DAPP-2024

Final project (farm)

Ruba Balubaid

RubaBalubaid@outlook.com

Order No. 124813149

farm project

1- Data cleaning:

I use python for clean data in (Carnids Farm Sales)

For Carnids Farm Sales

- Read data

```
import csv
import pandas as pd

with open('/content/Carnids Farm Sales.csv', 'r') as file:
    reader = csv.reader(file, delimiter=';')

    for row in reader:
        print(row)
```

```
['23/12/23', 'CID0656', 'Customer 656', 'Los Angeles', 'Meat', '3', '54.2', '67.75', '203.25', 'Yes', '1', 'Credit Card', 'One Time', '15/10/23', '17/12/23']
['16/07/23', 'CID0657', 'Customer 657', 'Phoenix', 'Packaged Foods', '10', '11.47', '14.34', '143.4', 'Yes', '1', 'Cash', 'One Time', '16/07/23', '16/07/23']
['11/09/23', 'CID0658', 'Customer 658', 'Chicago', 'Bakery', '14', '20.64', '25.8', '361.2', 'Yes', '1', 'Credit Card', 'Weekly', '29/07/23', '30/08/23']
['24/07/23', 'CID0659', 'Customer 659', 'Phoenix', 'Vegetables', '6', '70.09', '87.61', '525.66', 'Yes', '3', 'Online Payment', 'One Time', '09/06/23', '08/07/23']
['05/05/23', 'CID0660', 'Customer 660', 'Houston', 'Beverages', '3', '14.95', '18.69', '56.07', 'Yes', '1', 'Debit Card', 'Weekly', '20/03/23', '06/04/23']
['17/03/23', 'CID0661', 'Customer 661', 'New York', 'Bakery', '16', '22.56', '28.2', '451.2', 'No', '3', 'Online Payment', 'Weekly', '13/03/23', '14/03/23']
['02/12/23', 'CID0662', 'Customer 662', 'Phoenix', 'Dairy', '7', '10.72', '13.4', '93.8', 'No', '2', 'Debit Card', 'Weekly', '14/01/23', '12/03/23']
['19/12/23', 'CID0663', 'Customer 663', 'Phoenix', 'Fruits', '14', '54.22', '67.77', '948.78', 'Yes', '4', 'Cash', 'Monthly', '08/11/23', '29/11/23']
['03/01/23', 'CID0664', 'Customer 664', 'New York', 'Packaged Foods', '9', '75.54', '94.43', '849.87', 'No', '4', 'Debit Card', 'Weekly', '03/01/23', '03/01/23']
['20/01/24', 'CID0665', 'Customer 665', 'Houston', 'Beverages', '15', '21.81', '27.26', '408.9', 'Yes', '2', 'Credit Card', 'Quarterly', '22/12/23', '14/01/24']
['30/01/24', 'CID0666', 'Customer 666', 'Los Angeles', 'Dairy', '3', '38.24', '47.8', '143.4', 'Yes', '4', 'Credit Card', 'One Time', '17/05/23', '06/01/24']
['24/02/23', 'CID0667', 'Customer 667', 'Phoenix', 'Vegetables', '2', '50.9', '63.62', '127.24', 'Yes', '4', 'Cash', 'Monthly', '02/01/23', '12/02/23']
['06/08/23', 'CID0668', 'Customer 668', 'New York', 'Meat', '2', '79.96', '99.95', '199.9', 'Yes', '4', 'Debit Card', 'One Time', '16/02/23', '22/04/23']
['23/04/23', 'CID0669', 'Customer 669', 'Chicago', 'Bakery', '3', '8.74', '10.93', '32.79', 'Yes', '3', 'Cash', 'Monthly', '04/02/23', '27/02/23']
['01/04/23', 'CID0670', 'Customer 670', 'New York', 'Fruits', '27', '44.1', '55.121', '1488.27', 'Yes', '1', 'Credit Card', 'Quarterly', '31/03/23', '31/03/23']
['23/12/23', 'CID0671', 'Customer 671', 'Chicago', 'Bakery', '14', '31.06', '38.82', '543.48', 'Yes', '3', 'Online Payment', 'Monthly', '08/06/23', '13/12/23']
['23/09/23', 'CID0672', 'Customer 672', 'Houston', 'Packaged Foods', '18', '69.32', '86.65', '1559.7', 'Yes', '3', 'Cash', 'Monthly', '23/06/23', '05/08/23']
['26/08/23', 'CID0673', 'Customer 673', 'Los Angeles', 'Vegetables', '10', '38.29', '47.86', '478.6', 'Yes', '3', 'Online Payment', 'Monthly', '31/03/23', '14/06/2
```

