3 Descriptive Statistics - Measures of Central Tendency and variability Perform the following operations on any open source dataset (e.g., data.csv) 1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable

1. **Descriptive Statistics**: These are methods for summarizing or describing a set of data. It includes:
    - **Measures of Central Tendency**: Indicates the center of a data set.
        - **Mean**: The average value.
        - **Median**: The middle value when data is ordered.
        - **Mode**: The most frequent value.
    - **Measures of Variability (Spread)**: Shows how spread out the data is.
        - **Minimum** and **Maximum**: The smallest and largest values.
        - **Standard Deviation (std)**: Tells how much the values deviate from the mean.
        - **Percentiles**: Values below which a certain percentage of observations fall (e.g., 25th, 50th, 75th percentiles).
2. **Categorical Variables**: These are variables that represent categories (e.g., gender, age group).
3. **Quantitative Variables**: These are numeric variables (e.g., income, height).

✅ 1. Summary Statistics Grouped by a Categorical Variable

🔹 **Objective:**

Use a dataset (e.g., `data.csv`) with a categorical variable like "age group" and a numeric variable like "income". Group by the categorical variable and compute summary statistics.

🔹 **Sample Dataset:**

Assume this content is in `data.csv`:

```
Name,AgeGroup,Income
Alice,Young,45000
Bob,Middle-aged,55000
Charlie,Young,47000
David,Senior,30000
Eva,Middle-aged,60000
Frank,Senior,32000
Grace,Young,49000
```

🔹 **Python Code:**

```
import pandas as pd

# Load the dataset
df = pd.read_csv("data.csv")

# Group by categorical variable "AgeGroup" and describe "Income"
grouped = df.groupby("AgeGroup")["Income"].describe()
print(grouped)
```

🔹 **Output:**

```
              count    mean     std     min     25%     50%     75%
max
AgeGroup
Middle-aged       2.0  57500.0  3535.53  55000.0  56250.0  57500.0
58750.0  60000.0
Senior            2.0  31000.0  1414.21  30000.0  30500.0  31000.0
31500.0  32000.0
Young             3.0  47000.0  2000.00  45000.0  46000.0  47000.0
48000.0  49000.0
```

🔹 **Explanation:**

1. **`groupby("AgeGroup")`**: Groups the dataset by each unique age group.

2. `["Income"].describe()`: Computes statistics for the income of each group:
    - `count`: Number of people in the group.
    - `mean`: Average income.
    - `std`: Spread of income around the mean.
    - `min`, `25%`, `50% (median)`, `75%`, `max`: Show distribution of values.

✅ 2. Statistical Summary for Iris Dataset by Species

The Iris dataset is a classic dataset in machine learning. It has 3 species:

- Iris-setosa
- Iris-versicolor
- Iris-virginica

Each species has values for:

- SepalLength
- SepalWidth
- PetalLength
- PetalWidth

🔹 **Python Code:**

```
import pandas as pd

# Load the dataset
df = pd.read_csv("iris.csv")

# Group by species and describe statistics
summary = df.groupby("species").describe()
print(summary)
```

🔹 **Output Sample:**

```
              sepal_length
\
                   count mean       std  min   25%  50%   75%  max
species
Iris-setosa          50.0  5.01   0.3525  4.3  4.8  5.0  5.2  5.8
Iris-versicolor      50.0  5.94   0.5162  4.9  5.6  5.9  6.3  7.0
Iris-virginica       50.0  6.59   0.6359  4.9  6.2  6.5  6.9  7.9

... (same for sepal_width, petal_length, petal_width)
```

🔹 **Explanation:**

1. **`groupby("species")`**: Splits the dataset into three parts based on species.
2. **`.describe()`**: Applies summary statistics to each species for all features.
3. You get:
    - Central tendencies (`mean`, `50%`)
    - Spread (`std`, `min`, `max`)
    - Percentile distributions (`25%`, `75%`)

4 Data Analytics I Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.

✳️ Terminologies Explained

1. **Linear Regression**: A supervised machine learning algorithm that models the relationship between a dependent variable (target) and one or more independent variables (features) using a straight line.

