id < target_id:
```

```
        num_paths = len(paths)
        contrib = 1 / num_paths
        for path in paths:
            for id in path:
                if id not in [source_id, target_id]:
                    users[id]["betweenness_centrality"] += contrib

# Print the results
for user in users:
    print(f"User {user['name']} has betweenness centrality: {user['betweenness_centrality']:.4f}")
```

Eigenvector centrality

Eigenvector centrality is a measure of the influence of a node in a network. Unlike other centrality measures, it assigns relative scores to all nodes in the network based on the concept that connections to high-scoring nodes contribute more to the score of the node in question. It Measures a node's influence based on the number and quality of its connections. It is often used in identifying influential nodes in social networks

Example:

Consider a simple network with four nodes A, B, C, and D, with the following adjacency matrix A

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

This adjacency matrix represents the following graph:

- A is connected to B and C.
- B is connected to A, C, and D.
- C is connected to A, B, and D.
- D is connected to B and C

For a given adjacency matrix the Eigenvalues and Eigenvectors are

For matrix A, solve the characteristic equation $\det(A-\lambda I)=0$

Eigenvalues: $\lambda 1=2$, $\lambda 2=-1$, $\lambda 3=-1$, $\lambda 4=0$.

Eigenvector for $\lambda 1=2$

$$v_1 = \begin{bmatrix} 0.5 \\ 0.7071 \\ 0.5 \\ 0.5 \end{bmatrix}$$

Normalize the Principal Eigenvector**:**

$$v_1 = \begin{bmatrix} 0.2989 \\ 0.4226 \\ 0.2989 \\ 0.2989 \end{bmatrix}$$

The eigenvector centrality scores for nodes A, B, C, and D are approximately 0.2989, 0.4226, 0.2989, and 0.2989, respectively. Node B has the highest centrality score, indicating it is the most influential node in this network.

Code:

```
import numpy as np

def normalize(v):
    norm = np.linalg.norm(v)
    if norm == 0:
       return v
    return v / norm

def eigenvector_centrality(A, tolerance=1e-6, max_iter=1000):
    n = len(A)
    v = np.random.rand(n)
    v = normalize(v)

    for _ in range(max_iter):
        v_new = np.dot(A, v)
        v_new = normalize(v_new)

        if np.linalg.norm(v - v_new) < tolerance:
            return v_new

        v = v_new

    raise Exception("Eigenvector centrality did not converge")

# Example usage:
A = np.array([
    [0, 1, 1, 0],
    [1, 0, 1, 1],
    [1, 1, 0, 1],
    [0, 1, 1, 0]
])

centrality = eigenvector_centrality(A)
print("Eigenvector Centrality:", centrality)
```

PageRank

The PageRank algorithm, introduced by Google, evaluates the importance of nodes in a directed graph based on their connections. This is especially useful in social networks, where endorsements can signify the respect or influence of a person within a community.
The PageRank algorithm is a method for ranking nodes in a directed graph, such as webpages in a network of hyperlinks. The algorithm assigns a numerical weight to each node, representing the likelihood that a person randomly clicking on links will arrive at that particular node.
Algorithm

- Each node starts with an equal PageRank value.
- The total PageRank across all nodes is normalized to 1.
- PageRank is distributed from each node to its outbound links.
- A damping factor is used to simulate the probability that a person will continue clicking on links (typically set to 0.85).
- The algorithm iterates until the PageRank values stabilize (i.e., change very little between iterations).

The main idea behind PageRank is that a page is considered important if many other important pages link to it. The algorithm works as follows:

1. Every page is assigned an initial rank. If there are N pages, each page is typically assigned a rank of 1/N.
2. The rank of a page is determined by the ranks of the pages linking to it. If a page P has incoming links from pages $A_1,A_2,...,A_m$ with ranks. $R(A_1),R(A_2),...,R(A_m)$, and if each page A_i has $L(A_i)$ outgoing links, then the rank of page P is calculated as:

$$R(P) = (1 - d) + d\left(\sum_{i=1}^{m} \frac{R(A_i)}{L(A_i)}\right)$$

   Here, d is a damping factor, usually set to 0.85, which accounts for the probability that a user continues clicking on links rather than starting a new search.

3. The rank values are updated iteratively until they converge to a stable set of values.

PageRank revolutionized web search by providing a way to rank pages based on their relevance and importance rather than just keyword matching.

Recommender Systems

A recommendation system is a type of information filtering system that predicts and suggests items or content to users based on various criteria. These systems are widely used in various domains such as e-commerce, streaming services, social media, and content websites to enhance user experience by providing personalized recommendations.
Recommending what's popular is a straightforward approach in recommendation systems. This method involves identifying the most popular items based on user interactions and then recommending these items to users.

Steps:

- **Collect Data:** Gather data on user interactions with items. For example, this could be a list of items each user is interested in.
- **Calculate Popularity:** Count how often each item appears in the list of user interests.
- **Sort by Popularity:** Sort the items by their popularity
- **Recommend Popular Items:** Recommend the top N popular items that a user has not already interacted with.

Code:

```
from collections import Counter
from typing import List, Tuple

users_interests = [
   ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
   ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
   ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
   ["R", "Python", "statistics", "regression", "probability"],
   ["machine learning", "regression", "decision trees", "libsvm"],
   ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
   ["statistics", "probability", "mathematics", "theory"],
   ["machine learning", "scikit-learn", "Mahout", "neural networks"],
   ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
   ["Hadoop", "Java", "MapReduce", "Big Data"],
   ["statistics", "R", "statsmodels"],
   ["C++", "deep learning", "artificial intelligence", "probability"],
   ["pandas", "R", "Python"],
   ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
   ["libsvm", "regression", "support vector machines"]
]

popular_interests_sorted = popular_interests.most_common()
print(popular_interests_sorted)

def most_popular_new_interests(user_interests: List[str], max_results: int = 5) -> List[Tuple[str, int]]:
   # Filter out interests the user already has
   suggestions = [(interest, frequency) for interest, frequency in popular_interests_sorted if interest not in
user_interests]
   return suggestions[:max_results]

# Example usage:
user_id = 1  # User with interests ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"]
recommendations = most_popular_new_interests(users_interests[user_id], 5)
print(f"Recommendations for user {user_id}: {recommendations}")
```

Types of recommendations

The system uses two approaches– **content-based filtering** and **collaborative filtering-** to make recommendations.

Content-Based Filtering

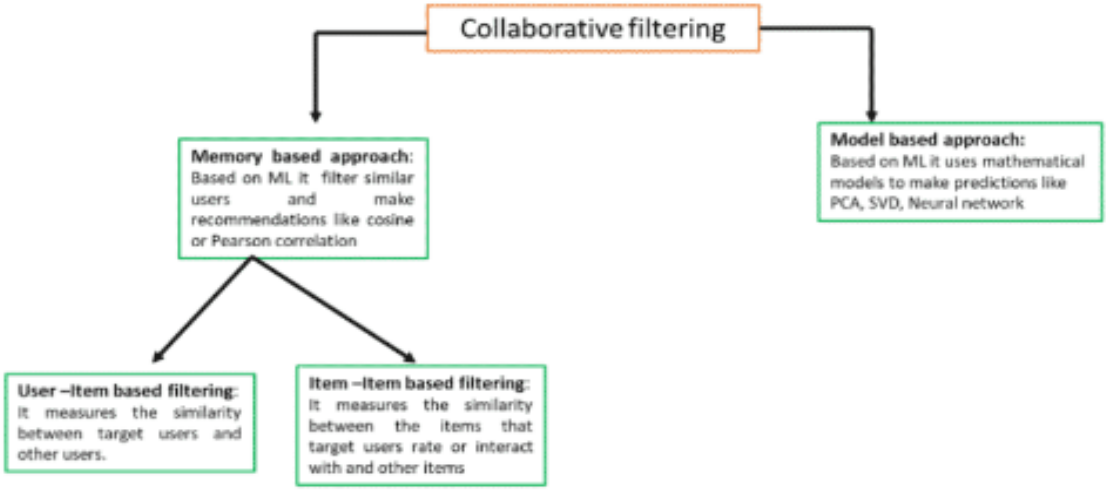
This approach recommends items based on user preferences. It matches the requirement,considering the past actions of the user, patterns detected, or any explicit feedback provided by the user, and accordingly, makes a recommendation.

Example: If a person prefer the chocolate flavor and purchase a chocolate ice cream, the next time he raise a query, the system shall scan for options related to chocolate, and then, recommend you to try a chocolate cake.

Collaborative Filtering

This approach uses similarities between users and items simultaneously, to provide recommendations. It is the idea of recommending an item or making a prediction, depending on other like-minded individuals. It could comprise a set of users, items, opinions about an item, ratings, reviews, or purchases.

Example: Suppose Persons A and B both like the chocolate flavor and have them have tried the ice-cream and cake, then if Person A buys chocolate biscuits, the system will recommend chocolate biscuits to Person B.



User-Based Collaborative Filtering

User-Based Collaborative Filtering is a technique used to predict the items that a user might like on the basis of ratings given to that item by other users who have similar taste with that of the target user. Many websites use collaborative filtering for building their recommendation system.

Steps for User-Based Collaborative Filtering:

1. **Collect Unique Interests**:

```
unique_interests = sorted(list({ interest for user_interests in users_interests for interest in
user_interests }))
```

2. **Create User Interest Vectors:**

```
def make_user_interest_vector(user_interests):
    return [1 if interest in user_interests else 0 for interest in unique_interests]
user_interest_matrix = map(make_user_interest_vector, users_interests)
```

3. **Compute Pairwise User Similarities:**

```
user_similarities = [[cosine_similarity(interest_vector_i, interest_vector_j)
                      for interest_vector_j in user_interest_matrix]
                     for interest_vector_i in user_interest_matrix]
```

4. **Find Most Similar Users:**

```
def most_similar_users_to(user_id):
    pairs = [(other_user_id, similarity)
             for other_user_id, similarity in enumerate(user_similarities[user_id])
             if user_id != other_user_id and similarity > 0]
    return sorted(pairs, key=lambda (_, similarity): similarity, reverse=True)
```

5. **Suggest New Interests:**

```
def user_based_suggestions(user_id, include_current_interests=False):
    suggestions = defaultdict(float)
    for other_user_id, similarity in most_similar_users_to(user_id):
        for interest in users_interests[other_user_id]:
            suggestions[interest] += similarity
    suggestions = sorted(suggestions.items(), key=lambda (_, weight): weight, reverse=True)
    if include_current_interests:
        return suggestions
    else:
        return [(suggestion, weight) for suggestion, weight in suggestions if suggestion not in
users_interests[user_id]]
```

6. **Users and Interests**

```
users_interests = [
    ["Hadoop", "Big Data", "HBase", "Java", "Spark", "Storm", "Cassandra"],
    ["NoSQL", "MongoDB", "Cassandra", "HBase", "Postgres"],
    ["Python", "scikit-learn", "scipy", "numpy", "statsmodels", "pandas"],
    ["R", "Python", "statistics", "regression", "probability"],
    ["machine learning", "regression", "decision trees", "libsvm"],
    ["Python", "R", "Java", "C++", "Haskell", "programming languages"],
    ["statistics", "probability", "mathematics", "theory"],
    ["machine learning", "scikit-learn", "Mahout", "neural networks"],
    ["neural networks", "deep learning", "Big Data", "artificial intelligence"],
    ["Hadoop", "Java", "MapReduce", "Big Data"],
```

```
        ["statistics", "R", "statsmodels"],
        ["C++", "deep learning", "artificial intelligence", "probability"],
        ["pandas", "R", "Python"],
        ["databases", "HBase", "Postgres", "MySQL", "MongoDB"],
        ["libsvm", "regression", "support vector machines"]
    ]
```

Item -Based Collaborative Filtering

Item-Based Collaborative Filtering is a recommendation system technique that focuses on the similarity between items to make recommendations to users. Instead of finding similar users, it finds items similar to the ones a user has shown interest in.

Steps in Item-Based Collaborative filtering

1. Create the User-Item Matrix: Collect data on user interactions with items.

```
user_interest_matrix = [

        [1, 0, 1],  # User 0
        [0, 1, 0],  # User 1
        [1, 1, 0]   # User 2
    ]
unique_interests = ["Interest A", "Interest B", "Interest C"]
```

2. **Compute Item Similarities**: Calculate the similarity score for each pair of items.

```
import numpy as np
    def cosine_similarity(v1, v2):
    dot_product = np.dot(v1, v2)
    norm_v1 = np.linalg.norm(v1)
    norm_v2 = np.linalg.norm(v2)
    return dot_product / (norm_v1 * norm_v2)
```

3. **Compute Item Similarities:** Store these scores in an item-item similarity matrix

```
def calculate_similarities(matrix):
    num_items = len(matrix[0])
    similarities = np.zeros((num_items, num_items))

    for i in range(num_items):
        for j in range(num_items):
            if i != j:
                similarities[i][j] = cosine_similarity(matrix[:, i], matrix[:, j])

