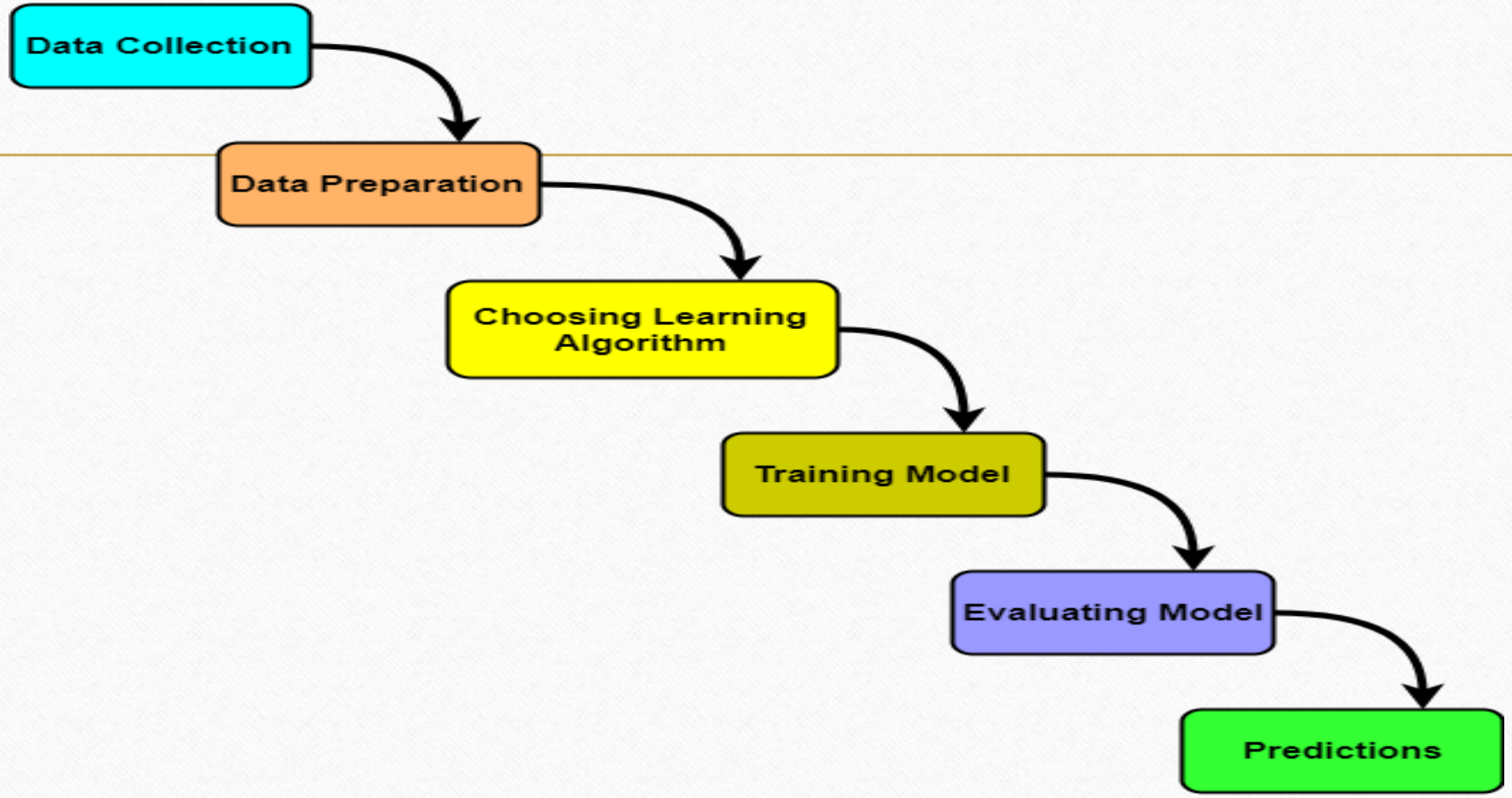
Data Preprocessing

Understanding the Data, Data Preprocessing- Data Normalization, Data Binning, Importing and Exporting Data in Python

Mr. Khushal Khairnar

Data Analytics Steps



Presented By: Khushal Khairnar

Machine Learning Workflow

Step: 2 Data Preprocessing

Presented By: Khushal Khairnar

What is data preprocessing?

- In the real world, several terabytes of data is generated by multiple sources.
- But all of it is not directly usable. Audio, video, images, text, charts, logs all of them contain data.
- But this data **needs to be cleaned** in a usable format for the machine learning algorithms to produce meaningful results.
- The process of cleaning raw data for it to be used for machine learning activities is **known as data pre-processing.**

Why data - preprocessing?