- Understand data

```
print(sales_data.describe()) # Print summary statistics of numerical columns
```

```
          Quantity   Unit Cost   Unit Price  Total Revenue  Delivery Days
count  1000.000000  1000.00000  1000.000000    1000.000000    1000.000000
mean     10.036000    40.47656    50.595761     513.013880       2.714000
std       5.842503    23.29664    29.120737     461.005353       1.091511
min       0.000000     0.42000     0.520000       2.220000       1.000000
25%       5.000000    20.39250    25.492000     137.352500       2.000000
50%      10.000000    41.29500    51.620000     378.130000       3.000000
75%      15.000000    60.89000    76.117500     750.395000       4.000000
max      28.000000    86.23000   107.789000    2594.840000       4.000000
```

```
print(sales_data.info())   # Print a concise summary of the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Date                1000 non-null   object
 1   Customer ID         1000 non-null   object
 2   Customer Name       1000 non-null   object
 3   City                1000 non-null   object
 4   Product Category    1000 non-null   object
 5   Quantity            1000 non-null   int64
 6   Unit Cost           1000 non-null   float64
 7   Unit Price          1000 non-null   float64
 8   Total Revenue       1000 non-null   float64
 9   New Customer        1000 non-null   object
 10  Delivery Days       1000 non-null   int64
 11  Payment Method      1000 non-null   object
```

- Convert data types:

```
# Convert date columns to datetime
sales_data['Date'] = pd.to_datetime(sales_data['Date'])
sales_data['First Purchase Date'] = pd.to_datetime(sales_data['First Purchase Date'])
sales_data['Last Purchase Date'] = pd.to_datetime(sales_data['Last Purchase Date'])
```

- Check and remove inconsistencies data in date and negative total

```
# Check if 'First Purchase Date' is before 'Last Purchase Date'
inconsistent_dates = sales_data[sales_data['First Purchase Date'] > sales_data['Last Purchase Date']]
print(inconsistent_dates)
```

```
          Date Customer ID Customer Name          City Product Category  \
1   2023-02-27     CID0002    Customer 2       Houston       Vegetables
12  2023-07-10     CID0013   Customer 13       Chicago            Meat
15  2023-05-08     CID0016   Customer 16       Chicago        Beverages
25  2023-12-04     CID0026   Customer 26      New York           Fruits
26  2024-02-01     CID0027   Customer 27       Houston   Packaged Foods
..         ...         ...           ...           ...              ...
970 2023-03-03     CID0971  Customer 971   San Antonio           Bakery
973 2024-01-26     CID0974  Customer 974       Phoenix           Fruits
991 2023-04-07     CID0992  Customer 992   San Antonio        Beverages
992 2023-02-04     CID0993  Customer 993   Los Angeles           Bakery
995 2023-12-29     CID0996  Customer 996  Philadelphia             Meat

     Quantity  Unit Cost  Unit Price  Total Revenue New Customer  \
1           6      16.52      20.646         134.20          Yes
12         16      62.50      78.130        1250.08          Yes
15         16      18.33      22.910         366.56          Yes
25         16      24.57      30.712         506.75          Yes
26         13      63.06      78.820        1024.66          Yes
..        ...        ...         ...            ...          ...
970        11      56.79      70.990         780.89          Yes
973        17      36.13      45.160         767.72          Yes
991        11      35.05      43.810         481.91          Yes
992        17      44.13      55.160         937.72          Yes
995        17      40.73      50.910         865.47          Yes
```

```
print(len(inconsistent_dates))
```

```
268
```

```
sales_data_cleaned = sales_data[sales_data['First Purchase Date'] <= sales_data['Last Purchase Date']]
```

```
# Identify negative values
negative_quantities = sales_data[sales_data['Quantity'] < 0]
print(negative_quantities)
```

```
Empty DataFrame
Columns: [Date, Customer ID, Customer Name, City, Product Category, Quantity, Unit Cost, Unit Price, Total Revenue, New Customer, De
Index: []
```

```
# Check for negative values in Unit Cost, Unit Price, and Total Revenue
negative_values = sales_data_cleaned[(sales_data_cleaned['Unit Cost'] < 0) |
                                     (sales_data_cleaned['Unit Price'] < 0) |
                                     (sales_data_cleaned['Total Revenue'] < 0)]
print(negative_values)
```

```
Empty DataFrame
Columns: [Date, Customer ID, Customer Name, City, Product Category, Quantity, Unit Cost, Unit Price, Total Revenue, New Customer, De
Index: []
```

- Add new column

```
# Calculate the duration between the first and last purchase
sales_data['Customer Relationship Duration'] = (sales_data['Last Purchase Date'] - sales_data['First Purchase Date']).dt.days
```

- Check size and Save file after cleaning

```
# Check the number of rows before cleaning
print(f"Original DataFrame size: {len(sales_data)}")
print(f"DataFrame after each cleaning step:")

# After removing date inconsistencies
print(f"Size after date consistency check: {len(sales_data_cleaned)}")

# After removing negative quantities
print(f"Size after removing negative quantities: {len(sales_data_cleaned)}")
```

```
Original DataFrame size: 1000
DataFrame after each cleaning step:
Size after date consistency check: 732
Size after removing negative quantities: 732
```

```
# Define the path where you want to save the cleaned data
file_path = '/content/cleaned_sales_data.csv'

# Save the cleaned DataFrame to a CSV file
sales_data_cleaned.to_csv(file_path, index=False)

print(f"Cleaned data saved to {file_path}")
```

```
Cleaned data saved to /content/cleaned_sales_data.csv
```

For reviews file:

Reviews file contain two columns for review and Customer ID I’m use NLP and pre-trained model as extra point.

