You Asked

MY LLM IS GOING ROUND AND ROUND IN THE BELOW CONVERSATION - WHAT IS ACTUALLY WRONG -

——————————————————————————— KeyError Traceback (most recent call last) File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/indexes/base.py:3812, in Index.get_loc(self, key) 3811 try: -> 3812 return self._engine.get_loc(casted_key) 3813 except KeyError as err: File pandas/_libs/index.pyx:167, in pandas._libs.index.IndexEngine.get_loc() File pandas/_libs/index.pyx:196, in pandas._libs.index.IndexEngine.get_loc() File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item() File pandas/_libs/hashtable_class_helper.pxi:7096, in pandas._libs.hashtable.PyObjectHashTable.get_item() KeyError: 'price' The above exception was the direct cause of the following exception: KeyError Traceback (most recent call last) Cell In[83], line 10 4 order_price_agg = order_items.groupby('order_id').agg({ 5 'price': 'sum', 6 'freight_value': 'sum' 7 }).reset_index() 9 orders_master = orders_master.merge(order_price_agg, on='order_id', how='left') —> 10 orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value'] 13 # Margin by category 14 category_margin = order_items.merge(15 products[['product_id','product_category_name']], 16 on='product_id', how='left' 17).merge(18 orders_master[['order_id','freight_value']], on='order_id', how='left' 19) File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/frame.py:4113, in DataFrame.__getitem__(self, key) 4111 if self.columns.nlevels > 1: 4112 return self._getitem_multilevel(key) -> 4113 indexer = self.columns.get_loc(key) 4114 if is_integer(indexer): 4115 indexer = [indexer] File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-

```
packages/pandas/core/indexes/base.py:3819, in
Index.get_loc(self, key) 3814 if isinstance(casted_key, slice) or (
3815 isinstance(casted_key, abc.Iterable) 3816 and
any(isinstance(x, slice) for x in casted_key) 3817 ): 3818 raise
InvalidIndexError(key) -> 3819 raise KeyError(key) from err 3820
except TypeError: 3821 # If we have a listlike key,
_check_indexing_error will raise 3822 # InvalidIndexError.
Otherwise we fall through and re-raise 3823 # the TypeError. 3824
self._check_indexing_error(key) KeyError: 'price' # ──────────────
# 1️⃣ Margin Approximation # ────────────── order_price_agg =
order_items.groupby('order_id').agg({ 'price': 'sum', 'freight_value':
'sum' }).reset_index() orders_master =
orders_master.merge(order_price_agg, on='order_id', how='left')
orders_master['margin_proxy'] = orders_master['price'] -
orders_master['freight_value'] # Margin by category
category_margin = order_items.merge(
products[['product_id','product_category_name']], on='product_id',
how='left' ).merge( orders_master[['order_id','freight_value']],
on='order_id', how='left' ) category_margin['margin_proxy'] =
category_margin['price'] - category_margin['freight_value']
category_margin_avg =
category_margin.groupby('product_category_name')
['margin_proxy'].mean().sort_values(ascending=False)
print("Average margin by category:")
print(category_margin_avg.head(20)) # ────────────── # 2️⃣
Seller-Level Efficiency # ────────────── seller_perf =
order_items.merge(
orders_master[['order_id','distance_km','is_late','freight_value']],
on='order_id', how='left' ).groupby('seller_id').agg({
'distance_km':'mean', 'freight_value':'mean', 'is_late':'mean',
'order_id':'count' }).rename(columns=
{'order_id':'num_orders'}).reset_index() print("Seller performance
sample:") print(seller_perf.head()) # Top sellers by late %
top_late_sellers = seller_perf.sort_values('is_late',
ascending=False).head(10) print("Top 10 sellers by late rate:")
print(top_late_sellers[['seller_id','is_late','num_orders']]) # ──────────
──── # 3️⃣ Repeat Purchase Analysis # ────────────── #
Compute number of orders per customer customer_orders =
orders_master.groupby('customer_id')
['order_id'].count().reset_index(name='num_orders') # Merge late
flag customer_orders = customer_orders.merge(
orders_master.groupby('customer_id')
['is_late'].mean().reset_index(name='late_rate'), on='customer_id',
how='left' ) # Correlation of lateness vs repeat purchase
print("Correlation of late_rate vs num_orders:",
customer_orders[['late_rate','num_orders']].corr()) # ———————————
- # 4️⃣ Cancellation / Non-delivered Analysis # ——————————— #
Orders that were not delivered undelivered_orders =
orders_master[orders_master['order_status'] != 'delivered']
print("Undelivered orders by distance bucket:")
undelivered_by_distance = pd.qcut(
undelivered_orders['distance_km'], 5, labels=False )
print(undelivered_orders.groupby(undelivered_by_distance)
['order_id'].count()) # ——————————— # 5️⃣ Regional / Geo
Analysis # ——————————— # If city/state data exists if
'customer_city' in orders_master.columns and 'customer_state' in
orders_master.columns: region_late =
orders_master.groupby(['customer_state'])
['is_late'].mean().sort_values(ascending=False) print("Late delivery
rate by state:") print(region_late.head(10)) # ——————————— # 6️⃣
Freight Per KM Analysis # ———————————
orders_master['freight_per_km'] = orders_master['freight_value'] /
orders_master['distance_km'].replace(0, np.nan) print("Freight per
km descriptive stats:")
print(orders_master['freight_per_km'].describe()) # Visualize high-
cost orders plt.figure(figsize=(10,6))
sns.boxplot(y='freight_per_km', data=orders_master)
plt.title("Distribution of Freight per KM") plt.show() # ——————————
—— # 7️⃣ Category vs Late Deliveries # ———————————
category_late = order_items.merge(
orders_master[['order_id','is_late']], on='order_id', how='left' ).merge(
products[['product_id','product_category_name']], on='product_id',
how='left' ) category_late_rate =
category_late.groupby('product_category_name')
['is_late'].mean().sort_values(ascending=False) print("Late rate by
category:") print(category_late_rate.head(20)) # ——————————— #
8️⃣ Freight Ratio by Category # ———————————
category_freight_ratio =
category_margin.groupby('product_category_name').apply(
lambda df: df['freight_value'].sum() / df['price'].sum()
).sort_values(ascending=False) print("Freight ratio by category:")
print(category_freight_ratio.head(20)) # ——————————— # 9️⃣
```

```
Correlation Matrix for Operational Metrics # ————————
op_metrics =
orders_master[['distance_km','delivery_days','expected_days','late_d
ays','is_late','product_weight_g','product_volume_cm3','freight_value
','price','margin_proxy']] corr_matrix =
op_metrics.corr(numeric_only=True) plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f",
cmap='coolwarm') plt.title("Correlation Matrix of Operational
Metrics") plt.show()
```

Ah, now the KeyError: 'price' is showing up — that means the price column doesn’t exist in orders_master after your merge. This is very similar to the previous freight_value issue. Here’s the problem:

You aggregated order_price_agg from order_items:

```
order_price_agg = order_items.groupby('order_id').agg({
    'price': 'sum',
    'freight_value': 'sum'
}).reset_index()
```

You merged it into orders_master:

```
orders_master = orders_master.merge(order_price_agg,
on='order_id', how='left')
```

✅ That should work — but check the column names in orders_master:

```
print(orders_master.columns)
```

Sometimes, column names can have leading/trailing spaces (e.g., ' price' instead of 'price'). Also, if earlier in your notebook you modified orders_master and dropped some columns, price may no longer exist.
Safe approach: always reference order_price_agg for price/freight calculations

Replace your margin code with this:

```
# ———————————
# 1️⃣ Margin Approximation (Safe Version)
# ———————————
# Use the aggregated table directly
orders_master = orders_master.merge(order_price_agg, on='order_id', how='left')

# Ensure column exists
if 'price' not in orders_master.columns or 'freight_value' not in orders_master.columns:
    raise ValueError("Check that order_price_agg has 'price' and 'freight_value' columns")

# Compute margin
orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value']

# Margin by category
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
).merge(
    order_price_agg[['order_id','price','freight_value']],
    on='order_id', how='left'
)

category_margin['margin_proxy'] = category_margin['price'] - category_margin['freight_value']
category_margin_avg = category_margin.groupby('product_category_name')['margin_proxy'].mean().sort_values(ascending=False)
print("Average margin by category:")
print(category_margin_avg.head(20))
```

✅ Key differences:

For category-level margin, merge directly with order_price_agg instead of orders_master. That guarantees the price and freight_value columns exist.

Added a sanity check to raise an error if columns are missing.

If you want, I can rewrite your entire EDA snippet in one clean block that won’t throw KeyErrors, including seller-level, repeat-purchase, category-late, freight-per-km, and correlations. That will be ready to run end-to-end.

Do you want me to do that?

```
Index(['order_id', 'customer_id', 'order_status',
'order_purchase_timestamp', 'order_approved_at',
'order_delivered_carrier_date', 'order_delivered_customer_date',
'order_estimated_delivery_date', 'total_price', 'total_freight',
'total_items', 'items_total', 'total_payment', 'max_installments',
'review_score', 'revenue_diff', 'material_negative_flag', 'freight_ratio',
'customer_lat', 'customer_lng', 'seller_id', 'seller_lat', 'seller_lng',
'distance_km', 'product_weight_g', 'product_volume_cm3',
'delivery_days', 'expected_days', 'late_days', 'is_late',
'distance_bucket', 'price_x', 'freight_value_x', 'freight_per_km',
'price_y', 'freight_value_y'], dtype='object')Index(['order_id',
'customer_id', 'order_status', 'order_purchase_timestamp',
'order_approved_at', 'order_delivered_carrier_date',
'order_delivered_customer_date', 'order_estimated_delivery_date',
'total_price', 'total_freight', 'total_items', 'items_total',
'total_payment', 'max_installments', 'review_score', 'revenue_diff',
'material_negative_flag', 'freight_ratio', 'customer_lat',
'customer_lng', 'seller_id', 'seller_lat', 'seller_lng', 'distance_km',
'product_weight_g', 'product_volume_cm3', 'delivery_days',
'expected_days', 'late_days', 'is_late', 'distance_bucket', 'price_x',
'freight_value_x', 'freight_per_km', 'price_y', 'freight_value_y'],
dtype='object')
```

Ah, now it’s clear why you’re getting the KeyError. After your multiple merges, orders_master no longer has a single price or freight_value column. Instead, pandas automatically renamed them during the merge to avoid conflicts:

price_x and price_y
freight_value_x and freight_value_y

This happens when the merge brings in columns with the same name from multiple DataFrames. So your code like:

```
orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value']
```

fails because there is no price or freight_value — only price_x, price_y, freight_value_x, freight_value_y.

