

## ✓ Day 1: Setup, Loading, Exploration

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

```
plt.style.use("seaborn-v0_8")
pd.set_option("display.max_columns", None)
```

### #Load data

```
sales_df = pd.read_csv("sales_data.csv")
churn_df = pd.read_csv("customer_churn.csv")
```

### #Quick exploration

```
print("Sales head:")
print(sales_df.head())
print("\nSales info:")
print(sales_df.info())
```

Sales head:

	Date	Product	Quantity	Price	Customer_ID	Region	Total_S
0	2024-01-01	Phone	7	37300	CUST001	East	26
1	2024-01-02	Headphones	4	15406	CUST002	North	6
2	2024-01-03	Phone	2	21746	CUST003	West	4
3	2024-01-04	Headphones	1	30895	CUST004	East	3
4	2024-01-05	Laptop	8	39835	CUST005	North	31

Sales info:

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 100 entries, 0 to 99
```

```
Data columns (total 7 columns):
```

#	Column	Non-Null Count	Dtype
0	Date	100 non-null	object
1	Product	100 non-null	object
2	Quantity	100 non-null	int64
3	Price	100 non-null	int64
4	Customer_ID	100 non-null	object
5	Region	100 non-null	object
6	Total_Sales	100 non-null	int64

```
dtypes: int64(3), object(4)
```

```
memory usage: 5.6+ KB
```

```
None
```

```
print("\nChurn head:")
print(churn_df.head())
print("\nChurn info:")
print(churn_df.info())
```

Churn head:

	CustomerID	Tenure	MonthlyCharges	TotalCharges	Contract	\
0	C00001	6	64	1540	One year	
1	C00002	21	113	1753	Month-to-month	
2	C00003	27	31	1455	Two year	
3	C00004	53	29	7150	Month-to-month	
4	C00005	16	185	1023	One year	

	PaymentMethod	PaperlessBilling	SeniorCitizen	Churn
0	Credit Card	No	1	0
1	Electronic Check	Yes	1	0
2	Credit Card	No	1	0
3	Electronic Check	No	1	0
4	Electronic Check	No	1	0

Churn info:

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 500 entries, 0 to 499

Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype
0	CustomerID	500 non-null	object
1	Tenure	500 non-null	int64
2	MonthlyCharges	500 non-null	int64
3	TotalCharges	500 non-null	int64
4	Contract	500 non-null	object
5	PaymentMethod	500 non-null	object
6	PaperlessBilling	500 non-null	object
7	SeniorCitizen	500 non-null	int64
8	Churn	500 non-null	int64

dtypes: int64(5), object(4)

memory usage: 35.3+ KB

None

## #Check missing values

```
print("\nMissing values (sales):")
print(sales_df.isna().sum())
print("\nMissing values (churn):")
print(churn_df.isna().sum())
```

Missing values (sales):

Date	0
Product	0
Quantity	0
Price	0
Customer_ID	0
Region	0
Total_Sales	0

dtype: int64

```
Missing values (churn):
CustomerID      0
Tenure          0
MonthlyCharges  0
TotalCharges    0
Contract        0
PaymentMethod   0
PaperlessBilling 0
SeniorCitizen   0
Churn           0
dtype: int64
```

## ✓ Day 2: Cleaning & Preparation

**#Strip** and standardize text columns

```
text_cols_sales = sales_df.select_dtypes(include="object").columns
sales_df[text_cols_sales] = sales_df[text_cols_sales].apply(
    lambda col: col.str.strip().str.lower()
)

text_cols_churn = churn_df.select_dtypes(include="object").columns
churn_df[text_cols_churn] = churn_df[text_cols_churn].apply(
    lambda col: col.str.strip().str.lower()
)
```

**#Convert date column** : Change 'order\_date' to your real date column name

```
sales_df["Date"] = pd.to_datetime(sales_df["Date"], errors="coerce")
```

**#Handle** missing values

