IMPORTING NECESSARIES LIBRARIES

In [3]:
```
import os
import numpy as np
import pandas as pd


import warnings
warnings.filterwarnings("ignore")
```

In [409…
```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
```

READING THE FILES AND LOAD THEM INTO DATAFRAME

In [5]:
```
folder_path = r"C:\Users\sohil\OneDrive\Desktop\Data\Data\Project Data"

df = {}
for file in os.listdir(folder_path):
    if file.endswith(".csv") and file[:-4] != "order_reviews":
        file_path = os.path.join(folder_path, file)
        df[file[:-4]] = pd.read_csv(file_path)


for key, value in df.items():
    print("*"*20, key.upper(), "*"*20)
    print(value.shape)
    display(df[key].sample(3))
    print()
```

```
******************** CUSTOMERS ********************
(99441, 5)
```

	customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state
33682	6c9265db41f93fd7cebb1ee9e90506f4	965d473350e68a4615c4804446a50ec7	21843	rio de janeiro	RJ
51703	e019c5cece8b3842ed7335d88bdab190	95742dcc3166c2d4aa27a4e9b68cd628	44572	santo antonio de jesus	BA
39969	13d270a62ccf028c0adf913af6be714f	e0c1b1f41aff706e03cd77b39835ec2e	30140	belo horizonte	MG

```
******************** GEOLOCATION ********************
(1000163, 5)
```

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state
571040	32041	-19.916598	-44.080782	contagem	MG
433272	21745	-22.879371	-43.411322	rio de janeiro	RJ
157765	6330	-23.560065	-46.830649	carapicuiba	SP

```
******************** ORDERS ********************
(99441, 8)
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at
1635	b28cd85f7b7464e08e64f88fc6f6123e	142971bb683b69e32bf1b691f9acc928	delivered	2017-06-02 08:48:11	2017-06-02 09:03:35
50790	9db3045b1374d5f2d301c412081ab1ee	5fab7785cdfea700ae1362b2c9c3d697	delivered	2018-03-16 23:42:40	2018-03-17 00:49:26
51127	b4ee8b4e6cdba36455ad826db9938e35	ed3fe46fbc557470fab546231b8b44dd	delivered	2018-01-18 22:44:34	2018-01-18 22:56:21

```
******************** ORDER_ITEMS ********************
(112650, 7)
```

	order_id	order_item_id	product_id	seller_id	shipping_li
22629	339c4a8fdff27916641be1118b71ddbe	1	87064fd995f81ddb8e735902047fe007	e8f6dc8e6a1dcde89d20e3995c8d90b3	2017-11-17
10088	172589c3df7261dca954a6c662c38acc	1	aca2eb7d00ea1a7b8ebd4e68314663af	955fee9216a65b617aa5c0531780ce60	2018-01-24
99105	e0ea34baecdf184089f368a8ba575907	1	3516632e8f52b679ff83d1665ecc990e	e9bc59e7b60fc3063eb2290deda4cced	2017-06-22

```
******************** ORDER_PAYMENTS ********************
(103886, 5)
```

	order_id	payment_sequential	payment_type	payment_installments	payment_value
23156	be9352379a16b3dbe8ef3a66a85b656f	1	credit_card	8	150.44
94069	e8e55eaa809f3091cc5060cc53735660	1	credit_card	3	83.01
97716	24991dc7bdf90c687d06964b2459932c	1	credit_card	1	128.19

```
******************* PRODUCTS *******************
(32951, 9)
```

	product_id	product_category_name	product_name_lenght	product_description_lenght	product_photos_qty	p
13017	fb2631932085fc37d651d3408aa65a8c	esporte_lazer	40.0	1343.0	4.0	
2550	abe356a1d236588f0fdb8099527108d2	ferramentas_jardim	57.0	417.0	1.0	
12988	d982791b03ad95faaa03fc746ad04672	informatica_acessorios	35.0	1543.0	3.0	

```
******************* SELLERS *******************
(3095, 4)
```

	seller_id	seller_zip_code_prefix	seller_city	seller_state
873	e46bc031f2c5bae4ccb40bb90712e9b4	5174	sao paulo	SP
2807	0be8ff43f22e456b4e0371b2245e4d01	4461	sao paulo	SP
1145	749e7cdabbaf72f16677859e27874ba5	7122	guarulhos	SP

DESCRIPTIVE STAT OF THE DATA

In [147…

