Capstone — Minimal Viable Submission

This single notebook implements the **minimal** end-to-end pipeline required for the capstone:

- Minimal cleaning
- Required EDA plots (5)
- Baseline forecasting (RandomForest) for total daily sales (next 30 days)
- Simple store segmentation (KMeans)
- Save outputs for submission

Follow cells in order and run everything. Paths and outputs are saved under `/mnt/data/`.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
# Imports and paths
import pandas as pd, numpy as np, matplotlib.pyplot as plt, os
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.cluster import KMeans
import pickle

DATA_PATH = '/content/drive/MyDrive/Retail_Capstone_Project/Retail_Sales_Data_Unlox_(1)[1].csv'
OUTPUT_DIR = '/content/drive/MyDrive/Retail_Capstone_Project/capstone_outputs'
os.makedirs(OUTPUT_DIR, exist_ok=True)
print('Output dir:', OUTPUT_DIR)
```

```
Output dir: /content/drive/MyDrive/Retail_Capstone_Project/capstone_outputs
```

```
# Load dataset
df = pd.read_csv(DATA_PATH)
print('Rows, cols:', df.shape)
df.head()
```

```
Rows, cols: (73000, 19)
```

	Date	Store_ID	Store_Location	Product_ID	Product_Category	Product_Subcategory	Brand	Unit_Price	Units_Sold	Tota
0	2023-04-13	STR_104	Chennai	PRD_072	Sports	Athletics	Reebok	29973.06	39	116
1	2024-10-25	STR_103	Delhi	PRD_492	Sports	Outdoor	Yonex	46933.78	2	9
2	2023-02-28	STR_107	Kolkata	PRD_958	Groceries	Household	Nestle	39280.28	44	172
3	2023-06-16	STR_102	Bangalore	PRD_014	Home Appliances	Kitchen	Whirlpool	40439.03	9	36
4	2024-05-23	STR_108	Ahmedabad	PRD_932	Fashion	Women Clothing	Puma	9193.58	4	3

Next steps: Generate code with df | New interactive sheet

```
# Minimal cleaning (only what's necessary)
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
# Remove rows with invalid dates (if any)
df = df.dropna(subset=['Date']).copy()
# Ensure numeric columns
for c in ['Unit_Price','Units_Sold','Total_Sales','Discount_Percentage','Revenue','Stock_On_Hand','Store_Rating']:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors='coerce')
# Fill small missing numeric values with 0 for simplicity
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
df[num_cols] = df[num_cols].fillna(0)
# Basic derived columns
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df['DayOfWeek'] = df['Date'].dt.dayofweek
df['Is_Weekend'] = df['DayOfWeek'].isin([5,6]).astype(int)
```

```
# Save cleaned sample
cleaned_path = os.path.join(OUTPUT_DIR, 'processed_sales_minimal.csv')
df.to_csv(cleaned_path, index=False)
print('Cleaned saved to', cleaned_path)
df.head()
```

```
Cleaned saved to /content/drive/MyDrive/Retail_Capstone_Project/capstone_outputs/processed_sales_minimal.csv
```

	Date	Store_ID	Store_Location	Product_ID	Product_Category	Product_Subcategory	Brand	Unit_Price	Units_Sold	Tota
0	2023-04-13	STR_104	Chennai	PRD_072	Sports	Athletics	Reebok	29973.06	39	116
1	2024-10-25	STR_103	Delhi	PRD_492	Sports	Outdoor	Yonex	46933.78	2	9
2	2023-02-28	STR_107	Kolkata	PRD_958	Groceries	Household	Nestle	39280.28	44	172
3	2023-06-16	STR_102	Bangalore	PRD_014	Home Appliances	Kitchen	Whirlpool	40439.03	9	36
4	2024-05-23	STR_108	Ahmedabad	PRD_932	Fashion	Women Clothing	Puma	9193.58	4	3

5 rows × 24 columns

```
# EDA: create aggregate daily sales (Total_Sales) and required plots
daily = df.groupby('Date', as_index=False)['Total_Sales'].sum().sort_values('Date')
daily = daily.rename(columns={'Total_Sales':'Daily_Sales'})

