CUSTOMER SEGMENTATION ANALYSIS

DATA WAREHOUSING

CODE: 22IPE517

Submitted By,

Aswin S
71772218102
B.Tech. IT

Abstract:

This project aims to create a robust customer analytics system capable of generating insights for customer segmentation, churn prediction, and lifecycle value estimation. Data is sourced from multiple CRM and operational systems, aggregated into CSV files, and processed using a custom ETL pipeline. The data is then stored in a star schema-based PostgreSQL data warehouse. An analysis layer performs advanced computations and exports the results to CSV for visualization. Tableau dashboards provide intuitive visualizations for actionable insights. This system demonstrates the seamless integration of data engineering, analysis, and visualization to enhance customer understanding and drive data-driven business decisions.

Introduction:

Problem Statement:

Understanding customer behavior is a cornerstone for businesses seeking to enhance customer retention and drive growth. Despite its importance, many organizations lack a systematic approach to analyze customer data effectively, which prevents them from gaining actionable insights and tailoring their strategies.

Relevance:

Adopting data-driven methodologies allows businesses to segment customers accurately, anticipate churn risks, and compute customer lifecycle value. By employing advanced ETL pipelines and visualization tools, organizations can transform raw data into precise and actionable insights, ultimately enhancing decision-making processes and strategic planning.

Scope:

This project aims to build an end-to-end system that converts raw customer data into meaningful insights. It involves designing ETL pipelines for data processing, implementing a star schema-based data warehouse for efficient storage and querying, performing advanced analyses, and leveraging Tableau for intuitive data visualizations. Together, these components provide a comprehensive framework for customer analytics.

Objectives:

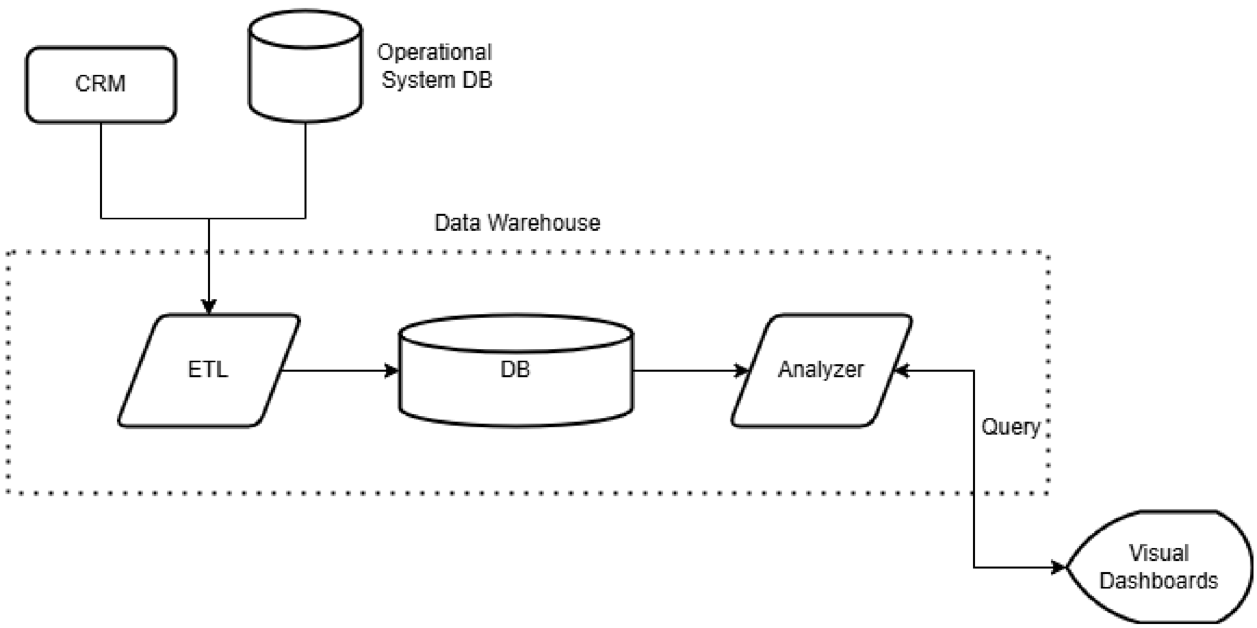
- Develop an ETL pipeline to transform raw CSV data into a clean, structured format.
- Implement a star schema data warehouse in PostgreSQL for efficient query performance.
- Perform advanced analysis for customer segmentation, churn prediction, and lifecycle value computation.
- Create dashboards using Tableau to visualize insights effectively.

