
DATA SCIENCE PROJECT

# Team Number : 4

- **Our Names :**
- Seif Mahmoud Ahmed
- Yassein Ahmed Mohamed
- Mohamed Ayman

# Sales Forecasting and Optimization

- **Project Overview:**
- The **Sales Forecasting and Optimization** project aims to predict future sales for a retail or e-commerce
- business by using historical sales data. The project involves data collection, cleaning, exploration, time-series
- forecasting model development, optimization, and deployment. The end goal is to have a model that can
- generate accurate sales predictions to help businesses optimize inventory, marketing, and sales strategies.

# Milestone 1: Data Collection, Exploration, and Preprocessing

1. Data Collection:

Acquire a dataset containing historical sales data (e.g., daily or weekly sales data from retail

or e-commerce platforms).

Ensure that the dataset includes relevant features like sales amount, date, promotions,

holidays, weather, etc.

## 1. Data Collection:

Acquire a dataset containing historical sales data (e.g., daily or weekly sales data from retail
or e-commerce platforms).
Ensure that the dataset includes relevant features like sales amount, date, promotions, holidays, weather, etc.

```
df=pd.read_csv(r"final_data.csv")
df
```

| | order_id | order_item_id | product_id | seller_id | shipping_limit_date | price | freigh |
|---|---|---|---|---|---|---|---|
| 0 | 00010242fe8c5a6d1ba2dd792cb16214 | 1 | 4244733e06e7ecb4970a6e2683c13e61 | 48436dade18ac8b2bce089ec2a041202 | 2017-09-19 09:45:35 | 58.90 | |
| 1 | 00018f77f2f0320c557190d7a144bdd3 | 1 | e5f2d52b802189ee658865ca93d83a8f | dd7ddc04e1b6c2c614352b383efe2d36 | 2017-05-03 11:05:13 | 239.90 | |
| 2 | 000229ec398224ef6ca0657da4fc703e | 1 | c777355d18b72b67abbeef9df44fd0fd | 5b51032eddd242adc84c38acab88f23d | 2018-01-18 14:48:30 | 199.00 | |
| 3 | 00024acbcdf0a6daa1e931b038114c75 | 1 | 7634da152a4610f1595efa32f14722fc | 9d7a1d34a5052409006425275ba1c2b4 | 2018-08-15 10:10:18 | 12.99 | |
| 4 | 00042b26cf59d7ce69dfabb4e55b4fd9 | 1 | ac6c3623068f30de03045865e4e10089 | df560393f3a51e74553ab94004ba5c87 | 2017-02-13 13:57:51 | 199.90 | |
| ... | ... | ... | ... | ... | ... | ... | |
| 116576 | fffc94f6ce00a00581880bf54a75a037 | 1 | 4aa6014eceb682077f9dc4bffebc05b0 | b8bc237ba3788b23da09c0f1f3a3288c | 2018-05-02 04:11:01 | 299.99 | |
| 116577 | fffcd46ef2263f404302a634eb57f7eb | 1 | 32e07fd915822b0765e448c4dd74c828 | f3c38ab652836d21de61fb8314b69182 | 2018-07-20 04:31:48 | 350.00 | |
| 116578 | fffce4705a9662cd70adb13d4a31832d | 1 | 72a30483855e2eafc67aee5dc2560482 | c3cfdc648177fdbbbb35635a37472c53 | 2017-10-30 17:14:25 | 99.90 | |
| 116579 | fffe18544ffabc95dfada21779c9644f | 1 | 9c422a519119dcad7575db5af1ba540e | 2b3e4a2a3ea8e01938cabda2a3e5cc79 | 2017-08-21 00:04:32 | 55.99 | |
| 116580 | fffe41c64501cc87c801fd61db3f6244 | 1 | 350688d9dc1e75ff97be326363655e01 | f7ccf836d21b2fb1de37564105216cc1 | 2018-06-12 17:10:13 | 43.00 | |

```
pd.set_option("display.max_columns",None)
```

# Here is what the data looks like.

```
[68]: df.shape

[68]: (116581, 40)
```

# What is Data Types?

```
[73]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116581 entries, 0 to 116580
Data columns (total 40 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   order_id                       116581 non-null  object
 1   order_item_id                  116581 non-null  int64
 2   product_id                     116581 non-null  object
 3   seller_id                      116581 non-null  object
 4   shipping_limit_date            116581 non-null  object
 5   price                          116581 non-null  float64
 6   freight_value                  116581 non-null  float64
 7   payment_sequential             116581 non-null  int64
 8   payment_type                   116581 non-null  object
 9   payment_installments           116581 non-null  int64
 10  payment_value                  116581 non-null  float64
 11  review_id                      116581 non-null  object
 12  review_score                   116581 non-null  int64
 13  review_comment_title           13996 non-null   object
 14  review_comment_message         49878 non-null   object
 15  review_creation_date           116581 non-null  object
 16  review_answer_timestamp        116581 non-null  object
 17  customer_id                    116581 non-null  object
 18  order_status                   116581 non-null  object
 19  order_purchase_timestamp       116581 non-null  object
 20  order_approved_at              116567 non-null  object
 21  order_delivered_carrier_date   115368 non-null  object
 22  order_delivered_customer_date  114066 non-null  object
 23  order_estimated_delivery_date  116581 non-null  object
 24  customer_unique_id             116581 non-null  object
 25  customer_zip_code_prefix       116581 non-null  int64
 26  customer_city                  116581 non-null  object
 27  customer_state                 116581 non-null  object
 28  product_category_name          116581 non-null  object
 29  product_name_lenght            116581 non-null  float64
 30  product_description_lenght     116581 non-null  float64
 31  product_photos_qty             116581 non-null  float64
 32  product_weight_g               116580 non-null  float64
 33  product_length_cm              116580 non-null  float64
 34  product_height_cm              116580 non-null  float64
 35  product_width_cm               116580 non-null  float64
 36  seller_zip_code_prefix         116581 non-null  int64
 37  seller_city                    116581 non-null  object
 38  seller_state                   116581 non-null  object
 39  product_category_name_english  116581 non-null  object
dtypes: float64(10), int64(6), object(24)
memory usage: 35.6+ MB
```

# NaN Values & Duplicated

```
71]: df.isnull().sum()

71]: order_id                            0
     order_item_id                       0
     product_id                          0
     seller_id                           0
     shipping_limit_date                 0
     price                               0
     freight_value                       0
     payment_sequential                  0
     payment_type                        0
     payment_installments                0
     payment_value                       0
     review_id                           0
     review_score                        0
     review_comment_title           102585
     review_comment_message          66703
     review_creation_date                0
     review_answer_timestamp             0
     customer_id                         0
     order_status                        0
     order_purchase_timestamp            0
     order_approved_at                  14
     order_delivered_carrier_date     1213
     order_delivered_customer_date    2515
     order_estimated_delivery_date       0
     customer_unique_id                  0
     customer_zip_code_prefix            0
     customer_city                       0
     customer_state                      0
     product_category_name               0
     product_name_lenght                 0
     product_description_lenght          0
     product_photos_qty                  0
     product_weight_g                    1
     product_length_cm                   1
     product_height_cm                   1
     product_width_cm                    1
     seller_zip_code_prefix              0
     seller_city                         0
     seller_state                        0
     product_category_name_english       0
     dtype: int64
```

```
]: df[df.duplicated()]
   # No Duplicated

]:
```

| order_id | order_item_id | product_id | seller_id | shipping_limit_date | price | freight_value | payment_sequential | payment_type | payment_installments | payment_va |
|---|---|---|---|---|---|---|---|---|---|---|

