MySQL to Excel

Dashboard Automation

Andra Constandache

12/10/24

Content

1. Finding the right data set
2. Cleaning data set with Python
3. Uploading data in MySQL
4. Selecting relevant data for Pivot tables
5. Creating automated connection between MySQL and Excel
6. Downloading relevant data to unique Excel sheets
7. Processing Pivot tables
8. Create useful Dashboard
9. Check “New Data” refresh working

1. Finding the right data set

https://www.kaggle.com/datasets/mpwolke/cusersmarildownloadshospcsv

"The dataset contains hours worked by hospital employee classification and by hospital cost center groupings."

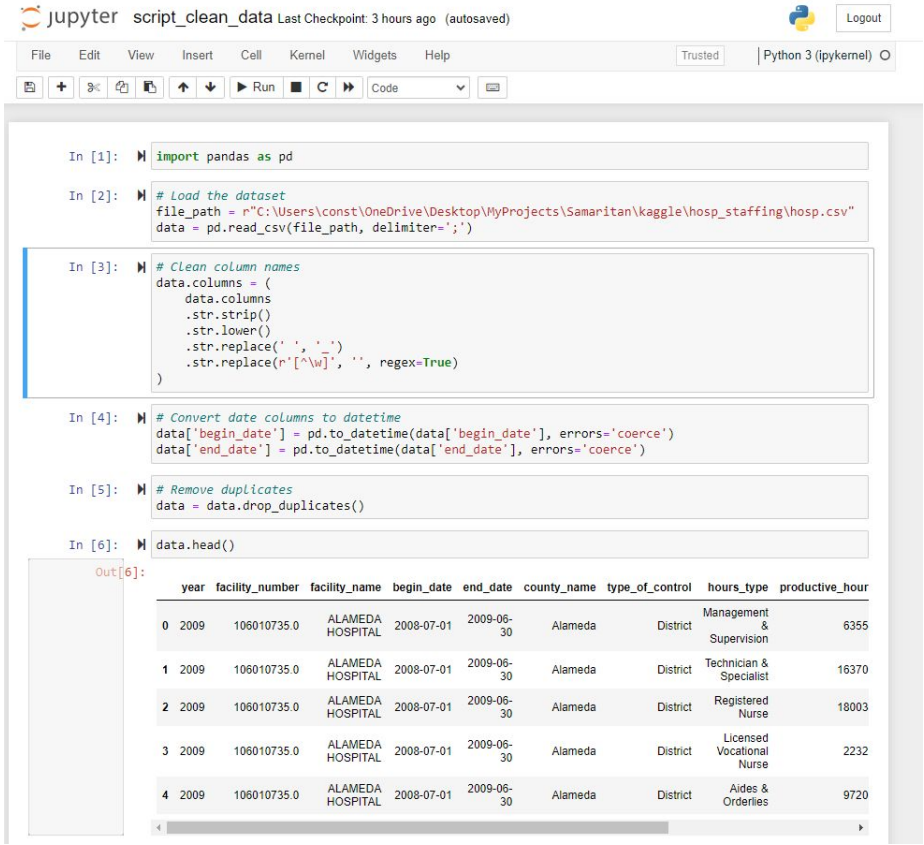
"The dataset contains hours worked by hospital employee classification and by hospital cost center groupings, as well as adjusted patient days for all licensed, comparable hospitals in California. State mental hospitals and psychiatric health facilities are excluded."

Source: http://hcai.ca.gov/HID/DataFlow/

# year	# facility_number	facility_name	begin_date	end_date	county_
year	facility_number	facility_name	begin_date	end date	county na
0 total values	37604 total values	[null] 100%	37604 total values	37604 total values	[null]
2009	106010735	ALAMEDA HOSPITAL	07/01/2008	06/30/2009	Alameda
2009	106010735	ALAMEDA HOSPITAL	07/01/2008	06/30/2009	Alameda
2009	106010735	ALAMEDA HOSPITAL	07/01/2008	06/30/2009	Alameda
2009	106010735	ALAMEDA HOSPITAL	07/01/2008	06/30/2009	Alameda
2009	106010735	ALAMEDA HOSPITAL	07/01/2008	06/30/2009	Alameda
2009	106010735	ALAMEDA HOSPITAL	07/01/2008	06/30/2009	Alameda
2009	106010735	ALAMEDA HOSPITAL	07/01/2008	06/30/2009	Alameda

2. Cleaning data set with Python

This script cleans and formats raw hospital staffing data, making it suitable for analysis, dashboards, and SQL-based reporting. It ensures data quality by handling missing values, ensuring compatibility with SQL, and preparing specialized datasets for different dashboard types.



