Arrays Datastructure

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

In [ ]:
```
names = np.array(['Claire', 'Darrin', 'Sean', 'Brosina', 'Andrew', 'Irene', 'Harold
ages = np.array([35, 26, 36, 44, 33, 33, 23, 22, 35, 53, 25, 41, 24, 31, 34, 43, 22
salary = np.array([ 88962,  67659, 117501, 149957,  32212,  63391,  14438,  22445,
designation = np.array(['Manager', 'Team Lead', 'Manager', 'Senior Manager', 'Team
```

Indexing and Slicing on arrays

In [ ]:
```
arr = np.random.randint(5, 50, 10)
arr
```

In [ ]:
```
arr[2]
```

In [ ]:
```
arr[-2]
```

In [ ]:
```
arr[0:3]
```

In [ ]:
```
arr[-5:]
```

In [ ]:
```
arr[[3, 1, 6]]
```

In [ ]:
```
arr = np.random.randint(5, 50, (4, 3))
arr
```

In [ ]:
```
arr[1, 2]
```

In [ ]:
```
arr[0:2, :] # first 2 rows
```

In [ ]:
```
arr[[1, 3], :] # Row index 1 and 3 and all columns
```

In [ ]:
```
arr[[1, 3]] # : for columns is not mandatory here
```

In [ ]:
```
arr[ :, 1:3] # : for rows is mandatory here
```

Extract the ages greater than 40

In [ ]:
```
ages[ages > 40] # Conditional Indexing
```

Operations on Arrays

Extract names of the employees whose age greater than 40

In [ ]:
```
names[ages > 40]
```

Apply 7% hike on salary to all employes in the array

In [ ]:
```
salary * 1.07
```

Apply 7% hike to all employes whose age is greater_eq than 40 and 10% hike to employees whose age is less than 40

In [ ]:
```
np.where(ages >= 40, salary * 1.07, salary * 1.10)
```

Ex. Applying multiple conditions

In [ ]:
```
conditions = [ages < 35, ages < 45, ages >= 45]
results = [salary * 1.10, salary * 1.07, salary * 1.05]

np.select(conditions, results)
```

In [ ]:
```
marks = np.random.randint(20, 100,(2, 3)) # 2D array
marks
```

Ex. Assign grades to students based on marks

In [ ]:
```
marks = np.random.randint(20, 100, 15)
marks
```

In [ ]:
```
conditions = [marks >= 75, marks >= 60, marks > 40, marks < 40]
# grades = ["Grade A", "Grade B", "Grade C", "Grade D"]
grades = [1, 2, 3, 4]
np.select(conditions, grades)
```

In [ ]:
```
conditions = [marks >= 75, marks >= 60, marks > 40, marks < 40]
grades = ["Grade A", "Grade B", "Grade C", "Grade D"]
np.select(conditions, grades, default= "NA")
```

In [ ]:
```
conditions = [marks >= 75, marks >= 60, marks >= 40]
grades = ["Grade A", "Grade B", "Grade C"]
np.select(conditions, grades, default= "Grade D")
```

In [ ]:
```
help(np.select)
```

Utility functions in NumPy Library

- np.all(), np.any()
- arr.sum()
- arr.min(), arr.max(), arr.argmin(), arr.agrmax()
- np.round()
- np.mean(), np.median(), np.average(), np.percentile()

In [ ]:
```
products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling', 'Decaf
sales = np.array([52248.0, 14068.0, 71060.0, 60014.0, 69925.0, 27711.0, 19231.0, 24
```

```
profits = np.array([17444.0, 5041.0, 28390.0, 20459.0, 23432.0, 7691.0, -2954.0, 71
```

Ex. Total Sales

In [ ]:
```
sales.sum()
```

In [ ]:
```
np.sum(sales)
```

Ex. Avg sales

In [ ]:
```
sales.mean()
```

Ex. Identify the products generating sales more than the average

In [ ]:
```
sales_p = products[sales >= sales.mean()]
sales_p
```

Ex. Identify the products generating profits more than the average

In [ ]:
```
profits_p = products[profits >= profits.mean()]
profits_p
```

Ex. Do all products that surpass their average sales also exceed their average profit? If not, identify the products that don't.