```
for col in ["quantity", "price"]:
    if col in sales_df.columns:
        sales_df[col] = sales_df[col].fillna(0)

sales_df = sales_df.dropna(subset=["Date", "Customer_ID"])
```

**#Create calculated columns** : Revenue = quantity \* price

```
sales_df["revenue"] = sales_df["Quantity"] * sales_df["Price"]

sales_df["year"] = sales_df["Date"].dt.year
sales_df["month"] = sales_df["Date"].dt.month
sales_df["day"] = sales_df["Date"].dt.day
sales_df["year_month"] = sales_df["Date"].dt.to_period("M").astype(st
```

```
print("\nSales with new columns:")
print(sales_df.head())
```

Sales with new columns:

	Date	Product	Quantity	Price	Customer_ID	Region	Total_Sa
0	2024-01-01	phone	7	37300	cust001	east	261
1	2024-01-02	headphones	4	15406	cust002	north	61
2	2024-01-03	phone	2	21746	cust003	west	43
3	2024-01-04	headphones	1	30895	cust004	east	30
4	2024-01-05	laptop	8	39835	cust005	north	318

	revenue	year	month	day	year_month
0	261100	2024	1	1	2024-01
1	61624	2024	1	2	2024-01
2	43492	2024	1	3	2024-01
3	30895	2024	1	4	2024-01
4	318680	2024	1	5	2024-01

## ✓ Day 3: Customer Analysis

**#Merge** sales with customers to get customer attributes : Make sure 'customer\_id' exists in both

```
sales_df = sales_df.rename(columns={'Customer_ID': 'customer_id'})
churn_df = churn_df.rename(columns={'CustomerID': 'customer_id'})
```

```
full_df = pd.merge(
    sales_df,
    churn_df,
    on="customer_id",
    how="left",
    suffixes=('_sale', '_cust'),
)
```

```
print("\nMerged data head:")
print(full_df.head())
```

Merged data head:

	Date	Product	Quantity	Price	customer_id	Region	Total_Sa
0	2024-01-01	phone	7	37300	cust001	east	261
1	2024-01-02	headphones	4	15406	cust002	north	61
2	2024-01-03	phone	2	21746	cust003	west	43
3	2024-01-04	headphones	1	30895	cust004	east	30
4	2024-01-05	laptop	8	39835	cust005	north	318

	revenue	year	month	day	year_month	Tenure	MonthlyCharges	Total
0	261100	2024	1	1	2024-01	NaN	NaN	
1	61624	2024	1	2	2024-01	NaN	NaN	
2	43492	2024	1	3	2024-01	NaN	NaN	
3	30895	2024	1	4	2024-01	NaN	NaN	

4	318680	2024	1	5	2024-01	NaN	NaN
	Contract	PaymentMethod	PaperlessBilling	SeniorCitizen	Churn		
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN

## #Customer-level aggregations

```
customer_agg = (
    full_df.groupby("customer_id")
    .agg(
        total_revenue=("revenue", "sum"),
        order_count=("Date", "count"),
        total_quantity=("Quantity", "sum"),
        first_purchase=("Date", "min"),
        last_purchase=("Date", "max"),
        region=("Region", "first"),
    )
    .reset_index()
)

customer_agg["avg_order_value"] = customer_agg["total_revenue"] / cus

print("\nCustomer-level metrics:")
print(customer_agg.head())
```

```
Customer-level metrics:
  customer_id  total_revenue  order_count  total_quantity  first_purcha
0    cust001         261100           1             7      2024-01-1
1    cust002         61624           1             4      2024-01-1
2    cust003         43492           1             2      2024-01-1
3    cust004         30895           1             1      2024-01-1
4    cust005        318680           1             8      2024-01-1

  last_purchase  region  avg_order_value
0  2024-01-01    east      261100.0
1  2024-01-02  north       61624.0
2  2024-01-03  west       43492.0
3  2024-01-04    east       30895.0
4  2024-01-05  north      318680.0
```

## #Identify top customers