# 1. Monthly sales trend
monthly = daily.copy()
monthly['Month'] = monthly['Date'].dt.to_period('M')
monthly_agg = monthly.groupby('Month')['Daily_Sales'].sum().reset_index()
plt.figure(figsize=(10,4)); plt.plot(monthly_agg['Month'].astype(str), monthly_agg['Daily_Sales']); plt.xticks(rotation=45)
plt.title('Monthly Sales Trend'); plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR,'plot_monthly_sales.png')); plt.close()

# 2. Category-wise sales (top 10)
cat_sales = df.groupby('Product_Category', as_index=False)['Total_Sales'].sum().sort_values('Total_Sales', ascending=False)
plt.figure(figsize=(8,4)); plt.bar(cat_sales['Product_Category'], cat_sales['Total_Sales']); plt.xticks(rotation=45)
plt.title('Top 10 Product Categories by Sales'); plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR,'plot_category_sales.png')); plt.close()

# 3. Region-wise sales
region_sales = df.groupby('Region', as_index=False)['Total_Sales'].sum()
plt.figure(figsize=(6,4)); plt.bar(region_sales['Region'], region_sales['Total_Sales']); plt.title('Region-wise Sales'); pl
plt.savefig(os.path.join(OUTPUT_DIR,'plot_region_sales.png')); plt.close()

# 4. Promotion vs Non-promotion sales
promo = df.copy()
promo['Promotion_Applied'] = promo['Promotion_Applied'].astype(str).str.lower().map({'yes':1,'no':0}).fillna(0).astype(int)
promo_sales = promo.groupby('Promotion_Applied')['Total_Sales'].sum().reset_index()
plt.figure(figsize=(5,4)); plt.bar(['No','Yes'], promo_sales['Total_Sales']); plt.title('Promotion vs Non-Promotion Sales')
plt.savefig(os.path.join(OUTPUT_DIR,'plot_promo_sales.png')); plt.close()

# 5. Discount vs Units_Sold scatter (sample)
plt.figure(figsize=(6,4));
sample = df.sample(n=min(2000,len(df)), random_state=42)
plt.scatter(sample['Discount_Percentage'], sample['Units_Sold'], alpha=0.3, s=10)
plt.xlabel('Discount %'); plt.ylabel('Units Sold'); plt.title('Discount % vs Units Sold (sample)'); plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR,'plot_discount_units.png')); plt.close()

print('EDA plots saved to', OUTPUT_DIR)
monthly_agg.head()
```

```
EDA plots saved to /content/drive/MyDrive/Retail_Capstone_Project/capstone_outputs
```

	Month	Daily_Sales
0	2023-01	2.001128e+09
1	2023-02	1.748971e+09
2	2023-03	1.954974e+09
3	2023-04	1.865035e+09
4	2023-05	1.901729e+09

Next steps: Generate code with `monthly_agg` New interactive sheet

```
# Forecasting: create lag features on daily aggregate
daily = daily.set_index('Date').asfreq('D').fillna(0).reset_index()
daily['y'] = daily['Daily_Sales']

# create lag features
for lag in [1,7,14,30]:
    daily[f'lag_{lag}'] = daily['y'].shift(lag).fillna(0)

# rolling means
daily['rmean_7'] = daily['y'].rolling(7,min_periods=1).mean().shift(1).fillna(0)
daily['rmean_30'] = daily['y'].rolling(30,min_periods=1).mean().shift(1).fillna(0)

# add day of week, month
daily['dow'] = daily['Date'].dt.dayofweek
daily['month'] = daily['Date'].dt.month

daily = daily.fillna(0)
daily.tail()
```

	Date	Daily_Sales	y	lag_1	lag_7	lag_14	lag_30	rmean_7	rmean_30	dow	month
726	2024-12-27	49543573.80	49543573.80	52014381.04	55721199.38	61127785.55	54323428.56	6.452705e+07	6.268173e+07	4	12
727	2024-12-28	70989430.94	70989430.94	49543573.80	64312383.92	58621313.06	58257104.87	6.364453e+07	6.252241e+07	5	12
728	2024-12-29	71025307.80	71025307.80	70989430.94	73074639.51	80097675.82	60277106.44	6.459840e+07	6.294682e+07	6	12

