رواد مصر الرقمية

وزارة الاتصالات
وتكنولوجيا المعلومات

EDUVERA

E-learning

Mostafa Mohamed Abdelaziz
Youssef Ahmed Ibrahem
Bassel Mohamed ALI
Sama Ahmed Mohamed
Manar Sayed Mohamed

E-Learning System Development and Data Analysis Documentation

Background About E-Learning System Development Dataset

Eduvera is an educational data platform designed to provide clear insights into schools across major Egyptian governorates. The dataset brings together essential information about public and private schools, including student populations, staffing levels, administrative structures, and foundation histories. Eduvera aims to support analysts, educators, and decision-makers by offering organized, reliable data that helps evaluate school performance, resource distribution, and educational diversity across regions.

Initial Data Inspection

Loaded all school-related datasets and inspected **data types, column names, and overall structure**.

Utilized Python functions such as info(), describe(), and head() for a **basic overview of each sheet**.

Checked for missing values, duplicates, and inconsistencies across all datasets.

Overview of Dataset Structure

Schools Sheet

This sheet provides comprehensive operational and administrative details about each school. Beyond basic identifiers, it includes information such as location (governorate), school type (public or private), staffing capacity, and foundational history. These fields help assess resource allocation, student–teacher ratios, and regional differences in school infrastructure. The data is essential for understanding institutional size, historical context, and administrative structure, which supports decision-making related to funding, staffing, and educational planning.

Students Sheet

This sheet contains detailed demographic and academic information about each student. It includes personal identifiers, gender, age, grade level, class assignment, enrollment codes, and status indicators such as attendance condition or special needs. These fields allow analysts to explore patterns in academic performance, enrollment distribution, and the relationship between student characteristics and outcomes. The dataset is key for segmentation, trend analysis across grade levels, and identifying groups that require additional academic or social support.

Employees Sheet

This sheet provides detailed information about school employees, covering both instructional and administrative roles. It includes identity details, gender, job titles, salary levels, experience years, and employment types. These fields allow evaluation of workforce structure, salary distribution, staff experience levels, and gender composition. The dataset enables comparisons between instructors and teaching assistants, analysis of how experience affects performance, and insights into staffing adequacy across schools.

Enrollment Sheet

This sheet outlines the subjects each student is enrolled in, using unique enrollment codes. It connects students to their courses and records their grades. The information is crucial for analyzing academic pathways, subject popularity, class size distribution, and academic performance across different disciplines. It also supports linking student records with performance metrics and evaluating workload balance for instructors based on subject distributions.

Student_Employee Sheet

This sheet defines the instructional relationships between students and the academic staff responsible for teaching or supporting them. Each record links a student to either an instructor or a teaching assistant for a specific subject. The dataset enables analysis of teacher–student distribution, instructional load, and the balance between instructors and assistants. It also supports tracking which staff members influence student outcomes, allowing deeper evaluation of teaching effectiveness across subjects and grade levels.

StudentPerformance Sheet

This sheet provides detailed insights into students' academic outcomes along with lifestyle and behavioral factors that may affect performance. It includes scores across subjects, study hours, participation habits, stress indicators, and other performance-related metrics. The combination of academic and lifestyle data allows for correlation analysis, identification of performance drivers, and detection of at-risk students. This sheet plays a central role in understanding learning patterns, predicting outcomes, and supporting targeted academic interventions.

StaffPerformance Sheet

This sheet contains comprehensive performance records for school employees, covering both instructional quality and administrative behavior. It includes metrics such as performance scores, attendance consistency, task completion, evaluation results, and promotion eligibility indicators. These fields allow assessment of staff productivity, reliability, and professional growth. The dataset is essential for identifying high-performing employees, flagging areas that require training or support, and ensuring fair and data-driven promotion decisions across the school workforce.

Data Cleaning & Preprocessing overview (EXCEL)

The data cleaning and preprocessing phase is a crucial step in preparing the school, students, and staff datasets for analysis and visualization. Its main purpose is to **ensure data quality, consistency, and usability**, enabling accurate insights and reliable dashboards.

Key Objectives:

- **Handle Missing Values:** Identify NULL or missing entries and replace them appropriately to avoid gaps in analysis.
- **Remove Duplicates:** Detect and eliminate duplicate records for students, employees, and enrollments to ensure each entity is represented only once.
- **Drop Irrelevant Columns:** Remove unnecessary or redundant columns that do not contribute to analysis or visualizations.
- **Generate Synthetic Data:** Create factitious names or IDs for employees and students to protect privacy while maintaining realistic datasets for dashboards.
- **Categorize Data:** Introduce new categorical columns (e.g., student level, staff role type, performance groups) to facilitate grouping, filtering, and comparative analysis.
- **Standardization & Formatting:** Normalize text fields, unify date formats, and create calculated columns such as average scores, attendance rates, and student-to-teacher ratios.

Outcome:

The datasets become **clean, consistent, enriched, and analysis-ready**, forming a strong foundation for generating actionable insights, visual dashboards, and reporting on student performance, staff efficiency, and school operations.

Identified Issues and Adjustments: Schools Sheet

The dataset contained inconsistent formats, redundant columns, and missing or invalid values that required data cleaning and standardization

Removed Empty Cells:

All rows containing missing or incomplete school data were removed to ensure data consistency.

Filled Missing Values:

Applied Excel functions (e.g., =IF() and =VLOOKUP()) to fill missing values for Number of Students and Number of Assistants using reference data from other sheets.

Added Foundation Date:

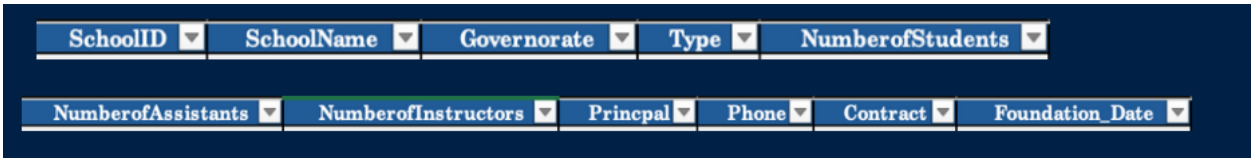
Introduced a new column “Foundation Date” to record when each school was established.

Fixed Data Formats:

1. Standardized numeric columns (e.g., student and assistant counts).
2. Converted all text columns (School Names, Governorates, Principals) to Proper Case using the Excel function =PROPER() for professional formatting.

After Cleaning – Final Columns:

- **Phone Number Formatting:** Ensured all phone numbers start with a leading zero using a formula like:
  =IF(LEFT(A2,1)="0", A2, "0"&A2)
- **Professional Column Naming:** Renamed and unified all column titles for clarity and consistency.



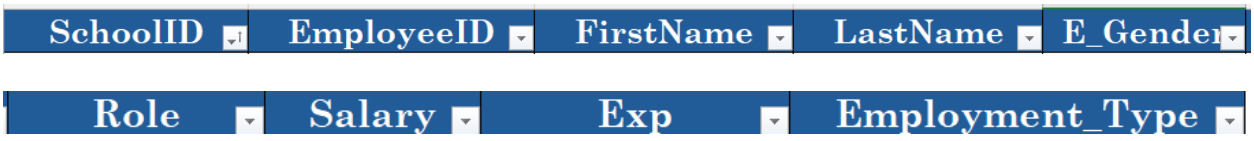
Identified Issues and Adjustments: Employees Sheet

The dataset contained inconsistent formats, missing or invalid entries, and redundant columns that required thorough cleaning and standardization.

- Applied **PROPER case formatting** to all names for consistency (e.g., "mona ali" → "Mona Ali").
- Standardized **salary format** by removing currency symbols and unifying numeric type.
- Adjusted **salary ranges according to role**:
  - Teachers: between 12,000 and 15,000 EGP.
  - Assistants: between 6,000 and 7,500 EGP.
- Ensured **experience values (Exp)** are stored as numbers and fall within valid ranges:
  - Teachers: between 3 and 20 years.
  - Assistants: between 1 and 5 years.
- Removed rows with teachers who have **no assigned students** or missing key data.
- Removed unnecessary columns for clarity.
- Unified **column naming structure** for easy merging and reference.

After Cleaning – Final Columns

These columns represent the cleaned and standardized dataset, containing all essential employee information:



Identified Issues and Adjustments: Students Sheet

The dataset contained inconsistent name capitalization, mixed gender formats, missing city or governorate information, improperly structured Level and Class columns, and empty or missing values that required cleaning and standardization.

- Standardized the **S_Gender** column by converting all entries to either **Male** or **Female**, avoiding mixed formats like M/F.
- Applied the **PROPER** function to unify and correctly capitalize all student names in **FullName**.
- Removed **IDs embedded within names** and organized the data using subtitles for better structure.
- Divided information clearly into **Level** and **Class** columns for easier categorization.
- Filled missing values in **City** and **Governorate** by referencing data from other sheets and ensuring consistency.
- Ensured **Age** is numeric and falls within valid ranges (e.g., 6–18 for school students).
- Verified **EnrollmentCode** for completeness and consistency.
- Standardized **Status** and **Type** columns by unifying labels and correcting inconsistencies.
- Updated **SpecialNeeds** column to use uniform values (Yes/No).
- Removed empty rows or rows containing only NaN values using Excel Filter and Power Query → Remove Blank functions.

After Cleaning – Final Columns

These columns represent the cleaned and standardized dataset, containing all essential student information

Identified Issues and Adjustments: Enrollment Sheet

- Removed **duplicate records** to ensure each enrollment entry is unique.
- Assigned **unique EnrollmentCode values** to all records for clear identification (e.g., SC1, SC2…).
- Specified **Grade** and assigned **Subject** according to the correct academic track level.
- Standardized **Subject** names using **PROPER case formatting** (e.g., "math" → "Math").
- Validated **Grade hierarchy** (1, 2, 3) to align with academic levels.
- Unified **column formats**: consistent text casing, no blanks, and properly aligned codes.
- Verified that each enrollment entry correctly links a student's **Grade** with the assigned **Subject**

Identified Issues and Adjustments: StudentEmployee Sheet

The dataset contained missing links between students and staff, inconsistent formatting, and incomplete associations between subjects and assigned teachers or assistants, which required cleaning and proper linking.

1. Removed all empty rows and blank cells to ensure **data consistency** and eliminate missing values.
2. Checked and corrected any formatting issues, including extra spaces, incorrect data types, or text misalignment.

Data Linking and Enrichment

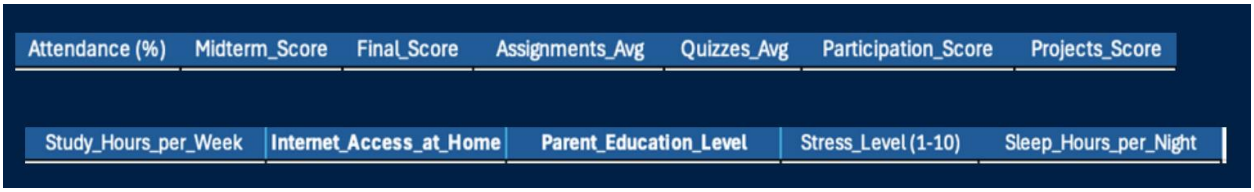
- Ensured that **each student in each subject** is linked to a **Teacher ID** and **Assistant ID**, guaranteeing that every student is associated with the correct staff.
- Some teachers are assigned to **multiple subjects** (e.g., Math 1 and Math 2) and are linked to one or more assistants, reflecting the real teaching assignments.
- Used **Excel formulas (VLOOKUP)** to automatically fetch teacher and assistant information based on IDs.
- Verified that for each subject, the student, teacher, and assistant associations are accurate, maintaining consistent links across all records.

