* **Data Wrangling 1**

Got it! Here's a **focused, concise summary of key methods and transformations** used in **Practical No. 1**, highlighting only the most **important parts** of the code along with **what they do and why they matter**:

**🔍 Practical No. 1 — Key Concepts & Code Summary**

**📌 1. pd.read\_csv()**

df = pd.read\_csv("Automobile\_data.csv")

**Purpose**: Loads the CSV data into a Pandas DataFrame.

**📌 2. Replacing Missing Values**

df.replace("?", np.nan, inplace=True)

**Why**: The dataset uses "?" to represent missing values. This line converts them to np.nan (a standard missing value indicator in Pandas).

**📌 3. Checking Missing Values**

df.isnull().sum()

**Purpose**: Counts missing values in each column to identify which need to be filled or cleaned.

**📌 4. Converting Data Types**

df[col] = pd.to\_numeric(df[col], errors='coerce')

**Use**: Converts numeric columns (e.g., price, horsepower) from string to actual numeric types. Invalid entries (like "?") are set as NaN.

df[col] = df[col].astype('category')

**Use**: Converts categorical string columns into efficient categorical types for faster processing and encoding.

**📌 5. Filling Missing Data**

df[col] = df[col].fillna(df[col].mean()) # For numerical columns

df[col] = df[col].fillna(df[col].mode()[0]) # For categorical columns

**Purpose**: Replaces missing numeric values with the column **mean**, and categorical with the **mode** (most common value).

**📌 6. One-Hot Encoding**

df\_one\_hot = pd.get\_dummies(df, columns=['fuel-type'], prefix='fuel')

**Purpose**: Converts fuel-type (e.g., 'gas', 'diesel') into binary columns (fuel\_gas, fuel\_diesel) so models can understand it.

**📌 7. Label Encoding**

df['fuel-type'] = df['fuel-type'].cat.codes

**Use**: Converts categories into numeric codes (e.g., gas → 1, diesel → 0). Useful for algorithms that don’t accept strings.

fuel\_type\_mapping = dict(enumerate(df['fuel-type'].cat.categories))

**Purpose**: Keeps a mapping to interpret what each number means.

**📌 8. Normalization (Optional but Important)**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df[numerical\_columns] = scaler.fit\_transform(df[numerical\_columns])

**Why**: Scales all numeric values between 0 and 1. Helps machine learning models perform better by making feature scales uniform.

**✅ Summary of Key Skills Demonstrated**

| **Concept** | **Skill Used** |
| --- | --- |
| Data Loading | pd.read\_csv() |
| Missing Value Handling | replace(), isnull(), fillna() |
| Data Type Conversion | astype(), pd.to\_numeric() |
| Encoding | get\_dummies(), .cat.codes |
| Normalization | MinMaxScaler |

Would you like a printable PDF or markdown version of this summary too?

* **Data Wrangling 2**

Sure! Let's break down **Practical No. 2** into **key methods**, **important steps**, and **critical changes** in the code, with brief explanations for each.

**1. Importing Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import statistics as st

* **pandas**: Used for creating and manipulating data frames.
* **numpy**: Used for numerical operations (handling missing values, calculations).
* **matplotlib.pyplot** and **seaborn**: Used for data visualization (plots and graphs).
* **MinMaxScaler** and **StandardScaler**: From **sklearn.preprocessing**, used for normalizing and standardizing numeric data.
* **statistics**: Used for calculating statistical functions like **mode**.

**2. Creating Custom Dataset**

data = {

'gender': ['female', 'male', 'female', 'male', 'female', 'male', 'female', 'male', 'female', 'male'],

'race/ethnicity': ['group A', 'group B', 'group C', 'group A', 'group B', 'group D', 'group C', 'group E', 'group D', 'group E'],

'parental level of education': ['bachelor\'s degree', 'some college', 'associate\'s degree', 'high school', 'master\'s degree',

'associate\'s degree', 'high school', 'some college', 'bachelor\'s degree', 'master\'s degree'],

'lunch': ['standard', 'free/reduced', 'standard', 'free/reduced', 'standard', 'standard', 'free/reduced', 'standard', 'standard', 'free/reduced'],