# Data Preprocessing:

- Data Preprocessing:
- 1-Handle missing values, remove duplicates, and address any data inconsistencies.
- 2-Engineer time-based features (e.g., day of the week, month, seasonality, promotional
- periods).
- 1-Handle missing values, remove duplicates, and address any data inconsistencies

# Handle the Dates

```
df['date of shipping_limit'] = pd.to_datetime(df['shipping_limit_date']).dt.date
```

```
df['day shipping_limit']=pd.to_datetime(df['date of shipping_limit']).dt.day
df['month shipping_limit']=pd.to_datetime(df['date of shipping_limit']).dt.month
df['year shipping_limit']=pd.to_datetime(df['date of shipping_limit']).dt.year
```

```
df['day of review creation']=pd.to_datetime(df['review_creation_date']).dt.day
df['month of review creation']=pd.to_datetime(df['review_creation_date']).dt.month
df['year of review creation']=pd.to_datetime(df['review_creation_date']).dt.year
df['date of review creation']=pd.to_datetime(df['review_creation_date']).dt.date


df['day of answer for the review creation']=pd.to_datetime(df['review_answer_timestamp']).dt.day
df['month of answer for the review creation']=pd.to_datetime(df['review_answer_timestamp']).dt.month
df['year of answer for the review creation']=pd.to_datetime(df['review_answer_timestamp']).dt.year
df['date of answer for the review creation']=pd.to_datetime(df['review_answer_timestamp']).dt.date



df['day of order buy']=pd.to_datetime(df['order_purchase_timestamp']).dt.day
df['month of order buy']=pd.to_datetime(df['order_purchase_timestamp']).dt.month
df['year of order buy']=pd.to_datetime(df['order_purchase_timestamp']).dt.year
df['date of order buy']=pd.to_datetime(df['order_purchase_timestamp']).dt.date


df['day of approve for the order']=pd.to_datetime(df['order_approved_at']).dt.day
df['month of approve for the order']=pd.to_datetime(df['order_approved_at']).dt.month
df['year of approve for the order']=pd.to_datetime(df['order_approved_at']).dt.year
df['date of approve for the order']=pd.to_datetime(df['order_approved_at']).dt.date


df['day of deliver for the Shipping company']=pd.to_datetime(df['order_delivered_carrier_date']).dt.day
df['month of deliver for the Shipping company']=pd.to_datetime(df['order_delivered_carrier_date']).dt.month
df['year of deliver for the Shipping company']=pd.to_datetime(df['order_delivered_carrier_date']).dt.year
df['date of deliver for the Shipping company']=pd.to_datetime(df['order_delivered_carrier_date']).dt.date


df['day of deliver for customer']=pd.to_datetime(df['order_delivered_customer_date']).dt.day
df['month of deliver for customer']=pd.to_datetime(df['order_delivered_customer_date']).dt.month
df['year of deliver for customer']=pd.to_datetime(df['order_delivered_customer_date']).dt.year
df['date of deliver for customer']=pd.to_datetime(df['order_delivered_customer_date']).dt.date


df['day of expected to deliver for the customer']=pd.to_datetime(df['order_estimated_delivery_date']).dt.day
df['month of expected to deliver for the customer']=pd.to_datetime(df['order_estimated_delivery_date']).dt.month
df['year of expected to deliver for the customer']=pd.to_datetime(df['order_estimated_delivery_date']).dt.year
df['date of expected to deliver for the customer']=pd.to_datetime(df['order_estimated_delivery_date']).dt.date
```

# Handle the data inconsistencies

1

```
df.loc[
    df['order_delivered_customer_date'] < df['order_purchase_timestamp'],
    'order_delivered_customer_date'
] = pd.NaT
```

```
df[df['order_delivered_customer_date'] < df['order_purchase_timestamp']]
```

| order_id | order_item_id | product_id | seller_id | shipping_limit_date | price | freight_value | payment_sequential | payment_type | payment_installments | payment_value | re |
|---|---|---|---|---|---|---|---|---|---|---|---|

# Handle the data inconsistencies 2

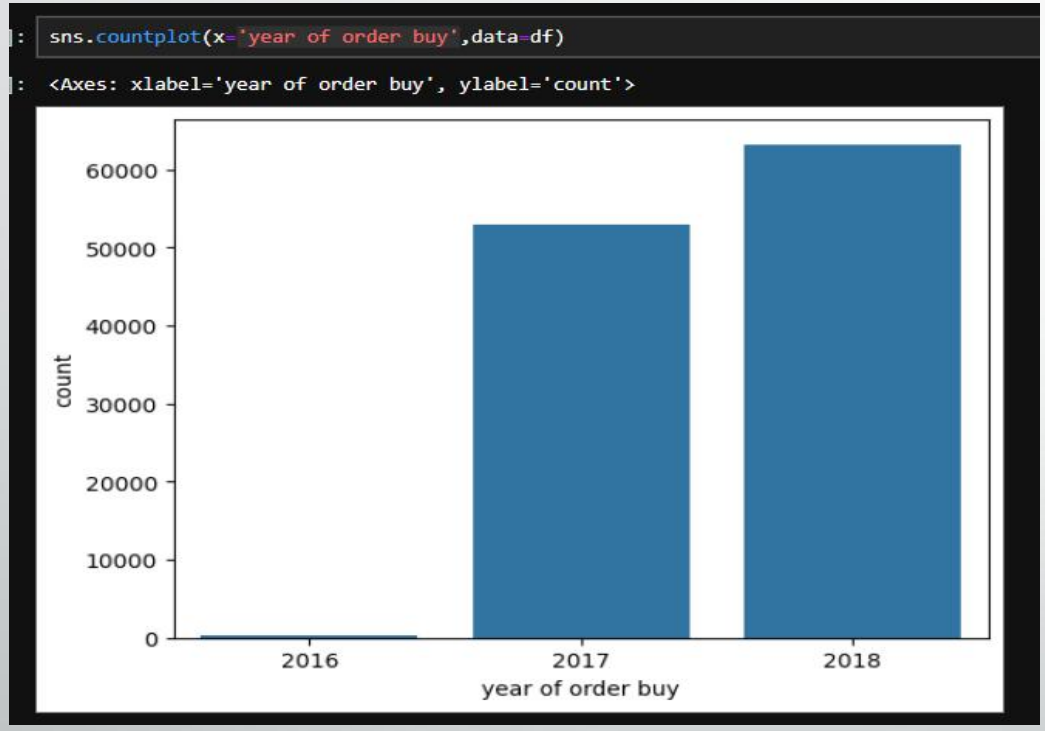
```
17]: columns_to_lower = ['product_category_name', 'payment_type', 'customer_city',
                         'customer_state', 'seller_city', 'seller_state', 'product_category_name_english']

     df[columns_to_lower] = df[columns_to_lower].apply(lambda x: x.str.lower())

18]: df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

# Milestone 2: Data Analysis and Visualization

- 1-Data Analysis:
- Perform statistical analysis to identify correlations between sales and other factors such as
- promotions, weather, holidays, and special events.
- Investigate seasonality and long-term trends in sales.
- 2. Data Visualization:
- Create visualizations like line graphs, bar charts, and scatter plots to display sales trends and
- seasonal patterns.
- Develop interactive dashboards (using Plotly or Dash) to allow users to explore trends and
- patterns in the sales data.

# Sales Over the Years

```
]: sns.countplot(x='year of order buy',data=df)