```
# Train/test split: last 60 days for testing, rest for training
train = daily.iloc[:-60].copy()
test = daily.iloc[-60:].copy()

FEATURES = [col for col in train.columns if col not in ['Date','y']]
X_train, y_train = train[FEATURES], train['y']
X_test, y_test = test[FEATURES], test['y']

rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=2)
rf.fit(X_train, y_train)
pred_test = rf.predict(X_test)

mae = mean_absolute_error(y_test, pred_test)
mse = mean_squared_error(y_test, pred_test)
rmse = np.sqrt(mse)
print(f'Test MAE: {mae:.2f}, RMSE: {rmse:.2f}')


# Save model
model_path = os.path.join(OUTPUT_DIR, 'rf_daily_sales_model.pkl')
with open(model_path,'wb') as f:
    pickle.dump(rf, f)
print('Saved model to', model_path)
```

```
Test MAE: 39072.34, RMSE: 59152.09
Saved model to /content/drive/MyDrive/Retail_Capstone_Project/capstone_outputs/rf_daily_sales_model.pkl
```

```
# Forecast next 30 days using iterative strategy (fixed for pandas without .append)
# We'll iteratively predict one day ahead, build features from the growing daily frame,
# and append new predictions using pd.concat.

# Make a working copy of daily (which already has required lag/rolling columns)
df_future = daily.copy().reset_index(drop=True)

# Ensure 'y' exists and Date is datetime
df_future['Date'] = pd.to_datetime(df_future['Date'])
df_future = df_future.sort_values('Date').reset_index(drop=True)

# Features list used for model
FEATURES = [col for col in df_future.columns if col not in ['Date','y']]

# Iteratively predict next 30 days
n_days = 30
preds = []

for i in range(n_days):
    last_date = df_future['Date'].iloc[-1]
    next_date = last_date + pd.Timedelta(days=1)

    # Build feature dict for the next_date using df_future values
    feat = {}
```

```
    # lag features
    for lag in [1,7,14,30]:
        if len(df_future) >= lag:
            feat[f'lag_{lag}'] = df_future['y'].iloc[-lag]
        else:
            feat[f'lag_{lag}'] = df_future['y'].iloc[-1]  # fallback to last available

    # rolling means (use previous window)
    if len(df_future) >= 1:
        feat['rmean_7'] = df_future['y'].iloc[-7:].mean() if len(df_future) >= 7 else df_future['y'].mean()
        feat['rmean_30'] = df_future['y'].iloc[-30:].mean() if len(df_future) >= 30 else df_future['y'].mean()
    else:
        feat['rmean_7'] = 0
        feat['rmean_30'] = 0

    feat['dow'] = next_date.dayofweek
    feat['month'] = next_date.month

    # Build DataFrame with same order as FEATURES
    X_cur = pd.DataFrame([{c: feat.get(c, 0) for c in FEATURES}])

    # Predict
    y_pred = rf.predict(X_cur)[0]

    # Create new row to append to df_future (must match columns)
    new_row = {'Date': next_date, 'y': y_pred}
    # add feature columns so df_future keeps consistent structure
    for c in FEATURES:
        new_row[c] = X_cur.iloc[0].get(c, 0)

    new_row_df = pd.DataFrame([new_row])
    df_future = pd.concat([df_future, new_row_df], ignore_index=True)

    preds.append((next_date, y_pred))

# Save forecast (last n_days)
future_forecast = pd.DataFrame(preds, columns=['Date','forecast'])
future_forecast['Date'] = pd.to_datetime(future_forecast['Date'])
future_forecast.to_csv(os.path.join(OUTPUT_DIR,'forecast_next_30_days.csv'), index=False)
print('Forecast saved to', os.path.join(OUTPUT_DIR,'forecast_next_30_days.csv'))
future_forecast.head()
```

```
Forecast saved to /content/drive/MyDrive/Retail_Capstone_Project/capstone_outputs/forecast_next_30_days.csv
```

	Date	forecast
0	2025-01-01	3.855773e+07
1	2025-01-02	3.890735e+07
2	2025-01-03	3.915980e+07
3	2025-01-04	3.886361e+07
4	2025-01-05	3.838019e+07

Next steps: Generate code with `future_forecast` New interactive sheet