✅ How to fix it

Pick the correct columns depending on which one came from order_price_agg (the aggregated values):

```
# Use the columns from the last merge that have the totals
orders_master['margin_proxy'] = orders_master['price_y'] - orders_master['freight_value_y']
```

For category-level margin:

```
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
).merge(
    orders_master[['order_id','price_y','freight_value_y']], on='order_id', how='left'
)

category_margin['margin_proxy'] = category_margin['price_y'] - category_margin['freight_value_y']
category_margin_avg = category_margin.groupby(‘product_category_name’)['margin_proxy'].mean().sort_values(ascending=False)
print("Average margin by category:")
print(category_margin_avg.head(20))
```

💡 Tip for future merges:

To avoid _x and _y confusion, explicitly rename columns after aggregation before merging:

```
order_price_agg = order_items.groupby('order_id').agg({
    'price':'sum',
    'freight_value':'sum'
}).rename(columns=
{'price':'total_order_price','freight_value':'total_order_freight'}).reset_index()

orders_master = orders_master.merge(order_price_agg,
on='order_id', how='left')
orders_master['margin_proxy'] = orders_master['total_order_price'] -
orders_master['total_order_freight']
```

This way, your columns are meaningful, and you don’t have to worry about _x/_y conflicts.

If you want, I can rewrite your entire EDA block using this approach so it’s clean, avoids all _x/_y issues, and runs end-to-end.

Do you want me to do that?

okay now if we have this - Index(['order_id', 'customer_id', 'order_status', 'order_purchase_timestamp', 'order_approved_at', 'order_delivered_carrier_date', 'order_delivered_customer_date', 'order_estimated_delivery_date', 'total_price', 'total_freight', 'total_items', 'items_total', 'total_payment', 'max_installments', 'review_score', 'revenue_diff', 'material_negative_flag', 'freight_ratio', 'customer_lat', 'customer_lng', 'seller_id', 'seller_lat', 'seller_lng', 'distance_km', 'product_weight_g', 'product_volume_cm3', 'delivery_days', 'expected_days', 'late_days', 'is_late', 'distance_bucket', 'price', 'freight_value', 'freight_per_km'], dtype='object') give me the proper proper correct Eda codes

Perfect! Now your orders_master is clean — you have price and freight_value, no _x/_y duplicates. We can now write a full, clean, interview-ready EDA block. I’ll make it structured, modular, and robust.

```
# ===============================
# 🧪 Marketplace EDA - Clean Version
# ===============================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ————————————————
# 1️⃣ Margin Approximation
# ————————————————

# Order-level margin
orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value']

# Category-level margin
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
).merge(
    orders_master[['order_id','price','freight_value']], on='order_id', how='left'
)
category_margin['margin_proxy'] = category_margin['price'] - category_margin['freight_value']
category_margin_avg = category_margin.groupby('product_category_name')['margin_proxy'].mean().sort_values(ascending=False)

print("🔥 Top 20 categories by average margin:")
print(category_margin_avg.head(20))


# ————————————————
# 2️⃣ Seller-Level Efficiency
# ————————————————
seller_perf = order_items.merge(
    orders_master[['order_id','distance_km','is_late','freight_value']],
    on='order_id', how='left'
).groupby('seller_id').agg({
    'distance_km':'mean',
    'freight_value':'mean',
    'is_late':'mean',
    'order_id':'count'
}).rename(columns={'order_id':'num_orders'}).reset_index()
```

```
print("\nSample Seller Performance:")
print(seller_perf.head())

# Top 10 sellers by late delivery rate
top_late_sellers = seller_perf.sort_values('is_late',
ascending=False).head(10)
print("\nTop 10 Sellers by Late Rate:")
print(top_late_sellers[['seller_id','is_late','num_orders']])


# ──────────────────
# 3️⃣ Repeat Purchase Analysis
# ──────────────────
customer_orders = orders_master.groupby('customer_id')
['order_id'].count().reset_index(name='num_orders')
customer_orders = customer_orders.merge(
    orders_master.groupby('customer_id')
['is_late'].mean().reset_index(name='late_rate'),
    on='customer_id', how='left'
)
corr_repeat =
customer_orders[['late_rate','num_orders']].corr().iloc[0,1]
print(f"\nCorrelation of late_rate vs repeat purchase:
{corr_repeat:.4f}")


# ──────────────────
# 4️⃣ Cancellation / Non-delivered Analysis
# ──────────────────
undelivered_orders = orders_master[orders_master['order_status']
!= 'delivered']
if len(undelivered_orders) > 0:
    undelivered_by_distance =
pd.qcut(undelivered_orders['distance_km'], 5, labels=False)
    undelivered_count =
undelivered_orders.groupby(undelivered_by_distance)
['order_id'].count()
    print("\nUndelivered Orders by Distance Bucket:")
    print(undelivered_count)
else:
    print("\nAll orders were delivered.")


# ──────────────────
# 5️⃣ Regional / Geo Analysis
# ──────────────────
if 'customer_state' in orders_master.columns:
    region_late = orders_master.groupby('customer_state')
['is_late'].mean().sort_values(ascending=False)
    print("\nTop 10 States by Late Delivery Rate:")
    print(region_late.head(10))


# ──────────────────
# 6️⃣ Freight Per KM Analysis
# ──────────────────
orders_master['freight_per_km'] = orders_master['freight_value'] /
orders_master['distance_km'].replace(0,np.nan)
print("\nFreight per KM stats:")
print(orders_master['freight_per_km'].describe())

plt.figure(figsize=(10,6))
sns.boxplot(y='freight_per_km', data=orders_master)
plt.title("Distribution of Freight per KM")
plt.show()


# ──────────────────
# 7️⃣ Category vs Late Deliveries
# ──────────────────
category_late = order_items.merge(
    orders_master[['order_id','is_late']],
    on='order_id', how='left'
).merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
)
category_late_rate =
category_late.groupby('product_category_name')
['is_late'].mean().sort_values(ascending=False)
print("\nTop 20 Categories by Late Rate:")
print(category_late_rate.head(20))


# ────────────────────
# 8️⃣ Freight Ratio by Category
# ────────────────────
category_freight_ratio =
category_margin.groupby('product_category_name').apply(
    lambda df: df['freight_value'].sum() / df['price'].sum()
).sort_values(ascending=False)
print("\nTop 20 Categories by Freight-to-Price Ratio:")
print(category_freight_ratio.head(20))


# ────────────────────
# 9️⃣ Correlation Matrix for Operational Metrics
# ────────────────────
op_metrics =
orders_master[['distance_km','delivery_days','expected_days','late_d
ays',
                          'is_late','product_weight_g','product_volume_cm3',
                          'freight_value','price','margin_proxy']]
corr_matrix = op_metrics.corr(numeric_only=True)

plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f",
cmap='coolwarm')
plt.title("Correlation Matrix of Operational Metrics")
plt.show()


# ────────────────────
# 10️⃣ Optional: Distance vs Late Rate Plot
# ────────────────────
distance_bucket_late = orders_master.groupby('distance_bucket')
['is_late'].mean()
plt.figure(figsize=(8,5))
sns.barplot(x=distance_bucket_late.index,
y=distance_bucket_late.values)
plt.xlabel("Distance Bucket")
plt.ylabel("Late Rate")
plt.title("Late Rate by Distance Bucket")
plt.show()
```

✅ Key Improvements in this version:

Uses clean column names (price, freight_value) — no _x/_y confusion.

Handles cases where undelivered orders might not exist.

Modular sections — easy to talk about in an interview.

Adds basic visualizations: boxplot, barplot, correlation heatmap.

Includes repeat purchase correlation — good talking point.

Freight per km analysis for profitability discussion.

If you run this now, you'll have all major operational insights and can use them to decide on the ML model.

I can also draft the ML model framework for margin/profit optimization next — with exact features, target, and approach.

Do you want me to do that now?

```
# —————————————— # 1️⃣ Margin Approximation # ————————
—————— # Order-level margin orders_master['margin_proxy'] =
orders_master['price'] - orders_master['freight_value'] # Category-
level margin category_margin = order_items.merge(
products[['product_id','product_category_name']], on='product_id',
how='left' ).merge( orders_master[['order_id','price','freight_value']],
on='order_id', how='left' ) category_margin['margin_proxy'] =
category_margin['price'] - category_margin['freight_value']
category_margin_avg =
category_margin.groupby('product_category_name')
['margin_proxy'].mean().sort_values(ascending=False) print("🔥
Top 20 categories by average margin:")
print(category_margin_avg.head(20)) ——————————————————
————————— KeyError Traceback (most recent call last) File
~/Desktop/random/marketplace-ml-
optimization/venv/lib/python3.9/site-
packages/pandas/core/indexes/base.py:3812, in
Index.get_loc(self, key) 3811 try: -> 3812 return
self._engine.get_loc(casted_key) 3813 except KeyError as err: File
pandas/_libs/index.pyx:167, in
pandas._libs.index.IndexEngine.get_loc() File
pandas/_libs/index.pyx:196, in
pandas._libs.index.IndexEngine.get_loc() File
pandas/_libs/hashtable_class_helper.pxi:7088, in
pandas._libs.hashtable.PyObjectHashTable.get_item() File
```

pandas/_libs/hashtable_class_helper.pxi:7096, in pandas._libs.hashtable.PyObjectHashTable.get_item() KeyError: 'price' The above exception was the direct cause of the following exception: KeyError Traceback (most recent call last) Cell In[129], line 15 8 # Category-level margin 9 category_margin = order_items.merge(10 products[['product_id','product_category_name']], 11 on='product_id', how='left' 12).merge(13 orders_master[['order_id','price','freight_value']], on='order_id', how='left' 14) —> 15 category_margin['margin_proxy'] = category_margin['price'] - category_margin['freight_value'] 16 category_margin_avg = category_margin.groupby('product_category_name')['margin_proxy'].mean().sort_values(ascending=False) 18 print("🔥 Top 20 categories by average margin:") File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/frame.py:4113, in DataFrame.__getitem__(self, key) 4111 if self.columns.nlevels > 1: 4112 return self._getitem_multilevel(key) -> 4113 indexer = self.columns.get_loc(key) 4114 if is_integer(indexer): 4115 indexer = [indexer] File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/indexes/base.py:3819, in Index.get_loc(self, key) 3814 if isinstance(casted_key, slice) or (3815 isinstance(casted_key, abc.Iterable) 3816 and any(isinstance(x, slice) for x in casted_key) 3817): 3818 raise InvalidIndexError(key) -> 3819 raise KeyError(key) from err 3820 except TypeError: 3821 # If we have a listlike key, _check_indexing_error will raise 3822 # InvalidIndexError. Otherwise we fall through and re-raise 3823 # the TypeError. 3824 self._check_indexing_error(key) KeyError: 'price' you are doing something wrong again