]: <Axes: xlabel='year of order buy', ylabel='count'>
```




| | |
|---|---|
| 2018 | 63199 |
| 2017 | 52994 |
| 2016 | 388 |

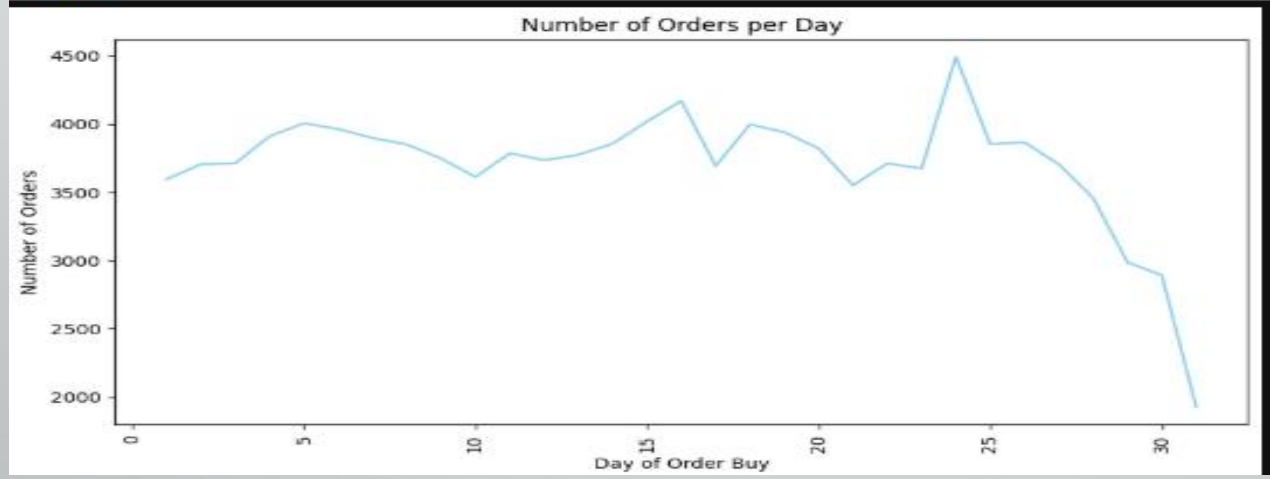
# Sales Over the Months

```
monthly_orders = df['month of order buy'].value_counts().sort_index()
plt.figure(figsize=(12, 6))
plt.plot(monthly_orders.index, monthly_orders.values, marker='o')
plt.xticks(rotation=0)
plt.title("Number Of Orders per Month")
plt.xlabel("Month")
plt.ylabel("Number of Orders")
plt.grid(True)
plt.show()
```


| | |
|---|---|
| 8 | 12569 |
| 5 | 12517 |
| 7 | 12108 |
| 3 | 11587 |
| 6 | 11128 |
| 4 | 10977 |
| 2 | 9912 |
| 1 | 9392 |
| 11 | 8963 |
| 12 | 6443 |
| 10 | 5926 |
| 9 | 5059 |

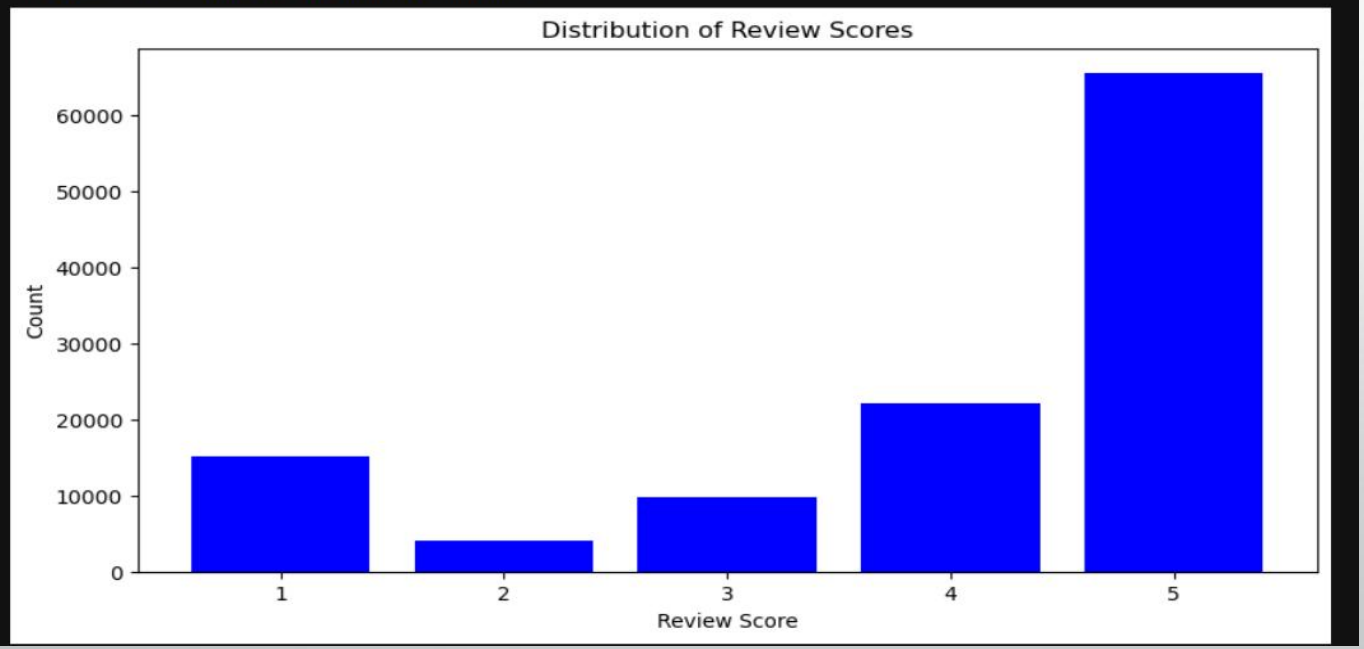
# Sales Over the Days

```
x = df['day of order buy'].value_counts().sort_index()
plt.figure(figsize=(10, 5))
plt.plot(x.index, x.values, color='skyblue')
plt.xlabel("Day of Order Buy")
plt.ylabel("Number of Orders")
plt.title("Number of Orders per Day")
plt.xticks(rotation=90)
plt.show()
```



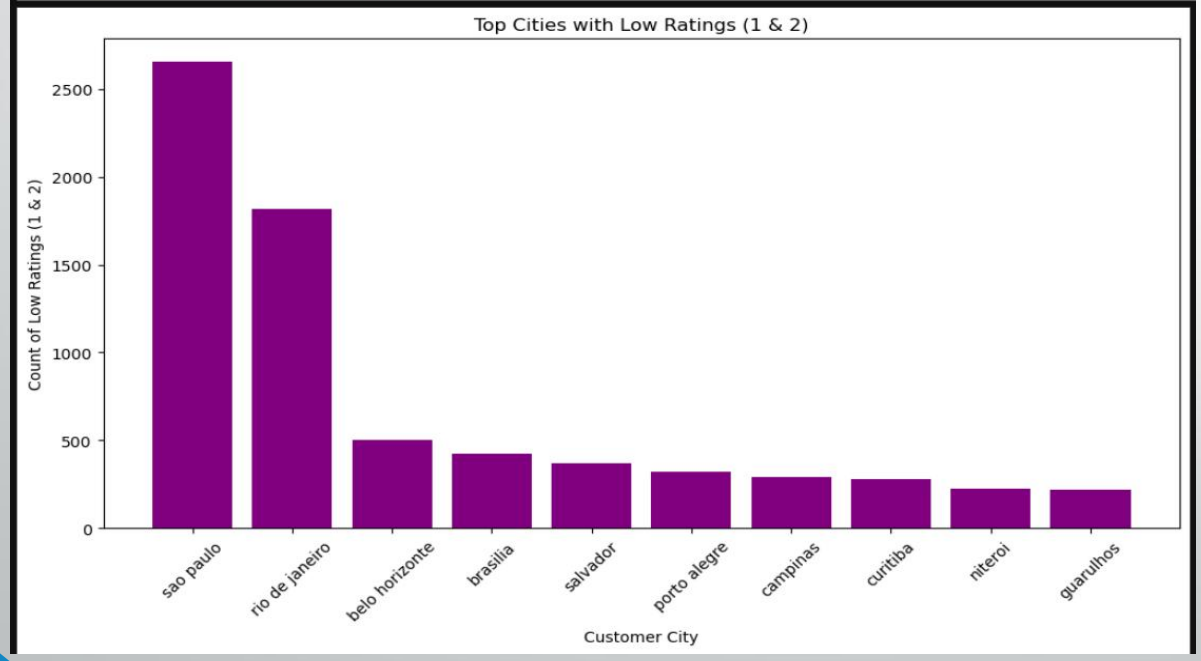
# Review Score

