TEAM 4

DATABASE FOR COMMERCIAL MARKETS

PROJECT3 - DATA ANALYTICS BOOTCAMP

ALBERTO SEVY LEVY
RUBÉN GARCÍA ANDRÉS
RICARDO GÓMEZ DÁVALOS
GERARDO OCHOA HERRERA
JESÚS ALBERTO SÁENZ BARBA
PATRICIO AUGUSTO PATIÑO ESPAÑA

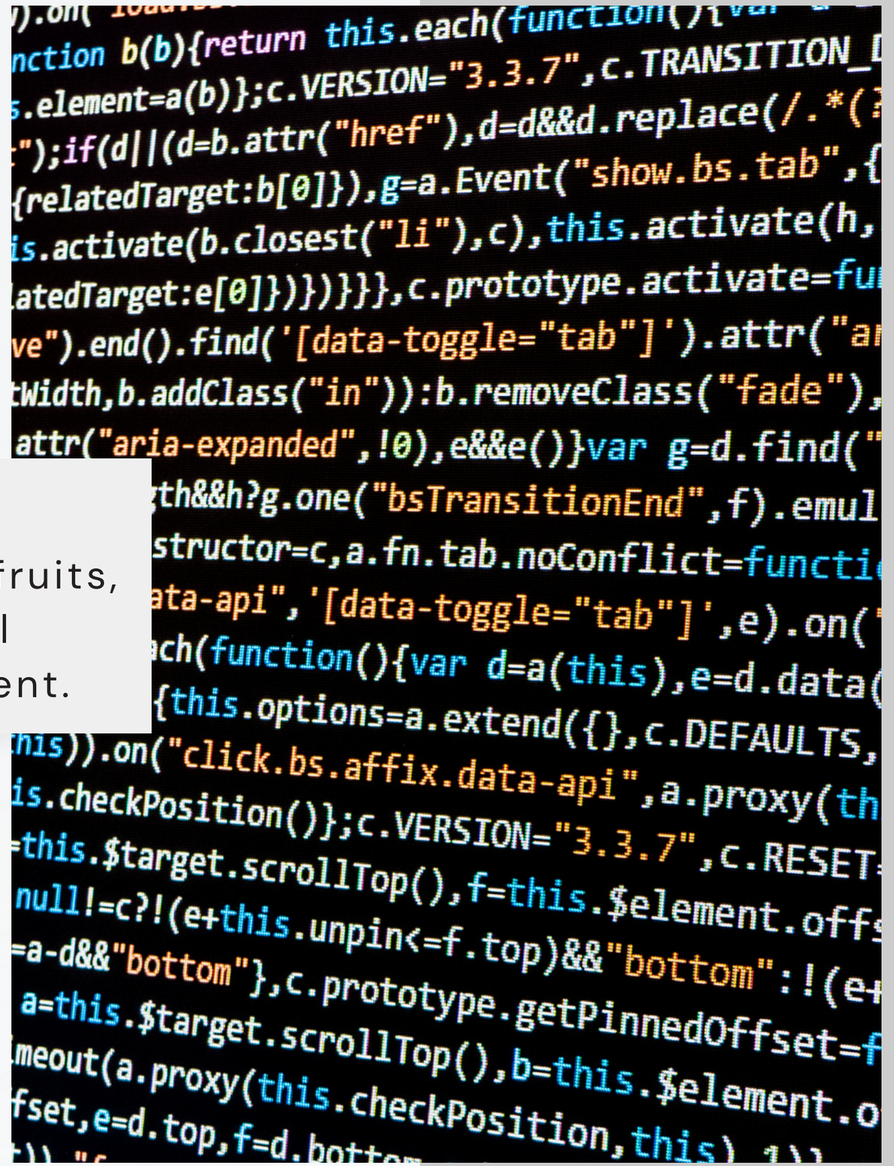
CONTENT

PROJECT THEME

Our interest began with the potential outcome you can get with the correct manipulation of a database, specially the ones that contain market behavior; which is how we got to out database.

OUR DATABASE

We selected our database from a company dedicated to the commercialization of dry fruits, it is a simulation of the operation of several years; constructed from the sales department.



