```
#%% Rayden Dodd Term Project
import pandas as pd
from pandas.plotting import scatter_matrix
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from matplotlib.ticker import FuncFormatter
from matplotlib.ticker import PercentFormatter
import seaborn as sns
from scipy import stats
from tabulate import tabulate
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, PowerTransformer
#%% Set Globabl values
np.set_printoptions(precision=2, floatmode='fixed')
title_font = {'family': 'serif', 'color': 'blue', 'size': 18}
label_font = {'family': 'serif', 'color': 'darkred', 'size': 14}
fmt_millions=FuncFormatter(lambda x,_:f'{x/1e6:.2f}M')
fmt_thousands=FuncFormatter(lambda x,_:f'{x/1e3:.2f}K')
fmt_billions=FuncFormatter(lambda x,_:f'{x/1e9:.2f}B')
periods = ['AM Peak (Open-9:30am)',
           'Midday (9:30am-3pm)',
           'PM Peak (3pm-7pm)',
           'Evening (7pm-12am)',
           'Late Night (12am-Close)']

#%% Upload dataset
dataset = 'Entries_by_Year_Full_Data_data/Entries_by_Year_Full_Data_data.csv'
mlb_schedule_dataset = 'Entries_by_Year_Full_Data_data/mlb-2024-orig.csv'

df = pd.read_csv(dataset)
mlb_schedule = pd.read_csv(mlb_schedule_dataset)
print(tabulate(df.head(10), headers='keys', tablefmt='pretty'))
print(mlb_schedule.head().to_string())
#%% Clean Data set
columns_to_remove = ["Avg Daily Tapped Entries", "SUM([NonTapped
Entries])/COUNTD([Date])", "Avg Daily Entries", "Avg Daily Entries (copy)", "Avg
Daily Entries Non-Rounded", "Avg Daily Exits Rounded", "Avg Daily NonTapped
Entries", "Data Range", "DAY_TYPE", "Tableau X 98", "Tableau Y 98", "Year", "Year
(Discrete)"]
df = df.drop(columns=[col for col in columns_to_remove if col in df.columns])
#%% Handle Missing Data
print("\nMissing values BEFORE cleaning:")
print(df.isna().sum())

# Strategy: Drop rows with any NaNs (common in observational datasets to preserve
integrity)
df_cleaned = df.dropna()

print("\nMissing values AFTER cleaning:")
print(df_cleaned.isna().sum())
# Holiday to True or False
df['Holiday'] = df['Holiday'].map({'Yes': True, 'No': False})
# Clean up dates
df["Date"]      = pd.to_datetime(df["Date"], format='%m/%d/%Y %I:%M:%S %p')
df.rename(columns={'Year of Date': 'Year'}, inplace=True)
df["Year"]        = df["Date"].dt.year
df["Month"]       = df["Date"].dt.month_name()
df["Day of Week"] = df["Date"].dt.day_name()
```

```
df['NonTapped Exits'] = (df['Exits'] - df['Tap Exits']).clip(lower=0)
# Clean up negative entries
cols_to_clip = ['Entries',
        'NonTapped Entries',
        'Tap Entries',
        'Exits',
        'Tap Exits',
        'NonTapped Exits']

df[cols_to_clip] = df[cols_to_clip].clip(lower=0)
# Print first 10 rows as a pretty table
print(tabulate(df.head(10), headers='keys', tablefmt='pretty'))
# Display summary statistics
summary_cols = [
    'Entries',
    'NonTapped Entries',
    'Tap Entries',
    'Exits',
    'Paid Entries',
    'Tap Exits',
    'NonTapped Exits'
]

# Summary statistics
print("\nSummary statistics for selected columns:")
print(tabulate(df[summary_cols].describe().T, headers='keys', tablefmt='pretty',
floatfmt=".2f"))

#%% PCA
# Select relevant numeric features
numeric_cols = ['Entries', 'NonTapped Entries', 'Tap Entries', 'Exits',
                'Hour', 'Paid Entries', 'Tap Exits', 'NonTapped Exits']
df_numeric = df[numeric_cols].dropna()

# Standardize data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_numeric)

# PCA analysis
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
explained_variance = pca.explained_variance_ratio_
singular_values = pca.singular_values_
condition_number = np.linalg.cond(X_scaled)
print("\nExplained Variance Ratio by Component:")
print(explained_variance)
print("\nSingular Values:")
print(singular_values)
print(f"\nCondition Number of Scaled Matrix: {condition_number:.2f}")

#%%  Line plot Total System Entries over the years
def plot_line_yearly_entries(df_temp):
    yearly = df_temp.groupby("Year")["Entries"].sum()
    ax = yearly.plot.line(marker="o", linewidth=2)

    ax.set_title("Year-over-Year Metrorail Entries", fontdict=title_font)
    ax.set_xlabel("Year", fontdict=label_font)
    ax.set_ylabel("Entries (millions)", fontdict=label_font)
    ax.yaxis.set_major_formatter(
```

```
        mtick.FuncFormatter(lambda x, pos: f'{x / 1_000_000:.1f} M')
    )