```
]: c = df['review_score'].value_counts().sort_index()
   plt.figure(figsize=(10, 5))
   plt.bar(c.index, c.values, color='blue')
   plt.xlabel("Review Score")
   plt.ylabel("Count")
   plt.title("Distribution of Review Scores")

]: Text(0.5, 1.0, 'Distribution of Review Scores')
```



# Cities with Low Ratings

```
plt.figure(figsize=(12, 6))
plt.bar(grouped_data['customer_city'][:10], grouped_data['count'][:10], color='purple')
plt.xlabel('Customer City')
plt.ylabel('Count of Low Ratings (1 & 2)')
plt.xticks(rotation=45)
plt.title('Top Cities with Low Ratings (1 & 2)')
plt.show()
lowrateC=grouped_data[['customer_city','count']]
```

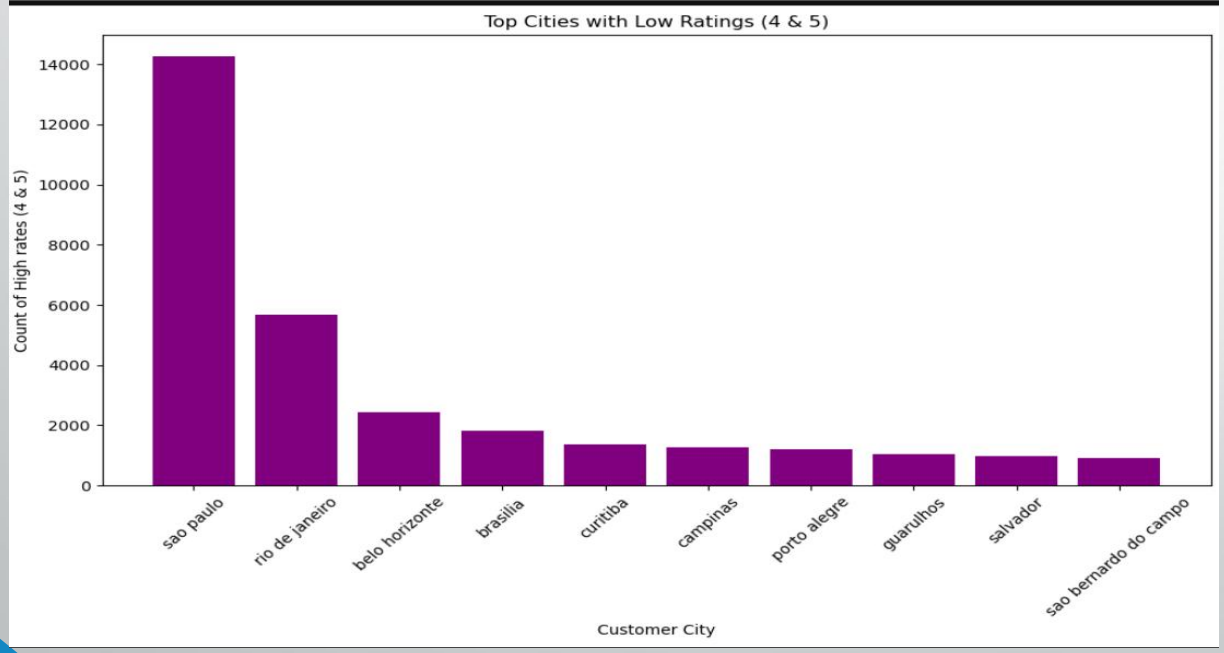



```
x = [1, 2]
filtered_df = df[df['review_score'].isin(x)]
grouped_data = filtered_df.groupby('customer_city').size().reset_index(name='count')
grouped_data = grouped_data.sort_values(by='count', ascending=False)
grouped_data.head(10)
```

| | customer_city | count |
|---|---|---|
| 1541 | sao paulo | 2652 |
| 1355 | rio de janeiro | 1818 |
| 193 | belo horizonte | 504 |
| 236 | brasilia | 424 |
| 1393 | salvador | 370 |
| 1269 | porto alegre | 319 |
| 305 | campinas | 294 |
| 511 | curitiba | 277 |
| 1062 | niteroi | 224 |
| 660 | guarulhos | 218 |

# Cities with High Ratings

```
plt.figure(figsize=(12, 6))
plt.bar(grouped_data['customer_city'][:10], grouped_data['count'][:10], color='purple')
plt.xlabel('Customer City')
plt.ylabel('Count of High rates (4 & 5)')
plt.xticks(rotation=45)
plt.title('Top Cities with Low Ratings (4 & 5)')
plt.show()
```



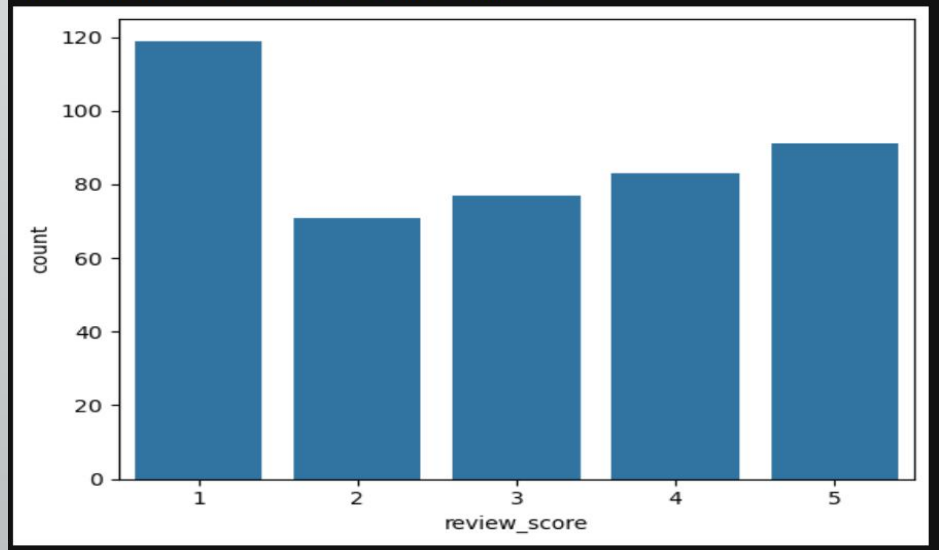