```
not_describe = ['customers', 'geolocation', 'sellers']
for key, value in df.items():
    if key not in not_describe:
        display(df[key].describe())
```

	purchase_timestamp	approved_at	delivered_carrier_date	delivered_customer_date	estimated_delivery_date	quantity	
count	99441	99281	97658	96476	99441	99441.000000	96462
mean	2017-12-31 08:43:12.776581120	2017-12-31 18:35:24.098800128	2018-01-04 21:49:48.138278656	2018-01-14 12:09:19.035542272	2018-01-24 03:08:37.730111232	9.997235	11
min	2016-09-04 21:15:19	2016-09-15 12:16:38	2016-10-08 10:34:01	2016-10-11 13:46:32	2016-09-30 00:00:00	1.000000	-7
25%	2017-09-12 14:46:19	2017-09-12 23:24:16	2017-09-15 22:28:50.249999872	2017-09-25 22:07:22.249999872	2017-10-03 00:00:00	5.000000	6
50%	2018-01-18 23:04:36	2018-01-19 11:36:13	2018-01-24 16:10:58	2018-02-02 19:28:10.500000	2018-02-15 00:00:00	10.000000	9
75%	2018-05-04 15:42:16	2018-05-04 20:35:10	2018-05-08 13:37:45	2018-05-15 22:48:52.249999872	2018-05-25 00:00:00	15.000000	15
max	2018-10-17 17:30:18	2018-09-03 17:40:06	2018-09-11 19:48:28	2018-10-17 13:22:46	2018-11-12 00:00:00	19.000000	208
std	NaN	NaN	NaN	NaN	NaN	5.482953	9

	order_item_id	price	freight_value
count	112650.000000	112650.000000	112650.000000
mean	1.197834	120.653739	19.990320
std	0.705124	183.633928	15.806405
min	1.000000	0.850000	0.000000
25%	1.000000	39.900000	13.080000
50%	1.000000	74.990000	16.260000
75%	1.000000	134.900000	21.150000
max	21.000000	6735.000000	409.680000

	payment_sequential	payment_installments	payment_value
count	103886.000000	103886.000000	103886.000000
mean	1.092679	2.853349	154.100380
std	0.706584	2.687051	217.494064
min	1.000000	0.000000	0.000000
25%	1.000000	1.000000	56.790000
50%	1.000000	1.000000	100.000000
75%	1.000000	4.000000	171.837500
max	29.000000	24.000000	13664.080000

	name_lenght	description_lenght	photos_qty	weight_g	length_cm	height_cm	width_cm
count	32341.000000	32341.000000	32341.000000	32340.000000	32340.000000	32340.000000	32340.000000
mean	48.476949	771.495285	2.188986	2276.956586	30.854545	16.958813	23.208596
std	10.245741	635.115225	1.736766	4279.291845	16.955965	13.636115	12.078762
min	5.000000	4.000000	1.000000	0.000000	7.000000	2.000000	6.000000
25%	42.000000	339.000000	1.000000	300.000000	18.000000	8.000000	15.000000
50%	51.000000	595.000000	1.000000	700.000000	25.000000	13.000000	20.000000
75%	57.000000	972.000000	3.000000	1900.000000	38.000000	21.000000	30.000000
max	76.000000	3992.000000	20.000000	40425.000000	105.000000	105.000000	118.000000

In [ ]:

BASIC CLEANING

In [ ]:
```
# UPDATING SELLERS TABLE HEADERS
df['sellers'].columns = df['sellers'].columns.str.replace("seller_","")\
                        .str.replace("_prefix", "")
```

In [352…
```
# NULL & DUPLICATE IN SELLERS DATA
df['sellers'].isna().sum().sum(), df['sellers'].duplicated().sum()
```

Out[352…
```
(0, 0)
```

In [ ]:

In [355…
```
# UPDATING PRODUCT TABLE HEADERS & NULL AND DUPLICATE FOR THAT DATA
df['products'].columns = df['products'].columns.str.replace('product_', "")
df['products'].isna().sum().sum(), df['products'].duplicated().sum()
```

Out[355…
```
(2448, 0)
```

In [357…
```
# NULL VALUES PERCENTAGE IN PRODUCTS COLUMNS
df['products'].isna().sum().apply(lambda x : (x/len(df['products'])*100))
```

Out[357…
```
id                    0.000000
category_name         1.851234
name_lenght           1.851234
description_lenght    1.851234
photos_qty            1.851234
weight_g              0.006070
length_cm             0.006070
height_cm             0.006070
width_cm              0.006070
dtype: float64
```

In [359…
```
# FILTERED OUT THE NULL VALUES IN PRODUCTS DATA
df['products'] = df['products'][(~df['products']['category_name'].isna())]
```

In [ ]:

In [362…
```
# UPDATING CUSTOMER TABLE COLUMNS NAMES
df['customers'].columns = df['customers'].columns.str.replace("customer_","").str.replace("_prefix", "")
```

In [364…
```
# THE NULL AND DUPLICATE VALUES IN CUSTOMERS TABLE
df['customers'].isna().sum().sum(), df['customers'].duplicated().sum().sum()
```

Out[364…
```
(0, 0)
```

In [366…
```
# UPDATING GEOLOCATION HEADERS
df['geolocation'].columns = df['geolocation'].columns.str.replace('geolocation_',"")\
```

```
                    .str.replace("_prefix", "")\
                    .str.replace("lat", "latitude")\
                    .str.replace("lng", "longitute")
```

In [368…
```
# NULL AND DUPLICATES IN GEOLOCATION
df['geolocation'].isna().sum().sum(), df['geolocation'].duplicated().sum().sum()
```

Out[368…
```
(0, 261831)
```

In [369…
```
# DROPPING DUPLICATES
df['geolocation'].drop_duplicates(keep='first', inplace=True)
```

In [ ]:

In [373…
```
# UPDATING COLUMNS HEADERS IN ORDERS
df['orders'].columns = df['orders'].columns.str.replace('orders_',"")
df['orders'].columns = df['orders'].columns.str.replace('order_',"")
```

In [375…
```
# UPDATING THE DATATYPES
date_col = ['purchase_timestamp', 'approved_at', 'delivered_carrier_date', 'delivered_customer_date', 'estimated_delivery_date