```
top_customers = customer_agg.sort_values("total_revenue", ascending=False)
print("\nTop 10 customers by revenue:")
print(top_customers)
```

Top 10 customers by revenue:

	customer_id	total_revenue	order_count	total_quantity	first_purchase
15	cust016	373932	1	9	2024-01-
6	cust007	363870	1	9	2024-01-
82	cust083	350888	1	8	2024-03-
72	cust073	349510	1	7	2024-03-
19	cust020	333992	1	8	2024-01-
83	cust084	324144	1	8	2024-03-
69	cust070	318762	1	9	2024-03-
4	cust005	318680	1	8	2024-01-
64	cust065	312564	1	7	2024-03-
27	cust028	304465	1	7	2024-01-

	last_purchase	region	avg_order_value
15	2024-01-16	south	373932.0
6	2024-01-07	south	363870.0
82	2024-03-23	north	350888.0
72	2024-03-13	west	349510.0
19	2024-01-20	north	333992.0
83	2024-03-24	south	324144.0
69	2024-03-10	south	318762.0
4	2024-01-05	north	318680.0
64	2024-03-05	south	312564.0
27	2024-01-28	north	304465.0

## #Regional distribution of customers

```
region_customer_counts = customer_agg["region"].value_counts()
print("\nCustomer counts by region:")
print(region_customer_counts)
```

```
Customer counts by region:
region
north    28
south    27
west     26
east     19
Name: count, dtype: int64
```

## ✓ Day 4: Sales Pattern Analysis

### #Monthly revenue

```
monthly_revenue = (
    full_df.groupby("year_month")
    .agg(
        total_revenue=("revenue", "sum"),
        order_count=("Date", "count"),
        total_quantity=("Quantity", "sum"),
    )
    .reset_index()
)
```

```

print("\nMonthly revenue summary:")
print(monthly_revenue.head())

# ---- Best-selling products
product_agg = (
    full_df.groupby("Product")
    .agg(
        total_revenue=("revenue", "sum"),
        total_quantity=("Quantity", "sum"),
        order_count=("Date", "count"),
    )
    .reset_index()
)
best_products = product_agg.sort_values("total_revenue", ascending=False)
print("\nTop 10 products by revenue:")
print(best_products)

```

Monthly revenue summary:

	year_month	total_revenue	order_count	total_quantity
0	2024-01	4120524	31	147
1	2024-02	2656050	29	112
2	2024-03	4485006	31	175
3	2024-04	1103468	9	44

Top 10 products by revenue:

	Product	total_revenue	total_quantity	order_count
1	laptop	3889210	136	24
4	tablet	2884340	127	26
3	phone	2859394	101	20
0	headphones	1384033	48	15
2	monitor	1348071	66	15

## #Regional performance

```

if "region_sale" in full_df.columns:
    region_col = "region_sale"
elif "region" in full_df.columns:
    region_col = "region"
else:
    region_col = None

if region_col:
    region_sales = (
        full_df.groupby(region_col)
        .agg(
            total_revenue=("revenue", "sum"),
            order_count=("order_id", "nunique"),
            total_quantity=("quantity", "sum"),
        )
        .reset_index()
    )
    print("\nSales by region:")

```

```
print(region_sales)
```

## ✓ Day 5: Advanced Analysis

**#Pivot** tables : Revenue by region and month

```
if region_col:
    pivot_region_month = pd.pivot_table(
        full_df,
        values="revenue",
        index=region_col,
        columns="year_month",
        aggfunc="sum",
        fill_value=0,
    )
    print("\nRevenue by region and month (pivot):")
    print(pivot_region_month)
```

**#Revenue** by region and product category