Notes on specific columns:

1. **TeacherID**: Linked each subject to the correct teacher, including those teaching multiple subjects.
2. **AssistantID**: Linked each teacher to one or more assistants, maintaining the teacher–assistant–student relationships.
3. **EnrollmentCode & Subject**: Verified for consistency and accurate mapping with teachers and assistant

Identified Issues and Adjustments: StudentPerformance Sheet

The dataset contained missing or inconsistent numeric values for scores and attendance, unstandardized column names, incomplete student identifiers, and uncalculated performance metrics that required cleaning, standardization, and calculation of averages and total scores.



1. Removed empty rows and unnecessary columns to ensure a clean dataset.
2. Ensured all numeric fields (scores, attendance rates, study hours, etc.) are **properly formatted as numbers**.
3. Standardized **column names** for consistency (e.g., Midterm_Score, Final_Score, Assignments_Avg, Quizzes_Avg, Projects_Score, Total_Score).
4. Used **VLOOKUP** to retrieve missing values such as student names, enrollment codes, or IDs from related sheets like **Students** or **Enrollment**.
5. Applied **IFERROR** to handle invalid lookups or calculation errors gracefully, filling missing or erroneous entries with blank cells or default values.
6. Calculated averages for assignments, quizzes, projects, and other components to provide a better representation of student performance across multiple tasks.

Total Score Calculation

Created a comprehensive formula to combine **weighted scores** from different performance metrics, including **Attendance Rate, Midterm Score, Final Exam Score, Assignments, Quizzes, Participation Score, Projects Score,** ensuring a total score that accurately reflects overall student performance.

Grade Classification

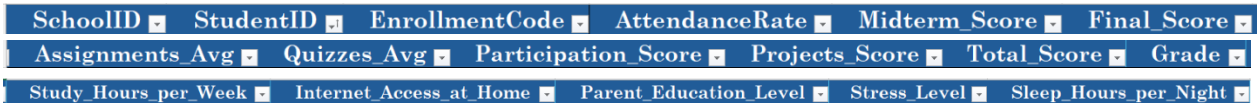
Developed a **grading system** to evaluate each student's performance based on their **Total_Score**.

- Students scoring **90 or above** are assigned **A (Excellent)**
- Scores **80–89** are assigned **B (Very Good)**
- Scores **70–79** are assigned **C (Good)**
- Scores **60–69** are assigned **D (Pass)**
- Scores **below 60** are assigned **F (Fail)**

*This system ensures **consistent and objective classification** of student performance across all records.*

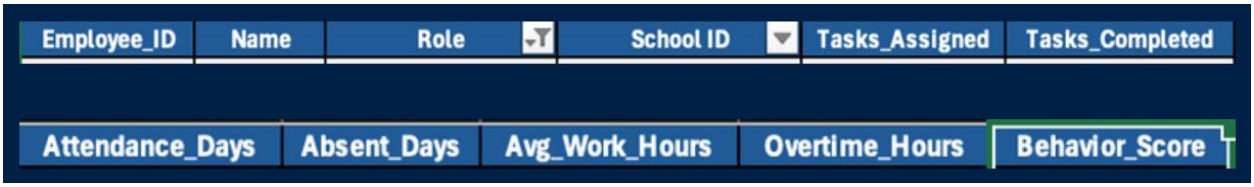
Notes on specific columns:

1. **Total Score:** Calculated to summarize student performance across multiple metrics.
2. **Grade:** Automatically assigned based on total score for consistent classification.
3. **Numeric fields and IDs:** Validated and formatted for consistency with related sheets



Identified Issues and Adjustments: StaffPerformance Sheet

The dataset contained missing values in staff identifiers and names, inconsistent numeric formatting for performance metrics, and unstandardized column names. This required cleaning, standardization, and calculation of overall performance scores for accurate evaluation.



1. Removed empty rows from **FirstName** and **SchoolID**, which previously contained missing or incomplete data.
2. Verified that all numeric fields such as **Tasks_Assigned, Tasks_Completed, Task_Completion_Rate, Attendance_Rate, Behavior_Score, and Performance_Score** were correctly formatted as numbers.
3. Standardized **column names** for consistency, including **EmployeeID, FirstName, Role, Tasks_Assigned, Tasks_Completed, Task_Completion_Rate, Behavior_Score, Performance_Score, Rank**, etc.

Performance Score Calculation

Developed a **weighted scoring system** to calculate each staff member's overall performance based on multiple evaluation metrics:

- **50% weight** for Task Completion Rate
- **30% weight** for Attendance Rate
- **10% weight** for Behavior Score
- **10% weight** for an adjusted attendance ratio combining Attendance Days and Absent Days

This approach ensures that **performance is measured fairly across multiple aspects of staff duties**.

Rank Assignment

Added a **grading system** to classify employees based on their overall Performance Score:

- A → Excellent performance (>95%)
- B → Very Good performance (81–95%)
- C → Good performance (71–80%)
- D → Average performance (61–70%)
- F → Needs Improvement (<60%)

This system provides a **consistent and objective evaluation** of staff performance across all records.

Notes on specific columns:

1. **Performance_Score:** Calculated using a weighted combination of task completion, attendance, behavior, and adjusted attendance ratio.
2. **Rank:** Assigned based on Performance_Score for clear categorization of staff effectiveness.

SchoolID	EmployeeID	FirstName	Role	Tasks_Assigned	Tasks_Completed	Task_Completion_Rate	Attendance_Days

Absent_Days	Attendance_Rate	Avg_Work_Hours	Overtime_Hours	Behavior_Score	Performance_Score	Rank	Promotion_Eligibility

Data Cleaning & Preprocessing Overview (PYTHON)

Schools Sheet

1. Checked the dataset shape to see the number of rows and columns.
2. Previewed the first few rows to understand the data structure.
3. Reviewed column names and their data types.
4. Identified any missing values to plan for data cleaning.

```
First few rows:
SchoolID SchoolName Governorate Type NumberofStudents \
0     SM       Smouha Alexandria Public            320
1     ZH Al-Zahraa Alexandria Public            470
2     SG St. George Alexandria Private          200
3     SJ St. Joseph Alexandria Public           390
4     RC Repton Cairo Alexandria Private        610

 NumberofAssistants NumberofInstructors Principal        Phone Contract  \
0                 22                  12 Ashraf 022540812 3 years
1                 30                  15    Ahmed 021750933 2 years
2                 24                  12     Mina 022840745 5 years
3                 25                  17 Francess 023360129 1 year
4                 36                  25 Eslam 024050987 4 years
```

```
Column names:
['SchoolID', 'SchoolName', 'Governorate', 'Type', 'NumberofStudents',
 'NumberofAssistants', 'NumberofInstructors', 'Principal', 'Phone',
 'Contract', 'Foundation_Date']

Data types:
SchoolID              object
SchoolName            object
Governorate           object
Type                  object
NumberofStudents      object
NumberofAssistants    object
NumberofInstructors    int64
Principal             object
Phone                  int64
Contract              object
Foundation_Date       object
dtype: object
```

```
Null values:
SchoolID              0
SchoolName            0
Governorate           0
Type                  0
NumberofStudents      0
NumberofAssistants    0
NumberofInstructors   0
Principal             0
Phone                 0
Contract              0
Foundation_Date       0
dtype: int64
```

Handling Data Type Conversion

We identify numeric columns that are stored as object type and convert them to proper numeric format.

```python
# Define numeric columns that need conversion
numeric_columns = ["StudentCount", "AssistantCount"]

for col in numeric_columns:
df[col] = pd.to_numeric(df[col], errors="coerce")    # convert to numeric
    df[col] = df[col].fillna(0).astype(int)          # fill NaN with 0 and cast to int

print("Data types after conversion:")
print(df.dtypes)
```

Output:

```
Data types after conversion:
SchoolID              object
SchoolName            object
Governorate           object
Type                  object
NumberofStudents       int64
NumberofAssistants     int64
NumberofInstructors    int64
Principal             object
dtype: object
```

- Used pd.to_numeric() with errors="coerce" to convert invalid values to NaN
- Filled any NaN values with 0
- Converted to integer type for proper calculations

Phone Number Formatting

We standardize phone numbers by ensuring they all start with "0".

```python
def format_phone(phone):
    phone = str(phone).strip()
    if not phone.startswith("0"):
        return "0" + phone
    return phone

df["Phone"] = df["Phone"].apply(format_phone)

print("Sample phone numbers after formatting:")
print(df[["SchoolID", "Phone"]].head(10))
```

```
Sample phone numbers after formatting:
  SchoolID       Phone
0       SM 022540812
1       ZH 021750933
2       SG 022840745
3       SJ 023360129
4       RC 024050987
5       SF 033120745
6       SH 044430982
7       SK 022350712
8       NL 022460518
9       ML 022540819
```

Formatting Rules:

- Convert phone number to string and remove whitespace
- If phone number doesn't start with "0", add "0" prefix
- Ensures consistent phone number format across all records

Date Format Conversion

Convert foundation dates from string format to proper datetime format.

```python
# Convert Foundation_Date to datetime format
df["Foundation_Date"] = pd.to_datetime(df["Foundation_Date"], format="%m/%d/%Y", errors="coerce")

# Format as YYYY-MM-DD for consistency
df["Foundation_Date"] = df["Foundation_Date"].dt.strftime("%Y-%m-%d")

print("Foundation dates after formatting:")
print(df[["SchoolID", "Foundation_Date"]].head(10))
```

Output:

```
Foundation dates after formatting:
SchoolID Foundation_Date
0      SM     2005-02-01
1      ZH     2018-07-01
2      SG     2020-09-01
3      SJ     2000-01-01
4      RC     2001-04-01
5      SF     2007-03-01
6      SH     2013-06-01
7      SK     2016-08-01
8      NL     2010-05-01
9      ML     2014-02-01
```

- Parse dates using pd.to_datetime() with format %m/%d/%Y
- Handle invalid dates by converting to NaT (Not a Time) with errors=" coerce"
- Reformat to standard YYYY-MM-DD format for consistency

Employees Sheet

Previewed the first few rows (head ()) to quickly understand the data.