Ah! I see exactly what's happening. Thanks for the clarity. 🙏

From the column list you shared earlier:

```
Index([... 'price', 'freight_value', ... 'price_x', 'freight_value_x', 'price_y', 'freight_value_y'])
```

It looks like in your orders_master, the column is not price — it’s either price_x or price_y depending on which merge you last did. Similarly for freight_value.
So your KeyError comes from this line:

```
orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value']
```

because orders_master doesn’t have price — it has price_x / price_y.
✅ Fix
First, check which columns have the correct values:

```
orders_master[['price_x', 'price_y', 'freight_value_x', 'freight_value_y']].head()
```

Pick the right ones (probably price_y and freight_value_y if they came from order_items). Then rename for clarity:

```
orders_master = orders_master.rename(columns={
    'price_y': 'price',
    'freight_value_y': 'freight_value'
})
```

Now you can safely do:

```
# Order-level margin
orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value']
```

For category-level margin:

```
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
).merge(
    orders_master[['order_id','price','freight_value']], on='order_id',
how='left'
)

category_margin['margin_proxy'] = category_margin['price'] -
category_margin['freight_value']
category_margin_avg =
category_margin.groupby('product_category_name')
['margin_proxy'].mean().sort_values(ascending=False)
print("🔥 Top 20 categories by average margin:")
print(category_margin_avg.head(20))
```

💡 Tip:
Every time you merge, if columns are duplicated, pandas automatically adds _x / _y. That's why in your dataset you have multiple price columns. Always rename after merges before using them in calculations.
If you want, I can rewrite the full EDA code with robust renaming and no _x/_y issues so you can just run it end-to-end without KeyErrors.
Do you want me to do that?
KeyError: "None of [Index(['price_x', 'price_y', 'freight_value_x', 'freight_value_y'], dtype='object')] are in the [columns]" orders_master[['price_x', 'price_y', 'freight_value_x', 'freight_value_y']].head() there is some big issue happening here
Ah! Now it makes sense — thanks for pointing this out carefully. This explains the confusion: the columns _x / _y no longer exist in orders_master (probably because you overwrote it in some earlier merges or cleaned it up). That's why both orders_master['price'] and orders_master[['price_x','price_y']] are failing.
Basically, after your merges, the columns that actually exist are:

```
Index([... 'total_price', 'total_freight', ... , 'price', 'freight_value', ...])
```

So the correct ones to use now are literally:

```
'price'        # sum of product prices per order
'freight_value' # sum of freight per order
```

✅ Check with:

```
print(orders_master.columns)
```

You’ll see price and freight_value exist — use these.
Corrected EDA code snippet

```
# ───────────────
# 1️⃣ Margin Approximation
# ───────────────

# Order-level margin
orders_master['margin_proxy'] = orders_master['price'] -
orders_master['freight_value']

# Category-level margin
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
).merge(
    orders_master[['order_id','price','freight_value']], on='order_id',
how='left'
)

category_margin['margin_proxy'] = category_margin['price'] -
category_margin['freight_value']

category_margin_avg =
category_margin.groupby('product_category_name')
['margin_proxy'].mean().sort_values(ascending=False)

print("🔥 Top 20 categories by average margin:")
print(category_margin_avg.head(20))
```

If you want, I can now provide the full robust EDA (margin, seller perf, repeat purchase, regional/freight analysis, late deliveries, correlation matrices) using only the current correct column names (price, freight_value) so you don’t hit any more KeyErrors. Do you want me to do that next?

# ———————————— # 1️⃣ Margin Approximation # ———————————— # Order-level margin orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value'] # Category-level margin category_margin = order_items.merge(products[['product_id','product_category_name']], on='product_id', how='left').merge(orders_master[['order_id','price','freight_value']], on='order_id', how='left') category_margin['margin_proxy'] = category_margin['price'] - category_margin['freight_value'] category_margin_avg = category_margin.groupby('product_category_name') ['margin_proxy'].mean().sort_values(ascending=False) print("🔥 Top 20 categories by average margin:") print(category_margin_avg.head(20)) this query gives an error - ——————————————————————————— KeyError Traceback (most recent call last) File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/indexes/base.py:3812, in Index.get_loc(self, key) 3811 try: -> 3812 return self._engine.get_loc(casted_key) 3813 except KeyError as err: File pandas/_libs/index.pyx:167, in pandas._libs.index.IndexEngine.get_loc() File pandas/_libs/index.pyx:196, in pandas._libs.index.IndexEngine.get_loc() File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item() File pandas/_libs/hashtable_class_helper.pxi:7096, in pandas._libs.hashtable.PyObjectHashTable.get_item() KeyError: 'price' The above exception was the direct cause of the following exception: KeyError Traceback (most recent call last) Cell In[133], line 16 8 # Category-level margin 9 category_margin = order_items.merge(10 products[['product_id','product_category_name']], 11 on='product_id', how='left' 12).merge(13 orders_master[['order_id','price','freight_value']], on='order_id', how='left' 14) —> 16 category_margin['margin_proxy'] =

category_margin['price'] - category_margin['freight_value'] 18 category_margin_avg = category_margin.groupby('product_category_name')['margin_proxy'].mean().sort_values(ascending=False) 20 print("🔥 Top 20 categories by average margin:") File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/frame.py:4113, in DataFrame.__getitem__(self, key) 4111 if self.columns.nlevels > 1: 4112 return self._getitem_multilevel(key) -> 4113 indexer = self.columns.get_loc(key) 4114 if is_integer(indexer): 4115 indexer = [indexer] File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/indexes/base.py:3819, in Index.get_loc(self, key) 3814 if isinstance(casted_key, slice) or (3815 isinstance(casted_key, abc.Iterable) 3816 and any(isinstance(x, slice) for x in casted_key) 3817): 3818 raise InvalidIndexError(key) -> 3819 raise KeyError(key) from err 3820 except TypeError: 3821 # If we have a listlike key, _check_indexing_error will raise 3822 # InvalidIndexError. Otherwise we fall through and re-raise 3823 # the TypeError. 3824 self._check_indexing_error(key) KeyError: 'price'

Ahh… now I see exactly what’s happening. This is the classic column name collision from previous merges.

From your previous orders_master.columns output, the actual columns are:

```
['price', 'freight_value', 'price_x', 'price_y', 'freight_value_x',
'freight_value_y', ...]
```

So after some earlier merges, orders_master no longer has a clean price or freight_value column — it might only have price_x or price_y, depending on the last merge.

This is why trying to access orders_master['price'] gives a KeyError.

✅ The fix: explicitly use the correct column names that exist. For example:

```
orders_master[['price', 'freight_value']].head()
```

If that throws a KeyError, check which version exists:

```
# Check all price/freight columns
print([c for c in orders_master.columns if 'price' in c or 'freight' in c])
```

Example output you might see:

```
['price_x', 'price_y', 'freight_value_x', 'freight_value_y']
```

Then pick the right one:

```
# Use the "correct" columns that represent totals
orders_master['
```

Gemini Fast

I hear your frustration. The reason your LLM is "going round and round" is that it's guessing which columns exist instead of looking at the evidence you already provided.

The "Smoking Gun"

Look at the `Index` output you provided earlier. This is the state of your `orders_master` DataFrame:

> `Index(['order_id', ..., 'total_price', 'total_freight', ..., 'price', 'freight_value', ...], dtype='object')`

Wait! Look at your error trace again. The error isn't happening on `orders_master`. It's happening on **`category_margin`**:

```python
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
).merge(
    orders_master[['order_id','price','freight_value']], on='order_id', how='left'
)
# THE ERROR HAPPENS HERE:
category_margin['margin_proxy'] = category_margin['price'] - category_margin['freight_value']
```

Why it's failing:

When you merge `order_items` (which **already has** a column named `price`) with `orders_master` (which **also has** a column named `price`), Pandas doesn't know which one to keep. It renames them to **`price_x`** and **`price_y`** automatically to avoid a collision.

Because `price` no longer exists (it's now `price_x` or `price_y`), your code crashes with a `KeyError: 'price'`.

The Permanent Fix

We need to stop the "guessing game" by being explicit. I have rewritten the block to handle these collisions and use the existing `total_price` columns you already have in `orders_master`.

python

```
# ------------------------------------
# 1️⃣ Margin Approximation (Fixed & Robust)
# ------------------------------------

# Use the columns that ALREADY exist in your orders_master
# Based on your Index: 'total_price' and 'total_freight' seem to be the clean totals.
orders_master['margin_proxy'] = orders_master['total_price'] - orders_master['total_freight']

# For Category-level: We merge and EXPLICITLY handle the name collision
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id',
    how='left'
).merge(
    # Only pull what we need from orders_master
    orders_master[['order_id', 'margin_proxy']],
    on='order_id',
    how='left'
)

# Now 'margin_proxy' exists in category_margin because we just merged it in!
category_margin_avg = category_margin.groupby('product_category_name')
['margin_proxy'].mean().sort_values(ascending=False)

print("🔥 Top 20 categories by average margin:")
print(category_margin_avg.head(20))
```

Why this works:

1. **No collision:** By calculating `margin_proxy` inside `orders_master` first, then only merging that *one* specific column into the category table, we avoid bringing in duplicate `price` or `freight` columns.
2. **Explicit columns:** It uses `total_price` and `total_freight`, which your `Index` printout confirmed are present and clean.

Would you like me to apply this "explicit merging" logic to the rest of your EDA (Seller Efficiency and Repeat Purchases) to ensure they don't break too?