GOALS

Objective n° 1

Clean the database and correct inconsistencies in order to present data in a better shape

Objective n° 2

Place correct information inside the columns that create the database, as well as the correct placement of the columns

Objective n° 3

Create tables that show the total sales from different stores, locations, zones, groups and subgroups

CODING APPROACH

Libraries

```
import pandas as pd
import csv
import numpy as np
from fuzzywuzzy import process
```

- The project started with three libraries we learned in class and finally we added one we did not see during the course.

- Our additional library will help us match the incorrect names from the columns that we need, in order to put the correct names for better visualization and interpretation

Fuzzywuzzy

DATA TECHNIQUES

```
items = "items.csv"

items_data = pd.read_csv(items, low_memory=False)

items_data.head()
```

	item	description	item_type	des_i_type	item_group	unit_measure
0	Item_1	SCRAP BAGS	1605	[M] Comp Packaging	Grocery	PZ
1	Item_2	APIO C/6KG	9001	Vegetables	Fruits and vegetables	KG
2	Item_3	PURPLE CABBAGE 2KG	9001	Vegetables	Fruits and vegetables	KG
3	Item_4	other products	1906	No	Grocery	KG
4	Item_5	THE NIGHTINGALE T-SHIRTS	1904	[M] Uniforms	Grocery	PZ

	period_s	item	quantity	sales	store	customer
0	202001	Item_1804	-2.0	140.0	StoreC	Customer1
1	202001	Item_2052	-20.0	260.0	StoreC	Customer1
2	202001	Item_2329	-1.0	275.0	StoreC	Customer1
3	202001	Item_2347	-1.0	30.0	StoreC	Customer1
4	202001	Item_2365	-1.0	450.0	StoreC	Customer1

```
sales = "sales.csv"

sales_data = pd.read_csv(sales, low_memory=False)

sales_data.head()
```

```
customer = "customer.csv"

customer_data = pd.read_csv(customer, low_memory=False)

customer_data.head()
```

	Customer	Class	Kcustomer	Location
0	Customer1	Class01	Local	CDMX
1	Customer2	Class01	Foraneo	HIDALGO
2	Customer3	Class01	Local	CDMX
3	Customer4	Class01	Foraneo	ESTADO DE MEXICO
4	Customer5	Class01	Foraneo	Queretaro

We began the process by placing our information in tables
in which we can transform and clean the data.

DATA TECHNIQUES

```python
# List of correct Mexican state names and mappings for common incorrect variations
correct_states = {
    "CDMX": "CIUDAD DE MEXICO",
    "QRO.": "QUERETARO",
    "CDMX.": "CIUDAD DE MEXICO",
    "EDMX": "CIUDAD DE MEXICO",
    "MX": "CIUDAD DE MEXICO",
    "D. F.": "CIUDAD DE MEXICO",
    "DF": "CIUDAD DE MEXICO",
    "D, F.": "CIUDAD DE MEXICO",
    "EDO Q. ROO": "QUINTANA ROO",
    "HGO": "HIDALGO",
    "HGO.": "HIDALGO",
    "SN LUIS POTOSI": "SAN LUIS POTOSI",
    "EDO MEX": "ESTADO DE MEXICO",
    "EDO VER": "VERACRUZ",
    "EDO HGO": "ESTADO DE MEXICO",
    "EDO GRO": "GUERRERO",
    "LA PAZ": "BAJA CALIFORNIA SUR",
    "HIDALGO": "HIDALGO",
    "ESTADO DE MEXICO": "ESTADO DE MEXICO",
    "QUERETARO": "QUERETARO",
    "CANCUN QUINTANAROO": "QUINTANA ROO",  # Correcting this entry
    "QUINTANAROO": "QUINTANA ROO",         # Another variation of "QUINTANA ROO"
    "VERACRUZ DE IGNACIO DE LA LLAVE": "VERACRUZ",  # Correcting Veracruz name
    # Additional mappings can be added here as needed
}

# Load the CSV file from the current working directory
df = pd.read_csv('customer.csv')

# Standardize the "Location" column to uppercase
df['Location'] = df['Location'].str.upper()

# Replace non-standard or incorrect names with correct ones using the updated mapping
df['Location'] = df['Location'].replace(correct_states)

df["Location"].fillna("CIUDAD DE MEXICO", inplace=True)
df.head()
```

	Customer	Class	Kcustomer	Location
0	Customer1	Class01	Local	CIUDAD DE MEXICO
1	Customer2	Class01	Foraneo	HIDALGO
2	Customer3	Class01	Local	CIUDAD DE MEXICO
3	Customer4	Class01	Foraneo	ESTADO DE MEXICO
4	Customer5	Class01	Foraneo	QUERETARO

We developed a dictionary with the correct names of the Mexican states.