NLP for Review file:

```python
import re
from nltk.tokenize import word_tokenize
import nltk

# Download necessary NLTK resources
nltk.download('punkt')

# Function to preprocess and tokenize text
def preprocess_text(text):
    text = re.sub(r'[^\w\s]', '', text.lower())  # Remove punctuation and lowercasing
    return word_tokenize(text)

# Apply preprocessing
reviews_df['tokens'] = reviews_df['Text'].apply(preprocess_text)
```

I used them to read and anlysis ‘Text’ column and model classify it to positive or negative:

```python
import pandas as pd
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Load the model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased-finetuned-sst-2-english")
model = AutoModelForSequenceClassification.from_pretrained("distilbert/distilbert-base-uncased-finetuned-sst-2-english")

# Initialize the sentiment analysis pipeline
sentiment_analyzer = pipeline('sentiment-analysis', model=model, tokenizer=tokenizer)

# Function to analyze sentiment
def analyze_sentiment(text):
    inputs = tokenizer(text, truncation=True, padding=True, max_length=512, return_tensors='pt')
    result = model(**inputs)
    logits = result.logits
    sentiment = 'POSITIVE' if logits.argmax() == 1 else 'NEGATIVE'
    return sentiment

# Load your reviews data
file_path = '/content/farm_reviw.txt'
reviews_df = pd.read_csv(file_path, delimiter='\t')

# Apply sentiment analysis
reviews_df['Sentiment'] = reviews_df['Text'].apply(analyze_sentiment)

# Save the results to a new CSV file
output_file_path = '/content/sentiment_analysis_results.csv'
reviews_df.to_csv(output_file_path, index=False)

# Display the path to the saved file
print(f"Results saved to {output_file_path}")
```

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Prepare data
X = reviews_df['Text']
y = reviews_df['Sentiment']
y_pred = reviews_df['predicted_label']

# Split data
X_train, X_test, y_train, y_test, y_pred_train, y_pred_test = train_test_split(X, y, y_pred, test_size=0.2, random_state=42)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred_test)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 100
```

2- Create Database:

At this stage I tried many database management like phpMyAdmin, mySQL, PostgreSQL and all failed and finally I relied on MSSQL with these codes:

- Create table

```sql
CREATE TABLE employees (
    employee_id INT IDENTITY(1,1) PRIMARY KEY,
    employee_name NVARCHAR(100),
    position NVARCHAR(50),
    department NVARCHAR(50),
    hire_date DATE,
    salary DECIMAL(10, 2),
    contact_number NVARCHAR(15),
    email NVARCHAR(100),
    address NVARCHAR(MAX),
    shift NVARCHAR(20)
);
```

```sql
-- Create table
CREATE TABLE animals (
    animal_id INT IDENTITY(1,1) PRIMARY KEY,
    animal_type NVARCHAR(50),
    breed NVARCHAR(50),
    date_of_birth DATE,
    purchase_date DATE,
    weight DECIMAL(6, 2),
    health_status NVARCHAR(50),
    last_vet_visit DATE,
    feeding_schedule NVARCHAR(50)
);
```

- Insert info to employee and animal tables