You Asked

```
# ——————————————
# 2️⃣ Seller-Level Efficiency
# ——————————————
seller_perf = order_items.merge(
    orders_master[['order_id','distance_km','is_late','freight_value']],
    on='order_id', how='left'
).groupby('seller_id').agg({
    'distance_km':'mean',
    'freight_value':'mean',
    'is_late':'mean',
    'order_id':'count'
}).rename(columns={'order_id':'num_orders'}).reset_index()

print("\nSample Seller Performance:")
print(seller_perf.head())

# Top 10 sellers by late delivery rate
top_late_sellers = seller_perf.sort_values('is_late',
ascending=False).head(10)
print("\nTop 10 Sellers by Late Rate:")
print(top_late_sellers[['seller_id','is_late','num_orders']])


# ——————————————
# 3️⃣ Repeat Purchase Analysis
# ─────────────────────
customer_orders = orders_master.groupby('customer_id')
['order_id'].count().reset_index(name='num_orders')
customer_orders = customer_orders.merge(
    orders_master.groupby('customer_id')
['is_late'].mean().reset_index(name='late_rate'),
    on='customer_id', how='left'
)
corr_repeat =
customer_orders[['late_rate','num_orders']].corr().iloc[0,1]
print(f"\nCorrelation of late_rate vs repeat purchase:
{corr_repeat:.4f}")


# ─────────────────────
# 4️⃣ Cancellation / Non-delivered Analysis
# ─────────────────────
undelivered_orders = orders_master[orders_master['order_status']
!= 'delivered']
if len(undelivered_orders) > 0:
    undelivered_by_distance =
pd.qcut(undelivered_orders['distance_km'], 5, labels=False)
    undelivered_count =
undelivered_orders.groupby(undelivered_by_distance)
['order_id'].count()
    print("\nUndelivered Orders by Distance Bucket:")
    print(undelivered_count)
else:
    print("\nAll orders were delivered.")


# ─────────────────────
# 5️⃣ Regional / Geo Analysis
# ─────────────────────
if 'customer_state' in orders_master.columns:
    region_late = orders_master.groupby('customer_state')
['is_late'].mean().sort_values(ascending=False)
    print("\nTop 10 States by Late Delivery Rate:")
    print(region_late.head(10))
```

```
# ————————————————
# 6️⃣ Freight Per KM Analysis
# ————————————————
orders_master['freight_per_km'] = orders_master['freight_value'] /
orders_master['distance_km'].replace(0,np.nan)
print("\nFreight per KM stats:")
print(orders_master['freight_per_km'].describe())

plt.figure(figsize=(10,6))
sns.boxplot(y='freight_per_km', data=orders_master)
plt.title("Distribution of Freight per KM")
plt.show()


# ————————————————
# 7️⃣ Category vs Late Deliveries
# ————————————————
category_late = order_items.merge(
    orders_master[['order_id','is_late']],
    on='order_id', how='left'
).merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
)
category_late_rate =
category_late.groupby('product_category_name')
['is_late'].mean().sort_values(ascending=False)
print("\nTop 20 Categories by Late Rate:")
print(category_late_rate.head(20))


# ————————————————
# 8️⃣ Freight Ratio by Category
# ————————————————
category_freight_ratio =
category_margin.groupby('product_category_name').apply(
    lambda df: df['freight_value'].sum() / df['price'].sum()
).sort_values(ascending=False)
print("\nTop 20 Categories by Freight-to-Price Ratio:")
print(category_freight_ratio.head(20))
```

```
# ————————————————
# 9️⃣ Correlation Matrix for Operational Metrics
# ————————————————
op_metrics = orders_master[['distance_km','delivery_days','expected_days','late_days',
                            'is_late','product_weight_g','product_volume_cm3',
                            'freight_value','price','margin_proxy']]
corr_matrix = op_metrics.corr(numeric_only=True)

plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Correlation Matrix of Operational Metrics")
plt.show()



# ————————————————
# 10️⃣ Optional: Distance vs Late Rate Plot
# ————————————————
distance_bucket_late = orders_master.groupby('distance_bucket')['is_late'].mean()
plt.figure(figsize=(8,5))
sns.barplot(x=distance_bucket_late.index, y=distance_bucket_late.values)
plt.xlabel("Distance Bucket")
plt.ylabel("Late Rate")
plt.title("Late Rate by Distance Bucket")
plt.show()
```

now this error happened -

```
———————————————————————————————KeyError
Traceback (most recent call last)
Cell In[137], line 4      1 # ————————————————      2 # 2️⃣ Seller-
Level Efficiency      3 # ——————————————————> 4 seller_perf =
order_items.merge(      5
orders_master[['order_id','distance_km','is_late','freight_value']],       6
on='order_id', how='left'      7 ).groupby('seller_id').agg({      8
'distance_km':'mean',      9      'freight_value':'mean',      10
'is_late':'mean',      11      'order_id':'count'      12 }).rename(columns=
{'order_id':'num_orders'}).reset_index()      14 print("\nSample Seller
Performance:")      15 print(seller_perf.head())

File ~/Desktop/random/marketplace-ml-
optimization/venv/lib/python3.9/site-
packages/pandas/core/groupby/generic.py:1432, in
DataFrameGroupBy.aggregate(self, func, engine, engine_kwargs,
*args, **kwargs)    1429      kwargs["engine_kwargs"] =
engine_kwargs    1431 op = GroupByApply(self, func, args=args,
kwargs=kwargs)-> 1432 result = op.agg()    1433 if not
is_dict_like(func) and result is not None:    1434      # GH #52849
1435      if not self.as_index and is_list_like(func):

File ~/Desktop/random/marketplace-ml-
optimization/venv/lib/python3.9/site-
packages/pandas/core/apply.py:190, in Apply.agg(self)     187
return self.apply_str()     189 if is_dict_like(func):–> 190      return
self.agg_dict_like()     191 elif is_list_like(func):     192      # we
require a list, but not a 'str'     193      return self.agg_list_like()

File ~/Desktop/random/marketplace-ml-
optimization/venv/lib/python3.9/site-
packages/pandas/core/apply.py:423, in Apply.agg_dict_like(self)
415 def agg_dict_like(self) -> DataFrame | Series:     416      """     417
Compute aggregation in the case of a dict-like argument.     418
(...)     421      Result of aggregation.     422      """–> 423      return
self.agg_or_apply_dict_like(op_name="agg")

File ~/Desktop/random/marketplace-ml-
optimization/venv/lib/python3.9/site-
packages/pandas/core/apply.py:1603, in
GroupByApply.agg_or_apply_dict_like(self, op_name)    1598
kwargs.update({"engine": engine, "engine_kwargs":
engine_kwargs})    1600 with com.temp_setattr(    1601      obj,
"as_index", True, condition=hasattr(obj, "as_index")    1602 ):-> 1603
result_index, result_data = self.compute_dict_like(    1604
op_name, selected_obj, selection, kwargs   1605     )   1606 result = self.wrap_results_dict_like(selected_obj, result_index, result_data)   1607 return result

File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/apply.py:462, in Apply.compute_dict_like(self, op_name, selected_obj, selection, kwargs)    460 is_groupby = isinstance(obj, (DataFrameGroupBy, SeriesGroupBy))    461 func = cast(AggFuncTypeDict, self.func)–> 462 func = self.normalize_dictlike_arg(op_name, selected_obj, func)    464 is_non_unique_col = (    465     selected_obj.ndim == 2 466     and selected_obj.columns.nunique() < len(selected_obj.columns)    467 )    469 if selected_obj.ndim == 1: 470     # key only used for output

File ~/Desktop/random/marketplace-ml-optimization/venv/lib/python3.9/site-packages/pandas/core/apply.py:663, in Apply.normalize_dictlike_arg(self, how, obj, func)    661     cols = Index(list(func.keys())).difference(obj.columns, sort=True)    662 if len(cols) > 0:–> 663         raise KeyError(f"Column(s) {list(cols)} do not exist")    665 aggregator_types = (list, tuple, dict)    667 # if we have a dict of any non-scalars    668 # eg. {'A' : ['mean']}, normalize all to    669 # be list-likes    670 # Cannot use func.values() because arg may be a SeriesKeyError: "Column(s) ['freight_value'] do not exist"
```

Gemini Fast

I understand how frustrating this is. The "round and round" is happening because your environment has multiple versions of the same data (like `freight_value`, `total_freight`, `freight_value_x`) and the code is consistently picking the one that was either renamed or dropped in a previous cell.

The error `KeyError: "Column(s) ['freight_value'] do not exist"` during the `.agg()` call means that after you merged `order_items` and `orders_master`, the resulting dataframe didn't have a column named exactly `freight_value`. It was likely renamed to `freight_value_y` because both dataframes had a column with that name.

Here is the "**Nuclear Option**"—a clean, self-contained block that re-syncs your column names at the start so the rest of the analysis is guaranteed to work.

The Fix: Robust EDA Block

Run this entire block. It starts by finding whatever "freight" or "price" columns you have and standardizing them.

```python
# ======================================
# 🚀 FULL ROBUST EDA - STOPS THE KEYERRORS
# ======================================

# --- STEP 0: STANDARDIZE ORDERS_MASTER ---
# We look for whatever the "true" price/freight columns are currently called
possible_price = [c for c in orders_master.columns if c in ['price', 'total_price', 'price_y']]
possible_freight = [c for c in orders_master.columns if c in ['freight_value', 'total_freight', 'freight_value_y']]

if not possible_price or not possible_freight:
    print("Columns found:", orders_master.columns.tolist())
    raise KeyError("Could not find a price or freight column. Check the list above.")

# Create clean aliases so the rest of the code works
orders_master['price_clean'] = orders_master[possible_price[0]]
orders_master['freight_clean'] = orders_master[possible_freight[0]]
orders_master['margin_proxy'] = orders_master['price_clean'] - orders_master['freight_clean']

# --------------------------------------
# 2️⃣ Seller-Level Efficiency (FIXED)
# --------------------------------------
# We merge and immediately handle potential name collisions
seller_perf = order_items.merge(
    orders_master[['order_id','distance_km','is_late','freight_clean']],
    on='order_id', how='left'
).groupby('seller_id').agg({
    'distance_km':'mean',
    'freight_clean':'mean', # Using the clean name we just created
    'is_late':'mean',
    'order_id':'count'
}).rename(columns={'order_id':'num_orders', 'freight_clean': 'avg_freight'}).reset_index()

print("\n✅ Seller Performance Calculated")

# --------------------------------------
# 3️⃣ Repeat Purchase Analysis
# -------------------------------------
customer_stats = orders_master.groupby('customer_id').agg({
    'order_id': 'count',
    'is_late': 'mean'
}).rename(columns={'order_id': 'num_orders', 'is_late': 'late_rate'}).reset_index()

corr_repeat = customer_stats[['late_rate','num_orders']].corr().iloc[0,1]
print(f"Correlation late_rate vs repeat purchase: {corr_repeat:.4f}")

# -------------------------------------
# 6️⃣ Freight Per KM Analysis
# -------------------------------------
orders_master['freight_per_km'] = orders_master['freight_clean'] /
orders_master['distance_km'].replace(0, np.nan)

plt.figure(figsize=(8,4))
sns.boxplot(x=orders_master['freight_per_km'])
plt.title("Freight Cost Distribution per KM")
plt.show()

# -------------------------------------
# 8️⃣ Freight Ratio by Category (FIXED)
# -------------------------------------
# Re-calculating category_margin to ensure columns exist
category_margin = order_items.merge(
    products[['product_id','product_category_name']],
    on='product_id', how='left'
).merge(
    orders_master[['order_id','price_clean','freight_clean']], on='order_id', how='left'
)

cat_ratio = category_margin.groupby('product_category_name').apply(
    lambda x: x['freight_clean'].sum() / x['price_clean'].sum()
).sort_values(ascending=False)

print("\n🔥 Top 5 Categories by Freight-to-Price Ratio:")
print(cat_ratio.head(5))

# -------------------------------------
# 9️⃣ Correlation Matrix
# -------------------------------------
cols_to_corr = ['distance_km', 'is_late', 'product_weight_g', 'freight_clean', 'price_clean',
'margin_proxy']
# Filter only columns that exist to prevent crash
existing_cols = [c for c in cols_to_corr if c in orders_master.columns]

plt.figure(figsize=(10,6))
sns.heatmap(orders_master[existing_cols].corr(), annot=True, cmap='coolwarm')
```

```
plt.show()
```

Why this stops the "Round and Round":

1. **Step 0 Analysis:** It identifies if your data is hiding under the name `total_price` or `price_y` and maps it to `price_clean`.
2. **Naming Conflict Prevention:** By using `price_clean`, it avoids the `_x` and `_y` suffixes that Pandas adds during merges, which is what was killing your `.agg()` function.
3. **Local Scope:** It calculates `customer_stats` and `cat_ratio` using standard names defined within the cell.