    return similarities

interest_user_matrix = np.array(user_interest_matrix).T
interest_similarities = calculate_similarities(interest_user_matrix).
```

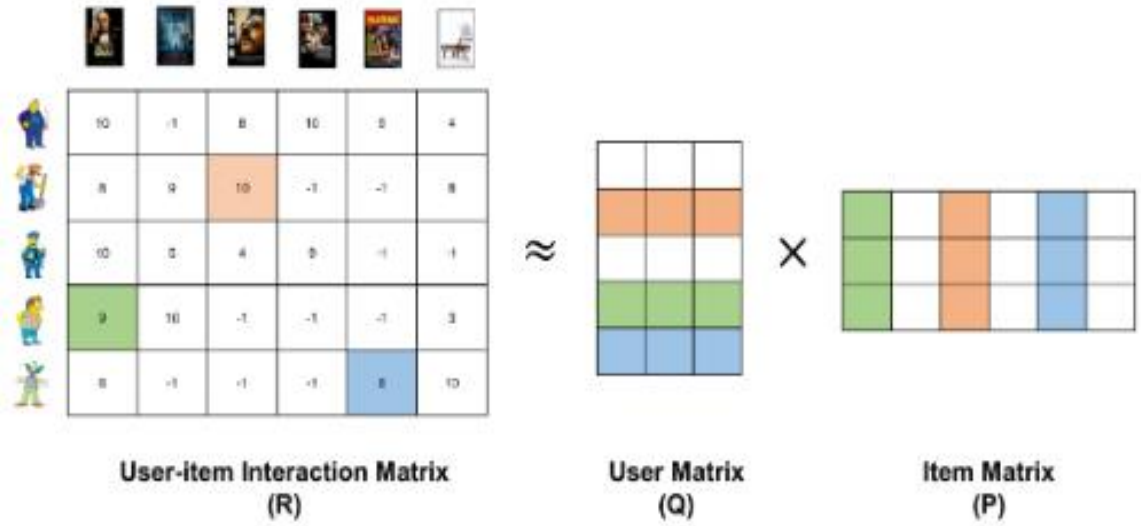
4. Generate Recommendations:

For a given user, identify items similar to the items they have interacted with.Aggregate these similar items to form a ranked list of recommendations.

```
def item_based_suggestions(user_id, user_interest_matrix, interest_similarities, unique_interests):

    user_interests = user_interest_matrix[user_id]
    scores = np.zeros(len(user_interests))

for i, interested in enumerate(user_interests):

        if interested:
          scores += interest_similarities[i]

recommendations = [(unique_interests[i], score) for i, score in enumerate(scores) if user_interests[i] == 0]

    recommendations.sort(key=lambda x: x[1], reverse=True)

    return recommendations

recommendations = item_based_suggestions(0, user_interest_matrix, interest_similarities,
unique_interests)
print(recommendations)
```

Matrix Factorization

- Matrix factorization is one of the most sought-after machine learning recommendation models.
- It acts as a catalyst, enabling the system to gauge the customer's exact purpose of the purchase, scan numerous pages, shortlist, and rank the right product or service, and recommend multiple options available.
- Once the output matches the requirement, the lead translates into a transaction and the deal clicks.
- Once an individual raises a query on a search engine, the machine deploys uses matrix factorization to generate an output in the form of recommendations.
- Matrix factorization is an extensively used technique in collaborative filtering recommendation systems.
- Its objective is to factorize a user-item matrix into two low-ranked matrices, the user-factor matrix and the item-factor matrix that can predict new items that users might be interested in.



If A is a $n_1 \times k_1$ matrix and B is a $n_2 \times k_2$ matrix, and if $k_1 = n_2$, then their product AB is the $n_1 \times k_2$ matrix whose (i,j)th entry is:

$$A_{i1}B_{1j} + A_{i2}B_{2j} + \cdots + A_{ik}B_{kj}$$

Which is just the dot product of the i_{th} row of A with the j_{th} column of B :

```
def matrix_product_entry(A, B, i, j):
return dot(get_row(A, i), get_column(B, j))


def matrix_multiply(A, B):
n1, k1 = shape(A)
n2, k2 = shape(B)
if k1 != n2:
raise ArithmeticError(&amp;quot;incompatible shapes!&amp;quot;)
return make_matrix(n1, k2, partial(matrix_product_entry, A, B))
```

Notice that if A is a n X K matrix and B is a k X 1 matrix, then AB is a n X 1 matrix.

If the vector consider as a one-column matrix and A as a function that maps k dimensional vectors to n-dimensional vectors, where the function is just matrix multiplication.
I,e

```
v = [1, 2, 3]
v_as_matrix = [[1],
               [2],
               [3]]
```

Therefore

```
def vector_as_matrix(v):
"""returns the vector v (represented as a list) as a n x 1 matrix"""
return [[v_i] for v_i in v]
```

```
def vector_from_matrix(v_as_matrix):
"""returns the n x 1 matrix as a list of values"""
return [row[0] for row in v_as_matrix]
```

matrix operation using matrix_multiply:

```
def matrix_operate(A, v):
v_as_matrix = vector_as_matrix(v)
product = matrix_multiply(A, v_as_matrix)
return vector_from_matrix(product)
```

When A is a square matrix, this operation maps n-dimensional vectors to other n dimensional vectors.