Start coding or generate with AI.

Project 2: Google PlayStore App Rating Prediction

Start coding or generate with AI.

Import libraries

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

Configurations

```
# Configurations
INPUT_CSV = "googleplaystore.csv"
OUTPUT_DIR = "output_googleplay"
CLEANED_CSV = os.path.join(OUTPUT_DIR, "cleaned_googleplay.csv")
REPORT_TXT = os.path.join(OUTPUT_DIR, "report_summary.txt")
os.makedirs(OUTPUT_DIR, exist_ok=True)
```

```
sns.set_theme(style="whitegrid")
```

Utility functions

```
# Utility functions
def safe_read_csv(path):
    """Try reading CSV robustly with common encodings and separators."""
    tries = [
        {"encoding": "utf-8", "sep": ","},
        {"encoding": "latin1", "sep": ","},
        {"encoding": "cp1252", "sep": ","},
        {"encoding": "utf-8", "sep": ";"},
    ]
    for t in tries:
        try:
            df = pd.read_csv(path, encoding=t["encoding"], sep=t["sep"], low_memory=False)
            return df
        except Exception:
            continue
    raise FileNotFoundError(f"Could not read file: {path}")
def clean_installs(x):
    if pd.isna(x): return np.nan
    s = str(x).replace(',', '').replace('+', '').strip()
    if s == '' or s.lower() == 'free': return np.nan
    try:
        return float(s)
    except:
        return np.nan
def clean_reviews(x):
    if pd.isna(x): return np.nan
    s = str(x).strip()
    # Sometimes "3.0M" or similar — handle M and k
    try:
        if s.endswith('M'):
            return float(s[:-1]) * 1_000_000
        if s.endswith('k'):
            return float(s[:-1]) * 1_000
        return float(s.replace(',', ''))
```

```
    except:
        try:
            return float(s)
        except:
            return np.nan

def clean_price(x):
    if pd.isna(x): return 0.0
    s = str(x).strip().replace('$','').replace(',', '')
    if s == '' or s.lower() in ['free', 'everyone']:
        return 0.0
    try:
        return float(s)
    except:
        return 0.0

def clean_size(x):
    if pd.isna(x): return np.nan
    s = str(x).strip()
    if s.lower() == "varies with device":
        return np.nan
    # e.g., "19M", "234k", "1.2M"
    try:
        if s.endswith('M'):
            return float(s[:-1]) * 1024  # convert MB to KB for consistent numeric (MB->KB)
        if s.endswith('k'):
            return float(s[:-1])  # already KB
        # sometimes "100000" etc
        return float(s)
    except:
        # try remove commas
        s2 = s.replace(',', '')
        try:
            return float(s2)
        except:
            return np.nan
```

Load Data

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

```
Loaded: 10841 rows, 13 columns
```

```
# Keep original copy
df_orig = df.copy()
```

Quick inspection

```
print("\nColumns:", df.columns.tolist())
print("\nFirst rows:")
display(df.head(5))
```

```
Columns: ['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating', 'Genres', 'Last Upd

First rows:
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	A
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	
	U Launcher Lite –												

Standardize column names (strip & replace spaces)

```
df.columns = [c.strip().replace(' ', '_') for c in df.columns]
```

Cleaning columns

```
# Convert Reviews, Installs, Price, Size to numeric
if 'Reviews' in df.columns:
    df['Reviews'] = df['Reviews'].apply(clean_reviews)

if 'Installs' in df.columns:
    df['Installs'] = df['Installs'].apply(clean_installs)

if 'Price' in df.columns:
    df['Price_USD'] = df['Price'].apply(clean_price)
else:
    df['Price_USD'] = 0.0

if 'Size' in df.columns:
    # we'll store size in KB numeric
    df['Size_KB'] = df['Size'].apply(clean_size)
else:
    df['Size_KB'] = np.nan

# Parse Last Updated -> datetime; also compute App Age (days since last update)
if 'Last_Updated' in df.columns:
    df['Last_Updated_dt'] = pd.to_datetime(df['Last_Updated'], errors='coerce')
else:
    df['Last_Updated_dt'] = pd.NaT

