

Aim: Introduction to Data science and Data preparation using Pandas steps.

Theory:

Data preparation is a fundamental step in data science, involving the cleaning and transformation of raw data into a structured and analyzable format. Pandas, a powerful Python library provides efficient tools for handling missing values, encoding categorical data, and scaling numerical features. Proper preprocessing enhances dataset quality, ensuring consistency and reliability for further analysis and machine learning models.

Problem Statement:

The Placement Data dataset contains various attributes related to students' academic performance, placement status, and salary packages. The objective of this experiment is to:

- Identify key trends in student placements based on academic performance.
- Analyze the distribution of salary packages.
- Handle missing data and remove inconsistencies.
- Standardize and normalize the data for further analysis.

By cleaning the placement dataset and applying data preprocessing steps, the goal is to improve data reliability, analyze student performance trends, and provide valuable insights for academic and recruitment decisions.

Dataset Overview:

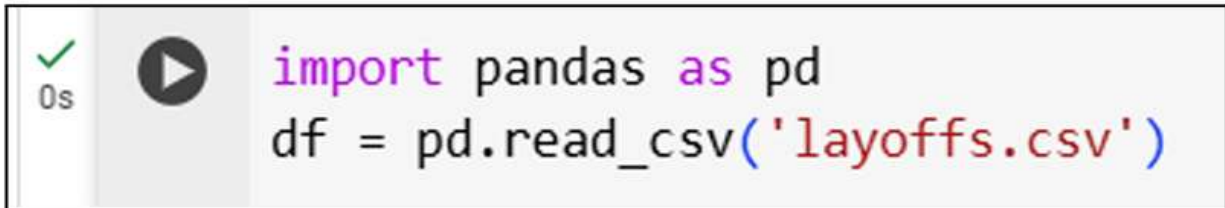
The dataset provides detailed information about student placements, academic performance, and salary distributions. It contains multiple columns, each capturing specific attributes related to students' educational backgrounds and employment outcomes. Below is a breakdown of the columns and their relevance: The dataset provides insights into student placements, containing columns such as:

1. **StudentID:** Unique identifier for each student.
2. **CGPA:** Cumulative Grade Point Average of the student.
3. **Internships:** Number of internships completed.
4. **Projects:** Number of academic or industry projects undertaken.
5. **Workshops/Certifications:** Number of workshops attended or certifications earned.
6. **Aptitude Test Score:** Score obtained in the aptitude test.
7. **Soft Skills Rating:** Rating of soft skills on a predefined scale.
8. **Extracurricular Activities:** Participation in extracurricular activities.
9. **Placement Training:** Whether the student underwent placement training (Yes/No).
10. **SSC Marks:** Secondary school exam scores.
11. **HSC Marks:** Higher secondary exam scores.
12. **Placement Status:** Whether the student was placed or not.

Steps:

Loading the Dataset:

The first step involves loading the dataset into a DataFrame using Pandas. This is typically done using the `read_csv()` function if the data is in CSV format:



```
import pandas as pd
df = pd.read_csv('layoffs.csv')
```

Description of Dataset.

A. Information about the Dataset.

Use functions like `.head()`, `.tail()`, `.shape`, and `.describe()` to get a general idea of what the dataset looks like.

```
df.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1839 entries, 0 to 1838
Data columns (total 18 columns):

#	Column	Non-Null Count	Dtype
0	#	1839 non-null	int64
1	Company	1839 non-null	object
2	Location_HQ	1839 non-null	object
3	Region	473 non-null	object
4	State	566 non-null	object
5	Country	1839 non-null	object
6	Continent	1839 non-null	object
7	Laid_Off	1677 non-null	float64
8	Date_layoffs	1839 non-null	object
9	Percentage	1667 non-null	object
10	Company_Size_before_Layoffs	1585 non-null	object
11	Company_Size_after_layoffs	1619 non-null	object
12	Industry	1839 non-null	object
13	Stage	1839 non-null	object
14	Money_Raised_in__mil	1692 non-null	float64
15	Year	1839 non-null	int64
16	latitude	1839 non-null	float64
17	longitude	1839 non-null	float64

dtypes: float64(4), int64(2), object(12)
memory usage: 258.7+ KB

B. Drop the columns that are not useful.