The following code is for the correct implementation of the actual names and for ordering the data.

DATA TECHNIQUES

```python
# List of valid Mexican state names
valid_states = [
    "CIUDAD DE MEXICO", "HIDALGO", "ESTADO DE MEXICO", "QUERETARO", "QUINTANA ROO",
    "VERACRUZ", "YUCATAN", "CHIAPAS", "JALISCO", "GUANAJUATO", "PUEBLA",
    # Add all other valid Mexican state names here



]

# Function to match incorrect names to valid state names using fuzzy matching
def match_state(state):
    if isinstance(state, str):  # Only apply fuzzy matching if the value is a string
        match, score = process.extractOne(state, valid_states)
        if score >= 80:  # Set a threshold score for matching (e.g., 80%)
            return match
    return state  # If not a string or no good match, return the original value

# Apply fuzzy matching to the "Location" column
df['Location'] = df['Location'].apply(match_state)
```

The additional library we implemented is used within this line of code. The process works with a match character within the columns that we needed, if the match is 80% or above; the function will place the value from the dictionary that is making the correct match.

DATA TECHNIQUES

```python
# Define the mapping of states to zones, all in uppercase
state_to_zone = {
    # South zone states
    'CHIAPAS': 'south',
    'OAXACA': 'south',
    'TABASCO': 'south',
    'YUCATAN': 'south',
    'CAMPECHE': 'south',
    'QUINTANA ROO': 'south',
    'VERACRUZ': 'south',

    # North zone states
    'NUEVO LEON': 'north',
    'SONORA': 'north',
    'CHIHUAHUA': 'north',
    'COAHUILA': 'north',
    'TAMAULIPAS': 'north',
    'BAJA CALIFORNIA SUR': 'north',
    'BAJA CALIFORNIA NORTE': 'north',
    'DURANGO': 'north',
    'SINALOA': 'north',

    # Center zone states
    'CIUDAD DE MEXICO': 'center',
    'JALISCO': 'center',
    'PUEBLA': 'center',
    'QUERETARO': 'center',
    'HIDALGO': 'center',
    'ESTADO DE MEXICO': 'center',
    'GUERRERO': 'center',
    'TLAXCALA': 'center',
    'COLIMA': 'center',
    'GUANAJUATO': 'center',
    'MICHOACAN': 'center',
    'SAN LUIS POTOSI': 'center',
    'MORELOS': 'center',
    'NAYARIT': 'center',
    'ZACATECAS': 'center',
    'AGUASCALIENTES': 'center',
}

# Ensure 'Kcustomer' column is in uppercase before mapping
df['Location'] = df['Location'].str.upper().str.strip()

df['Kcustomer'] = df['Location'].map(state_to_zone).fillna('center')

# Display the DataFrame
df.head()
```

	Customer	Class	Kcustomer	Location
0	Customer1	Class01	center	CIUDAD DE MEXICO
1	Customer2	Class01	center	HIDALGO
2	Customer3	Class01	center	CIUDAD DE MEXICO
3	Customer4	Class01	center	ESTADO DE MEXICO
4	Customer5	Class01	center	QUERETARO

The following dictionary helps us to segment the states by three regions, so the dictionary divides the states accordingly.

DATA TECHNIQUES

```
df=df.rename(
columns = {
"Customer": "customer",
"Class": "type_c",
"Kcustomer": "consumption_c",
"Location": "location_c"
}
)
df.head()
```

	customer	type_c	consumption_c	location_c
0	Customer1	Class01	center	CIUDAD DE MEXICO
1	Customer2	Class01	center	HIDALGO
2	Customer3	Class01	center	CIUDAD DE MEXICO
3	Customer4	Class01	center	ESTADO DE MEXICO
4	Customer5	Class01	center	QUERETARO

We renamed the columns in order to be consistent with the SQL code

```
# Save the corrected dataframe to a new CSV file in the same directory
df.to_csv('customer_corrected.csv', index=False)

df.head()
```