	School ID	EmployeeID	Employee Name	...	Hire Date	Date Of Birth	Employee Type
0	SM	SM001	Salah Ishaq	...	06-2020	22-11-1980	Full Time
1	SM	SM002	Nadia Iyas	...	08-2024	12-11-1991	Full Time
2	SM	SM003	Mahmoud Jaleel	...	03-2020	05-12-1996	Full Time
3	SM	SM004	Nadia Jameel	...	09-2021	22-08-1990	Full Time
4	SM	SM005	Tarek Jamal	...	01-2020	09-02-1979	Full Time

Renaming Columns:

After loading the dataset, we standardized column names for consistency. For example, we renamed the column “Phone Number” to “Email” while keeping the existing data unchanged. The actual email values will be updated later during the cleaning process.

```
df.rename(columns={
    'School ID':   'SchoolID',
    'Phone Number': 'Email',
    'Hire Date': 'Hire_Date',
    'Date Of Birth': 'Date_Of_Birth',
    'Employee Type': 'Employment_Type'
}, inplace=True )
print(df.head())
```

	SchoolID	EmployeeID	Employee Name	...	Hire_Date	Date_Of_Birth	Employment_Type
0	SM	SM001	Salah Ishaq	...	06-2020	22-11-1980	Full Time
1	SM	SM002	Nadia Iyas	...	08-2024	12-11-1991	Full Time
2	SM	SM003	Mahmoud Jaleel	...	03-2020	05-12-1996	Full Time
3	SM	SM004	Nadia Jameel	...	09-2021	22-08-1990	Full Time
4	SM	SM005	Tarek Jamal	...	01-2020	09-02-1979	Full Time

Splitting Full Name into First and Last Name:

We separated the Employee Name column into two new columns: FirstName and LastName. After splitting, the original Employee Name column was removed to avoid redundancy.

```
df[['FirstName', 'LastName']] = df[
expand=True)

df.drop(columns=['Employee Name'
print(df.head())

'Employee Name'].str.split(pat=' ', n=1,
```

	SchoolID	EmployeeID	Gender	...	Employment_Type	FirstName	LastName
0	SM	SM001	M	...	Full Time	Salah	Ishaq
1	SM	SM002	F	...	Full Time	Nadia	Iyas
2	SM	SM003	M	...	Full Time	Mahmoud	Jaleel
3	SM	SM004	F	...	Full Time	Nadia	Jameel
4	SM	SM005	M	...	Full Time	Tarek	Jamal

Checking and Adjusting Data Types:

We examined the data types of all columns to ensure they were appropriate for analysis. This step helps identify columns that may need conversion, such as dates or numeric values, to enable accurate calculations and processing.

```
SchoolID            object
EmployeeID          object
FirstName           object
LastName            object
Gender              object
Email               object
Role                object
Salary             float64
Hire_Date           object
Date_Of_Birth       object
Employment_Type     object
dtype: object
```

The Hire_Date and Date_Of_Birth columns need to be converted to datetime objects:

```
df['Hire_Date'] = pd.to_datetime(df['Hire_Date'], errors='coerce',
dayfirst=True)

df['Date_Of_Birth'] = pd.to_datetime(df['Date_Of_Birth'], errors='coerce',
dayfirst=True)


print(df[['Hire_Date', 'Date_Of_Birth']].head(5))


print(df.dtypes)
```

```
SchoolID                  object
EmployeeID                object
FirstName                 object
LastName                  object
Gender                    object
Email                     object
Role                      object
Salary                   float64
Hire_Date          datetime64[ns]
Date_Of_Birth      datetime64[ns]
Employment_Type           object
```

Proper Case Formatting We ensure that specific text columns follow proper capitalization

```
for col in cols_to_format:
    df[col] = df[col].str.strip().str.title()

print(df[['FirstName', 'LastName', 'Role', 'Employment_Type']].head())
```

```
  FirstName LastName        Role Employment_Type
0     Salah    Ishaq  Instructor        Full Time
1     Nadia     Iyas  Instructor        Full Time
2   Mahmoud   Jaleel  Instructor        Full Time
3     Nadia   Jameel  Instructor        Full Time
4     Tarek    Jamal  Instructor        Full Time
```

Handling Null Values

```
print(df.isnull().sum())
```

```
SchoolID          0
EmployeeID        0
FirstName         1
LastName          2
Gender            1
Email             0
Role              2
Salary            5
Hire_Date         3
Date_Of_Birth     1
Employment_Type   0
```

For text columns, nulls are replaced with "Not Specified". For numeric columns (e.g., Salary), we replace nulls with the mean value.

```
cols_fill_text = ['FirstName', 'LastName', 'Gender', 'Role', 'Hire_Date',
'Date_Of_Birth']
df[cols_fill_text] = df[cols_fill_text].fillna('Not Specified')
df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
print(df.isnull().sum())
```

```
SchoolID          0
EmployeeID        0
FirstName         0
LastName          0
Gender            0
Email             0
Role              0
Salary            0
Hire_Date         0
Date_Of_Birth     0
Employment_Type   0
```

Check for Duplicates

```
duplicates = df[df.duplicated()]
print("No. of Duplicates", len(duplicates))
```

```
No. of Duplicates 0
```

Detecting Outliers in Salary

We use the IQR (Interquartile Range) method to identify potential outliers in the Salary column

```
Q1 = df['Salary'].quantile(0.25)
Q3 = df['Salary'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df[(df['Salary'] < lower_bound) | (df['Salary'] > upper_bound)]
print("No. of Outliers", len(outliers))
```

```
No. of Outliers 0
```

Final Check

We display a sample of the cleaned data and check the final data types.

```
print(df.head())
print(df.dtypes)
```

```
  SchoolID  EmployeeID FirstName  ...  Hire_Date Date_Of_Birth Employment_Type
0       BM      BM001     Salah  ... 2020-06-01    1980-11-22       Full Time
1       BM      BM002     Nadia  ... 2024-08-01    1991-11-12       Full Time
2       BM      BM003   Mahmoud  ... 2020-03-01    1996-12-05       Full Time
3       BM      BM004     Nadia  ... 2021-09-01    1990-08-22       Full Time
4       BM      BM005     Tarek  ... 2020-01-01    1979-02-09       Full Time

[5 rows x 11 columns]
SchoolID                  object
EmployeeID                object
FirstName                 object
LastName                  object
Gender                    object
Email                     object
Role                      object
Salary                     int64
Hire_Date          datetime64[ns]
Date_Of_Birth      datetime64[ns]
Employment_Type           object
dtype: object
```

```
        count    mean     std     min     25%     50%      75%      max  \
Salary 692.00 9534.45 3434.65 6004.00 6672.00 7289.00 13272.50 14997.00

        median
Salary 7289.00

Date Columns Statistics:

Column: Hire_Date
 - Count: 692
 - Min Date: 2020-01-01 00:00:00
 - Max Date: 2025-09-01 00:00:00

Column: Date_Of_Birth
 - Count: 692
 - Min Date: 1976-02-23 00:00:00
 - Max Date: 2004-12-20 00:00:00

 Text Columns Statistics:

Column: SchoolID
 - Count: 692
 - Unique values: 15
 - Most frequent value: NS
```

```
Column: EmployeeID
 - Count: 692
 - Unique values: 692
 - Most frequent value: BM001

Column: FirstName
 - Count: 692
 - Unique values: 64
 - Most frequent value: Ragab

Column: LastName
 - Count: 692
 - Unique values: 261
 - Most frequent value: Tariq

Column: Gender
 - Count: 692
 - Unique values: 2
 - Most frequent value: M

Column: Email
 - Count: 692
 - Unique values: 678
 - Most frequent value: AmerIsmail@gmail.com
```

```
Column: Role
 - Count: 692
 - Unique values: 2
 - Most frequent value: Teaching Assistant

Column: Employment_Type
 - Count: 692
 - Unique values: 2
 - Most frequent value: Full Time

 Number of Duplicated Rows: 0

 Null Values per Column:
SchoolID           0
EmployeeID         0
FirstName          0
LastName           0
Gender             0
Email              0
Role               0
Salary             0
Hire_Date          0
Date_Of_Birth      0
Employment_Type    0
dtype: int64
```

Students Sheet

```
import pandas as pd
df = pd.read_excel(r"C:\Users\OMEN\Desktop\before.xlsx",
sheet_name='Students')
pd.set_option('display.max_columns', None)

print(df.head())
```

```
  Student ID School ID                Full Name  Gender  Address  \
0      S1000        SM       Alia Mansour Fekry     NaN      NaN
1      S1001        SM              Remas Wahba     NaN      NaN
2      S1002        SM  Abdelmoneim Alaa Khalil     NaN      NaN
3      S1003        SM     Miriam Maged Zakaria     NaN      NaN
4      S1004        SM   Julie Mohamed Mokhless     NaN      NaN

                         Email         Phone Enrollment ID          Track  \
0          alia.fekry@gmail.com  2.010707e+11           SC1  1st Secondary
1         remas.wahba@gmail.com  9.747056e+10           SC1  1st Secondary
2  abdelmoneim.khalil@gmail.com           NaN           SC1  1st Secondary
3      miriam.zakaria@gmail.com  9.665501e+11           SC1  1st Secondary
4      julie.mokhless@gmail.com  2.012797e+11           SC1  1st Secondary

   Status
0  Active
1  Active
2  Active
3  Active
4  Active
```

Renaming & Modifying Columns:

After loading the dataset, we standardized column names for clarity and consistency. We also modified the dataset structure by adding necessary columns for analysis and removing irrelevant ones.

```
df.rename(columns={
 'Student ID': 'StudentID',
 'School ID': 'SchoolID',
 'Full Name': 'FullName',
 'Address': 'City',
 'Email': 'Email',
 'Phone': 'Phone',
 'Enrollment ID': 'EnrollmentCode',
 'Track': 'Level',
 'Status': 'Status'
}, inplace=True)
print(df.head)
```

```
  StudentID SchoolID                FullName  City  \
0     S1000       SM       Alia Mansour Fekry   NaN
1     S1001       SM             Remas Wahba   NaN
2     S1002       SM  Abdelmoneim Alaa Khalil   NaN
3     S1003       SM     Miriam Maged Zakaria   NaN
4     S1004       SM   Julie Mohamed Mokhless   NaN

                         Email         Phone EnrollmentCode          Level  \
0          alia.fekry@gmail.com  2.010707e+11            SC1  1st Secondary
1         remas.wahba@gmail.com  9.747056e+10            SC1  1st Secondary
2  abdelmoneim.khalil@gmail.com           NaN            SC1  1st Secondary
3      miriam.zakaria@gmail.com  9.665501e+11            SC1  1st Secondary
4      julie.mokhless@gmail.com  2.012797e+11            SC1  1st Secondary

   Status Governorate  Type SpecialNeeds Grade   Age
0  Active        None  None         None  None  None
1  Active        None  None         None  None  None
2  Active        None  None         None  None  None
3  Active        None  None         None  None  None
4  Active        None  None         None  None  None
```

Rearranging Columns For better structure

We rearrange the columns in a more logical order

Checking and Adjusting Data Types

We inspect the data types of each column.

```
print(df.dtypes)
```

```
SchoolID          object
StudentID         object
FullName          object
City             float64
Governorate       object
Age               object
Email             object
Phone            float64
EnrollmentCode    object
Level             object
Grade             object
Status            object
Type              object
SpecialNeeds      object
dtype: object
```