for col in df['orders'].columns:
    if col in date_col:
        df['orders'][col] = pd.to_datetime(df['orders'][col])
```

In [377…
```
df['orders']['quantity'] = np.random.randint(1,20, size=len(df['orders']))
```

In [ ]:

In [ ]:

BUSINESS PROBLEMS

PROBELM 1.

Finance team wants to optimize payment infrastructure.

- Revenue contribution by payment type (credit card, boleto, etc.)
- Average order value by payment type

Installment behavior:

- Do higher installments lead to higher cart value?
- Failed or canceled order impact on revenue

In [383…
```
# Revenue contribution by payment type (credit card, boleto, etc.
revenue = df['order_payments'].groupby('payment_type')['payment_value'].sum()\
                    .reset_index()\
                    .sort_values(by='payment_value', ascending= False)
revenue = revenue[revenue['payment_type']!= 'not_defined']
revenue
```

Out[383…

	payment_type	payment_value
1	credit_card	12542084.19
0	boleto	2869361.27
4	voucher	379436.87
2	debit_card	217989.79

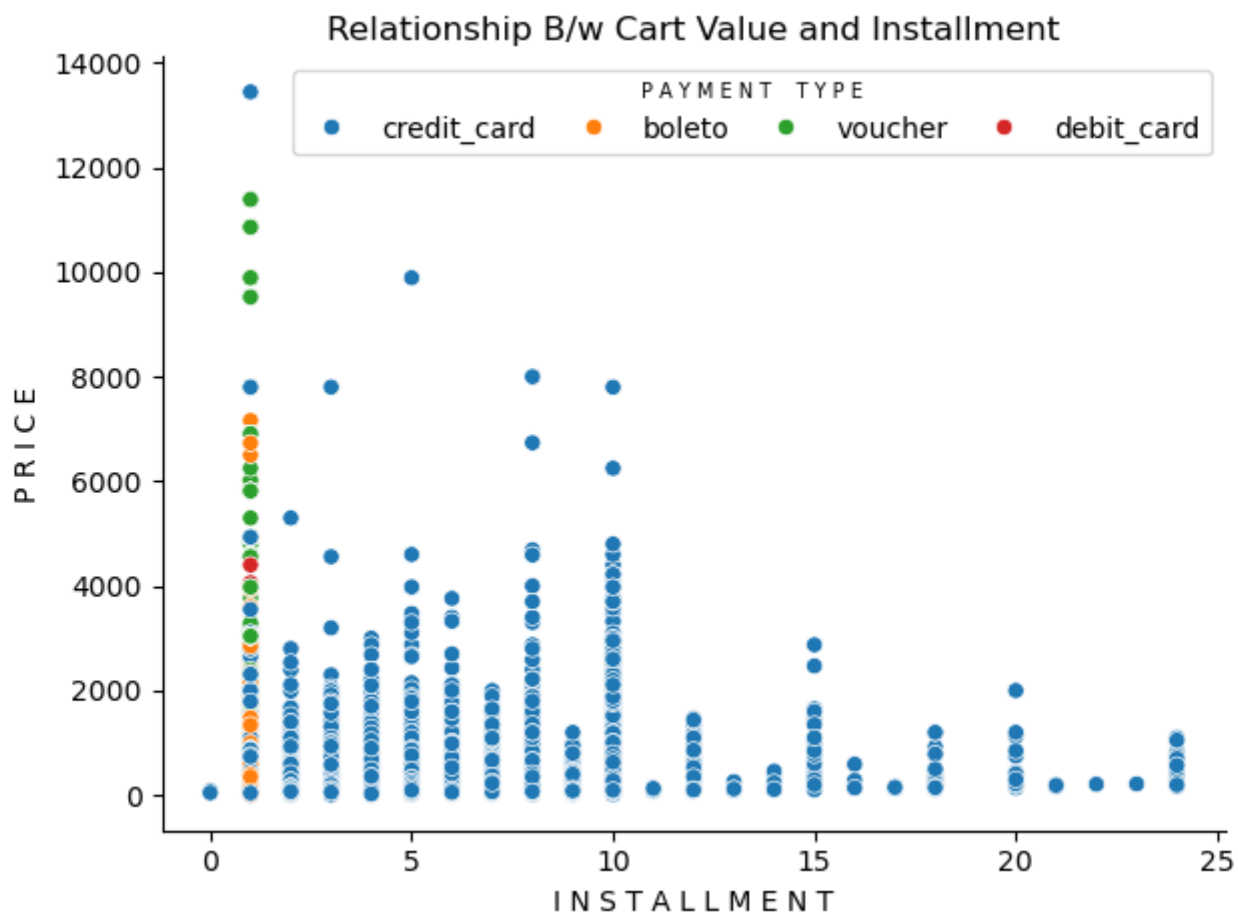
In [106…
```
# Average order value by payment type
temp = df['orders'].merge(
    df['order_items'],
    how='inner',
    left_on='id',
    right_on='order_id'
)[['quantity', 'price', 'id']]

temp1 = temp.merge(
    df['order_payments'],
    how='inner',
    left_on='id',
    right_on='order_id')

temp1['revenue'] = temp1['quantity'] * temp1['price']
result = temp1.groupby('payment_type')['revenue'].mean()

result.reset_index().sort_values('revenue', ascending=False)
```

Out[106…

	payment_type	revenue
1	credit_card	1259.429415
2	debit_card	1096.679326
0	boleto	1042.586711
3	voucher	1022.350907

In [451…

```
# Do higher installments lead to higher cart value?
cart_value = temp1.groupby('id')['price'].sum().reset_index()
cart_value = cart_value.merge(df['order_payments'], how='inner', left_on='id', right_on='order_id')[['price', 'payment_install
                                                                                                      'payment_value', 'payment_


sns.scatterplot(x='payment_installments',
               y='price', data=cart_value,
               hue='payment_type')
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.title('Relationship B/w Cart Value and Installment')
plt.xlabel('I N S T A L L M E N T')
plt.ylabel('P R I C E')

ax.legend(
    title='P A Y M E N T    T Y P E',
    title_fontsize=7,
    ncol=4,              # all legend items in one row
    frameon=True,        # rectangular legend box
    handlelength=2,    # makes legend markers rectangular
    handleheight=1,
    columnspacing=1,
    markerscale=1
)

plt.tight_layout()
plt.show()
```



CONCLUSION

Credit card dominates installment usage

- Only credit card payments spread across multiple installments (1–24).
- Other payment types (boleto, voucher, debit card) are mostly at 1 installment.

Higher prices tend to appear at lower installments

- The highest prices (₹ 10k – ₹ 14k range) appear mostly at: 1–4 installments
- As installments increase (> 10): Prices are much lower and tightly clustered

No strong linear relationship

- There is no direct linear trend between: Installments and price

- But a clear behavioral pattern exists: High price → fewer installments | Many installments → smaller order value

In [ ]:

PROBLEM 2

Product Performance & Category Analysis

Category managers want to improve product portfolio.

- Top revenue-generating product categories

Products with:

- Does product weight/size impact delivery time?

In [502…

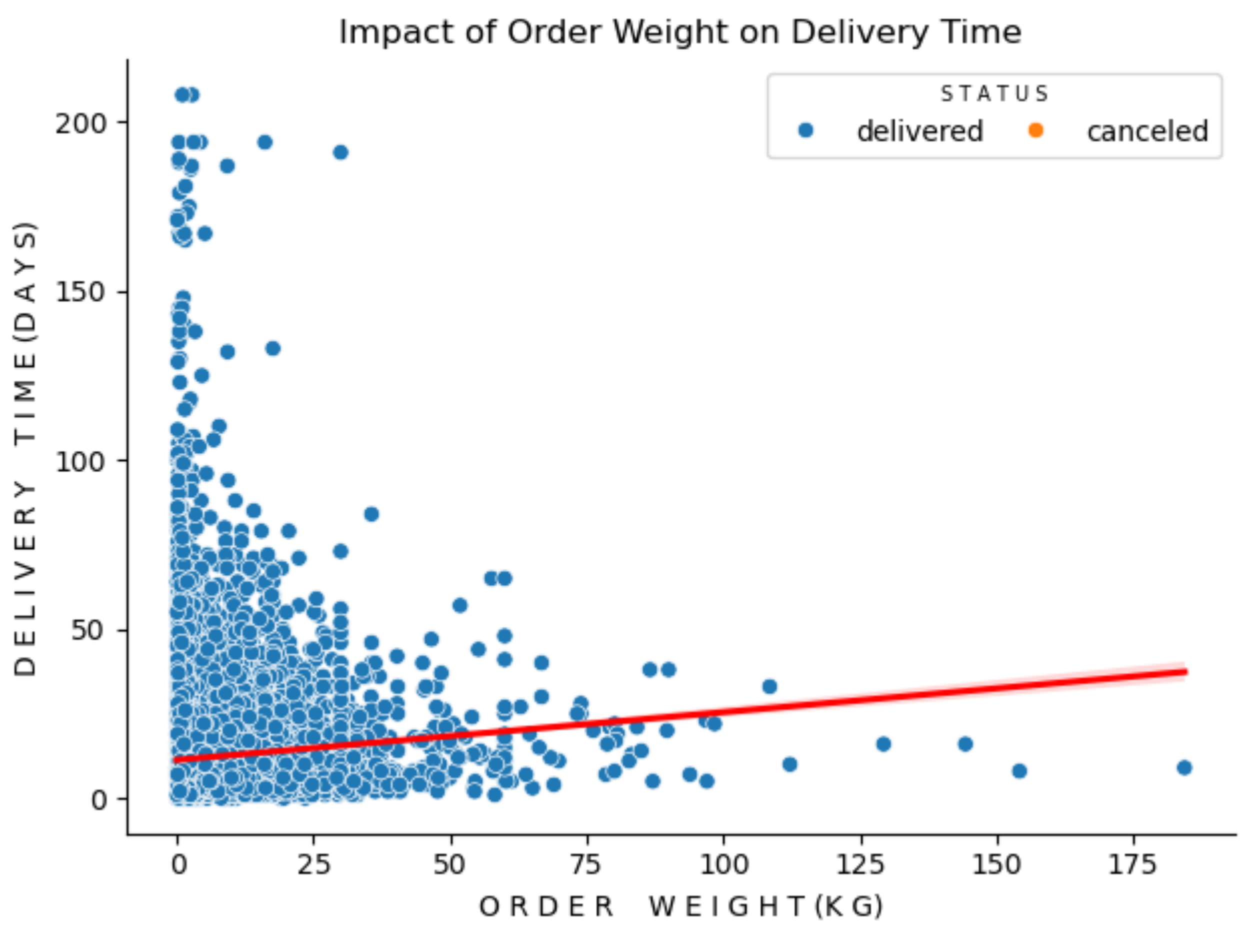
```
# Top 10 revenue-generating product categories
temp = df['products'].merge(df['order_items'], how='inner',
                            left_on='id', right_on='product_id')[['order_id','category_name','price']]
temp1= temp.merge(df['orders'], how='inner',
           left_on='order_id', right_on='id')[['category_name','price','status','quantity']]