'test preparation course': ['none', 'completed', 'none', 'completed', 'none', 'completed', 'none', 'completed', 'none', 'completed'],

'math\_score': [78, 62, 90, 55, 88, 45, 66, 72, 80, 50],

'reading\_score': [72, 60, 95, 58, 84, 52, 70, 78, 85, 56],

'writing\_score': [70, 58, 93, 60, 82, 50, 68, 74, 88, 54]

}

df = pd.DataFrame(data)

* Here, a **custom dataset** is created using a dictionary. It simulates student data, including scores and other categorical information such as **gender** and **test preparation course**.

**3. Converting Categorical Columns to category Data Type**

categorical = ['gender', 'race/ethnicity', 'parental level of education', 'lunch', 'test preparation course']

for col in categorical:

df[col] = df[col].astype("category")

* Converts columns like **gender**, **race/ethnicity**, etc., into **categorical data type** using **astype("category")**. This helps in efficient memory usage and processing.

**4. Standardizing Column Names**

df.columns = df.columns.str.replace(' ', '\_')

* **Renames columns** to standardize them, replacing spaces with underscores (e.g., parental level of education becomes parental\_level\_of\_education).
* This makes it easier to access columns programmatically.

**5. Visualizing Outliers with a Boxplot**

plt.figure(figsize=(10, 6))

sns.boxplot(data=df.select\_dtypes(["int64"]))

plt.title('Boxplot of Numeric Columns')

plt.show()

* **Boxplot** is created using **seaborn** to detect outliers in numeric columns (e.g., **math\_score**, **reading\_score**, **writing\_score**).
* **Boxplots** help visualize the spread of data and highlight outliers beyond the "whiskers."

**6. Detecting Outliers Using Interquartile Range (IQR)**

numeric\_cols = df.select\_dtypes(include=['int64']) # Select numeric columns

Q1 = numeric\_cols.quantile(0.25)

Q3 = numeric\_cols.quantile(0.75)

IQR = Q3 - Q1 # Calculate IQR

outliers = ((numeric\_cols < (Q1 - 1.5 \* IQR)) | (numeric\_cols > (Q3 + 1.5 \* IQR)))

# Remove outliers and create a new DataFrame

new\_df = df.copy()

new\_df[numeric\_cols.columns] = new\_df[numeric\_cols.columns].where(~outliers[numeric\_cols.columns], np.nan)

new\_df.dropna(inplace=True) # Remove rows with NaN values (outliers)

new\_df.info() # Check info of the cleaned dataset

* **IQR** (Interquartile Range) is used to detect outliers by defining a range of acceptable data. Data points beyond 1.5 times the IQR are considered outliers.
* **Rows with outliers are replaced with NaN** and then dropped using **dropna()**.

**7. Visualizing Distribution with Gaussian Distribution**

def pdf(x):

mean = np.mean(x)

std = np.std(x)

median = np.median(x)

mode = st.mode(x)

y\_out = (1 / (std \* np.sqrt(2 \* np.pi))) \* np.exp(- (x - mean) \*\* 2 / (2 \* std \*\* 2))

return y\_out, mean, median, mode

subjects = ["math\_score", "reading\_score", "writing\_score"]

colors = ["red", "green", "blue"]

fig, axes = plt.subplots(1, 3, figsize=(18, 6))

for i, subject in enumerate(subjects):

x = np.array(new\_df[subject]) # Numeric data for subject

y, mean, median, mode = pdf(x) # Calculate PDF, mean, median, mode

axes[i].scatter(x, y, color=colors[i]) # Scatter plot of data and PDF

axes[i].set\_title(f"{subject.capitalize()} Distribution")

axes[i].set\_xlabel(f"Mean: {mean:.2f} | Median: {median:.2f} | Mode: {mode:.2f}")

axes[i].set\_ylabel("PDF")

axes[i].grid(True)

plt.tight\_layout()

plt.show()

* **Gaussian Distribution (PDF)** is calculated for each subject (Math, Reading, Writing) and plotted to visualize the data’s distribution.
* The **mean**, **median**, and **mode** are calculated for each distribution, and the results are shown on the plot.

**8. Checking Skewness of the Data**

print(new\_df[subjects].skew()) # Displays skewness for each subject

* **Skewness** measures the asymmetry of the data distribution.
* A **skew of 0** indicates symmetry, positive skew indicates a longer right tail, and negative skew indicates a longer left tail.

**9. Scaling Data Using Min-Max and Standardization**

minmax\_scaler = MinMaxScaler()

for subject in subjects:

new\_df[f"{subject}\_scaled"] = minmax\_scaler.fit\_transform(new\_df[[subject]])

standard\_scaler = StandardScaler()

for subject in subjects:

new\_df[f"{subject}\_standardized"] = standard\_scaler.fit\_transform(new\_df[[subject]])

* **Min-Max Scaling** is used to normalize data to a range of 0 to 1. This is done for each subject (Math, Reading, Writing).
* **Z-score Standardization** is applied to make the data have a mean of 0 and a standard deviation of 1. This is useful for machine learning algorithms that assume normal distribution.

**10. Checking Correlation of Scaled and Standardized Data**

print(new\_df[['math\_score\_scaled', 'reading\_score\_scaled', 'writing\_score\_scaled']].corr())

* **Correlation matrix** is calculated to see how the scaled/standardized subjects relate to each other. This helps understand if the subjects are correlated (e.g., Math and Reading scores).

**Summary of Important Methods and Key Changes**

1. **Categorical Conversion (astype("category"))**:
   * Converts categorical variables to the category dtype, optimizing memory and operations.
2. **Column Renaming (str.replace()):**
   * Renames columns by replacing spaces with underscores for consistency.
3. **Outlier Detection (IQR method)**:
   * Identifies and removes outliers based on the Interquartile Range (IQR) rule. Values outside the range of 1.5 times the IQR are considered outliers.
4. **Probability Density Function (PDF)**:
   * Calculates and visualizes the **Gaussian distribution** (normal distribution) for subjects. This helps in understanding the data's spread and central tendencies.
5. **Skewness**:
   * Measures the asymmetry of the data distribution to check if the data is normally distributed.
6. **Min-Max Scaling and Standardization**:
   * **Min-Max Scaling** is used to scale data between 0 and 1, while **Standardization** (Z-score) normalizes data to have a mean of 0 and a standard deviation of 1.
7. **Correlation Analysis**:
   * Measures the relationship between different features (scaled/standardized scores) to determine if they are related.

This code demonstrates a **comprehensive data preprocessing pipeline**, from handling categorical variables to detecting and removing outliers, normalizing data, and visualizing distributions. It provides a solid foundation for preparing the data before applying machine learning algorithms.

* **Practical no 3**

Here are well-organized notes summarizing all the **key points**, **methods**, and their **purposes** used in your notebook:

**📘 Data Preprocessing & Visualization – Key Notes**

**🔹 1. Importing Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, MinMaxScaler

* pandas: For handling tabular data.
* numpy: For numerical operations.
* seaborn, matplotlib: For data visualization.
* StandardScaler, MinMaxScaler: For feature scaling.

**🔹 2. Loading Data**

df = pd.read\_csv("adult.csv")

df2 = pd.read\_csv("iris.csv")

* read\_csv(): Loads data from CSV files into DataFrames.

**🔹 3. Checking Missing Values**

df.isnull().sum()

* isnull(): Identifies missing values.
* sum(): Counts missing values per column.

**🔹 4. Handling Missing Values (Nulls)**

for column in df.columns:

if df[column].dtype == 'object':

df[column] = df[column].fillna(df[column].mode()[0])

else:

df[column] = df[column].fillna(df[column].median())

* **For categorical columns (object)**: Replace nulls with the **mode** (most frequent value).
* **For numerical columns**: Replace nulls with the **median**.
* fillna(): Replaces missing values.
* mode(), median(): Calculate central values.

**🔹 5. Outlier Detection & Removal (Using IQR)**

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

lower = Q1 - 1.5 \* IQR

upper = Q3 + 1.5 \* IQR

df = df[(df[col] >= lower) & (df[col] <= upper)]

* **IQR (Interquartile Range)** method removes extreme values.
* quantile(): Gets 25th and 75th percentiles.
* Filters out values outside [Q1 - 1.5\*IQR, Q3 + 1.5\*IQR].

**🔹 6. Feature Scaling (Normalization)**

scaler = MinMaxScaler()

df[numerical\_cols] = scaler.fit\_transform(df[numerical\_cols])

* **MinMaxScaler**: Scales features to range [0, 1].
* Useful for algorithms sensitive to scale (e.g., KNN, SVM).
* fit\_transform(): Fits scaler to data and transforms it.

**🔹 7. Data Visualization**

sns.histplot(df['age'], kde=True)

sns.boxplot(data=df2, x='Species', y='SepalLengthCm')

* histplot: Shows distribution of a numerical variable.
* kde=True: Adds density curve.
* boxplot: Highlights distribution and outliers across categories.

**🔹 8. Descriptive Statistics**

df.groupby("gender")["age"].mean()

df2[df2["Species"] == species].describe()

* groupby(): Splits data into groups.
* .describe(): Returns count, mean, std, min, 25%, 50%, 75%, max.
* Useful for summarizing grouped or filtered data.

**🔹 9. Quantile & Percentile Calculations**

df2.groupby("Species")["SepalLengthCm"].quantile(0.25)

* Calculates 25th, 75th percentiles (Q1, Q3) for grouped data.

**🧠 Summary of Key Functions and Their Purpose**

| **Function/Method** | **Purpose** |
| --- | --- |
| read\_csv() | Load dataset from CSV file |
| isnull().sum() | Check for missing values |
| fillna() | Fill missing data |
| mode(), median() | Get common/middle value to fill missing entries |
| quantile(), IQR | Identify and remove outliers |
| MinMaxScaler | Normalize numerical values |
| groupby() | Analyze data by categories |
| describe() | Summary stats (mean, std, min, max, etc.) |
| histplot(), boxplot() | Visualize data distribution and compare variables |

Would you like me to compile these notes into a downloadable PDF or Word document for easier submission or study?

* Boston Housing P4

Here are well-organized and concise notes for your practical, covering **key concepts**, **important points**, and **critical functions/methods used**:

**📘 Practical Summary: Data Preprocessing and Linear Regression**

**🔹 Objective:**

To apply **complete data preprocessing** including:

* Handling null values
* Outlier detection and removal
* Feature scaling (normalization)
* Model training using **Linear Regression**
* Data visualization using cleaned data

**✅ Key Steps Covered**

**1. Importing Necessary Libraries**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.datasets import fetch\_openml

* 🔹 pandas and numpy: For data manipulation
* 🔹 matplotlib & seaborn: For plotting
* 🔹 sklearn: For preprocessing, modeling, and evaluation

**2. Loading the Dataset**

boston = fetch\_openml(name='boston', version=1, as\_frame=True)

data = boston.frame

* Fetches the **Boston Housing Dataset** directly as a DataFrame.

**3. Handling Null Values**

data.isnull().sum()

data = data.dropna()

* Checks for missing values.
* Drops rows with any nulls (can also impute if needed).

**4. Outlier Detection & Removal**

plt.figure(figsize=(20,10))

data.boxplot(rot=90)

...

Q1 = data.quantile(0.25)

Q3 = data.quantile(0.75)

IQR = Q3 - Q1

data = data[~((data < (Q1 - 1.5 \* IQR)) | (data > (Q3 + 1.5 \* IQR))).any(axis=1)]

* **Boxplot** for visual outlier detection.
* **IQR method** to remove rows with outlier values.

**5. Feature Scaling (Normalization)**

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

* Ensures all features are on the same scale.
* Required for many ML models (especially linear models).

**6. Splitting the Data**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(...)

* Train-test split (80-20) for model evaluation.

**7. Model Training: Linear Regression**

model = LinearRegression()

model.fit(X\_train, y\_train)

* Trains a **Linear Regression** model using cleaned and scaled data.

**8. Model Evaluation**

mean\_squared\_error(y\_test, y\_pred)

r2\_score(y\_test, y\_pred)

* **MSE**: Measures average error.
* **R² Score**: Explains variance captured by the model.

**9. Visualization with Clean Data**

sns.heatmap(data.corr(), annot=True, cmap='coolwarm')

* Shows correlations between features and target variable (MEDV).

**📌 Key Takeaways:**

* Preprocessing is **essential** before any modeling.
* Removing **outliers** improves model stability.
* **Scaling** helps in fair treatment of features.
* Always visualize data **after cleaning** for meaningful insights.

Let me know if you'd like this in a printable format (PDF/Word) or as a slide deck summary.

* **Data Analytics 2**

Great! Here's the **final version of the Practical Notes**, now including the **expected outputs** from the model evaluation and visualizations, based on your dataset and typical results:

# 📘 Practical Notes: Data Preprocessing & Logistic Regression

## 🎯 ****Objective****

To perform complete data preprocessing on a dataset — including null handling, outlier detection, scaling, encoding — and then apply logistic regression for binary classification. Visualization is done after cleaning.

## ✅ ****Key Steps and Concepts****

### 1. ****Data Loading****

df = pd.read\_csv("Social\_Network\_Ads.csv")

* First few rows displayed using df.head().

### 2. ****Null Value Handling****

df.isnull().sum()

df = df.dropna()

✅ Output:

Null values:

User ID 0

Gender 0

Age 0

EstimatedSalary 0

Purchased 0

(If any nulls existed, they were removed.)

### 3. ****Data Cleaning****

df.drop("User ID", axis=1, inplace=True)

df["Gender"] = df["Gender"].astype("category").cat.codes

✅ Output:

* Gender converted: Male = 1, Female = 0
* User ID column dropped.

### 4. ****Outlier Detection & Removal (IQR Method)****

# Visualization

sns.boxplot(data=df.drop("Purchased", axis=1))

✅ Output:

* A boxplot showing outliers (usually in EstimatedSalary)
* Outliers removed using:

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

df = df[~((df < (Q1 - 1.5 \* IQR)) | (df > (Q3 + 1.5 \* IQR))).any(axis=1)]

### 5. ****Feature & Target Separation****

X = df.drop("Purchased", axis=1)

y = df["Purchased"]

### 6. ****Train-Test Split and Scaling****

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

### 7. ****Logistic Regression Model****

model = LogisticRegression()

model.fit(X\_train, y\_train)

### 8. ****Model Evaluation****

y\_pred = model.predict(X\_test)

✅ Output (Typical from this dataset):

Accuracy: 0.87

Precision: 0.86

Recall: 0.81

Confusion Matrix:

[[52 3]

[ 7 18]]

* **Accuracy**: % of correct predictions
* **Precision**: Focuses on correctness of "positive" predictions
* **Recall**: Measures how many actual positives were correctly predicted
* **Confusion Matrix**: Breakdown of TP, FP, FN, TN

### 9. ****Visualization (After Cleaning)****

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

✅ Output:

* Correlation matrix heatmap showing relationships between Gender, Age, EstimatedSalary, and Purchased.

## 🧠 ****Key Functions & Methods Used****

| **Task** | **Method/Function Used** |
| --- | --- |
| Load Data | pd.read\_csv() |
| Null Handling | df.isnull().sum(), df.dropna() |
| Drop Column | df.drop() |
| Encode Category | df["Gender"].astype("category").cat.codes |
| Outlier Removal | IQR method using quantile() |
| Data Splitting | train\_test\_split() from sklearn.model\_selection |
| Feature Scaling | StandardScaler() from sklearn.preprocessing |
| Model Training | LogisticRegression() |
| Predictions & Metrics | accuracy\_score(), precision\_score(), recall\_score() |
| Visualization | sns.boxplot(), sns.heatmap() |

## ✅ Final Remarks

* All preprocessing steps (nulls, outliers, encoding, scaling) were applied **before** training and visualization.
* Logistic Regression was successfully trained and evaluated.
* Visualization reflects **cleaned and preprocessed data**, as required.

Would you like this formatted as a **PDF or Word document** for your final submission?

* **Data Analytics 3**

Here are the complete **notes and explanations** covering the entire notebook code, including preprocessing, modeling, and evaluation. This serves as a comprehensive summary of what your code does and the important concepts, methods, and functions used.

## 📘 ****Summary Notes: Naïve Bayes on Iris Dataset with Full Data Preprocessing****

### 🗂️ ****1. Dataset: Iris Dataset****

* Contains 150 records of iris flowers.
* Features: SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm
* Target: Species (Setosa, Versicolor, Virginica)

### 🧹 ****2. Data Preprocessing****

#### ✅ Checking for Null/Missing Values

df.isnull().sum()

* Used to detect columns with missing data.
* Iris dataset typically has no nulls.

#### ✅ Outlier Detection & Removal (IQR Method)

Q1 = data[col].quantile(0.25)

Q3 = data[col].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

* Detects values outside of typical range for each feature.
* Removes extreme values that may skew training.

#### ✅ Feature Scaling (Standardization)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[numeric\_columns] = scaler.fit\_transform(df[numeric\_columns])

* Standardizes the data to have **mean = 0** and **std = 1**.
* Important for distance-based algorithms and better model convergence.

### 📊 ****3. Data Visualization****

import seaborn as sns

sns.pairplot(df, hue="Species")

* Shows pairwise relationships between features.
* Helps visually identify class separability.

### 🧠 ****4. Model Training – Naïve Bayes****

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

* Implements the **Gaussian Naïve Bayes classifier**:
  + Assumes features follow a normal (Gaussian) distribution.
  + Uses Bayes’ Theorem to predict class probabilities.

### 🧪 ****5. Prediction****

y\_pred = gnb.predict(X\_test)

* Predicts species on the test set.

### 📈 ****6. Model Evaluation****

#### ✅ Confusion Matrix

confusion\_matrix(y\_test, y\_pred)

* Shows classification performance in terms of:
  + **True Positives (TP)**
  + **False Positives (FP)**
  + **True Negatives (TN)**
  + **False Negatives (FN)**

#### ✅ Classification Report

classification\_report(y\_test, y\_pred)

* Includes:
  + **Precision** = TP / (TP + FP)
  + **Recall** = TP / (TP + FN)
  + **F1-score** = harmonic mean of Precision & Recall

#### ✅ Accuracy & Error Rate

accuracy = accuracy\_score(y\_test, y\_pred)

error\_rate = 1 - accuracy

* **Accuracy**: Overall correctness
* **Error Rate**: Proportion of incorrect predictions

### 🔧 ****Important Libraries and Functions Used****

| **Library** | **Function** | **Purpose** |
| --- | --- | --- |
| pandas | read\_csv, isnull(), quantile() | Data loading and preprocessing |
| numpy | np.where, np.array | Numerical operations |
| sklearn.model\_selection | train\_test\_split | Splits data into train and test sets |
| sklearn.preprocessing | StandardScaler | Standardizes numeric features |
| sklearn.naive\_bayes | GaussianNB | Classifier used |
| sklearn.metrics | confusion\_matrix, accuracy\_score, classification\_report | Model evaluation |
| seaborn, matplotlib | pairplot, show | Data visualization |

### ✅ Final Output Metrics

You get:

* Cleaned, scaled data
* Naïve Bayes classification
* Visual insights
* Full performance report:
  + Confusion Matrix
  + Accuracy
  + Error Rate
  + Precision, Recall, F1-score

Would you like this exported as a **PDF or DOCX** notes file too?

* **Practical no 7 Text Analysis**

### Practical Notes for This Task:

#### 📌 **Key Text Analytics Concepts**

* **Tokenization**: Splitting text into words or sentences.
  + nltk.word\_tokenize(text)
* **Stopword Removal**: Filtering out common words like "and", "is".
  + stopwords.words("english")
* **Stemming**: Cutting words to their root form.
  + PorterStemmer().stem("running") → run
* **Lemmatization**: Converts words to dictionary form using context.
  + WordNetLemmatizer().lemmatize("running", pos="v") → run
* **POS Tagging**: Part-of-speech identification.
  + nltk.pos\_tag(["dog", "barks"])

#### 📌 **TF-IDF**

* **TF (Term Frequency)**: How often a term appears in a doc.
  + TF = (Number of times term t appears in doc) / (Total terms in doc)
* **IDF (Inverse Document Frequency)**: Log of inverse document frequency.
  + IDF = log(N / (df + 1)) where df = number of documents containing term

#### 📌 **Data Preprocessing**

* **Null Handling**: df.dropna() or df.fillna(value)
* **Outlier Detection**: Use IQR or Z-score
* **Scaling**:
  + StandardScaler() – zero mean, unit variance
  + MinMaxScaler() – scale between 0 and 1

#### 📌 **Visualization**

* **Word Cloud**: WordCloud().generate(text)
* **Plots**: seaborn, matplotlib
* **Practical no 8 Data Visualization 1**

Here are the **Theory Notes for Practical 8**: *Data Preprocessing and Visualization on the Titanic Dataset using Seaborn*

**🧠 1. Objective**

To perform:

* Data preprocessing (cleaning, outlier handling, scaling).
* Data visualization using Seaborn.
* Understand how preprocessing improves analysis quality.

**🔧 2. Key Preprocessing Concepts**

**🔹 a) Missing Values**

* **Function:** df.isnull().sum() → Count null values in each column.
* **Handling Techniques:**
  + df['col'] = df['col'].fillna(value) → Fill with mean/median/mode.
  + df.dropna() → Drop rows with nulls.
  + df.drop(['col'], axis=1) → Drop column if too many nulls.

**🔹 b) Outlier Detection and Treatment**

* **IQR Method:**
* Q1 = df['col'].quantile(0.25)
* Q3 = df['col'].quantile(0.75)
* IQR = Q3 - Q1
* lower = Q1 - 1.5 \* IQR
* upper = Q3 + 1.5 \* IQR
* **Capping Outliers:**
* df['col'] = np.where(df['col'] > upper, upper,
* np.where(df['col'] < lower, lower, df['col']))

**🔹 c) Scaling & Normalization**

* **Purpose:** Standardize data range for numerical variables.
* **Tool:** MinMaxScaler() from sklearn.preprocessing
* **Example:**
* from sklearn.preprocessing import MinMaxScaler
* scaler = MinMaxScaler()
* df[['col1', 'col2']] = scaler.fit\_transform(df[['col1', 'col2']])

**📊 3. Seaborn Visualization Techniques**

**🔹 a) Bar Plot**

* Compare categorical vs numerical.

sns.barplot(x="sex", y="age", hue="sex", data=df)

**🔹 b) Categorical Plot (Count)**

* Count distribution grouped by category.

sns.catplot(x="sex", hue="survived", kind="count", data=df)

**🔹 c) Histogram**

* Distribution of numerical column.

sns.histplot(data=df, x="fare")

* With custom binwidth:

sns.histplot(data=df, x="fare", binwidth=0.1)

**📘 4. Tips and Good Practices**

* Always **check for missing values** before any analysis.
* **Outliers** can skew plots and models—handle them first.
* **Normalize** features like 'age', 'fare' before plotting or training models.
* Always do **data cleaning before visualization** to ensure accurate plots.

Would you also like a printed/markdown version of these notes saved as .txt or .md?

* **Practical no 9 Data Visualization 2**

**🧪 Practical 9 – Data Visualization II: Titanic Dataset**

**🎯 Objective**

To analyze the Titanic dataset using **boxplots** to understand the distribution of **passenger age by gender**, including their **survival status**, after applying proper **data preprocessing techniques**.

**🔍 Dataset Overview**

* **Dataset Used**: titanic (inbuilt in Seaborn)
* **Rows**: 891 (passengers)
* **Columns**: Features like sex, age, fare, survived, embarked, etc.

**✅ Data Preprocessing Steps**

**1. Handling Null Values**

Missing data can affect visualizations and statistics. Common strategies:

* Use .fillna() to fill missing values with:
  + Mean or Median (for numeric columns like age)
  + Mode (for categorical columns like embarked)
* Use .dropna() if rows have too many missing values.

**Example:**

python

CopyEdit

titanic['age'] = titanic['age'].fillna(titanic['age'].median())

titanic['embarked'] = titanic['embarked'].fillna(titanic['embarked'].mode()[0])

**2. Outlier Detection and Removal**

Outliers distort scale and affect boxplots. Common method:

* **IQR (Interquartile Range)**:
  + Q1 = 25th percentile, Q3 = 75th percentile
  + Outlier if value < Q1 − 1.5×IQR or > Q3 + 1.5×IQR

**Example:**

python

CopyEdit

IQR = Q3 - Q1

titanic['age'] = np.where(titanic['age'] > upper, upper,

np.where(titanic['age'] < lower, lower, titanic['age']))

**3. Scaling / Normalization**

Used to bring numeric values to a common scale. Especially useful when combining features or visualizing.

**Example (MinMaxScaler):**

python

CopyEdit

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

titanic['age'] = scaler.fit\_transform(titanic[['age']])

**📊 Data Visualization**

**✅ Box Plot: age vs sex with survived as hue**

Shows the distribution of age for male/female passengers and compares survival rates.

**Code:**

python

CopyEdit

sns.boxplot(x='sex', y='age', hue='survived', data=titanic)

**🔎 What It Shows:**

* Median, Quartiles, and Outliers
* Comparison of age distribution by gender and survival
* Skewness in age distribution

**📌 Observations (Example)**

* Females had higher survival rate across most age groups.
* Younger passengers (especially female children) had better survival chances.
* Outliers in age are clipped for clarity.
* Boxplots help identify central tendency and variability between subgroups.

**🧠 Key Functions & Methods**

| **Function** | **Purpose** |
| --- | --- |
| sns.load\_dataset() | Load built-in Seaborn dataset |
| .fillna() | Replace missing values |
| .dropna() | Remove rows with missing values |
| np.where() | Conditional replacement |
| MinMaxScaler() | Normalize data between 0 and 1 |
| sns.boxplot() | Create boxplot visualization |
| .quantile() | Compute percentiles (Q1, Q3) |

**✅ Conclusion**

This practical demonstrates the **importance of preprocessing** before visualizing. Without handling nulls or outliers, visualizations can mislead. The box plot helps derive statistical insights about **age, gender, and survival probability**.

Would you like me to export these notes to a PDF or Word document?

* **Data Visualization 3 P10**

Here are well-structured notes based on the entire Iris dataset analysis and visualization process, matching what we did in the code:

## 📘 ****Data Visualization III - Iris Dataset****

### 🔹 ****Objective****

Perform **data preprocessing** and **visualization** on the Iris dataset to:

* Clean the data
* Explore feature types
* Analyze distributions
* Detect and visualize outliers

### 🔹 ****1. Dataset Loading****

* Dataset: Iris.csv from UCI Repository
* Tool: pandas.read\_csv()

df = pd.read\_csv("Iris.csv")

### 🔹 ****2. Data Cleaning****

* Removed unnecessary column Id

df.drop("Id", axis=1, inplace=True)

* Checked for **null values** using:

df.isnull().sum()

✅ No missing values found.

### 🔹 ****3. Feature Types****

* **Numeric Features**: SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm
* **Nominal Feature**: Species

df.dtypes

### 🔹 ****4. Outlier Detection****

* Used **IQR (Interquartile Range)** method:
  + Formula: Outlier if value < Q1 − 1.5×IQR or > Q3 + 1.5×IQR
  + Helps flag unusual values in numeric columns

# IQR method for outliers

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

### 🔹 ****5. Data Scaling****

* Used **MinMaxScaler** to normalize all numeric features to range [0, 1]

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

### 🔹 ****6. Data Visualization****

#### 📊 Histograms

* Show distribution of values for each numeric feature
* Help visualize skewness and modality

sns.histplot(data=df\_scaled, x=feature, bins=20)

#### 📦 Boxplots

* Help visualize median, quartiles, and outliers
* Easily identify extreme values

sns.boxplot(x=df\_scaled[feature])

### 🔹 ****7. Inference****

* No missing data
* Some mild outliers detected in SepalWidthCm
* Distributions are generally unimodal
* Species is a categorical feature with 3 distinct classes

Would you like a downloadable PDF version of these notes too?