	year	facility_number	facility_name	begin_date	end_date	county_name	type_of_control	hours_type	productive_hour
0	2009	106010735.0	ALAMEDA HOSPITAL	2008-07-01	2009-06-30	Alameda	District	Management & Supervision	6355
1	2009	106010735.0	ALAMEDA HOSPITAL	2008-07-01	2009-06-30	Alameda	District	Technician & Specialist	16370
2	2009	106010735.0	ALAMEDA HOSPITAL	2008-07-01	2009-06-30	Alameda	District	Registered Nurse	18003
3	2009	106010735.0	ALAMEDA HOSPITAL	2008-07-01	2009-06-30	Alameda	District	Licensed Vocational Nurse	2232
4	2009	106010735.0	ALAMEDA HOSPITAL	2008-07-01	2009-06-30	Alameda	District	Aides & Orderlies	9720

2. Cleaning data set with Python

Import and Load the Dataset

- **Purpose**: Load the dataset from a CSV file into a Pandas DataFrame for processing.
- **Key Steps**:
  - The file is read using `pd.read_csv()` with `;` as the delimiter.
  - Column names are standardized for consistency:
    - Stripped of spaces.
    - Converted to lowercase.
    - Replaced spaces and special characters with underscores.

In [1]:
```
import pandas as pd
```

In [2]:
```
# Load the dataset
file_path = r"C:\Users\const\OneDrive\Desktop\MyProjects\Samaritan\kaggle\hosp_staffing\hosp.csv"
data = pd.read_csv(file_path, delimiter=';')
```

2. Cleaning data set with Python

Clean and Format Data

- **Standardizing Columns**:
  - Converts `begin_date` and `end_date` columns to a proper `datetime` format to ensure they can be used in time-based calculations.
- **Remove Duplicates**:
  - Removes duplicate rows to avoid redundant entries.

In [3]:
```
# Clean column names
data.columns = (
    data.columns
    .str.strip()
    .str.lower()
    .str.replace(' ', '_')
    .str.replace(r'[^\w]', '', regex=True)
)
```

In [4]:
```
# Convert date columns to datetime
data['begin_date'] = pd.to_datetime(data['begin_date'], errors='coerce')
data['end_date'] = pd.to_datetime(data['end_date'], errors='coerce')
```

In [5]:
```
# Remove duplicates
data = data.drop_duplicates()
```

2. Cleaning data set with Python

Analyze Missing Data

- **Inspect Missing Values**:
  - Uses `data.isnull().sum()` to summarize missing values for each column.
  - Displays rows with missing data for manual inspection (`data[data.isnull().any(axis=1)]`).
- **Purpose**: Helps decide whether to drop, fill, or flag missing values based on their significance.

```
]:  # Check for missing values in the dataset
    missing_values_before = data.isnull().sum()

    # Display the missing values for each column
    missing_values_before
```

```
it[9]: year                                    0
       facility_number                        85
       facility_name                          85
       begin_date                             85
       end_date                               85
       county_name                             0
       type_of_control                        85
       hours_type                              0
       productive_hours                        0
       productive_hours_per_adjusted_patient_day    187
       dtype: int64
```

```
]:  # Filter rows with missing values to inspect them
    missing_rows = data[data.isnull().any(axis=1)]

    # Display the first few rows with missing values
    missing_rows.head()
```

2. Cleaning data set with Python

Create Specialized DataFrames for Dashboards

1. **Facility Productivity Dashboard**:
   - Drops rows with missing `facility_name` or `productive_hours`.
   - Fills missing `productive_hours_per_adjusted_patient_day` with `0`.
2. **Staffing Composition Dashboard**:
   - Drops rows with missing `hours_type` or `productive_hours`.
   - Focuses on workforce distribution and efficiency.
3. **Yearly Trends Dashboard**:
   - Drops rows missing `year` or `productive_hours`.
   - Fills missing values for `productive_hours_per_adjusted_patient_day` with `0`.
4. **Operational Risk Dashboard**:
   - Retains rows with missing `productive_hours_per_adjusted_patient_day`.
   - Adds a flag (`is_risk`) to highlight rows with critical data issues.
5. **Facility Comparison Report**:
   - Drops rows with missing `facility_name`, `productive_hours`, or `begin_date`.
   - Fills missing productivity-related values with `0` for meaningful comparisons.

1. Facility Productivity Dashboard

```
n [11]:
# Drop rows with missing facility_name or productive_hours
data_productivity = data.dropna(subset=['facility_name', 'productive_hours']).copy()

# Fill missing values for productive_hours_per_adjusted_patient_day with 0
data_productivity.loc[:, 'productive_hours_per_adjusted_patient_day'] = data_productivity['product
```

2. Staffing Composition by Role

```
n [12]:
# Drop rows with missing hours_type or productive_hours
data_staffing = data.dropna(subset=['hours_type', 'productive_hours'])

# Retain county_name and facility_name NaNs if grouping is not critical
# No additional handling needed for non-numeric fields here.
```

3. Yearly Trend Analysis

```
n [13]:
# Drop rows with missing year or productive_hours
data_trend = data.dropna(subset=['year', 'productive_hours'])

# Fill NaNs in productive_hours_per_adjusted_patient_day with 0
data_trend['productive_hours_per_adjusted_patient_day'] = data_trend['productive_hours_per_adjuste
```