temp1['revenue'] = temp1['quantity'] * temp1['price']
top_categories = temp1.groupby('category_name')['revenue'].sum().reset_index()\
                                                        .sort_values(by='revenue', ascending=False).head(10)
top_categories['category_name']= top_categories['category_name'].str.title().str.replace('_',' ')
top_categories
```

Out[502…

	category_name	revenue
11	Beleza Saude	12555790.14
66	Relogios Presentes	12080765.91
13	Cama Mesa Banho	10366265.07
32	Esporte Lazer	9646493.45
44	Informatica Acessorios	9200577.36
54	Moveis Decoracao	7379808.21
26	Cool Stuff	6342763.02
72	Utilidades Domesticas	6233688.00
8	Automotivo	5794215.23
40	Ferramentas Jardim	4948716.31

In [116…

```python
# Does product weight/size impact delivery time?
df['orders']['days'] = (df['orders']['delivered_customer_date'] - df['orders']['approved_at']).dt.days
orders = df['orders'].dropna(
    subset=['delivered_customer_date', 'approved_at'])
final_df['status'] = final_df['status'].str.replace('canceled','cancelled')

orders['days'] = (orders['delivered_customer_date'] - orders['approved_at']).dt.days
orders = orders[orders['days'] >= 0]

order_weight = (
    df['order_items']
    .merge(df['products'], left_on='product_id', right_on='id')
    .groupby('order_id')['weight_g']
    .sum()
    .reset_index() )

final_df = orders.merge(order_weight, left_on='id', right_on='order_id')

sns.scatterplot(
    data=final_df,
    x=final_df['weight_g'] / 1000,   # kg
    y='days', hue='status')

sns.regplot(
    data=final_df, x=final_df['weight_g'] / 1000, y='days', scatter=False, color='red')

plt.xlabel('O R D E R    W E I G H T (K G)')
plt.ylabel('D E L I V E R Y    T I M E (D A Y S)')
plt.title('Impact of Order Weight on Delivery Time')


ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

ax.legend(
    title='S T A T U S', title_fontsize=7, ncol=2, frameon=True, handlelength=2,
    handleheight=1, columnspacing=0.8, markerscale=1)

plt.tight_layout()
plt.show()
```



CONCLUSION

Relationship between weight and delivery time is weakly positive

- The red trend line slopes upward, which means: As order weight increases, delivery time tends to increase slightly
- However, the slope is not steep: Order weight has some impact, but it is not a strong driver of delivery time.

High variability at low weights

- For lightweight orders (0–20 kg): Delivery time ranges from a few days to over 200 days
- This large spread tells us: Weight alone cannot explain delivery delays

Order weight shows a weak positive relationship with delivery time. While heavier orders tend to take slightly longer, delivery duration is largely influenced by other operational and logistical factors. The high variability among lightweight orders indicates that weight alone is insufficient to predict delivery performance.”

In [ ]:

```

```

PROBLEM 3

Order Fulfillment & Delivery Performance Analysis

Management wants to understand delivery efficiency and customer experience.

- What percentage of orders are delivered on time vs late?
- Average delivery time (order → customer) by: Customer state?

In [117…

```python
# What percentage of orders are delivered on time vs late?

orders = df['orders'].copy()
delivered_orders = orders[
    orders['status'] == 'delivered'
].copy()

delivered_orders['delivery_status'] = (
    delivered_orders['delivered_customer_date']
    <= delivered_orders['estimated_delivery_date'])

delivered_orders['delivery_status'] = delivered_orders['delivery_status'].map(
    {True: 'On Time', False: 'Late'})

delivery_pct = (
    delivered_orders['delivery_status']
    .value_counts(normalize=True) * 100 ).reset_index()

delivery_pct.columns = ['Delivery Status', 'Percentage']
delivery_pct
```

Out[117…

	Delivery Status	Percentage
0	On Time	91.880014
1	Late	8.119986

In [807…