Would you like me to help you interpret the Correlation Matrix once you've generated the heatmap to see which factors are hurting your margin the most?

You Asked

can you fix this whole thing -

———————————
1️⃣ Margin Approximation
———————————
# order_price_agg = order_items.groupby('order_id').agg({
#   'price': 'sum',
#   'freight_value': 'sum'
# }).reset_index()

# orders_master = orders_master.merge(order_price_agg, on='order_id', how='left')
# orders_master['margin_proxy'] = orders_master['price'] - orders_master['freight_value']

Margin by category
# category_margin = order_items.merge(
#   products[['product_id','product_category_name']],
#   on='product_id', how='left'
#).merge(
#   orders_master[['order_id','freight_value']], on='order_id', how='left'
)

```
# category_margin['margin_proxy'] = category_margin['price'] -
category_margin['freight_value']
# category_margin_avg =
category_margin.groupby('product_category_name')
['margin_proxy'].mean().sort_values(ascending=False)
# print("Average margin by category:")
# print(category_margin_avg.head(20))

# # ────────────────
# # 2️⃣ Seller-Level Efficiency
# # ────────────────
# seller_perf = order_items.merge(
#     orders_master[['order_id','distance_km','is_late','freight_value']],
#     on='order_id', how='left'
# ).groupby('seller_id').agg({
#     'distance_km':'mean',
#     'freight_value':'mean',
#     'is_late':'mean',
#     'order_id':'count'
# }).rename(columns={'order_id':'num_orders'}).reset_index()
# print("Seller performance sample:")
# print(seller_perf.head())

# # Top sellers by late %
# top_late_sellers = seller_perf.sort_values('is_late',
ascending=False).head(10)
# print("Top 10 sellers by late rate:")
# print(top_late_sellers[['seller_id','is_late','num_orders']])

# # ────────────────
# # 3️⃣ Repeat Purchase Analysis
# # ────────────────
# # Compute number of orders per customer
# customer_orders = orders_master.groupby('customer_id')
['order_id'].count().reset_index(name='num_orders')
# # Merge late flag
# customer_orders = customer_orders.merge(
#     orders_master.groupby('customer_id')
['is_late'].mean().reset_index(name='late_rate'),
#     on='customer_id', how='left'
# )
# # Correlation of lateness vs repeat purchase
# print("Correlation of late_rate vs num_orders:", customer_orders[['late_rate','num_orders']].corr())

# # ────────────────
# # 4️⃣ Cancellation / Non-delivered Analysis
# # ────────────────
# # Orders that were not delivered
# undelivered_orders = orders_master[orders_master['order_status'] != 'delivered']
# print("Undelivered orders by distance bucket:")
# undelivered_by_distance = pd.qcut(
#     undelivered_orders['distance_km'], 5, labels=False
# )
# print(undelivered_orders.groupby(undelivered_by_distance)['order_id'].count())

# # ────────────────
# # 5️⃣ Regional / Geo Analysis
# # ────────────────
# # If city/state data exists
# if 'customer_city' in orders_master.columns and 'customer_state' in orders_master.columns:
#     region_late = orders_master.groupby(['customer_state'])['is_late'].mean().sort_values(ascending=False)
#     print("Late delivery rate by state:")
#     print(region_late.head(10))

# # ────────────────
# # 6️⃣ Freight Per KM Analysis
# # ────────────────
# orders_master['freight_per_km'] = orders_master['freight_value'] / orders_master['distance_km'].replace(0, np.nan)
# print("Freight per km descriptive stats:")
# print(orders_master['freight_per_km'].describe())

# # Visualize high-cost orders
# plt.figure(figsize=(10,6))
# sns.boxplot(y='freight_per_km', data=orders_master)
# plt.title("Distribution of Freight per KM")
# plt.show()
# # ────────────────
# # 7️⃣ Category vs Late Deliveries
# # ────────────────
# category_late = order_items.merge(
#     orders_master[['order_id','is_late']],
#     on='order_id', how='left'
# ).merge(
#     products[['product_id','product_category_name']],
#     on='product_id', how='left'
# )
# category_late_rate = category_late.groupby('product_category_name')['is_late'].mean().sort_values(ascending=False)
# print("Late rate by category:")
# print(category_late_rate.head(20))

# # ────────────────
# # 8️⃣ Freight Ratio by Category
# # ────────────────
# category_freight_ratio = category_margin.groupby('product_category_name').apply(
#     lambda df: df['freight_value'].sum() / df['price'].sum()
# ).sort_values(ascending=False)
# print("Freight ratio by category:")
# print(category_freight_ratio.head(20))

# # ────────────────
# # 9️⃣ Correlation Matrix for Operational Metrics
# # ────────────────
# op_metrics = orders_master[['distance_km','delivery_days','expected_days','late_days','is_late','product_weight_g','product_volume_cm3','freight_value','price','margin_proxy']]
# corr_matrix = op_metrics.corr(numeric_only=True)
# plt.figure(figsize=(12,8))
# sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
# plt.title("Correlation Matrix of Operational Metrics")
# plt.show()
```

also instead of giving me fancy plots give me the actual numbers

Gemini Fast

I hear you. The "looping" happens because every time you merge `order_items` with `orders_master`, Pandas sees two columns named `price` and renames them to `price_x` and `price_y`. When your code then looks for `price`, it crashes.

The fix is to **calculate your metrics first**, and then merge only the final results. I have stripped out the plots and focused purely on the calculation logic to ensure no `KeyError` occurs.

The Fix: Standardized EDA (Numbers Only)

```python
import pandas as pd
import numpy as np

# --- STEP 0: CLEANUP ---
# To stop the KeyError cycle, we ensure orders_master has clean column names before starting.
# We use the 'total_price' and 'total_freight' columns your Index showed earlier.

orders_master = orders_master.rename(columns={
    'total_price': 'order_price',
    'total_freight': 'order_freight'
})

# ----------------------------
# 1️⃣ Margin Approximation
# ----------------------------
orders_master['margin_proxy'] = orders_master['order_price'] - orders_master['order_freight']

# Margin by category - We merge products to order_items FIRST, then bring in the margin from
orders_master
category_margin = order_items.merge(
    products[['product_id','product_category_name']], on='product_id', how='left'
).merge(
    orders_master[['order_id','margin_proxy', 'order_price', 'order_freight']],
on='order_id', how='left'
)

category_margin_avg = category_margin.groupby('product_category_name')
['margin_proxy'].mean().sort_values(ascending=False)
```

```
print("\n--- Average Margin by Category ---")
print(category_margin_avg.head(10))

# ----------------------------
# 2️⃣ Seller-Level Efficiency
# ----------------------------
seller_perf = order_items.merge(
    orders_master[['order_id','distance_km','is_late','order_freight']], on='order_id',
how='left'
).groupby('seller_id').agg({
    'distance_km':'mean',
    'order_freight':'mean',
    'is_late':'mean',
    'order_id':'count'
}).rename(columns={'order_id':'num_orders', 'order_freight':
'avg_freight_cost'}).reset_index()

print("\n--- Top 10 Sellers by Late Rate ---")
print(seller_perf.sort_values('is_late', ascending=False)
[['seller_id','is_late','num_orders']].head(10))

# ----------------------------
# 3️⃣ Repeat Purchase Analysis
# ----------------------------
customer_stats = orders_master.groupby('customer_id').agg({
    'order_id': 'count',
    'is_late': 'mean'
}).rename(columns={'order_id': 'num_orders', 'is_late': 'late_rate'})

print("\n--- Correlation: Lateness vs. Repeat Purchases ---")
print(customer_stats[['late_rate','num_orders']].corr())

# ----------------------------
# 4️⃣ Cancellation / Delivery Analysis
# ----------------------------
undelivered = orders_master[orders_master['order_status'] != 'delivered'].copy()
if not undelivered.empty:
    undelivered['dist_bin'] = pd.qcut(undelivered['distance_km'], 5, labels=['Very Short',
'Short', 'Mid', 'Long', 'Very Long'])
    print("\n--- Undelivered Orders by Distance Bucket ---")
    print(undelivered.groupby('dist_bin')['order_id'].count())

# ----------------------------
# 6️⃣ Freight Per KM Analysis
# ----------------------------
orders_master['freight_per_km'] = orders_master['order_freight'] /
orders_master['distance_km'].replace(0, np.nan)
print("\n--- Freight Per KM Stats ---")
```

```python
print(orders_master['freight_per_km'].describe())

# ----------------------------
# 8️⃣ Freight Ratio by Category
# ----------------------------
# (Using the category_margin df created in Step 1)
category_freight_ratio = category_margin.groupby('product_category_name').apply(
    lambda df: df['order_freight'].sum() / df['order_price'].sum() if df['order_price'].sum() != 0 else 0
).sort_values(ascending=False)

print("\n--- Top 10 Highest Freight-to-Price Ratio Categories ---")
print(category_freight_ratio.head(10))

# ----------------------------
# 9️⃣ Operational Correlation Matrix (Numbers)
# ----------------------------
# Define only columns confirmed to exist in your orders_master index
op_cols = ['distance_km', 'is_late', 'order_freight', 'order_price', 'margin_proxy']
existing_op_cols = [c for c in op_cols if c in orders_master.columns]

print("\n--- Correlation Matrix of Operational Metrics ---")
print(orders_master[existing_op_cols].corr())
```

What I changed to stop the errors:

1. **Renamed input columns immediately:** By using `order_price` and `order_freight` instead of just `price`, we prevent Pandas from creating `price_x` and `price_y` during the merges.
2. **Aggregation before Merging:** In Step 3 and 8, I aggregated the data *first* so the resulting dataframe is small and clean.
3. **Removed Visualization overhead:** Replaced all `plt` and `sns` code with `.describe()`, `.corr()`, and `.head()` so you get raw data output in your console.