# Derive app_age_days (relative to today)
today = pd.to_datetime(datetime.now().date())
df['app_age_days'] = (today - df['Last_Updated_dt']).dt.days

# Ensure Rating is numeric
if 'Rating' in df.columns:
    df['Rating'] = pd.to_numeric(df['Rating'], errors='coerce')

# Standardize text columns
text_cols = ['App', 'Category', 'Type', 'Content_Rating', 'Genres', 'Current_Ver', 'Android_Ver']
for c in text_cols:
    if c in df.columns:
        df[c] = df[c].astype(str).str.strip()

# Drop exact duplicates
df.drop_duplicates(inplace=True)
```

Impute / handle missing ratings: fill with category median where possible

```
if 'Rating' in df.columns and 'Category' in df.columns:
    cat_median = df.groupby('Category')['Rating'].median()
    def fill_rating(row):
        if pd.isna(row['Rating']):
            m = cat_median.get(row['Category'], np.nan)
            if pd.isna(m):
                return df['Rating'].median()
            return m
        return row['Rating']
    df['Rating'] = df.apply(fill_rating, axis=1)
else:
    # if no category, fill with global median
    df['Rating'] = df['Rating'].fillna(df['Rating'].median())
```

Feature engineering

```
# Price buckets
def price_bucket(p):
    if p <= 0: return 'Free'
    if p <= 5: return 'Cheap (<=5$)'
    if p <= 20: return 'Mid (5-20$)'
    return 'Premium (>20$)'

df['Price_Bucket'] = df['Price_USD'].apply(price_bucket)

# Rating groups
def rating_group(r):
    if pd.isna(r): return 'Unknown'
    if r < 3.0: return 'Low (<3)'
    if r < 4.0: return 'Medium (3-4)'
    return 'High (>=4)'

df['Rating_Group'] = df['Rating'].apply(rating_group)

# Size categories in MB (convert KB to MB for category)
df['Size_MB'] = df['Size_KB'] / 1024.0
def size_bucket(mb):
    if pd.isna(mb): return 'Unknown'
    if mb < 10: return 'Small (<10MB)'
    if mb <= 50: return 'Medium (10-50MB)'
    return 'Large (>50MB)'

df['Size_Bucket'] = df['Size_MB'].apply(size_bucket)

# Apps per year updated
df['Last_Updated_Year'] = df['Last_Updated_dt'].dt.year

# Installs per MB metric (careful with zero or missing)
df['Installs_per_MB'] = df.apply(lambda r: (r['Installs'] / r['Size_MB']) if (pd.notna(r['Installs']) and pd.notna(r['Size_MB']) and r['S

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

```
Cleaned data saved -> output_googleplay\cleaned_googleplay.csv
```