```python
# Average delivery time (order → customer) by:--->> Customer state

delivered_orders = df['orders'][
    df['orders']['status'] == 'delivered'
].copy()

delivered_orders['delivery_days'] = (
    delivered_orders['delivered_customer_date']
    - delivered_orders['purchase_timestamp']
).dt.days

cust_orders = delivered_orders.merge(
    df['customers'][['id', 'state']],
    left_on='customer_id', right_on='id', how='left')

cust_state_delivery = (
    cust_orders
    .groupby('state', as_index=False)
    .agg(avg_delivery_days=('delivery_days', 'mean'))
    .sort_values('avg_delivery_days', ascending=False) )

cust_state_delivery.head()
```

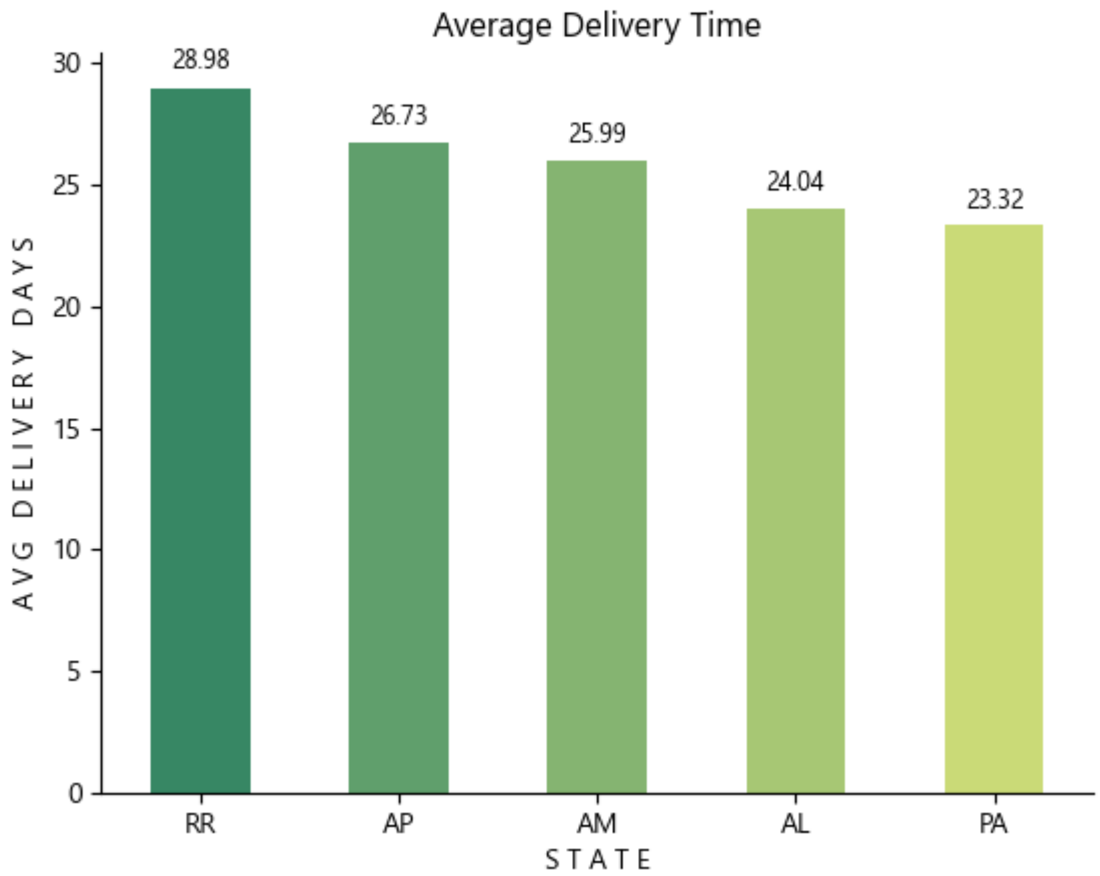
Out[807…

	state	avg_delivery_days
21	RR	28.975610
3	AP	26.731343
2	AM	25.986207
1	AL	24.040302
13	PA	23.316068

In [147…

```python
plt.title('Average Delivery Time')
sns.barplot(data=cust_state_delivery.head(), x='state', y='avg_delivery_days', palette='summer', width=0.5)
plt.xlabel('S T A T E')
plt.ylabel('A V G   D E L I V E R Y   D A Y S')

for x,y in cust_state_delivery.head().values:
    plt.text(x, y + 0.02*y, f"{y:.2f}", ha='center', va='bottom', fontsize=9)
```

```
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.show()
```



CONCLUSION:

- The analysis shows that the majority of orders (91.88%) are delivered on time, indicating strong overall logistics and fulfillment performance. However, 8.12% of orders are delivered late, which still represents a meaningful volume at scale and can negatively impact customer satisfaction. This delay segment highlights opportunities for operational improvements, particularly in seller performance and last-mile delivery efficiency. Targeted interventions in high-delay regions and with underperforming sellers could further enhance service reliability.
- The state RR is at the top in Average delivery Days and followed by AP.

In [ ]:

PROBLEM 3

Logistics & Freight Cost Analysis

Logistics team wants to reduce shipping costs

- Freight cost as a % of product price
- Top 5 cities with highest average freight cost
- Top 5 cities with lowest average freight cost and average price ?
- Identify products where freight > product price (loss risk) & their precentage ?

In [910…

```
# Freight cost as a % of product price

freight_pct = ((df['order_items']['freight_value'] / df['order_items']['price']) * 100).mean()
print(f"Freight cost as a % of product price: {freight_pct}")
```

```
Freight cost as a % of product price: 32.08635490797801
```

In [929…

```
# Top 5 cities with highest average freight cost

temp = df['order_items'].merge(df['sellers'], how='left',
                               left_on= 'seller_id', right_on='id')[['city','freight_value','price']]
temp.groupby('city')['freight_value'].mean()\
                                        .reset_index().sort_values(by='freight_value',
                                                            ascending= False).head()
```

Out[929…

	city	freight_value
291	lages - sc	168.533333
498	sao francisco do sul	150.220000
94	california	143.775000
490	sao jose dos pinhais	142.400000
366	nova trento	131.850000

In [941…

```
# Top 5 cities with lowest average freight cost

temp = df['order_items'].merge(df['sellers'], how='left',
                               left_on= 'seller_id', right_on='id')[['city','freight_value','price']]
low_freight= temp.groupby('city')['freight_value'].mean()\
                                        .reset_index().sort_values(by='freight_value',
                                                            ascending= True).head()

low_price= temp.groupby('city')['price'].mean()\
                                        .reset_index().sort_values(by='price',
                                                            ascending= True).head()
display(low_freight, low_price)
```

	city	freight_value
267	jacarei / sao paulo	8.602222
519	sao paulo / sao paulo	9.170000
84	brotas	9.512500
522	sao pauo	9.560000
116	carapicuiba / sao paulo	11.103333

	city	price
84	brotas	6.25
382	palotina	9.99
187	floranopolis	9.99
311	macatuba	13.00
279	jarinu	14.63

In [965…