System Architecture:

The system consists of four main components:

- **ETL Layer**: Processes raw data collected from multiple CRM and operational systems and loads it into the data warehouse.
- **Data Warehouse**: Stores processed data in a star schema for efficient querying and analysis.
- **Analysis Layer**: Extracts data from the warehouse for advanced computations.
- **Visualization Layer**: Uses Tableau to present the analysis results through interactive dashboards.

Architecture Diagram:



Methodology:

Data Collection:

Data for this project is sourced from various Customer Relationship Management (CRM) platforms and operational systems. These systems provide rich datasets, including customer demographics, transaction history, and engagement metrics. The data is exported in CSV format, which serves as the input for the ETL process.

Data Transformation:

The extracted CSV files undergo rigorous cleaning and transformation to ensure consistency and accuracy. This includes handling missing values, resolving inconsistencies, and normalizing data formats. For example:

- Standardizing date formats across all datasets.

- Converting categorical fields like customer_tier into numerical representations for better analysis.

Data Integration:

Transformed data from the CSV files is integrated into a star schema in the PostgreSQL data warehouse. The schema design prioritizes query performance and analytical efficiency, aligning with business intelligence best practices.

Analysis Techniques:

The analysis layer retrieves data from the warehouse and applies advanced statistical and machine learning models to derive insights such as:

- Customer segmentation based on demographic and transactional behaviors.
- Predicting churn likelihood using historical engagement patterns.
- Estimating customer lifecycle value through transactional trends.

Visualization:

Results from the analysis layer are exported into structured CSV files. Tableau consumes these outputs to create dynamic and interactive dashboards. These dashboards allow users to:

- Explore segmentation clusters visually.
- Track churn prediction trends.
- Analyze lifecycle value distributions across customer cohorts

Implementation Details:

ETL Process:

The ETL (Extract, Transform, Load) pipeline is the backbone of this project. It consists of:

1. Extraction: Reading raw data from CSV files provided by CRM and operational systems.
2. Transformation: Cleaning and enriching the data, including:
    - Removing duplicates.
    - Filling missing values with appropriate statistical imputation.
    - Encoding categorical variables for downstream processing.
3. Loading: Populating the star schema in the PostgreSQL database.

The ETL process is implemented using Python, leveraging libraries like Pandas for data manipulation and SQLAlchemy for database interaction.

Data Warehouse:

The PostgreSQL database is designed with a star schema structure. Key details include:

- **Fact Tables**: transactions and engagements contain the core transactional and behavioral data.

- **Dimension Tables**: customers and products store descriptive details supporting the fact tables.
- **Indexes and Constraints**: Implemented to optimize query performance and maintain data integrity.

Analysis Layer:

The analysis layer uses Python-based tools, including Pandas and NumPy, to perform complex calculations. For instance:

- Customer segmentation is achieved through k-means clustering.
- Churn prediction leverages logistic regression models.
- Lifecycle value computation uses cumulative revenue metrics.

Visualization Layer:

Interactive dashboards built in Tableau provide actionable insights. Key features include:

- Cluster analysis visualization to distinguish customer groups.
- Predictive dashboards highlighting churn risks.
- Lifecycle value heatmaps for strategic planning

Source Code:

DB Schema:

```
CREATE TABLE customer_tiers (
    tier_id SERIAL PRIMARY KEY,
    tier_name VARCHAR(20),
    discount_rate DECIMAL(5, 2)
);

CREATE TABLE product_categories (
    category_id SERIAL PRIMARY KEY,
    category_name VARCHAR(50)
);

CREATE TABLE payment_methods (
    payment_method_id SERIAL PRIMARY KEY,
    payment_method_name VARCHAR(50)
);

CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    phone_number VARCHAR(15),
    gender VARCHAR(10),
    dob DATE,
```

```
    age INTEGER,
    city VARCHAR(50),
    state VARCHAR(50),
    country VARCHAR(50),
    signup_date DATE,
    is_active BOOLEAN,
    customer_tier INTEGER REFERENCES customer_tiers(tier_id)
);

CREATE TABLE transactions (
    transaction_id SERIAL PRIMARY KEY,
    customer_id INTEGER REFERENCES customers(customer_id),
    transaction_date TIMESTAMP,
    amount DECIMAL(10, 2),
    payment_method INTEGER REFERENCES payment_methods(payment_method_id),
    product_id VARCHAR(50),
    product_category INTEGER REFERENCES product_categories(category_id),
    quantity INTEGER,
    discount_applied BOOLEAN,
    transaction_status VARCHAR(20)
);

CREATE TABLE engagements (
    engagement_id SERIAL PRIMARY KEY,
    customer_id INTEGER REFERENCES customers(customer_id),
    engagement_date DATE,
    login_frequency INTEGER,
    time_spent DECIMAL(5, 2),
    pages_visited INTEGER,
    purchase_clicks INTEGER,
    feedback_score INTEGER,
    email_open_rate DECIMAL(5, 2),
    promo_redemptions INTEGER
);
```