In [ ]:
```
np.isin(sales_p, profits_p).all()
```

In [ ]:
```
np.any(np.isin(sales_p, profits_p))
```

In [ ]:
```
all([1, 2, 3, 4, 0]) # checks if all values are true
```

In [ ]:
```
any([1, 2, 3, 4, 0]) # checks if any one value is true
```

Ex. Manipulating the results to explore set function on arrays

In [ ]:
```
sales_p = np.append(sales_p, "Green Tea")  # np.append() is a function
sales_p
```

In [ ]:
```
profits_p = np.insert(profits_p, 2, "Filter Coffee")  # np.insert() is a function
profits_p
```

In [ ]:
```
profits_p = np.delete(profits_p, 6)  # np.delete() is a function to remove
profits_p
```

In [ ]:
```
np.isin(sales_p, profits_p)
```

Ex. Find elements in sales_p but not in profit_p

In [ ]:
```
np.setdiff1d(sales_p, profits_p)
```

Ex. Common elements

In [ ]:
```
np.intersect1d(sales_p, profits_p)
```

Ex. Identify the product with maximum Sales

In [ ]:
```
# Option 1
products[np.max(sales) == sales] # Conditional indexing
```

In [ ]:
```
np.argmax(sales) # returns index position of the max value
```

In [ ]:
```
# Option 2
products[np.argmax(sales)] # Indexing
```

In [ ]:
```
arr = np.array([2, 3, 5, 5 ])
names = np.array(["Jane", "George", "Rosie", "Sam"])
arr.max()
```

In [ ]:
```
arr.argmax()
```

In [ ]:
```
names[arr.argmax()]
```

In [ ]:
```
names[arr == np.max(arr)]
```

In [ ]:
```
arr = np.random.randint(1, 10, (3, 5))
arr
```

In [ ]:
```
np.insert(arr, 5, 1)
```

Flattening the array

In [ ]:
```
arr.flatten() # converted 2D array to 1D
```

In [ ]:
```
np.insert(arr, 1, [1, 2, 3], axis=1)
```

In [ ]:
```
np.insert(arr, 1, [1, 2, 3, 4, 5], axis=0)
```

Basics of Data Visualisation using Matplotlib

In [ ]:
```
import numpy as np
import matplotlib.pyplot
dates = np.arange('2019-01', '2022-01', dtype='datetime64[M]')
sales = np.array([42390, 77560, 77385, 76039, 42968, 53833, 47205, 68936, 51175, 48
profits = np.array([ 7206.3 ,  8531.6 , 13155.45,  9885.07,  7304.56,  9689.94, 566
```

Line Chart

In [ ]:
```
plt.figure(figsize=(10, 3))
plt.plot(dates, sales, color = "Teal", marker = "o")

plt.title("Sales over Months ", loc = "left",
          fontdict= {'fontsize': 10, 'color': "Blue"})

y_tcks = np.arange((sales.min() - 1000), (sales.max() + 1000), 5000)

plt.xlabel("Months")
plt.ylabel("Sales")

plt.yticks(y_tcks)

plt.grid(axis="y", color = "orange", alpha = 0.4, ls = "--")

# Adding Annotations---
# for x, y in zip(dates, sales):
#     plt.annotate(f'{y}', xy=(x, y))

# Adding Annotations only for max value
plt.annotate(f"Max : {np.max(sales)}", xy = (dates[np.argmax(sales)], np.max(sales)

plt.show()
```

In [ ]:
```
plt.figure(figsize=(10, 3))
plt.plot(dates, sales, color = "Teal", marker = "o")
plt.plot(dates, profits, color = "Cyan", marker = "o")
plt.title("Sales over Months ", loc = "left", fontdict= {'fontsize': 10, 'color': "
plt.xlabel("Months")
plt.ylabel("Sales")
plt.grid(axis="y", color = "orange", alpha = 0.4, ls = "--")

plt.show()
```