```
y= [4, 5]
filtered_df = df[df['review_score'].isin(y)]
grouped_data = filtered_df.groupby('customer_city').size().reset_index(name='count')
grouped_data = grouped_data.sort_values(by='count', ascending=False)
grouped_data.head(10)
```

| | customer_city | count |
|---|---|---|
| 3335 | sao paulo | 14256 |
| 2922 | rio de janeiro | 5688 |
| 419 | belo horizonte | 2421 |
| 515 | brasilia | 1829 |
| 1062 | curitiba | 1379 |
| 655 | campinas | 1273 |
| 2745 | porto alegre | 1198 |
| 1418 | guarulhos | 1049 |
| 3010 | salvador | 979 |
| 3180 | sao bernardo do campo | 897 |

# This graph shows how an increase in the number of days between order and delivery affects the review rating

```
sns.countplot(x=dfnew['review_score'])
# Rate by2l lma yt25r
```

```
<Axes: xlabel='review_score', ylabel='count'>
```




```
df[['review_score','Difference between request order and arrive']].corr()
```

| | review_score | Difference between request order and arrive |
|---|---|---|
| review_score | 1.000000 | -0.304133 |
| Difference between request order and arrive | -0.304133 | 1.000000 |

# Outliers

Outliers were detected in both payment value and weight, and they were dropped to ensure data quality and accuracy in analysis.

```
[ ]: df.query("product_weight_g >40000")
```

| | order_id | order_item_id | product_id | seller_id | shipping_limit_date | price | freight_valu |
|---|---|---|---|---|---|---|---|
| 34006 | 4a45f9f66971302cf881ecfa142f42ba | 1 | 26644690fde745fc4654719c3904e1db | 1835b56ce799e6a4dc4eddc053f04066 | 2017-12-29 19:13:20 | 80.0 | 16.3 |
| 50643 | 6ecf1a4051b4c5ed613624b460970a26 | 1 | 26644690fde745fc4654719c3904e1db | 1835b56ce799e6a4dc4eddc053f04066 | 2017-11-23 08:55:48 | 80.0 | 12.0 |
| 66372 | 9223919b300f6989e1715333fca0d6ce | 1 | 26644690fde745fc4654719c3904e1db | 1835b56ce799e6a4dc4eddc053f04066 | 2018-03-07 23:15:32 | 80.0 | 16.3 |

```
[ ]: df.drop([34006,50643,66372],axis=0,inplace=True)
```

```
[ ]: df = df[df['payment_value'] <= 5000]
```

# Correlation Analysis Between Freight, Volume, and Weight

```
71]: df[['volume', 'freight_value']].corr()
```

71]:

| | volume | freight_value |
|---|---|---|
| **volume** | 1.000000 | 0.583976 |
| **freight_value** | 0.583976 | 1.000000 |

```
74]: df[['product_weight_g', 'freight_value']].corr()
```

74]:

| | product_weight_g | freight_value |
|---|---|---|
| **product_weight_g** | 1.000000 | 0.615762 |
| **freight_value** | 0.615762 | 1.000000 |

# There is a weak correlation between price and both weight and volume

```
[880]: df[['product_weight_g', 'price']].corr()
```

[880]:

| | product_weight_g | price |
|---|---|---|
| **product_weight_g** | 1.000000 | 0.343402 |
| **price** | 0.343402 | 1.000000 |

```
[879]: df[['volume', 'price']].corr()
```

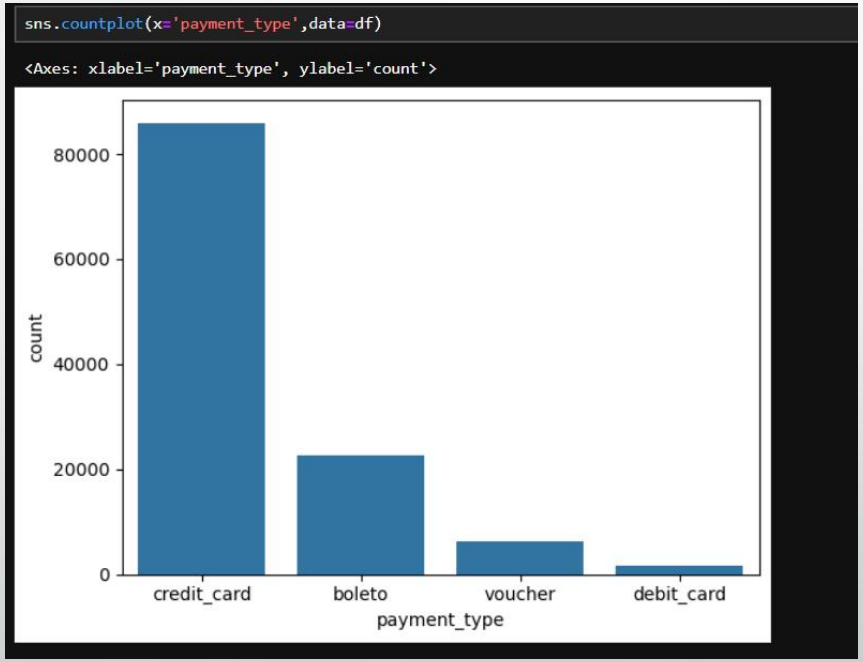
[879]:

| | volume | price |
|---|---|---|
| **volume** | 1.000000 | 0.299733 |
| **price** | 0.299733 | 1.000000 |

# Payment Type

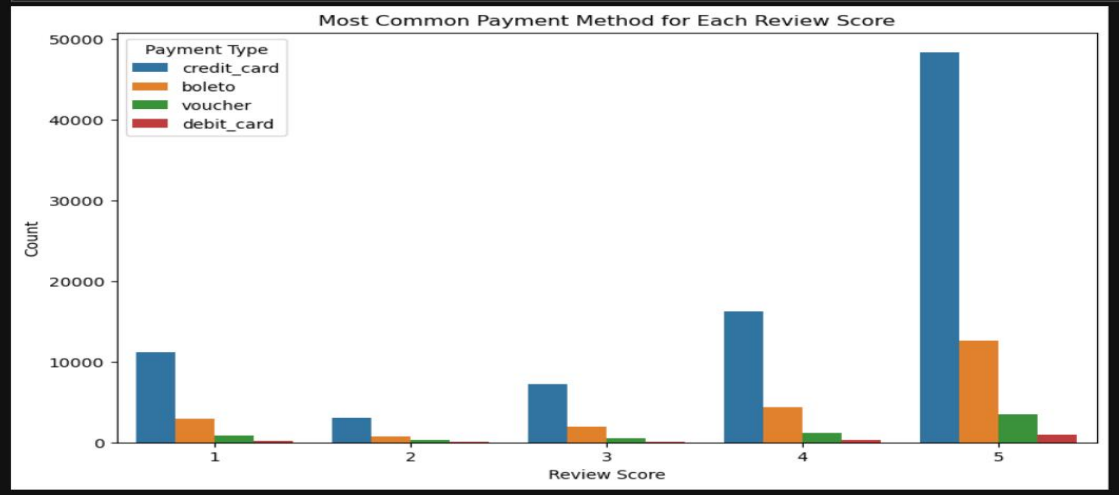
```
sns.countplot(x='payment_type',data=df)
```

```
<Axes: xlabel='payment_type', ylabel='count'>
```



**The graph illustrates the relationship between payment methods and product ratings.**

```
plt.figure(figsize=(10, 6))
sns.countplot(x="review_score", hue="payment_type", data=df)
plt.title("Most Common Payment Method for Each Review Score")
plt.xlabel("Review Score")
plt.ylabel("Count")
plt.legend(title="Payment Type")
plt.show()
```




# The best-selling products.