```sql
-- Insert 1000 rows of dummy data
DECLARE @i INT = 1;
WHILE @i <= 1000
BEGIN
    INSERT INTO employees (
        employee_name, position, department, hire_date, salary,
        contact_number, email, address, shift
    ) VALUES (
        'Employee ' + CAST(@i AS NVARCHAR(10)),
        CASE
            WHEN ABS(CHECKSUM(NEWID()) % 3) = 0 THEN 'Farm Manager'
            WHEN ABS(CHECKSUM(NEWID()) % 3) = 1 THEN 'Veterinarian'
            ELSE 'Farm Worker'
        END,
        CASE
            WHEN ABS(CHECKSUM(NEWID()) % 3) = 0 THEN 'Administration'
            WHEN ABS(CHECKSUM(NEWID()) % 3) = 1 THEN 'Animal Care'
            ELSE 'Support'
        END,
        DATEADD(DAY, ABS(CHECKSUM(NEWID()) % 1825) - 365, GETDATE()),  -- Random date within the last 5 years
        ROUND(RAND() * 50000, 2),
        '555-' + RIGHT('0000' + CAST(ABS(CHECKSUM(NEWID()) % 10000) AS NVARCHAR(4)), 4),
        'employee' + CAST(@i AS NVARCHAR(10)) + '@farm.com',
    health_status, last_vet_visit, feeding_schedule
) VALUES (
    CASE
        WHEN ABS(CHECKSUM(NEWID()) % 3) = 0 THEN 'Cow'
        WHEN ABS(CHECKSUM(NEWID()) % 3) = 1 THEN 'Chicken'
        ELSE 'Pig'
    END,
    CASE
        WHEN ABS(CHECKSUM(NEWID()) % 2) = 0 THEN 'Breed A'
        ELSE 'Breed B'
    END,
    DATEADD(DAY, -ABS(CHECKSUM(NEWID()) % 730), GETDATE()),  -- Random date within the last 2 years
    DATEADD(DAY, -ABS(CHECKSUM(NEWID()) % 365), GETDATE()),   -- Random date within the last year
    ROUND(RAND() * 500, 2),
    CASE
        WHEN ABS(CHECKSUM(NEWID()) % 3) = 0 THEN 'Healthy'
        WHEN ABS(CHECKSUM(NEWID()) % 3) = 1 THEN 'Sick'
        ELSE 'Needs Attention'
    END,
    DATEADD(DAY, -ABS(CHECKSUM(NEWID()) % 180), GETDATE()),   -- Random date within the last 6 months
    CASE
```

- View table

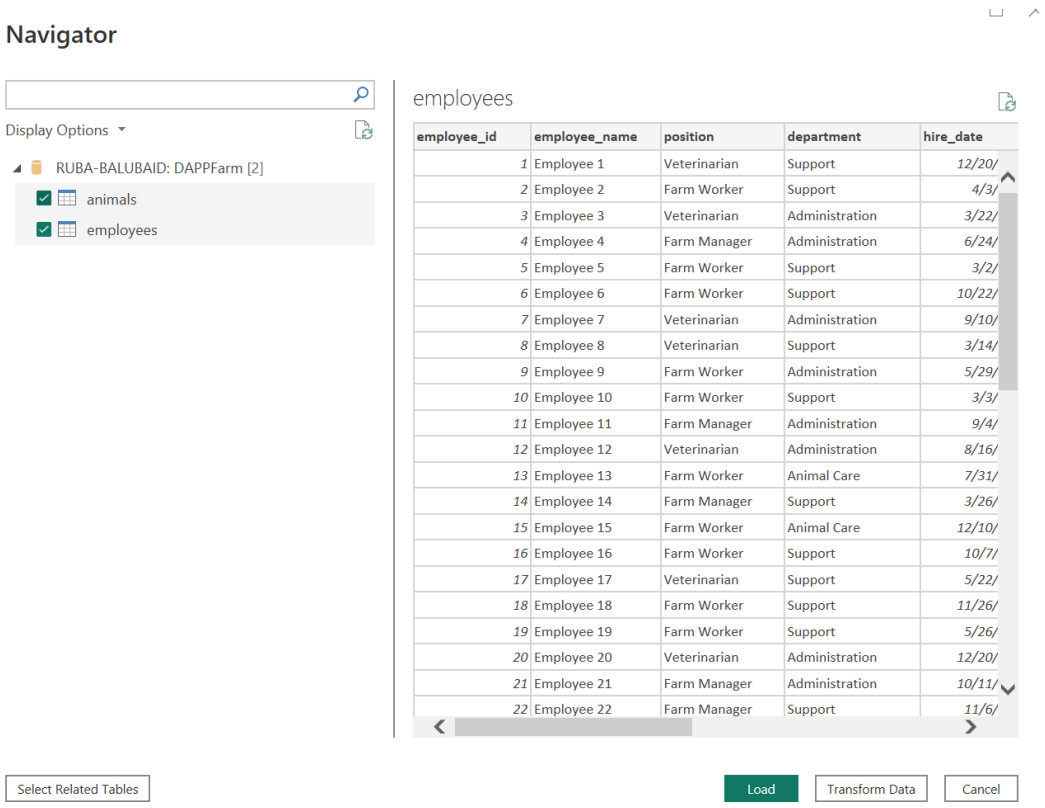
	employee_id	employee_name	position	department	hire_date	salary	contact_number	email	address	shift
243	243	Employee 243	Farm Worker	Administration	2024-05-09	19266.16	555-9407	employee243@farm.c...	Address ...	Morning
244	244	Employee 244	Farm Manager	Support	2027-07-03	44386.00	555-0280	employee244@farm.c...	Address ...	Evening
245	245	Employee 245	Veterinarian	Support	2026-03-11	8637.26	555-5346	employee245@farm.c...	Address ...	Morning