In [ ]:
```
fig, ax = plt.subplots(nrows=2, figsize = (12, 4))
ax[0].plot(dates, sales, color = "Teal", marker = "o")
ax[1].plot(dates, profits, color = "Cyan", marker = "o")

ax[0].set_title("Sales", loc = "left")
ax[1].set_title("Profit", loc = "left")

ax[0].set_xticks([])

plt.show()
```

In [ ]:
```
products = np.array(['Caffe Latte', 'Cappuccino', 'Colombian', 'Darjeeling', 'Decaf
sales = np.array([52248.0, 14068.0, 71060.0, 60014.0, 69925.0, 27711.0, 19231.0, 24
profits = np.array([17444.0, 5041.0, 28390.0, 20459.0, 23432.0, 7691.0, -2954.0, 71
```

Bar Chart

In [ ]:
```
np.sort(sales)
```

In [ ]:
```
sort_ord = np.argsort(sales)[::-1]
```

In [ ]:
```
plt.figure(figsize=(10, 2))
plt.bar(products[sort_ord], sales[sort_ord], color = "teal", edgecolor = "black")
plt.title("Sales across Products", loc = "left",fontdict= {'fontsize': 10, 'color':
plt.xticks(rotation = 20, fontsize='x-small')
plt.yticks(fontsize='x-small')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(10, 2))
plt.bar(products, sales, color = "teal", edgecolor = "black")
plt.bar(products, profits, color = "cyan", edgecolor = "black")
plt.title("Sales across Products", loc = "left",fontdict= {'fontsize': 10, 'color':
plt.xticks(rotation = 20, fontsize='x-small')
plt.yticks(fontsize='x-small')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(10, 2))
plt.bar(products, sales, color = "teal", edgecolor = "black", width = -0.4, align="
plt.bar(products, profits, color = "cyan", edgecolor = "black", width = 0.4, align=
plt.title("Sales across Products", loc = "left",fontdict= {'fontsize': 10, 'color':
plt.xticks(rotation = 20, fontsize='x-small')
plt.yticks(fontsize='x-small')
plt.show()
```

Data Manipulation using DataFrames

In [ ]:
```
!pip install sqlalchemy
```

Ex. Read data from employee.sqlite3

In [ ]:
```
from sqlalchemy import create_engine, text
conn = create_engine("sqlite:///employee.sqlite3")
conn
```

Ex. Read data from Employee table

In [ ]:
```
df = pd.read_sql("Employee", conn)
df.head(2)
```

Ex. Filter all managers from the data and add to the database as new table

In [ ]:
```
df_1 = df[df.Designation == "Manager"]
df_1.to_sql("Managers", conn)
```

Ex. Read data from new Manager table

In [ ]:
```
df = pd.read_sql("Managers", conn)
```

Ex. Data Cleaning - remove index col

In [ ]:
```
df.drop(columns=["index"], inplace=True)
df.head()
```

Indexing Columns

In [ ]:
```
df.head()  # first n rows n = 5 default
```

In [ ]:
```
df.tail()  # last n rows n = 5 default
```

In [ ]:
```
df.shape
```

In [ ]:
```
df.dtypes
```

Ex. Extraxt Salary col

In [ ]:
```
df.Salary
```

In [ ]:
```
df["Salary"]
```

Ex. Extract Name and Salary

In [ ]:
```
df[["Name", "Salary"]].head(2)
```

Filtering Data

Ex. Find avg salary of managers

In [ ]:
```
(df.Designation == "Manager").any()
```

In [ ]:
```
df[df.Designation == "Manager"]
```

In [ ]:
```
df[df.Designation == "Manager"].Salary.mean().round(2)
```

Ex. Are there any employee who age is more than 40 years?

In [ ]:
```
(df.Age >= 40).any()
```

Ex. How many employees have age > 40?