```
df['product_category_name'].value_counts().head()
```

```
product_category_name
cama_mesa_banho           11987
beleza_saude              10025
esporte_lazer              9005
moveis_decoracao           8833
informatica_acessorios     8151
Name: count, dtype: int64
```

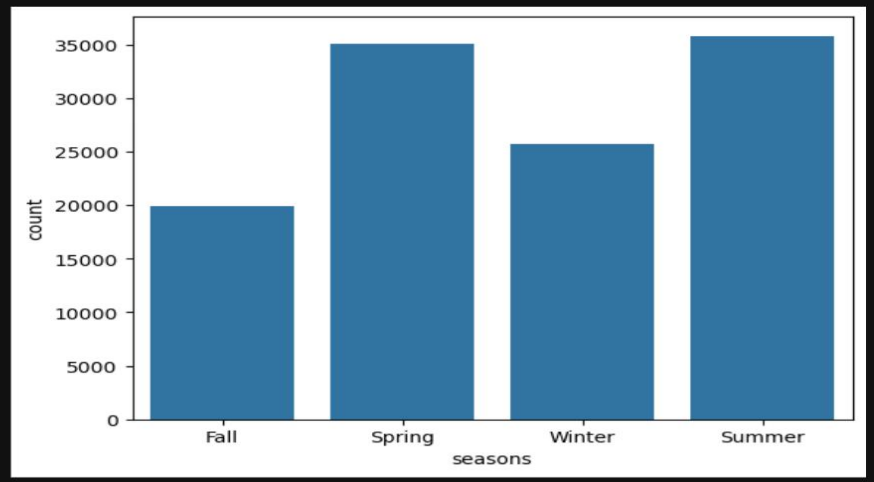
# The least-selling products.

```
df['product_category_name_english'].value_counts().tail(10)
```

```
product_category_name_english
music                          40
diapers_and_hygiene            39
flowers                        33
home_comfort_2                 31
fashion_sport                  31
arts_and_craftmanship          24
la_cuisine                     16
cds_dvds_musicals              14
fashion_childrens_clothes       8
security_and_services           2
Name: count, dtype: int64
```

# Sales across seasons.

```
[119]: sns.countplot(x='seasons',data=df)

[119]: <Axes: xlabel='seasons', ylabel='count'>
```




```
[120]: df['seasons'].value_counts()

[120]: seasons
       Summer    35794
       Spring    35073
       Winter    25745
       Fall      19932
```

```
condition=[(df['month of order buy'].isin([12,1,2])),(df['month of order buy'].isin([3,4,5])),(df['month of order buy'].isin([6,7,8])),(df['month of
choice=['Winter','Spring','Summer','Fall']
df['seasons']=np.select(condition,choice,default='Nothing')
df
```

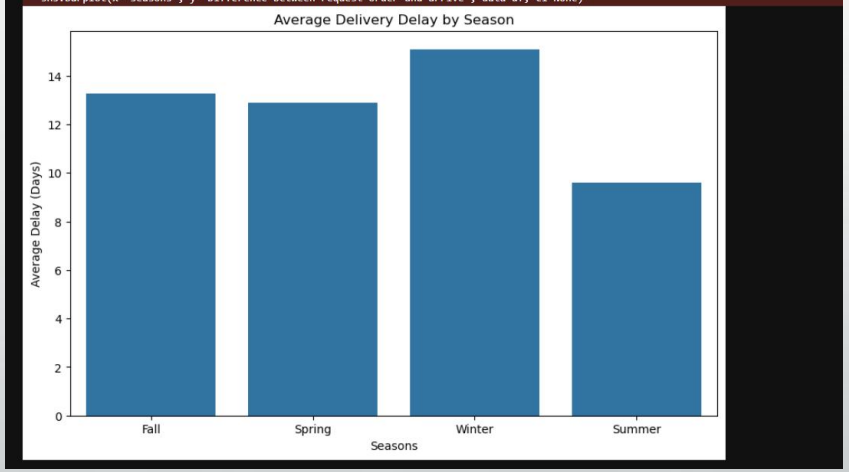
```
seasons
Summer    35794
Spring    35073
Winter    25745
Fall      19932
```

Winter: 12, 1, 2
Spring: 3, 4, 5
Summer: 6, 7, 8
Fall: 9, 10, 11

# Order delivery delays across seasons.

```
]: plt.figure(figsize=(10,6))
   sns.barplot(x='seasons', y='Difference between request order and arrive', data=df, ci=None)
   plt.title('Average Delivery Delay by Season')
   plt.xlabel('Seasons')
   plt.ylabel('Average Delay (Days)')
   plt.show()
```

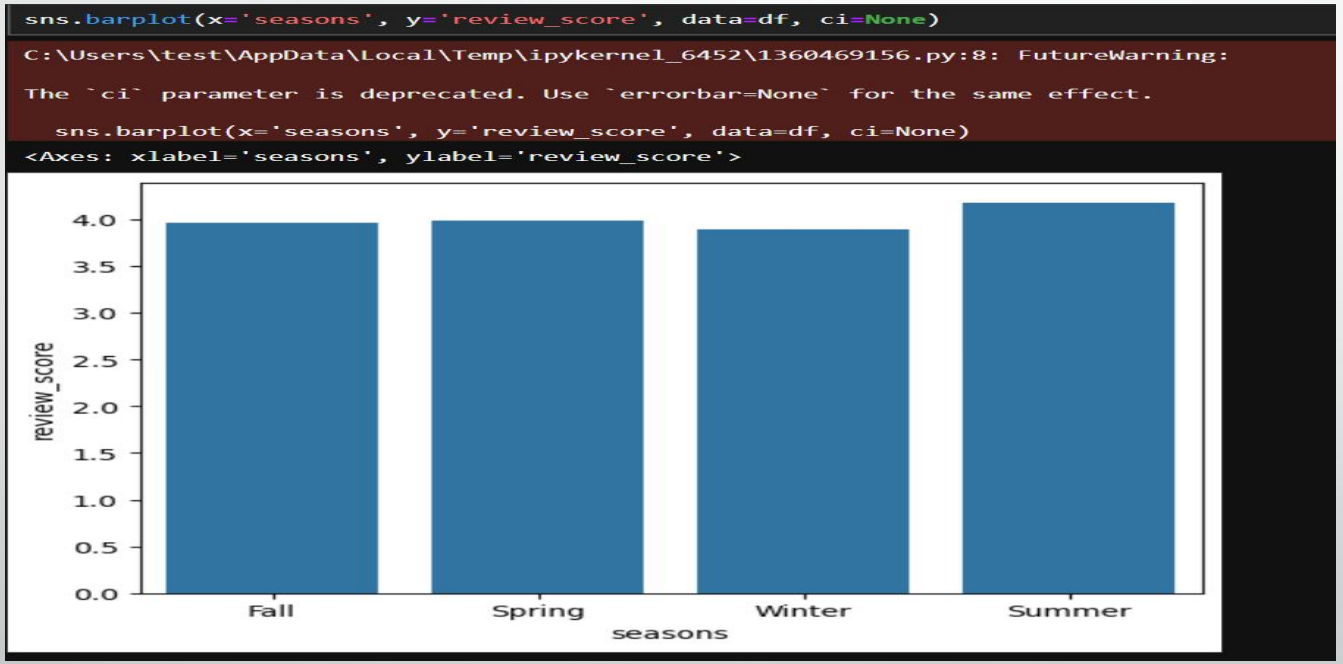
```
C:\Users\test\AppData\Local\Temp\ipykernel_6452\2273605181.py:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='seasons', y='Difference between request order and arrive', data=df, ci=None)
```



# Review ratings across seasons.

