Start coding or generate with AI.

Project 1: Daily Transaction

Start coding or generate with AI.

```
# Import libraries
import os
import warnings
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings("ignore")
sns.set_theme(style="whitegrid")
```

Configuration

```
# Configuration
INPUT_CSV = "Daily Household Transactions.csv"
OUTPUT_DIR = "./output_project"
CLEANED_CSV = os.path.join(OUTPUT_DIR, "cleaned_transactions.csv")
REPORT_TXT = os.path.join(OUTPUT_DIR, "report_summary.txt")
```

```
os.makedirs(OUTPUT_DIR, exist_ok=True)
```

Utility functions

```
# Utility functions
def safe_read_csv(path):
    """
    Read CSV with robust parsing. Try common encodings and separators.
    """
    try:
        df = pd.read_csv(path, parse_dates=['Date'], dayfirst=True, low_memory=False)
        return df
    except Exception:
        # Fallbacks: try different encodings / sep
        for enc in ['utf-8', 'latin1', 'cp1252']:
            for sep in [',', ';', '\t']:
                try:
                    df = pd.read_csv(path, encoding=enc, sep=sep,
                                     parse_dates=['Date'], dayfirst=True, low_memory=False)
                    return df
                except Exception:
                    continue
    raise RuntimeError(f"Failed to read CSV: {path}")
def normalize_text_columns(df, cols):
    for c in cols:
        if c in df.columns:
            df[c] = df[c].astype(str).str.strip().str.lower().replace({'nan': np.nan})
    return df

def ensure_amount_numeric(df, col='Amount'):
    if col not in df.columns:
        raise KeyError(f"Column '{col}' not found in dataframe.")
    # Remove any stray currency symbols or commas and convert to float
    df[col] = df[col].astype(str).str.replace(r'[^\d\.\-]', '', regex=True)
    df[col] = pd.to_numeric(df[col], errors='coerce')
    return df
```

```
def derive_datetime_parts(df, date_col='Date'):
    df['date'] = pd.to_datetime(df[date_col], errors='coerce')
    df['year'] = df['date'].dt.year
    df['month'] = df['date'].dt.month
    df['month_name'] = df['date'].dt.strftime('%b')
    df['day'] = df['date'].dt.day
    df['weekday'] = df['date'].dt.day_name()
    df['hour'] = df['date'].dt.hour
    return df
```

Load Data

```
# Load data
df_raw = safe_read_csv(INPUT_CSV)
print(f"Loaded data: {df_raw.shape[0]} rows, {df_raw.shape[1]} columns")
```

```
Loaded data: 2461 rows, 8 columns
```

Initial Inspection

```
# Initial Inspection
print("\nColumns found:", df_raw.columns.tolist())
print("\nSample rows:")
display_sample = df_raw.head(5)
print(display_sample.to_string(index=False))
```

```
Columns found: ['Date', 'Mode', 'Category', 'Subcategory', 'Note', 'Amount', 'Income/Expense', 'Currency']

Sample rows:
            Date                Mode       Category            Subcategory                    Note  Amount Income/Expense Currency
20/09/2018 12:04:08                Cash Transportation                  Train        2 Place 5 to Place 0    30.0        Expense      INR
20/09/2018 12:03:15                Cash           Food                snacks Idli medu Vada mix 2 plates    60.0        Expense      INR
        19/09/2018 Saving Bank account 1    subscription                Netflix         1 month subscription   199.0        Expense      INR
17/09/2018 23:41:17 Saving Bank account 1    subscription Mobile Service Provider           Data booster pack    19.0        Expense      INR
16/09/2018 17:15:08                Cash      Festivals            Ganesh Pujan                 Ganesh idol   251.0        Expense      INR
```