2. **Independent Variables (Features)**: Input variables that influence the output (e.g., `CRIM`, `RM`, `LSTAT`).

3. **Dependent Variable (Target)**: Output we are predicting – here, it's `MEDV` (Median value of owner-occupied homes in $1000s).

4. **Training & Testing Sets**:
    - **Training Set**: Used to train the model.
    - **Testing Set**: Used to evaluate the model's performance.

5. **R² Score**: Metric to evaluate how well the predictions match the actual values (closer to 1 is better).

✅ Step-by-step Code using Python

🔹 Step 1: Import Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_boston
import matplotlib.pyplot as plt
import seaborn as sns
```

> Note: `load_boston` is deprecated in newer versions of scikit-learn. You can instead download it from Kaggle or use a `.csv`.

🔹 **Step 2: Load Dataset**

```
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['MEDV'] = boston.target
```

🔹 **Step 3: Exploratory Data Analysis (EDA)**

```
print(df.head())
print(df.describe())
sns.pairplot(df[['RM', 'LSTAT', 'MEDV']])
plt.show()
```

- `RM`: Average number of rooms per dwelling
- `LSTAT`: % lower status population
- `MEDV`: Median value of home (target)

🔹 **Step 4: Split Data**

python
CopyEdit

```
X = df.drop('MEDV', axis=1)
y = df['MEDV']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

🔹 **Step 5: Train Linear Regression Model**

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

🔹 **Step 6: Predict and Evaluate**

python
CopyEdit

```
y_pred = lr.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")
```

🔹 **Step 7: Visualize Prediction vs Actual**

python
CopyEdit

```
plt.scatter(y_test, y_pred)
plt.xlabel("Actual MEDV")
plt.ylabel("Predicted MEDV")
plt.title("Actual vs Predicted Home Prices")
plt.plot([0, 50], [0, 50], color='red')
plt.show()
```

✅ Sample Output (may vary):

yaml
CopyEdit

```
Mean Squared Error: 24.29
R² Score: 0.71
```

This means your model explains ~71% of the variance in house prices.

5 Data Analytics II 1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset. 2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset

✅ Objective:

1. Build a **Logistic Regression classifier** to predict if a user purchases a product based on features like Age and Estimated Salary.

2. Compute evaluation metrics: **TP**, **FP**, **TN**, **FN**, **Accuracy**, **Error Rate**, **Precision**, **Recall** using the **Confusion Matrix**.

✳ Terminologies Explained

1. **Logistic Regression**: A classification algorithm that predicts binary outcomes (like 0/1, yes/no).

2. **Confusion Matrix**: A table used to evaluate the performance of a classification model.

    - **TP (True Positive)**: Correctly predicted 1s

    - **TN (True Negative)**: Correctly predicted 0s

    - **FP (False Positive)**: Predicted 1 when it was 0

    - **FN (False Negative)**: Predicted 0 when it was 1

3. **Accuracy**: `(TP + TN) / Total`

4. **Error Rate**: `(FP + FN) / Total`

5. **Precision**: `TP / (TP + FP)`

6. **Recall (Sensitivity)**: `TP / (TP + FN)`

✅ Step-by-step Code using Python

🔹 Step 1: Import Required Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score
```

🔹 Step 2: Load Dataset

```
df = pd.read_csv("Social_Network_Ads.csv")
print(df.head())
```

Typical columns: `User ID`, `Gender`, `Age`, `EstimatedSalary`, `Purchased`

We'll use **Age** and **EstimatedSalary** as features and **Purchased** as the target.

🔹 Step 3: Preprocess Data

```
X = df[['Age', 'EstimatedSalary']]
y = df['Purchased']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.25, random_state=0)
```

🔹 Step 4: Train Logistic Regression Model

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

🔹 **Step 5: Confusion Matrix and Evaluation Metrics**