```
if region_col and "category" in full_df.columns:
    pivot_region_cat = pd.pivot_table(
        full_df,
        values="revenue",
        index=region_col,
        columns="category",
        aggfunc="sum",
        margins=True,
        margins_name="total",
        fill_value=0,
    )
    print("\nRevenue by region and category (pivot):")
    print(pivot_region_cat)
```

**#Multiple** aggregations in pivot

```
if "category" in full_df.columns:
    pivot_multi = pd.pivot_table(
        full_df,
        values=["revenue", "quantity"],
        index="category",
        aggfunc={"revenue": ["sum", "mean"], "quantity": "sum"},
        fill_value=0,
    )
    print("\nCategory pivot with multiple aggregations:")
    print(pivot_multi)
```

**#Cross-selling:** products that sell together in the same order

```
def get_product_pairs(df):
    pairs_list = []
    # Create a synthetic order_id by grouping by customer_id and Date
    # This assumes that all items bought by the same customer on the
    df_copy = df.copy()
    df_copy['synthetic_order_id'] = df_copy['customer_id'].astype(str)

    grouped = df_copy.groupby("synthetic_order_id")["Product"].apply(
        lambda products: sorted(set(products))
    )
    for products in grouped:
        unique_products = sorted(set(products))
        if len(unique_products) < 2:
            continue
        for i in range(len(unique_products)):
            for j in range(i + 1, len(unique_products)):
                pairs_list.append((unique_products[i], unique_products[j]))
    return pairs_list

product_pairs = get_product_pairs(full_df)
pairs_df = pd.DataFrame(product_pairs, columns=["product_a", "product_b"])
pair_counts = (
    pairs_df.value_counts()
    .reset_index(name="pair_count")
    .sort_values("pair_count", ascending=False)
)
top_pairs = pair_counts.head(10)
print("\nTop product pairs (cross-sell candidates):")
print(top_pairs)
```

```
Top product pairs (cross-sell candidates):
Empty DataFrame
Columns: [product_a, product_b, pair_count]
Index: []
```

**#Retention** (repeat purchase) and churn : Repeat purchase rate (simple retention proxy)

```
repeat_customers = customer_agg[customer_agg["order_count"] > 1].shape[0]
total_customers = customer_agg.shape[0]
repeat_rate = repeat_customers / total_customers
print(f"\nRepeat purchase rate: {repeat_rate:.2%}")