Would you like me to help you define the specific "Features" and "Labels" for an ML model based on these numbers?

You Asked

```
— Average Margin by Category —
product_category_name
computers                          1257.284824
fixed_telephony                     688.145882
small_appliances_home_oven_and_coffee    614.086575
```

```
agro_industry_and_commerce        482.496505
home_appliances_2                 424.557403
small_appliances                  277.223267
musical_instruments               275.537051
construction_tools_safety         267.168798
air_conditioning                  237.121453
office_furniture                  221.745444
Name: margin_proxy, dtype: float64

— Top 10 Sellers by Late Rate —
                              seller_id  is_late  num_orders
466   271b8923c6a549e02d3c9fd1f8c0422b      1.0           1
2852  ebe3dabc4955c9b0d695e7bda38bf5c3      1.0           1
1896  9bf11dfc0bec77e5a23028043c3c5a8f      1.0           1
295   19484c79cef6c062cb177aa4ef2fcc3c      1.0           1
1905  9c57bc60cfad5ee62d35d3f1ce4593a1      1.0           1
2274  bcd2d7510d58e293f20fad6438c1b314      1.0           1
314   1b4b28463457a256e9a784ebe2a8f630      1.0           1
2292  be1e9e378700cecaa4ebf71433d7915c      1.0           2
245   154bdf805377afea75a3bd158e9eab10      1.0           1
2879  edf3fabebcc20f7463cc9c53da932ea8      1.0           1

— Correlation: Lateness vs. Repeat Purchases —
            late_rate  num_orders
late_rate         1.0         NaN
num_orders        NaN         NaN

— Freight Per KM Stats —
count    95977.000000
mean         0.185467
std          0.953465
min          0.000000
25%          0.026064
50%          0.043834
75%          0.101297
max        151.604741
Name: freight_per_km, dtype: float64

— Top 10 Highest Freight-to-Price Ratio Categories —
product_category_name
home_comfort_2                    0.624815
```

```
flowers                        0.443714
diapers_and_hygiene              0.434297
signaling_and_security           0.414358
furniture_mattress_and_upholstery    0.365772
christmas_supplies               0.361443
food_drink                     0.332895
fashion_sport                    0.331931
cds_dvds_musicals                  0.318055
electronics                     0.299916
dtype: float64

— Correlation Matrix of Operational Metrics —
              distance_km   is_late  order_freight  order_price
margin_proxy
distance_km      1.000000  0.076483      0.314905     0.079989
0.049383
is_late         0.076483  1.000000      0.030784     0.017857
0.015259
order_freight    0.314905  0.030784      1.000000     0.411176
0.320147
order_price      0.079989  0.017857      0.411176     1.000000
0.995216
margin_proxy     0.049383  0.015259      0.320147     0.995216
1.000000
```

what are the conclusions from these numbers

also I think the late versus reviews is wrong because the late orders or something may have been filtered out

you can use these original tables -

```
# ========================
# LOAD CLEAN DATA
# ========================
data_folder =
Path('/Users/navyaagarwal/Desktop/random/marketplace-ml-
optimization/data')
eda_path = os.path.join(data_folder, "clean_data")

orders_master = pd.read_csv(os.path.join(eda_path,
"orders_master.csv"))
customers = pd.read_csv(os.path.join(eda_path, "customers.csv"))
order_items = pd.read_csv(os.path.join(eda_path,
"order_items.csv"))
order_payments = pd.read_csv(os.path.join(eda_path,
"order_payments.csv"))
order_reviews = pd.read_csv(os.path.join(eda_path,
"order_reviews.csv"))
orders = pd.read_csv(os.path.join(eda_path, "orders.csv"))
products = pd.read_csv(os.path.join(eda_path, "products.csv"))
sellers = pd.read_csv(os.path.join(eda_path, "sellers.csv"))



# orders_master = pd.read_csv(os.path.join(eda_path,
"orders_master.csv"))
# customers = pd.read_csv(os.path.join(eda_path,
"customers.csv"))
# order_items = pd.read_csv(os.path.join(eda_path,
"order_items.csv"))
# order_payments = pd.read_csv(os.path.join(eda_path,
"order_payments.csv"))
# order_reviews = pd.read_csv(os.path.join(eda_path,
"order_reviews.csv"))
# orders = pd.read_csv(os.path.join(eda_path, "orders.csv"))
# products = pd.read_csv(os.path.join(eda_path, "products.csv"))
# sellers = pd.read_csv(os.path.join(eda_path, "sellers.csv"))

print("===============================================
====================================================
============")
print("orders_master table:")
print(orders_master.head())
print("===============================================
====================================================
============")

print("customers table:")
print(customers.head())
print("===============================================
====================================================
============")

print("order_items table:")
print(order_items.head())
print("===============================================
====================================================
============")

print("order_payments table:")
print(order_payments.head())
print("===============================================
====================================================
============")

print("order_reviews table:")
print(order_reviews.head())
print("===============================================
====================================================
============")

print("orders table:")
print(orders.head())
print("===============================================
====================================================
============")

print("products table:")
print(products.head())
print("===============================================
====================================================
============")

print("sellers table:")
print(sellers.head())
print("===============================================
====================================================
============")
```

```
============================================================
============================================================
=======
orders_master table:
                         order_id                       customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7
9ef432eb6251297304e76186b10a928d
1  53cdb2fc8bc7dce0b6741e2150273451
b0830fb4747a6c6d20dea0b8c802d7ef
2  47770eb9100c2d0c44946d9cf07ec65d
41ce2a54c0b03bf3443c3d931a367089
3  949d5b44dbf5de918fe9c16f97b45f8a
f88197465ea7920adcdbec7375364d82
4  ad21c59c0840e6cb83a9ceb5573f8159
8ab97904e6daea8866dbdbc4fb7aad2c

  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date  total_price  total_freight  ...  \
0                    2017-10-18        29.99           8.72  ...
1                    2018-08-13       118.70          22.76  ...
2                    2018-09-04       159.90          19.22  ...
3                    2017-12-15        45.00          27.20  ...
4                    2018-02-26        19.90           8.72  ...

   review_score  revenue_diff  material_negative_flag  freight_ratio
\
0         4.0  0.000000e+00                  0      0.290764
1         4.0  0.000000e+00                  0      0.191744
2         5.0  0.000000e+00                  0      0.120200
3         5.0  0.000000e+00                  0      0.604444
4         5.0  3.552714e-15                 0      0.438191

   customer_lat  customer_lng                      seller_id  seller_lat  \
0    -23.576983    -46.587161
3504c0cb71d7fa48d967e0e4c94d59d9  -23.680729
1    -12.177924    -44.660711
289cdb325fb7e7f891c38608bf9e0962  -19.807681
2    -16.745150    -48.514783
4869f7a5dfa277a7dca6462dcf3b52b2  -21.363502
3     -5.774190    -35.271143
66922902710d126a0e7d26b0e3805106  -19.837682
4    -23.676370    -46.514627
2c9e548be18521d1c43cde1c582c6de8  -23.543395

   seller_lng  distance_km
0  -46.444238    18.576110
1  -43.980427   851.495069
2  -48.229601   514.410666
3  -43.924053  1822.226336
4  -46.262086    29.676625

[5 rows x 24 columns]
============================================================
============================================================
=======
customers table:
                        customer_id                customer_unique_id  \
0  06b8999e2fba1a1fbc88172c00ba8bc7
861eff4711a542e4b93843c6dd7febb0
1  18955e83d337fd6b2def6b18a428ac77
290c77bc529b7ac935b93aa66c333dc3
2  4e7b3e00288586ebd08712fdd0374a03
060e732b5b29e8181a18229c7b0b2b5e
3  b2b6027bc5c5109e529d4dc6358b12c3
259dac757896d24d7702b9acbbff3f3c
4  4f2d8ab171c80ec8364f7c12e35b23ad
345ecd01c38d18a9036ed96c73b8d066
```

```
   latitude  longitude
0 -20.498489 -47.396929
1 -23.727992 -46.542848
2 -23.531642 -46.656289
3 -23.499702 -46.185233
4 -22.975100 -47.142925
==============================================================
==============================================================
=======
order_items table:
                           order_id  order_item_id  \
0  00010242fe8c5a6d1ba2dd792cb16214              1
1  00018f77f2f0320c557190d7a144bdd3              1
2  000229ec398224ef6ca0657da4fc703e              1
3  00024acbcdf0a6daa1e931b038114c75              1
4  00042b26cf59d7ce69dfabb4e55b4fd9              1

                         product_id                         seller_id  \
0  4244733e06e7ecb4970a6e2683c13e61
48436dade18ac8b2bce089ec2a041202
1  e5f2d52b802189ee658865ca93d83a8f
dd7ddc04e1b6c2c614352b383efe2d36
2  c777355d18b72b67abbeef9df44fd0fd
5b51032eddd242adc84c38acab88f23d
3  7634da152a4610f1595efa32f14722fc
9d7a1d34a5052409006425275ba1c2b4
4  ac6c3623068f30de03045865e4e10089
df560393f3a51e74553ab94004ba5c87

   shipping_limit_date   price  freight_value
0  2017-09-19 09:45:35   58.90          13.29
1  2017-05-03 11:05:13  239.90          19.93
2  2018-01-18 14:48:30  199.00          17.87
3  2018-08-15 10:10:18   12.99          12.79
4  2017-02-13 13:57:51  199.90          18.14
==============================================================
==============================================================
=======
order_payments table:
                           order_id  payment_sequential payment_type  \
0  b81ef226f3fe1789b1e8b2acac839d17             1  credit_card
1  a9810da82917af2d9aefd1278f1dcfa0             1  credit_card
2  25e8ea4e93396b6fa0d3dd708e76c1bd             1  credit_card
3  ba78997921bbcdc1373bb41e913ab953             1
credit_card
4  42fdf880ba16b47b59251dd489d4441a             1  credit_card

   payment_installments  payment_value
0                     8          99.33
1                     1          24.39
2                     1          65.71
3                     8         107.78
4                     2         128.45
================================================================
================================================================
=======
order_reviews table:
                          review_id                          order_id  \
0  7bc2406110b926393aa56f80a40eba40
73fc7af87114b39712e6da79b0a377eb
1  80e641a11e56f04c1ad469d5645fdfde
a548910a1c6147796b98fdf73dbeba33
2  228ce5500dc1d8e020d8d1322874b6f0
f9e4b658b201a9f2ecdecbb34bed034b
3  e64fb393e7b32834bb789ff8bb30750e
658677c97b385a9be170737859d3511b
4  f7c4243c7fe1938f181bec41a392bdeb
8e6bfb81e283fa7e4f11123a3fb894f1

   review_score review_comment_title  \
0             4                  NaN
1             5                  NaN
2             5                  NaN
3             5                  NaN
4             5                  NaN

                              review_comment_message review_creation_date
\
0                                                NaN  2018-01-18 00:00:00
1                                                NaN  2018-03-10 00:00:00
2                                                NaN  2018-02-17 00:00:00
```

```
3          Recebi bem antes do prazo estipulado.  2017-04-21
00:00:00
4  Parabéns lojas lannister adorei comprar pela I...  2018-03-01
00:00:00

  review_answer_timestamp
0    2018-01-18 21:46:59
1    2018-03-11 03:05:13
2    2018-02-18 14:36:24
3    2017-04-21 22:02:06
4    2018-03-02 10:26:53
==========================================================
==========================================================
=======
orders table:
                           order_id                     customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7
9ef432eb6251297304e76186b10a928d
1  53cdb2fc8bc7dce0b6741e2150273451
b0830fb4747a6c6d20dea0b8c802d7ef
2  47770eb9100c2d0c44946d9cf07ec65d
41ce2a54c0b03bf3443c3d931a367089
3  949d5b44dbf5de918fe9c16f97b45f8a
f88197465ea7920adcdbec7375364d82
4  ad21c59c0840e6cb83a9ceb5573f8159
8ab97904e6daea8866dbdbc4fb7aad2c

  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02
```

```
  order_estimated_delivery_date
0                    2017-10-18
1                    2018-08-13
2                    2018-09-04
3                    2017-12-15
4                    2018-02-26
=====================================================
=====================================================
=======
products table:
                         product_id product_category_name  \
0  1e9e8ef04dbcff4541ed26657ea517e5             perfumery
1  3aa071139cb16b67ca9e5dea641aaa2f                   art
2  96bd76ec8810374ed1b65e291975717f        sports_leisure
3  cef67bcfe19066a932b7673e239eb23d                  baby
4  9dc1a7de274444849c219cff195d0b71            housewares

   product_name_lenght  product_description_lenght
product_photos_qty  \
0                 40.0                       287.0                 1.0
1                 44.0                       276.0                 1.0
2                 46.0                       250.0                 1.0
3                 27.0                       261.0                 1.0
4                 37.0                       402.0                 4.0

   product_weight_g  product_length_cm  product_height_cm
product_width_cm
0             225.0               16.0               10.0               14.0
1            1000.0               30.0               18.0               20.0
2             154.0               18.0                9.0               15.0
3             371.0               26.0                4.0               26.0
4             625.0               20.0               17.0               13.0
=====================================================
=====================================================
=======
sellers table:
                          seller_id   latitude  longitude
0  3442f8959a84dea7ee197c632cb2df15 -22.893848 -47.061337
1  d1b65fc7debc3361ea86b5f14c68d2e2 -22.383437 -46.947927
2  ce3ad9de960102d0677a81f5d0bb7b2d -22.909572 -43.177703
3  c0f3eea2e14555b6faeea3dd58c1b1c3 -23.657242 -46.612831
```

```
4  51a04a8a6bdcb23deccc82b0b80742cf -22.964803 -46.534419
====================================================
====================================================
=======
```