Analysis View

```
CREATE VIEW customer_summary AS
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    MAX(t.transaction_date) AS last_transaction_date,
    COUNT(t.transaction_id) AS total_transactions,
    SUM(t.amount) AS total_spent,
    AVG(t.amount) AS avg_transaction_amount,
    EXTRACT(DAY FROM '2021-12-31' - MAX(t.transaction_date)) AS recency,
    CASE
```

```sql
        WHEN EXTRACT(DAY FROM '2021-12-31' - MAX(t.transaction_date)) > 180 THEN
TRUE
        ELSE FALSE
    END AS churn_flag
FROM
    customers c
LEFT JOIN
    transactions t ON c.customer_id = t.customer_id
GROUP BY
    c.customer_id, c.first_name, c.last_name;
```

DB Models:

```python
from sqlalchemy import Column, Integer, String, Numeric, Boolean, Date, DateTime,
ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class CustomerTier(Base):
    __tablename__ = 'customer_tiers'

    tier_id = Column(Integer, primary_key=True, autoincrement=True)
    tier_name = Column(String(20), nullable=False, unique=True)
    discount_rate = Column(Numeric(5, 2), nullable=True)
    customers = relationship("Customer", back_populates="tier")

class ProductCategory(Base):
    __tablename__ = 'product_categories'

    category_id = Column(Integer, primary_key=True, autoincrement=True)
    category_name = Column(String(50), nullable=False, unique=True)

    transactions = relationship("Transaction", back_populates="category")

class PaymentMethod(Base):
    __tablename__ = 'payment_methods'

    payment_method_id = Column(Integer, primary_key=True, autoincrement=True)
    payment_method_name = Column(String(50), nullable=False, unique=True)

    transactions = relationship(
        "Transaction",
        back_populates="payment_method_rel",
        foreign_keys="[Transaction.payment_method]"
    )
```

```
class Customer(Base):
    __tablename__ = 'customers'

    customer_id = Column(Integer, primary_key=True, autoincrement=True)
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    email = Column(String(100), unique=True)
    phone_number = Column(String(15))
    gender = Column(String(10))
    dob = Column(Date)
    age = Column(Integer)
    city = Column(String(50))
    state = Column(String(50))
    country = Column(String(50))
    signup_date = Column(Date)
    is_active = Column(Boolean, default=True)

    customer_tier = Column(Integer, ForeignKey('customer_tiers.tier_id'))

    tier = relationship("CustomerTier", back_populates="customers")

    transactions = relationship("Transaction", back_populates="customer")
    engagements = relationship("Engagement", back_populates="customer")

class Transaction(Base):
    __tablename__ = 'transactions'

    transaction_id = Column(Integer, primary_key=True, autoincrement=True)

    customer_id = Column(Integer, ForeignKey('customers.customer_id'))
    payment_method = Column(Integer, ForeignKey('payment_methods.payment_method_id'))
    product_category = Column(Integer, ForeignKey('product_categories.category_id'))

    transaction_date = Column(DateTime)
    amount = Column(Numeric(10, 2))
    product_id = Column(String(50))
    quantity = Column(Integer)
    discount_applied = Column(Boolean)
    transaction_status = Column(String(20))

    customer = relationship("Customer", back_populates="transactions")
    payment_method_rel = relationship("PaymentMethod", back_populates="transactions")
    category = relationship("ProductCategory", back_populates="transactions")

class Engagement(Base):
```

```
    __tablename__ = 'engagements'

    engagement_id = Column(Integer, primary_key=True, autoincrement=True)
    customer_id = Column(Integer, ForeignKey('customers.customer_id'))
    engagement_date = Column(Date)
    login_frequency = Column(Integer)
    time_spent = Column(Numeric(5, 2))
    pages_visited = Column(Integer)
    purchase_clicks = Column(Integer)
    feedback_score = Column(Numeric(3, 1))
    email_open_rate = Column(Numeric(5, 2))
    promo_redemptions = Column(Integer)
    customer = relationship("Customer", back_populates="engagements")
```