```
# Identify products where freight > product price (loss risk)
df['order_items'][df['order_items']['freight_value'] > df['order_items']['price']].head()
```

Out[965…

	order_id	order_item_id	product_id	seller_id	shipping_li
58	0025081dcf9330f9a5052ae82c6ce396	1	4e3f399366b0047a572b6682f9bb166e	5f3ae9136c875522250f8184f253413a	2018-04-02
80	002f98c0f7efd42638ed6100ca699b42	1	d41dc2f2979f52d75d78714b378d4068	7299e27ed73d2ad986de7f7c77d919fa	2017-08-10
110	003edccf16bc5ec447f592913b3df2b4	1	500870614ddcf5bd84f7d26861026c8a	ef506c96320abeedfb894c34db06f478	2018-07-12
125	00482f2670787292280e0a8153d82467	1	a9c404971d1a5b1cbc2e4070e02731fd	702835e4b785b67a084280efca355756	2017-02-17
156	00602f25bffa1dcfb71e202fbf9824fb	1	32a8448d1612773bcfd0c5a8dd235e4e	86ccac0b835037332a596a33b6949ee1	2017-11-08

In [984…

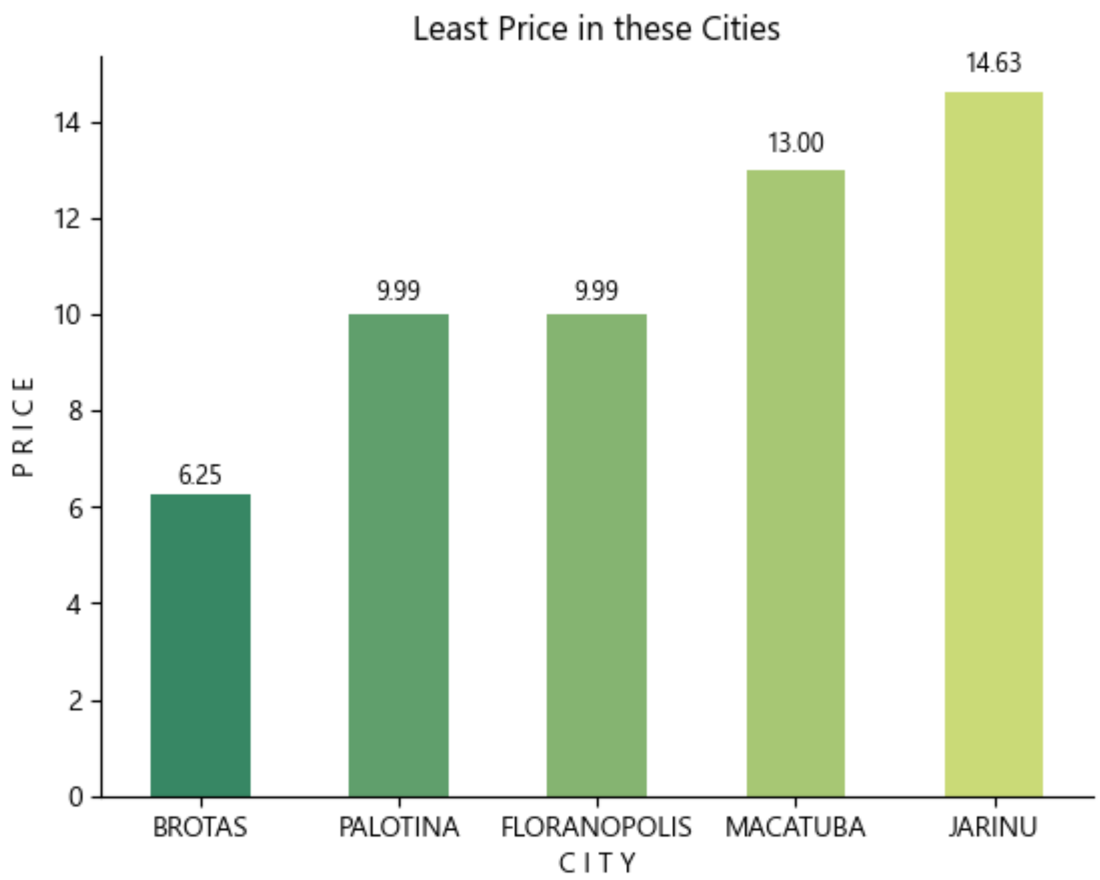
```
# Percentage of products where freight > product price (loss risk)
freight_pct = len(df['order_items'][df['order_items']['freight_value'] > df['order_items']['price']]) / len(df['order_items'])
print(f'The tottal Percentage of products where freight > product price is : {freight_pct}')
```

```
The tottal Percentage of products where freight > product price is : 3.6608965823346646
```

In [147…

```
plt.title('Least Price in these Cities')
sns.barplot(data=low_price, x='city', y='price', palette='summer', width=0.5,
            errorbar=('ci', 95))
plt.ylabel('P R I C E')
plt.xlabel('C I T Y')

for i, y in enumerate(low_price['price']):
    plt.text(i, y + 0.02*y, f"{y:.2f}", ha='center', va='bottom', fontsize=9)
```

```
plt.xticks(range(len(low_price)), low_price['city'].str.upper())

ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.show()
```



In [ ]:

PROBLEM 4

Demand Trend & Seasonality Analysis

Operations want better inventory planning.

- Monthly and weekly order trends
- Peak demand periods

In [144…

```
orders = df['orders'].copy()

valid_orders = orders[
    orders['status'].isin(['delivered', 'shipped', 'invoiced'])
].copy()

valid_orders['order_month'] = valid_orders['purchase_timestamp'].dt.to_period('M')
valid_orders['order_week'] = valid_orders['purchase_timestamp'].dt.to_period('W')
valid_orders['order_date'] = valid_orders['purchase_timestamp'].dt.date
valid_orders['weekday'] = valid_orders['purchase_timestamp'].dt.day_name()

monthly_trend = (valid_orders.groupby('order_month', as_index=False).agg(total_orders=('id', 'count')))

weekly_trend = (valid_orders.groupby('order_week', as_index=False).agg(total_orders=('id', 'count')))
```

In [147…