4. Operational Risk Dashboard

```
n [14]:
# Retain rows with missing productive_hours_per_adjusted_patient_day for analysis
data_risk = data.copy()

# Add a flag column to highlight rows with missing values in critical columns
data_risk['is_risk'] = data_risk['productive_hours_per_adjusted_patient_day'].isnull()
```

5. Facility Comparison Report

```
n [15]:
# Drop rows with missing facility_name, productive_hours, or begin_date
data_comparison = data.dropna(subset=['facility_name', 'productive_hours', 'begin_date']).copy()

# Fill missing values for productive_hours_per_adjusted_patient_day with 0
data_comparison['productive_hours_per_adjusted_patient_day'] = data_comparison['productive_hours_p
```

2. Cleaning data set with Python

Prepare Data for SQL Export

- **Add a Primary Key**:
  - Adds a unique `id` column as an auto-incrementing identifier for SQL import.
- **Ensure SQL-Compatible Data Types**:
  - Converts `facility_number` to `Int64` for consistency.
  - Replaces single quotes in text fields with double single quotes to avoid SQL errors.
- **Format Date Columns**:
  - Converts `datetime` columns (`begin_date`, `end_date`) to SQL-friendly `YYYY-MM-DD` format.
- **Round Float Values**:
  - Rounds `productive_hours_per_adjusted_patient_day` to two decimal places for precision.
- **Handle Null Values**:
  - Replaces missing values (`NaN`) with SQL-compatible `NULL` using `None`.

SQL PREPARATION

In [19]:
```
data_comparison['id'] = range(1, len(data_comparison) + 1)
```

In [17]:
```
print(data_comparison.dtypes)
```

```
year                                      int64
facility_number                         float64
facility_name                            object
begin_date                       datetime64[ns]
end_date                         datetime64[ns]
county_name                              object
type_of_control                          object
hours_type                               object
productive_hours                          int64
productive_hours_per_adjusted_patient_day       float64
dtype: object
```

In [20]:
```
# Ensure correct data types and clean for SQL import
# Assuming 'data_comparison' is your cleaned DataFrame.

# Convert facility_number to an integer type if applicable (handles NaN as NULL-friendly Int64)
# This avoids unnecessary decimals in SQL.
data_comparison['facility_number'] = data_comparison['facility_number'].astype('Int64')
```

In [21]:
```
# Replace special characters in facility_name to ensure SQL compatibility
# Replacing single quotes with double single quotes to prevent SQL errors.
data_comparison['facility_name'] = data_comparison['facility_name'].str.replace("'", "''")
```

In [22]:
```
# Format date columns for SQL-friendly format (YYYY-MM-DD)
# Ensures that the dates are stored in a readable and consistent SQL format.
data_comparison['begin_date'] = data_comparison['begin_date'].dt.strftime('%Y-%m-%d')
data_comparison['end_date'] = data_comparison['end_date'].dt.strftime('%Y-%m-%d')
```

In [23]:
```
# Round float values for precision consistency in SQL
# Limiting to 2 decimal places for clarity and storage efficiency.
data_comparison['productive_hours_per_adjusted_patient_day'] = (
    data_comparison['productive_hours_per_adjusted_patient_day'].round(2)
)
```

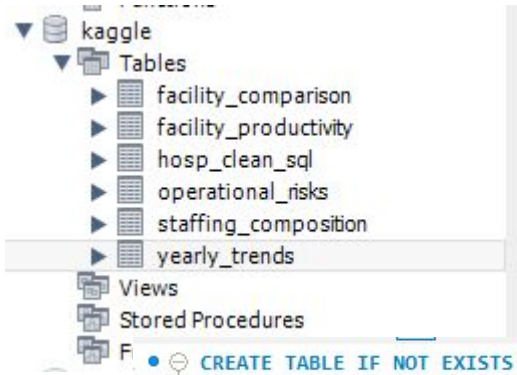
In [24]:
```
# Replace NaN values with None (to map to SQL NULL values)
# This ensures missing data is properly interpreted as NULL in SQL.
data_comparison = data_comparison.where(pd.notnull(data_comparison), None)
```

In [26]:
```
# Export cleaned data to a CSV file for SQL import
# This CSV can then be uploaded to a database directly or through an ETL tool.
csv_output_path = 'hosp_clean_sql.csv'
data_comparison.to_csv(csv_output_path, index=False)
```

3. Uploading data in MySQL

This step involves creating a structured table in the MySQL database (`kaggle` schema) and uploading the cleaned data into the database for further analysis or querying.