ETL Layer:

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.dialects.postgresql import insert

from ..db_connection import DB_USER, DB_PASSWORD, DB_HOST, DB_PORT,
DB_NAME

from ..models import (
    Base,
    CustomerTier,
    PaymentMethod,
    ProductCategory,
    Customer,
    Transaction,
    Engagement
)

def create_db_engine():
    try:
        connection_string =
f'postgresql+psycopg2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_
NAME}'
        engine = create_engine(connection_string)
        return engine
    except Exception as error:
        print(f"Error creating database engine: {error}")
        return None

def get_or_create_reference_data(session, model, name_column, value):
    existing = session.query(model).filter(
```

```
        getattr(model, name_column) == value
    ).first()

    if existing:
        return existing

    new_record = model(**{name_column: value})
    session.add(new_record)
    session.commit()
    return new_record

def process_customers_data(session, df):
    for _, row in df.iterrows():
        tier = get_or_create_reference_data(
            session,
            CustomerTier,
            'tier_name',
            row['customer_tier']
        )

        customer = Customer(
            first_name=row['first_name'],
            last_name=row['last_name'],
            email=row['email'],
            phone_number=row['phone_number'],
            gender=row['gender'],
            dob=row['dob'],
            age=row['age'],
            city=row['city'],
            state=row['state'],
            country=row['country'],
            signup_date=row['signup_date'],
            is_active=row['is_active'],
            customer_tier=tier.tier_id
        )

        session.add(customer)

    session.commit()

def process_transactions_data(session, df):
    for _, row in df.iterrows():
        payment_method = get_or_create_reference_data(
            session,
            PaymentMethod,
            'payment_method_name',
```

```
            row['payment_method']
        )
        product_category = get_or_create_reference_data(
            session,
            ProductCategory,
            'category_name',
            row['product_category']
        )

        transaction = Transaction(
            customer_id=row['customer_id'],
            transaction_date=row['transaction_date'],
            amount=row['amount'],
            payment_method=payment_method.payment_method_id,
            product_id=row['product_id'],
            product_category=product_category.category_id,
            quantity=row['quantity'],
            discount_applied=row['discount_applied'],
            transaction_status=row['transaction_status']
        )

        session.add(transaction)

    session.commit()

def process_engagements_data(session, df):
    for _, row in df.iterrows():
        customer = session.query(Customer).filter_by(
            customer_id=row['customer_id']
        ).first()

        engagement = Engagement(
            customer_id=row['customer_id'],
            engagement_date=row['engagement_date'],
            login_frequency=row['login_frequency'],
            time_spent=row['time_spent'],
            pages_visited=row['pages_visited'],
            purchase_clicks=row['purchase_clicks'],
            feedback_score=row['feedback_score'],
            email_open_rate=row['email_open_rate'],
            promo_redemptions=row['promo_redemptions']
        )

        session.add(engagement)