```
monthly_trend['rolling_avg'] = monthly_trend['total_orders'].rolling(3).mean()

peak_periods = monthly_trend[
    monthly_trend['total_orders'] > monthly_trend['rolling_avg']]
peak_periods.sample(3)
```

Out[147…

	order_month	total_orders	rolling_avg
6	2017-04-01	2366	2215.000000
5	2017-03-01	2594	1685.666667
16	2018-02-01	6618	6453.000000

In [146…

```
monthly_trend['rolling_avg'] = monthly_trend['total_orders'].rolling(3).mean()

peak_periods = monthly_trend[
    monthly_trend['total_orders'] > monthly_trend['rolling_avg']]

peak_periods.sample(3)
```

Out[146...

	order_month	total_orders	rolling_avg
18	2018-04-01	6911	6896.000000
3	2017-01-01	778	356.666667
7	2017-05-01	3617	2859.000000

In [ ]:

DASHBOARD

In [148...

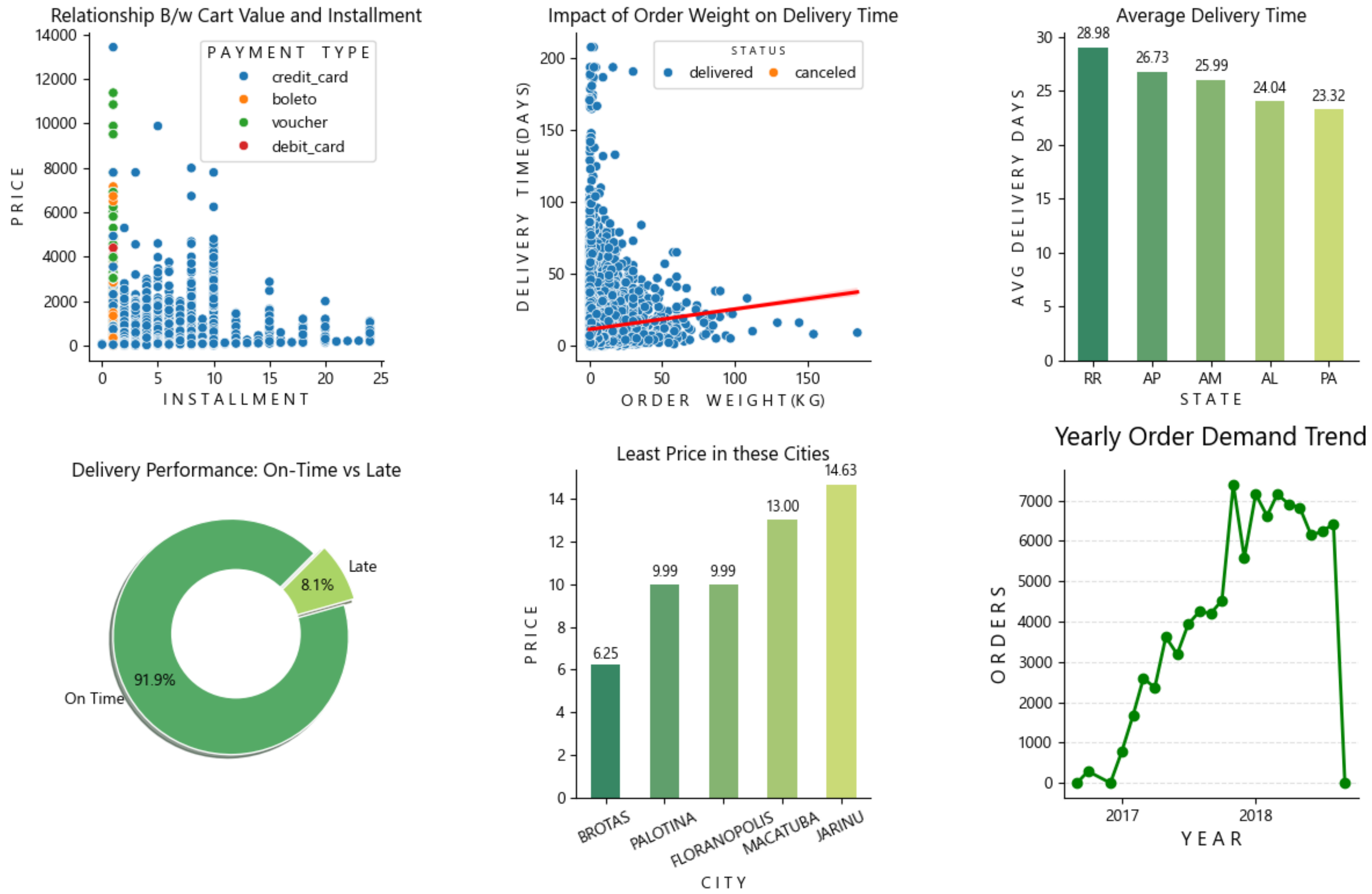
```
fig = plt.figure(figsize=(12,12))

# =============== KPI's ===============
Total_Orders = df['orders']['id'].count()
Total_Revenue = temp1['revenue'].sum()
Total_Customers = df['customers']['id'].nunique()
Total_Sellers = df['sellers']['id'].nunique()
Average_Price = df['order_items']['price'].mean().round(2)
Average_Freight = df['order_items']['freight_value'].mean().round(2)
Avg_Delivery_Time = df['orders']['days'].mean().round(2)


# =============== KPI's Charts ===============
plt.subplot2grid((3,6),(0,0))
plt.axis('off')
plt.text(0.5, 0.55, f"{Total_Orders:,}", fontsize=20, ha='center',fontweight='bold',family='DejaVu Sans', color='red')
plt.text(0.5, 0.45, "Total Orders", fontsize=12, ha='center')

plt.subplot2grid((3,6),(0,1))
plt.axis('off')
plt.text(0.5, 0.55, f"{Total_Customers:,}", fontsize=20, ha='center',fontweight='bold', family='DejaVu Sans', color='red')
plt.text(0.5, 0.45, "Total Customers", fontsize=12, ha='center')

plt.subplot2grid((3,6),(0,2))
plt.axis('off')
plt.text(0.5, 0.55, f"{Total_Sellers:,}", fontsize=20, ha='center',fontweight='bold', family='DejaVu Sans', color='red')
plt.text(0.5, 0.45, "Total Sellers", fontsize=12, ha='center')

plt.subplot2grid((3,6),(0,3))
plt.axis('off')
plt.text(0.5, 0.55, f"₹{Average_Price:,.2f}", fontsize=20, ha='center',fontweight='bold', family='DejaVu Sans', color='red')
plt.text(0.5, 0.45, "Avg Price", fontsize=12, ha='center')

plt.subplot2grid((3,6),(0,4))
plt.axis('off')
plt.text(0.5, 0.55, f"₹{Average_Freight:,.2f}", fontsize=20, ha='center', fontweight='bold', family='DejaVu Sans', color='red'
plt.text(0.5, 0.45, "Avg Freight", fontsize=12, ha='center')

plt.subplot2grid((3,6),(0,5))
plt.axis('off')
plt.text(0.5, 0.55, f"{Avg_Delivery_Time:.1f}", fontsize=20, ha='center', fontweight='bold', color='red', family='DejaVu Sans'
plt.text(0.5, 0.45, "Avg Delivery (Days)", fontsize=12, ha='center');