    ax.grid(True)
    plt.tight_layout()
    plt.show()

def plot_scatter_per_day(df_temp):
    daily = (df.groupby(df["Date"].dt.date)["Entries"].sum().sort_index())

    fig, ax = plt.subplots(figsize=(18, 6))
    sns.scatterplot(x=daily.index, y=daily.values, ax=ax, lw=2)

    ax.set_title("Daily Metrorail Ridership (Entries)", fontdict=title_font)
    ax.set_xlabel("Date", fontdict=label_font)
    ax.set_ylabel("Entries (Millions)", fontdict=label_font)
    ax.yaxis.set_major_formatter(fmt_thousands)
    ax.grid(True, alpha=0.4)

    plt.tight_layout()
    plt.show()

def line_pct_unpaid_daily(df_temp):
    sub = df.loc[df['Date'] >= '2023-01-01', ['Date', 'NonTapped Entries', 'Tap
Entries', 'Entries']].copy()
    daily = (sub.groupby(sub['Date'].dt.date)[['NonTapped Entries', 'Tap Entries',
'Entries']].sum().assign(pct=lambda x: x['NonTapped Entries'] / (x['Entries'])))
    ax = daily['pct'].plot(figsize=(12,4), lw=2, label='% unpaid')
    ax.set_title('Daily Share of Un-Paid (Non-Tapped) Entries — 2023-present',
fontdict=title_font)
    ax.set_xlabel('Date', fontdict=label_font)
    ax.set_ylabel('Un-Paid Share', fontdict=label_font)
    ax.yaxis.set_major_formatter(PercentFormatter(1.0))
    ax.grid(True)
    ax.legend()
    plt.tight_layout()
    plt.show()

def plot_station_hourly_flow(df_temp):
    top_stations = df_temp.groupby("Station Name")
["Entries"].sum().nlargest(6).index
    df_top = df_temp[df_temp["Station Name"].isin(top_stations)]

    hour_labels = [(str(h % 12 or 12) + " " + ("AM" if h < 12 else "PM")) for h in
range(24)]

    fig, axes = plt.subplots(3, 2, figsize=(16, 10), sharex=True, sharey=True)
    axes = axes.flatten()

    for i, station in enumerate(top_stations):
        ax = axes[i]
        station_data = df_top[df_top["Station Name"] == station]
        hourly_avg = station_data.groupby("Hour")[["Entries",
"Exits"]].mean().reindex(range(24), fill_value=0)

        ax.plot(hourly_avg.index, hourly_avg["Entries"], label="Entries",
color="blue", marker="o")
        ax.plot(hourly_avg.index, hourly_avg["Exits"], label="Exits", color="red",
marker="o")
        ax.set_title(f"{station}", fontdict=title_font)
```

```
        ax.set_xlabel("Hour", fontdict=label_font)
        ax.set_ylabel("Average Count", fontdict=label_font)
        ax.set_xticks(range(24))
        ax.set_xticklabels(hour_labels, rotation=45, fontsize=9)
        ax.grid(True, alpha=0.3)

        if i % 2 == 0:
            ax.set_ylabel("Average Count")
        if i >= 2:
            ax.set_xlabel("Hour")

    fig.suptitle("Average Hourly Flow – Top 4 Stations",
color='blue',family='serif', fontsize=16, fontweight='bold')
    fig.tight_layout(rect=[0, 0, 1, 0.95])
    plt.legend(loc="upper right")
    plt.show()

def plot_navy_yard_2024_with_games(df_temp, games_df):
    navy = df_temp[
        (df_temp['Station Name'] == "Navy Yard-Ballpark") &
        (df_temp['Year'] == 2024)
    ]

    daily = navy.groupby('Date')['Entries'].sum().reset_index()

    home_games = games_df[games_df['Home'] == 'Washington Nationals'].copy()
    home_games['Date'] = pd.to_datetime(home_games['Date'])

    plt.figure(figsize=(16, 6))
    plt.plot(daily['Date'], daily['Entries'], label='Daily Entries', linewidth=2)
    plt.scatter(home_games['Date'],
                daily.set_index('Date').reindex(home_games['Date'])['Entries'],
                color='red', label='Nationals Home Games', zorder=5)

    plt.title("Navy Yard-Ballpark Daily Entries – 2024\nWith Nationals Home Games
Highlighted", fontdict=title_font)
    plt.xlabel("Date", fontdict=label_font)
    plt.ylabel("Entries", fontdict=label_font)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.show()


def plot_game_day_hourly(df_temp, games_df):
    home_games = games_df[games_df['Home'] == 'Washington Nationals'].copy()
    home_games['Date'] = pd.to_datetime(home_games['Date'])
    first_game = home_games.iloc[0]

    game_date = first_game['Date']
    game_start = pd.to_datetime(first_game['Start Time (EDT)'])
    game_end = game_start + pd.Timedelta(hours=3)

    day_df = df_temp[
        (df_temp['Station Name'] == 'Navy Yard-Ballpark') &
        (df_temp['Date'].dt.date == game_date.date())
    ]

    hourly = day_df.groupby('Hour')[['Entries', 'Exits']].sum().reset_index()
```

```
    hour_labels = pd.to_datetime(hourly['Hour'], format='%H').dt.strftime('%I:00
%p')

    fig, axs = plt.subplots(1, 2, figsize=(16, 6), sharex=True)
    axs[0].bar(hour_labels, hourly['Entries'], color='blue')
    axs[1].bar(hour_labels, hourly['Exits'], color='red')

    for ax, metric in zip(axs, ['Entries', 'Exits']):
        ax.set_title(f'{metric} by Hour – {game_date.date()}', fontdict=title_font)
        ax.set_xlabel('Hour', fontdict=label_font)
        ax.set_ylabel(metric, fontdict=label_font)
        ax.axvspan(game_start.hour, game_end.hour, color='black', alpha=0.4,
label='Game Time')
        ax.legend()
        ax.tick_params(axis='x', rotation=45)
        ax.grid(axis='y', alpha=0.3)

    plt.suptitle(f"Navy Yard-Ballpark – First Nationals Home Game
{game_date.date()}", fontsize=20, family='serif', fontweight='bold', color='blue')
    plt.tight_layout()
    plt.show()

def plot_game_day_hourly_6(df_temp, games_df):
    games = games_df[games_df['Home'] == 'Washington Nationals'].copy()
    games['Date'] = pd.to_datetime(games['Date'], format="mixed", errors="coerce")
    games['Start Time (EDT)'] = pd.to_datetime(games['Start Time (EDT)'],
format="mixed", errors="coerce")

    first_six = games.head(6)
    fig, axs = plt.subplots(2, 3, figsize=(20, 12), sharex=True, sharey=True)
    axs = axs.flatten()

    full_hours   = pd.RangeIndex(24)
    ampm_labels  = [pd.Timestamp(h, unit='h').strftime('%I %p').lstrip('0')
                    for h in full_hours]

    for i, (_, game) in enumerate(first_six.iterrows()):
        game_date = game['Date'].date()
        start_h   = game['Start Time (EDT)'].hour
        end_h     = (game['Start Time (EDT)'] + pd.Timedelta(hours=3)).hour

        day_df = df_temp[(df_temp['Station Name'] == 'Navy Yard-Ballpark') &
                    (df_temp['Date'].dt.date == game_date)]

        hourly = (day_df.groupby('Hour')[['Entries', 'Exits']]
                        .sum()
                        .reindex(full_hours, fill_value=0)
                        .reset_index())

        ax = axs[i]
        ax.bar(full_hours-0.2, hourly['Entries'], width=0.4,
               color='steelblue', label='Entries')
        ax.bar(full_hours+0.2, hourly['Exits'],   width=0.4,
               color='indianred', label='Exits')
        ax.axvspan(start_h, end_h, color='black', alpha=.25,
                   label='Game Window')

        ax.set_title(f"{game_date:%b %d} vs {game['Visitor']}",
```

```
                    fontsize=24, family='serif', color='blue')
        ax.set_xticks(full_hours)
        ax.set_xticklabels(ampm_labels, rotation=45, ha='right',
                           fontsize=10, family='serif', color='black')
        ax.tick_params(labelbottom=True)
        ax.set_ylabel('Count',
                      fontsize=24, family='serif', color='darkred')
        ax.grid(axis='y', alpha=.3)
        if i == 0: ax.legend()

    # delete unused axes
    for j in range(len(first_six), 6):
        fig.delaxes(axs[j])

    # global x-axis label
    fig.text(0.5, 0.03, 'Hour of Day (AM / PM)',
             ha='center', fontsize=24, family='serif', color='darkred')

    fig.suptitle("Navy Yard–Ballpark – Entries & Exits on First 6 Nationals Home
Games (2024)",
                 fontsize=24, family='serif', color='blue', y=0.96)

    plt.tight_layout(rect=[0, 0.05, 1, 0.93])
    plt.show()
#%% Line Plot – Avg Daily Metro Entries by Month
def plot_monthly_ridership_by_year(df_temp):
    daily = df_temp.groupby(['Date', 'Year', 'Month'])
['Entries'].sum().reset_index()
    monthly_avg = (
        daily.groupby(['Year', 'Month'])['Entries']
        .mean()
        .reset_index()
    )

    pivot = monthly_avg.pivot(index='Month', columns='Year', values='Entries')
    month_order = ['January', 'February', 'March', 'April', 'May', 'June',
                   'July', 'August', 'September', 'October', 'November',
'December']
    pivot = pivot.reindex(month_order)
    plt.figure(figsize=(24, 10))
    colors = plt.cm.tab20(np.linspace(0, 1, pivot.shape[1]))
    pivot.plot(marker='o', color=colors)

    plt.title("Average Daily Metro Entries by Month", fontdict=title_font)
    plt.xlabel("Month", fontdict=label_font)
    plt.ylabel("Avg Daily Entries", fontdict=label_font)

    month_labels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                    'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    plt.xticks(ticks=range(0, 12), labels=month_labels)

    plt.grid(True, alpha=0.4)
    plt.legend(
        title="Year",
        bbox_to_anchor=(1.02, 1),
        loc="upper left",
        borderaxespad=0
    )
    plt.tight_layout()
```

```
    plt.subplots_adjust(right=0.85)
    plt.show()
#%% Bar Charts
def bar_stack_tap_nontap_year(df_temp):
    years=[2023,2024,2025]
    agg=df_temp[df['Year'].isin(years)].groupby('Year')[['NonTapped Entries','Tap
Entries']].sum()
    ax=agg[['NonTapped Entries','Tap
Entries']].plot(kind='bar',stacked=True,figsize=(8,4),width=0.8,color=['red','steel
blue'])
    ax.set_title('Non-Tapped vs Tap Entries — 2023-2025',fontdict=title_font)
    ax.set_xlabel('Year',fontdict=label_font)
    ax.set_ylabel('Count (Millions)',fontdict=label_font)
    ax.yaxis.set_major_formatter(fmt_millions)
    ax.legend(title='Entry Type')
    ax.grid(axis='y',alpha=0.6)
    plt.tight_layout(); plt.show()

def bar_group_entries_exits(df_temp):
    top5=(df_temp.groupby('Station Name')
[['Entries','Exits']].sum().sort_values('Entries',ascending=False).head(5))

ax=top5[['Entries','Exits']].plot(kind='bar',stacked=False,figsize=(9,4),width=0.8,
color=sns.color_palette('Set1',2))
    ax.set_title('Top-5 Stations – Entries vs Exits',fontdict=title_font)
    ax.set_ylim(60_000_000, 85_000_000)
    ax.set_xlabel('Station',fontdict=label_font)
    ax.set_ylabel('Count (Millions)',fontdict=label_font)
    ax.yaxis.set_major_formatter(fmt_millions)
    ax.legend(title='Type')
    ax.grid(axis='y',alpha=0.6)
    plt.tight_layout()
    plt.show()

def bar_total_ridership_per_station(df_temp):
    tot=df_temp.groupby('Station Name')
['Entries'].sum().sort_values(ascending=False)

ax=tot.plot(kind='bar',figsize=(25,10),color=sns.color_palette('crest',len(tot)))
    big_title = {**title_font, 'size': 35}
    big_label = {**label_font, 'size': 25}
    ax.set_title('Total Ridership per Station',fontdict=big_title)
    ax.set_xlabel('Station',fontdict=big_label)
    ax.set_ylabel('Entries (Millions)',fontdict=big_label)
    ax.yaxis.set_major_formatter(fmt_millions)
    ax.grid(axis='y',alpha=0.6)
    plt.tight_layout()
    plt.show()

def bar_avg_entries_exits_by_weekday(df_temp):
    order=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
    daily = df_temp.groupby(df_temp['Date'].dt.date)[['Entries', 'Exits']].sum()
    daily['Day of Week'] = pd.to_datetime(daily.index).day_name()
    group=daily.groupby('Day of Week')[['Entries', 'Exits']].mean().reindex(order)

ax=group[['Entries','Exits']].plot(kind='bar',figsize=(9,4),width=0.8,color=sns.col
or_palette('Set1',2))
    ax.set_title('Average Entries and Exits per Weekday',fontdict=title_font)
    ax.set_xlabel('Day of Week',fontdict=label_font)
```

```
    ax.set_ylim(150_000, 550_000)
    ax.set_ylabel('Average Count (Millions)',fontdict=label_font)
    ax.yaxis.set_major_formatter(fmt_thousands)
    ax.legend(title='Type')
    ax.grid(axis='y',alpha=0.6)
    plt.tight_layout()
    plt.show()

def bar_total_entries_exits_by_hour_group(df_temp):
    group=df_temp.groupby('Hour')[['Entries','Exits']].sum().sort_index()

ax=group[['Entries','Exits']].plot(kind='bar',figsize=(10,4),width=0.8,color=sns.co
lor_palette('husl',2))
    ax.set_title('Total Entries and Exits per Hour',fontdict=title_font)
    hour_labels = [f'{(h % 12 or 12)}:00 {"am" if h < 12 else "pm"}' for h in
group.index]
    ax.set_xticklabels(hour_labels, rotation=45, ha='right')
    ax.set_xlabel('Hour of Day',fontdict=label_font)
    ax.set_ylabel('Count (Millions)',fontdict=label_font)
    ax.yaxis.set_major_formatter(fmt_millions)
    ax.legend(title='Type')
    ax.grid(axis='y',alpha=0.6)
    plt.tight_layout()
    plt.show()

def stacked_entries_exits_timeperiod(df_temp):
    e=df_temp.pivot_table(values='Entries',index='Station Name',
                          columns='Time Period',aggfunc='sum')[periods]
    x=df_temp.pivot_table(values='Exits',  index='Station Name',
                          columns='Time Period',aggfunc='sum')[periods]
    order = (e.sum(axis=1) + x.sum(axis=1)).sort_values(ascending=False).index
    e = e.loc[order]  # Entries now in the same station order …
    x = x.loc[order]
    blues=sns.color_palette('Blues_r',n_colors=len(periods))
    reds =sns.color_palette('Reds_r', n_colors=len(periods))
    fig,ax=plt.subplots(figsize=(25,10))
    bar_w=0.35
    xs=np.arange(len(order))
    bottom_E=np.zeros(len(order))
    bottom_X=np.zeros(len(order))
    for i,p in enumerate(periods):
        ax.bar(xs-bar_w/2,e[p],bottom=bottom_E,width=bar_w,color=blues[i])
        bottom_E+=e[p]
        ax.bar(xs+bar_w/2,x[p],bottom=bottom_X,width=bar_w,color=reds[i])
        bottom_X+=x[p]
    ax.set_xticks(xs); ax.set_xticklabels(order,rotation=45,ha='right')
    big_title={**title_font,'size':35}
    big_label={**label_font,'size':25}
    ax.set_title('Entries vs Exits by Time Period per Station',fontdict=big_title)
    ax.set_xlabel('Station',fontdict=big_label)
    ax.set_ylabel('Count (Millions)',fontdict=big_label)
    ax.yaxis.set_major_formatter(fmt_millions)
    ax.grid(axis='y',alpha=0.6)
    handles=[plt.Rectangle((0,0),1,1,color=c) for c in blues+reds]
    labels=[f'E: {p}' for p in periods]+[f'X: {p}' for p in periods]
    ax.legend(handles,labels,title='Period /
Type',ncol=2,fontsize=16,title_fontsize=18)
    plt.tight_layout()
    plt.show()
```

```
#%% Box Plot – Raw Entries by Service Group and Holiday
def box_by_service_and_holiday(df_temp):
    valid = ['Weekday', 'Saturday', 'Sunday']
    df_filtered = df_temp[df_temp['Service Type'].isin(valid)].copy()
    df_filtered['Service Group'] = df_filtered['Service Type'].map(
        lambda x: 'Weekend' if x in ['Saturday', 'Sunday'] else 'Weekday'
    )
    df_filtered['HolidayFlag'] = df_filtered['Holiday'].map({True: 'Holiday',
False: 'Normal'})
    fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

    sns.boxplot(data=df_filtered, x='Service Group', y='Entries', order=['Weekday',
'Weekend'], ax=axes[0])
    axes[0].set_title('Entries – Weekday vs Weekend', fontdict=title_font)
    axes[0].set_xlabel('Day Type', fontdict=label_font)
    axes[0].set_ylabel('Entries', fontdict=label_font)
    axes[0].grid(True, alpha=0.4)

    sns.boxplot(data=df_filtered, x='HolidayFlag', y='Entries', order=['Normal',
'Holiday'], ax=axes[1])
    axes[1].set_title('Entries – Normal vs Holiday', fontdict=title_font)
    axes[1].set_xlabel('Day Type', fontdict=label_font)
    axes[1].grid(True, alpha=0.4)

    plt.tight_layout()
    plt.show()

def bar_late_night(df_temp):
    late_night_df = df_temp[df_temp["Time Period"] == "Late Night (12am-Close)"]

    top_entries = late_night_df.groupby("Station Name")
["Entries"].sum().nlargest(10)
    top_exits = late_night_df.groupby("Station Name")["Exits"].sum().nlargest(10)

    fig, axes = plt.subplots(1, 2, figsize=(25, 10), sharex=False)

    top_entries.sort_values().plot.barh(
        ax=axes[0],
        color=sns.color_palette("Blues_r", len(top_entries))[::-1]
    )
    axes[0].set_title("Top 10 Stations – Late Night Entries",
fontdict={**title_font, 'size': 30})
    axes[0].set_xlabel("Entries", fontdict={**label_font, 'size': 20})
    axes[0].set_ylabel("Station", fontdict={**label_font, 'size': 20})
    axes[0].xaxis.set_major_formatter(fmt_millions)
    axes[0].grid(axis='x', alpha=0.6)

    top_exits.sort_values().plot.barh(
        ax=axes[1],
        color=sns.color_palette("Reds_r", len(top_exits))[::-1]
    )
    axes[1].set_title("Top 10 Stations – Late Night Exits", fontdict={**title_font,
'size': 30})
    axes[1].set_xlabel("Exits", fontdict={**label_font, 'size': 20})
    axes[1].set_ylabel("Station", fontdict={**label_font, 'size': 20})
    axes[1].xaxis.set_major_formatter(fmt_millions)
    axes[1].grid(axis='x', alpha=0.6)

    plt.tight_layout()
```

```
    plt.show()
#%% COUNT plot
def plot_count_timeperiod(df_temp):
    ax = sns.countplot(data=df, x='Time Period', order=periods, color='steelblue')
    plt.title('Record Count by Time Period', fontdict=title_font)
    ax.yaxis.set_major_formatter(fmt_millions)
    plt.xlabel('Time Period', fontdict=label_font)
    plt.ylabel('Row Count', fontdict=label_font)
    plt.tick_params(axis='x', rotation=25)
    plt.grid(axis='y', alpha=.6)
    plt.tight_layout()
    plt.show()
#%% Pie Chart
def plot_pie_days(df_temp):
    days = df.groupby("Day of Week")["Entries"].sum()
    colors = sns.color_palette("pastel")[0:7]
    plt.figure(figsize=(6,6))
    explode = [0.05 for i in range(len(days))]
    plt.pie(days, labels=days.index, autopct="%1.2f%%", startangle=140,
            pctdistance=0.8, colors=colors, explode=explode)
    plt.title("Ridership Share by Day Of Week", fontdict=title_font)
    plt.tight_layout()
    plt.show()

def plot_pie_average_daily_per_station(df_temp):
    avg_daily_station = (
        df_temp.groupby("Station Name")["Entries"]
        .mean()
        .sort_values(ascending=False)
    )

    total = avg_daily_station.sum()
    percent = avg_daily_station / total * 100

    major = percent[percent > 0.30]
    other = percent[percent <= 0.30]

    combined = major.copy()
    if not other.empty:
        combined["Other"] = other.sum()

    labels = combined.index
    sizes = combined.values
    explode = [0.03] * len(labels)

    fig, ax = plt.subplots(figsize=(12, 12))
    wedges, texts, autotexts = ax.pie(
        sizes,
        labels=labels,
        explode=explode,
        autopct='%1.2f%%',
        startangle=140,
        textprops={'rotation_mode': 'anchor'}
    )
    for text, autotext, wedge in zip(texts, autotexts, wedges):
        angle = (wedge.theta2 + wedge.theta1) / 2
        for t in (text, autotext):
            t.set_rotation(angle)
            if 90 < angle < 270:
```

```
                t.set_rotation(angle + 180)
                t.set_horizontalalignment('right')
            else:
                t.set_horizontalalignment('left')

    ax.set_title("Average Daily Entries Share by Station", fontdict=title_font,
pad=85)
    plt.tight_layout()
    plt.show()

def plot_pie_days_by_time_period(df_temp):
    day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
    pivot = df_temp.pivot_table(
        index="Day of Week",
        columns="Time Period",
        values="Entries",
        aggfunc="sum"
    ).fillna(0).reindex(index=day_order, columns=periods)

    percentages = pivot.div(pivot.sum(axis=1), axis=0)

    fig, axes = plt.subplots(2, 4, figsize=(22, 12))
    axes = axes.flatten()

    for i, day in enumerate(percentages.index):
        data = percentages.loc[day]
        sizes = data.values
        labels = data.index

        explode = [0.1 if val == sizes.max() else 0.03 for val in sizes]

        wedges, texts, autotexts = axes[i].pie(
            sizes,
            labels=labels,
            autopct="%1.2f%%",
            startangle=140,
            explode=explode,
            textprops={'fontsize': 9}
        )

        axes[i].set_title(day, fontdict=title_font)

    for j in range(len(percentages.index), len(axes)):
        axes[j].axis('off')

    fig.suptitle("Time-of-Day Ridership Share by Day of Week", family='serif',
color='blue', fontsize=25)
    plt.tight_layout()
    plt.show()
#%% Dist Chart
def plot_dist_entries(df_temp):
    plt.figure(figsize=(24, 6))
    ax = sns.histplot(df['Entries'], kde=True, stat='density', bins=240)
    ax.set_title('Distribution of Entries', fontdict=title_font)
    ax.set_xlabel('Entries', fontdict=label_font)
    ax.set_ylabel('Density', fontdict=label_font)
    ax.set_ylim(0, 0.010)
    plt.tight_layout()
```

```
    plt.show()

#%% Pair plot
def plot_ridership_pairplot(df_temp):
    cols = ['Entries', 'NonTapped Entries', 'Tap Entries', 'Exits','Tap Exits',
'NonTapped Exits', 'Hour']
    big_title = dict(title_font)
    big_title.pop('fontsize', None)
    big_title['size'] = 22

    ax = sns.pairplot(df_temp[cols], corner=True,
                      diag_kind='kde',
                      plot_kws=dict(alpha=0.4, s=10))

    ax.fig.suptitle('Pairwise Relationships (Entries, Exits, Hour, …)',
family='serif', color='blue')
    ax.fig.subplots_adjust(top=0.93)
    plt.show()

#%% HeatMap plot
def plot_corr_heatmap(df_temp):
    cols = ['Entries', 'NonTapped Entries', 'Tap Entries', 'Exits', 'Tap Exits',
'NonTapped Exits', 'Hour']
    corr = df[cols].corr()
    sns.heatmap(corr, annot=True, cmap="coolwarm", cbar_kws={"label": "ρ"})
    plt.title("Correlation Matrix",fontdict=title_font)
    plt.tight_layout()
    plt.show()
#%% Hist plot + KDE
def plot_hist_kde(df_temp):
    sns.histplot(df_temp["Entries"], kde=True, bins=240, stat="density")
    plt.title("Entries Histogram with KDE overlay",fontdict=title_font)
    plt.tight_layout()
    plt.show()
def plot_log_hist_kde(df_temp):
    data = np.log10(df_temp['Entries'] + 1)
    plt.figure(figsize=(10,6))
    sns.histplot(data, kde=True, bins=240, stat='density')
    plt.title('Entries Distribution (logarithmic scale)', fontdict=title_font)
    plt.xlabel('log(Entries + 1)',fontdict=label_font)
    plt.ylabel('Density',fontdict=label_font)
    plt.grid(True, alpha=.3)
    plt.tight_layout()
    plt.show()
#%% QQ-plot
def plot_qq(df_temp):
    stats.probplot(df["Entries"], dist="norm", plot=plt)
    plt.title("QQ-plot of Entries",fontdict=title_font)
    plt.tight_layout()
    plt.show()

#%% FILLED KDE
def plot_kde_filled(df_temp):
    plt.figure(figsize=(24, 10))
    for s, sub in df_temp.groupby("Service Type"):
        sns.kdeplot(
            sub["Entries"],
            fill=True,
            alpha=0.6,
```

```
            linewidth=2,
            label=s,
            common_norm=False
        )
    plt.title("Entries KDE by Service Type", fontdict=title_font)
    plt.xlabel("Entries", fontdict=label_font)
    plt.ylabel("Density", fontdict=label_font)
    plt.legend()
    plt.tight_layout()
    plt.show()

#%% REG plot – Entries vs Exits
def plot_reg(df_temp):
    sns.regplot(
        x="NonTapped Entries",
        y="NonTapped Exits",
        data=df_temp.sample(5000),
        scatter_kws={"s": 8},
        line_kws={"lw": 2}
    )
    plt.title("NonTapped Entries vs NonTapped Exits with Regression Line",
fontdict=title_font)
    plt.xlabel("NonTapped Entries", fontdict=label_font)
    plt.ylabel("NonTapped Exits", fontdict=label_font)
    plt.tight_layout()
    plt.show()

#%% MULTIVARIATE BOX
def plot_multi_box(df_temp):
    plt.figure(figsize=(6, 5))
    sns.boxplot(data=df_temp, x="Day of Week", y="Entries")
    plt.yscale("log")
    plt.title("Box plot – Entries by Day of Week", fontdict=title_font)
    plt.xlabel("Day of Week", fontdict=label_font)
    plt.ylabel("Entries", fontdict=label_font)
    plt.grid(True)
    plt.tight_layout()
    plt.show()

#%% MULTIVARIATE BOXEN
def plot_multi_boxen(df_temp):
    plt.figure(figsize=(6, 5))
    sns.boxenplot(data=df_temp, x="Day of Week", y="Entries", palette="pastel")
    plt.yscale("log")
    plt.title("Boxen plot – Entries by Day of Week", fontdict=title_font)
    plt.xlabel("Day of Week", fontdict=label_font)
    plt.ylabel("Entries", fontdict=label_font)
    plt.grid(True)
    plt.tight_layout()
    plt.show()

#%% AREA plot – cumulative yearly ridership
def plot_area_cumulative(df_temp):
    yearly = df_temp.groupby("Year")["Entries"].sum().sort_index().cumsum()
    ax = yearly.plot.area(alpha=0.4)
    plt.title("Cumulative Entries Over Time", fontdict=title_font)
    ax.yaxis.set_major_formatter(fmt_billions)
    plt.xlabel("Year", fontdict=label_font)
    plt.ylabel("Cumulative Entries", fontdict=label_font)
```

```
    plt.grid(True)
    plt.tight_layout()
    plt.show()

#%% Violin Plot – Entries by Day of Week
def plot_violin(df_temp):
    sns.violinplot(
        data=df_temp,
        x="Day of Week",
        y="Entries",
        hue="Day of Week",
        palette="Set3",
        inner="quartile",
        legend=False
    )
    plt.yscale("log")
    plt.title("Entries Violin by Day of Week", fontdict=title_font)
    plt.xlabel("Day of Week", fontdict=label_font)
    plt.ylabel("Entries", fontdict=label_font)
    plt.tight_layout()
    plt.show()


#%% Joint Plot – Entries vs Tap Entries
def plot_joint(df_temp):
    ax = sns.jointplot(
        data=df_temp.sample(4000),
        x="Entries",
        y="Tap Entries",
        kind="kde",
        fill=True,
        alpha=0.4
    )
    big_label = dict(label_font)
    big_label.pop('size', None)
    big_label['fontsize'] = 16
    ax.ax_joint.set_xlabel("Entries", fontdict=big_label)
    ax.ax_joint.set_ylabel("Tap Entries", fontdict=big_label)
    ax.ax_joint.grid(True, alpha=0.6)
    ax.fig.set_size_inches(10,6)
    big_title = dict(title_font)
    big_title.pop('size', None)
    big_title['fontsize'] = 65
    ax.fig.suptitle("Joint KDE – Entries vs Tap Entries", y=0.98,
fontdict=big_title)
    ax.fig.subplots_adjust(top=0.93)
    plt.show()

#%% RUG plot – quick glance at Entries
def plot_rug(df_temp):
    sns.rugplot(df_temp["Entries"].sample(1000), height=0.05)
    plt.title("Entries Rug Plot", fontdict=title_font)
    plt.xlabel("Entries", fontdict=label_font)
    plt.xscale("log")
    plt.tight_layout()
    plt.show()

#%% 3-D SCATTER & CONTOUR (Entries by Year & Hour)
def plot_3d_contour(df_temp):
```

```
    fig = plt.figure(figsize=(6, 5))
    ax = fig.add_subplot(111, projection='3d')
    samp = df_temp.sample(3000)
    ax.scatter(samp["Year"], samp["Hour"], samp["Entries"], s=5)
    ax.set_title("3-D Scatter: Entries vs Year & Hour", fontdict=title_font)
    ax.set_xlabel("Year", fontdict=label_font)
    ax.set_ylabel("Hour", fontdict=label_font)
    ax.set_zlabel("Entries", fontdict=label_font)
    plt.tight_layout()
    plt.show()

    pivot = samp.pivot_table(values="Entries", index="Hour", columns="Year",
aggfunc="mean")
    sns.heatmap(pivot, cmap="viridis")
    plt.title("Average Entries – Contour (Hour × Year)", fontdict=title_font)
    plt.xlabel("Year", fontdict=label_font)
    plt.ylabel("Hour", fontdict=label_font)
    plt.tight_layout()
    plt.show()

#%% CLUSTER MAP – station similarity on traffic patterns
def plot_cluster(df_temp):
    matrix = (
        df_temp.pivot_table(index="Station Name", columns="Year", values="Entries",
aggfunc="sum")
        .fillna(0)
        .apply(lambda x: x / x.max(), axis=1)
    )
    ax = sns.clustermap(matrix, cmap="magma", figsize=(24, 30), yticklabels=False)
    big_title = dict(title_font)
    big_title.pop('size', None)
    big_title['fontsize'] = 28
    ax.fig.text(0.005, 1.0,
                "Ridership Pattern Cluster Map",
                fontdict=big_title, va='top')
    plt.show()

#%% HEXBIN – Entries vs Exits density
def plot_hexbin(df_temp):
    sub = df[['Entries', 'Exits']] + 1

    fig, ax = plt.subplots(figsize=(8, 6))
    hb = ax.hexbin(sub['Entries'], sub['Exits'],
                   gridsize=40,
                   cmap='viridis',
                   xscale='log', yscale='log',
                   mincnt=1)

    ax.set_title('Entries vs Exits – Hex Density log +1 shift',
fontdict=title_font)
    ax.set_xlabel('Entries + 1', fontdict=label_font)
    ax.set_ylabel('Exits + 1', fontdict=label_font)

    fig.colorbar(hb, ax=ax).set_label('Observations per hex')
    ax.grid(True, which='both',  alpha=.3)
    plt.tight_layout()
    plt.show()

#%% Strip Plot – Entries vs Time Period
```

```
def plot_strip(df_temp):
    plt.figure(figsize=(6, 5))
    sns.stripplot(data=df_temp, x="Time Period", y="Entries", alpha=0.4)
    plt.yscale("log")
    plt.title("Strip Plot – Entries by Time Period", fontdict=title_font)
    plt.xlabel("Time Period", fontdict=label_font)
    plt.ylabel("Entries", fontdict=label_font)
    plt.xticks(rotation=20, ha='right')
    plt.tight_layout()
    plt.show()

#%% Swarm Plot – Entries vs Time Period
def plot_swarm(df_temp):
    plt.figure(figsize=(6, 5))
    sns.swarmplot(data=df_temp.sample(2000), x="Time Period", y="Entries", size=3)
    plt.yscale("log")
    plt.title("Swarm Plot – Entries by Time Period", fontdict=title_font)
    plt.xlabel("Time Period", fontdict=label_font)
    plt.ylabel("Entries", fontdict=label_font)
    plt.xticks(rotation=20, ha='right')
    plt.tight_layout()
    plt.show()

#%% Plotting Static Plots
plot_line_yearly_entries(df)
plot_scatter_per_day(df)
line_pct_unpaid_daily(df)
plot_monthly_ridership_by_year(df)
plot_station_hourly_flow(df)
plot_navy_yard_2024_with_games(df,mlb_schedule)
plot_game_day_hourly(df,mlb_schedule)
plot_game_day_hourly_6(df,mlb_schedule)
box_by_service_and_holiday(df)
bar_stack_tap_nontap_year(df)
bar_group_entries_exits(df)
bar_total_ridership_per_station(df)
bar_avg_entries_exits_by_weekday(df)
bar_total_entries_exits_by_hour_group(df)
stacked_entries_exits_timeperiod(df)
bar_late_night(df)
#%%
plot_count_timeperiod(df)
#%%
plot_pie_days(df)
plot_pie_average_daily_per_station(df)
plot_pie_days_by_time_period(df)

#%%
plot_dist_entries(df)
#%%
plot_corr_heatmap(df)
#%%
plot_hist_kde(df)
plot_log_hist_kde(df)
#%%
plot_qq(df)
#%% KDE Filled
plot_kde_filled(df)
#%% Regression Plot
```

```
plot_reg(df)
#%% Box Plot
plot_multi_box(df)
#%% Boxen Plot
plot_multi_boxen(df)
#%% Area Plot
plot_area_cumulative(df)
#%% Violin Plot
plot_violin(df)
#%% Joint Plot
plot_joint(df)
#%% Rug Plot
plot_rug(df)
#%% 3D Scatter and Contour Plot
plot_3d_contour(df)
#%% Cluster Map
plot_cluster(df)
#%% Hexbin Plot
plot_hexbin(df)
#%% Plot Strip
plot_strip(df)
#%% Plot Swarm
plot_swarm(df)
#%%
plot_ridership_pairplot(df)
```