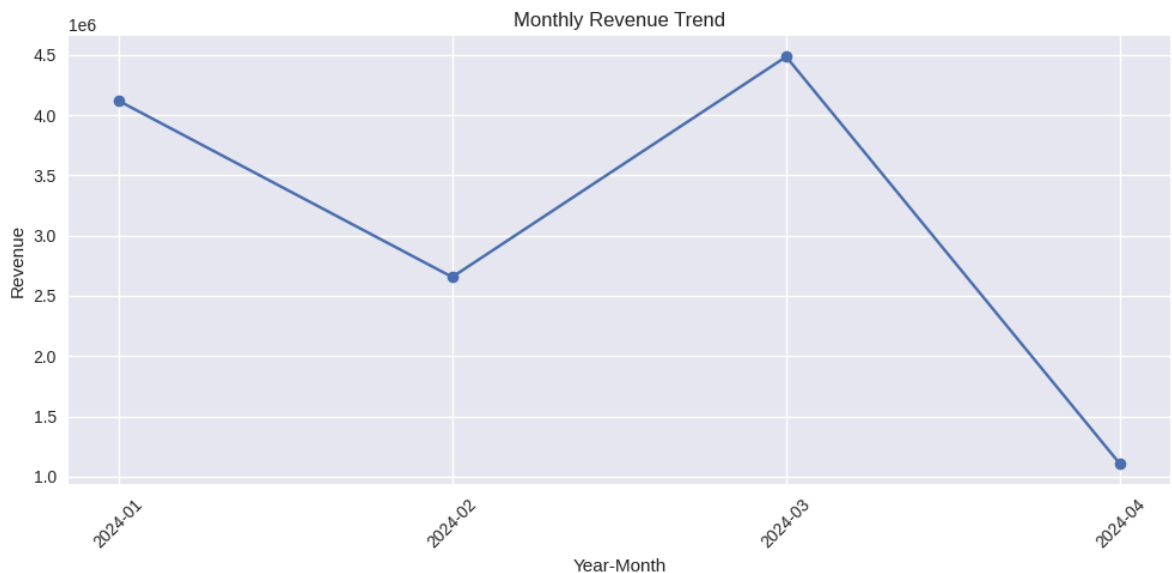
# If churn_df has columns ['customer_id', 'churned'], compute churn rate
if "churned" in churn_df.columns:
    churn_merged = pd.merge(customer_agg, churn_df, on="customer_id",
                             how="left")
    churn_rate = churn_merged["churned"].mean()
    print(f"Churn rate (from churn_df): {churn_rate:.2%}")
```

Repeat purchase rate: 0.00%

## ✓ Day 6: Dashboard Visualizations

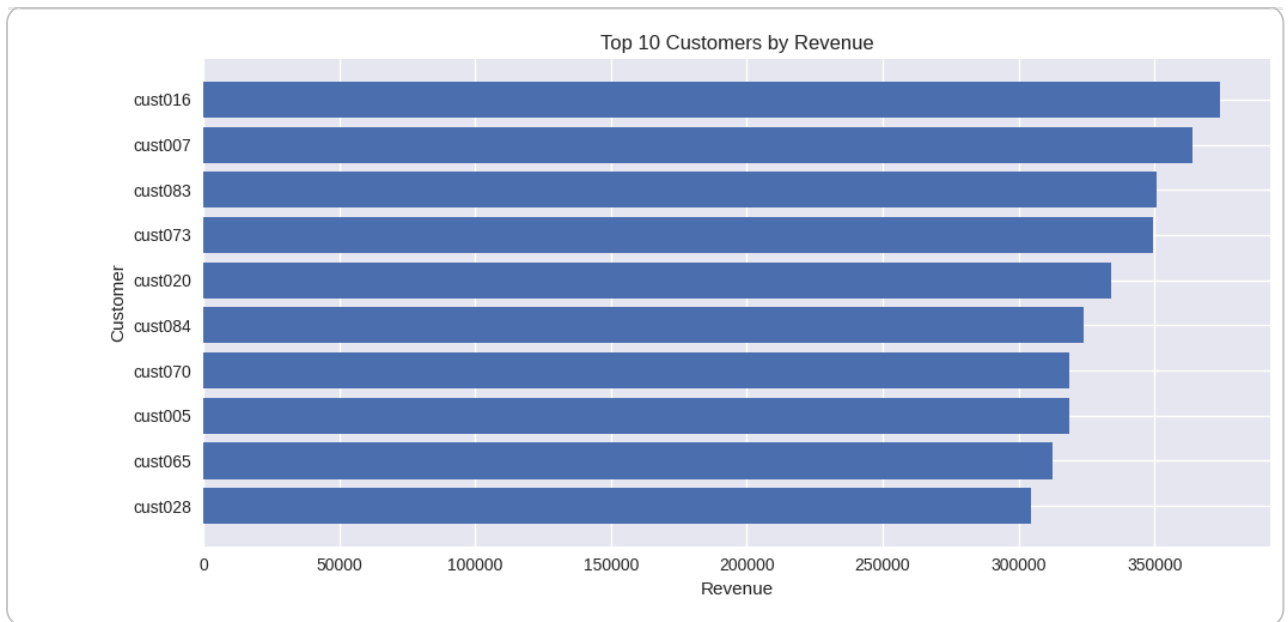
### Monthly revenue trend