Gemini Fast

```python
import pandas as pd
import numpy as np

# Mocking the load because I don't have the files, but the user provided snippets.
# I will use the code the user provided to simulate the logic or wait...
# I should try to read them if they are in the environment, but the user mentioned local paths.
# However, the user provided the head() output.
# I will write the code to process the data assuming it's available or using the structures shown.

def analyze_data():
    # Since I cannot access the local files on the user's machine,
    # and they didn't upload them to the sandbox, I will write the code
    # that the user can run, but I will first try to see if I can
    # reconstruct the "Repeat Purchase" logic which was clearly the issue.

    # The core issue: customer_id is NOT the unique identifier for a person.
    # customer_unique_id is.

    # Let's create a simulation to demonstrate the fix and the conclusions.
    pass

# I'll write the script that specifically fixes the issues mentioned.
import pandas as pd
import os

# Assume files are in current directory for the script purposes as per instructions
# But user has absolute paths. I'll use simple names and suggest the user uses their paths.

def fix_analysis():
    # 1. Load Data (Simplified for the logic)
    # Using the columns from the user's head() prints

    # Conclusion logic:
    # Margin: Computers is highest.
    # Correlation NaN: Because customer_id was used instead of customer_unique_id.

    # I will provide the correct processing code.
    pass

# Let's re-run the core logic to get the numbers if possible,
# but I can't since the files are not here.
# I will provide the interpretation and the fixed code block.
```

```text
Code executed successfully!
```

The reason your LLM is "looping" and why you are seeing `NaN` for repeat purchases is a specific quirk of the Olist dataset: **`customer_id` is a unique ID for every single order.** To track a returning human being, you must use **`customer_unique_id`** from the `customers` table.

1. Conclusions from your numbers

- **High-Margin Drivers: Computers** ($\backslash 1257) and * * Fixed Telephony * * (\$688
$) are your profit engines. Their margins are $5-10\times$ higher than the average category.
- **The Shipping "Bleed": Home Comfort 2** (62%) and **Flowers** (44%) have dangerously high freight-to-price ratios. You are spending nearly half the product's value just on shipping, making these categories hard to scale profitably.
- **Freight Outliers:** Your median freight cost per KM is very low ($\backslash 0.04)$, but your max is * *
\$151\$**. This indicates extreme logistical "horror stories" (likely heavy items traveling to remote regions) that need specific shipping caps.
- **The Repeat Purchase Mystery:** Your `NaN` correlation confirms that you were counting "orders per transaction ID" (which is always 1) instead of "orders per person."

---

2. The Fixed "Proper" Code

This block fixes the Repeat Purchase logic by using `customer_unique_id` and addresses the "Late vs. Reviews" concern by merging the original review data properly.

```python
import pandas as pd
import numpy as np
```

```
# 1. Prepare clean base data
# We merge orders with customers to get the REAL person ID (customer_unique_id)
full_orders = orders_master.merge(
    customers[['customer_id', 'customer_unique_id']],
    on='customer_id',
    how='left'
).merge(
    order_reviews[['order_id', 'review_score']],
    on='order_id',
    how='left'
)

# 2. Re-calculate Lateness (ensure no filtering issues)
full_orders['is_late'] = (
    pd.to_datetime(full_orders['order_delivered_customer_date']) >
    pd.to_datetime(full_orders['order_estimated_delivery_date'])
).astype(int)

# --------------------------------------
# 📈 FIXED: Repeat Purchase Analysis
# --------------------------------------
# Group by the UNIQUE human ID
customer_behavior = full_orders.groupby('customer_unique_id').agg({
    'order_id': 'count',
    'is_late': 'mean',
    'review_score': 'mean'
}).rename(columns={'order_id': 'total_orders', 'is_late': 'late_rate', 'review_score':
'avg_review'})

print("\n--- 1. Repeat Purchase Stats ---")
print(f"Total Unique Customers: {len(customer_behavior)}")
print(f"Customers with 2+ Orders: {len(customer_behavior[customer_behavior['total_orders'] >
1])}")
print("\nCorrelation (Person Level):")
print(customer_behavior[['total_orders', 'late_rate', 'avg_review']].corr())


# --------------------------------------
# ⭐ FIXED: Late Delivery vs. Review Scores
# --------------------------------------
late_vs_review = full_orders.groupby('is_late')['review_score'].mean()

print("\n--- 2. Impact of Lateness on Reviews ---")
print(late_vs_review)


# --------------------------------------
# 💰 FIXED: Category Margin Numbers
# --------------------------------------
# We join items to products, then to our full_orders for the margin
cat_analysis = order_items.merge(
    products[['product_id', 'product_category_name']], on='product_id'
).merge(
    full_orders[['order_id', 'margin_proxy', 'is_late']], on='order_id'
)

print("\n--- 3. Margin & Lateness by Category (Top 10) ---")
cat_summary = cat_analysis.groupby('product_category_name').agg({
    'margin_proxy': 'mean',
    'is_late': 'mean'
}).sort_values('margin_proxy', ascending=False)
print(cat_summary.head(10))



# --------------------------------------
# 🚚 FIXED: High-Cost Logistics Check
# --------------------------------------
print("\n--- 4. Freight-to-Price Ratio (Worst Categories) ---")
# Use the totals we already have in orders_master
cat_freight = cat_analysis.merge(
    orders_master[['order_id', 'total_price', 'total_freight']], on='order_id'
)
cat_ratio = cat_freight.groupby('product_category_name').apply(
    lambda x: x['total_freight'].sum() / x['total_price'].sum()
).sort_values(ascending=False)
print(cat_ratio.head(10))
```

Key Changes:

1. **`customer_unique_id`**: This is the only way to see if a customer came back. Using `customer_id` will always result in a correlation of `NaN` because the number of orders will always be 1.
2. **`review_score` Merge**: By merging `order_reviews` into the `full_orders` set *before* any grouping, we ensure that late orders aren't filtered out of the satisfaction analysis.
3. **Lateness Impact**: Added a direct comparison of `is_late` vs. `review_score`. Usually, you will see the review score drop by **2.0+ points** for late orders.

You Asked

KeyError: "Column(s) ['review_score'] do not exist"

Gemini Fast

This `KeyError` is happening because `order_reviews` often contains multiple reviews for the same `order_id` (or none at all), and your previous merge might have resulted in a column name change or a failed join.

In the Olist dataset, the `order_reviews` table is the "source of truth" for satisfaction, but it needs to be cleaned and deduplicated before it can be used in a merge, otherwise, you'll create duplicate rows in your master table.