In [ ]:
```
(df.Age >= 40).sum()
```

Ex. Name the employees whoes age is > 40

In [ ]:
```
df[df.Age >= 40].Name  # Returns series object
```

In [ ]:
```
df[df.Age >= 40].Name.to_list()
```

Ex. Extract employees whoes age is in range of 25-45 and give the count

In [ ]:
```
(df.Age.between(25, 45)).sum()
```

In [ ]:
```
df[df.Age.between(25, 45)].shape[0]
```

In [ ]:
```
print(df[df.Age.between(25, 45)].Name.to_list())
```

Ex. Give names of employees who are Managers and Team Leaders

In [ ]:
```
df[df.Designation.isin(("Manager", "Team Lead"))].Name
```

Ex. Give list employees who are Managers but age is less than 40

In [ ]:
```
np.logical_and(df.Age < 40, df.Designation == "Manager").sum()
```

In [ ]:
```
df[np.logical_and(df.Age < 40, df.Designation == "Manager")].Name
```

Data Cleaning

Ex. Read data from excel file

In [ ]:
```
pd.read_excel("coffee_sales.xlsx", sheet_name="Sheet1")  # just for reference
```

Ex. Read data from csv file

In [ ]:
```
df = pd.read_csv("coffee_sales.csv", header=3)
df.dropna(axis=1, how="all", inplace=True)
df.dropna(axis=0, how="all", inplace=True)
df.head()
```

In [ ]:
```
obj = str.maketrans("", "", "$,")
df["Sales"] = df["Sales"].str.translate(obj).astype(float)
df["Target Sales"] = df["Target Sales"].str.translate(obj).astype(float)
df["Profit"] = df["Profit"].str.translate(obj).astype(float)
df["Target Profit"] = df["Target Profit"].str.translate(obj).astype(float)
```

In [ ]:
```
df.head(2)
```

Operations on DataFrame

- Adding new column by calculation
- Sorting and Ranking
- map(), replace(), apply()

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

In [ ]:

Explolatory Data Analysis

Types of data

Numerical (Quantitative) Data:

Numerical data consists of numbers and is measured on a continuous or discrete scale.

- Continuous numerical data can take any value within a range (e.g., height, temperature).
- Discrete numerical data can take only specific, distinct values (e.g., number of siblings, number of cars).

Categorical (Qualitative) Data:

Categorical data represents categories or labels and is not inherently numerical.

- Nominal categorical data has categories with no inherent order or ranking (e.g., gender, eye color).
- Ordinal categorical data has categories with a specific order or ranking (e.g., education level, socioeconomic status).

Population and Sample

Population:

The population is the entire group or set of individuals, items, or elements that you are interested in studying and drawing conclusions about. It represents the complete set of possible observations that share a common characteristic or attribute. For example, if you are studying the heights of all adult males in a country, the population would consist of the heights of all adult males in that country.

Sample:

A sample is a subset or a smaller representative group selected from the population. It is used to make inferences or draw conclusions about the population without having to collect data from every individual in the population.

The process of selecting a sample from the population is known as sampling. For example, instead of measuring the heights of all adult males in a country (which may be impractical or

too costly), you might select a random sample of adult males and measure their heights to estimate the average height of the entire population.

Key points to note about population and sample:

- The population represents the entire group under study, while the sample represents a subset of that group.
- In many cases, it is not feasible or practical to collect data from the entire population, so researchers use samples to make inferences about the population.
- The goal of sampling is to obtain a representative sample that accurately reflects the characteristics of the population. Statistical techniques are used to analyze sample data and make generalizations or predictions about th##e population.

Characteristics of Population

- Mean (Average): The mean of a population is the average value of a quantitative variable across all individuals in the population. It represents the central tendency of the population distribution.
- Median: The median of a population is the middle value of a quantitative variable when all observations are arranged in ascending order. It is another measure of central tendency that is less affected by extreme values (outliers) compared to the mean.
- Mode: The mode of a population is the most frequently occurring value or category of a variable. It represents the value with the highest frequency in the population distribution.
- Variance: The variance of a population measures the spread or dispersion of values around the mean. It quantifies the average squared deviation of individual observations from the mean.
- Standard Deviation: The standard deviation of a population is the square root of the variance. It provides a measure of the average distance between individual observations and the mean.
- Range: The range of a population is the difference between the maximum and minimum values of a variable. It provides a simple measure of the spread of values in the population.
- Distribution: The distribution of a population describes how the values of a variable are spread or distributed across the population. Common types of distributions include normal (bell-shaped), skewed (asymmetric), and ##uniform (evenly distributed).

