Contents

[Data Analysis: by SQL (cleaning) and Python(visualization) 1](#_Toc183729031)

[cleaning data by sql 3](#_Toc183729032)

[1. Creating the Database: 3](#_Toc183729033)

[visualize data by python 8](#_Toc183729034)

[Libraries Used: 8](#_Toc183729035)

[Summary: 14](#_Toc183729036)

**Hr data analysis**

**(final project decumentation)**

***Team members***

Mahmoud yasser Osman mohamed

# Data Analysis: by SQL (cleaning) and Python(visualization)

In the ever-evolving world of data science, the combination of SQL for data cleaning and Python for visualization has become an indispensable toolkit for data analysts. This powerful duo allows professionals to efficiently process vast amounts of raw data and transform it into meaningful insights through compelling visual representations. By leveraging the strengths of both SQL and Python, analysts can streamline their workflow, enhance data quality, and create impactful visualizations that drive informed decision-making across various industries. This comprehensive guide will explore the synergy between SQL and Python in the data analysis process, providing you with the knowledge and techniques to elevate your data manipulation and visualization skills. From fundamental concepts to advanced strategies, we'll delve into the intricacies of data cleaning with SQL and the art of data visualization with Python, equipping you with the tools to tackle real-world data challenges effectively. by ENG. Mahmoud Yasser Introduction to Data Analysis 1 2 3 4 Data Collection The first step in data analysis involves gathering relevant information from various sources, such as databases, APIs, or external files. This phase sets the foundation for all subsequent analysis. Data Cleaning Raw data often contains errors, inconsistencies, and missing values. The cleaning



process involves identifying and correcting these issues to ensure data quality and reliability. Data Analysis Once the data is clean, analysts apply statistical methods and machine learning algorithms to uncover patterns, trends, and insights hidden within the dataset.

# **cleaning data by sql**

## **1. Creating the Database:**

sql

CREATE DATABASE EmployeeData;

GO

USE EmployeeData;

**Explanation**: This section creates a new database called EmployeeData and then sets it as the active database for the subsequent operations.

**2. Creating Tables:**

* **EducationLevel Table**:

sql

CREATE TABLE EducationLevel (

EducationLevelID INT PRIMARY KEY,

EducationLevel NVARCHAR(100)

);

**Explanation**: Creates a table to store different education levels. Each row has an ID and a corresponding description of the education level ("Bachelors", "Masters").

* **RatingLevel Table**:

sql

CREATE TABLE RatingLevel (

RatingID INT PRIMARY KEY,

RatingLevel NVARCHAR(100)

);

**Explanation**: Creates a table to store performance ratings ("Needs Improvement", "Meets Expectation").

* **SatisfactionLevel Table**:

sql

CREATE TABLE SatisfactionLevel (

SatisfactionID INT PRIMARY KEY,

SatisfactionLevel NVARCHAR(100)

);

**Explanation**: This table is for storing employee satisfaction levels ( "Satisfied", "Very Dissatisfied").

**3. Inserting Data:**

* **EducationLevel Data**:

sql

INSERT INTO EducationLevel (EducationLevelID, EducationLevel) VALUES

(1, 'No Formal Qualifications'),

(2, 'High School'),

(3, 'Bachelors'),

(4, 'Masters'),

(5, 'Doctorate');

**Explanation**: Inserts predefined education levels into the EducationLevel table.

* **RatingLevel Data**:

sql

INSERT INTO RatingLevel (RatingID, RatingLevel) VALUES

(1, 'Unacceptable'),

(2, 'Needs Improvement'),

(3, 'Meets Expectation'),

(4, 'Exceeds Expectation'),

(5, 'Above and Beyond');

**Explanation**: Inserts performance rating levels.

* **SatisfactionLevel Data**:

sql

INSERT INTO SatisfactionLevel (SatisfactionID, SatisfactionLevel) VALUES

(1, 'Very Dissatisfied'),

(2, 'Dissatisfied'),

(3, 'Neutral'),

(4, 'Satisfied'),

(5, 'Very Satisfied');

**Explanation**: Inserts satisfaction levels for employee feedback.

**4. Employee Table (Partially Displayed):**

sql

CREATE TABLE Employee (

EmployeeID NVARCHAR(50) PRIMARY KEY,

FirstName NVARCHAR(50),

LastName NVARCHAR(50),

...

**Explanation**: This creates a table for employee details like EmployeeID, FirstName, and LastName.

.

data Visualization The final step involves presenting the findings through charts, graphs, and interactive dashboards, making complex information easily understandable for stakeholders. Importance of Data Cleaning 1 Ensuring Data Accuracy Clean data is crucial for accurate analysis and decision making. By removing errors and inconsistencies, analysts can trust the integrity of their insights and avoid costly mistakes based on faulty data. 2 Improving Efficiency Clean data streamlines the analysis process, reducing the time spent on troubleshooting and data reconciliation. This efficiency allows analysts to focus on extracting valuable insights rather than wrestling with data quality issues. 3 Enhancing Data Integration When combining data from multiple sources, clean and standardized data ensures seamless integration. This is especially important in big data environments where diverse datasets need to be merged for comprehensive analysis. 4 Maintaining Compliance In many industries, data cleanliness is not just a best practice but a regulatory requirement. Clean data helps organizations maintain compliance with data protection laws and industry standards. SQL: The Language for Data Manipulation Powerful Querying SQL excels at retrieving and filtering large datasets efficiently. Its declarative nature allows analysts to express complex data requirements in a concise and readable manner.

A person holding a transparent display

Description automatically generated

# **visualize data by python**

## **Libraries Used:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

**pandas**: For data manipulation and analysis.

**numpy**: For advanced numerical computations (though it is not directly used here).

**matplotlib**: For creating visualizations and plots.

**Detailed Code Walkthrough:**

**1. Plotting the Distribution of Relationship Satisfaction:**

relationship\_satisfaction\_distribution = data['RelationshipSatisfaction'].value\_counts()

relationship\_satisfaction\_distribution.plot(kind='bar')

plt.title('Relationship Satisfaction Distribution')

plt.ylabel('Number of Employees')

for i, v in enumerate(relationship\_satisfaction\_distribution):

plt.text(i, v + 1, str(v), ha='center', va='bottom')

plt.show()

* **Purpose:** Displays the number of employees based on their level of satisfaction with relationships at work.
* **Use:** Helps assess how satisfied employees are with interpersonal relationships in the workplace.

**2. Counting Employees with High Job Satisfaction:**

high\_job\_satisfaction\_count = data[data['JobSatisfaction'] >= 4].shape[0]

plt.bar(['High Job Satisfaction'], [high\_job\_satisfaction\_count])

plt.title('Count of Employees with High Job Satisfaction')

plt.ylabel('Number of Employees')

plt.text(0, high\_job\_satisfaction\_count, str(high\_job\_satisfaction\_count), ha='center', va='bottom')

plt.show()

* **Purpose:** Counts the employees with job satisfaction ratings of 4 or 5.
* **Use:** Provides insights into how many employees are highly satisfied with their jobs.

**3. Distribution of Work-Life Balance:**

work\_life\_balance\_count = data['WorkLifeBalance'].value\_counts(normalize=True) \* 100

plt.bar(work\_life\_balance\_count.index.astype(str), work\_life\_balance\_count)

plt.title('Work Life Balance Distribution')

plt.ylabel('Percentage')

for i, v in enumerate(work\_life\_balance\_count):

plt.text(i, v, f'{v:.2f}%', ha='center', va='bottom')

plt.show()

* **Purpose:** Displays the percentage distribution of employees according to their work-life balance.
* **Use:** Helps evaluate how well employees are balancing their personal and professional lives.

**4. Impact of Training on Job Satisfaction:**

training\_effect = data.groupby('TrainingOpportunitiesTaken')['JobSatisfaction'].mean()

training\_effect.plot(kind='bar')

plt.title('Job Satisfaction vs. Training Opportunities Taken')

plt.ylabel('Average Job Satisfaction')

for i, v in enumerate(training\_effect):

plt.text(i, v, f'{v:.4f}', ha='center', va='bottom')

plt.show()

* **Purpose:** Analyzes the relationship between the number of training opportunities and employees' job satisfaction.
* **Use:** Helps understand if providing more training improves employee satisfaction.

**5. Distribution of Training Opportunities:**

training\_satisfaction\_distribution = data['TrainingOpportunitiesWithinYear'].value\_counts()

plt.bar(training\_satisfaction\_distribution.index.astype(str), training\_satisfaction\_distribution)

plt.title('Training Opportunities Distribution')

plt.ylabel('Number of Employees')

for i, v in enumerate(training\_satisfaction\_distribution):

plt.text(i, v + 1, str(v), ha='center', va='bottom')

plt.show()

* **Purpose:** Displays how frequently employees received training during the year.
* **Use:** Helps analyze the availability of training programs for employees.

**6. Relationship Satisfaction vs. Manager Rating Scatter Plot:**

plt.scatter(data['RelationshipSatisfaction'], data['ManagerRating'])

plt.title('Relationship Satisfaction vs. Manager Rating')

plt.xlabel('Relationship Satisfaction')

plt.ylabel('Manager Rating')

for i in range(len(data)):

plt.text(data['RelationshipSatisfaction'][i], data['ManagerRating'][i], str(data['ManagerRating'][i]), fontsize=8)

plt.show()

* **Purpose:** Plots the relationship between employee satisfaction with relationships and manager ratings.
* **Use:** Identifies any potential correlation between the two metrics.