The Fix: Robust Review Merge

Run this block to safely bring the review scores into your analysis:

```python
# 1. Deduplicate reviews (Take the latest review for each order)
clean_reviews = order_reviews.sort_values('review_answer_timestamp', ascending=False) \
                             .drop_duplicates('order_id') \
                             [['order_id', 'review_score']]

# 2. Merge into orders_master
# We check if review_score already exists to prevent duplicates
if 'review_score' in orders_master.columns:
    orders_master = orders_master.drop(columns=['review_score'])

orders_master = orders_master.merge(clean_reviews, on='order_id', how='left')

# 3. Handle missing values (Orders that never got a review)
# We fill with the median so the correlation doesn't break
orders_master['review_score'] =
orders_master['review_score'].fillna(orders_master['review_score'].median())

print("✅ review_score successfully merged into orders_master")
print(f"Average Review Score: {orders_master['review_score'].mean():.2f}")
```

Why the previous code likely failed:

1. **Name Collisions:** If `review_score` already existed in `orders_master` from a previous run, Pandas might have renamed the new one to `review_score_y`.
2. **Missing Merge:** The `orders_master` CSV you loaded at the start might not have included the join with the `order_reviews.csv` file yet.
3. **The "NaN" Problem:** If you try to run correlations on a column that is entirely `NaN` (which happens if the merge failed), you get the results you saw in your previous output.

Now, run the actual numbers:

Once the block above is finished, run this to get the real impact of lateness on satisfaction:

```python
# Actual Satisfaction Impact Numbers
satisfaction_impact = orders_master.groupby('is_late')['review_score'].agg(['mean',
'count']).rename(columns={'mean': 'avg_rating', 'count': 'total_orders'})

print("\n--- Impact of Lateness on Review Scores ---")
print(satisfaction_impact)

# Correlation between Delivery Days and Review Score
# (Assuming 'delivery_days' exists in your orders_master)
if 'delivery_days' in orders_master.columns:
    corr_val = orders_master[['delivery_days', 'review_score']].corr().iloc[0,1]
    print(f"\nCorrelation between Delivery Time and Rating: {corr_val:.4f}")
```

This comparison usually shows a massive "cliff" where on-time orders cluster around 4-5 stars, while late orders see a huge spike in 1-star reviews.

Would you like me to show you how to bin the `distance_km` to see at exactly what distance the review scores start to drop?

You Asked

give me the whole corrected thing -

```
import pandas as pd
import numpy as np



# 1. Deduplicate reviews (Take the latest review for each order)
clean_reviews = order_reviews.sort_values('review_answer_timestamp', ascending=False) \
                .drop_duplicates('order_id') \
                [['order_id', 'review_score']]

# 2. Merge into orders_master
# We check if review_score already exists to prevent duplicates
```

```
if 'review_score' in orders_master.columns:
    orders_master = orders_master.drop(columns=['review_score'])

orders_master = orders_master.merge(clean_reviews, on='order_id', how='left')

# 3. Handle missing values (Orders that never got a review)
# We fill with the median so the correlation doesn't break
orders_master['review_score'] = orders_master['review_score'].fillna(orders_master['review_score'].median())

print("✅ review_score successfully merged into orders_master")
print(f"Average Review Score: {orders_master['review_score'].mean():.2f}")

# 1. Prepare clean base data
# We merge orders with customers to get the REAL person ID (customer_unique_id)
full_orders = orders_master.merge(
    customers[['customer_id', 'customer_unique_id']],
    on='customer_id',
    how='left'
).merge(
    order_reviews[['order_id', 'review_score']],
    on='order_id',
    how='left'
)

# 2. Re-calculate Lateness (ensure no filtering issues)
full_orders['is_late'] = (
    pd.to_datetime(full_orders['order_delivered_customer_date']) >
    pd.to_datetime(full_orders['order_estimated_delivery_date'])
).astype(int)

# ──────────────────────────
# 📈 FIXED: Repeat Purchase Analysis
# ──────────────────────────
# Group by the UNIQUE human ID
customer_behavior = full_orders.groupby('customer_unique_id').agg({
    'order_id': 'count',
    'is_late': 'mean',
    'review_score': 'mean'
}).rename(columns={'order_id': 'total_orders', 'is_late': 'late_rate',
'review_score': 'avg_review'})

print("\n— 1. Repeat Purchase Stats —")
print(f"Total Unique Customers: {len(customer_behavior)}")
print(f"Customers with 2+ Orders:
{len(customer_behavior[customer_behavior['total_orders'] > 1])}")
print("\nCorrelation (Person Level):")
print(customer_behavior[['total_orders', 'late_rate',
'avg_review']].corr())


# ————————————————————
# ⭐ FIXED: Late Delivery vs. Review Scores
# ————————————————————
late_vs_review = full_orders.groupby('is_late')
['review_score'].mean()

print("\n— 2. Impact of Lateness on Reviews —")
print(late_vs_review)


# ————————————————————
# 💰 FIXED: Category Margin Numbers
# ————————————————————
# We join items to products, then to our full_orders for the margin
cat_analysis = order_items.merge(
    products[['product_id', 'product_category_name']],
on='product_id'
).merge(
    full_orders[['order_id', 'margin_proxy', 'is_late']], on='order_id'
)

print("\n— 3. Margin & Lateness by Category (Top 10) —")
cat_summary =
cat_analysis.groupby('product_category_name').agg({
    'margin_proxy': 'mean',
    'is_late': 'mean'
```

```
}).sort_values('margin_proxy', ascending=False)
print(cat_summary.head(10))


# ────────────────────
# 🚚 FIXED: High-Cost Logistics Check
# ────────────────────
print("\n— 4. Freight-to-Price Ratio (Worst Categories) —")
# Use the totals we already have in orders_master
cat_freight = cat_analysis.merge(
    orders_master[['order_id', 'total_price', 'total_freight']],
on='order_id'
)
cat_ratio = cat_freight.groupby('product_category_name').apply(
    lambda x: x['total_freight'].sum() / x['total_price'].sum()
).sort_values(ascending=False)
print(cat_ratio.head(10))
```

Gemini Fast

Here is the full, consolidated block. I have fixed the logic for the **Repeat Purchase Analysis** (using `customer_unique_id`), handled the **Review Score merge** robustly to avoid `KeyError` or suffix collisions, and cleaned up the **Category Analysis**.

I also removed the redundant merge that was causing the `review_score` error in your previous attempt.

```python
import pandas as pd
import numpy as np

# =================================================================
# 1. CLEAN & SYNC BASE DATA
# =================================================================

# Deduplicate reviews first so we don't multiply rows during the merge
clean_reviews = order_reviews.sort_values('review_answer_timestamp', ascending=False) \
                             .drop_duplicates('order_id') \
                             [['order_id', 'review_score']]

# Prepare orders_master (Standardizing column names for Price and Freight)
orders_master = orders_master.rename(columns={
    'total_price': 'order_price',
    'total_freight': 'order_freight'
})

# Drop review_score if it exists to prevent suffixes (_x, _y), then merge clean version
if 'review_score' in orders_master.columns:
    orders_master = orders_master.drop(columns=['review_score'])
orders_master = orders_master.merge(clean_reviews, on='order_id', how='left')

# Calculate Margin and Lateness at the order level
orders_master['margin_proxy'] = orders_master['order_price'] - orders_master['order_freight']
orders_master['is_late'] = (
    pd.to_datetime(orders_master['order_delivered_customer_date']) >
    pd.to_datetime(orders_master['order_estimated_delivery_date'])
).astype(int)

# Fill missing reviews with median so correlations don't break
orders_master['review_score'] =
orders_master['review_score'].fillna(orders_master['review_score'].median())

# ==================================================================
# 2. FULL ORDERS JOIN (For Person-Level Analysis)
# ==================================================================

# We merge orders with customers to get the REAL person ID (customer_unique_id)
full_orders = orders_master.merge(
    customers[['customer_id', 'customer_unique_id']],
    on='customer_id',
    how='left'
)

# ==================================================================
# 3. STATISTICAL ANALYSES
# ==================================================================

# --- 📈 1. Repeat Purchase Analysis (Person Level) ---
customer_behavior = full_orders.groupby('customer_unique_id').agg({
    'order_id': 'count',
    'is_late': 'mean',
    'review_score': 'mean'
}).rename(columns={'order_id': 'total_orders', 'is_late': 'late_rate', 'review_score':
'avg_review'})

print("\n--- 1. Repeat Purchase Stats (Using Unique Customer ID) ---")
print(f"Total Unique Customers: {len(customer_behavior)}")
print(f"Customers with 2+ Orders: {len(customer_behavior[customer_behavior['total_orders'] >
1])}")
print("\nCorrelation (Person Level):")
```

```python
print(customer_behavior[['total_orders', 'late_rate', 'avg_review']].corr())

# --- ⭐ 2. Impact of Lateness on Reviews ---
late_vs_review = full_orders.groupby('is_late')['review_score'].mean()
print("\n--- 2. Average Review Score: On-Time (0) vs Late (1) ---")
print(late_vs_review)

# --- 💰 3. Category Margin Numbers ---
cat_analysis = order_items.merge(
    products[['product_id', 'product_category_name']], on='product_id', how='left'
).merge(
    full_orders[['order_id', 'margin_proxy', 'is_late', 'order_price', 'order_freight']], on='order_id', how='left'
)

print("\n--- 3. Margin & Lateness by Category (Top 10) ---")
cat_summary = cat_analysis.groupby('product_category_name').agg({
    'margin_proxy': 'mean',
    'is_late': 'mean'
}).sort_values('margin_proxy', ascending=False)
print(cat_summary.head(10))

# --- 🚚 4. Freight-to-Price Ratio (Worst Categories) ---
print("\n--- 4. Top 10 Highest Freight-to-Price Ratio Categories ---")
cat_ratio = cat_analysis.groupby('product_category_name').apply(
    lambda x: x['order_freight'].sum() / x['order_price'].sum() if x['order_price'].sum() > 0 else 0
).sort_values(ascending=False)
print(cat_ratio.head(10))

# --- 📉 5. Operational Correlation ---
print("\n--- 5. Operational Correlation Matrix ---")
op_cols = ['distance_km', 'is_late', 'order_freight', 'order_price', 'margin_proxy', 'review_score']
print(orders_master[op_cols].corr())
```

What this code fixes:

1. **Deduplication**: By deduplicating reviews *before* merging, you ensure that an order with two reviews doesn't accidentally double the "order count" or the revenue in your analysis.
2. **Explicit Columns**: It uses `order_price` and `order_freight` consistently, preventing the `_x` and `_y` naming collisions that were breaking your code.
3. **Human-Centric Logic**: It correctly uses `customer_unique_id` to calculate repeat purchases. You should now see actual numbers in your correlation matrix instead of `NaN`.

4. **Zero-Division Handling**: Added a check (`if x['order_price'].sum() > 0`) to the freight ratio to prevent errors on free or buggy items.

Would you like me to help you interpret the correlation between **distance** and **review score** to see if there's a specific "danger zone" for shipping?