1. **Table Name**: `kaggle.hosp_clean`.
   - This table serves as the foundational dataset in MySQL, capturing cleaned and structured hospital-related data.
2. **Schema Overview**:
   - **Primary Key (`id`)**:
     - Automatically increments for each row, ensuring every record is uniquely identifiable.
   - **Columns**:
     - Represent critical attributes like `facility_name`, `county_name`, `begin_date`, and productivity metrics (`productive_hours`, `productive_hours_per_adjusted_patient_day`).
     - Data types (`INT`, `VARCHAR`, `DATE`, `FLOAT`) ensure the table is optimized for SQL querying.
3. **CREATE TABLE Statement**:
   - Includes the `IF NOT EXISTS` clause to prevent overwriting if the table already exists.
   - Comments within the SQL script provide clarity on each column's purpose.




```sql
CREATE TABLE IF NOT EXISTS kaggle.hosp_clean (
    id INT AUTO_INCREMENT PRIMARY KEY, -- Unique identifier
    year INT,
    facility_number INT,
    facility_name VARCHAR(500),
    begin_date DATE,
    end_date DATE,
    county_name VARCHAR(500),
    type_of_control VARCHAR(500),
    hours_type VARCHAR(500),
    productive_hours INT,
    productive_hours_per_adjusted_patient_day FLOAT
);
```

3. Uploading data in MySQL

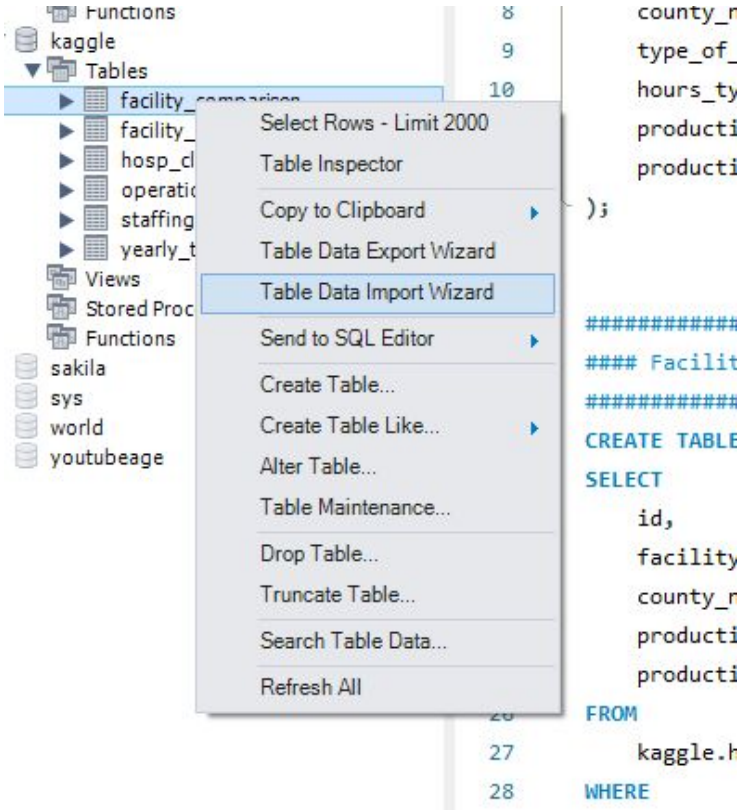
Define Table:

- The `CREATE TABLE` statement structures the data in MySQL according to the schema.

Upload Cleaned Data:
After defining the table, cleaned data (e.g., from `hosp_clean_sql.csv`) can be uploaded using SQL commands or the Wizard

Purpose:

- This step ensures data consistency, making it accessible for analysis via SQL queries or connecting to visualization tools like Power BI or Excel.