```
import pandas as pd
df = pd.read_csv('layoffs.csv')
cols = ['latitude', 'longitude']
df = df.drop(cols, axis=1)
df.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1839 entries, 0 to 1838
Data columns (total 16 columns):

#	Column	Non-Null Count	Dtype
0	#	1839 non-null	int64
1	Company	1839 non-null	object
2	Location_HQ	1839 non-null	object
3	Region	473 non-null	object
4	State	566 non-null	object
5	Country	1839 non-null	object
6	Continent	1839 non-null	object
7	Laid_Off	1677 non-null	float64
8	Date_layoffs	1839 non-null	object
9	Percentage	1667 non-null	object
10	Company_Size_before_Layoffs	1585 non-null	object
11	Company_Size_after_layoffs	1619 non-null	object
12	Industry	1839 non-null	object
13	Stage	1839 non-null	object
14	Money_Raised_in__mil	1692 non-null	float64
15	Year	1839 non-null	int64

dtypes: float64(2), int64(2), object(12)
memory usage: 230.0+ KB

Columns Present in the Database are:

The `.info()` method provides insight into the structure of the dataset — data types, non-null values, and memory usage:

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1839 entries, 0 to 1838  
Data columns (total 16 columns):
```

#	Column	Non-Null Count	Dtype
0	#	1839 non-null	int64
1	Company	1839 non-null	object
2	Location_HQ	1839 non-null	object
3	Region	473 non-null	object
4	State	566 non-null	object
5	Country	1839 non-null	object
6	Continent	1839 non-null	object
7	Laid_Off	1677 non-null	float64
8	Date_layoffs	1839 non-null	object
9	Percentage	1667 non-null	object
10	Company_Size_before_Layoffs	1585 non-null	object
11	Company_Size_after_layoffs	1619 non-null	object
12	Industry	1839 non-null	object
13	Stage	1839 non-null	object
14	Money_Raised_in__mil	1692 non-null	float64
15	Year	1839 non-null	int64

dtypes: float64(2), int64(2), object(12)
memory usage: 230.0+ KB

C. Now we will drop the rows with missing.

Sometimes, some columns do not contribute any meaningful information to the analysis. For example, **StudentID** might be dropped if it is only a unique identifier.

```
import pandas as pd
df = pd.read_csv('layoffs.csv')
cols = ['latitude', 'longitude']
df = df.drop(cols, axis=1)
df = df.dropna()
df.info()
```

<class 'pandas.core.frame.DataFrame'>
Index: 403 entries, 2 to 1836
Data columns (total 16 columns):

#	Column	Non-Null Count	Dtype
0	#	403 non-null	int64
1	Company	403 non-null	object
2	Location_HQ	403 non-null	object
3	Region	403 non-null	object
4	State	403 non-null	object
5	Country	403 non-null	object
6	Continent	403 non-null	object
7	Laid_Off	403 non-null	float64
8	Date_layoffs	403 non-null	object
9	Percentage	403 non-null	object
10	Company_Size_before_Layoffs	403 non-null	object
11	Company_Size_after_layoffs	403 non-null	object
12	Industry	403 non-null	object
13	Stage	403 non-null	object
14	Money_Raised_in_mil	403 non-null	float64
15	Year	403 non-null	int64

dtypes: float64(2), int64(2), object(12)
memory usage: 53.5+ KB

D. Now we are creating dummy variables:

Categorical columns like **Placement Training** or **Placement Status** need to be converted to numeric format using dummy variables:

```
import pandas as pd
df = pd.read_csv('layoffs.csv')
cols = ['latitude', 'longitude']
df = df.drop(cols, axis=1)
df = df.dropna()
dummies = []
cols = ['State', 'Stage',]
for col in cols:
    dummies.append(pd.get_dummies(df[col]))
layoffs_dummies = pd.concat(dummies, axis=1)
df = pd.concat((df, layoffs_dummies), axis=1)
df.info()
```

>>> <class 'pandas.core.frame.DataFrame'>
Index: 403 entries, 2 to 1836
Data columns (total 32 columns):
Column Non-Null Count Dtype
--- --- -
0 # 403 non-null int64
1 Company 403 non-null object
2 Location_HQ 403 non-null object
3 Region 403 non-null object
4 State 403 non-null object
5 Country 403 non-null object
6 Continent 403 non-null object
7 Laid_Off 403 non-null float64
8 Date_layoffs 403 non-null object
9 Percentage 403 non-null object

E. Taking care of missing data.

Instead of dropping, sometimes missing data is filled with appropriate values:

```
import pandas as pd
df = pd.read_csv('layoffs.csv')
df['Money_Raised_in__mil'] = df['Money_Raised_in__mil'].interpolate()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1839 entries, 0 to 1838
Data columns (total 18 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   #                                     1839 non-null  int64
1   Company                             1839 non-null  object
2   Location_HQ                         1839 non-null  object
3   Region                             473 non-null   object
4   State                              566 non-null   object
5   Country                            1839 non-null  object
6   Continent                          1839 non-null  object
7   Laid_Off                           1677 non-null  float64
8   Date_layoffs                       1839 non-null  object
9   Percentage                          1667 non-null  object
10  Company_Size_before_Layoffs         1585 non-null  object
11  Company_Size_after_layoffs          1619 non-null  object
12  Industry                            1839 non-null  object
13  Stage                               1839 non-null  object
14  Money_Raised_in__mil                1839 non-null  float64
15  Year                                1839 non-null  int64
16  latitude                            1839 non-null  float64
17  longitude                           1839 non-null  float64
dtypes: float64(4), int64(2), object(12)
memory usage: 258.7+ KB
```

F. Finding out outliers:

Outliers can skew the analysis. They can be detected using statistical methods like the IQR method or visualizations like box plots.

```
import pandas as pd

# Load dataset
df = pd.read_csv('layoffs.csv')

# Interpolate missing values
df['Money_Raised_in__mil'] = df['Money_Raised_in__mil'].interpolate()

# Function to detect outliers using IQR
def find_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
    return outliers

# Find outliers in 'Money_Raised_in__mil' column
outliers = find_outliers_iqr(df, 'Money_Raised_in__mil')
print("Outliers based on IQR method:\n", outliers)
```

G. Normalization the columns: Normalization scales the data to a range (usually 0 to 1). It's useful when features have different units or scales:

```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Normalize the 'Money_Raised_in_mil' column
df['Money_Raised_Normalized'] = scaler.fit_transform(df[['Money_Raised_in_mil']])

print(df[['Money_Raised_in_mil', 'Money_Raised_Normalized']].head())
```

	Money_Raised_in_mil	Money_Raised_Normalized
0	90.0	0.000730
1	45.0	0.000361
2	1.0	0.000000
3	6.0	0.000041
4	79.0	0.000640

H. Standardization the columns :

Standardization rescales data to have a mean of 0 and a standard deviation of 1. It's particularly useful for algorithms like SVM or KNN:

```
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Standardize the 'Money_Raised_in_mil' column
df['Money_Raised_Standardized'] = scaler.fit_transform(df[['Money_Raised_in_mil']])

print(df[['Money_Raised_in_mil', 'Money_Raised_Standardized']].head())
```

	Money_Raised_in_mil	Money_Raised_Standardized
0	90.0	-0.133289
1	45.0	-0.143487
2	1.0	-0.153458
3	6.0	-0.152325
4	79.0	-0.135782

Conclusion:

In this experiment, we highlighted the importance of data preparation in the data science process using the Pandas library. By working with a student placement dataset, we performed essential preprocessing tasks such as handling missing values, cleaning inconsistencies, encoding categorical variables, and standardizing numerical data. These steps helped transform the raw dataset into a structured and consistent format suitable for analysis. As a result, we were able to identify meaningful trends in student placements, understand the distribution of salary packages, and evaluate the impact of academic performance, internships, soft skills, and training on placement outcomes.

This preparation not only improved the reliability of the dataset but also provided a strong foundation for future analyses and decision-making in academic and recruitment contexts.