Data Cleaning

```
# Data Cleaning
df = df_raw.copy()
```

```
# Standardize column names (strip spaces)
df.columns = [c.strip().replace(' ', '_') for c in df.columns]
```

```
# Normalize common text columns
text_cols = [c for c in ['Mode', 'Category', 'Subcategory', 'Note', 'Income/Expense', 'Currency'] if c in df.columns]
df = normalize_text_columns(df, text_cols)
```

```
# Convert amount to numeric
df = ensure_amount_numeric(df, col='Amount')
```

```
# Drop rows without any date or amount
df = df.dropna(subset=['Date', 'Amount']).reset_index(drop=True)
```

```
# Remove exact duplicates
df = df.drop_duplicates()
```

```
# Clean Income/Expense values and unify them to 'income' / 'expense'
if 'Income/Expense' in df.columns:
    df['Income/Expense'] = df['Income/Expense'].str.lower().str.replace(r'[^a-z]', '', regex=True)
    df['Income_Expense'] = df['Income/Expense'].map(lambda x: 'income' if 'inc' in str(x) else ('expense' if 'exp' in str(x) else x))
    df.drop(columns=['Income/Expense'], inplace=True)
else:
    # If no explicit column, derive from sign of Amount (positive => income)
    df['Income_Expense'] = np.where(df['Amount'] >= 0, 'income', 'expense')
```

```
# Standardize Category and Subcategory - fill missing with Unknown
for c in ['Category', 'Subcategory', 'Mode', 'Note', 'Currency']:
    if c in df.columns:
        df[c] = df[c].fillna('unknown')
```

```
# Derive datetime parts
df = derive_datetime_parts(df, date_col='Date')
```

```
# A quick correction: if Amount column is always positive but Income_Expense says expense, convert sign convention
# We'll create signed_amount where expenses are negative and incomes positive
df['signed_amount'] = df['Amount'].copy()
df.loc[df['Income_Expense'] == 'expense', 'signed_amount'] = -df.loc[df['Income_Expense'] == 'expense', 'Amount']
df.loc[df['Income_Expense'] == 'income', 'signed_amount'] = df.loc[df['Income_Expense'] == 'income', 'Amount']
```

```
# Save cleaned file
df.to_csv(CLEANED_CSV, index=False)
print(f"Cleaned data saved to {CLEANED_CSV}")
```

```
Cleaned data saved to ./output_project\cleaned_transactions.csv
```

Exploratory Data Analysis (EDA)

```
# 1. Basic summaries
summary_stats = df['signed_amount'].describe()
num_transactions = len(df)
num_categories = df['Category'].nunique() if 'Category' in df.columns else 0
num_modes = df['Mode'].nunique() if 'Mode' in df.columns else 0

print("\nSummary statistics for signed amounts:")
print(summary_stats)
```

```
Summary statistics for signed amounts:
count      2452.000000
mean       1163.049641
std       12789.038604
min     -100000.000000
25%        -296.500000
50%         -64.410000
75%         -23.750000
max      250000.000000
Name: signed_amount, dtype: float64
```

```
# 2. Top categories by total spend (absolute)
if 'Category' in df.columns:
    category_agg = df.groupby('Category')['signed_amount'].sum().sort_values()
    top_expense_categories = category_agg.head(10)  # most negative (largest expense)
    top_income_categories = category_agg.tail(10)   # positive incomes
else:
    category_agg = pd.Series(dtype=float)
    top_expense_categories = pd.Series(dtype=float)
    top_income_categories = pd.Series(dtype=float)
```

