From the Code LookALikeModel.iypnb code

```
From sklearn.preprocessing import StandardScaler

# Specify the correct numerical columns for scaling
numerical_cols = ['TotalValue', 'TransactionID', 'Price_y']

# Standardize numerical columns
scaler = StandardScaler()
customer_profiles[numerical_cols] = scaler.fit_transform(customer_profiles[numerical_cols])

print(customer_profiles.head())
from sklearn.preprocessing import StandardScaler

# Specify the correct numerical columns for scaling
numerical_cols = ['TotalValue', 'TransactionID', 'Price_y']

# Standardize numerical columns
scaler = StandardScaler()
customer_profiles[numerical_cols] = scaler.fit_transform(customer_profiles[numerical_cols])

print(customer_profiles.head())
from sklearn.preprocessing import StandardScaler

# Specify the correct numerical columns for scaling
numerical_cols = ['TotalValue', 'TransactionID', 'Price_y']

# Standardize numerical columns
scaler = StandardScaler()
customer_profiles[numerical_cols] = scaler.fit_transform(customer_profiles[numerical_cols])

print(customer_profiles.head())
from sklearn.preprocessing import StandardScaler
```

```
# Specify the correct numerical columns for scaling
numerical_cols = ['TotalValue', 'TransactionID', 'Price_y']

# Standardize numerical columns
scaler = StandardScaler()
customer_profiles[numerical_cols] = scaler.fit_transform(customer_profiles[numerical_cols])

print(customer_profiles.head())

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler

# Preprocessing: Merge customer and transaction data
customer_data = pd.merge(transactions, customers, on="CustomerID", how="left")
customer_data = customer_data.groupby('CustomerID').agg({
    'Price': 'mean',
    'Quantity': 'sum',
    'TotalValue': 'sum',
    'Region': 'first'
}).reset_index()

# Feature engineering: Normalize numerical features
scaler = StandardScaler()
customer_data[['Price', 'Quantity', 'TotalValue']] = scaler.fit_transform(customer_data[['Price', 'Quantity',
'TotalValue']])

# Create a function to calculate similarity
def get_similar_customers(input_customer_id, customer_data, top_n=3):
    # Select the input customer's data based on CustomerID
    input_customer = customer_data[customer_data['CustomerID'] == input_customer_id][['Price', 'Quantity',
'TotalValue']]

    # Calculate cosine similarity using only numerical features
    similarities = cosine_similarity(input_customer, customer_data[['Price', 'Quantity', 'TotalValue']])
    similarity_scores = similarities.flatten()

    # Get top N most similar customers
    similar_customers_idx = similarity_scores.argsort()[-top_n:][::-1]
    similar_customers = customer_data.iloc[similar_customers_idx]
    return similar_customers[['CustomerID', 'Region']], similarity_scores[similar_customers_idx]

# Generate Lookalike for the first 20 customers
lookalike_results = {}
for cust_id in customer_data['CustomerID'][:20]:
    # Pass only the CustomerID to the function
    similar_customers, similarity_scores = get_similar_customers(cust_id, customer_data)
    lookalike_results[cust_id] = list(zip(similar_customers['CustomerID'], similarity_scores))

# Save the lookalike results to a CSV
lookalike_df = pd.DataFrame.from_dict(lookalike_results, orient='index')
lookalike_df.to_csv("FirstName_LastName_Lookalike.csv", header=False)
```

Code Explanation:

1. Standardizing Numerical Columns:

python

```
from sklearn.preprocessing import StandardScaler

# Specify the correct numerical columns for scaling
numerical_cols = ['TotalValue', 'TransactionID', 'Price_y']

# Standardize numerical columns
scaler = StandardScaler()
customer_profiles[numerical_cols] = scaler.fit_transform(customer_profiles[numerical_cols])

print(customer_profiles.head())
```

This block of code imports the StandardScaler from sklearn.preprocessing and standardizes the numerical columns TotalValue, TransactionID, and Price_y in the customer_profiles DataFrame. Standardization ensures that the features have a mean of 0 and a standard deviation of 1, which is useful for many machine learning algorithms.

2. Preprocessing and Aggregating Data:

python

```
from sklearn.metrics.pairwise import cosine_similarity

from sklearn.preprocessing import StandardScaler


# Preprocessing: Merge customer and transaction data

customer_data = pd.merge(transactions, customers, on="CustomerID", how="left")

customer_data = customer_data.groupby('CustomerID').agg({

    'Price': 'mean',

    'Quantity': 'sum',

    'TotalValue': 'sum',

    'Region': 'first'

}).reset_index()
```

This block merges transaction and customer data on CustomerID, and then aggregates the data by CustomerID to calculate the mean price, total quantity, and total value of transactions per customer. It also retains the region information.

3. Normalizing Numerical Features:

python

```
# Feature engineering: Normalize numerical features

scaler = StandardScaler()

customer_data[['Price', 'Quantity', 'TotalValue']] = scaler.fit_transform(customer_data[['Price', 'Quantity', 'TotalValue']])
```

Similar to the first block, this code normalizes the numerical features in customer_data using StandardScaler.

4. Calculating Cosine Similarity:

python

```
# Create a function to calculate similarity

def get_similar_customers(input_customer_id, customer_data, top_n=3):

    # Select the input customer's data based on CustomerID

    input_customer = customer_data[customer_data['CustomerID'] == input_customer_id][['Price', 'Quantity', 'TotalValue']]


    # Calculate cosine similarity using only numerical features

    similarities = cosine_similarity(input_customer, customer_data[['Price', 'Quantity', 'TotalValue']])

    similarity_scores = similarities.flatten()
    # Get top N most similar customers
    similar_customers_idx = similarity_scores.argsort()[-top_n:][::-1]
    similar_customers = customer_data.iloc[similar_customers_idx]
    return similar_customers[['CustomerID', 'Region']], similarity_scores[similar_customers_idx]
```

This function calculates the cosine similarity between a given customer and all other customers based on the numerical features Price, Quantity, and TotalValue. It then returns the top N most similar customers.

5. Generating Lookalike Results:

python

```
# Generate Lookalike for the first 20 customers
lookalike_results = {}
for cust_id in customer_data['CustomerID'][:20]:
    # Pass only the CustomerID to the function
    similar_customers, similarity_scores = get_similar_customers(cust_id, customer_data)
    lookalike_results[cust_id] = list(zip(similar_customers['CustomerID'], similarity_scores))
```

This loop generates lookalike results for the first 20 customers by finding the top 3 most similar customers for each of them using the get_similar_customers function.

6. Saving Results:

python

```
# Save the lookalike results to a CSV
lookalike_df = pd.DataFrame.from_dict(lookalike_results, orient='index')
lookalike_df.to_csv("FirstName_LastName_Lookalike.csv", header=False)
```

This code saves the lookalike results to a CSV file.

Insights:

1. **Customer Segmentation**: By identifying customers who are similar based on purchasing behavior and other features, you can create more targeted marketing campaigns. Customers with similar profiles might respond better to similar offers.
2. **Personalized Recommendations**: The lookalike results can be used to recommend products to new or existing customers based on what similar customers have purchased.
3. **Improved Customer Retention**: Understanding customer similarities can help in developing strategies to retain high-value customers by offering them personalized services and incentives.
4. **Identifying Trends**: Analyzing similarities among customers can reveal trends in purchasing behavior, which can inform inventory management, product development, and pricing strategies.