E-COMMERCE CUSTOMER BEHAVIOUR ANALYSIS AND INSIGHTS

Jarvin Mutatiina
Supervisor

Imad Abo Dehn
Team Leader - Trainee

Ameen Abass
Trainee

Majd Al-Batran
Trainee

Sakaria Abukar
Trainee

SpikeUp.AI 14-09-2024

AGENDA

- Introduction
- Data Overview
- Data Processing with Spark
- Analysis & Insights
- Orchestration with Airflow
- CI/CD Pipeline with GitHub Actions
- Challenges & Solutions
- Conclusion
- Q&A

INTRODUCTION

❖ Objective:

Analyze e-commerce customer behavior to derive insights that can improve business decision-making.

❖ Scope:

1. **Data Processing with Spark**
2. **Orchestration with Airflow**
3. **Deployment with GitHub Actions and GCP.**

DATA OVERVIEW

❖ Dataset Description:

Customer transactions including age, city, items purchased, and total spend.

❖ Data Source:

Stored in Google Cloud Storage, sourced from Kaggle.

❖ Key Variables:

Discount Applied, Membership Type, Satisfaction Level and Total Spend.

❖ Columns in this file:

1. Customer ID - Integer
2. Gender – String
3. Age – Integer
4. City – String
5. Membership Type – String
6. Total Spend – Numeric
7. Items Purchased – Integer
8. Average Rating – Numeric
9. Discount Applied – Boolean
10. Days Since Last Purchase – Integer
11. Satisfaction Level – String

DATA PROCESSING WITH SPARK

❖ **Spark Setup:**

Configuring Spark with GCP connector to read and write data.

```
from datetime import datetime, timedelta
from pyspark import SparkConf
from pyspark.sql import SparkSession


spark = SparkSession.builder \
    .appName('data-engineering-capstone') \
    .config("spark.jars", "https://storage.googleapis.com/hadoop-lib/gcs/gcs-connector-hadoop3-latest.jar") \
    .config("spark.sql.repl.eagerEval.enabled", True) \
    .getOrCreate()

# Set GCS credentials. Ensure path points to you downloaded key file
spark._jsc.hadoopConfiguration().set(
    "google.cloud.auth.service.account.json.keyfile",
    "C:\pro\gcp-key.json")
```

DATA PROCESSING WITH SPARK

❖ ETL Pipeline:

1. Extract: Read data from GCP.

```
# file path to data in GCS bucket

file_path = "gs://ecommerce-customer/e-commerce-customer-behavior.csv"


df = spark.read.csv(file_path, header=True, inferSchema=True)

df.show(5)
```

2. Transform: cleaning the data.

```
# Drop all rows that contain any null values in any column
df = df.dropna()
# Remove duplicate rows from the DataFrame
df = df.dropDuplicates()
```

3. Load: Write transformed data back to GCP.

```
# Get current datetime in the format MMDDYYYYHHMMSS
datetime_now = datetime.now().strftime("%m%d%Y%H%M%S")

# Define the base output path using formatted strings
base_output_path = f"gs://ecommerce-customer/output/{datetime_now}"

# Write inactive customers to GCS with overwrite mode
inactive_customers_output_path = f"{base_output_path}/inactive_customers"
inactive_customers.write.csv(inactive_customers_output_path, header=True, mode='overwrite')

# Write recent customers to GCS with overwrite mode
recent_customers_output_path = f"{base_output_path}/recent_customers"
recent_customers.write.csv(recent_customers_output_path, header=True, mode='overwrite')

# Write all customers to GCS with overwrite mode
customers_output_path = f"gs://ecommerce-customer/customers"
segmentation_df.write.csv(customers_output_path, header=True, mode='overwrite')
```

ANALYSIS & INSIGHTS

❖ Customer Segmentation:

- **Segmenting customers based on spending habits and activity**
- **Adding a new column " Spending Category"**

```
from pyspark.sql.functions import when

segmentation_df = df.withColumn("SpendingCategory",
                                when(df["Total Spend"] > 1000, "High")
                                .when(df["Total Spend"] > 500, "Medium")
                                .otherwise("Low"))
```

