www.svnapro.com

Exploratory Data Analysis

info@svnapro.com
+91 79899 75085

svnapro

Imports & Notebook Settings

• Import essential libraries: Pandas (data handling), NumPy (numerical operations), Matplotlib (visualization).
• Apply a consistent plotting style (ggplot) for uniform visuals.
• Define dataset file paths upfront to avoid repetitive typing.
• Definition: This step is called “notebook setup” — it ensures all students work in the same environment.
• Expected outcome: Notebook ready, style consistent, dataset path available.

Code crux

main.py

```
import pandas as pd, numpy as np, matplotlib.pyplot as plt
plt.style.use('ggplot')
DATA_PATH = "/mnt/data/Realistic_E-Commerce_Dataset.csv"
```

svnapro

Load dataset

- Load dataset into Pandas DataFrame.
- Use head() to preview the first rows.
- Use shape to check dataset size (rows × columns).
- Use columns to list fields.
- Definition: Loading is the first validation step — we confirm that the dataset exists and is usable.
- Expected outcome: DataFrame with key columns (purchase, date, etc.) identified.

Code crux

main.py

```
df = pd.read_csv(DATA_PATH)
df.head()
df.shape
df.columns
```

svnapro

Quick numeric summary & missing values

- Generate descriptive statistics for numeric columns (mean, median, quartiles, min, max).
- Spot anomalies (e.g., unrealistic ages).
- Compare mean vs. median to detect skew.
- Check missing values and sort by frequency.
- Definition: This is part of data quality assessment — spotting issues early.
- Expected outcome: Awareness of missing data, skewness, and potential outliers.

Code crux

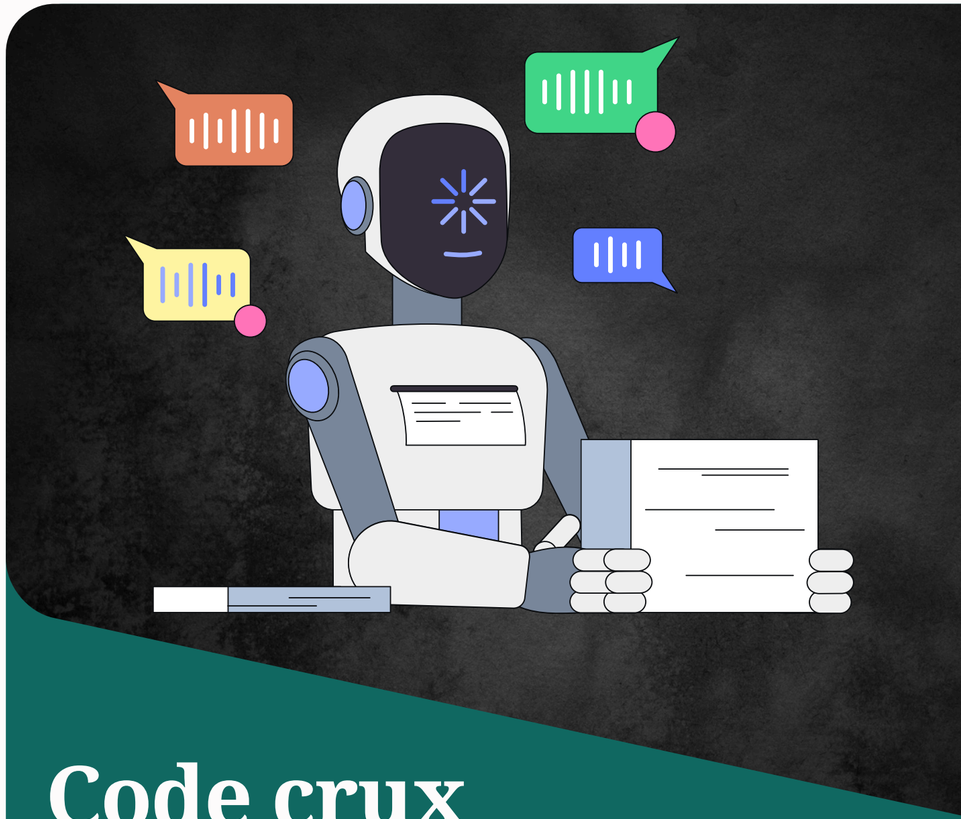
main.py

```
df.select_dtypes(include='number').describe().T
df.isna().sum().sort_values(ascending=False)
```

svnapro

Clean types, parse dates, check duplicates

- Convert dates to datetime format for time analysis.
- Coerce numeric columns to valid numbers.
- Check for duplicate rows that may bias analysis.
- Definition: Type cleaning ensures each column is stored in the right data type (datetime, numeric, categorical).
- Expected outcome: Proper datatypes and confirmation of duplicates (ideally none).