Visualizations

```
# Helper for saving figures
def save_fig(fig, name):
    path = os.path.join(OUTPUT_DIR, name)
    fig.savefig(path, bbox_inches='tight', dpi=150)
    plt.close(fig)
    print(f"Saved figure: {path}")
```

```
# Plot 1: Distribution of transaction amounts (signed)
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1,1,1)
sns.histplot(df['signed_amount'], bins=60, kde=True, ax=ax)
ax.set_title('Distribution of Signed Transaction Amounts (income positive, expense negative)')
```

```
ax.set_xlabel('Signed Amount')
save_fig(fig, "distribution_signed_amounts.png")
```

```
Saved figure: ./output_project\distribution_signed_amounts.png
```

```
# Plot 2: Count of transactions by category (top 12)
if 'Category' in df.columns:
    top_cats = df['Category'].value_counts().nlargest(12)
    fig = plt.figure(figsize=(12,6))
    ax = fig.add_subplot(1,1,1)
    sns.barplot(x=top_cats.values, y=top_cats.index, orient='h', ax=ax)
    ax.set_title('Top 12 Categories by Transaction Count')
    ax.set_xlabel('Transaction Count')
    save_fig(fig, "top12_categories_count.png")
```

```
Saved figure: ./output_project\top12_categories_count.png
```

```
# Plot 3: Monthly trend (sum of signed amounts)
monthly = df.set_index('date').resample('M')['signed_amount'].sum()
fig = plt.figure(figsize=(12,5))
ax = fig.add_subplot(1,1,1)
monthly.plot(marker='o', ax=ax)
ax.set_title('Monthly Net Amount (Income - Expense)')
ax.set_ylabel('Net Amount')
ax.grid(True)
save_fig(fig, "monthly_net_amount.png")
```

```
Saved figure: ./output_project\monthly_net_amount.png
```

```
# Plot 4: 7-day rolling average of daily absolute expense (expenses only)
daily = df.set_index('date').resample('D')['signed_amount'].sum()
rolling7 = daily.rolling(window=7, min_periods=1).sum()  # 7-day rolling total net
fig = plt.figure(figsize=(12,5))
ax = fig.add_subplot(1,1,1)
rolling7.plot(ax=ax)
ax.set_title('7-day Rolling Net Amount')
ax.set_ylabel('7-day Net Amount')
save_fig(fig, "rolling7_net_amount.png")
```

```
Saved figure: ./output_project\rolling7_net_amount.png
```

```
# Plot 5: Income vs Expense monthly stacked
if 'Income_Expense' in df.columns:
    inc_exp_month = df.copy()
    inc_exp_month['abs_amount'] = inc_exp_month['Amount'].abs()
    pivot_month = inc_exp_month.groupby([inc_exp_month['date'].dt.to_period('M'), 'Income_Expense'])['abs_amount'].sum().unst
    pivot_month.index = pivot_month.index.to_timestamp()
    fig = plt.figure(figsize=(12,6))
    ax = fig.add_subplot(1,1,1)
    pivot_month.plot(kind='bar', stacked=True, ax=ax, width=0.8)
    ax.set_title('Monthly Income vs Expense (stacked absolute amounts)')
    ax.set_xlabel('Month')
    ax.set_ylabel('Amount')
    plt.xticks(rotation=45)
    save_fig(fig, "monthly_income_vs_expense_stacked.png")
```

```
Saved figure: ./output_project\monthly_income_vs_expense_stacked.png
```

```
# Plot 6: Average transaction amount per weekday
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(1,1,1)
weekday_order = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
weekday_avg = df.groupby('weekday')['signed_amount'].mean().reindex(weekday_order)
sns.barplot(x=weekday_avg.index, y=weekday_avg.values, ax=ax)
ax.set_title('Average Signed Transaction Amount by Weekday')
ax.set_xlabel('Weekday')
ax.set_ylabel('Average Signed Amount')
plt.xticks(rotation=45)
save_fig(fig, "weekday_avg_signed_amount.png")
```

```
Saved figure: ./output_project\weekday_avg_signed_amount.png
```

```
# Plot 7: Pie chart of expense composition (only expenses)
expenses = df[df['Income_Expense'] == 'expense']
if not expenses.empty:
    expense_by_cat = expenses.groupby('Category')['Amount'].sum().sort_values(ascending=False).head(8)
    fig = plt.figure(figsize=(7,7))
    ax = fig.add_subplot(1,1,1)
    ax.pie(expense_by_cat, labels=expense_by_cat.index, autopct='%1.1f%%', startangle=140)
    ax.set_title('Top 8 Expense Categories by Spend Share')
    save_fig(fig, "expense_pie_top8.png")
```

```
Saved figure: ./output_project\expense_pie_top8.png
```

```
# Plot 8: Boxplot of amounts by top 6 categories (to show spread/outliers)
if 'Category' in df.columns:
    top6 = df['Category'].value_counts().nlargest(6).index
    fig = plt.figure(figsize=(10,6))
    ax = fig.add_subplot(1,1,1)
    sns.boxplot(data=df[df['Category'].isin(top6)], x='Category', y='Amount', ax=ax)
    ax.set_title('Amount Distribution for Top 6 Categories')
    ax.set_ylabel('Amount (absolute)')
    plt.xticks(rotation=45)
    save_fig(fig, "boxplot_top6_categories.png")
```

```
Saved figure: ./output_project\boxplot_top6_categories.png
```

```
# Plot 9: Heatmap of category correlations (based on monthly sums)
if 'Category' in df.columns:
    pivot = df.pivot_table(index=df['date'].dt.to_period('M'), columns='Category', values='Amount', aggfunc='sum', fill_value=0)
    corr = pivot.corr()
    fig = plt.figure(figsize=(12,10))
    ax = fig.add_subplot(1,1,1)
    sns.heatmap(corr, vmax=1.0, vmin=-1.0, center=0, cmap='coolwarm', square=True, linewidths=.5, cbar_kws={"shrink": .5}, ax=
    ax.set_title('Correlation between Categories (monthly sums)')
    save_fig(fig, "category_correlation_heatmap.png")
```

```
Saved figure: ./output_project\category_correlation_heatmap.png
```