# =============== SCATTER PLOT ===============
plt.subplot2grid((3,3), (1,0))
# Do higher installments lead to higher cart value?
cart_value = temp1.groupby('id')['price'].sum().reset_index()
cart_value = cart_value.merge(df['order_payments'], how='inner', left_on='id', right_on='order_id')[['price', 'payment_install
                                                                                                      'payment_value', 'payment_
sns.scatterplot(x='payment_installments', y='price', data=cart_value, hue='payment_type')
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title('Relationship B/w Cart Value and Installment')
plt.xlabel('I N S T A L L M E N T')
plt.ylabel('P R I C E')
ax.legend(
    title='P A Y M E N T    T Y P E')


# ======================= SCATTER PLOT 2 =======================
df['orders']['days'] = (df['orders']['delivered_customer_date'] - df['orders']['approved_at']).dt.days
orders = df['orders'].dropna( subset=['delivered_customer_date', 'approved_at'])
final_df['status'] = final_df['status'].str.replace('canceled','cancelled')

orders['days'] = (orders['delivered_customer_date'] - orders['approved_at']).dt.days
orders = orders[orders['days'] >= 0]

order_weight = ( df['order_items'] .merge(df['products'], left_on='product_id', right_on='id') .groupby('order_id')['weight_g'
final_df = orders.merge(order_weight, left_on='id', right_on='order_id')
```

```python
plt.subplot2grid((3,3), (1,1))
sns.scatterplot(data=final_df, x=final_df['weight_g'] / 1000,y='days', hue='status')
sns.regplot(data=final_df, x=final_df['weight_g'] / 1000, y='days', scatter=False, color='red')

plt.xlabel('O R D E R    W E I G H T (K G)')
plt.ylabel('D E L I V E R Y    T I M E (D A Y S)')
plt.title('Impact of Order Weight on Delivery Time')

ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.legend(title='S T A T U S', title_fontsize=7, ncol=2, frameon=True, handlelength=1, handleheight=1, columnspacing=0.7, mark



# ============================= BAR CHART DELIVERY PERFORMENCE ===============================
plt.subplot2grid((3,3), (1,2))
plt.title('Average Delivery Time')
sns.barplot(data=cust_state_delivery.head(), x='state', y='avg_delivery_days', palette='summer', width=0.5)
plt.xlabel('S T A T E')
plt.ylabel('A V G   D E L I V E R Y   D A Y S')

for x,y in cust_state_delivery.head().values:
    plt.text(x, y + 0.02*y, f"{y:.2f}", ha='center', va='bottom', fontsize=9)

ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)



# ================================ PIE CHART FOR ON-TIME VS LATE ======================================
plt.subplot2grid((3,3), (2,0))
colors = sns.color_palette("summer", len(delivery_pct))
plt.pie(delivery_pct['Percentage'], labels=delivery_pct['Delivery Status'], autopct='%1.1f%%', startangle=45, colors=colors,
    explode=[0.05]*len(delivery_pct), pctdistance=0.75, labeldistance=1.05,wedgeprops={'edgecolor': 'white', 'linewidth': 1},
# Donut effect (professional look)
centre_circle = plt.Circle((0,0), 0.55, fc='white')
plt.gca().add_artist(centre_circle)
plt.title( 'Delivery Performance: On-Time vs Late')


# ================================ BAR CHART FOR LEAST PRICE ======================================
plt.subplot2grid((3,3), (2,1))
plt.title('Least Price in these Cities')
sns.barplot(data=low_price, x='city', y='price', palette='summer', width=0.5, errorbar=('ci', 95))
plt.ylabel('P R I C E')
plt.xlabel('C I T Y')

for i, y in enumerate(low_price['price']):
    plt.text(i, y + 0.02*y, f"{y:.2f}", ha='center', va='bottom', fontsize=9)
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.xticks(range(len(low_price)), low_price['city'].str.upper(), rotation=25)

plt.tight_layout()
plt.rcParams['font.family'] = 'Segoe UI Emoji'
plt.suptitle("📊⚡ E-Commerce KPI Dashboard 💰🛒", fontsize=25, color='red',y=0.95)


# ========================================= LINE PLOT ============================================
orders = df['orders'].copy()
valid_orders = orders[orders['status'].isin(['delivered', 'shipped', 'invoiced'])].copy()
valid_orders['order_month'] = valid_orders['purchase_timestamp'].dt.to_period('M')
valid_orders['order_week'] = valid_orders['purchase_timestamp'].dt.to_period('W')
valid_orders['order_date'] = valid_orders['purchase_timestamp'].dt.date
valid_orders['weekday'] = valid_orders['purchase_timestamp'].dt.day_name()
monthly_trend = (valid_orders.groupby('order_month', as_index=False).agg(total_orders=('id', 'count')))
weekly_trend = (valid_orders.groupby('order_week', as_index=False).agg(total_orders=('id', 'count')))
#-------------------------------------------------------------------------------------
# PLOTTING - LINE
monthly_trend['order_month'] = monthly_trend['order_month'].dt.to_timestamp()
plt.subplot2grid((3,3), (2,2))
plt.plot(monthly_trend['order_month'],monthly_trend['total_orders'],marker='o',linewidth=2, color='green')

plt.title('Yearly Order Demand Trend', fontsize=16, fontweight='bold', pad=15)
plt.xlabel('Y E A R', fontsize=12)
plt.ylabel('O R D E R S', fontsize=12)

# Show only year on x-axis
ax = plt.gca()
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

# Styling
plt.grid(axis='y', linestyle='--', alpha=0.4)
```

```
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.tight_layout()
plt.show()



plt.show()
```

📊 ⚡ E-Commerce KPI Dashboard 💰 🛒

99,441	99,441	3,095	₹120.65	₹19.99	11.6
Total Orders	Total Customers	Total Sellers	Avg Price	Avg Freight	Avg Delivery (Days)




```
In [ ]:
```