4. Selecting relevant data for Pivot tables

```
##################################
#### Facility Productivity Table ###
##################################
CREATE TABLE IF NOT EXISTS kaggle.facility_productivity AS
SELECT
    id,
    facility_name,
    county_name,
    productive_hours,
    productive_hours_per_adjusted_patient_day
FROM
    kaggle.hosp_clean_sql
WHERE
    productive_hours IS NOT NULL;
SELECT * FROM kaggle.facility_productivity LIMIT 10;
```

```
##################################
#### Staffing Composition Table ###
##################################
CREATE TABLE IF NOT EXISTS kaggle.staffing_composition AS
SELECT
    id,
    facility_name,
    hours_type,
    productive_hours
FROM
    kaggle.hosp_clean_sql
WHERE
    hours_type IS NOT NULL AND productive_hours IS NOT NULL;
SELECT * FROM kaggle.staffing_composition LIMIT 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch

id	facility_name	hours_type	productive_hours
1	ALAMEDA HOSPITAL	Management & Supervision	63558
2	ALAMEDA HOSPITAL	Technician & Specialist	163706
3	ALAMEDA HOSPITAL	Registered Nurse	180034
4	ALAMEDA HOSPITAL	Licensed Vocational Nurse	22323
5	ALAMEDA HOSPITAL	Aides & Orderlies	97205
6	ALAMEDA HOSPITAL	Clerical & Other Administrative	134039
7	ALAMEDA HOSPITAL	Environmental & Food Services	80229
8	ALAMEDA HOSPITAL	Other	0
9	ALAMEDA HOSPITAL	Contracted Registry Nursing	31481

staffing_composition 15

Simplifies Data Handling:

- Extracts only the columns and rows needed for specific dashboards or pivot tables, reducing unnecessary complexity.

Optimized for Pivot Tables:

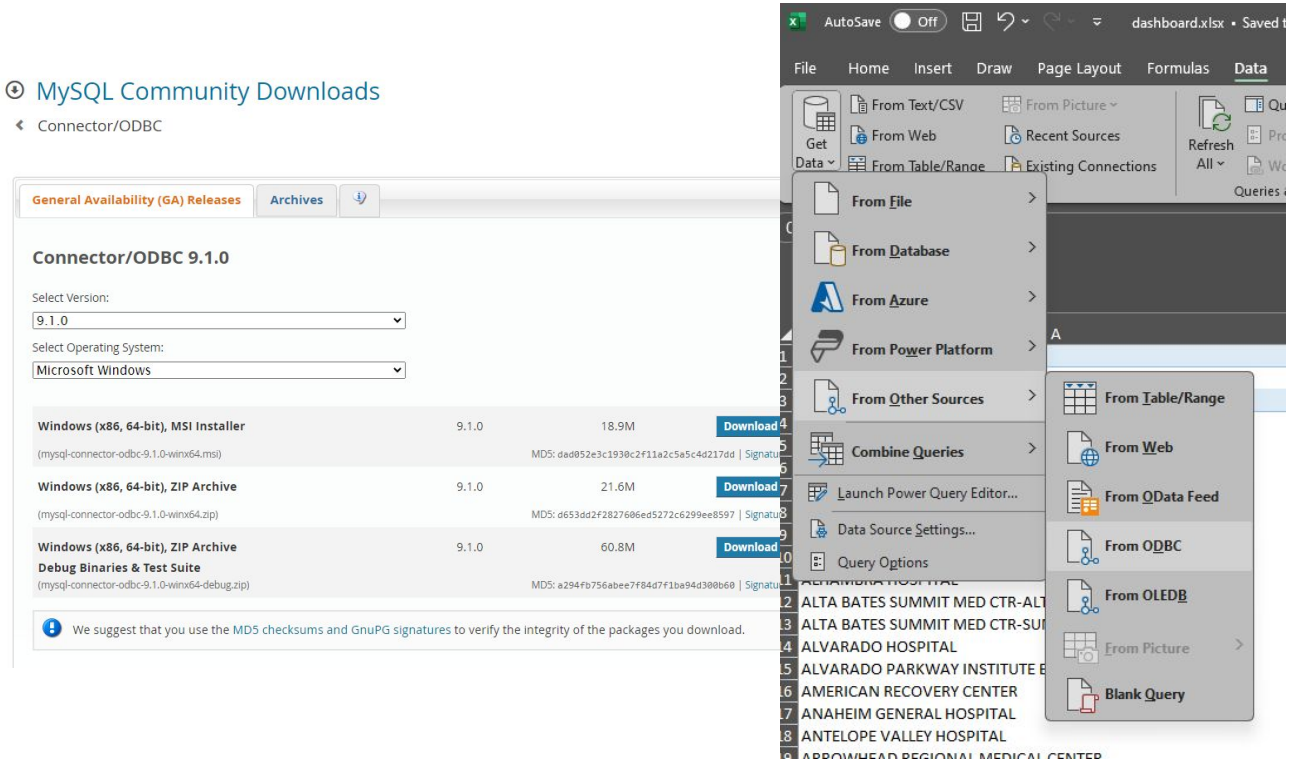
- Tables are tailored to answer specific business questions (e.g., productivity by facility, staffing by roles), making them ready for visualization without additional processing.

Improves Performance:

- Narrowing down the dataset improves query and reporting speed, especially for large datasets.

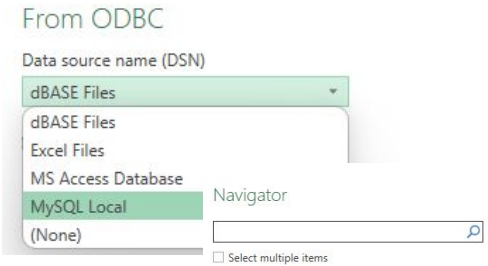
5. Creating automated connection between MySQL and Excel