We have to change the data types of certain columns: converting the Age column to integer and the City and Phone columns to string to ensure proper formatting and consistency across the dataset.

```
df['Phone'] = df['Phone'].astype('Int64').astype(str)
df['City'] = df['City'].astype(str)
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
print(df[['Phone', 'City','Age']].dtypes)
```

```
Phone      object
City       object
Age       float64
dtype: object
```

Proper Case Formatting:

We applied proper capitalization to specific text columns so that each word starts with a capital letter, ensuring consistent and professional formatting throughout the dataset.

```
cols_to_proper = ['FullName', 'City', 'Governorate', 'Grade', 'Status',
'Type', 'SpecialNeeds']
for col in cols_to_proper:
    df[col] = df[col].astype(str).str.title()
```

Handling Null Values:

We filled columns with missing values, such as Age, Grade, and Governorate, by deriving logical values from existing data using predefined rules. Columns that could not be inferred logically were completed by referencing additional sources or related datasets to ensure data completeness.

```
import numpy as np

df['Grade'] = np.where(df['Level'] == '1st Secondary', 'Grade 10',
               np.where(df['Level'] == '2nd Secondary', 'Grade 11',
               np.where(df['EnrollmentCode'].isin(['LT', 'SB', 'SM']), 'Grade
12', 'Not Specified')))

df['Age'] = np.where(df['Grade'] == 'Grade 10', 16,
             np.where(df['Grade'] == 'Grade 11', 17, 18))
```

	Age	Grade
0	16	Grade 10
1	16	Grade 10
2	16	Grade 10
3	16	Grade 10
4	16	Grade 10

Mapping Governorate:

We populated the Governorate column by matching SchoolID with the Schools sheet. This ensures that each student is assigned the correct governorate based on their school.

```
import pandas as pd

students_df = pd.read_excel(r"C:\Users\OMEN\Desktop\before.xlsx",
sheet_name='Students')


schools_df = pd.read_excel(r"C:\Users\OMEN\Desktop\before.xlsx",
sheet_name='Schools')

schools_df.columns = schools_df.columns.str.strip()
students_df.columns = students_df.columns.str.strip()

gov_map = dict(zip(schools_df['SchoolID'], schools_df['Governorate']))

students_df['Governorate'] = students_df['SchoolID'].map(gov_map)


print(students_df[['SchoolID', 'Governorate']].head(2000))
```

```
     SchoolID Governorate
0          SM  Alexandria
1          SM  Alexandria
2          SM  Alexandria
3          SM  Alexandria
4          SM  Alexandria
...       ...         ...
1995       SF       Cairo
1996       SF       Cairo
1997       SF       Cairo
1998       SF       Cairo
1999       SF       Cairo

[2000 rows x 2 columns]
```

Checking for Duplicates:

We scanned the dataset to identify and remove any duplicate rows, ensuring that each record is unique and preventing redundancy in the analysis.

```
duplicates = df[df.duplicated()]
print("No. of Duplicates", len(duplicates))
```

```
No. of Duplicates 0
```

Final Check

We display a sample of the cleaned data and check the final data types.

```
print(df.head())
print(df.dtypes)
```

```
  SchoolID StudentID                FullName City  Governorate  Age  \
0       SM     S1000       Alia Mansour Fekry  Nan   Alexandria   16
1       SM     S1001             Remas Wahba  Nan   Alexandria   16
2       SM     S1002  Abdelmoneim Alaa Khalil  Nan   Alexandria   16
3       SM     S1003     Miriam Maged Zakaria  Nan   Alexandria   16
4       SM     S1004   Julie Mohamed Mokhless  Nan   Alexandria   16

                         Email         Phone EnrollmentCode          Level  \
0          alia.fekry@gmail.com  201070679109            SC1  1st Secondary
1         remas.wahba@gmail.com  201277662350            SC1  1st Secondary
2  abdelmoneim.khalil@gmail.com  201211667717            SC1  1st Secondary
3      miriam.zakaria@gmail.com  201002771278            SC1  1st Secondary
4      julie.mokhless@gmail.com  201279713142            SC1  1st Secondary

      Grade  Status Type SpecialNeeds
0  Grade 10  Active  nan          nan
1  Grade 10  Active  nan          nan
2  Grade 10  Active  nan          nan
3  Grade 10  Active  nan          nan
4  Grade 10  Active  nan          nan
```

SchoolID	object
StudentID	object
FullName	object
City	object
Governorate	object
Age	int64
Email	object
Phone	object
EnrollmentCode	object
Level	object
Grade	object
Status	object
Type	object
SpecialNeeds	object

dtype: object

```
Column: EnrollmentCode
 - Count (non-null): 6250
 - Unique values: 5
 - Most frequent value: SC2

Column: Level
 - Count (non-null): 6250
 - Unique values: 5
 - Most frequent value: 2nd Secondary

Column: Grade
 - Count (non-null): 6250
 - Unique values: 3
 - Most frequent value: Grade 12

Column: Status
 - Count (non-null): 6250
 - Unique values: 2
 - Most frequent value: Active

Number of Duplicated Rows: 0
```

```
Text Columns Statistics:

Column: SchoolID
 - Count (non-null): 6250
 - Unique values: 15
 - Most frequent value: NS

Column: StudentID
 - Count (non-null): 6250
 - Unique values: 6250
 - Most frequent value: S1000

Column: FullName
 - Count (non-null): 6250
 - Unique values: 6020
 - Most frequent value: Mohamed Ahmed

Column: Governorate
 - Count (non-null): 6250
 - Unique values: 4
 - Most frequent value:  Cairo

Column: Email
 - Count (non-null): 6250
 - Unique values: 6249
 - Most frequent value: omar.ahmed@gmail.com
```

```
Null Values per Column:
SchoolID             0
StudentID            0
FullName             0
City              6250
Governorate          0
Age                  0
Email                0
Phone                0
EnrollmentCode       0
Level                0
Grade                0
Status               0
Type              6250
SpecialNeeds      6250
```

Enrollment Sheet

```
print(df.shape)
print("\nFirst few rows:")
print(df.head(10))
print("\nColumn names:")
print(df.columns.tolist())
print("\nData types:")
print(df.dtypes)
print("\nInitial null counts:")
print(df.isnull().sum())
```

```
First few rows:
EnrollmentCode Subject          Level     Grade
0          SC1     ASC1 1st Secondary Grade 10
1          SC1       E    1st Secondary Grade 10
2          SC1      M1     1st Secondary Grade 10
3          SC1      IS    1st Secondary Grade 10
4          SC2     His     1st Secondary Grade 10
5          SC2      Ar    2nd Secondary Grade 11
6          SC2       E   2nd Secondary Grade 11
7          SC2      M2    2nd Secondary Grade 11
8          SC2    Chem     Phy    2nd Secondary Grade 11
9                                2nd Secondary Grade 11

Column names:
["EnrollmentCode", "Subject", "Level", "Grade"]

Data types:
EnrollmentCode object
Subject         object
Level           object
Grade           object
dtype: object

Initial null counts:
EnrollmentCode 0
Subject         0
Level           0
Grade           0
dtype: int64
```

After cleaning

	EnrollmentCode	Subject	Level	Grade
0	1 SC1	SCAr	1st Secondary	Grade 10
2	SC1	E	1st Secondary	Grade 10
3	SC1	M1	1st Secondary	Grade 10
4	SC1	IS	1st Secondary	Grade 10
5	SC2	His	1st Secondary	Grade 10
6	SC2	Ar	2nd Secondary	Grade 11
7	SC2	E	2nd Secondary	Grade 11
8	SC2	M2	2nd Secondary	Grade 11
9	SC2	Chem Phy	2nd Secondary	Grade 11
10			2nd Secondary	Grade 11
11	SB	Ar	Science-Biology Track	Grade 12
12	SB	E	Science-Biology Track	Grade 12
13	SB	Bio	Science-Biology Track	Grade 12
14	SB	Phy	Science-Biology Track	Grade 12
15	SB	Chem	Science-Biology Track	Grade 12
16	SM	Ar	Science-Math Track	Grade 12
17	SM	E	Science-Math Track	Grade 12
18	SM	M3	Science-Math Track	Grade 12
19	SM	Phy	Science-Math Track	Grade 12
20	SM	Chem	Science-Math Track	Grade 12
21				
22	LT	Ar	Literary track	Grade 12
23	LT	E	Literary track	Grade 12
	LT	His	Literary track	Grade 12
	LT	Geo	Literary track	Grade 12

Final shape: (24, 4)

Final shape: (24, 4)

Data types:

EnrollmentCode object

Subject object

Level object

Grade object

dtype: object

Student performance Sheet

Read the main dataset from an Excel file named Studentperformance.xlsx.

Remove Empty Rows and Unnecessary Columns

- dropna(how='all') removes any rows that are completely empty.
- The second line removes any unnamed columns (usually artifacts from Excel exports). This ensures the dataset only contains meaningful data.
- Standardize Column Names

```
df.columns = df.columns.str.strip().str.replace(' ', '_').str.title()
```

- Cleans up column headers by: Removing extra spaces, replacing spaces with underscores.
- Capitalizing first letters for consistency. Example: "final score" → "Final_Score".

Ensure Numeric Fields Are Properly Formatted

```
numeric_cols = ['Midterm_Score', 'Final_Score', 'Assignments_Avg', 'Quizzes_Avg', 'Projects_Score']
for col in numeric_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
```

- Defines a list of columns expected to contain numbers.
- Converts each to a numeric type using pd.to_numeric().
- Invalid or non-numeric values are turned into NaN (handled gracefully).
- Prevents formula or string errors during calculations

Check and Handle Missing (Null) Values

```
null_summary = df.isnull().sum()
print(" Missing values per column:")
print(null_summary)

df.fillna({
    'Student_Name': 'Unknown',
    'Midterm_Score': 0,
    'Final_Score': 0,
    'Assignments_Avg': 0,
    'Quizzes_Avg': 0,
    'Projects_Score': 0
}, inplace=True)
```

```
Missing values per column:
Schoolid                    0
Studentid                   0
Enrollmentcode              0
Attendancerate              0
Midterm_Score               0
Final_Score                 0
Assignments_Avg             0
Quizzes_Avg                 0
Participation_Score         0
Projects_Score              0
Total_Score                 0
Grade                       0
Study_Hours_Per_Week        0
Internet_Access_At_Home     0
Parent_Education_Level     61
Stress_Level                0
Sleep_Hours_Per_Night       0
dtype: int64
 Cleaning complete! File saved as 'Cleaned_Student_Data.xlsx'
```

- Displays how many missing values exist in each column.
- Replaces missing names with "Unknown" and missing numeric values with 0 to maintain data integrity.

This is similar to how IFERROR in Excel replaces blank cells or invalid lookups

Check for duplicates

```
duplicates = df[df.duplicated()]
print(f" Found {duplicates.shape[0]} duplicate rows.")

if not duplicates.empty:
    df.drop_duplicates(inplace=True)
    print("Duplicates removed successfully.")
```

- Identifies duplicate rows using duplicated().
- Prints how many duplicates exist. Removes all duplicates with drop_duplicates() to ensure each record is unique. Keeps the first occurrence by default.