Code crux

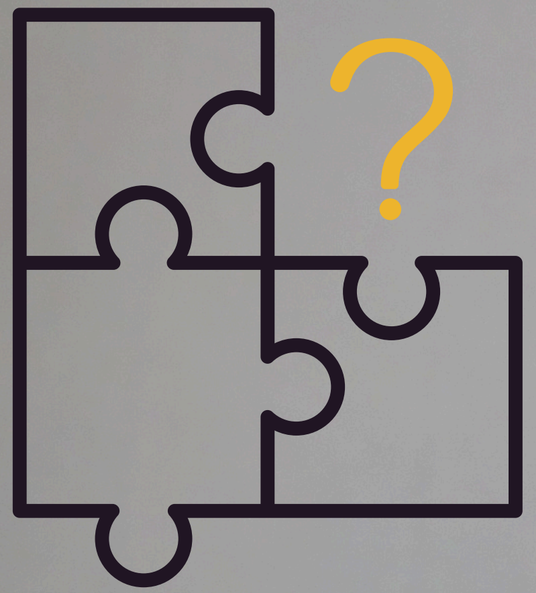
main.py

```
df['date'] = pd.to_datetime(df['date'], errors='coerce')
 for c in
['age','time_spent_minutes','pages_viewed','cart_it
ems']:
 df[c] = pd.to_numeric(df[c], errors='coerce')
 df.duplicated().sum()
```

svnapro

Handling missing values

- If dataset is too clean, inject a few missing values for demonstration.
- Example: remove values from age and time_spent_minutes.
- Definition: This is a teaching trick, not a real-world step — it allows practicing imputation.
- Expected outcome: A few visible NaNs to work with.



Code crux

main.py

```
# only if df.isna().sum().sum() == 0
df.loc[[5, 12, 55], 'age'] = pd.NA
df.loc[[10, 22, 78], 'time_spent_minutes'] = pd.NA
```

svnapro

Imputation (simple & explainable)

- Replace missing numeric values with the median.
- Replace missing categorical values with “Unknown”.
- Definition: Imputation is the process of filling missing values to maintain dataset usability.
- Why median? Robust against skewed distributions.
- Expected outcome: No missing values in key columns.

Code crux

main.py

```
df['age'].fillna(df['age'].median(), inplace=True)

df['time_spent_minutes'].fillna(df['time_spent_minutes'].median(), inplace=True)
df['product_category'].fillna('Unknown', inplace=True)
```

svnapro

Derived features

- Engineer new features to capture behavior and patterns:
  - – Day of week, Hour (from date).
  - – Age groups (binned).
  - – Cart-to-wishlist ratio.
  - – Engagement score combining multiple signals.
  - – Value per cart item.
- Definition: Derived (engineered) features are created from existing ones to reveal deeper insights.
- Expected outcome: New, interpretable columns useful for analysis and modeling.