We saved it to the needed directory

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

```
customer          0
type_c            0
consumption_c     0
location_c        0
dtype: int64
```

Finally we checked if there was an empty row

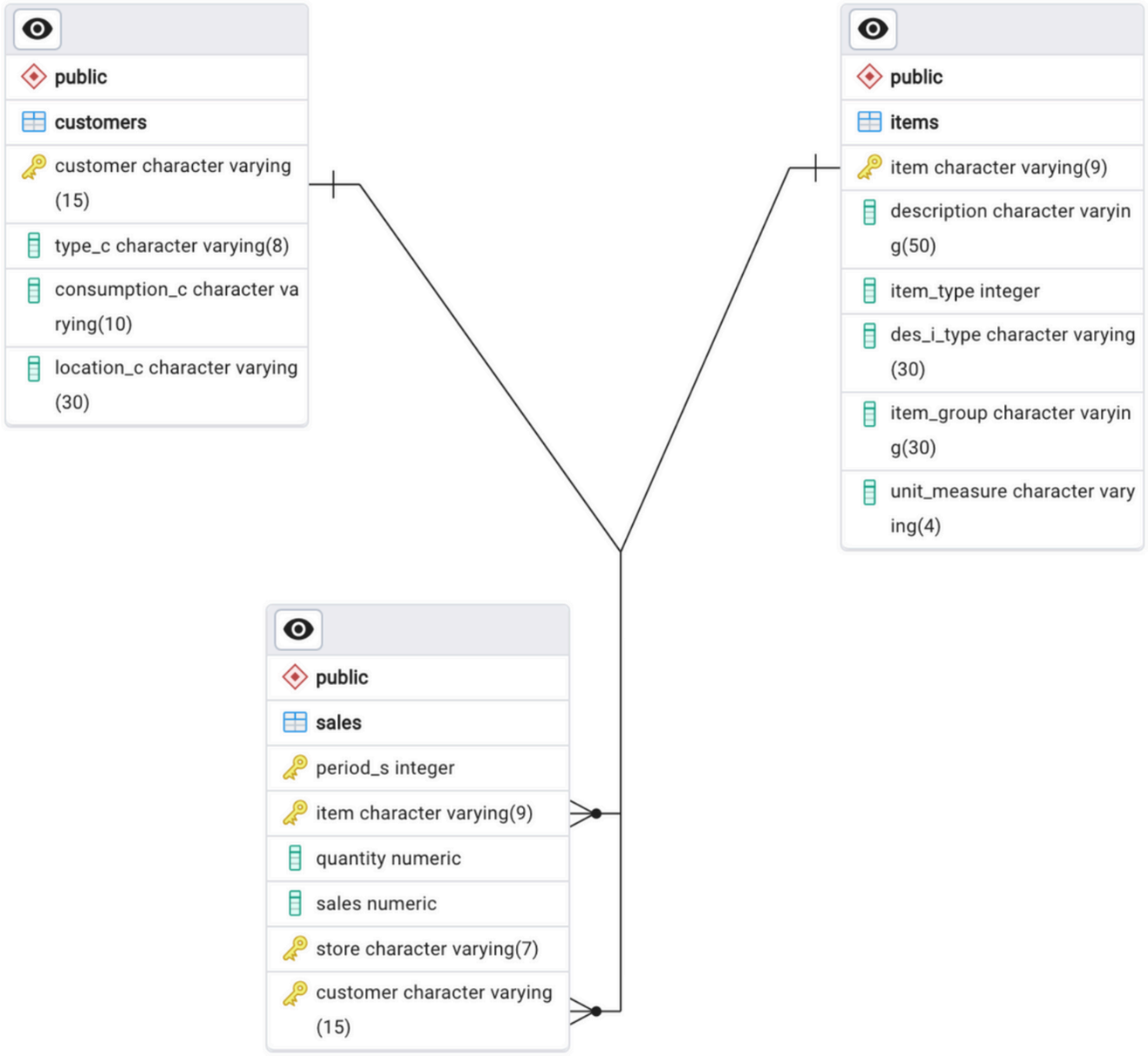
VISUALIZATION

```
DROP TABLE IF EXISTS customers CASCADE;
DROP TABLE IF EXISTS items CASCADE;
DROP TABLE IF EXISTS sales CASCADE;
```

```
CREATE TABLE customers(
    customer VARCHAR(15) PRIMARY KEY NOT NULL,
    type_c VARCHAR(8) NOT NULL,
    consumption_c VARCHAR(10) ,
    location_c  VARCHAR(30)
);
```

```
CREATE TABLE items(
    item VARCHAR(9) PRIMARY KEY NOT NULL,
    description VARCHAR(50) NOT NULL,
    item_type INT NOT NULL,
    des_i_type VARCHAR(30) NOT NULL,
    item_group VARCHAR(30) NOT NULL,
    unit_measure VARCHAR(4)
);
```

```
CREATE TABLE sales(
    period_s INT NOT NULL,
    item VARCHAR(9) NOT NULL,
    quantity NUMERIC ,
    sales NUMERIC ,
    store VARCHAR(7) NOT NULL,
    customer VARCHAR(15)NOT NULL,
    PRIMARY KEY (period_s,item,store, customer),
    FOREIGN KEY (item) REFERENCES items(item),
    FOREIGN KEY (customer) REFERENCES customers(customer)
);
```



VISUALIZATION

```
CREATE VIEW sales_store
    as
    select store, sum (sales) from sales
    group by store;

    select* from sales_store;
```

	store character varying (7)	sum numeric
1	StoreA	194203273.06
2	StoreB	119981127.64
3	StoreC	937468637.32
4	StoreD	52011531.30

VISUALIZATION

```
CREATE VIEW sales_location
    as
    select location_c, sum(sales) from customers, sales
    where customers.customer = sales.customer
    group by location_c;

    select* from sales_location;
```

	location_c character varying (30)	sum numeric
1	06600	341.22
2	15810	1255
3	5524110399	194.0
4	86690	131627.18
5	AGUASCALIENTES	267820
6	BAJA CALIFORNIA	146072.20
7	BAJA CALIFORNIA SUR	16073
8	C.A.	109236.56
9	C.D.	382
10	CAMPECHE	2359253.02

VISUALIZATION

```
CREATE VIEW  sales_zone
    as
    select consumption_c, sum(sales) from customers, sales
    where customers.customer = sales.customer
    group by consumption_c;

    select* from sales_zone;
```

	consumption_c character varying (10)	sum numeric
1	center	1231824695.17
2	north	13420696.73
3	south	58419177.42

VISUALIZATION

```sql
CREATE VIEW item_group
    AS
    SELECT items.item_group, SUM(sales.sales) AS total_sales
    FROM items
    JOIN sales ON items.item = sales.item
    GROUP BY items.item_group
    ORDER BY total_sales DESC;

    SELECT * FROM item_group;
```

	item_group character varying (30)	total_sales numeric
1	Nuts	229279896.15
2	Dried Fruits	211245494.27
3	Dried pepper	199941450.35
4	Spices	156932002.30
5	Seeds	142489692.25
6	Grocery	137071210.00
7	Seafood	112375631.03
8	Whie label	106152388.21
9	Fruits and vegetables	8176804.76

VISUALIZATION

```sql
CREATE VIEW item_subgroup
    AS
    SELECT items.des_i_type, item_group, SUM(sales.sales) AS total_sales
    FROM items
    JOIN sales ON items.item = sales.item
    GROUP BY des_i_type, item_group
    ORDER BY total_sales DESC
    limit 10;

    SELECT * FROM item_subgroup;
```

	des_i_type character varying (30)	item_group character varying (30)	total_sales numeric
1	Nut	Nuts	142122036.69
2	Guajillo	Dried pepper	75627067.61
3	Almond	Nuts	56652637.28
4	Cinnamon	Spices	56026457.35
5	Jamaica	Spices	52954557.61
6	Shrimp	Seafood	45945379.08
7	Blueberry	Dried Fruits	41524646.78
8	Tree	Dried pepper	34282058.92
9	Bean	Seeds	32687852.21
10	Cod	Seafood	29866794.04

CONCLUSION

We confirm that a basabase can be well used by doing a deep cleaning process in which you simplify the data and accomodate it, in order to create visualizations that helps the decision making process in any company department.

THANK'S FOR WATCHING