Output

```
Found 0 duplicate rows.
Cleaning complete! File saved as 'Cleaned_Student_Data.xlsx'
```

Calculate Averages (Assignments & Quizzes)

```
df['Total_Score'] = (
    df.get('Final_Score', 0)*0.4 +
    df.get('Midterm_Score', 0)*0.2 +
    df.get('Assignments_Avg', 0)*0.1 +
    df.get('Quizzes_Avg', 0)*0.1 +
    df.get('Projects_Score', 0)*0.1
)
```

Dynamically detects all columns containing “Assign” or “Quiz” in their names.
Calculates row-wise averages for each student using mean(axis=1).
Adds new columns Assignments_Avg and Quizzes_Avg to summarize performance across multiple components.

Assign Letter Grades

```
def grade(score):
    if pd.isna(score):
        return ""
    elif score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

df['Grade'] = df['Total_Score'].apply(grade)
```

Defines a function that classifies each student based on their total score.
Applies the same logic as Excel’s nested IF() formula:
A = 90+, B = 80+, C = 70+, D = 60+, F = below 60.
The function is applied to the Total_Score column to create a new Grade column.

Staff performance Sheet

```
import pandas as pd
import numpy as np

df = pd.read_csv('Before.csv')
print("Shape:", df.shape)
print("\nFirst few rows:")
print(df.head())
print("\nColumn names:")
print(df.columns.tolist())
print("\nData types:")
print(df.dtypes)
print("\nNull values:")
print(df.isnull().sum())
```

The raw data contains 706 rows and 11 columns with some null values that need to be addressed.

Renaming Columns

Once the data is displayed, we start by renaming some of the columns to follow a consistent naming convention

```
df.rename(columns={
    'School ID': 'SchoolID',
    'Name': 'FirstName',
    'Employee_ID': 'EmployeeID'
}, inplace=True)

print("Columns after renaming:")
print(df.columns.tolist())
```

The column names are now standardized to match the final output format.

Handling Null Values

We check for null values in the dataset and apply appropriate strategies:

```
# For strings NULLSS
df['SchoolID'] = df['SchoolID'].fillna('Not Specified')
df['FirstName'] = df['FirstName'].fillna('Not Specified')

# For numeric NULLSSS
df['Attendance_Days'] = df['Attendance_Days'].fillna(df['Attendance_Days'].mean()).round(0).astype(int)
df['Overtime_Hours'] = df['Overtime_Hours'].fillna(df['Overtime_Hours'].mean()).round(0).astype(int)

print("\nNull values AFTER cleaning:")
print(df.isnull().sum())
```

Handling Missing Values by Type:

- For text columns, missing entries were replaced with "Not Specified".
- For integer columns, missing values were filled with the mean of the column and rounded to the nearest integer.

Checking and Adjusting Data Types

We inspect the data types of each column to ensure they are appropriate for calculations.

```
print("Current data types:")
print(df.dtypes)
```

Creating Calculated Columns

We create new calculated metrics including scaled behavior scores, completion rates, and overall performance scores.

```
df['Behavior_Score'] = df['Behavior_Score'] * 20
df['Task_Completion_Rate'] = ((df['Tasks_Completed'] / df['Tasks_Assigned']) * 100).round(1)
df['Attendance_Rate'] = ((df['Attendance_Days'] / 22) * 100).round(1)

# Formula: 0.5*Task_Completion_Rate + 0.3*Attendance_Rate + 0.1*Behavior_Score + 0.1*MIN(((Avg_Work_Hours + Ov
work_hours_component = np.minimum(((df['Avg_Work_Hours'] + df['Overtime_Hours']) / 8) * 100, 100)

df['Performance_Score'] = (
    0.5 * df['Task_Completion_Rate'] +
    0.3 * df['Attendance_Rate'] +
    0.1 * df['Behavior_Score'] +
    0.1 * work_hours_component
).round(2)

print("First 10 rows with calculated columns:")
print(df[['EmployeeID', 'Task_Completion_Rate', 'Attendance_Rate', 'Behavior_Score', 'Performance_Score']].head(10))
```

Calculations:

Behavior_Score: Multiply by 20 to convert from 5-point scale to 100-point scale Task_Completion_Rate: (Tasks_Completed ÷ Tasks_Assigned) × 100, rounded to 1 decimal place Attendance_Rate: (Attendance_Days ÷ 22) × 100, rounded to 1 decimal place Performance_Score: Weighted formula with 4 components, rounded to 2 decimal places 50% weight: Task Completion Rate 30% weight: Attendance Rate 10% weight: Behavior Score 10% weight: Work Hours Efficiency (capped at 100) The work hours component calculates: MIN(((Avg_Work_Hours + Overtime_Hours) ÷ 8) × 100, 100) This represents efficiency based on an 8-hour workday standard, with a maximum value of 100.

Creating Rank and Promotion Eligibility

Assign letter grades based on Performance_Score thresholds and determine promotion eligibility

```
def assign_rank(score):
 if score > 95:
 return 'A'
 elif score > 80:
 return 'B'
 elif score > 70:
 return 'C'
 elif score > 60:
 return 'D'
 else:
 return 'F'
df['Rank'] = df['Performance_Score'].apply(assign_rank)
df['Promotion_Eligibility'] = df['Rank'].apply(lambda x: 'Yes' if x == 'A'
else 'No')
```

Rearranging Columns

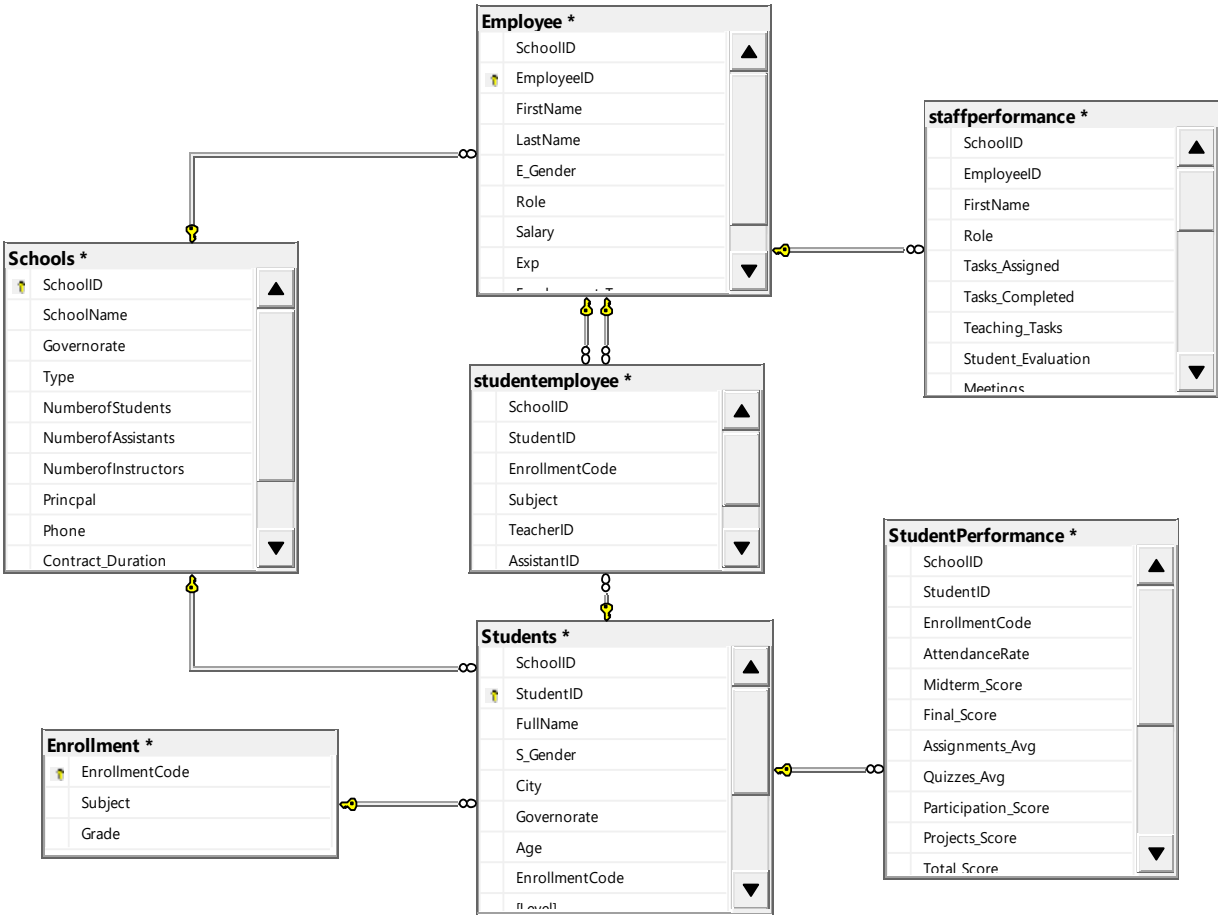
For better structure, we rearrange the columns in a logical order matching the desired output.

```
final_columns = [

'SchoolID','EmployeeID','FirstName','Role','Tasks_Assigned','Tasks_Completed'
, 'Task_Completion_Rate', 'Attendance_Days', 'Absent_Days','Attendance_Rate',
 'Avg_Work_Hours', 'Overtime_Hours', 'Behavior_Score', 'Performance_Score',
'Rank', 'Promotion_Eligibility']
df = df[final_columns]
print("Column order:")
print(df.columns.tolist())
print("\nFirst 10 rows of cleaned data:")
print(df.head(10))
print("\nDataFrame shape:", df.shape)
print("\nData types:")
print(df.dtypes)
```

Data Modeling and Relationships

This section illustrates the database structure and the relationships between tables in our system. A visual representation of the data model from SQL Server is provided, showing how schools, students, employees, enrollments, and performance data are interconnected to maintain data integrity and support analysis.



- Schools ↔ Students: Each school can have multiple students, and every student belongs to a specific school. This connection is established via the SchoolID column.
- Schools ↔ Staff: Each school employs multiple staff members, including teachers and assistants. The link is through the SchoolID column, ensuring that each staff member is associated with the correct school.
- Students ↔ StudentPerformance: Each student has performance records detailing their grades, attendance, and other academic metrics. This relationship is maintained using the StudentID column.
- Staff ↔ StaffPerformance: Every staff member has a corresponding performance record that tracks tasks completed, attendance, behavior, and overall evaluation. The connection is based on EmployeeID.
- Students ↔ Enrollment: Students are enrolled in different subjects or tracks. This association is made through the EnrollmentCode, linking each student to their respective courses.
- Students ↔ StudentEmployee: Each student is linked to their assigned teacher and assistant(s) for each subject. The relationship uses StudentID to ensure correct assignment.
- Teachers ↔ StudentEmployee: Teachers and their assistants can be associated with multiple students across different subjects. The link uses TeacherID and AssistantID to map staff members to their assigned students.

This structure ensures proper tracking of academic and staff assignments, as well as performance across all schools.

Data Analysis:

After completing data wrangling and cleaning, we proceed to data analysis. This stage focuses on exploring the dataset to answer key business questions and derive actionable insights.