- Real-world data is often noisy, incomplete with missing entries, and more often than not unsuitable for direct use for building models or solving complex data-related problems.
- There might be erroneous data, or the data might be unordered, unstructured, and unformatted.
- The above reasons render the collected data unusable for machine learning purposes.

Data pre-processing steps

✓ It involves below steps:

- **Getting the dataset (collection of data)**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data**
- **Data Binning**
- **Data Scaling**

Data pre-processing steps

- **Getting the dataset (collection of data)**
  - Data collection is the stage when we collect data from various sources.
  - Data might be laying across several storages or several servers and we need to get all that data collected in one single location for the ease of access.
  - Data is present in many formats. So we need to devise a common format for data collection.
  - Eg: CSV, xlsx, HTML,JSON,etc....

Data collection :

goto: https://www.kaggle.com/datasets

Data pre-processing steps

- **Importing libraries**
  - Data import is the process of importing data into the software such as R or python for data cleaning purposes.
  - Tools like pandas, dask, NumPy, and matplotlib are handy when operating on such huge volumes of data.

Importing Libraries :

Install pandas using following command on terminal:

```
pip install pandas
```

Import the pandas library using:

```
import pandas
```

Presented By: Khushal Khairnar

Data pre-processing steps

- **Importing datasets**
  - Now we are ready to use pandas, and you can write your code in the next cells.

  - Import pandas as pd

  - `df=pd.read_csv("filename.csv")`
  - `print(df)`

Import the dataset

- First, we import pandas. Then we use the read_csv() function of pandas to read the file in computer memory.
- Inside the read_csv function, we have passed the dataset name as an argument.
- This is because the dataset is in the same directory as that of the python file.

- Import pandas as pd
    - df=pd.read_csv("filename.csv")

```
print(df)
```

Presented By: Khushal Khairnar

Data pre-processing steps

- **Getting the dataset**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data**

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

Presented By: Khushal Khairnar

• Data pre-processing steps: **Finding Missing Data**

•Identify Missing values

For example, the column Age and Salary is not available for all the rows. In some cases it presents the NaN value, which means that the value is missing.

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

• Data pre-processing steps: **Finding Missing Data**

•Finding Missing Data:

- In order to check whether our dataset contains missing values, we can use the function `isna()`, which returns if an cell of the dataset if NaN or not. Then we can count how many missing values there are for each column.

```
Import pandas as pd
    df=pd.read_csv("filename.csv")
print(df)
```

df.isna

df.isna()

df.isna().sum()

Output:

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

• Data pre-processing steps: **Finding Missing Data**

•Finding Missing Data:

- Now we can count the percentage of missing values for each column, simply by dividing the previous result by the length of the dataset (`len(df)`) and multiplying per 100.

```
Import pandas as pd
    df=pd.read_csv("filename.csv")
print(df)
```

```
print(df.isna().sum()/len(df)*100)
```

Output:

```
Country       0.0
Age          10.0
Salary       10.0
Purchased     0.0
dtype: float64
```

What we have to do with missing values????

Presented By: Khushal Khairnar

• Data pre-processing steps: **Finding Missing Data**

•When dealing with missing values, different alternatives can be applied:

- check the source, for example by contacting the data source to correct the missing values
- drop missing values
- replace the missing value with a value
- leave the missing value as it is.

• Data pre-processing steps: **Finding Missing Data**

- **Drop missing values:**
- ✓ remove **rows** having missing values
- ✓ remove **the whole column** containing missing values
- **We can use the dropna() by specifying the axis to be considered.**

```
➢ Import pandas as pd
➢  df=pd.read_csv("filename.csv")
   print(df)
```

```
print(df.dropna())
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
5	France	35.0	58000.0	Yes
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

• Data pre-processing steps: **Finding Missing Data**

- **Replace Missing Value:**

✓ A good strategy when dealing with missing values involves their replacement with another value.

✓ Usually, the following strategies are adopted:

✓ for numerical values replace the missing value with the average value of the column

✓ for categorial values replace the missing value with the most frequent value of the column

✓ use other functions

• Data pre-processing steps: **Finding Missing Data**

- **Replace Missing Value:**

✓ In order to replace missing values, three functions can be used:

- `fillna()`,
- `replace()` and
- `interpolate()`.

• Data pre-processing steps: **Finding Missing Data**

- **Replace Missing Value:**

✓ In order to replace missing values, three functions can be used:

➢ `fillna()`: The `fillna()` function replaces all the NaN values with the value passed as argument.

```
➢ Import pandas as pd
➢  df=pd.read_csv("filename.csv")
   print(df)
```

```
print(df.fillna(12))
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	12.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	12.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

• Data pre-processing steps: **Finding Missing Data**

- **Replace Missing Value:**

✓ In order to replace missing values, three functions can be used:

➢ `fillna()`: The `fillna()` function replaces all the NaN values with the value passed as argument.

➢ Import pandas as pd
➢ `df=pd.read_csv("filename.csv")`
`print(df)`

```
print(df.fillna(df.mean()))
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	63777.777778	Yes
5	France	35.0	58000.0	Yes
6	Spain	38.777778	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

• Data pre-processing steps: **Finding Missing Data**

- **Replace Missing Value:**

➢ `interpolate()`: Another solution to replace missing values involves the usage of other functions, such as **linear interpolation.**

➢ In this case, for example, we could replace a missing value over a column, **with the interpolation between the previous and the next ones.**

• Data pre-processing steps: **Finding Missing Data**

- Import pandas as pd
- df=pd.read_csv("filename.csv")
  print(df)

```
print(df.interpolate())
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	59500.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	41.5	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Data pre-processing steps

- **Data Binning:**
  - ✓ Data binning (or bucketing) groups data in bins (or buckets),
  - ✓ in the sense that it replaces values contained into a small interval with a single representative value for that interval.
  - ✓ Sometimes binning improves accuracy in predictive models.

Data pre-processing steps

- **Data Binning:**
  - ✓ We make bin(group for 'age' or 'Salary' columns)

- **we can 'bin' age into [20,30,40,50]**

Country	Age	Salary	Purchased	age_category
0 France	44.0	72000.0	No	senior
1 Spain	27.0	48000.0	Yes	youth
2 Germany	30.0	54000.0	No	youth
3 Spain	38.0	61000.0	No	adult
4 Germany	40.0	NaN	Yes	adult
5 France	35.0	58000.0	Yes	adult
6 Spain	NaN	52000.0	No	NaN
7 France	48.0	79000.0	Yes	senior
8 Germany	50.0	83000.0	No	senior
9 France	37.0	67000.0	Yes	adult

Data pre-processing steps

- **we can 'bin' age into [20,30,40,50]**

```
bins=[20,30,40,50]

groups_name=['youth','adult','senior']

df['age_category']=pd.cut(df['Age'],bins,labels=groups_name)

print(df)
```

	Country	Age	Salary	Purchased	age_category
0	France	44.0	72000.0	No	senior
1	Spain	27.0	48000.0	Yes	adult
2	Germany	30.0	54000.0	No	adult
3	Spain	38.0	61000.0	No	adult
4	Germany	40.0	NaN	Yes	adult
5	France	35.0	58000.0	Yes	adult
6	Spain	NaN	52000.0	No	NaN
7	France	48.0	79000.0	Yes	senior
8	Germany	50.0	83000.0	No	senior
9	France	37.0	67000.0	Yes	adult

Group by

The **groupby** function is used to group data in a DataFrame based on one or more columns and then perform operations on each group separately. This is a common operation in data analysis and is similar to SQL's GROUP BY clause.

Syntax:

```
grouped = dataframe.groupby(by)
```

Where **dataframe** is your Pandas DataFrame, and **by** is a specification of how to group the data. We can specify the grouping by one or more columns, a function, or a combination of those.

Group by

```
grouped = df.groupby('Country')
Grouped      #it will print object at address
```

```
average_age = grouped['Salary'].sum()
average_age
```

```
Country
France     276000.0
Germany    137000.0
Spain      161000.0
Name: Salary, dtype: float64
```

Group by

```
for country, group_data in grouped:
    print(f"Country: {country}")
    print(group_data)
    print("\n")
```

```
Country: France
  Country   Age   Salary Purchased age_category
0  France  44.0  72000.0        No       senior
5  France  35.0  58000.0       Yes        adult
7  France  48.0  79000.0       Yes       senior
9  France  37.0  67000.0       Yes        adult


Country: Germany
   Country   Age   Salary Purchased age_category
2  Germany  30.0  54000.0        No        youth
4  Germany  40.0      NaN       Yes        adult
8  Germany  50.0  83000.0        No       senior


Country: Spain
  Country   Age   Salary Purchased age_category
1   Spain  27.0  48000.0       Yes        youth
3   Spain  38.0  61000.0        No        adult
6   Spain   NaN  52000.0        No          NaN
```

We can filter DataFrame:

- To find the individuals who have a salary of 52000

```
result = df[df['Salary'] == 52000.0]['Country']

print(result)
```

```
6    Spain
Name: Country, dtype:object
```

We can filter DataFrame:

- Filter the DataFrame for individuals with Purchased == 'No' and select the 'Age' column

```
ages_purchased_no = df[df['Purchased'] == 'No']['Age']

print(ages_purchased_no)
```

```
0    44.0
2    30.0
3    38.0
6     NaN
8    50.0
Name: Age, dtype: float64
```