```
sns.barplot(x='seasons', y='review_score', data=df, ci=None)
```

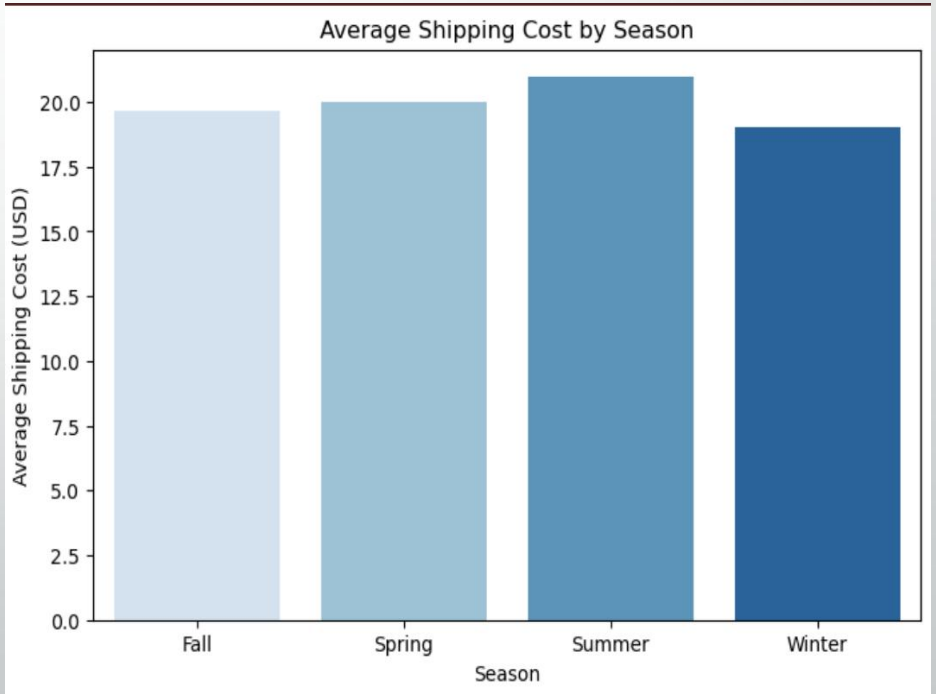
```
C:\Users\test\AppData\Local\Temp\ipykernel_6452\1360469156.py:8: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='seasons', y='review_score', data=df, ci=None)
<Axes: xlabel='seasons', ylabel='review_score'>
```



# Delivery price across seasons.

```
seasonal_shipping_cost = df.groupby("seasons")["freight_value"].mean()
print(seasonal_shipping_cost)
```

```
seasons
Fall      19.673781
Spring    19.975340
Summer    20.968218
Winter    19.037837
Name: freight_value, dtype: float64
```



# Best-selling and least-selling products across seasons.

```
items0 = df.groupby("seasons")['product_category_name_english'].agg(lambda x: x.value_counts().idxmax())
items0
```

```
seasons
Fall      bed_bath_table
Spring    bed_bath_table
Summer    bed_bath_table
Winter    bed_bath_table
Name: product_category_name_english, dtype: object
```

```
items1 = df.groupby("seasons")['product_category_name_english'].agg(lambda x: x.value_counts().idxmin())
items1
```

```
seasons
Fall          security_and_services
Spring            cds_dvds_musicals
Summer    fashion_childrens_clothes
Winter            cds_dvds_musicals
Name: product_category_name_english, dtype: object
```

# Top 10 cities with the most delivery delays along with their review scores.

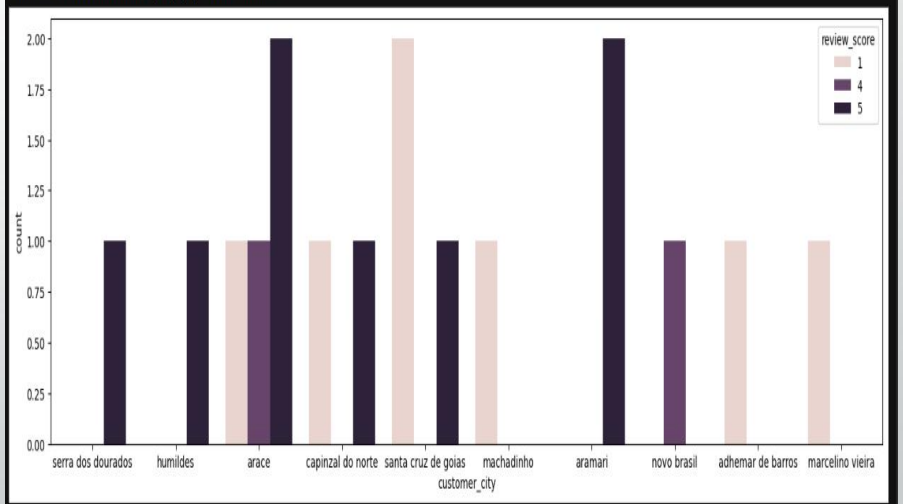
```
result_df = pd.DataFrame(result, columns=['Difference between request order and arrive']).reset_index()
```

```
result_df
```

| | customer_city | Difference between request order and arrive |
|---|---|---|
| 0 | novo brasil | 148.000000 |
| 1 | capinzal do norte | 109.000000 |
| 2 | adhemar de barros | 98.000000 |
| 3 | santa cruz de goias | 86.666667 |
| 4 | arace | 86.500000 |
| 5 | marcelino vieira | 78.000000 |
| 6 | serra dos dourados | 75.000000 |
| 7 | aramari | 74.000000 |
| 8 | humildes | 72.000000 |
| 9 | machadinho | 65.000000 |

```
plt.figure(figsize=(18,5))
sns.countplot(x='customer_city',hue='review_score',data=filtered_df)
```

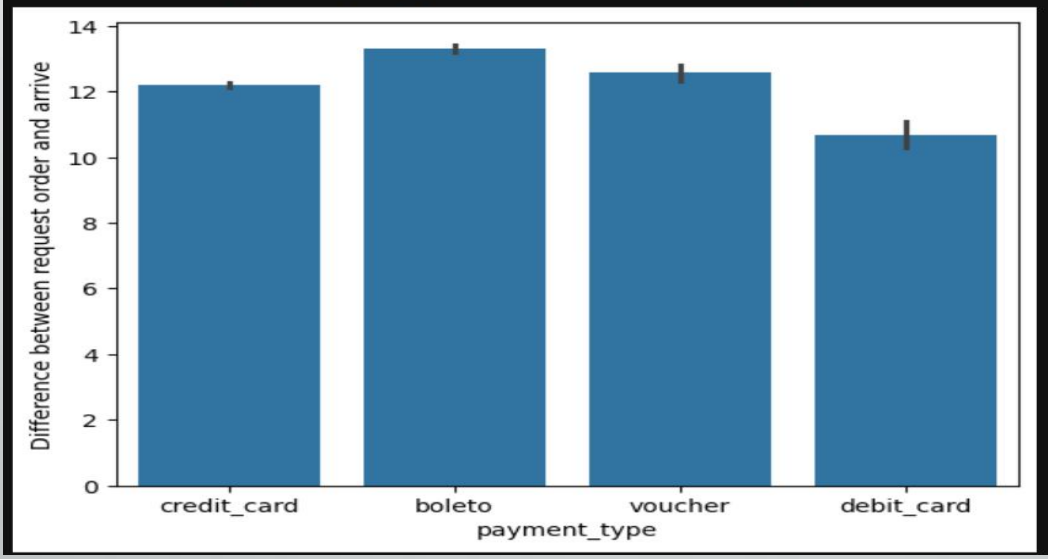
```
<Axes: xlabel='customer_city', ylabel='count'>
```



# The graph shows which payment method experiences the most delivery delays.