The analysis examines three key areas:

- Schools: Understanding each school's capacity, resources, and distribution across governorates to identify operational strengths and potential gaps.
- Students: Analyzing enrollment, performance, and demographics to assess learning outcomes and track trends across different grades and levels.
- Staff: Evaluating teachers and assistants, their workload, performance scores, and assignments to ensure effective teaching support and identify areas for improvement

Notes:

- The dataset includes information from 15 schools across 4 governorates.
- This report is intended for submission to the Ministry of Education.

Analytical Questions Answered by SQL:

This section presents the key queries used to extract insights from the database.

- **How many schools are in each governate?**

```sql
SELECT
    Governorate,
    COUNT(SchoolID) AS Number_of_Schools
FROM Schools
GROUP BY Governorate
       ORDER BY Number_of_Schools DESC;
```

	Governorate	Number_of_Schools
1	Alexandria	5
2	Cairo	5
3	Giza	3
4	Qaliobia	2

- **How are students distributed across governates?**

```sql
SELECT
    Governorate,
    COUNT(*) AS Number_of_Students,
    ROUND((COUNT(*) * 100.0 / (SELECT COUNT(*) FROM Students)), 2) AS Percentage
FROM Students
GROUP BY Governorate
       ORDER BY Number_of_Students DESC;
```

	Governorate	Number_of_Students	Percentage
1	Cairo	2020	32.320000000000
2	Alexandria	1990	31.840000000000
3	Giza	1280	20.480000000000
4	Qaliobia	960	15.360000000000

- **What's the performance score per student for each governate?**

```sql
SELECT
    s.Governorate,
    AVG(sp.Total_Score) AS Avg_Student_Performance
FROM Students s
JOIN StudentPerformance sp
    ON s.StudentID = sp.StudentID
GROUP BY s.Governorate
ORDER BY Avg_Student_Performance DESC;
```

	Governorate	Avg_Student_Performance
1	Cairo	80.5998803887204
2	Alexandria	75.268496745549
3	Giza	74.9313783116225
4	Qaliobia	66.4575363119443

- **What's the performance score per staff member for each governate?**

```
SELECT
    sc.Governorate,
    AVG(sp.Performance_Score) AS Avg_Staff_Performance
FROM Employee e
JOIN StaffPerformance sp
    ON e.EmployeeID = sp.EmployeeID
JOIN Schools sc
    ON e.SchoolID = sc.SchoolID
GROUP BY sc.Governorate
    ORDER BY Avg_Staff_Performance DESC;
```

	Governorate	Avg_Staff_Performance
1	Cairo	81.4122936003799
2	Giza	81.1977623626069
3	Alexandria	78.8875228811842
4	Qaliobia	74.9320357331133

- What are the top 3 schools ranked by student performance percentage?

```
SELECT TOP 3
    sc.SchoolName,
    ROUND(AVG(sp.Total_Score), 2) AS Avg_Student_Performance
FROM Students st
JOIN StudentPerformance sp
    ON st.StudentID = sp.StudentID
JOIN Schools sc
    ON st.SchoolID = sc.SchoolID
GROUP BY sc.SchoolName

    ORDER BY Avg_Student_Performance DESC;
```

	SchoolName	Avg_Student_Performance
1	Sakara	85.96
2	El Nile	85.13
3	Al Manhal	84.54

- **What are the top 3 schools by average performance score?**

```
SELECT TOP 3
    s.SchoolName,
    CAST(AVG(sp.Performance_Score) AS DECIMAL(5,2)) AS Avg_Staff_Performance
FROM Schools s
JOIN StaffPerformance sp
    ON s.SchoolID = sp.SchoolID
GROUP BY s.SchoolName
       ORDER BY Avg_Staff_Performance DESC;
```

	SchoolName	Avg_Staff_Performance
1	Al Manhal	85.94
2	El Nile	85.10
3	El Tahrir	84.72

- **What's the male/female ratio among students?**

```
SELECT
    SUM(CASE WHEN S_Gender = 'M' THEN 1 ELSE 0 END) AS Male_Count,
    SUM(CASE WHEN S_Gender = 'F' THEN 1 ELSE 0 END) AS Female_Count,
    CAST(SUM(CASE WHEN S_Gender = 'M' THEN 1 ELSE 0 END) AS FLOAT)
        / NULLIF(SUM(CASE WHEN S_Gender = 'F' THEN 1 ELSE 0 END), 0) AS
Male_to_Female_Ratio
FROM Students;
```

	Male_Count	Female_Count	Male_to_Female_Ratio
1	3712	2538	1.46256895193065

- **What's the male/female ratio among staff?**

```
SELECT
    SUM(CASE WHEN E_Gender = 'M' THEN 1 ELSE 0 END) AS Male_Count,
    SUM(CASE WHEN E_Gender = 'F' THEN 1 ELSE 0 END) AS Female_Count,
    CAST(SUM(CASE WHEN E_Gender = 'M' THEN 1 ELSE 0 END) AS FLOAT)
        / NULLIF(SUM(CASE WHEN E_Gender = 'F' THEN 1 ELSE 0 END), 0)
AS Male_to_Female_Ratio

   FROM Employee;
```

	Male_Count	Female_Count	Male_to_Female_Ratio
1	541	151	3.58278145695364

- **How many instructors vs teaching assistants are in each school?**

```
SELECT
    SchoolID,
    SUM(CASE WHEN Role = 'Instructor' THEN 1 ELSE 0 END) AS Num_Instructors,
    SUM(CASE WHEN Role = 'Teaching Assistant' THEN 1 ELSE 0 END) AS Num_TAs
FROM Employee
GROUP BY SchoolID;
```

	SchoolID	Num_Instructors	Num_TAs
1	BM	28	33
2	HR	16	20
3	IZ	14	17
4	ML	24	16
5	NL	16	21
6	NS	27	55
7	RC	25	36
8	SF	19	29
9	SG	12	24
10	SH	21	48
11	SJ	17	25
12	SK	12	12
13	SM	12	22
14	TR	20	26
15	ZH	15	30

- **What's the average staff experience per school?**

```
SELECT
    SchoolID,
    AVG(Exp) AS Avg_Experience
FROM Employee

    GROUP BY SchoolID;
```

	SchoolID	Avg_Experience
1	BM	10
2	HR	9
3	IZ	8
4	ML	11
5	NL	7
6	NS	8
7	RC	9
8	SF	8
9	SG	8
10	SH	8
11	SJ	8
12	SK	6
13	SM	7
14	TR	9
15	ZH	8

- **How are instructors and teaching assistants distributed across school types?How many male/female instructors and TAs are there?**

```
SELECT
    s.Type AS SchoolType,
    e.Role,
    COUNT(*) AS Count
FROM Employee e
JOIN Schools s
    ON e.SchoolID = s.SchoolID
WHERE e.Role IN ('Instructor', 'Teaching Assistant')
GROUP BY s.Type, e.Role;
```

	SchoolType	Role	Count
1	Private	Instructor	125
2	Public	Instructor	153
3	Private	Teaching Assistant	155
4	Public	Teaching Assistant	259

- **What's the average experience for instructors vs TAs, broken down by gender?**

```
SELECT
    Role,
    E_Gender,
    AVG(Exp) AS Average_Experience
FROM Employee
GROUP BY Role, E_Gender;
```

	Role	E_Gender	Average_Experience
1	Instructor	F	14
2	Teaching Assistant	F	4
3	Instructor	M	16
4	Teaching Assistant	M	4

- **What's the average salary for instructors vs TAs, broken down by gender?**

```
SELECT
    Role,
    E_Gender,
    AVG(Salary) AS Average_Salary
FROM Employee
GROUP BY Role, E_Gender;
```

	Role	E_Gender	Average_Salary
1	Instructor	F	18709
2	Teaching Assistant	F	8605
3	Instructor	M	18287
4	Teaching Assistant	M	9499

- **How does salary increase with experience levels?**

```
SELECT
    Exp AS Experience_Years,
    AVG(Salary) AS Average_Salary
FROM Employee
GROUP BY Exp
ORDER BY Exp;
```

	Experience_Years	Average_Salary
1	1	7990
2	2	8204
3	3	8880
4	4	10658
5	5	10378
6	6	11387
7	7	11375
8	8	12664
9	9	17782
10	10	14851
11	11	16518
12	12	18220
13	13	17687
14	14	16554
15	15	19137
16	16	19414
17	17	15776
18	18	20978
19	19	20975
20	20	21793
21	21	23440
22	22	18500
23	23	20176
24	24	21200
25	25	20381
26	26	20719
27	27	22612
28	28	21303
29	30	23300

- **What percentage of students have special needs?**

```
SELECT
    (COUNT(CASE WHEN SpecialNeeds = 'Yes' THEN 1 END) * 100.0) / COUNT(*) AS
SpecialNeeds_Percentage
FROM Students;
```

	SpecialNeeds_Percentage
1	0.704000000000

- **How are male/female students distributed across Grades?**

```
SELECT
    Class,
    S_Gender,
    COUNT(*) AS StudentCount
FROM Students
GROUP BY Class, S_Gender
ORDER BY Class, S_Gender;
```

	Class	S_Gender	StudentCount
1	Grade 10	F	618
2	Grade 10	M	919
3	Grade 11	F	636
4	Grade 11	M	1014
5	Grade 12	F	1284
6	Grade 12	M	1779

- **How many students are in each grade level ?**

```
SELECT
    class,
    COUNT(*) AS StudentCount
FROM Students
GROUP BY class
ORDER BY class;
```

	class	StudentCount
1	Grade 10	1537
2	Grade 11	1650
3	Grade 12	3063

- **How many students fall into each status category?**

```
SELECT
    Status,
    COUNT(*) AS StudentCount
FROM Students
GROUP BY Status
ORDER BY Status;
```

	Status	StudentCount
1	Active	5826
2	Pending	424

- **How does parent education affect student performance?**

```
SELECT
    Parent_Education_Level,
    AVG(Total_Score) AS AvgPerformance
FROM StudentPerformance
GROUP BY Parent_Education_Level
ORDER BY AvgPerformance DESC;
```

	Parent_Education_Level	AvgPerformance
1	PhD	80.5461153590145
2	Master's	77.7075375207786
3	Bachelor's	74.7059524285695
4	High School	69.9655627103269
5	Uneducated	61.1386981899455

- **Which schools have the top performing TAs?**

```
SELECT
    s.SchoolID,
    AVG(p.Performance_Score) AS AvgTA_Performance
FROM Employee AS s
JOIN StaffPerformance AS p
    ON s.EmployeeID = p.EmployeeID
WHERE s.Role = 'Teaching Assistant'
GROUP BY s.SchoolID
ORDER BY AvgTA_Performance DESC;
```