Characteristics of a Sample

- Sample Size: The sample size is the number of observations or individuals included in the sample. It represents the amount of data available for analysis and inference.

- Sampling Method: The sampling method describes how the sample was selected from the population. Common sampling methods include simple random sampling, stratified sampling, cluster sampling, and systematic sampling.
- Descriptive Statistics: Descriptive statistics summarize the main features of the sample data. Common descriptive statistics include measures of central tendency (mean, median, mode), measures of dispersion (range, variance, standard deviation), and measures of shape (skewness, kurtosis).
- Sample Proportion: The sample proportion represents the fraction or percentage of observations with a specific attribute or characteristic in the sample. It provides insights into the relative frequency of different categories in the sample.
- Confidence Interval: The confidence interval is a range of values that is likely to contain the true population parameter with a certain level of confidence. It is used to estimate the precision or uncertainty of sample statistics, such as the sample mean or proportion.
- Sampling Bias: Sampling bias refers to the systematic distortion or deviation of the sample from the population due to the sampling method used. It can affect the representativeness and generalizability of the sample data to the population. the sample data to the population. distributed).

Types of Variables

- `Features` : Features, also known as independent variables or input variables, are the attributes or characteristics of the data that are used as input to the machine learning model to make predictions. Features represent the variables that the model learns from to make predictions or classifications. Each feature can be either numerical or categorical and may have different scales or levels of measurement.
- `Labels` : Labels, also known as target variables or dependent variables, are the outputs or predictions that the machine learning model aims to predict based on the input features. Labels represent the target variable that the model is trying to learn or predict. In supervised learning tasks, the labels are typically known for a subset of the data, and the goal is to train the model to accurately predict the labels for unseen data.

Descriptive Statistics

Descriptive statistics deals with summarizing and describing the features of a dataset or sample. Descriptive statistics provides a summary of the main features of the data, including measures of central tendency, dispersion, shape, and relationships between variables.

Measures of Central Tendency:

- Mean: The average value of the data points.
- Median: The middle value of the data when arranged in ascending order.
- Mode: The most frequently occurring value in the dataset.

Measures of Dispersion:

- Range: The difference between the maximum and minimum values in the dataset.
- Variance: The average of the squared differences from the mean.
- Standard Deviation: The square root of the variance, representing the average deviation from the mean.

Measures of Shape:

- Skewness: A measure of the asymmetry of the distribution.
    - Positive skewness indicates a longer right tail and a concentration of data on the left side.
    - Negative skewness indicates a longer left tail and a concentration of data on the right side.
    - Skewness close to zero indicates approximate symmetry around the mean.

- Kurtosis: A measure of the "peakedness" or "flatness" of the distribution.
    - Positive kurtosis indicates heavy tails and a sharp peak (leptokurtic).
    - Negative kurtosis indicates light tails and a flat peak (platykurtic).
    - A kurtosis of 0 indicates a distribution with similar tails to the normal distribution (mesokurtic).

Frequency Distribution:

- Frequency table: A table that shows the frequency or count of each value in the dataset.
- Histogram: A graphical representation of the frequency distribution, showing the distribution of values in bins or intervals.

Measures of Association:

- Correlation: A measure of the strength and direction of the linear relationship between two variables.
- Covariance: A measure of the joint variability between two variables.