    session.commit()
```

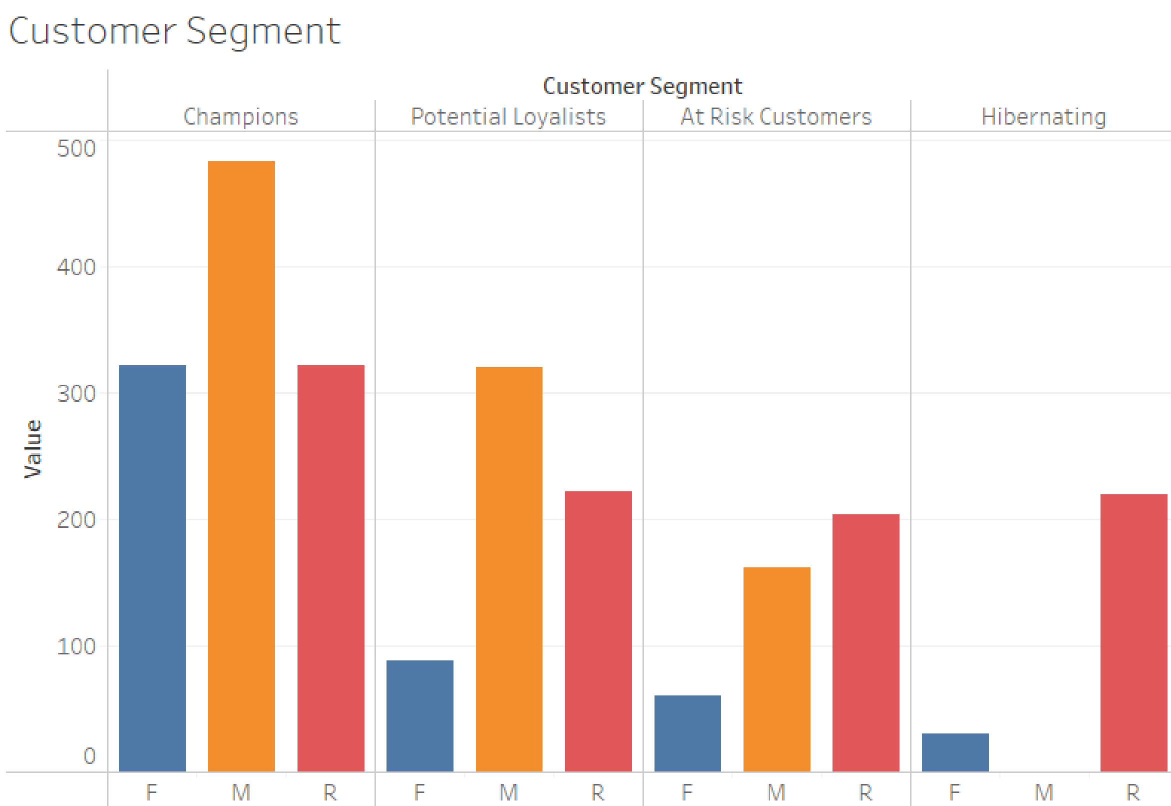
```
def etl_process(csv_file, process_function):
    try:
        engine = create_db_engine()
        if not engine:
            return
        Base.metadata.create_all(engine)
        Session = sessionmaker(bind=engine)
        session = Session()
                df = pd.read_csv(csv_file)
        process_function(session, df)

        session.close()

        print(f"Data loaded successfully from {csv_file}")

    except Exception as error:
        print(f"Error in ETL process: {error}")

def main():
    customers_file =
r'E:\Programs\CustomerSegmentation_DW\data\processed\cleaned_customers.csv'
    transactions_file =
r'E:\Programs\CustomerSegmentation_DW\data\processed\cleaned_transactions.csv'
    engagements_file =
r'E:\Programs\CustomerSegmentation_DW\data\processed\cleaned_engagements.csv'
    etl_process(customers_file, process_customers_data)
    etl_process(transactions_file, process_transactions_data)
    etl_process(engagements_file, process_engagements_data)