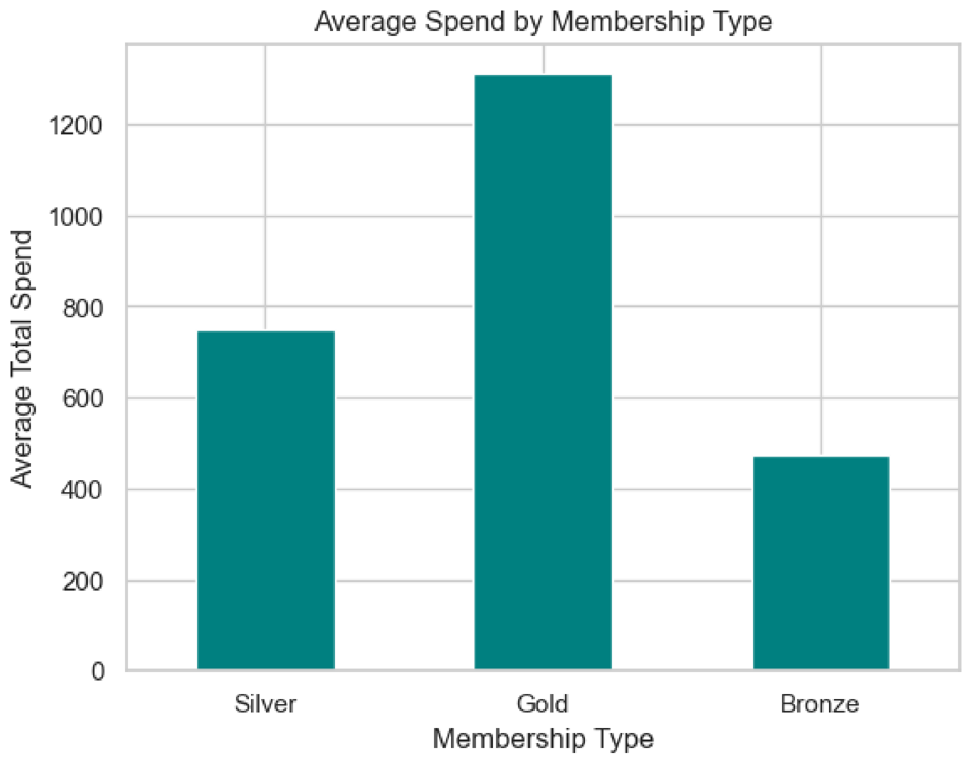
```
# Identify inactive customers (no purchase in the last 30 days)
inactive_customers = segmentation_df.filter((segmentation_df["Days Since Last Purchase"]) > 30)

# Identify recent customers (purchased within the last 7 days)
recent_customers = segmentation_df.filter((segmentation_df["Days Since Last Purchase"]) <= 30)
```

ANALYSIS & INSIGHTS

❖ Spending Behaviour Analysis:

Plotting Average Spend by Membership Type

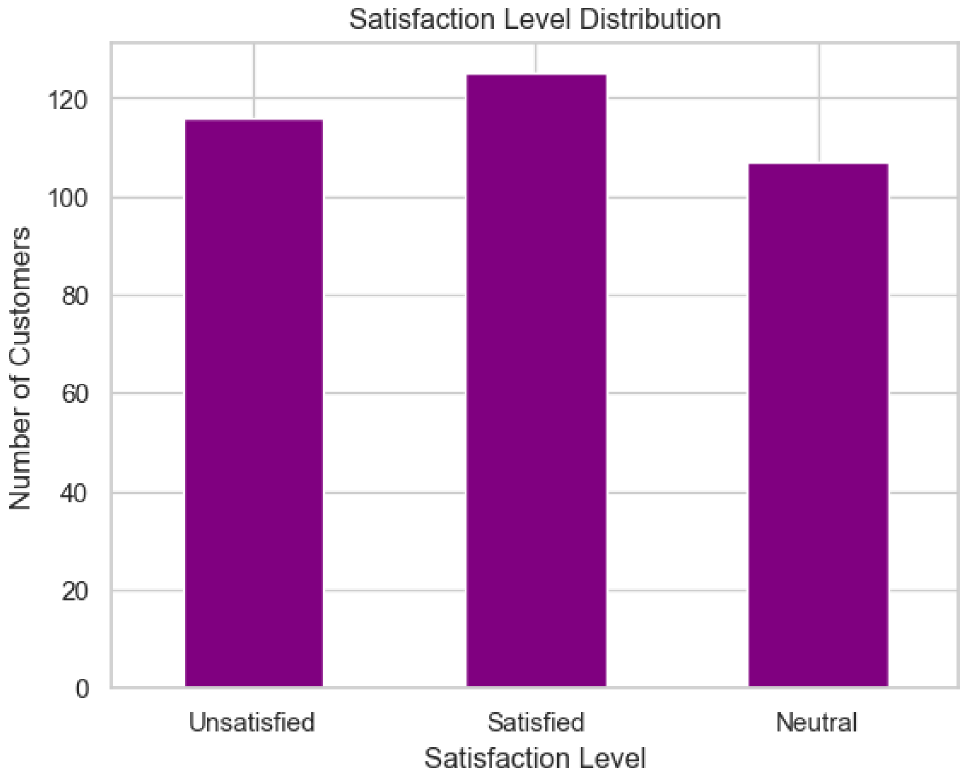


Plotting Average Spend by Discount Applied

ANALYSIS & INSIGHTS

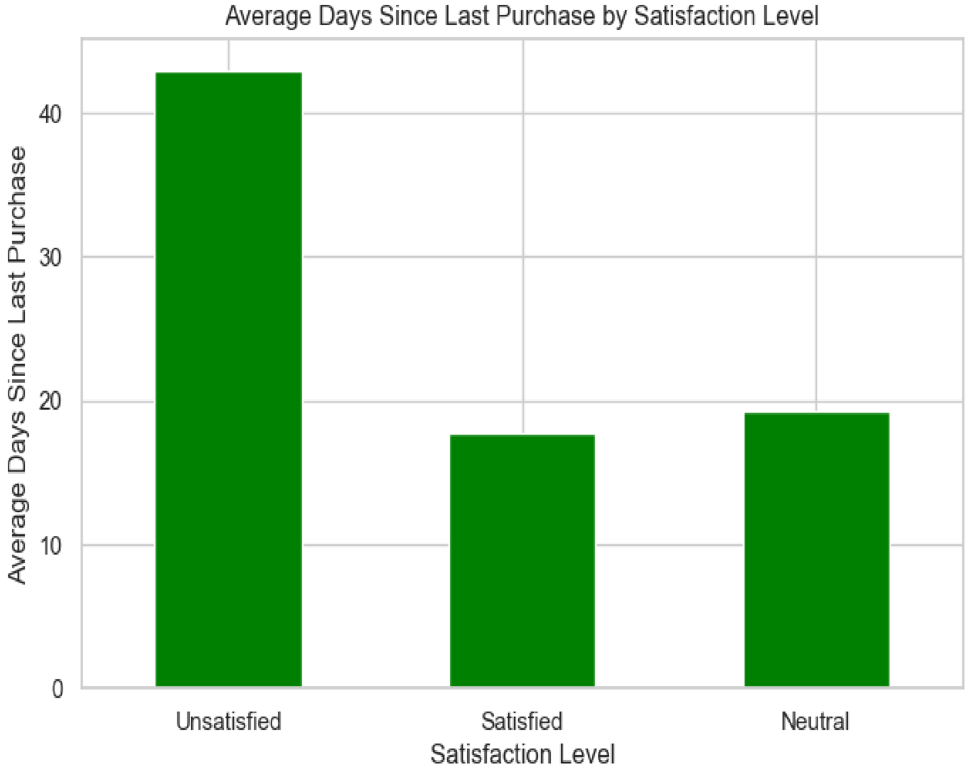
❖ **Satisfaction Analysis:**

Satisfaction Level Distribution



❖ **Churn Prediction Indicators:**

Average Days Since Last Purchase by Satisfaction Level



ORCHESTRATION WITH AIRFLOW

❖ DAG Components 1:

➢ **Library imports:**

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from pyspark.sql import SparkSession, DataFrame, functions as sf
```

➢ **DAG arguments:**

```
# Default arguments for the DAG
default_args = {
    "depends_on_past": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}
```

➢ **DAG definition:**

DAGs are scheduled to run daily at 10 AM.

```
# Define the DAG
with DAG(
    dag_id="insights_dag_11",
    start_date=datetime(2024, 7, 14),
    schedule_interval="0 10 * * *",  # Daily interval at 10am
    catchup=False,
    tags=[
        "Orchestration",
        "Customer Insights",
        "Ecommerce"
    ],
) as dag:
```

ORCHESTRATION WITH AIRFLOW

❖ DAG Components 2:

➢ **ETL Task** "etl_with_spark()" **steps:**

- **Configuring Spark:**

```python
spark = None
try:
    spark = SparkSession.builder \
        .appName('data-engineering-capstone') \
        .config("spark.jars", "https://storage.googleapis.com/hadoop-lib/gcs/gcs-connector-hadoop3-latest.jar") \
        .config("spark.sql.repl.eagerEval.enabled", True) \
        .getOrCreate()
    file_path = f"gs://{GCS_BUCKET}/customers/*.csv"
```

- **Extract:**

```python
def read_data(spark: SparkSession, file_path: str) -> DataFrame:
    return spark.read.csv(file_path, header=True, inferSchema=True)
```