**7. Calculating the Average Manager Rating:**

mean\_manager\_rating = data['ManagerRating'].mean()

plt.bar(['Average Manager Rating'], [mean\_manager\_rating])

plt.title('Average Manager Rating')

plt.ylabel('Rating Level')

plt.text(0, mean\_manager\_rating, f'{mean\_manager\_rating:.2f}', ha='center', va='bottom')

plt.show()

* **Purpose:** Calculates and displays the average manager rating across all employees.
* **Use:** Provides insights into employees' perception of their managers.

**8. Counting Dissatisfied Employees:**

dissatisfied\_employees\_count = data[data['JobSatisfaction'] < 3].shape[0]

plt.bar(['Dissatisfied Employees'], [dissatisfied\_employees\_count])

plt.title('Count of Dissatisfied Employees')

plt.ylabel('Number of Employees')

plt.text(0, dissatisfied\_employees\_count, str(dissatisfied\_employees\_count), ha='center', va='bottom')

plt.show()

* **Purpose:** Counts the number of employees with low job satisfaction (less than 3).
* **Use:** Helps identify employees who may require attention or workplace improvements.

# Summary:

This file provides useful insights into employee satisfaction through data visualizations.

Data Transformation With SQL, analysts can perform sophisticated data transformations, including aggregations, joins, and subqueries. These capabilities are essential for preparing data for analysis and visualization. Database Integration SQL's widespread adoption means it integrates seamlessly with various database systems. This versatility allows analysts to work with diverse data sources and structures across different platforms. SQL Functions for Data Cleaning Function TRIM COALESCE CASE REGEXP\_REPLACE Purpose Remove leading/trailing spaces Handle NULL values Example TRIM(' data ') COALESCE(column, 'default') Conditional data transformation Pattern-based text cleaning CASE WHEN condition THEN result END REGEXP\_REPLACE(column, pattern, replacement) Python: The Power of Data Visualization Versatility Python's extensive library ecosystem provides tools for creating a wide range of visualizations, from simple plots to complex interactive dashboards. Integration Python seamlessly integrates with other data analysis tools and workflows, enabling a smooth transition from data manipulation to visualization. Customization Python offers granular control over visualization elements, allowing analysts to create tailored, publication-quality graphics that effectively communicate insights. Scalability Python's visualization libraries can handle large datasets efficiently, making it suitable for big data applications and real-time data visualization needs. Pandas Library for Data Manipulation Data Structures Pandas introduces powerful data structures like DataFrames and Series, which provide intuitive ways to work with structured data. These structures offer methods for efficient data manipulation and analysis. Data Transformation With Pandas, analysts can easily reshape, merge, and aggregate data. The library's pivoting and melting functions are particularly useful for transforming data between wide and long formats. Data Cleaning Functions Pandas offers a suite of functions for data cleaning, including handling missing values, removing duplicates, and transforming data types. These functions streamline the data preparation process. Integration with Visualization Pandas integrates seamlessly with visualization libraries like Matplotlib and Seaborn, allowing for quick and easy creation of plots directly from DataFrames. Matplotlib and Seaborn for Visualization 1 2 3 4 Choose Library Decide between Matplotlib for low-level control or Seaborn for high-level, statistical visualizations based on your specific needs and the complexity of your data. Prepare Data Ensure your data is in the correct format for visualization, typically as a Pandas DataFrame or NumPy array. Perform any necessary aggregations or transformations. Create Plot Use the appropriate function to create your desired plot type, such as scatter plots, line charts, or heatmaps. Customize colors, labels, and other visual elements as needed. Enhance and Polish Add finishing touches like titles, legends, and annotations. Adjust the layout and styling to ensure the visualization effectively communicates your insights. Combining SQL and Python for Efficient Workflows 1 Data Extraction Use SQL to efficiently query and extract relevant data from databases, leveraging its powerful filtering and joining capabilities. 2 Initial Cleaning Perform basic data cleaning operations in SQL, such as removing duplicates and handling NULL values, to reduce the data volume for Python processing. 3 Data Loading Import the cleaned data into Python using libraries like pandas, which can directly execute SQL queries and load results into DataFrames. 4 Advanced Processing Utilize Python's rich ecosystem of data manipulation libraries for complex transformations, feature engineering, and statistical analysis. 5 Visualization Create insightful visualizations using Python libraries like Matplotlib or Seaborn, leveraging the processed data to generate impactful charts and graphs. Case Study: Applying the Techniques Data Cleaning with SQL In this e-commerce case study, SQL queries were used to clean transaction data by removing duplicates, standardizing product categories, and handling missing customer information. This initial cleaning significantly reduced data inconsistencies. Data Transformation with Python The cleaned data was then imported into Python, where pandas was used for advanced transformations. This included creating new features like customer lifetime value and purchase frequency, enriching the dataset for deeper analysis. Visualization and Insights Finally, Matplotlib and Seaborn were employed to create a comprehensive dashboard. This visualization revealed key insights such as top-selling product categories, customer segmentation based on purchasing behavior, and seasonal sales trends.