This step establishes a dynamic connection between MySQL and Excel using the **MySQL ODBC Connector**. By downloading and configuring the connector, Excel can pull live data directly from MySQL tables, enabling automated updates for dashboards and reports.



6. Downloading relevant data to unique Excel sheets

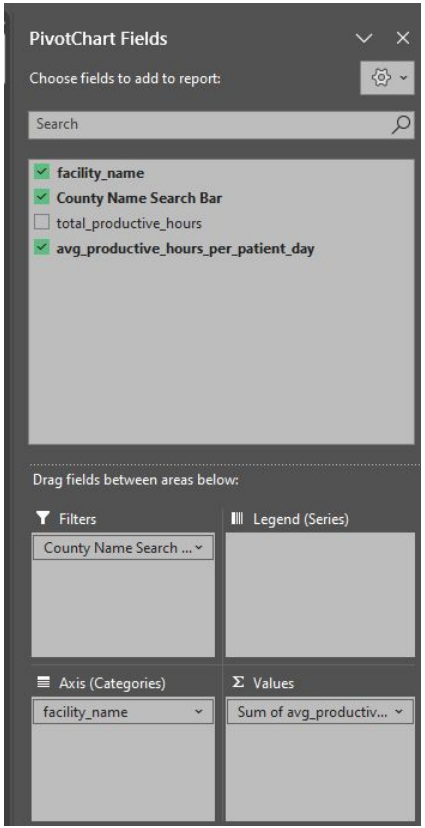
This step connects Excel to MySQL via the ODBC connector and allows users to select specific tables (e.g., `facility_comparison`, `facility_productivity`) from the database. The chosen data is downloaded into unique Excel sheets, enabling analysis and visualization directly within Excel.



	A	B
1	County Name Search Bar	(All)
2		
3	Row Labels	Sum of avg_productive
4	AAATest Facility	22
5	ADVENTIST MEDICAL CENTER - HANFORD	2.759705882
6	ADVENTIST MEDICAL CENTER - REEDLEY	3.808823529
7	AGNEWS STATE HOSPITAL	0
8	AHMC ANAHEIM REGIONAL MEDICAL CENTER	2.919411765
9	ALAMEDA COUNTY MEDICAL CENTER	3.271882353
10	ALAMEDA HOSPITAL	1.701529412
11	ALHAMBRA HOSPITAL	2.678
12	ALTA BATES SUMMIT MED CTR-ALTA BATES CAMPUS	4.193882353
13	ALTA BATES SUMMIT MED CTR-SUMMIT CAMPUS-HAWTHORNE	3.867411765
14	ALVARADO HOSPITAL	3.590588235
15	ALVARADO PARKWAY INSTITUTE BHS	1.494705882
16	AMERICAN RECOVERY CENTER	0.376588235
17	ANAHEIM GENERAL HOSPITAL	3.140294118
18	ANTELOPE VALLEY HOSPITAL	3.455294118
19	ARROWHEAD REGIONAL MEDICAL CENTER	3.316823529
20	ARROYO GRANDE COMMUNITY HOSPITAL	3.455411765
21	ATASCADERO STATE HOSPITAL	0
22	AURORA BEHAVIORAL HEALTHCARE - SANTA ROSA	11.05647059
23	AURORA CHARTER OAK	1.472235294
24	AURORA LAS ENCINAS HOSPITAL	2.004235294
25	AURORA SAN DIEGO	1.439411765
26	AURORA VISTA DEL MAR HOSPITAL	1.463529412
27	BAKERSFIELD HEART HOSPITAL	3.368470588
28	BAKERSFIELD MEMORIAL HOSPITAL	2.957647059
29	BALLARD REHABILITATION HOSPITAL	2.821882353
30	BANNER LASSEN MEDICAL CENTER	3.266117647
31	BARLOW HOSPITAL	2.799764706
32	BARSTOW COMMUNITY HOSPITAL	3.489607843
33	BARTON MEMORIAL HOSPITAL	2.571294118
34	BEAR VALLEY COMMUNITY HOSPITAL	0.788235294

7. Processing Pivot tables

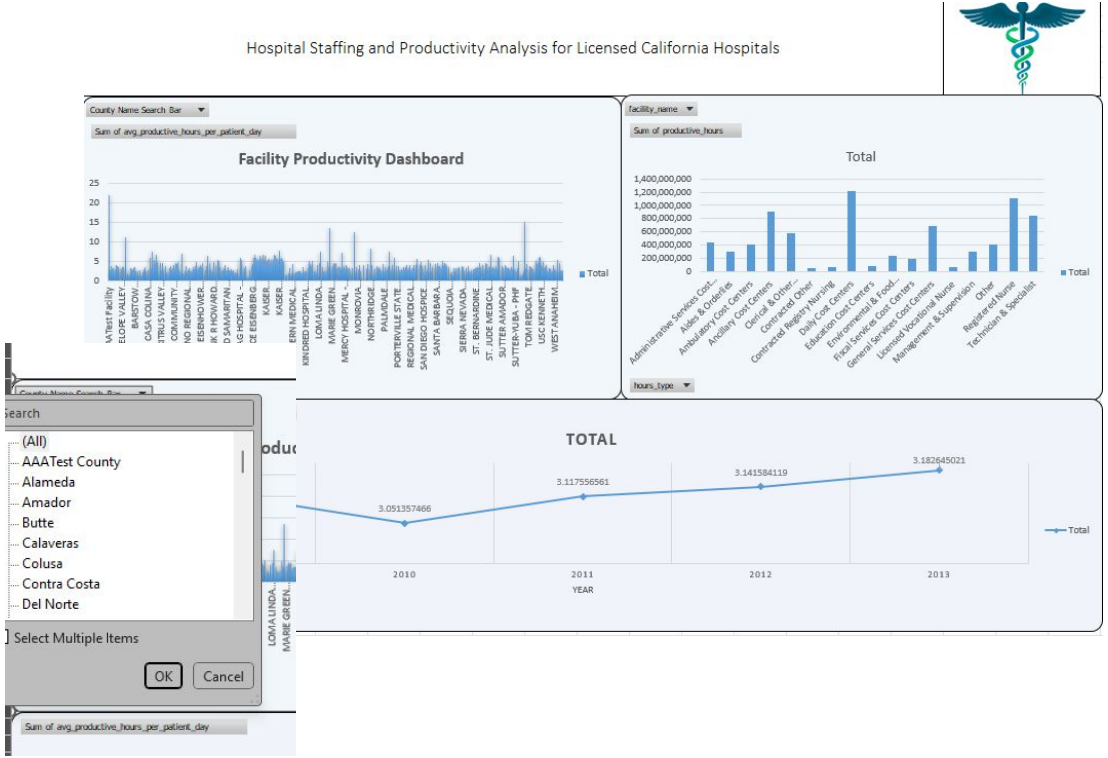