```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

accuracy = (tp + tn) / (tp + tn + fp + fn)
error_rate = (fp + fn) / (tp + tn + fp + fn)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print(f"Confusion Matrix:\n{cm}")
print(f"TP: {tp}, FP: {fp}, TN: {tn}, FN: {fn}")
print(f"Accuracy: {accuracy:.2f}")
print(f"Error Rate: {error_rate:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

✅ Sample Output (may vary):

```
Confusion Matrix:
[[64  4]
 [ 7 25]]
TP: 25, FP: 4, TN: 64, FN: 7
Accuracy: 0.89
Error Rate: 0.11
Precision: 0.86
Recall: 0.78
```

6. Data Analytics III 1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset. 2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

✅ Objective:

1. Use the **Naïve Bayes classification algorithm** to classify flower species from the **iris.csv** dataset.

2. Compute **Confusion Matrix**, **TP**, **FP**, **TN**, **FN**, **Accuracy**, **Error Rate**, **Precision**, and **Recall**.

✳ Terminologies Explained

1. **Naïve Bayes Algorithm**: A probabilistic classifier based on **Bayes' Theorem**, assuming independence between features.

2. **Iris Dataset**: Contains 150 samples of iris flowers from 3 species: `setosa`, `versicolor`, `virginica` with 4 features:

    - `sepal length`, `sepal width`, `petal length`, `petal width`.

3. **Confusion Matrix for Multiclass**: Shows how well your model predicted across **multiple classes**.

4. **Metrics for Multiclass**:

    - Use **macro average** or **per-class values** for Precision and Recall.

    - **Accuracy**: Correct predictions / Total predictions.

    - **Precision**: Correctly predicted class i / Total predicted as class i.

    - **Recall**: Correctly predicted class i / Total actual class i.

✅ Step-by-step Code using Python

🔹 **Step 1: Import Libraries**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score
```

🔹 **Step 2: Load Dataset**

python
CopyEdit

```
df = pd.read_csv("iris.csv")
print(df.head())
```

> Ensure the dataset has 4 feature columns and 1 target column (`Species` or similar).

🔹 **Step 3: Split Data**

```
X = df.iloc[:, :-1]  # All columns except the last (features)
y = df.iloc[:, -1]   # Last column (target)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

🔹 **Step 4: Train Naïve Bayes Classifier**

python
CopyEdit

```
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

🔹 **Step 5: Confusion Matrix and Evaluation Metrics**

```
cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')

print("Confusion Matrix:\n", cm)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision (macro avg): {precision:.2f}")
print(f"Recall (macro avg): {recall:.2f}")
```

✅ Sample Output (may vary):

lua
CopyEdit

```
Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
Accuracy: 0.97
Precision (macro avg): 0.97
Recall (macro avg): 0.97
```

🔹 About TP, FP, TN, FN in Multiclass

For multiclass, TP, FP, TN, FN are calculated **per class**. For example:

For class `setosa`:

- TP = model predicts `setosa` and actual is `setosa`
- FP = model predicts `setosa` but actual is not
- FN = model doesn't predict `setosa` but actual is
- TN = everything else

To get detailed values:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

7. Text Analytics 1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization. 2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.

✅ Objective:

1. Preprocess a sample document using:
    - **Tokenization**
    - **POS Tagging**
    - **Stop Words Removal**
    - **Stemming**
    - **Lemmatization**
2. Convert the document into a **TF-IDF vector representation**.

✳️ Terminologies Explained

1. **Tokenization**: Splitting a sentence/document into individual words or terms.
2. **POS Tagging**: Assigning part-of-speech (noun, verb, adjective, etc.) to each token.
3. **Stop Words**: Common words (like “the”, “is”, “and”) that carry little meaning.
4. **Stemming**: Reducing words to their base/root form by chopping suffixes (e.g., "running" → "run").
5. **Lemmatization**: Like stemming, but returns a proper word (e.g., "better" → "good").
6. **TF (Term Frequency)**: How often a word appears in a document.
7. **IDF (Inverse Document Frequency)**: Measures how important a word is by checking its rarity across documents.
8. **TF-IDF**: Product of TF and IDF – gives higher weight to important, uncommon words.

✅ Step-by-Step Code in Python

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

# Sample document
document = "Text analytics is the process of extracting useful
information from text. It involves several preprocessing steps such as
tokenization, stemming, and lemmatization."

# 1. Tokenization
tokens = word_tokenize(document)
print("Tokens:", tokens)

# 2. POS Tagging
pos_tags = pos_tag(tokens)
print("POS Tags:", pos_tags)

# 3. Stop Words Removal
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in
stop_words]
print("Filtered Tokens (No Stopwords):", filtered_tokens)

# 4. Stemming
stemmer = PorterStemmer()
stemmed = [stemmer.stem(word) for word in filtered_tokens]
print("Stemmed Tokens:", stemmed)

# 5. Lemmatization (with POS mapping)
```

```
lemmatizer = WordNetLemmatizer()

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

lemmatized = [lemmatizer.lemmatize(word, get_wordnet_pos(pos)) for
word, pos in pos_tag(filtered_tokens)]
print("Lemmatized Tokens:", lemmatized)
```