Code crux

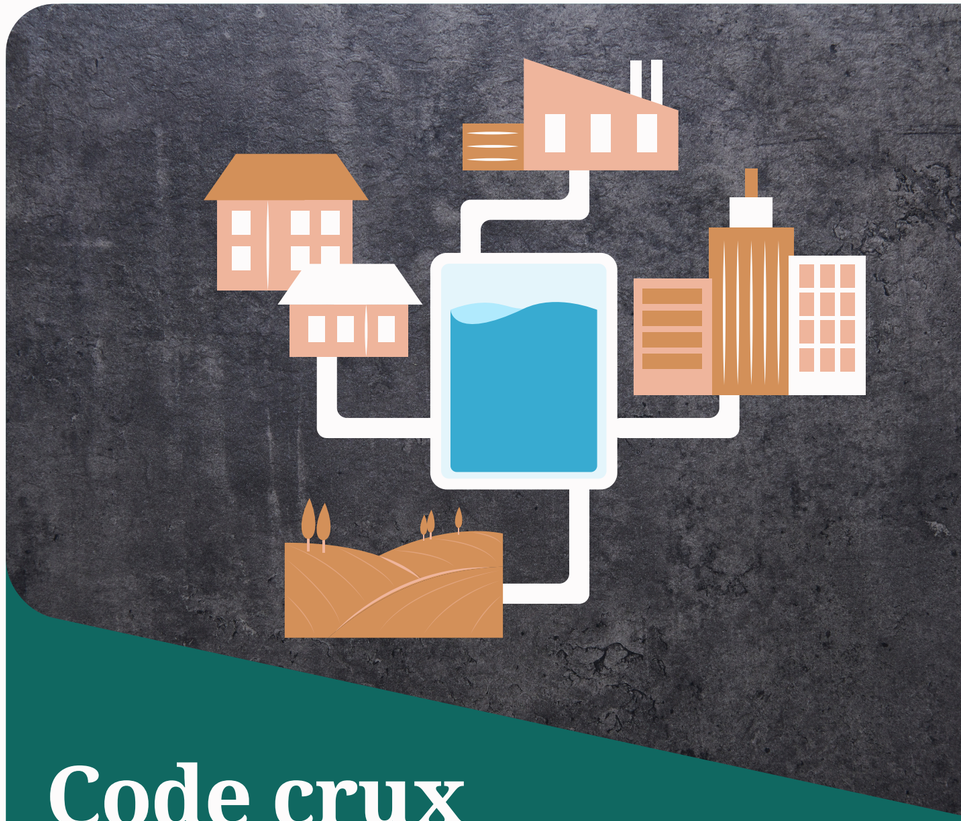
main.py

```
df['day_of_week'] = df['date'].dt.day_name()
df['hour'] = df['date'].dt.hour
df['age_group'] = pd.cut(df['age'], bins=[17,25,40,65], labels=['18-25','26-40','41-65'])
df['cart_to_wishlist_ratio'] = df.apply(lambda r: r.cart_items/r.wishlist_items if r.wishlist_items>0 else r.cart_items, axis=1)
```

svnapro

Univariate EDA — Numeric distributions

- Univariate analysis: studying one variable at a time.
- Focus on distribution, skew, and outliers.
- Numeric features (e.g., session value, age) analyzed via histograms/boxplots.
- Helps decide on transformations (log, binning, capping).
- Expected outcome: Identify skewed or outlier-heavy distributions.



Code crux

main.py

```
plt.hist(df['avg_session_value'], bins=30)
plt.axvline(df['avg_session_value'].median(), color='black')
plt.axvline(df['avg_session_value'].mean(), linestyle='--')
```

svnapro

Univariate EDA — Categorical counts

- Univariate analysis for categorical variables: examine frequency of categories.
- Detect dominant vs. rare categories.
- Helps decide whether to merge rare ones into “Other”.
- Expected outcome: Clear picture of category distribution.

Code crux

main.py

```
df['product_category'].value_counts().plot(kind='bar')
```

svnapro

Target variable — Purchase

- Analyze the distribution of the target column purchase.
- Check proportion of buyers vs. non-buyers.
- Definition: This is a univariate check of the dependent variable.
- Expected outcome: Understanding of class balance (e.g., 10% buyers).

Code crux

main.py

```
rate = df['purchase'].mean()
plt.pie([1-rate, rate], labels=['No','Yes'],
autopct='%1.1f%%')
```

svnapro

Bivariate EDA — Numeric vs. target

- Bivariate analysis: exploring relationship between two variables.
- Here: numeric feature vs. target variable.
- Boxplots highlight differences between buyers and non-buyers.
- Expected outcome: Numeric variables that strongly separate classes.

Code crux

main.py

```
plt.boxplot([df[df.purchase==0]
['time_spent_minutes'], df[df.purchase==1]
['time_spent_minutes']], labels=['No','Yes'])
```

svnapro

Bivariate EDA — Categorical vs. target

- Bivariate analysis with categorical variables: compare categories against target outcome.
- Conversion rates show which categories perform better.
- Expected outcome: Identification of high-performing product categories.