```python
# Read data from GCS
customer_df = read_data(spark=spark, file_path=file_path)
```

ORCHESTRATION WITH AIRFLOW

❖ DAG Components 3:

➢ **ETL Task** "etl_with_spark()" **steps:**

- **Transform:**

```python
def calculate_highest_spend(df: DataFrame) -> DataFrame:Spend_by_City = (
        df.groupBy("City", "SpendingCategory")
          .agg(
            # Age calculations
            sf.round(sf.mean("age"), 1).alias("Average_age"),

            # Items Purchased calculations
            sf.round(sf.mean("Items Purchased"), 0).alias("Average_items_purchased"),

            # Total Spend calculations
            sf.round(sf.mean("Total Spend"), 2).alias("Average_spend"),
            sf.round(sf.sum("Total Spend"), 2).alias("Sum_spend")
          )
        .sort("Average_spend", ascending=False)
    )
    return Spend_by_City
```

```python
# Perform data transformations
insights_df = calculate_highest_spend(df=customer_df)
```

- **Load:**

```python
def write_to_gcs(df: DataFrame, filepath: str):
    df.toPandas().to_csv(filepath, index=False)
```

```python
# Write insights to GCS
datetime_now = datetime.now().strftime("%m%d%Y%H%M%S")
write_path = f"gs://{GCS_BUCKET}/insights/{datetime_now}.csv"
write_to_gcs(df=insights_df, filepath=write_path)
```

ORCHESTRATION WITH AIRFLOW

❖ DAG Components 2:

➢ **ETL Task definition:**

```
# Define the ETL task
etl_with_spark_task = PythonOperator(
    task_id="etl_with_spark",
    python_callable=etl_with_spark
)
```

➢ **Empty Task definition:**

```
# Define an empty task
does_nothing = EmptyOperator(task_id="does_nothing")
```

➢ **Tasks pipeline:**

```
# Set task dependencies
etl_with_spark_task >> does_nothing
```

CI/CD PIPELINE WITH GITHUB ACTIONS

❖ **Workflow Setup:**

Configuring GitHub Actions to deploy DAGs to GCP.

❖ **Code Snippets:**

YAML configuration for GitHub Actions.

1. **Event Trigger (on push):**

```yaml
name: Deploy to Google Cloud

# This workflow runs whenever there is a push event to the main branch.
on:
  push:
    branches:
      - main  # Specify the branch (main) for the workflow to trigger on
```

2. **Job (deploy-dags):**

```yaml
jobs:
  deploy-dags:
    permissions:
      contents: 'read'
      id-token: 'write'

    runs-on: ubuntu-latest
```

3. **Environment Variable (env):**

```yaml
env:
  DAGS_BUCKET: 'europe-west1-ecommerce-cust-d866a2be-bucket'
```

4. **Steps:**

```yaml
steps:
  - name: Checkout repository
    uses: actions/checkout@v4

  - name: Authenticate to Google Cloud
    uses: google-github-actions/auth@v2
    with:
      credentials_json: ${{ secrets.GCP_SA_KEY }}

  - name: Upload DAGs to GCS
    uses: google-github-actions/upload-cloud-storage@v2
    with:
      path: 'dags'
      destination: "${{ env.DAGS_BUCKET }}/dags"
      process_gcloudignore: false
      parent: false
```

CHALLENGES & SOLUTIONS

❖ **Deployment:**

- **Overcame issues with CI/CD setup and Airflow deployment.**
- **Overcame issues with Git version control**

❖ **Security:**

- **Overcame issue with secure handling of GCP credentials.**

Policy violation

Immediate action required: Resources associated with your Google Cloud Platform/API project data-eng-capstone-434311 are being restricted

Dear Developer,

We detected and will disable a publicly exposed service account authentication credential associated with the following Google Cloud Platform account:

deng-capstone-service-account@data-eng-capstone-434311.iam.gserviceaccount.com with key ID 7422a9a40dba7608e8d7b7ba303c18c8cc3b5f4e

This key was found at the following URL: https://github.com/Ameen1nl/project/blob/84cab7b558c50f5bbb282f97757f75a06c7d77a8/gcp-key.json

CONCLUSION

❖ **Summary:**

In conclusion, our analysis of customer data has revealed valuable insights into satisfaction levels, spending patterns, and membership behavior.

❖ **Recommended actions:**

- **Prioritize customer satisfaction.**
- **Optimize discount strategies.**
- **Enhance membership benefits.**
- **Segment customers.**

Q&A

Thank you for
your time and attention

Thank you