In this step, a **PivotChart** is created by selecting relevant fields from the data source connected to MySQL. The `facility_name` is set as the category on the **Axis**, while the **Values** section summarizes `avg_productive_hours_per_patient_day` to compare performance across facilities. The **County Name Search Bar** filter allows users to interactively refine the chart based on specific counties, making it dynamic and ready for integration into a live Excel dashboard.



8. Create useful Dashboard

In this step, a dynamic and interactive Excel dashboard is created to present key insights from hospital staffing and productivity data. Multiple pivot charts and tables are combined to provide visualizations for facility productivity, staffing composition, and yearly trends.

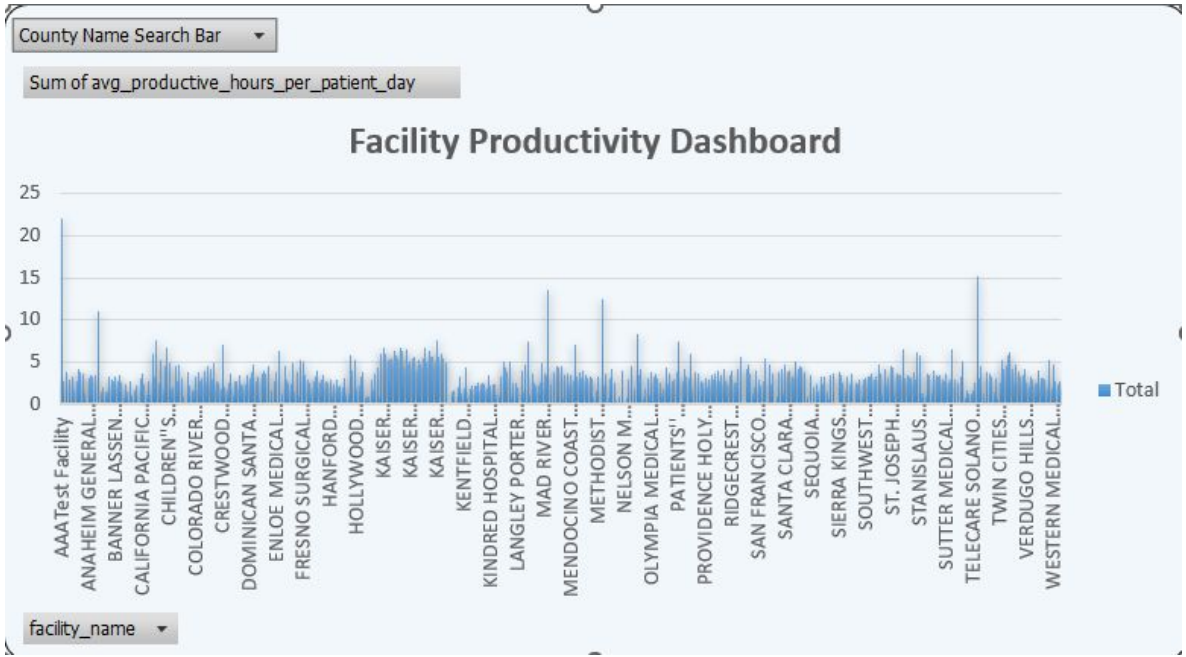
Filters, such as the **County Name Search Bar**, allow users to interact with the data and customize views for specific regions or metrics, making the dashboard actionable and user-friendly for decision-makers.



8. Create useful Dashboard

Importance of the Facility Productivity Dashboard for Leadership

This dashboard provides a clear visualization of **average productive hours per adjusted patient day** for each facility, enabling leadership to quickly assess and compare performance across hospitals. The inclusion of an interactive **County Name Search Bar** allows leaders to focus on specific regions, making it easier to identify underperforming facilities or areas of operational excellence. By highlighting productivity variations, this dashboard supports data-driven decision-making to allocate resources, improve operational efficiency, and benchmark performance across facilities.



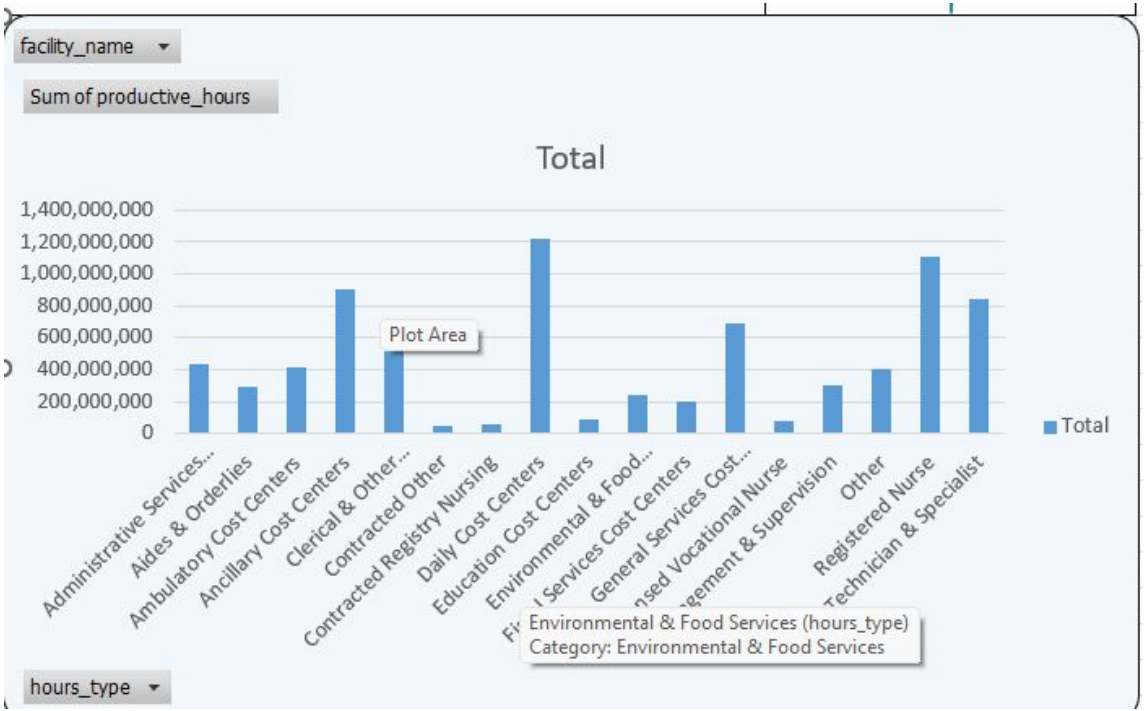
8. Create useful Dashboard

Importance of the Staffing Composition Dashboard for Leadership

This dashboard provides insights into the distribution of **productive hours by job roles (hours_type)** across facilities. Leadership can use it to evaluate workforce allocation, identifying which roles (e.g., nurses, technicians, or support staff) consume the most hours and how they compare across various departments. By analyzing this data, leaders can:

- Address staffing imbalances.
- Optimize workforce planning.
- Make informed decisions on hiring or reallocating resources to meet operational demands and improve efficiency.

It ensures that staffing aligns with organizational goals and patient care needs.



8. Create useful Dashboard

Importance of the Yearly Trends Dashboard for Leadership

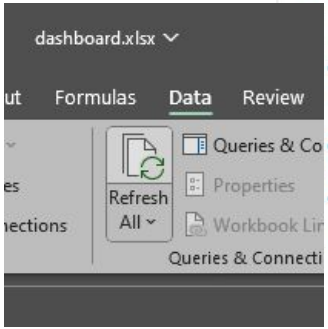
This dashboard tracks the **average productive hours per adjusted patient day** over multiple years, providing leadership with insights into long-term performance trends. It highlights improvements or declines in operational efficiency, enabling leaders to evaluate the effectiveness of past initiatives and identify years with significant changes. By analyzing this data, leadership can:

- Set strategic goals based on historical performance.
- Monitor progress toward improving productivity.
- Allocate resources or implement changes to sustain or accelerate positive trends.

9. Check “New Data” refresh working

By inserting a test row into the `facility_comparison` table, it ensures that updates in the database are accurately reflected in the Excel dashboard after refreshing data.

I hope this presentation showcases critical skills in creating **dynamic, automated reporting systems**, enabling real-time data-driven decisions with efficient database-to-dashboard integration.




```
102
103   -- test autorefresh by adding a test row in facility_comparison table
104   DESCRIBE kaggle.facility_comparison;
105
106   INSERT INTO kaggle.facility_comparison (
107       facility_name,
108       county_name,
109       total_productive_hours,
110       avg_productive_hours_per_patient_day
111   )
112   VALUES (
113       'AAATest Facility',       -- Replace with a test facility name
114       'AAATest County',         -- Replace with a test county name
115       1000,                  -- Total productive hours
116       5.5                    -- Average productive hours per patient day
      );

      SELECT * FROM kaggle.facility_comparison
      WHERE facility_name = 'AAATest Facility';

      SET SQL_SAFE_UPDATES = 0;

      DELETE FROM kaggle.facility_comparison
      WHERE facility_name = 'Test Facility';
```

Thank you.