if __name__ == '__main__':
    main()
```

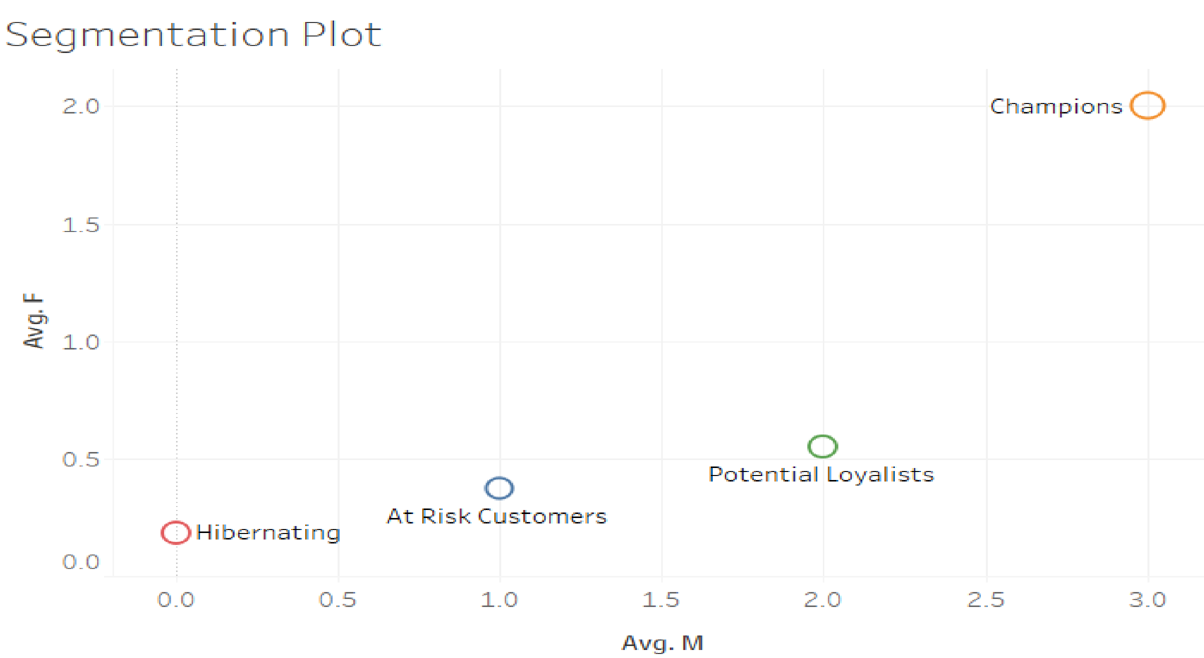
Analysis Layer:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
from sqlalchemy import create_engine
from ..db_connection import DB_USER, DB_PASSWORD, DB_HOST, DB_PORT,
DB_NAME
```