246	246	Employee 246	Farm Worker	Support	2027-01-19	7332.83	555-6264	employee246@farm.c...	Address ...	Morning
247	247	Employee 247	Veterinarian	Support	2024-07-03	37438.57	555-2586	employee247@farm.c...	Address ...	Morning
248	248	Employee 248	Farm Manager	Administration	2028-04-11	30813.08	555-9376	employee248@farm.c...	Address ...	Morning
249	249	Employee 249	Veterinarian	Administration	2024-09-11	11849.49	555-0442	employee249@farm.c...	Address ...	Morning
250	250	Employee 250	Farm Worker	Administration	2024-01-13	40076.94	555-1662	employee250@farm.c...	Address ...	Evening
251	251	Employee 251	Farm Worker	Administration	2024-01-10	10819.96	555-8448	employee251@farm.c...	Address ...	Evening
252	252	Employee 252	Farm Manager	Administration	2025-01-04	42850.84	555-0567	employee252@farm.c...	Address ...	Morning
253	253	Employee 253	Farm Worker	Support	2025-09-25	21645.47	555-1619	employee253@farm.c...	Address ...	Morning

	animal_id	animal_type	breed	date_of_birth	purchase_date	weight	health_status	last_vet_visit	feeding_schedule
1	1	Cow	Breed A	2024-02-16	2023-11-14	12.40	Needs Attention	2024-03-21	Once a day
2	2	Cow	Breed B	2023-01-04	2024-05-28	452.10	Needs Attention	2024-04-23	Twice a day
3	3	Pig	Breed B	2022-10-07	2023-11-30	3.58	Healthy	2024-06-01	Twice a day
4	4	Pig	Breed A	2023-12-18	2024-05-16	65.59	Needs Attention	2024-05-20	Once a day
5	5	Cow	Breed B	2022-08-30	2023-09-29	477.50	Needs Attention	2024-07-17	Once a day
6	6	Chicken	Breed B	2023-05-04	2024-07-06	130.42	Healthy	2024-03-04	Twice a day
7	7	Cow	Breed A	2024-05-15	2023-08-30	356.73	Healthy	2024-02-16	Once a day
8	8	Chicken	Breed B	2023-02-14	2023-08-26	365.17	Healthy	2024-07-28	Once a day
9	9	Pig	Breed B	2024-04-04	2024-06-07	304.97	Sick	2024-02-13	Once a day
10	10	Chicken	Breed A	2023-05-24	2024-04-19	445.11	Healthy	2024-07-30	Twice a day
11	11	Chicken	Breed A	2023-06-10	2024-07-04	380.13	Sick	2024-03-13	Once a day

3- Connect between SQL and CSV files on power bi

To connect between then on power bi I add animal_id and employee_id columns in CSV file

- Load MSSQL database



- Load CSV and reviews files
- For connect between all tabels on power bi we need to add column contain animal_id and employee_id. Add index to fill each column

Custom Column

Add a column that is computed from the other columns.

New column name

animal_id

Custom column formula

