
NITTE
EDUCATION TRUST
Data Science using Python
Dr.Vani Vasudevan

Outline

- Introduction to Data Science
- Data Processing
- Data Visualization
- Introduction to Machine Learning
- Foundation of Neural Networks and Deep Learning

EDUCATION TRUST

Unit II

Data Processing in Python for Data Science

NITTE
EDUCATION TRUST

What is Data Processing?

- Data processing involves collecting and cleaning
- Transforming raw data into useful formats
- Preparing data for machine learning models
- Identifying patterns and extracting key trends
- Iterative workflow ensures accurate analysis
- Enhances insights and overall decision making

Importance in Data Science

- Raw data often messy and inconsistent
- Missing values noise mislead learning algorithms
- Cleaned transformed data improves model performance
- Ensures analysis uses high quality data
- Reduces biases errors improves overall robustness
- Enables reliable predictions and better insights

Key Steps in Data Processing…

1. Collection:

- Collect data from multiple diverse sources
- Sources include databases, APIs, sensors, files
- Ensure data gathered is relevant accurate
- Completeness of data ensures better analysis
- Accuracy improves overall task or model
- First step essential in data processing

NITTE
EDUCATION TRUST

Key Steps in Data Processing...

2. Cleaning:

- Raw data scrubbed for inconsistencies missing
- Remove duplicates and irrelevant noisy entries
- Handle missing values remove or impute
- Correct errors outliers for accurate analysis
- Poor quality data leads inaccurate models
- Cleaning critical ensures reliable business insights

Key Steps in Data Processing

3. Transformation:

- Cleaned data transformed into usable format
- Normalize scale data for consistent ranges
- Encode categorical variables into numerical values
- Create new features through feature engineering
- Transformation ensures readiness for machine learning
- Required specific formats enable accurate analysis

Key Steps in Data Processing

4. Analysis:

- Analyze processed data for deeper insights
- Apply statistical methods detect patterns trends
- Use algorithms to generate reliable predictions
- Visualize data effectively convey meaningful findings
- Goal is deriving conclusions for decisions
- Guides business strategies and future actions

Understanding Data Processing

- Data processing transforms raw into structured
- Business analytics improves decisions with insights
- Scientific research analyzes data confirm hypotheses
- Machine learning prepares data for predictions
- Healthcare identifies patterns using patient records
- Finance processes data detect fraud risks

Data Collection vs. Data Processing

Data collection focuses on gathering data whereas,

Data processing deals with transforming that data into a usable state for analysis.

Role of Python in Data Processing

- Python versatile simple powerful for data
- Widely used language in data science
- Libraries make handling data more efficient
- NumPy supports arrays matrices mathematical functions
- Pandas enables preprocessing filtering aggregating transforming
- Ideal tool for robust data workflows

Role of Python in Data Processing…

- Python **handles large datasets** with scalability
- Dask, PySpark enable distributed data processing
- Efficient across multiple machines big projects
- **Automates** repetitive tasks through flexible scripting
- Builds pipelines for collection cleaning transformation
- Improves workflows efficiency across data processes

Role of Python in Data Processing…

- Python integrates seamlessly with diverse sources
- Supports SQL NoSQL databases APIs connections
- SQLAlchemy PyMongo enable smooth data integration
- Facilitates collection initial processing of datasets
- Handles transformations text numeric feature engineering
- Versatile across multiple different data types

Overview of Data Clening…

Why Clean Data?

- Data cleaning essential in analysis pipeline
- Raw data often inconsistent erroneous irrelevant
- Errors lead to misleading incorrect results
- Clean data ensures accuracy in analysis
- Reliable insights come from processed information
- Meaningful conclusions guide better data decisions

Overview of Data Cleaning…

Some reasons why data cleaning is crucial include:

- Accuracy removing errors incorrect raw values
- Consistency harmonizes formats units date representations
- Missing data handled prevents bias distortions
- Outliers identified addressed avoid skewed analysis
- Improved performance for machine learning models
- Clean data enhances precision generalization ability

Identifying Dirty Data

- Dirty data incomplete inconsistent or incorrect
- Outliers skew results errors natural variability
- Missing values arise from omissions malfunctions
- Duplicates distort analyses overrepresent certain observations
- Detection uses visualization and summary statistics
- Pandas functions identify isnull duplicated checks

Identifying outliers using statistics…

1. Z-Score (Standard Score) Method

- Identify outliers using statistical score method
- Z score measures deviations from dataset mean
- Greater than three or less minus three
- Indicates data point considered significant outlier
- Formula:

$$Z = (X - \mu) / \sigma$$

Where:

X is the data point, μ is the mean,

Σ is the standard deviation.

Identifying outliers using statistics…

```
import numpy as np
import pandas as pd
# Sample dataset
data = {'Value': [10, 12, 13, 10, 12, 11, 14, 13, 15, 150]}  # 150 is an outlier
df = pd.DataFrame(data)
# Calculate Z-scores
df['Z-Score'] = (df['Value'] - df['Value'].mean()) / df['Value'].std()
# Identifying outliers with Z-score > 3 or < -3
outliers = df[df['Z-Score'].abs() > 3]
print(outliers)
```

Identifying outliers using statistics…

2. IQR (Interquartile Range) Method

This method uses percentiles to identify outliers. Data points that fall below the lower bound or above the upper bound are considered outliers.

Formula:

$$Lower\ Bound = Q1 - 1.5 \times IQR$$
$$Upper\ Bound = Q3 + 1.5 \times IQR$$

Where:

- $Q1$ is the first quartile (25th percentile),
- $Q3$ is the third quartile (75th percentile),
- IQR is the interquartile range ($Q3 - Q1$).

Identifying outliers using statistics

```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df['Value'].quantile(0.25)
Q3 = df['Value'].quantile(0.75)
IQR = Q3 - Q1
# Calculate the lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
# Identify outliers
outliers = df[(df['Value'] < lower_bound) | (df['Value'] >
upper_bound)]
print(outliers)
```

Identifying missing data

```
data = {'CustomerID': [1, 2, 3, 4], 'Email': ['abc@example.com',
None,'def@example.com', None]}
df = pd.DataFrame(data)

# Identifying missing data
df.isnull()
```

Identifying duplicate data

```
data = {'TransactionID': [1, 2, 2, 4], 'Amount': [100, 200, 200, 300]}
df = pd.DataFrame(data)

# Checking for duplicates
df.duplicated()
```

Data Cleaning Techniques…

Some common techniques are:

- **Handling Missing Data**:
  - **Imputation**: Replace missing values with statistical values such as mean, median, or mode.
  - **Removal**: Remove rows or columns with too many missing values.
  - **Forward/Backward Filling**: For time series data, use adjacent values to fill in missing data (forward fill or backward fill).

Data Cleaning Techniques…

Imputation: Suppose you have a dataset with some missing salary values. You can fill in these missing values with the median salary of the group.

```
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Salary': [50000, None, 55000]}
df = pd.DataFrame(data)

# Fill missing salary with the median
df['Salary'].fillna(df['Salary'].median(), inplace=True)
```

Data Cleaning Techniques…

Forward/Backward Filling: Useful for time-series data when missing values can be inferred from nearby data points.

```
data = {'Date': pd.date_range(start='2024-01-01', periods=5),
'Stock Price': [100, None, 102, None, 105]}
df = pd.DataFrame(data)

# Forward fill the missing values
df['Stock Price'].fillna(method='ffill', inplace=True)
```

Data Cleaning Techniques…

- **Dealing with Outliers**:
  - **Trimming**: Remove outliers that fall outside a specific percentile range.
  - **Capping**: Replace outliers with the closest acceptable values (e.g., limiting values to within a certain range).
  - **Transformation**: Apply transformations such as log or square root to reduce the impact of outliers.

Data Cleaning Techniques…

Dealing with Outliers:

#Trimming: Remove the top and bottom 1% of values to eliminate extreme #outliers.

```
# Remove top and bottom 1% of data
df = df[df['House Price'].between(df['House Price'].quantile(0.01), df['House Price'].quantile(0.99))]
```

Data Cleaning Techniques…

Transformation: Apply log transformation to reduce the effect of outliers.

```
df['Log_Price'] = np.log(df['House Price'])
```

Data Cleaning Techniques…

- **Addressing Duplicates**:
  - **Deduplication**: Use tools like Pandas' drop_duplicates() function to identify and remove duplicate rows.
  - **Aggregation**: In some cases, duplicates may not be removed but aggregated (e.g., averaging values for repeated entries).
- **Standardization**: Convert data into a consistent format (e.g., date formats, categorical variable labels).

Data Cleaning Techniques…

#Addressing Duplicates:

Remove duplicate rows

- df.drop_duplicates(inplace=True)

Data Cleaning Techniques

#Standardization: If you have inconsistent date formats in your dataset,

```
# standardize them using Pandas.
data = {'Date': ['07/10/2024', '2024-10-07', '07-Oct-2024']}
df = pd.DataFrame(data)

# Standardize date format
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
```

Slicing and Indexing in Pandas

- Indexing: Label-based, Integer-based, Mixed
- Slicing: Row/Column Selection
- Example Code:

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
# Selecting a single column
df['A']
# Slicing rows
df.iloc[0:2]
```

Manipulating and Cleaning Pandas DataFrames

- Adding, Deleting Columns
- Merging, Concatenating DataFrames
- Filtering and Sorting Data
- Example Code:

```
df['C'] = df['A'] + df['B']
df.drop('B', axis=1)
pd.concat([df1, df2], axis=0)
```

Working with Missing Data in Pandas

- Strategies for Handling Missing Data
- Dropping and Filling Missing Values
- Example Code:

```
df.dropna()
df.fillna(0)
df.isna().sum()
```

Pandas and CSV Files

- Reading and Writing CSV Files with Pandas
- Example Code:

```
df = pd.read_csv('data.csv')
df.to_csv('output.csv', index=False)
```

NITTE EDUCATION TRUST

Pandas and JSON Files

- Working with JSON Files in Pandas
- Example Code:

```
df = pd.read_json('data.json')
df.to_json('output.json')
```

Python Relational Database

- Working with SQL in Python
- SQLite and Pandas Integration
- Example Code:

```
import sqlite3
conn = sqlite3.connect('example.db')
df = pd.read_sql_query('SELECT * FROM table_name', conn)
```

Summary – Data Processing in Python

- Data processing essential for accurate insights
- Steps include collection cleaning transformation analysis
- Clean data ensures reliability and consistency
- Python powerful versatile libraries NumPy Pandas
- Supports scalability automation integration flexibility
- Crucial across domains business healthcare finance

Summary – Data Cleaning & Detection

- Dirty data incomplete inconsistent or incorrect
- Outliers missing values duplicates distort analysis
- Detection uses statistics visualization programmatic checks
- Cleaning techniques include imputation filling trimming
- Deduplication and standardization improve dataset quality
- Clean data ensures reliable robust model performance