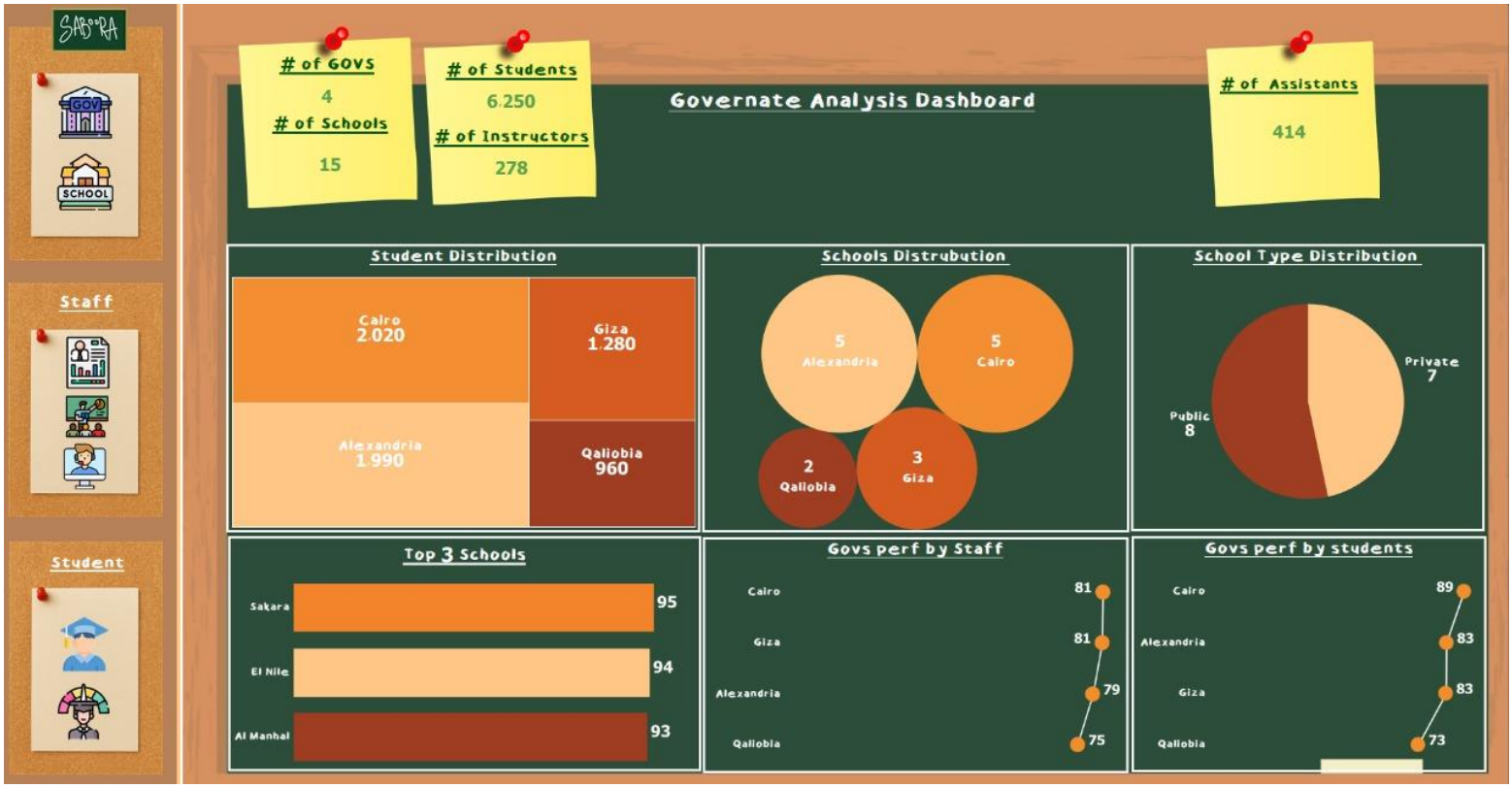
	SchoolID	AvgTA_Performance
1	SK	83.1624997456868
2	NL	82.7485714867001
3	ML	82.2206254005432
4	TR	80.8003848149226
5	SG	80.5175005594889
6	RC	80.2533331976996
7	HR	80.111499786377
8	SH	75.2681249777476
9	SF	73.704137999436
10	SM	73.642726724798
11	NS	71.0112730546431
12	BM	69.8596971685236
13	IZ	69.5723531386432
14	SJ	69.3828004455566
15	ZH	68.6730000813802

- **Which schools have the top performing instructors?**

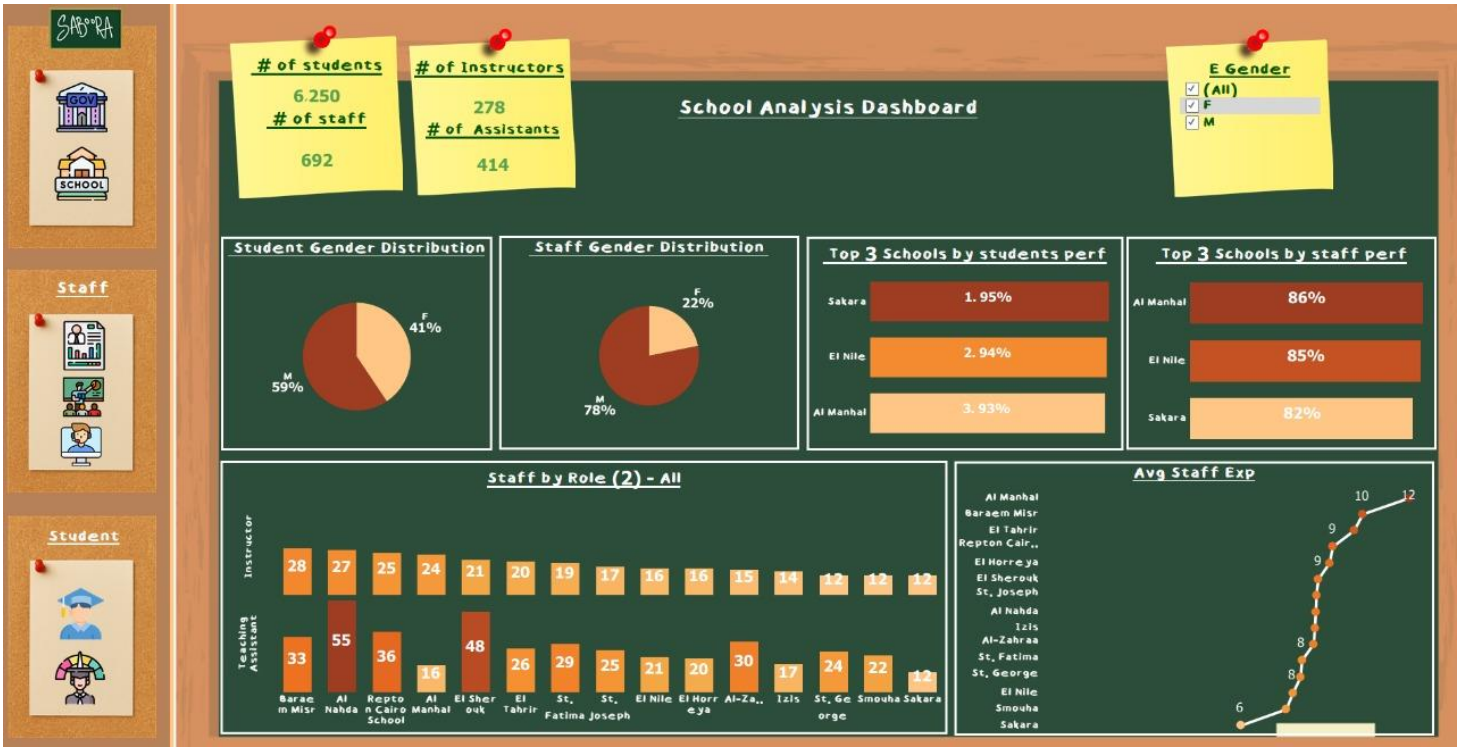
```
SELECT
    s.SchoolID,
    AVG(p.Performance_Score) AS AvgInstructor_Performance
FROM Employee AS s
JOIN StaffPerformance AS p
    ON s.EmployeeID = p.EmployeeID
WHERE s.Role = 'Instructor'
GROUP BY s.SchoolID
ORDER BY AvgInstructor_Performance DESC;
```

	SchoolID	AvgInstructor_Performance
1	TR	89.8180004119873
2	ML	88.4116662343343
3	SG	88.3824990590413
4	NL	88.196249961853
5	ZH	87.9906661987305
6	HR	87.8075008392334
7	SH	87.0842859177362
8	RC	86.7852001953125
9	BM	85.7710710253034
10	SF	84.6910528885691
11	SJ	83.7747053258559
12	NS	82.7718519987883
13	SM	82.2325000762939
14	IZ	81.7235723223005
15	SK	81.2891667683919