```
plt.figure(figsize=(10, 5))
monthly_revenue_sorted = monthly_revenue.sort_values("year_month")
plt.plot(monthly_revenue_sorted["year_month"], monthly_revenue_sorted)
plt.xticks(rotation=45)
plt.title("Monthly Revenue Trend")
plt.xlabel("Year-Month")
plt.ylabel("Revenue")
plt.tight_layout()
plt.show()
```



### Top 10 customers by revenue

```
plt.figure(figsize=(10, 5))
top_customers_plot = top_customers.sort_values("total_revenue")
plt.barh(top_customers_plot["customer_id"], top_customers_plot["total_revenue"])
plt.title("Top 10 Customers by Revenue")
plt.xlabel("Revenue")
plt.ylabel("Customer")
plt.tight_layout()
plt.show()
```



### Revenue by region

```
if region_col:
    plt.figure(figsize=(8, 5))
    sns.barplot(
        data=region_sales.sort_values("total_revenue", ascending=False),
        x="total_revenue",
        y=region_col,
    )
    plt.title("Revenue by Region")
    plt.xlabel("Revenue")
    plt.ylabel("Region")
    plt.tight_layout()
    plt.show()
```

### Revenue by product category

```
if "category" in full_df.columns:
    category_revenue = (
        full_df.groupby("category")["revenue"].sum().sort_values(ascending=False)
    )
    plt.figure(figsize=(8, 5))
    sns.barplot(
        x=category_revenue.values,
        y=category_revenue.index,
    )
    plt.title("Revenue by Product Category")
    plt.xlabel("Revenue")
    plt.ylabel("Category")
    plt.tight_layout()
    plt.show()
```

### Heatmap of region vs month (from pivot)

```

if region_col:
    plt.figure(figsize=(10, 6))
    sns.heatmap(pivot_region_month, annot=False, cmap="Blues")
    plt.title("Revenue Heatmap: Region vs Month")
    plt.xlabel("Year-Month")
    plt.ylabel("Region")
    plt.tight_layout()
    plt.show()

```

## ✓ Day 7: Basic printed KPIs

```

total_revenue = full_df["revenue"].sum()
total_customers_kpi = customer_agg["customer_id"].nunique()
# Calculate total number of unique orders by considering unique custo
total_unique_orders = full_df[['customer_id', 'Date']].drop_duplicate
avg_order_value = total_revenue / total_unique_orders

top_customer_row = top_customers.iloc[0]
top_customer_name = top_customer_row["customer_id"] # Corrected from
top_customer_revenue = top_customer_row["total_revenue"]

print("\n==== CUSTOMER SALES ANALYSIS REPORT (KPI SUMMARY) =====")
print(f"Total Revenue: {total_revenue:,.2f}")
print(f"Total Customers: {total_customers_kpi:,}")
print(f"Average Order Value: {avg_order_value:,.2f}")
print(f"Top Customer: {top_customer_name} - {top_customer_revenue:,.2f}")
print(f"Repeat Purchase Rate: {repeat_rate:.2%}")
if "churn" in churn_df.columns: # Check for 'churn' column in churn_d
    # Recalculate churn_rate if not already done, or ensure it's avai
    if 'churn_merged' in locals() and 'Churn' in churn_merged.columns
        churn_rate = churn_merged["Churn"].mean()
        print(f"Churn Rate: {churn_rate:.2%}")
    elif 'Churn' in churn_df.columns: # Fallback if churn_merged wasn
        # Need to ensure that churn_df contains unique customer_ids i
        # For this specific KPI, it's safer to base it off the origin
        # Assuming 'Churn' column is 0/1 indicating non-churn/churn
        churn_rate_from_df = churn_df['Churn'].mean()
        print(f"Churn Rate: {churn_rate_from_df:.2%}")

```

```

===== CUSTOMER SALES ANALYSIS REPORT (KPI SUMMARY) =====
Total Revenue: 12,365,048.00
Total Customers: 100
Average Order Value: 123,650.48
Top Customer: cust016 - 373,932.00
Repeat Purchase Rate: 0.00%

```