```
# Simple store segmentation
store_agg = df.groupby('Store_ID').agg(Avg_Sales=('Total_Sales','mean'),
                                       Avg_Discount=('Discount_Percentage','mean'),
                                       Avg_Stock=('Stock_On_Hand','mean')).reset_index().fillna(0)
X_cluster = store_agg[['Avg_Sales','Avg_Discount','Avg_Stock']].copy()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)

kmeans = KMeans(n_clusters=3, random_state=42)
store_agg['cluster'] = kmeans.fit_predict(X_scaled)

store_agg.to_csv(os.path.join(OUTPUT_DIR,'store_segmentation.csv'), index=False)
print('Store segmentation saved to', os.path.join(OUTPUT_DIR,'store_segmentation.csv'))
store_agg.head()
```

```
Store segmentation saved to /content/drive/MyDrive/Retail_Capstone_Project/capstone_outputs/store_segmentation.csv
```

	Store_ID	Avg_Sales	Avg_Discount	Avg_Stock	cluster
0	STR_101	636296.448477	10.091824	274.836486	2
1	STR_102	627656.097619	9.998365	277.107151	0
2	STR_103	632622.548469	9.968931	275.589883	0
3	STR_104	625933.088165	10.113760	273.573155	2
4	STR_105	619965.442792	10.076724	273.825805	1

Next steps: Generate code with store_agg | New interactive sheet

```
# Create a minimal Streamlit app skeleton (not executed here)
app_code = '''import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
st.set_page_config(layout='wide', page_title='Retail Analytics - Minimal')

st.title('Retail Analytics & Forecasting - Minimal Submission')

@st.cache_data
def load_data():
    df = pd.read_csv('/mnt/data/capstone_outputs/processed_sales_minimal.csv')
    return df

df = load_data()

st.header('Overview')
col1, col2, col3 = st.columns(3)
col1.metric('Total Sales', f"{df['Total_Sales'].sum():,.0f}")
col2.metric('Total Units', f"{df['Units_Sold'].sum():,.0f}")
col3.metric('Unique Stores', df['Store_ID'].nunique())

st.header('Forecast (next 30 days)')
fc = pd.read_csv('/mnt/data/capstone_outputs/forecast_next_30_days.csv')
st.line_chart(fc.set_index('Date')['forecast'])

st.header('Store Segmentation Sample')
seg = pd.read_csv('/mnt/data/capstone_outputs/store_segmentation.csv')
st.dataframe(seg.head(20))
'''

os.makedirs('/mnt/data/capstone_outputs', exist_ok=True)
with open('/mnt/data/dashboard_app_minimal.py','w') as f:
    f.write(app_code)
print('Streamlit app skeleton saved to /mnt/data/dashboard_app_minimal.py')
```

```
Streamlit app skeleton saved to /mnt/data/dashboard_app_minimal.py
```

```
# List outputs and show quick samples
import os
out = OUTPUT_DIR
print('Files in output dir:')
print(os.listdir(out))
print('\nForecast sample:')
print(pd.read_csv(os.path.join(out,'forecast_next_30_days.csv')).head())
print('\nStore segmentation sample:')
print(pd.read_csv(os.path.join(out,'store_segmentation.csv')).head())
```

```
Files in output dir:
['processed_sales_minimal.csv', 'plot_monthly_sales.png', 'plot_category_sales.png', 'plot_region_sales.png', 'plot_promo_sa

Forecast sample:
         Date      forecast
0  2025-01-01  3.855773e+07
1  2025-01-02  3.890735e+07
2  2025-01-03  3.915980e+07
3  2025-01-04  3.886361e+07
4  2025-01-05  3.838019e+07

Store segmentation sample:
  Store_ID      Avg_Sales  Avg_Discount   Avg_Stock  cluster
0  STR_101  636296.448477     10.091824  274.836486        2
1  STR_102  627656.097619      9.998365  277.107151        0
2  STR_103  632622.548469      9.968931  275.589883        0
3  STR_104  625933.088165     10.113760  273.573155        2
4  STR_105  619965.442792     10.076724  273.825805        1
```