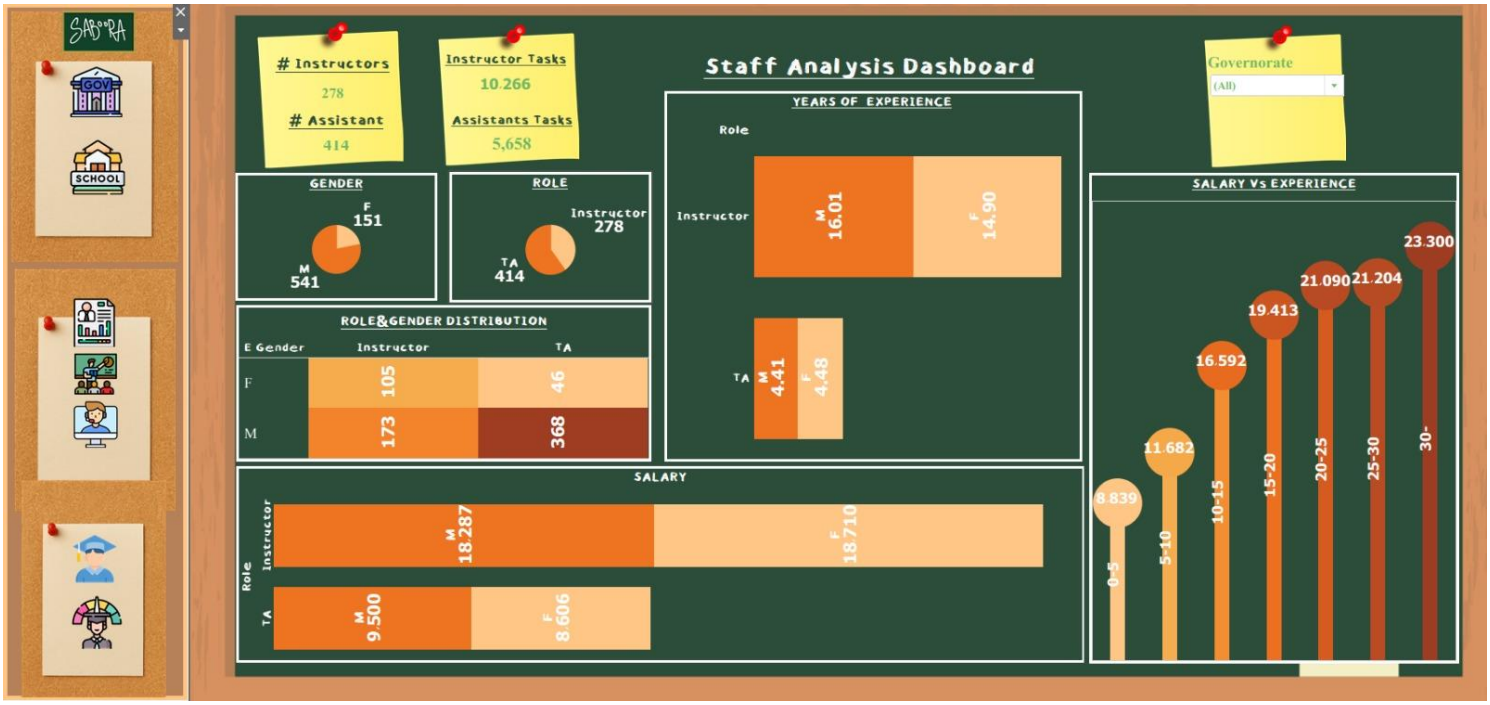
Governorate Dashboard



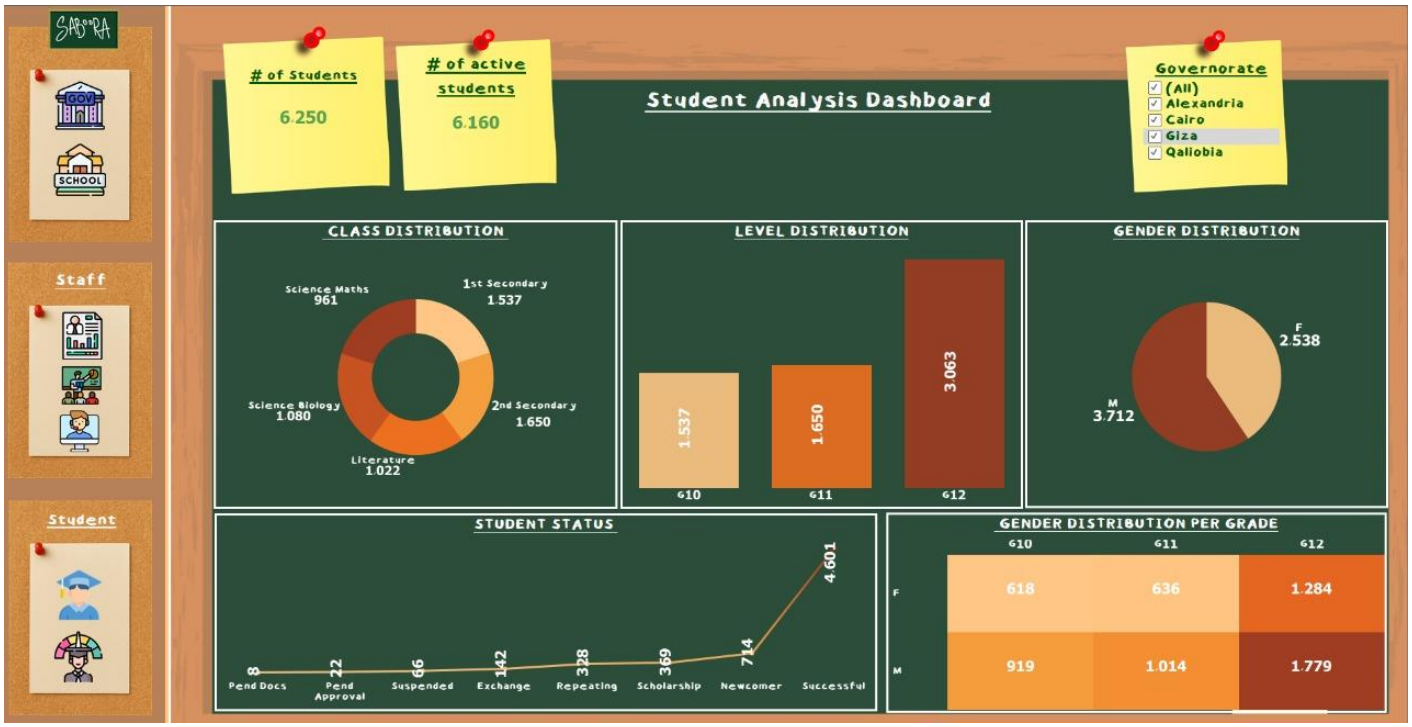
School Dashboard



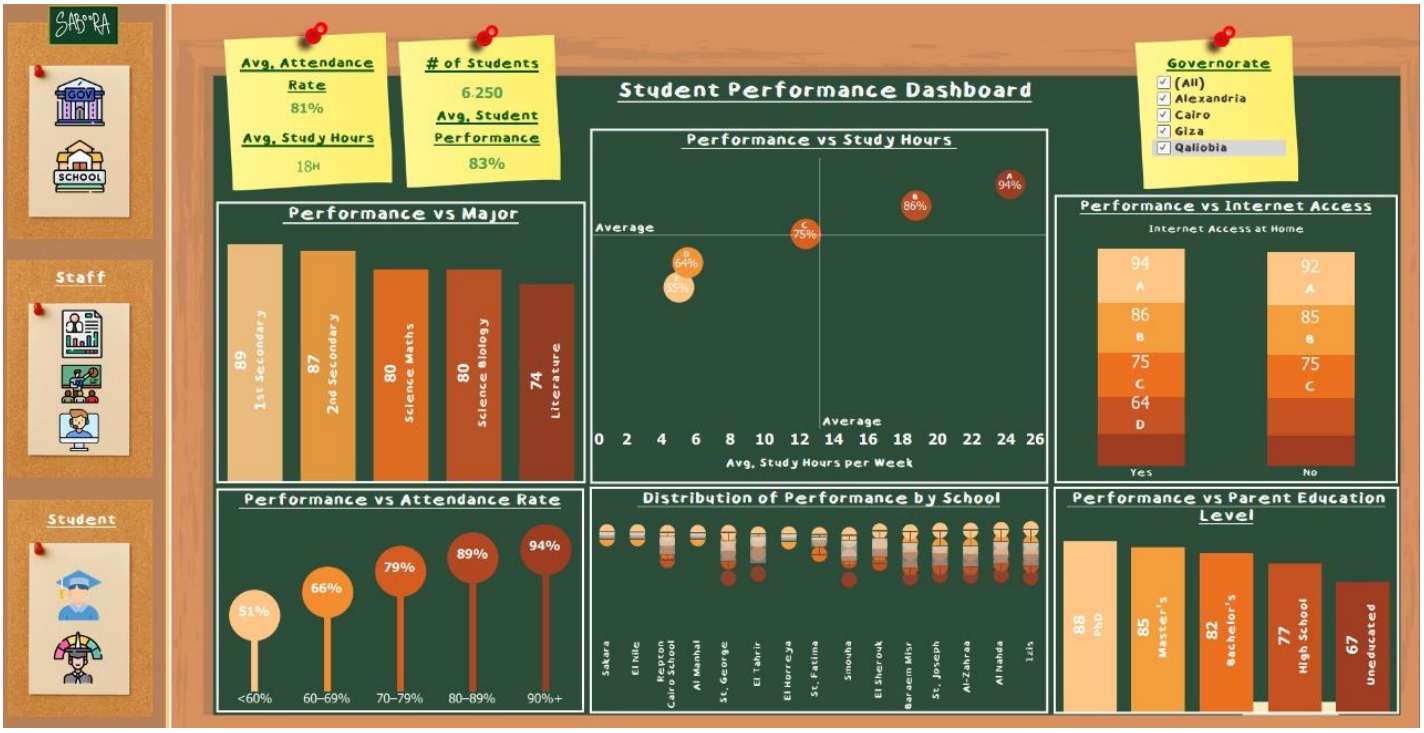
Staff Dashboard



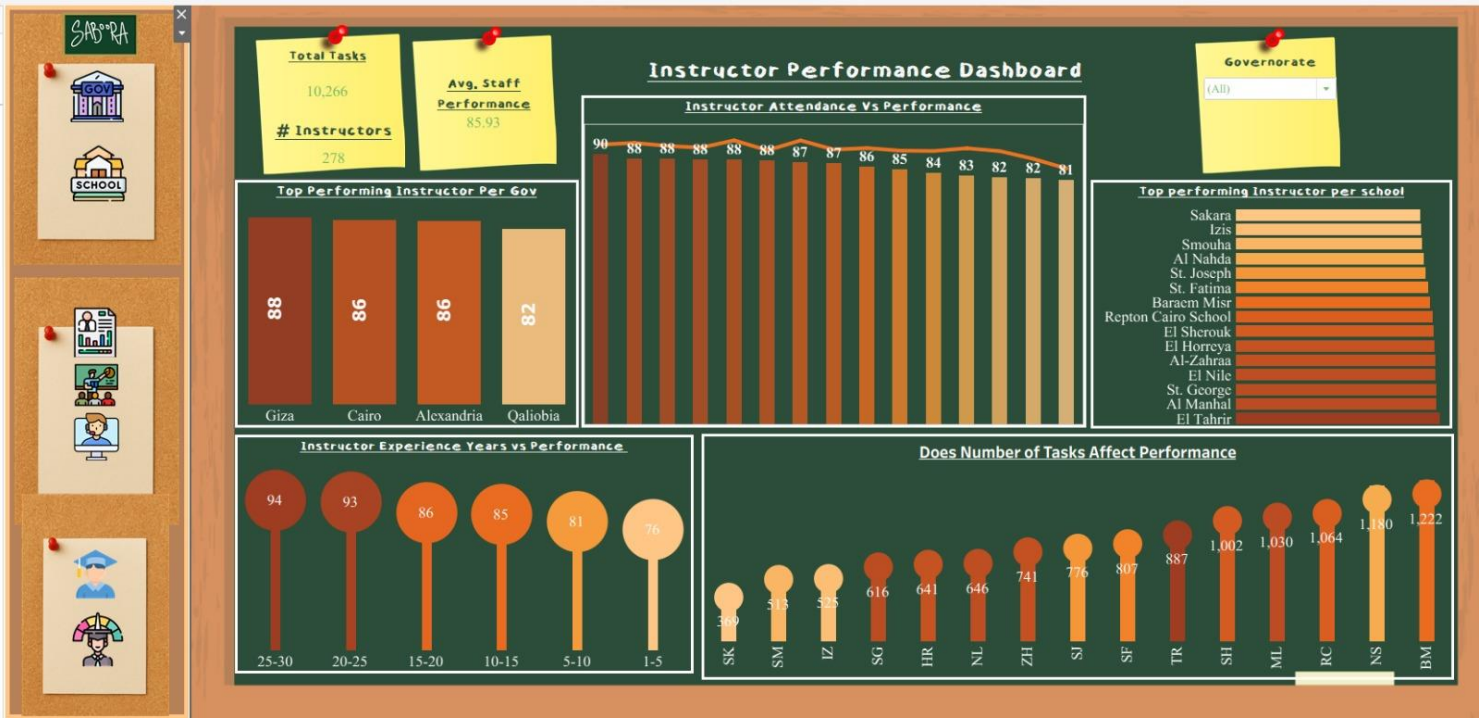
Student Dashboard



Student Performance Dashboard



Instructor Performance Dashboard



TA Performance Dashboard