Exploratory Analysis & Visualizations

```
def save_fig(fig, name):
    path = os.path.join(OUTPUT_DIR, name)
    fig.savefig(path, bbox_inches='tight', dpi=140)
    plt.close(fig)
    print("Saved:", path)

# 1) Count of apps by Price Bucket
fig, ax = plt.subplots(figsize=(8,5))
```

```
order = df['Price_Bucket'].value_counts().index
sns.countplot(data=df, x='Price_Bucket', order=order, ax=ax)
ax.set_title("App Count by Price Bucket")
ax.set_xlabel("")
for p in ax.patches:
    ax.annotate(int(p.get_height()), (p.get_x()+p.get_width()/2, p.get_height()), ha='center', va='bottom', fontsize=9)
save_fig(fig, "count_by_price_bucket.png")

# 2) Average Rating by Size Bucket (bar)
fig, ax = plt.subplots(figsize=(8,5))
size_order = ['Small (<10MB)', 'Medium (10-50MB)', 'Large (>50MB)', 'Unknown']
avg_rating_by_size = df.groupby('Size_Bucket')['Rating'].mean().reindex(size_order)
sns.barplot(x=avg_rating_by_size.index, y=avg_rating_by_size.values, ax=ax)
ax.set_title("Average Rating by App Size Category")
ax.set_ylabel("Average Rating")
ax.set_xlabel("")
save_fig(fig, "avg_rating_by_size_bucket.png")

# 3) Installs distribution by Rating Group (boxplot, log-scale on installs)
fig, ax = plt.subplots(figsize=(9,6))
plot_df = df[df['Installs'].notna() & df['Installs']>0].copy()
sns.boxplot(data=plot_df, x='Rating_Group', y='Installs', ax=ax)
ax.set_yscale('log')
ax.set_title("Installs by Rating Group (log scale)")
ax.set_ylabel("Installs (log scale)")
save_fig(fig, "installs_by_rating_group_boxplot.png")

# 4) Top 10 Categories by average installs (horizontal bar)
if 'Category' in df.columns:
    cat_avg_installs = df.groupby('Category')['Installs'].mean().dropna().sort_values(ascending=False).head(10)
    fig, ax = plt.subplots(figsize=(10,6))
    sns.barplot(x=cat_avg_installs.values, y=cat_avg_installs.index, ax=ax)
    ax.set_title("Top 10 Categories by Average Installs")
    ax.set_xlabel("Average Installs")
    save_fig(fig, "top10_categories_by_avg_installs.png")

# 5) Apps updated per year (trend)
years = df['Last_Updated_Year'].dropna().astype(int)
if not years.empty:
    upd = years.value_counts().sort_index()
    fig, ax = plt.subplots(figsize=(10,5))
    upd.plot(marker='o', ax=ax)
    ax.set_title("Number of Apps Updated per Year")
    ax.set_xlabel("Year")
    ax.set_ylabel("Number of Apps Updated")
    save_fig(fig, "apps_updated_per_year.png")

# 6) Relationship between app_age_days and rating (scatter + regression)
fig, ax = plt.subplots(figsize=(9,6))
scatter_df = df[df['app_age_days'].notna() & df['Rating'].notna()].copy()
# sample to speed plotting if dataset large
if scatter_df.shape[0] > 3000:
    scatter_df_plot = scatter_df.sample(3000, random_state=42)
else:
    scatter_df_plot = scatter_df
sns.regplot(x='app_age_days', y='Rating', data=scatter_df_plot, scatter_kws={'s':10, 'alpha':0.5}, line_kws={'color':'red'}, ax=ax)
ax.set_title("App Age (days since last update) vs Rating")
ax.set_xlabel("App age (days)")
ax.set_ylabel("Rating")
save_fig(fig, "age_vs_rating_regression.png")

# 7) Paid apps: Price vs Installs (log installs), highlight price buckets
paid = df[df['Price_USD'] > 0].copy()
if not paid.empty:
    fig, ax = plt.subplots(figsize=(9,6))
    paid_plot = paid.sample(n=min(1200, paid.shape[0]), random_state=42)
    sc = ax.scatter(paid_plot['Price_USD'] + 0.01, paid_plot['Installs'].replace(0, np.nan),
                    c=paid_plot['Price_USD'], cmap='viridis', alpha=0.7, s=40)
```

```
    ax.set_yscale('log')
    ax.set_xscale('symlog')  # allow small prices
    ax.set_xlabel("Price (USD)")
    ax.set_ylabel("Installs (log scale)")
    ax.set_title("Paid Apps: Price vs Installs (log scale)")
    cbar = plt.colorbar(sc, ax=ax)
    cbar.set_label('Price (USD)')
    save_fig(fig, "paid_price_vs_installs.png")

# 8) Installs per MB distribution (histogram)
fig, ax = plt.subplots(figsize=(9,5))
ipm = df['Installs_per_MB'].dropna()
# cap to a reasonable range for visualization
ipm_plot = ipm[(ipm > 0) & (ipm < ipm.quantile(0.99))]
sns.histplot(ipm_plot, bins=40, ax=ax, kde=True)
ax.set_title("Installs per MB (filtered)")
ax.set_xlabel("Installs per MB")
save_fig(fig, "installs_per_mb_hist.png")

# 9) Correlation heatmap of numeric features (Rating, Installs, Reviews, Size_MB, Price)
num_cols = ['Rating','Installs','Reviews','Size_MB','Price_USD','app_age_days','Installs_per_MB']
num_df = df[num_cols].copy()
corr = num_df.corr()
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", ax=ax)
ax.set_title("Correlation (numeric features)")
save_fig(fig, "numeric_correlation_heatmap.png")
```

```
Saved: output_googleplay\count_by_price_bucket.png
Saved: output_googleplay\avg_rating_by_size_bucket.png
Saved: output_googleplay\installs_by_rating_group_boxplot.png
Saved: output_googleplay\top10_categories_by_avg_installs.png
Saved: output_googleplay\apps_updated_per_year.png
Saved: output_googleplay\age_vs_rating_regression.png
Saved: output_googleplay\paid_price_vs_installs.png
Saved: output_googleplay\installs_per_mb_hist.png
Saved: output_googleplay\numeric_correlation_heatmap.png
```