```
sns.barplot(x='payment_type', y='Difference between request order and arrive', data=df, estimator=np.mean)
```

```
<Axes: xlabel='payment_type', ylabel='Difference between request order and arrive'>
```

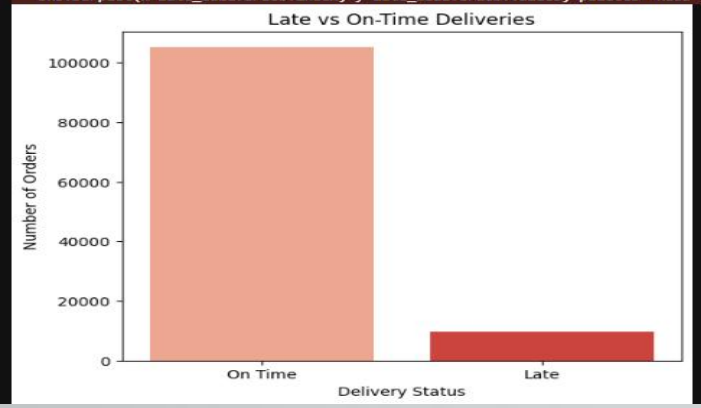



```
payment_type
credit_card
boleto
voucher
debit_card
```

# Comparison Between Late and On-Time Deliveries

```
df['late_delivery'] = df['order_delivered_customer_date'] > df['order_estimated_delivery_date']
late_deliveries = df['late_delivery'].value_counts()
plt.figure(figsize=(6, 5))
sns.barplot(x=late_deliveries.index, y=late_deliveries.values, palette="Reds")
plt.xticks(ticks=[0, 1], labels=["On Time", "Late"])
plt.xlabel("Delivery Status")
plt.ylabel("Number of Orders")
plt.title("Late vs On-Time Deliveries")
plt.show()
```

```
C:\Users\test\AppData\Local\Temp\ipykernel_14380\2472588398.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.barplot(x=late_deliveries.index, y=late_deliveries.values, palette="Reds")
```



- Milestone 3: Forecasting Model Development and Optimization
- Extract features and target variable from the DataFrame

# Build The Model
# 1-Choose The Features

Handle the outliers

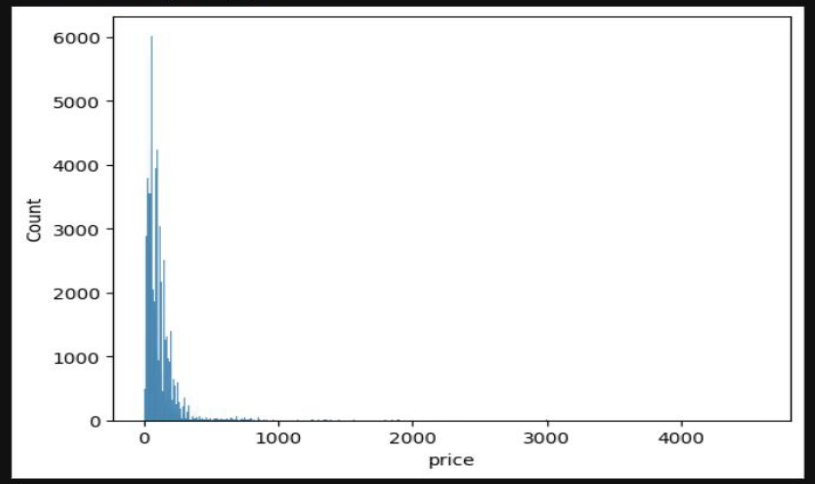
```
df=df.query("product_weight_g < 15000")
df=df.query("freight_value < 100")
```

```
numerical_cols = [
    "payment_sequential",
    "payment_installments",
    "product_name_lenght",
    "product_description_lenght",
    "product_photos_qty",
    "product_weight_g",
    "product_length_cm",
    "product_height_cm",
    "product_width_cm",
    "freight_value"
]

categorical_cols = [
    "order_status",
    "customer_city",
    "product_category_name_english",
    "seller_city",
]

x_num_df = df[numerical_cols].copy()

valid_rows = (x_num_df >= 0).all(axis=1)

df_valid = df[valid_rows].copy()
x_num_df_valid = x_num_df[valid_rows].copy()
x_num_df_log = np.log1p(x_num_df_valid)

x_catboost = pd.concat([x_num_df_log, df_valid[categorical_cols]], axis=1)


y=df['price']
```

# 2-Apply Log Transformation to avoid the skewness

```
[116]: sns.histplot(df['price'])
```

```
[116]: <Axes: xlabel='price', ylabel='Count'>
```




Apply Log Transformation to avoid the skewness

```
[117]: y_log = np.log(df['price']).values
```

# 3-Split thd data

```
from sklearn.model_selection import train_test_split
xtrain_cat, xtest_cat, ytrain_cat, ytest_cat = train_test_split(x_catboost, y_log, random_state=0, test_size=0.2)
cat_features_indices = [x_catboost.columns.get_loc(col) for col in categorical_cols]
```

```
df.isnull().sum()
```

```
order_id                                    0
order_item_id                               0
product_id                                  0
seller_id                                   0
shipping_limit_date                         0
                                           ..
Difference between request order and arrive 0
volume                                      0
days of answer for review                   0
seasons                                     0
late_delivery                               0
Length: 75, dtype: int64
```

# 4-Make RobustScaler

```
22]: from sklearn.preprocessing import RobustScaler
     xtrain_num = xtrain_cat[numerical_cols]
     xtest_num = xtest_cat[numerical_cols]
     xtrain_cat_part = xtrain_cat[categorical_cols]
     xtest_cat_part = xtest_cat[categorical_cols]

     scaler = RobustScaler()
     xtrain_num_scaled = scaler.fit_transform(xtrain_num)
     xtest_num_scaled = scaler.transform(xtest_num)

     xtrain_num_scaled = pd.DataFrame(xtrain_num_scaled, columns=numerical_cols, index=xtrain_num.index)
     xtest_num_scaled = pd.DataFrame(xtest_num_scaled, columns=numerical_cols, index=xtest_num.index)

     xtrain_final = pd.concat([xtrain_num_scaled, xtrain_cat_part], axis=1)
     xtest_final = pd.concat([xtest_num_scaled, xtest_cat_part], axis=1)
```

# 5-Use the Model “CATBOOSTING”

```python
from catboost import CatBoostRegressor
from sklearn.metrics import r2_score


cat_model = CatBoostRegressor(verbose=0)
cat_model.fit(xtrain_final, ytrain_cat, cat_features=cat_features_indices)
y_pred_cat = cat_model.predict(xtest_final)
r2_cat = r2_score(ytest_cat, y_pred_cat)
r2_cat = round(r2_cat, 2) * 100
print("CatBoost Test R² Score:", r2_cat, "%")

acc_cat = round(r2_score(ytrain_cat, cat_model.predict(xtrain_final)), 2) * 100
print("CatBoost Train R² Score:", acc_cat, "%")
```

```
Loading widget...
 CatBoost Test R² Score: 78.0 %
 CatBoost Train R² Score: 82.0 %
```

- ***Deployment***

# Deploy the model using Streamlit

```
# using streamlit

import pickle
pickle.dump(cat_model,open(r"C:\Users\test\DEPI Project\Project DEPI.pkl", "wb"))
```

# The interface