🔹 TF-IDF Representation

```
# Sample corpus of documents
corpus = [
    "Text analytics extracts meaningful information from text data.",
    "Preprocessing includes tokenization, stopword removal, and
stemming.",
    "Lemmatization converts words into their base form."
]

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(corpus)

# Display TF-IDF values
df_tfidf = pd.DataFrame(tfidf_matrix.toarray(),
columns=vectorizer.get_feature_names_out())
print("TF-IDF Matrix:\n", df_tfidf)
```

✅ Sample Output (abbreviated):

```
Tokens: ['Text', 'analytics', 'is', 'the', 'process', ...]
POS Tags: [('Text', 'NNP'), ('analytics', 'NNS'), ...]
```

```
Filtered Tokens: ['Text', 'analytics', 'process', 'extracting',
'useful', 'information', ...]
Stemmed Tokens: ['text', 'analyt', 'process', 'extract', 'use',
'inform', ...]
Lemmatized Tokens: ['Text', 'analytics', 'process', 'extract',
'useful', 'information', ...]

TF-IDF Matrix:
       analytics  base  converts  data  ...  text  their  tokenization
words
0          0.438   0.0       0.0   0.5  ...  0.438   0.0          0.0
0.0
1          0.000   0.0       0.0   0.0  ...  0.000   0.0          0.5
0.0
2          0.000   0.5       0.5   0.0  ...  0.000   0.5          0.0
0.5
```

8.1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data. 2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

✳️ Dataset Info (Seaborn's Titanic)

- The Titanic dataset includes columns such as:
    - `survived`: Survival (0 = No, 1 = Yes)
    - `pclass`: Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
    - `sex`, `age`, `sibsp`, `parch`, `fare`, `embarked`, etc.

✅ Step-by-Step Python Code

🔹 Step 1: Load libraries and dataset

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load Titanic dataset
titanic = sns.load_dataset('titanic')

# View the first few rows
print(titanic.head())
```

🔹 Step 2: Explore patterns in the dataset (example visualizations)

1. Survival by gender

```
sns.countplot(x='sex', hue='survived', data=titanic)
plt.title("Survival Count by Gender")
plt.show()
```

2. Survival by class

```
sns.countplot(x='pclass', hue='survived', data=titanic)
plt.title("Survival Count by Passenger Class")
plt.show()
```

3. Age distribution of survivors

```
sns.histplot(data=titanic, x='age', hue='survived', bins=30, kde=True)
plt.title("Age Distribution by Survival")
plt.show()
```

🔹 **Step 3: Plot histogram of 'fare' (Ticket Price)**

python
CopyEdit