Ex. Find Mean, Median, Mode, Range, Standard Deviation, Skewness, Kurtosis, Frequency Distribution - Histogram

Example 1

In [ ]:
```
# dataset consists of weights children in the age group of 0 to 10 years
weights = np.array([20.8,15.3,23.2,15.5,17.5,27.3,23.3,20.5,16.4,17.4,22.6,20.8,16.
```

Example 2 -

In [ ]:
```
# dataset consists of Salaries of employees in an organisation
salaries = np.array([29756,20014,20347,57214,41327,40209,93390,122004,17725,47210,4
```

Example 3 -

In [ ]:
```
# dataset consists of life-expectancy data
life_expectancy = np.array([52, 61, 58, 75, 74, 77, 58, 74, 54, 65, 67, 75, 74, 70,
```

Handling Null/Missing Values

- Deletion:
  - Listwise Deletion: Remove entire rows containing null values.
  - Pairwise Deletion: Analyze data based on available pairs of variables, ignoring rows with null values in other variables.
- Imputation:
  - Mean/Median/Mode Imputation: Replace null values with the mean, median, or mode of the respective column.
  - Forward Fill/Backward Fill: Fill null values with the preceding or succeeding non-null value in the same column.
  - Linear Interpolation: Interpolate null values using linear interpolation based on neighboring data points.
  - K-Nearest Neighbors (KNN) Imputation: Use the values of nearest neighbors to impute null values.
  - Random Imputation: Replace null values with random values from the same column's distribution.
- Bucketing: Group data into buckets or categories and treat null values as a separate category.

In [ ]:

In [ ]:

Handling Outliers -

`Z-Score Method:`

- The z-score method involves calculating the z-score for each data point, which represents the number of standard deviations away from the mean. Data points with z-scores beyond a certain threshold (e.g., |z-score| > 3) are considered outliers and can be removed or treated separately. The z-score method is sensitive to the mean and standard deviation of the data, and it assumes that the data is normally distributed. This method is useful when the data is approximately normally distributed and when the goal is to identify outliers based on their deviation from the mean.

`IQR Method:`

- The IQR method involves calculating the interquartile range (IQR), which is the difference between the third quartile (Q3) and the first quartile (Q1) of the data. Outliers are defined as data points that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR. The IQR method is robust to outliers and does not assume any specific distribution of the data. This method is useful when the data is skewed or not normally distributed, as it focuses on the middle 50% of the data and is less influenced by extreme values. In general, if the data is approximately normally distributed and the goal is to identify outliers based on their deviation from the mean, the z-score method may be more appropriate. On the other hand, if the data is skewed or not normally distributed, or if the goal is to identify outliers based on their relative position within the dataset, the IQR method may be a better choice.

In [ ]:
```
salaries = np.array([29756,20014,20347,57214,41327,40209,93390,122004,17725,47210,4
```

In [ ]:
```

```

Univariate Analysis

Univariate analysis is a statistical method used to describe and analyze data consisting of only one variable. It focuses on understanding the characteristics and distribution of a single variable without considering the relationship with other variables.

- Descriptive Statistics
- Frequency Distribution
- Measures of Central Tendency
- Measures of Dispersion
- Visualization:
  - Box plots: Displaying the distribution of data using quartiles.
  - Histograms: Showing the frequency distribution of continuous variables.
  - Bar charts: Displaying the frequency distribution of categorical variables.
- Probability Distribution:
  - Normal distribution: Assessing if the data follows a normal distribution using graphical methods or statistical tests.

Read data from autompg.csv

In [ ]:

Bivariate Analysis

Bivariate analysis is a statistical method used to analyze the relationship between two variables simultaneously.

Numerical-Numerical Analysis:

- Scatter Plots: Scatter plots with a regression line can show the relationship between two continuous variables. Each data point represents a combination of values from both variables.
- Correlation Analysis: Quantifies the strength and direction of the linear relationship between two continuous variables. Pearson correlation coefficient (r) measures the degree of linear association between variables.
    - It ranges from -1 to 1, where:
    - r = 1: Perfect positive correlation
    - r = -1: Perfect negative correlation
    - r = 0: No correlation

Categorical-Categorical Analysis:

- Contingency tables (also known as cross-tabulations) display the frequency distribution of categories for two categorical variables.
- Chi-square test assesses the independence or association between two categorical variables.

Categorical-Numerical Analysis:

- Box plots or bar charts with groupings display the distribution of a numerical variable across different categories of a categorical variable.
- ANOVA (Analysis of Variance) tests the equality of means across different categories of a categorical variable.

In [ ]:
