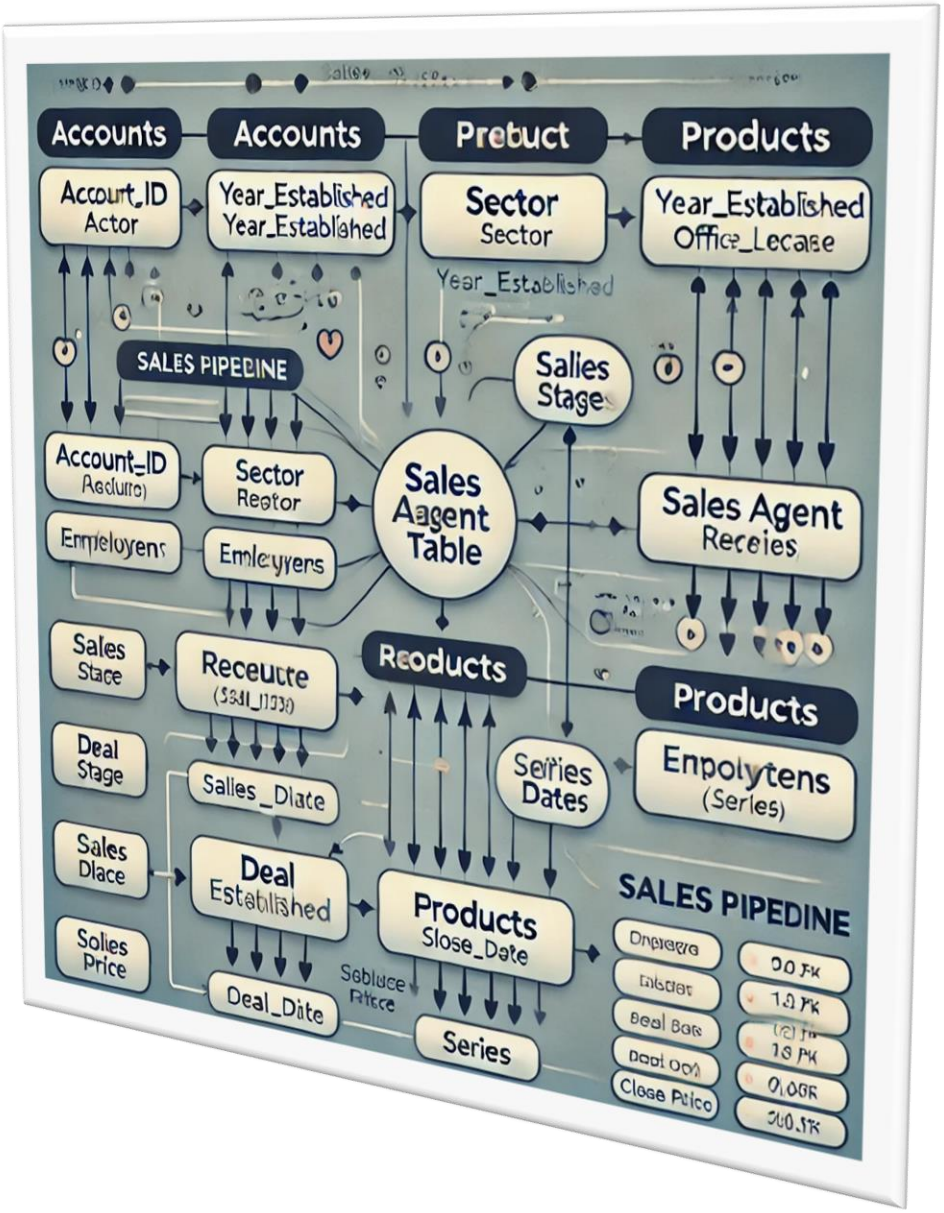
DEPI Project



Group:

ALX1_DAT1_G4d

By:

- **Abdelrahman Saeed**
- **Mohamed Saied**
- **Mohamed Ismael**
- **Noha Mohamed**

Instructor:

Jackie Abualem

Sales Opportunities by CRM Project

Microsoft SQL Server

python™

Power BI

CRM Sales Opportunities Project

Description:

This project involves the analysis of B2B sales pipeline data for a fictitious company that specializes in selling computer hardware. The objective is to gain actionable insights into the company's sales operations by examining various aspects of the data, including sales opportunities, product performance, account activities, and sales team efficiency.

Data Source: Maven Analytics.

Objectives:

The main goals of this project are to:

- **Assess Sales Team Performance.** How is each sales team performing in comparison to others? Identifying the top-performing teams and those that may need improvement.
- **Evaluate Sales Agent Productivity.** Are any sales agents significantly lagging behind? Determining which agents are underperforming and may require additional support or training.
- **Identify Trends.** Can any quarter-over-quarter trends be observed? Understanding patterns in sales performance.
- **Analyze Product Success Rates.** Do any products have better win rates than others? Investigating which products are more successful in closing deals.

Tables:

- accounts.csv: Contains information about the companies (accounts) involved in sales opportunities.
    - *account*: Company name
    - *sector*: Industry sector of the company
    - *year_established*: Year the company was established

    - *revenue*: Annual revenue of the company (in millions of USD)
    - *employees*: Number of employees
    - *office_location*: Headquarters location
    - *subsidiary_of*: Parent company, if applicable
- sales_teams.csv: Provides details about the sales agents and their management.
    - *sales_agent*: Name of the sales agent
    - *manager*: Name of the respective sales manager
    - *regional_office*: Location of the regional office
- products.csv: Contains data on the computer hardware products offered by the company.
    - *product*: Product name
    - *series*: Product series or family
    - *sales_price*: Suggested retail price
- sales_pipeline.csv: Records details of each sales opportunity within the pipeline.
    - *opportunity_id*: Unique identifier for each sales opportunity
    - *sales_agent*: Sales agent responsible for the opportunity
    - *product*: Product involved in the opportunity
    - *account*: Company involved in the opportunity, if applicable
    - *deal_stage*: Stage of the sales pipeline (e.g., Prospecting > Engaging > Won / Lost)
    - *engage_date*: Date when the "Engaging" stage of the deal was initiated, if applicable
    - *close_date*: Date when the deal was either "Won" or "Lost", if applicable
    - *close_value*: Revenue generated from the deal, if applicable

Data Transformation

The following transformation steps were taken to prepare the data for efficient analysis.

1. **Add ID columns:** Unique identifier (ID) columns were added to the products.csv, accounts.csv, and sales_teams.csv files. These IDs are now referenced in the sales_pipeline.csv file, standardizing the data and improving query efficiency.

```
import pandas as pd
import numpy as np
import csv
import json
```

```
path = 'original_files/'
modified_path = 'modified_files/'
dataframes = ['accounts.csv', 'products.csv', 'sales_teams.csv']
```

```
for dataframe in dataframes:

    # Construct the full path to the CSV file
    full_path = path + dataframe

    # Read the CSV file into a DataFrame
    df = pd.read_csv(full_path)

    # Find the maximum index in the DataFrame and create a new range for IDs
    max_idx = df.index.max() + 1  # Get the maximum index and add 1 for ID generation
    id_values = np.arange(1, max_idx + 1, 1)  # Create an array of IDs starting from 1

    # Insert the new 'id' column at the beginning of the DataFrame
    df.insert(0, 'id', id_values)

    # Construct the save path for the modified CSV file
    save_path = modified_path + dataframe

    # Save the modified DataFrame back to a CSV file without the index
    df.to_csv(save_path, index=False)
```

2. **Create mapping dictionaries:** Dictionaries were created to map IDs to full names for the products.csv, accounts.csv, and sales_teams.csv files. These dictionaries function as lookup tables, ensuring that each ID accurately corresponds to the appropriate name in the data.

```
dict_name = ['accounts_dict', 'products_dict', 'teams_dict']

for i in range(len(dict_name)):
    # Assuming modified_path and dataframe are defined elsewhere
    full_path = modified_path + dataframes[i]
    df = pd.read_csv(full_path)

    # Create the dictionary from the DataFrame
    temp_dict = df.iloc[:, 1].to_dict()

    # Increment keys by 1 and convert to string keys
    temp_dict = {key + 1: value for key, value in temp_dict.items()}
    temp_dict = {str(key): value for key, value in temp_dict.items()}
    # Swap keys and values
    temp_dict = {value: key for key, value in temp_dict.items()}
    # Store the dictionary in the global namespace using the name from dict_name
    globals()[dict_name[i]] = temp_dict

print(products_dict)
```

```
{'GTX Basic': '1', 'GTX Pro': '2', 'MG Special': '3', 'MG Advanced': '4', 'GTX Plus Pro': '5', 'GTX Plus Basic': '6', 'GTK 500': '7'}
```

3. **Validate data consistency:** The dictionaries were cross-referenced with the sales_pipeline.csv file to identify any discrepancies, such as unmatched names between the sales_pipeline.csv file and the dictionaries. This validation step was essential for maintaining data integrity.

```
# Define the path to the sales pipeline CSV file
sales_pipeline_path = f'{path}sales_pipeline.csv'

# Read the sales pipeline data into a DataFrame
df_sales = pd.read_csv(sales_pipeline_path)

# Define a dictionary mapping column names to their corresponding dictionaries
data = {'product': products_dict, 'account': accounts_dict, 'sales_agent': teams_dict}

# Iterate over each column and its corresponding dictionary
for column, dict in data.items():
    print(f'Sales_pipeline column - {column}:')

    # Check for non-matching values in the sales pipeline column against the dictionary keys
    x = df_sales[~df_sales[column].isin(dict.keys())][column].unique()

    if len(x) != 0:
        # If there are non-matching values, print them
        print(f'Non matching values in sales_pipeline column: {x}.')

        # Check for any keys in the dictionary that are not present in the DataFrame's unique values
        for keys in dict.keys():
            y = df_sales[column].unique()  # Get unique values from the DataFrame column
            if keys not in y:
                # If a key from the dictionary is not found in the DataFrame's unique values, print it
                print(f'Non matching values in the dictionary: {keys}.')

    else:
        # If there are no non-matching values, print a message
        print(f'No missing values in the dictionary.')
```

```
Sales_pipeline column - product:
Non matching values in sales_pipeline column: ['GTXPro'].
Non matching values in the dictionary: GTX Pro.
Sales_pipeline column - account:
Non matching values in sales_pipeline column: [nan].
Sales_pipeline column - sales_agent:
No missing values in the dictionary.
```

4. **Correct misspelled values:** Any misspelled values in the dictionaries were corrected to ensure consistency across all files. This helped prevent potential errors during data replacement that could have led to incorrect insights in future analyses.

```
# replace 'GTX Pro' to 'GTXPro'
products_dict['GTXPro'] = products_dict['GTX Pro']

del products_dict['GTX Pro']
```

5. **Replace full names with IDs:** The full names in the sales_pipeline.csv file were replaced with the corresponding IDs for sales agents, products, and accounts, standardizing the data for optimal querying.

```
for column, dict in data.items():
    df_sales[column] = df_sales[column].map(dict)

# Assuming df_sales is your sales pipeline DataFrame
df_sales = df_sales.rename(columns={
    'sales_agent': 'agent_id',
    'product': 'product_id',
    'account': 'account_id'
})

# Display the first few rows to verify the changes
print(df_sales.head())
```

Data Cleaning

1. **Check on data “Sales”**

```
# Display info about data
print(f"{df_sales.shape[0]} is number of rows")
print(f"{df_sales.shape[1]} is number of columns")
print(f"{df_sales.size} is number of data")
```

```
8800 is number of rows
8 is number of columns
70400 is number of data
```

```
df_sales.isnull().sum()
```

```
opportunity_id       0
agent_id             0
product_id           0
account_id        1425
deal_stage           0
engage_date        500
close_date        2089
close_value       2089
dtype: int64
```

2. **Exploration nulls values:** Null values were identified in the data frame when the 'deal_stage' was either 'Prospecting' or 'Engaging'. In the 'Prospecting' rows, 'account_id', 'engage_date', 'close_date', and 'close_value' were missing. For 'Engaging' rows, 'engage_date', 'close_date', and 'close_value' were missing, with some rows also lacking 'account_id'.

```python
# Display rows with any null values
rows_with_nulls = df_sales[df_sales.isnull().any(axis=1)]
rows_with_nulls
```

	opportunity_id	agent_id	product_id	account_id	deal_stage	engage_date	close_date	close_value
9	HAXMC4IX	30	4	NaN	Engaging	2016-11-03	NaN	NaN
25	UP409DSB	34	4	26	Engaging	2016-11-10	NaN	NaN
42	EG7OFLFR	32	1	NaN	Engaging	2016-11-14	NaN	NaN
44	OLVI7L8M	19	2	NaN	Engaging	2016-11-16	NaN	NaN
56	F5U1ACDD	32	6	NaN	Engaging	2016-11-19	NaN	NaN
...	...	...	...	...	...	...	...	...
8795	9MIWFW5J	3	4	NaN	Prospecting	NaN	NaN	NaN
8796	6SLKZ8FI	3	4	NaN	Prospecting	NaN	NaN	NaN
8797	LIB4KUZJ	3	4	NaN	Prospecting	NaN	NaN	NaN
8798	18IUIUK0	3	4	NaN	Prospecting	NaN	NaN	NaN
8799	8I5ONXJX	3	4	NaN	Prospecting	NaN	NaN	NaN

2089 rows × 8 columns

```python
df_sales['deal_stage'].value_counts()
```

```
deal_stage
Won            4238
Lost           2473
Engaging       1589
Prospecting     500
Name: count, dtype: int64
```

```python
#Display any row contain 'deal_stage:Prospecting'
prospecting_rows = df_sales[df_sales['deal_stage'] == 'Prospecting']
prospecting_rows
```

	opportunity_id	agent_id	product_id	account_id	deal_stage	engage_date	close_date	close_value
8300	6CWZFOHJ	1	1	33	Prospecting	NaN	NaN	NaN
8301	3LCLVRVV	1	1	NaN	Prospecting	NaN	NaN	NaN
8302	YIU1B39V	1	1	NaN	Prospecting	NaN	NaN	NaN
8303	8E0VRCLW	1	1	NaN	Prospecting	NaN	NaN	NaN
8304	G99CS23F	1	1	NaN	Prospecting	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
8795	9MIWFW5J	3	4	NaN	Prospecting	NaN	NaN	NaN
8796	6SLKZ8FI	3	4	NaN	Prospecting	NaN	NaN	NaN
8797	LIB4KUZJ	3	4	NaN	Prospecting	NaN	NaN	NaN
8798	18IUIUK0	3	4	NaN	Prospecting	NaN	NaN	NaN
8799	8I5ONXJX	3	4	NaN	Prospecting	NaN	NaN	NaN

500 rows × 8 columns

```
# Display rows with [close_date] null values
close_date_rows = df_sales[df_sales['close_date'].isnull()]
close_date_rows
```

	opportunity_id	agent_id	product_id	account_id	deal_stage	engage_date	close_date	close_value
9	HAXMC4IX	30	4	NaN	Engaging	2016-11-03	NaN	NaN
25	UP409DSB	34	4	26	Engaging	2016-11-10	NaN	NaN
42	EG7OFLFR	32	1	NaN	Engaging	2016-11-14	NaN	NaN
44	OLVI7L8M	19	2	NaN	Engaging	2016-11-16	NaN	NaN
56	F5U1ACDD	32	6	NaN	Engaging	2016-11-19	NaN	NaN
...	...	...	...	...	...	...	...	...
8795	9MIWFW5J	3	4	NaN	Prospecting	1900-12-31	NaN	NaN
8796	6SLKZ8FI	3	4	NaN	Prospecting	1900-12-31	NaN	NaN
8797	LIB4KUZJ	3	4	NaN	Prospecting	1900-12-31	NaN	NaN
8798	18IUIUK0	3	4	NaN	Prospecting	1900-12-31	NaN	NaN
8799	8I5ONXJX	3	4	NaN	Prospecting	1900-12-31	NaN	NaN

2089 rows × 8 columns

```
accounts_rows = df_sales[df_sales['account_id'].isnull()]
accounts_rows
```

	opportunity_id	agent_id	product_id	account_id	deal_stage	engage_date	close_date	close_value
9	HAXMC4IX	30	4	NaN	Engaging	2016-11-03	1900-12-31	0.0
42	EG7OFLFR	32	1	NaN	Engaging	2016-11-14	1900-12-31	0.0
44	OLVI7L8M	19	2	NaN	Engaging	2016-11-16	1900-12-31	0.0
56	F5U1ACDD	32	6	NaN	Engaging	2016-11-19	1900-12-31	0.0
60	ZZY4516R	26	4	NaN	Engaging	2016-11-20	1900-12-31	0.0
...	...	...	...	...	...	...	...	...
8795	9MIWFW5J	3	4	NaN	Prospecting	1900-12-31	1900-12-31	0.0
8796	6SLKZ8FI	3	4	NaN	Prospecting	1900-12-31	1900-12-31	0.0
8797	LIB4KUZJ	3	4	NaN	Prospecting	1900-12-31	1900-12-31	0.0
8798	18IUIUK0	3	4	NaN	Prospecting	1900-12-31	1900-12-31	0.0
8799	8I5ONXJX	3	4	NaN	Prospecting	1900-12-31	1900-12-31	0.0

1425 rows × 8 columns

```python
#Display any row contain 'deal_stage:Engaging'
Engaging_rows = df_sales[df_sales['deal_stage'] == 'Engaging']
Engaging_rows
```

	opportunity_id	agent_id	product_id	account_id	deal_stage	engage_date	close_date	close_value
9	HAXMC4IX	30	4	NaN	Engaging	2016-11-03	NaN	NaN
25	UP409DSB	34	4	26	Engaging	2016-11-10	NaN	NaN
42	EG7OFLFR	32	1	NaN	Engaging	2016-11-14	NaN	NaN
44	OLVI7L8M	19	2	NaN	Engaging	2016-11-16	NaN	NaN
56	F5U1ACDD	32	6	NaN	Engaging	2016-11-19	NaN	NaN
...	...	...	...	...	...	...	...	...
8277	NGTVHTFH	22	2	NaN	Engaging	2017-12-19	NaN	NaN
8283	HB740BLB	26	3	63	Engaging	2017-12-20	NaN	NaN
8285	HCQK8NQ8	16	6	NaN	Engaging	2017-12-20	NaN	NaN
8286	RDHTQLNI	19	5	NaN	Engaging	2017-12-21	NaN	NaN
8290	DB801ISB	28	3	NaN	Engaging	2017-12-22	NaN	NaN

1589 rows × 8 columns

3. **Cleaning Data:** Null values in 'engage_date' and 'close_date' will be updated to 31-12-1900. Null values in 'close_value' and 'account_id' will be changed to 0.

```python
#Filling rows is null of engage_date
df_sales['engage_date'] = df_sales['engage_date'].fillna(pd.to_datetime('1900-12-31').date())
```

```python
df_sales['close_date'] = df_sales['close_date'].fillna(pd.to_datetime('1900-12-31').date())
df_sales['close_value'] = df_sales['close_value'].fillna(0)
```

```python
df_sales['account_id'] = df_sales['account_id'].fillna(0)
```

4. **Checking a Dimension tables:** The code initializes an empty dictionary `df_dict` to store Data Frames read from CSV files listed in `dataframes`. It iterates over each file, extracting the base filename (without the `.csv` extension) to create a corresponding key. Each CSV file is loaded into a pandas Data Frame using `pd.read_csv()`, which is then stored in `df_dict`. The Data Frame is displayed interactively using `display()` in a Jupyter environment, and the name of the displayed DataFrame is printed. Finally, global variables (like `df_accounts`, `df_products`, and `df_sales_teams`) are dynamically created to reference the Data Frames stored in `df_dict`, allowing easy access for further analysis or manipulation.

```
# Dictionary to store DataFrames by their filenames (without .csv)
df_dict = {}

# Iterate over the file names, read each CSV into a DataFrame, store it in the dictionary, and display
for file in dataframes:
    # Create a name for each DataFrame from the file name (without .csv extension)
    df_name = file.split('.')[0]

    # Load the CSV into a DataFrame
    df_dict[df_name] = pd.read_csv(f"{modified_path}{file}")

    # Display the DataFrame in an interactive view (if in Jupyter)
    display(df_dict[df_name])

    # Print the name of the DataFrame after displaying it
    print(f"Displayed DataFrame: {df_name}\n")

    # Dynamically create variables (df_account, df_products, df_sales_teams) using globals()
    globals()[f'df_{df_name}'] = df_dict[df_name]

# Now you have variables: df_accounts, df_products, and df_sales_teams created dynamically
```

5. Continue data cleaning with the new data frames

```
df_accounts.isnull().sum()
```

```
id                  0
account             0
sector              0
year_established    0
revenue             0
employees           0
office_location     0
subsidiary_of      70
dtype: int64
```

```
# Calculate the percentage of missing values and format them
missing_values = (df_accounts.isnull().sum() / df_accounts.shape[0]) * 100  # same code  'df_account.isnull().sum()/( len(df))*100'
missing_values = missing_values.round(2)  # Round to 2 decimal places
print(missing_values.astype(str) + '%')   # Add percentage sign
```

```
id                   0.0%
account              0.0%
sector               0.0%
year_established     0.0%
revenue              0.0%
employees            0.0%
office_location      0.0%
subsidiary_of       82.35%
dtype: object
```

```
#subsidiary_of: Parent company, if applicable
df_accounts['subsidiary_of'] = df_accounts['subsidiary_of'].fillna('Independent')
```

```
df_products.isnull().sum()
```

```
id             0
product        0
series         0
sales_price    0
dtype: int64
```

```
df_sales_teams.isnull().sum()
```

```
id                0
sales_agent       0
manager           0
regional_office   0
dtype: int64
```

6. **Save the updated data**: The updated data was saved in the sales_teams.csv, products.csv, accounts.csv, and sales_pipeline.csv files, ensuring it is properly prepared and optimized for the next phases of the project.

```
df_sales.to_csv(f"{modified_path}sales_pipeline.csv", index=False)
df_accounts.to_csv(f"{modified_path}accounts.csv", index=False)
df_products.to_csv(f"{modified_path}products.csv", index=False)
df_sales_teams.to_csv(f"{modified_path}sales_teams.csv", index=False)
```

Data Visualization

1. Identifying Trends

This analysis will look at how sales performance changes over time, typically grouped by quarter or year.

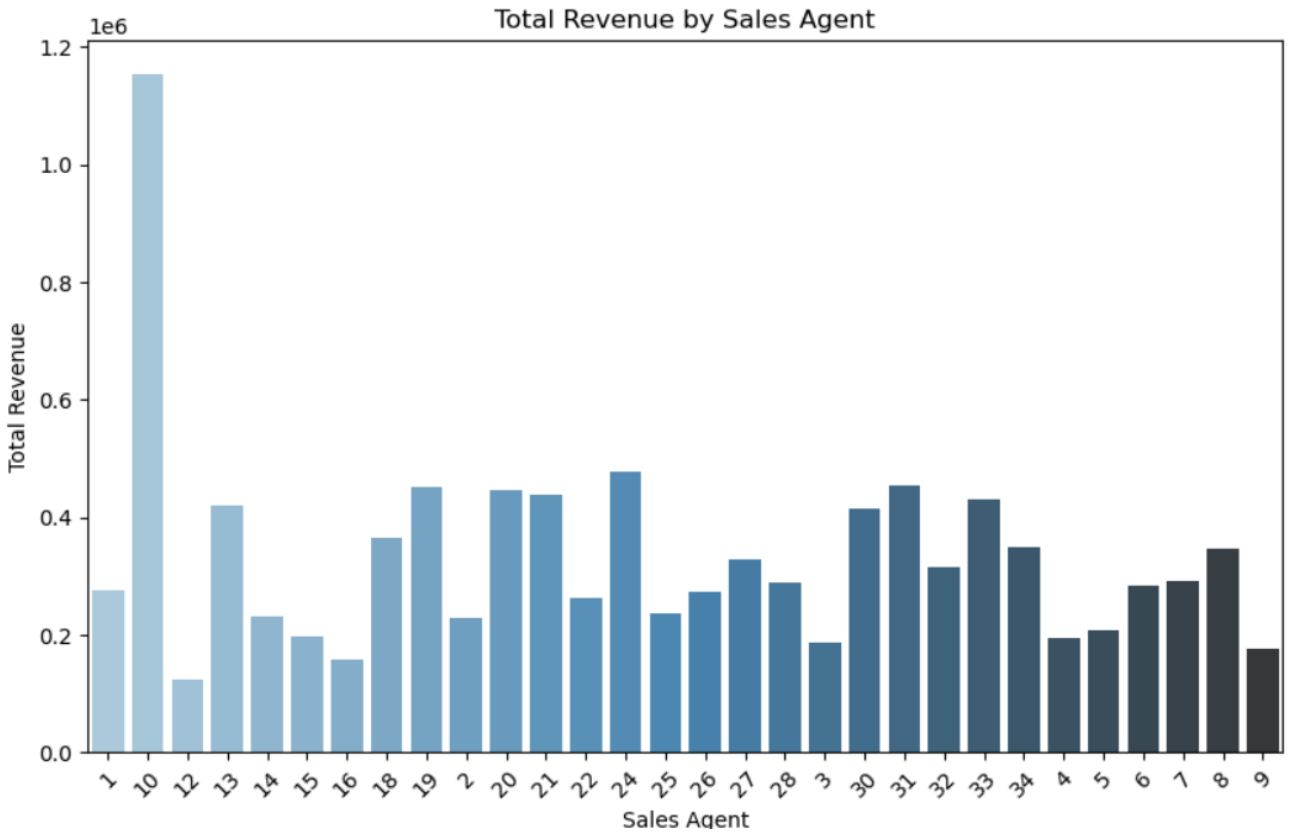
```
# Convert 'close_date' to datetime
df_sales['close_date'] = pd.to_datetime(df_sales['close_date'])

# Extract year and quarter
df_sales['year'] = df_sales['close_date'].dt.year
df_sales['quarter'] = df_sales['close_date'].dt.quarter

# Group by year and quarter to analyze trends
sales_trends = df_sales.groupby(['year', 'quarter']).agg(
    total_revenue=('close_value', 'sum'),
    total_deals=('opportunity_id', 'count'),
    win_rate=('deal_stage', lambda x: (x == 'Won').mean() * 100)
).reset_index()

# Format the 'win_rate' column to 2 decimal places and add a percentage sign
sales_trends['win_rate'] = sales_trends['win_rate'].apply(lambda x: f"{x:.2f}%")

# Display the updated DataFrame
print(sales_trends)
```

```
   year  quarter  total_revenue  total_deals win_rate
0  1900        4            0.0         2089    0.00%
1  2017        1      1134672.0          647   82.07%
2  2017        2      3086111.0         2032   61.71%
3  2017        3      2982255.0         2047   61.41%
4  2017        4      2802496.0         1985   60.25%
```

2. Sales Team Performance Visualization:

A bar plot showing the total revenue generated by each sales agent or team.

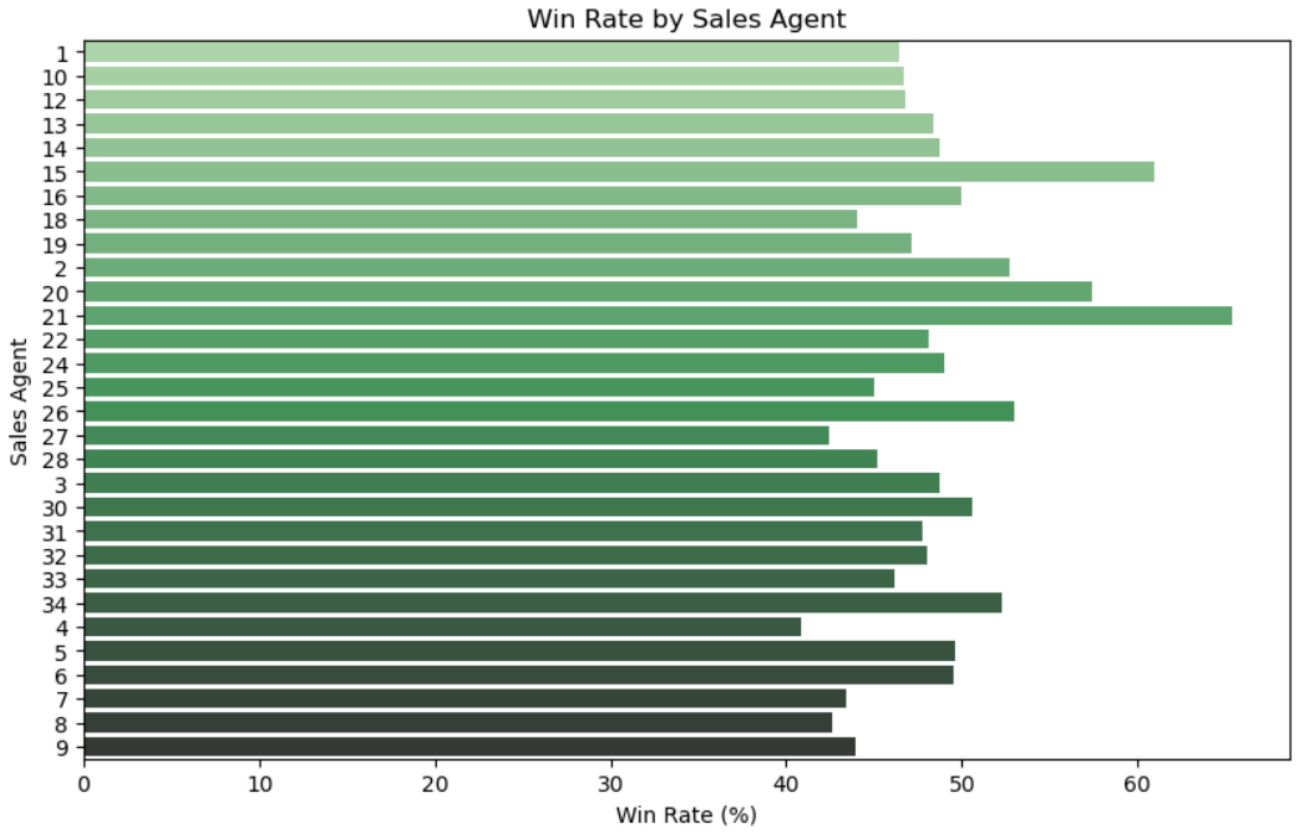
```
# Bar plot: Total revenue by sales agent
plt.figure(figsize=(10, 6))
sns.barplot(x='agent_id', y='total_revenue', data=sales_team_performance, hue='agent_id', palette='Blues_d', legend=False)
plt.title('Total Revenue by Sales Agent')
plt.xlabel('Sales Agent')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45)
plt.show()
```



3. Sales Agent Productivity Visualization:

A horizontal bar plot showing the win rate for each sales agent.

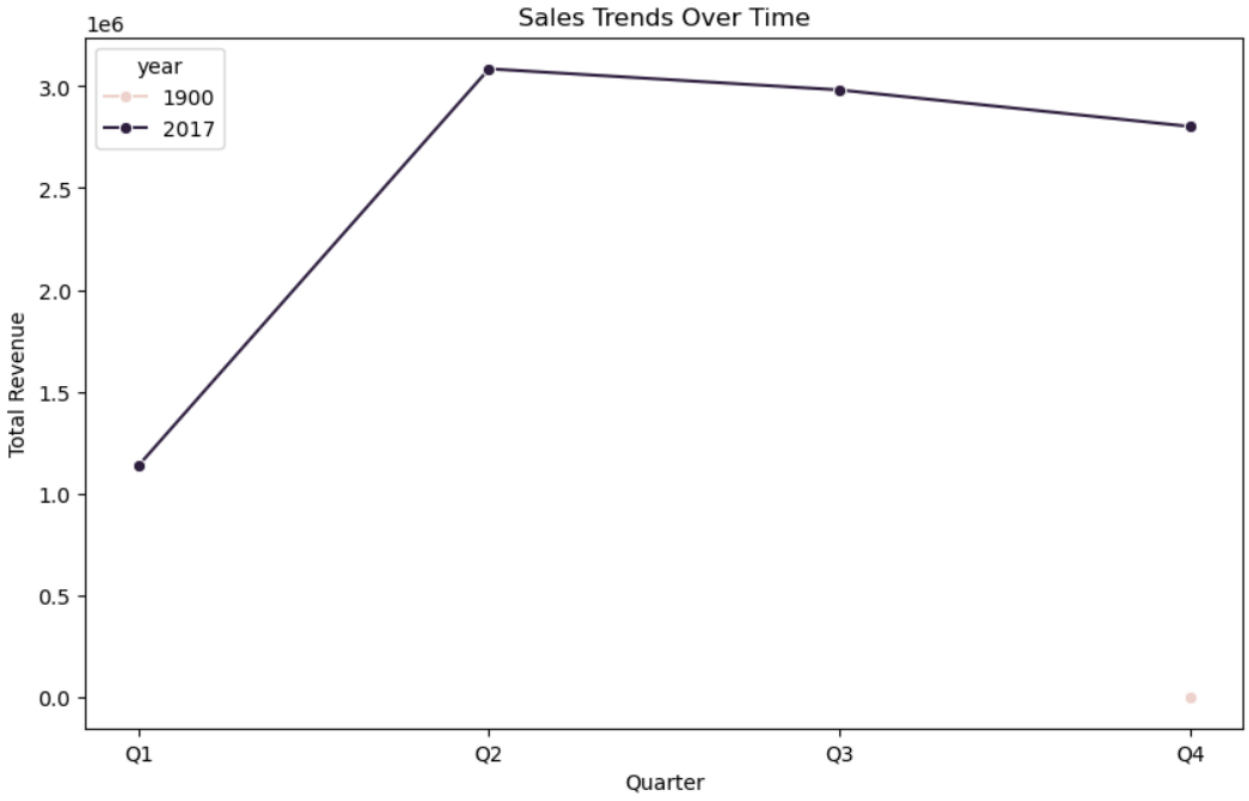
```
# Horizontal bar plot: Win rate by sales agent
plt.figure(figsize=(10, 6))
sns.barplot(x='win_rate', y='agent_id', data=sales_team_performance, hue='agent_id', palette='Greens_d', legend=False)
plt.title('Win Rate by Sales Agent')
plt.xlabel('Win Rate (%)')
plt.ylabel('Sales Agent')
plt.show()
```



4. Sales Trends Over Time:

A line plot showing total revenue over time (grouped by year and quarter).

```
# Line plot: Sales trends over time
plt.figure(figsize=(10, 6))
sns.lineplot(x='quarter', y='total_revenue', hue='year', data=sales_trends, marker='o')
plt.title('Sales Trends Over Time')
plt.xlabel('Quarter')
plt.ylabel('Total Revenue')
plt.xticks([1, 2, 3, 4], ['Q1', 'Q2', 'Q3', 'Q4'])
plt.show()
```



Data Modelling

Star Schema Structure

In this project, I used a star schema structure to organize the data for efficient querying and analysis.

Fact table

- sales_pipeline
    - *opportunity_id*: The primary key (PK) for the fact table, uniquely identifying each sales opportunity.
    - *account_id*: A foreign key (FK) linking to the accounts dimension table, identifying the company involved in the sales opportunity.
    - *agent_id*: A foreign key (FK) linking to the sales_teams dimension table, identifying the sales agent responsible for the opportunity.
    - *product_id*: A foreign key (FK) linking to the products dimension table, identifying the product involved in the sales opportunity.

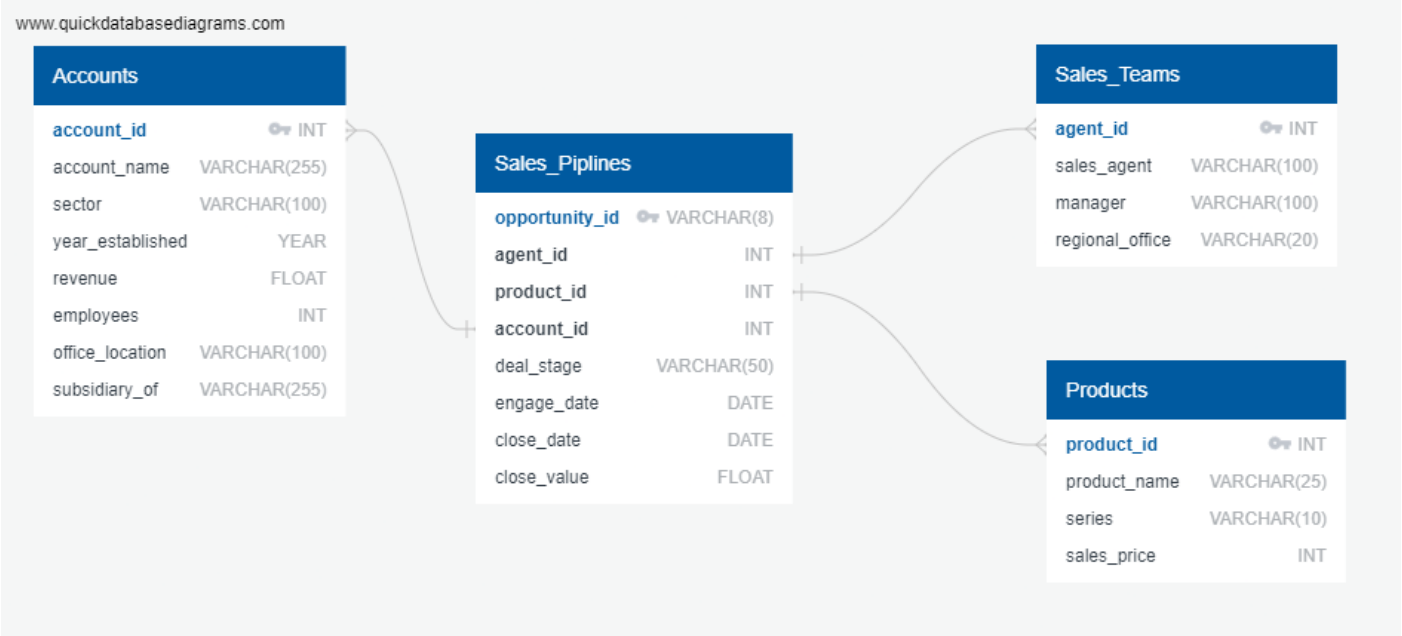
Dimension tables

Each dimension table has a many-to-one relationship with the fact table, meaning multiple records in the fact table can relate to a single record in a dimension table.

- accounts:
    - *account_id*: The primary key (PK) that uniquely identifies each company.
- sales_teams:
    - *agent_id*: The primary key (PK) that uniquely identifies each sales agent or team member.
- products:
    - *product_id*: The primary key (PK) that uniquely identifies each product.

Entity-Relationship Diagram

The following Entity-Relationship Diagram (ERD) visually represents the star schema structure, providing a clear overview of the database architecture and the relationships between entities



Database and Tables Creation

The database was designed and implemented in Microsoft SQL Server, with each table (fact and dimension) created based on the defined schema. The creation process involved:

1. **Create 'sales_opportunity' database**

```
CREATE DATABASE sales_opportunity;
```

2. **Create 'sales_teams' table**

```
CREATE TABLE [sales_teams] (
    [agent_id] INT  NOT NULL ,
    [sales_agent] VARCHAR(100)  NOT NULL ,
    [manager] VARCHAR(100) ,
    [regional_office] VARCHAR(20) ,
    CONSTRAINT [PK_sales_teams] PRIMARY KEY CLUSTERED (
        [agent_id] ASC
    )
)
```

3. **Create 'products' table**

```
CREATE TABLE [products] (
    [product_id] INT  NOT NULL ,
    [product_name] VARCHAR(25)  NOT NULL ,
    [series] VARCHAR(10)  NOT NULL ,
    [sales_price] INT  NOT NULL ,
    CONSTRAINT [PK_products] PRIMARY KEY CLUSTERED (
        [product_id] ASC
    )
)
```

4. Create 'accounts' table

```
CREATE TABLE [accounts] (
    [account_id] INT  NOT NULL ,
    [account_name] VARCHAR(255)  NOT NULL ,
    [sector] VARCHAR(100)  NOT NULL ,
    [year_established] INT ,
    [revenue] FLOAT ,
    [employees] INT ,
    [office_location] VARCHAR(100) ,
    [subsidiary_of] VARCHAR(255) ,
    CONSTRAINT [PK_accounts] PRIMARY KEY CLUSTERED (
        [account_id] ASC
    )
)
```

5. Create 'sales' table

```
CREATE TABLE [sales] (
    [opportunity_id] VARCHAR(8)  NOT NULL ,
    [agent_id] INT ,
    [product_id] INT ,
    [account_id] INT ,
    [deal_stage] VARCHAR(50) ,
    [engage_date] DATE ,
    [close_date] DATE ,
    [close_value] FLOAT ,
    CONSTRAINT [PK_sales] PRIMARY KEY CLUSTERED (
        [opportunity_id] ASC
    )
)
```

6. Establishing Foreign Key Constraints for Data Integrity

```
ALTER TABLE [sales] WITH CHECK ADD CONSTRAINT [FK_sales_agent_id] FOREIGN KEY([agent_id])
REFERENCES [sales_teams] ([agent_id])

ALTER TABLE [sales] CHECK CONSTRAINT [FK_sales_agent_id]

ALTER TABLE [sales] WITH CHECK ADD CONSTRAINT [FK_sales_product_id] FOREIGN KEY([product_id])
REFERENCES [products] ([product_id])

ALTER TABLE [sales] CHECK CONSTRAINT [FK_sales_product_id]

ALTER TABLE [sales] WITH CHECK ADD CONSTRAINT [FK_sales_account_id] FOREIGN KEY([account_id])
REFERENCES [accounts] ([account_id])

ALTER TABLE [sales] CHECK CONSTRAINT [FK_sales_account_id]
```

7. Loading data

```
-- loading data into the sales_teams table - file: sales_teams.csv
BULK INSERT [dbo].[sales_teams]
FROM 'path\modified_files\sales_teams.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

-- loading data into the products table - file: products.csv
BULK INSERT[dbo].[products]
FROM 'path\modified_files\products.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

-- loading data into the accounts table - file: accounts.csv
BULK INSERT [dbo].[accounts]
FROM 'path\modified_files\accounts.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);

-- loading data into the sales_pipeline table - file: sales_pipeline.csv
BULK INSERT [dbo].[sales]
FROM 'path\modified_files\sales_pipeline.csv'
WITH (
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    FIRSTROW = 2
);
```

Data Analysis

1. Assessing Sales Team Performance

- **Key Metrics:** Number of deals won, total revenue generated, average deal size, and win rate (deals won vs. total deals).
- **Approach:**
  - Group sales opportunities by sales agents and managers to analyze their performance.
  - Create a leaderboard for sales teams based on total revenue or deal win rate.
  - Compare regional office performance by aggregating deals and revenue per region.

```sql
--Number of deals won, total revenue generated, average deal size, and win rate (deals won vs. total deals)
SELECT
    manager,
    SUM(close_value) AS total_revenue,
    COUNT(opportunity_id) AS total_deals,
    SUM(CASE WHEN deal_stage = 'Won' THEN 1 ELSE 0 END) AS deals_won,
    CASE
        WHEN COUNT(opportunity_id) = 0 THEN '0.00%'  -- Handle division by zero
        ELSE FORMAT(SUM(CASE WHEN deal_stage = 'Won' THEN 1 ELSE 0 END) * 100.0 / COUNT(opportunity_id), 'N2') + '%'
    END AS win_rate
FROM
    sales
JOIN
    sales_teams ON sales.agent_id = sales_teams.agent_id
WHERE
    close_date IS NOT NULL
GROUP BY
    manager
ORDER BY
    total_revenue DESC;
```

Results | Messages

	manager	total_revenue	total_deals	deals_won	win_rate
1	Melvin Marxen	2251930	1929	882	45.72%
2	Summer Sewald	1964750	1701	828	48.68%
3	Rocco Neubert	1960545	1327	691	52.07%
4	Celia Rouche	1603897	1296	610	47.07%
5	Cara Losch	1130049	964	480	49.79%
6	Dustin Brinkmann	1094363	1583	747	47.19%

2. Evaluating Sales Agent Productivity

- **Key Metrics:** Number of deals handled, revenue per agent, and individual win rates.
- **Approach:**
  - Rank agents based on the number of deals they handled and total revenue generated.
  - Identify outliers—agents with unusually high or low productivity.
  - You can track their performance over time to see if trends appear, e.g., improving or declining performance.

```sql
SELECT
    s.agent_id,
    st.sales_agent,  -- Select the sales agent name
    COUNT(s.opportunity_id) AS total_deals,
    SUM(s.close_value) AS total_revenue,
    CASE
        WHEN COUNT(s.opportunity_id) = 0 THEN '0.00%'  -- Handle division by zero
        ELSE FORMAT(SUM(CASE WHEN s.deal_stage = 'Won' THEN 1 ELSE 0 END) * 100.0 / COUNT(s.opportunity_id), 'N2') + '%'
    END AS win_rate
FROM
    sales s
JOIN
    sales_teams st ON st.agent_id = s.agent_id  -- Join with sales_teams
GROUP BY
    st.sales_agent, s.agent_id  -- Group by sales agent and agent_id
ORDER BY
    total_revenue DESC;
```

Results | Messages

	agent_id	sales_agent	total_deals	total_revenue	win_rate
1	10	Darcel Schlecht	747	1153214	46.72%
2	24	Vicki Laflamme	451	478396	49.00%
3	31	Kary Hendrixson	438	454298	47.72%
4	19	Cassey Cress	346	450489	47.11%
5	20	Donn Cantrell	275	445860	57.45%
6	21	Reed Clapper	237	438336	65.40%
7	33	Zane Levy	349	430068	46.13%
8	13	Corliss Cosme	310	421036	48.39%
9	30	James Ascencio	267	413533	50.56%
10	18	Daniell Hammack	259	364229	44.02%
11	34	Maureen Marcano	285	350395	52.28%
12	8	Gladys Colclough	317	345674	42.59%
13	27	Markita Hansen	306	328792	42.48%
14	32	Kami Bicknell	362	316456	48.07%

3. Identifying Trends

- **Key Metrics:** Quarter-over-quarter (QoQ) growth in revenue, number of opportunities, and win rates.
- **Approach:**
  - Group deals by quarter or month and track performance metrics over time.
  - You can visualize the trends using a line or bar chart.

```sql
SELECT
    CASE
        WHEN MONTH(close_date) IN (1, 2, 3) THEN 1
        WHEN MONTH(close_date) IN (4, 5, 6) THEN 2
        WHEN MONTH(close_date) IN (7, 8, 9) THEN 3
        WHEN MONTH(close_date) IN (10, 11, 12) THEN 4
    END AS quarter,
    YEAR(close_date) AS year,
    SUM(close_value) AS total_revenue,
    COUNT(opportunity_id) AS total_deals,
    FORMAT(SUM(CASE WHEN deal_stage = 'Won' THEN 1 ELSE 0 END) * 100.0 / NULLIF(COUNT(opportunity_id), 0), 'N2') + '%' AS win_rate
FROM
    sales
WHERE
    YEAR(close_date) = 2017  -- Filter for the year 2017
GROUP BY
    YEAR(close_date),
    CASE
        WHEN MONTH(close_date) IN (1, 2, 3) THEN 1
        WHEN MONTH(close_date) IN (4, 5, 6) THEN 2
        WHEN MONTH(close_date) IN (7, 8, 9) THEN 3
        WHEN MONTH(close_date) IN (10, 11, 12) THEN 4
    END
ORDER BY
    YEAR(close_date),
    quarter;
```

Results | Messages

	quarter	year	total_revenue	total_deals	win_rate
1	1	2017	1134672	647	82.07%
2	2	2017	3086111	2032	61.71%
3	3	2017	2982255	2047	61.41%
4	4	2017	2802496	1985	60.25%

4. Analyzing Product Success Rates

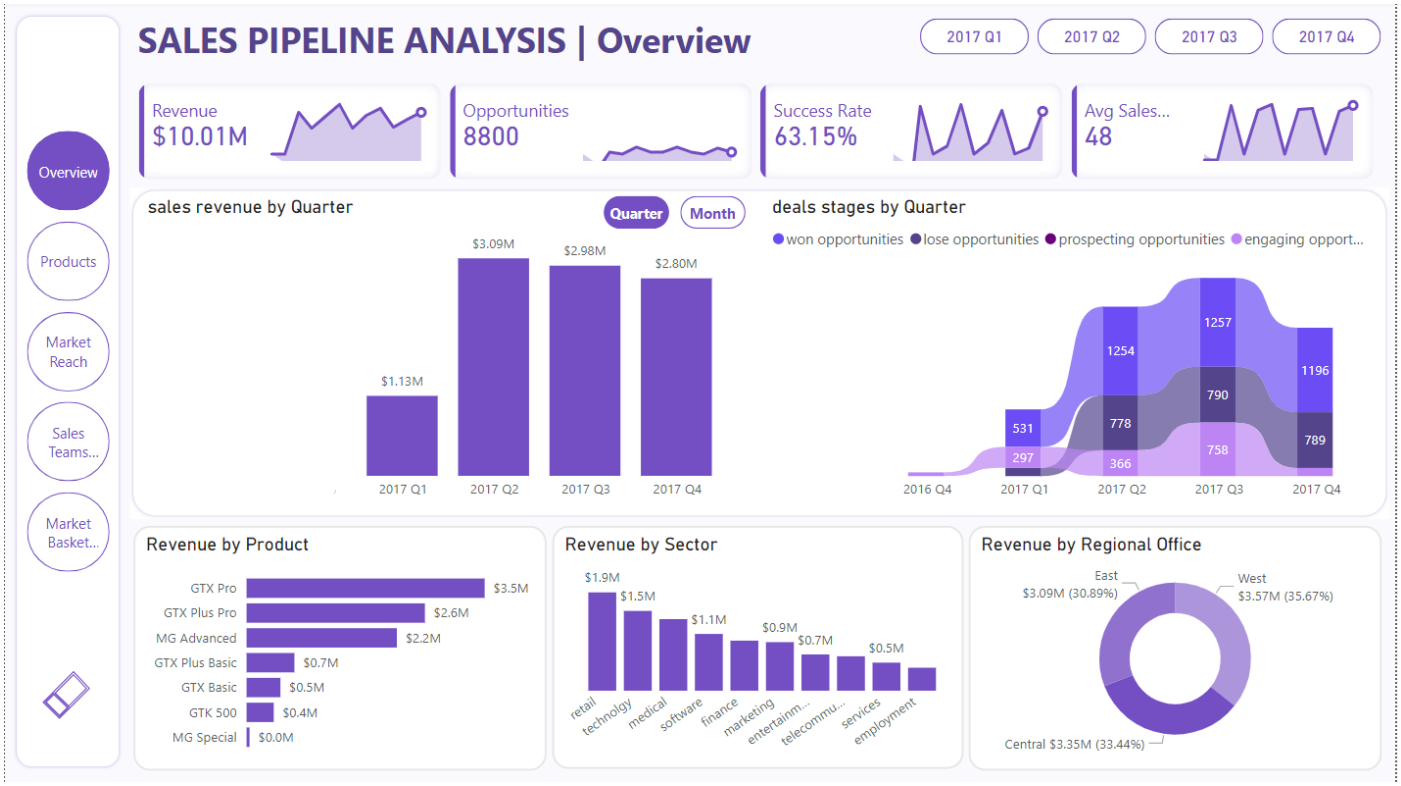
- **Key Metrics:** Win rates by product and product revenue.
- **Approach:**
  - Calculate win rates for each product by comparing the number of won deals to total deals.
  - Calculate win rates for each product by comparing the number of won deals to total deals.

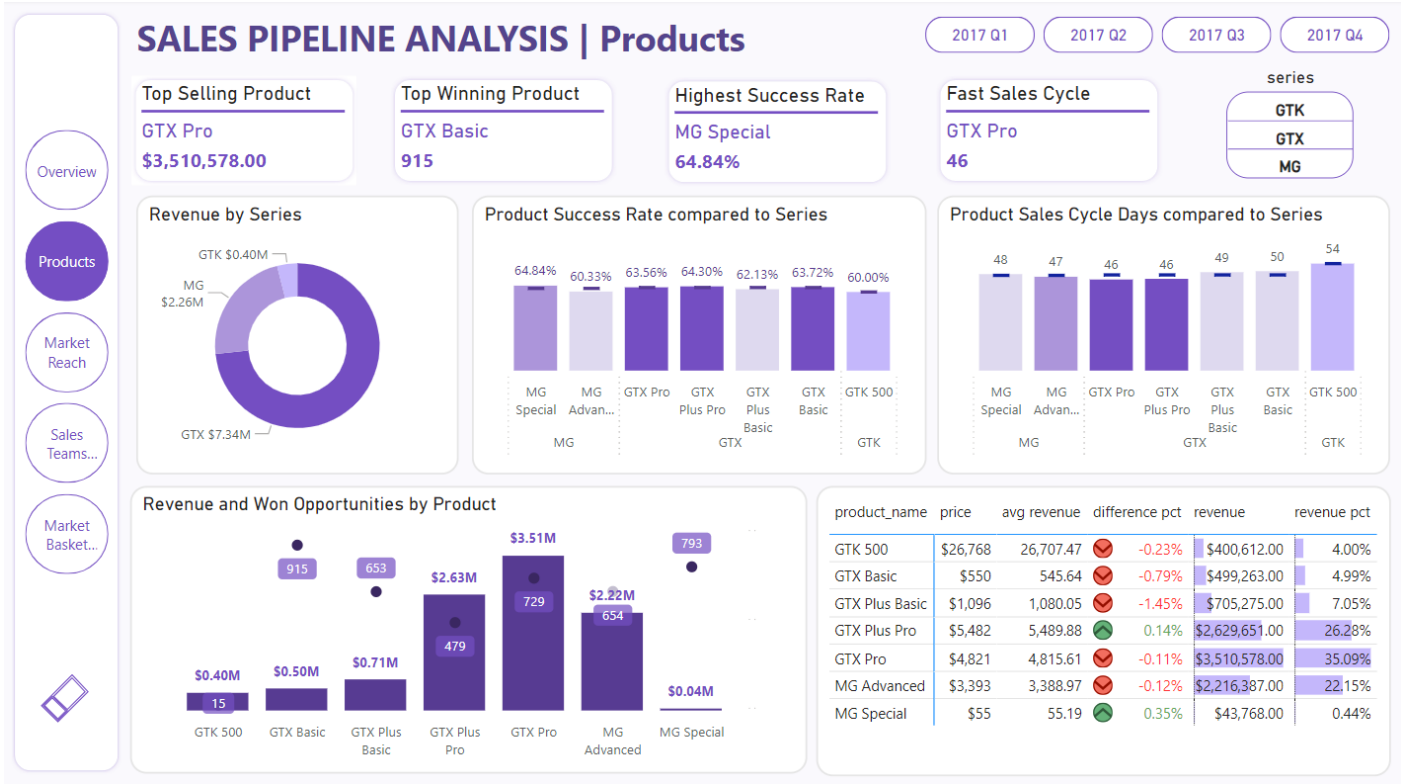
```sql
SELECT
    s.product_id,
    p.product_name AS product_name,  -- Select product name from the products table
    FORMAT(SUM(CASE WHEN s.deal_stage = 'Won' THEN 1 ELSE 0 END) * 100.0 / NULLIF(COUNT(s.opportunity_id), 0), 'N2') + '%' AS win_rate,
    SUM(s.close_value) AS total_revenue
FROM
    sales s
JOIN
    products p ON s.product_id = p.product_id  -- Join with the products table
GROUP BY
    p.product_name, s.product_id  -- Group by product name and product_id
ORDER BY
    total_revenue DESC;  -- Order by total revenue in descending order
```

Results | Messages

	product_id	product_name	win_rate	total_revenue
1	2	GTX Pro	49.26%	3510578
2	5	GTX Plus Pro	49.48%	2629651
3	4	MG Advanced	46.32%	2216387
4	6	GTX Plus Basic	47.22%	705275
5	1	GTX Basic	49.04%	499263
6	7	GTK 500	37.50%	400612
7	3	MG Special	48.03%	43768

Power BI





product_name	price	avg revenue	difference pct	revenue	revenue pct
GTK 500	$26,768	26,707.47	-0.23%	$400,612.00	4.00%
GTX Basic	$550	545.64	-0.79%	$499,263.00	4.99%
GTX Plus Basic	$1,096	1,080.05	-1.45%	$705,275.00	7.05%
GTX Plus Pro	$5,482	5,489.88	0.14%	$2,629,651.00	26.28%
GTX Pro	$4,821	4,815.61	-0.11%	$3,510,578.00	35.09%
MG Advanced	$3,393	3,388.97	-0.12%	$2,216,387.00	22.15%
MG Special	$55	55.19	0.35%	$43,768.00	0.44%

Power BI

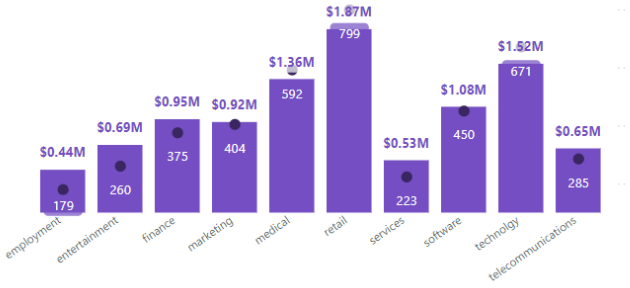
SALES PIPELINE ANALYSIS | Market Reach

2017 Q1 | 2017 Q2 | 2017 Q3 | 2017 Q4

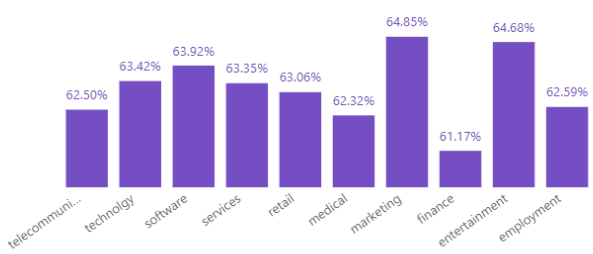
Overview
Products
Market Reach
Sales Teams...
Market Basket...

15 Countries	10 Sectors	85 Accounts
Top 3 Countries	Top 3 Sectors	Top 3 Accounts
United States	**technolgy**	**Konex**
$8,426,955.00	$1,515,487.00	$269,245.00
Korea	**retail**	**Kan-code**
$194,957.00	$1,867,528.00	$341,455.00
Jordan	**medical**	**Condax**
$163,339.00	$1,359,595.00	$206,410.00

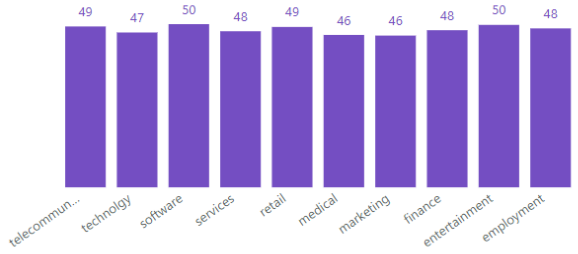
Revenue and Won Opportunities by Sector



Success Rate by sector



sales cycle duration by sector



SALES PIPELINE ANALYSIS | Sales Teams

2017 Q1 | 2017 Q2 | 2017 Q3 | 2017 Q4

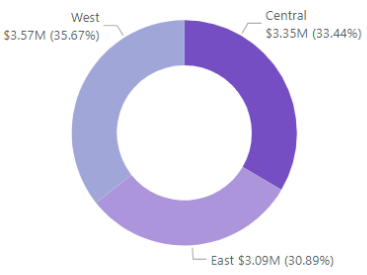
Overview
Products
Market Reach
Sales Teams...
Market Basket...

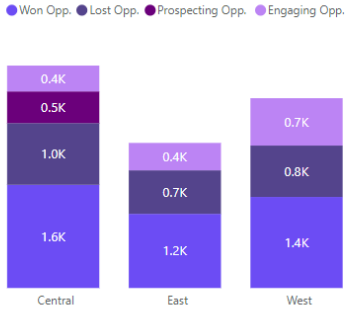
agents without deals
- Carl Lin
- Carol Thompson
- Elizabeth Anderson
- Mei-Mei Johns
- Natalya Ivanova

Top 3 Managers
- Melvin Marxen
- $2,251,930.00
- Summer Sewald
- $1,964,750.00
- Rocco Neubert
- $1,960,545.00

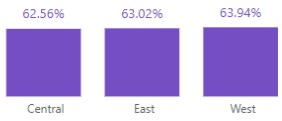
manager	sales revenue	won deals	contribution revenue agent	contribution revenue manager
Melvin Marxen	$2,251,930.00	882	22.51%	100.00%
Summer Sewald	$1,964,750.00	828	19.64%	100.00%
Rocco Neubert	$1,960,545.00	691	19.59%	100.00%
Celia Rouche	$1,603,897.00	610	16.03%	100.00%
Cara Losch	$1,130,049.00	480	11.29%	100.00%
Dustin Brinkmann	$1,094,363.00	747	10.94%	100.00%
Total	**$10,005,534.00**	**4238**	**100.00%**	**100.00%**

sales agent | manager | regional office

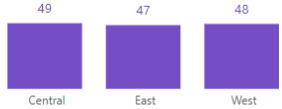




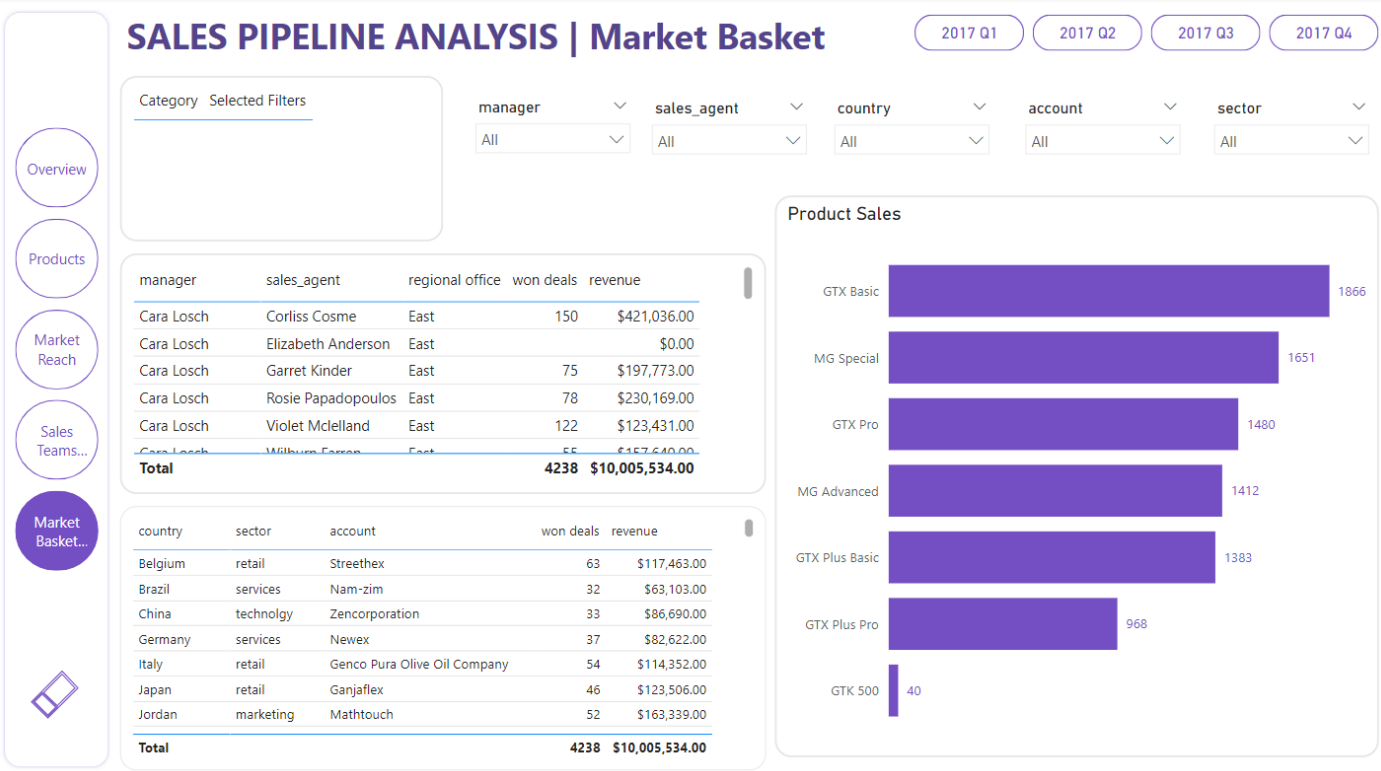
Success Rate vs. Average



Sales Cycle Days vs. Average



Power BI



Thanks

☺