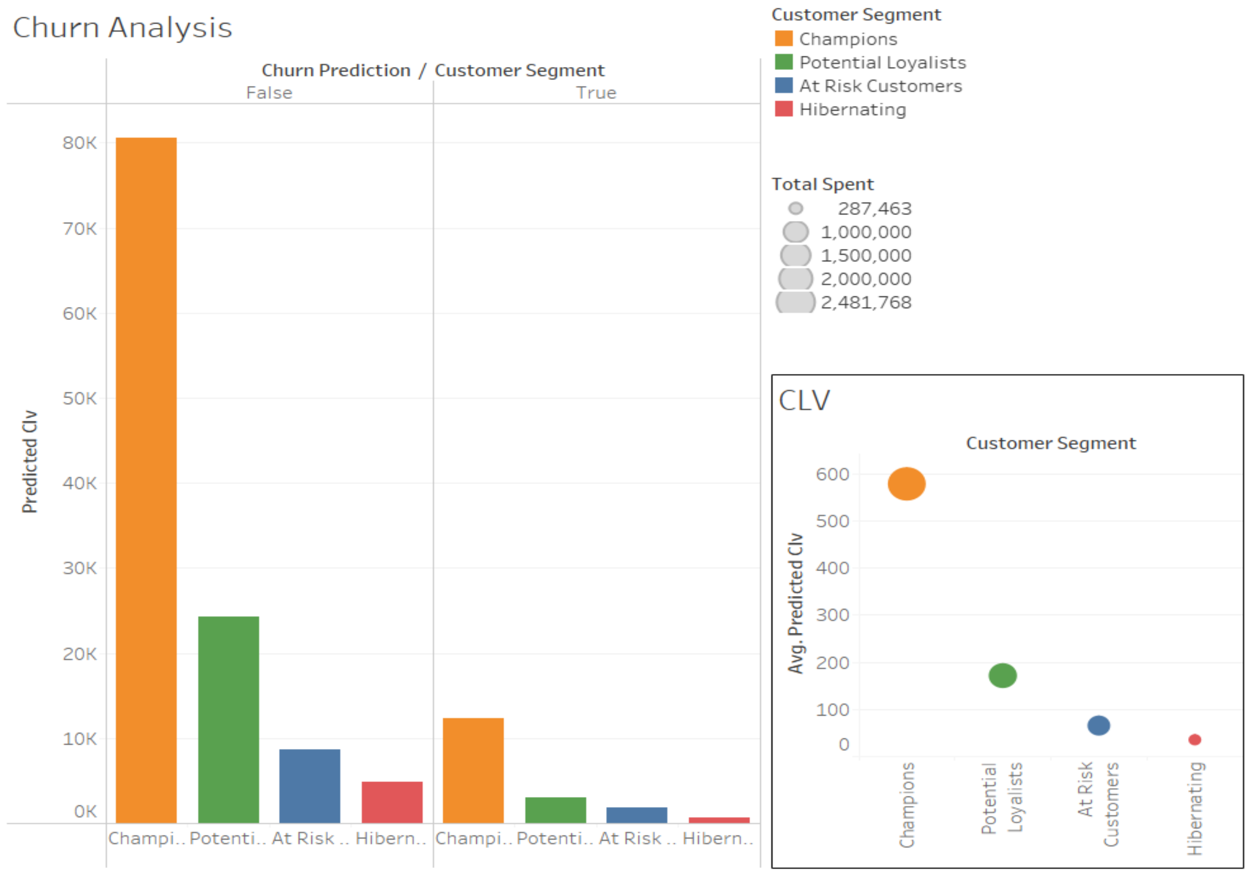
```
engine =
create_engine(f'postgresql+psycopg2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_
PORT}/{DB_NAME}')

df = pd.read_sql("SELECT * FROM customer_summary", engine)
df['R'] = pd.qcut(df['recency'], 4, labels=[3, 2, 1, 0], duplicates="drop")  # Higher recency =
lower score
df['F'] = pd.qcut(df['total_transactions'], 4, labels=[0, 1, 2], duplicates="drop")  # Higher
frequency = higher score
df['M'] = pd.qcut(df['total_spent'], 4, labels=[0, 1, 2, 3], duplicates="drop")  # Higher monetary
= higher score
df['RFM_Score'] = df['R'].astype(str) + df['F'].astype(str) + df['M'].astype(str)

def rfm_level(df):
    if df['RFM_Score'][1:] in ['33', '23', '13', '03']:
        return 'Champions'
    elif df['RFM_Score'][1:] in ['32', '22', '12', '02']:
        return 'Potential Loyalists'
    elif df['RFM_Score'][1:] in ['31', '21', '11', '01']:
        return 'At Risk Customers'
    elif df['RFM_Score'][1:] in ['30', '20', '10', '00']:
        return 'Hibernating'
    else:
        return 'Other'

df['Customer_Segment'] = df.apply(rfm_level, axis=1)
X = df[['recency', 'total_transactions', 'total_spent']]
y = df['churn_flag']
imputer = SimpleImputer(strategy='mean')  # You can also use 'median' or 'most_frequent'
X = imputer.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Churn Prediction Model Accuracy: {accuracy * 100:.2f}%")

df['churn_prediction'] = model.predict(X)
df['predicted_clv'] = df['total_spent'] * (df['total_transactions'] / (df['recency'] + 1))

df = df.dropna()
clv_csv_path =
r'E:\Programs\CustomerSegmentation_DW\src\visualizer\exports\clv_estimations.csv'
df.to_csv(clv_csv_path, index=False)
print(f"CLV estimations saved to {clv_csv_path}")
```

Visualizations:
Customer Segments:





Churn Prediction and CLV:



Results and Visualizations:

Segmentation Insights:

Tableau dashboards reveal distinct customer clusters based on demographics and transaction behavior. For example:

- High-value customers with frequent transactions.
- Low-engagement customers with sporadic purchases.

Churn Prediction:

Predictive models achieve high accuracy in identifying churn risks. Dashboards provide a visual representation of churn likelihood across customer segments, aiding in targeted retention strategies.

Lifecycle Value Estimation:

Visualizations highlight trends in customer lifetime value (CLV), enabling businesses to identify high-value customers and allocate resources effectively.

Challenges and Limitations:

Challenges:

- Data Quality Issues: Handling missing and inconsistent data required extensive preprocessing.
- Schema Design: Optimizing the star schema for diverse analytical queries was non-trivial.
- ETL Automation: Ensuring the robustness of the ETL process for large datasets posed challenges.

Limitations:

- The system relies on periodic CSV exports, which might not reflect real-time data changes.
- Predictive models require continuous refinement for accuracy improvements.

Future Scope:

- Real-time Data Integration: Implementing APIs for real-time data updates from CRM systems.
- Advanced Analytics: Incorporating deep learning models for enhanced predictive capabilities.
- Scalability: Transitioning to cloud-based architectures for handling larger datasets and concurrent users.

Appendix:

Github Source: https://github.com/I-am-Aswin/Customer_Segmentation_and_Analysis_DW.git