```
sns.histplot(data=titanic, x='fare', bins=30, kde=True)
plt.title("Distribution of Ticket Fare")
plt.xlabel("Fare")
plt.ylabel("Number of Passengers")
plt.show()
```

✅ Output (Interpretation):

- **Survival vs Gender**: More women survived than men.
- **Survival vs Class**: First-class passengers had higher survival rates.
- **Fare Histogram**: The fare distribution is **right-skewed**, meaning most passengers paid lower fares, but a few paid very high fares.

9. 1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age') 2. Write observations on the inference from the above statistics.

✅ Objective:

1. Use Seaborn's built-in Titanic dataset.
2. Plot a **box plot** to visualize the **distribution of age**:
    - Grouped by **gender** (`sex`)
    - Separated by **survival status** (`survived`)
3. Write observations based on the plot.

✅ Step-by-Step Python Code

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load Titanic dataset
titanic = sns.load_dataset('titanic')

# Create boxplot: Age distribution by gender and survival
sns.boxplot(data=titanic, x='sex', y='age', hue='survived')
plt.title('Age Distribution by Gender and Survival Status')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.legend(title='Survived', labels=['No', 'Yes'])
plt.show()
```

📊 Box Plot Explanation:

- **X-axis**: Gender (`sex`)
- **Y-axis**: Age (`age`)
- **Hue**: Survival status (`survived`)
    - 0 = Did not survive
    - 1 = Survived

Box plot elements:

- **Box**: Interquartile range (25th to 75th percentile)
- **Line inside the box**: Median (50th percentile)
- **Whiskers**: Show range excluding outliers
- **Dots outside whiskers**: Outliers

✅ Observations (Inferences):

1. **Females**:
    - **Median age of survivors** is slightly **lower** than non-survivors.
    - Younger women had a better chance of survival.
    - The spread (IQR) is more compact for female survivors.

2. **Males**:
    - **Median age of survivors** is noticeably **lower** than male non-survivors.
    - Many older males did not survive.
    - Some young boys also did not survive, indicating survival was not guaranteed even for young males.

3. **General**:
    - The plot reinforces the historical fact that **"women and children first"** was a likely factor in survival outcomes.

    - Survivors in both genders tend to have a lower age median.

10. Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris). Scan the dataset and give the inference as: 1. List down the features and their types (e.g., numeric, nominal) available in the dataset. 2. Create a histogram for each feature in the dataset to illustrate the feature distributions. 3. Create a boxplot for each feature in the dataset. 4. Compare distributions and identify outliers.

✅ Objective:

1. Load the Iris dataset.
2. Identify features and their data types.
3. Plot **histograms** for feature distributions.
4. Plot **box plots** to detect **outliers**.
5. Compare distributions and make inferences.

✅ Step 1: Load the Dataset

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load Iris dataset from seaborn
iris = sns.load_dataset('iris')
print(iris.head())
```

✅ Step 2: List Features and Their Types

```
print("Features and Data Types:\n")
print(iris.dtypes)
```

- **Output:**

Feature	Type
sepal_length	float64 (numeric)
sepal_width	float64 (numeric)
petal_length	float64 (numeric)
petal_width	float64 (numeric)
species	object (nominal)

✅ Step 3: Plot Histograms for Each Feature

```
iris.hist(edgecolor='black', figsize=(10, 6))
plt.suptitle('Histograms of Iris Features', fontsize=16)
plt.tight_layout()
plt.show()
```

✅ Step 4: Boxplots for Each Feature

```
features = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width']
plt.figure(figsize=(12, 8))
for i, feature in enumerate(features):
    plt.subplot(2, 2, i+1)
    sns.boxplot(y=iris[feature])
    plt.title(f'Boxplot of {feature}')
plt.tight_layout()
plt.show()
```

✅ Step 5: Compare Distributions & Identify Outliers

🔹 **Observations:**

1. **Distributions**:
    - Most features are **normally distributed** or slightly skewed.
    - `petal_length` and `petal_width` show **distinct separation** between species (good for classification).
2. **Outliers** (from box plots):
    - `sepal_width` has a few **mild outliers** on both high and low ends.
    - Other features (`sepal_length`, `petal_length`, `petal_width`) are relatively clean with minimal or no outliers.