Extract insights

```
insights = []
# Net observations
total_apps = df.shape[0]
free_pct = (df['Price_Bucket'] == 'Free').mean() * 100
median_rating = df['Rating'].median()
mean_installs = df['Installs'].median()  # median to avoid skew
insights.append(f"Total apps analyzed: {total_apps}")
insights.append(f"Free apps: {free_pct:.1f}%")
insights.append(f"Median rating across dataset: {median_rating:.2f}")
insights.append(f"Median installs (per app): {mean_installs:,.0f}")

# Top categories by count and by avg installs
if 'Category' in df.columns:
    top_categories_count = df['Category'].value_counts().head(3)
    insights.append("Top 3 categories by app count:")
    for c, v in top_categories_count.items():
        insights.append(f" - {c}: {v} apps")
    top_by_installs = df.groupby('Category')['Installs'].mean().dropna().sort_values(ascending=False).head(3)
    insights.append("Top 3 categories by average installs:")
    for c, v in top_by_installs.items():
        insights.append(f" - {c}: {v:,.0f} avg installs")

# Size vs rating insight
avg_rating_by_size = df.groupby('Size_Bucket')['Rating'].mean().dropna()
best_size = avg_rating_by_size.idxmax() if not avg_rating_by_size.empty else 'N/A'
insights.append(f"App size category with highest average rating: {best_size}")

# Paid apps installs behavior
if not paid.empty:
```

```
    median_installs_paid = paid['Installs'].median()
    insights.append(f"Median installs for paid apps: {median_installs_paid:,.0f}")

# Age vs rating correlation
if 'app_age_days' in df.columns and df['app_age_days'].notna().any():
    corr_age_rating = df[['app_age_days','Rating']].dropna().corr().iloc[0,1]
    insights.append(f"Correlation between app age and rating: {corr_age_rating:.3f}")

# Save summary report
with open(REPORT_TXT, 'w') as f:
    f.write("Google Play Store — Project Summary\n")
    f.write("="*60 + "\n\n")
    for line in insights:
        f.write(line + "\n")
    f.write("\nPlots and cleaned data are saved in the folder: " + OUTPUT_DIR + "\n")

print("\nSummary insights:")
for ln in insights:
    print(" -", ln)

print("\nOutput folder contents:")
print(os.listdir(OUTPUT_DIR))
```

```
Summary insights:
 - Total apps analyzed: 10358
 - Free apps: 92.6%
 - Median rating across dataset: 4.30
 - Median installs (per app): 100,000
 - Top 3 categories by app count:
 -  - FAMILY: 1943 apps
 -  - GAME: 1121 apps
 -  - TOOLS: 843 apps
 - Top 3 categories by average installs:
 -  - COMMUNICATION: 65,989,826 avg installs
 -  - SOCIAL: 44,692,385 avg installs
 -  - VIDEO_PLAYERS: 35,554,301 avg installs
 - App size category with highest average rating: Unknown
 - Median installs for paid apps: 1,000
 - Correlation between app age and rating: -0.126

Output folder contents:
['age_vs_rating_regression.png', 'apps_updated_per_year.png', 'avg_rating_by_size_bucket.png', 'cleaned_googleplay.csv', 'count_by_price_bucket.png', 'installs_by_ratin
```

End