Insights Calculations

```
insights = []
# 1. Net balance
net_balance = df['signed_amount'].sum()
insights.append(f"Net balance over period (income positive, expenses negative): {net_balance:,.2f}")
```

```
# 2. Top expense categories (by absolute spend)
if not expenses.empty:
    expense_by_cat_all = expenses.groupby('Category')['Amount'].sum().sort_values(ascending=False)
    top3_expense = expense_by_cat_all.head(3)
    insights.append("Top 3 expense categories by total spend:")
    for cat, val in top3_expense.items():
        insights.append(f"  - {cat}: {val:,.2f}")
```

```
# 3. Month with highest net expense (most negative net)
monthly_net = df.set_index('date').resample('M')['signed_amount'].sum()
if not monthly_net.empty:
    worst_month = monthly_net.idxmin()
    best_month = monthly_net.idxmax()
    insights.append(f"Month with lowest net (largest net expense): {worst_month.strftime('%Y-%m')} -> {monthly_net.min():,.2f}"
```

```
# 4. Average transaction size and median
avg_trans = df['Amount'].mean()
median_trans = df['Amount'].median()
insights.append(f"Average transaction (absolute) amount: {avg_trans:,.2f}")
insights.append(f"Median transaction amount: {median_trans:,.2f}")
```

```
# 5. Suggestion: if a weekday has consistently high spending
weekday_spend = df.groupby('weekday')['signed_amount'].sum().reindex(weekday_order)
high_spend_day = weekday_spend.idxmin()  # because expenses negative -> min is highest expense
insights.append(f"Weekday with highest net expense: {high_spend_day} (net: {weekday_spend.min():,.2f})")
```

Save Report

```
with open(REPORT_TXT, 'w') as f:
    f.write("Daily Household Transactions — Analysis Summary\n")
    f.write("="*60 + "\n\n")
    f.write(f"Data file: {INPUT_CSV}\n")
    f.write(f"Rows processed: {num_transactions}\n")
    f.write(f"Unique categories: {num_categories}\n")
    f.write(f"Unique payment modes: {num_modes}\n\n")
    f.write("Key insights:\n")
    for line in insights:
        f.write(line + "\n")
    f.write("\nFiles produced (plots and cleaned CSV) are in the 'output_project' folder.\n")

print(f"\nReport saved to {REPORT_TXT}")
```

```
Report saved to ./output_project\report_summary.txt
```

End

```
print("\nProject complete. Outputs:")
for name in os.listdir(OUTPUT_DIR):
    print(" -", name)
```

```
Project complete. Outputs:
 - boxplot_top6_categories.png
 - category_correlation_heatmap.png
 - cleaned_transactions.csv
 - distribution_signed_amounts.png
 - expense_pie_top8.png
 - monthly_income_vs_expense_stacked.png
 - monthly_net_amount.png
 - report_summary.txt
 - rolling7_net_amount.png
 - top12_categories_count.png
 - weekday_avg_signed_amount.png
```

Start coding or generate with AI.