Code crux

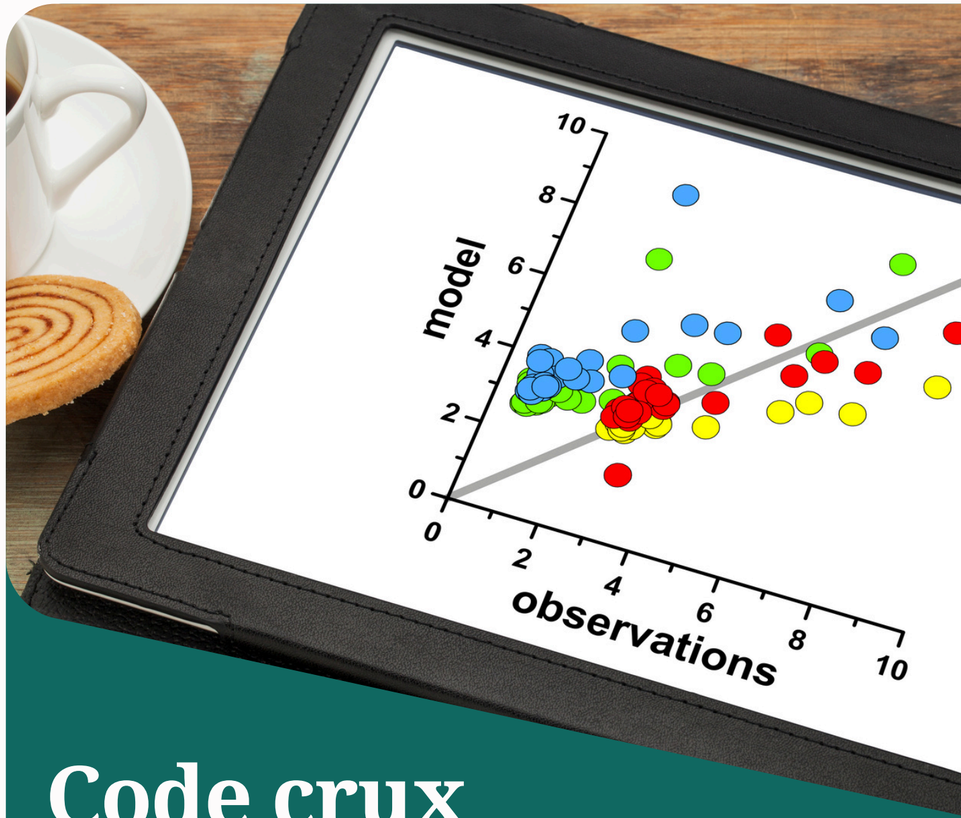
main.py

```
conv = df.groupby('product_category')
['purchase'].mean().sort_values(ascending=False)
conv.head(10).plot(kind='bar')
```

svnapro

Multivariate EDA — Correlation matrix

- Multivariate analysis: studying relationships among 3+ variables.
- Correlation matrix shows pairwise relationships between numeric features.
- Helps detect redundancy (multicollinearity).
- Expected outcome: Clear heatmap of strong/weak correlations.



Code crux

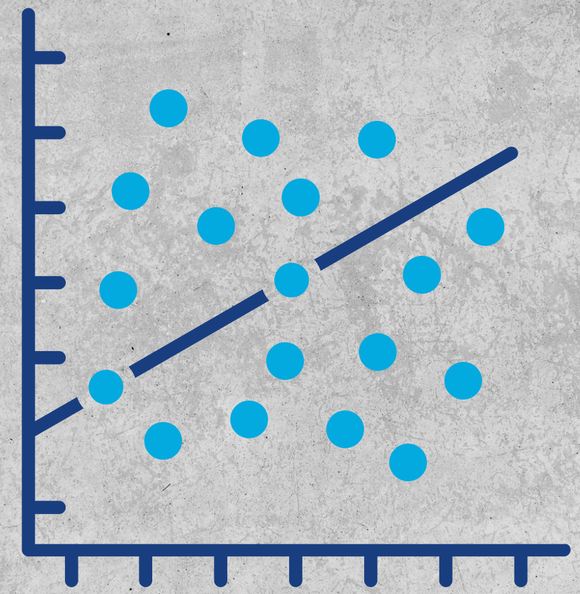
main.py

```
num = df.select_dtypes(include='number')
plt.imshow(num.corr(), cmap='coolwarm', vmin=-1, vmax=1)
```

svnapro

Multivariate EDA — Pairwise scatter

- Another form of multivariate analysis.
- Scatter plots visualize relationships among multiple variables.
- Good for spotting clusters, non-linear patterns, and outliers.
- Expected outcome: Recognition of hidden structures in the data.



Code crux

main.py

```
pd.plotting.scatter_matrix(df[['time_spent_minutes','pages_viewed','avg_session_value']].sample(300), diagonal='hist')
```

svnapro

Segmented analysis — Pivot & grouped views

- Segmented analysis: study performance broken down by groups.
- Example: membership type × product category.
- Pivot tables reveal how conversions vary across combinations.
- Expected outcome: Identification of valuable customer segments.

Code crux

main.py

```
pivot = df.pivot_table(index='membership_status',
columns='product_category', values='purchase',
aggfunc='mean')
 display(pivot)
```

svnapro

Derived metrics exploration

- Derived features: engineered from existing columns to capture deeper insights.
- Example: engagement_score combines clicks, time, and pages.
- Compare separation power between derived vs. raw features.
- Expected outcome: Derived features that show stronger predictive signals.

Code crux

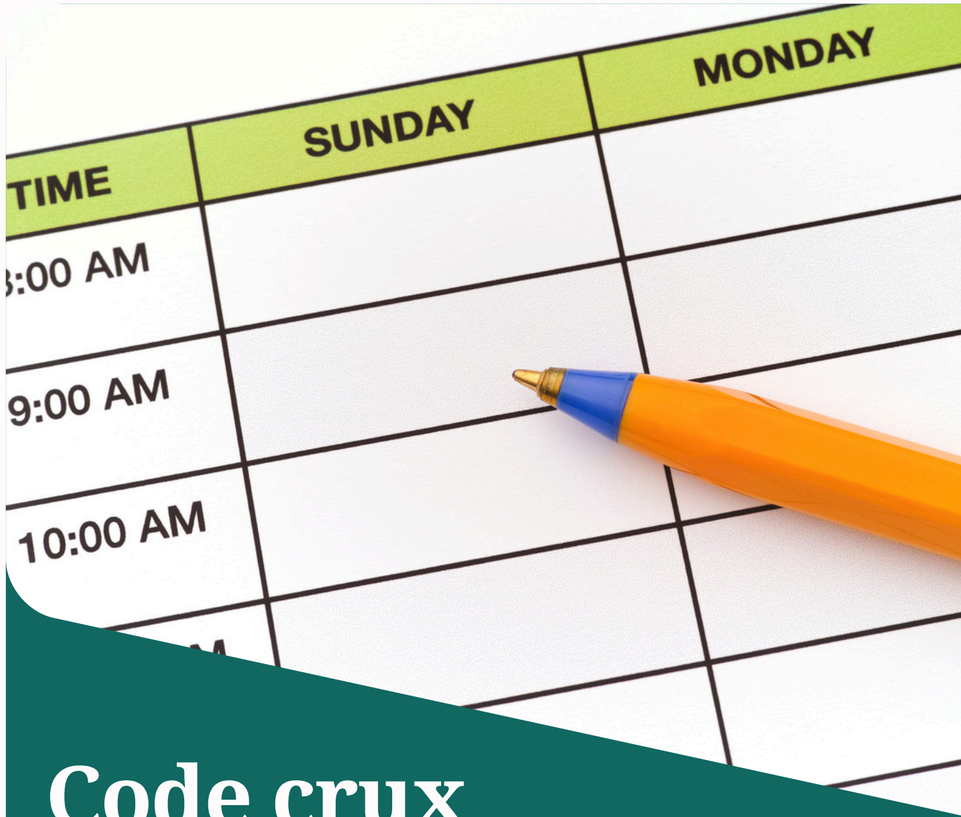
main.py

```
plt.boxplot([df[df.purchase==0]
['engagement_score'], df[df.purchase==1]
['engagement_score']], labels=['No','Yes'])
```

svnapro

Time-series EDA — Daily & hourly patterns

- Aggregate data by day and hour.
- Detect seasonality, spikes, and best conversion hours.
- Definition: Time-series EDA focuses on temporal patterns in data.
- Expected outcome: Daily/weekly trends and hourly peaks for campaigns.



Code crux

main.py

```
df['date_only'] = df['date'].dt.date
daily = df.groupby('date_only').agg(total=('purchase','count'), purchases=('purchase','sum'))
daily['rate'] = daily['purchases'] / daily['total']
daily['rate'].plot()
hourly = df.groupby('hour')['purchase'].mean()
hourly.plot()
```

svnapro

Summary of EDA findings

- Key observations:
  – Engagement metrics correlate positively with purchase.
  – Premium users convert higher than guests.
  – Evenings show stronger conversions.
  – Monetary variables are skewed, log-transform may help.
- Definition: A summary consolidates insights into actionable findings.
- Expected outcome: Clear recommendations for modeling and business use.

Thank You!

www.svnapro.com

info@svnapro.com
+91 79899 75085

svnapro