```
= "A" & Text.PadStart(Number.ToText([Index]), 4, "0")
```

Available columns

- Date
- Customer ID
- Customer Name
- City
- Product Category
- Quantity
- Unit Cost

<< Insert

Learn about Power Query formulas

No syntax errors have been detected.

OK Cancel

Index	animal_id	employee_id
0	A0000	E0000
1	A0001	E0001
2	A0002	E0002
3	A0003	E0003
4	A0004	E0004
5	A0005	E0005
6	A0006	E0006
7	A0007	E0007
8	A0008	E0008
9	A0009	E0009
10	A0010	E0010
11	A0011	E0011
12	A0012	E0012
13	A0013	E0013
14	A0014	E0014
15	A0015	E0015
16	A0016	E0016
17	A0017	E0017
18	A0018	E0018
19	A0019	E0019

PROPERTIES

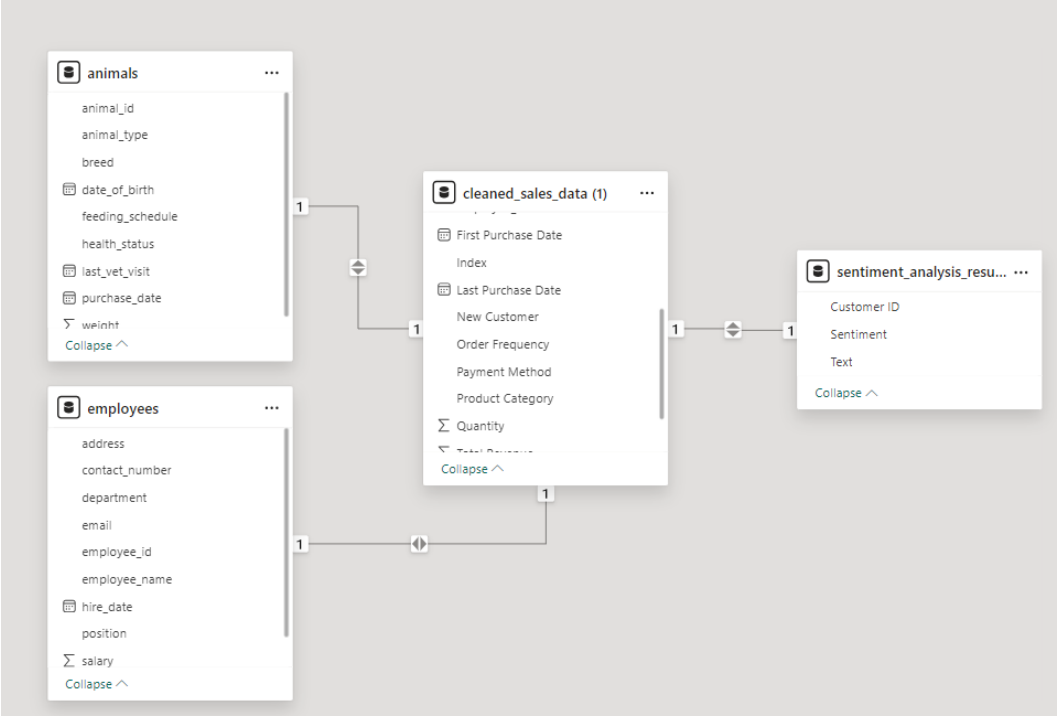
Name

cleaned_sales_data (1)

All Properties

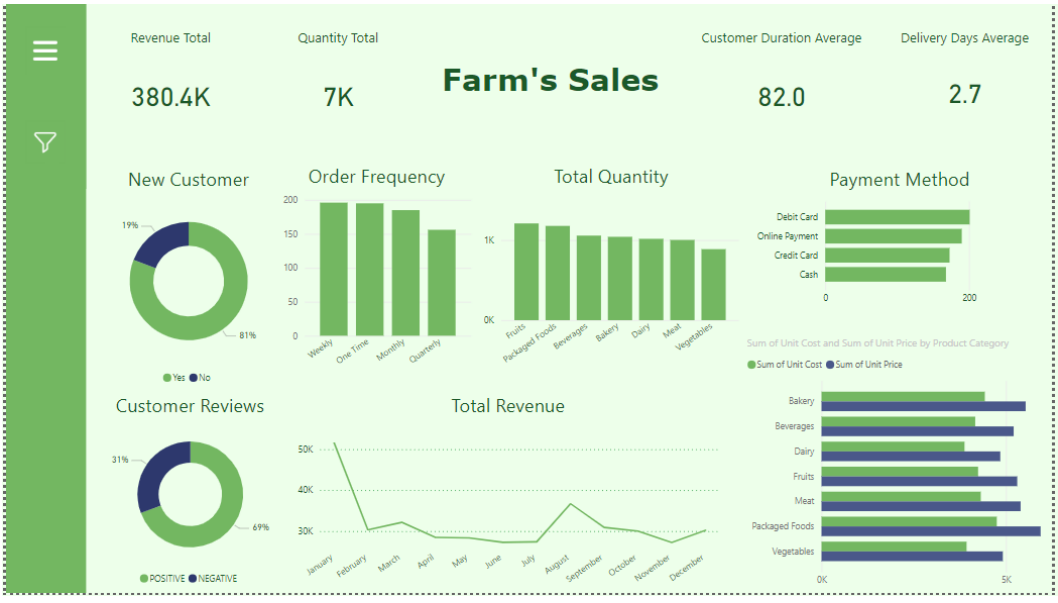
APPLIED STEPS

- Source
- Promoted Headers
- Changed Type
- Added Index
- Added Custom
- Added Custom1

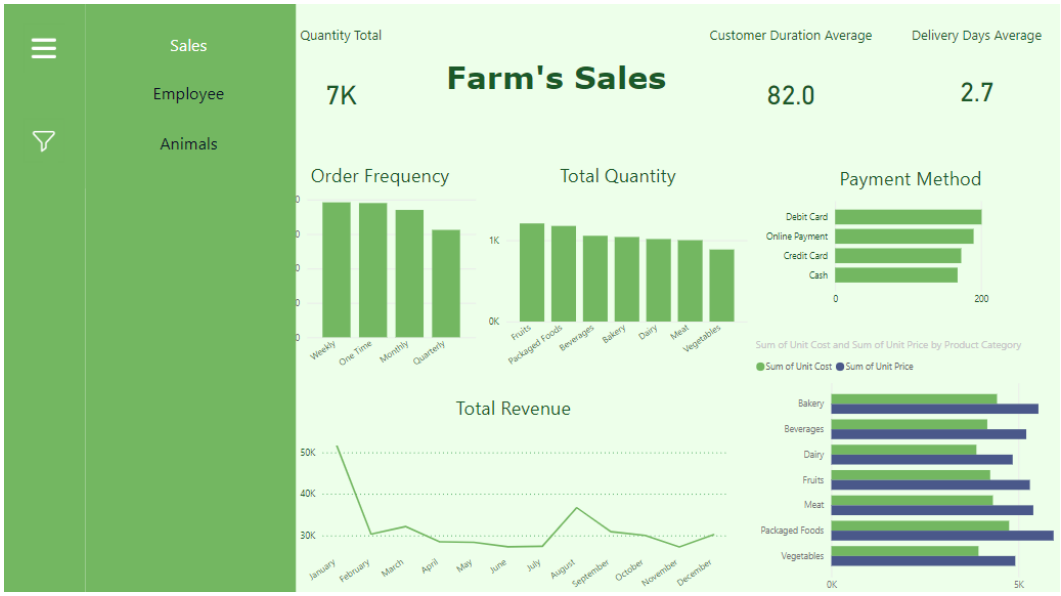


My dashboard consists of three pages

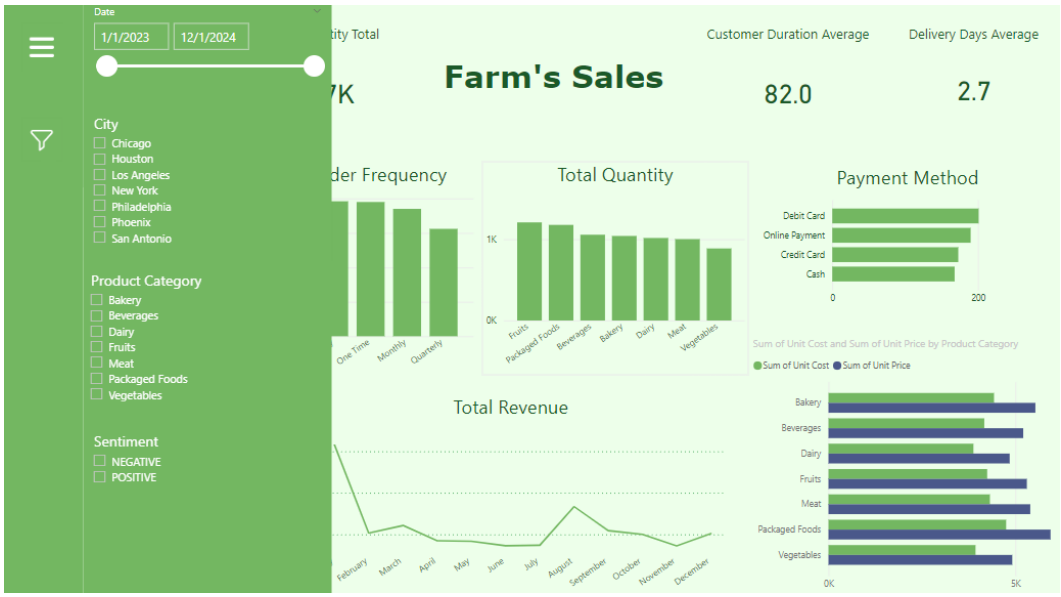
- Page 1:



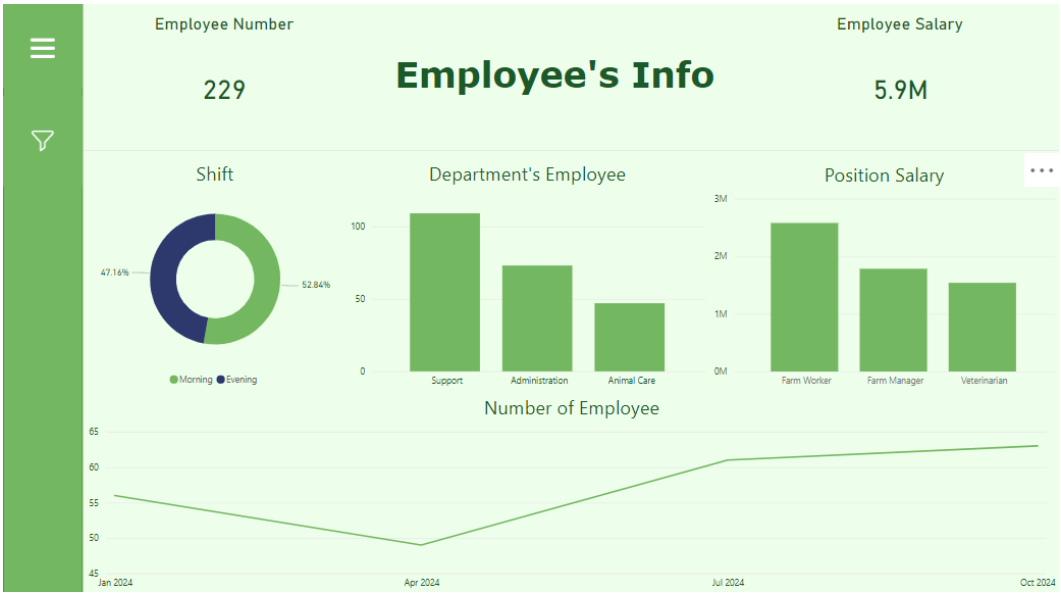
This page show farm sales which contain Customers New and analysis reviwe’s if positive or negative by NLP, details of order frequency and other details. In top left we have two buttons first for menu:



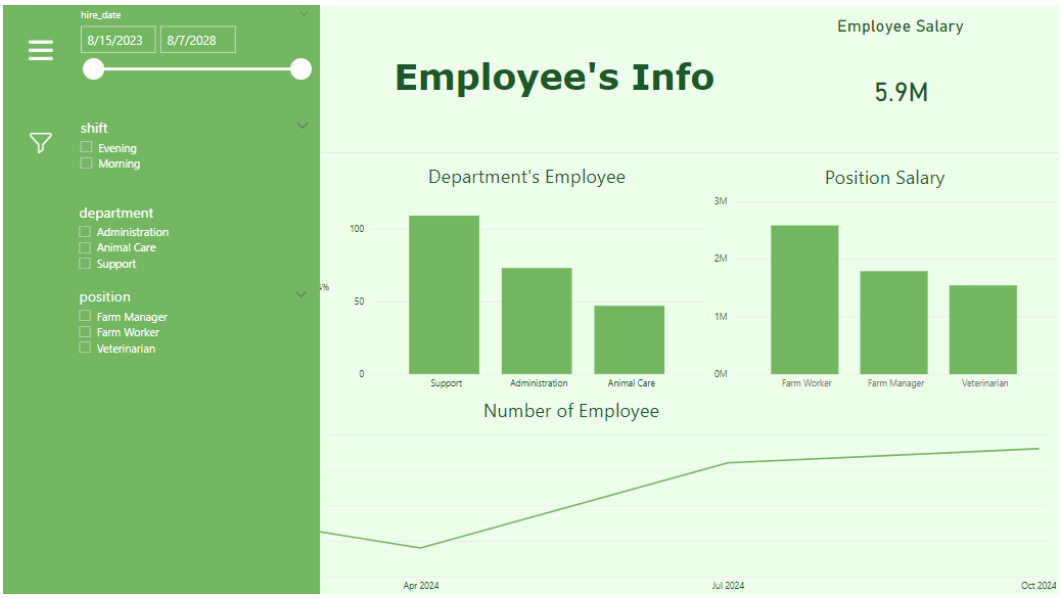
Witch allow navigate between pages. Second for filter:



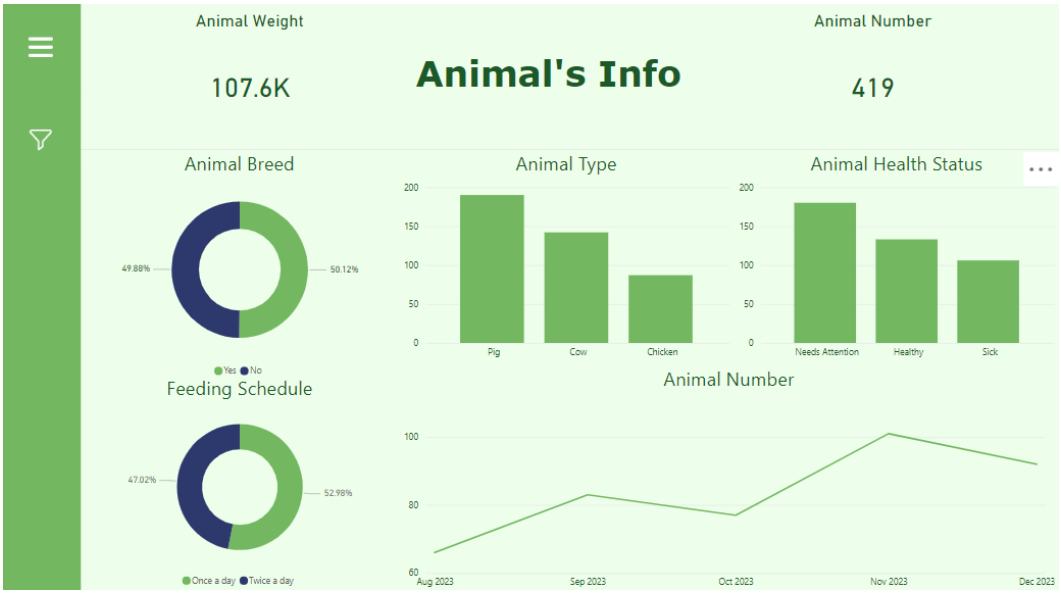
Second page for employees:



Filter in employees:



Last page for animals:



Filter for animals: