# Machine Learning - 2301CS621

# Lab - 2

635 | Vishal Baraiya | 23010101014

# EDA & Pipeline: Google Play Store Apps

**Dataset:** Google Play Store Apps (Available on Kaggle)
**Objective:** Transform raw, messy data into clean, actionable insights using Pandas and Scikit-Learn pipelines.
**Focus:** Data Cleaning, String Sanitization, Advanced Imputation, Correlation, and Pipelines.

## 1. Setup & Initialization

**Exercise 1: Import Dependencies**

- Import `pandas`, `numpy`, `matplotlib.pyplot`, and `seaborn`.
- Set pandas options to display all columns (visual aid).

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

**Exercise 2: Data Loading & Initial Inspection**

- Load the `googleplaystore.csv` file.
- Display the first 5 rows.
- **Check:** Look closely at the `Installs`, `Size`, and `Price` columns. Notice they are currently Objects (strings), not numbers.

In [2]:
```
df = pd.read_csv("googleplaystore.csv")
df.head(5)
```

Out[2]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Photo Editor & Candy Camera & Grid & ScrapBook | ART_AND_DESIGN | 4.1 | 159 | 19M | 10,000+ | Free | 0 | Everyone |
| **1** | Coloring book moana | ART_AND_DESIGN | 3.9 | 967 | 14M | 500,000+ | Free | 0 | Everyone |
| **2** | U Launcher Lite – FREE Live Cool Themes, Hide ... | ART_AND_DESIGN | 4.7 | 87510 | 8.7M | 5,000,000+ | Free | 0 | Everyone |
| **3** | Sketch - Draw & Paint | ART_AND_DESIGN | 4.5 | 215644 | 25M | 50,000,000+ | Free | 0 | Teen |
| **4** | Pixel Draw - Number Art Coloring Book | ART_AND_DESIGN | 4.3 | 967 | 2.8M | 100,000+ | Free | 0 | Everyone |

## 2. Data Integrity Check

**Exercise 3: Audit Data Types and Missing Values**

- Use a single command to view data types ( `dtypes` ) and non-null counts.
- Calculate the *percentage* of missing values for each column.

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   App             10841 non-null  object 
 1   Category        10841 non-null  object 
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object 
 4   Size            10841 non-null  object 
 5   Installs        10841 non-null  object 
 6   Type            10840 non-null  object 
 7   Price           10841 non-null  object 
 8   Content Rating  10840 non-null  object 
 9   Genres          10841 non-null  object 
 10  Last Updated    10841 non-null  object 
 11  Current Ver     10833 non-null  object 
 12  Android Ver     10838 non-null  object 
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

In [4]:
```
print((df.isnull().sum()/len(df))*100)
```

```
App               0.000000
Category          0.000000
Rating           13.596532
Reviews           0.000000
Size              0.000000
Installs          0.000000
Type              0.009224
Price             0.000000
Content Rating    0.009224
Genres            0.000000
Last Updated      0.000000
Current Ver       0.073794
Android Ver       0.027673
dtype: float64
```

**Exercise 4: Handling Duplicates**

- Duplicate entries skew results. Check for duplicate rows.
- Drop duplicates, keeping the *first* occurrence. Verify the shape change.

In [5]:
```
# For Checking Duplicates use duplicated()
# Use drop_duplicates to Drop()

print(df.duplicated().sum())
df.drop_duplicates(inplace=True)
print(df.duplicated().sum())
```

```
483
0
```

# 3. Advanced String Sanitization (Crucial Step)

### Exercise 5: Cleaning the 'Installs' Column

- The `Installs` column contains characters like `+` and `,` (e.g., "10,000+").
- Remove these characters.
- Convert the column to a numeric integer type.

In [6]:
```
# use Column.astype(str).str.replace
df["Installs"] = df["Installs"].astype(str).str.replace('+','').str.replace(',','')
```

In [7]:
```
# use to_numeric to convert to int
df["Installs"] = pd.to_numeric(df['Installs'],errors='coerce')
```

In [8]:
```
print(df['Installs'].dtypes)
```

```
float64
```

In [9]:
```
df.head(5)
```

Out[9]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Photo Editor & Candy Camera & Grid & ScrapBook | ART_AND_DESIGN | 4.1 | 159 | 19M | 10000.0 | Free | 0 | Everyone |
| **1** | Coloring book moana | ART_AND_DESIGN | 3.9 | 967 | 14M | 500000.0 | Free | 0 | Everyone |
| **2** | U Launcher Lite – FREE Live Cool Themes, Hide ... | ART_AND_DESIGN | 4.7 | 87510 | 8.7M | 5000000.0 | Free | 0 | Everyone |
| **3** | Sketch - Draw & Paint | ART_AND_DESIGN | 4.5 | 215644 | 25M | 50000000.0 | Free | 0 | Teen |
| **4** | Pixel Draw - Number Art Coloring Book | ART_AND_DESIGN | 4.3 | 967 | 2.8M | 100000.0 | Free | 0 | Everyone |

### Exercise 6: Cleaning the 'Price' Column

- The `Price` column contains the `$` symbol (e.g., "$4.99").

- Remove the symbol.
- Convert the column to a `float`.

In [10]:
```
# Same as Above
df["Price"] = df["Price"].astype(str).str.replace('$','')
df["Price"] = pd.to_numeric(df["Price"],errors='coerce')
print(df['Price'].dtypes)
```

```
float64
```

**Exercise 7: Complex Logic - Sanitizing 'Size'**

- The `Size` column is messy. It contains 'M' (Megabytes), 'k' (kilobytes), and string 'Varies with device'.
- **Task:** Write a function (or apply lambda) to:
  1. Replace 'k' with 'e+3' and 'M' with 'e+6'.
  2. Coerce 'Varies with device' to `NaN`.
  3. Convert the string to a number.

In [11]:
```
# Hint: Define a function clean_size(x).
# Hint: If 'M' in x: return float(x.replace('M', '')) * 1000000
# Hint: Handle the 'Varies with device' edge case carefully.

def clean_size(x):
    x = str(x)
    if 'M' in x:
        # Convert 19M to 19000000
        return float(x.replace('M', '')) * 1_000_000

    elif 'k' in x:
        # Convert 500k to 500000
        return float(x.replace('k', '')) * 1_000

    elif 'Varies with device' in x:
        # Handle string edge case
        return np.nan

    else:
        # Attempt to convert or return NaN
        try:
            return float(x)
        except:
            return np.nan

# Use apply Method to apply above fun
df["Size"] = df["Size"].apply(clean_size)
print("DataSet is Cleaned")
df.head(10)
```

```
DataSet is Cleaned
```

Out[11]:

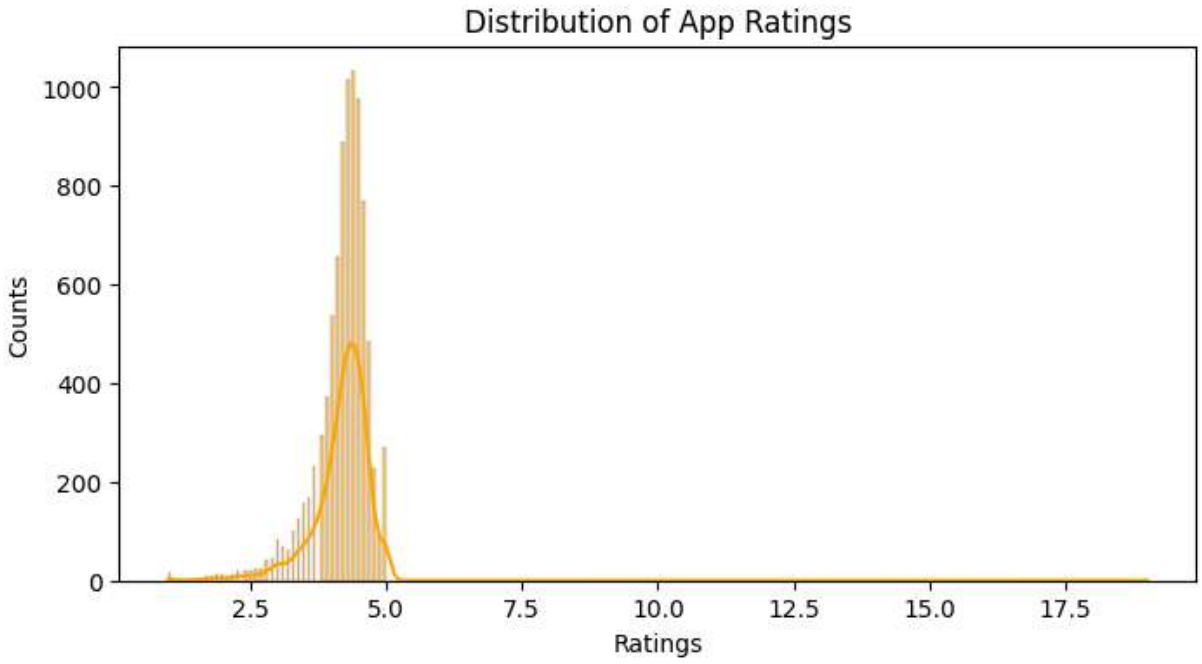
| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Co I |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Photo Editor & Candy Camera & Grid & ScrapBook | ART_AND_DESIGN | 4.1 | 159 | 19000000.0 | 10000.0 | Free | 0.0 | Eve |
| **1** | Coloring book moana | ART_AND_DESIGN | 3.9 | 967 | 14000000.0 | 500000.0 | Free | 0.0 | Eve |
| **2** | U Launcher Lite – FREE Live Cool Themes, Hide ... | ART_AND_DESIGN | 4.7 | 87510 | 8700000.0 | 5000000.0 | Free | 0.0 | Eve |
| **3** | Sketch - Draw & Paint | ART_AND_DESIGN | 4.5 | 215644 | 25000000.0 | 50000000.0 | Free | 0.0 | |
| **4** | Pixel Draw - Number Art Coloring Book | ART_AND_DESIGN | 4.3 | 967 | 2800000.0 | 100000.0 | Free | 0.0 | Eve |
| **5** | Paper flowers instructions | ART_AND_DESIGN | 4.4 | 167 | 5600000.0 | 50000.0 | Free | 0.0 | Eve |
| **6** | Smoke Effect Photo Maker - Smoke Editor | ART_AND_DESIGN | 3.8 | 178 | 19000000.0 | 50000.0 | Free | 0.0 | Eve |
| **7** | Infinite Painter | ART_AND_DESIGN | 4.1 | 36815 | 29000000.0 | 1000000.0 | Free | 0.0 | Eve |
| **8** | Garden Coloring Book | ART_AND_DESIGN | 4.4 | 13791 | 33000000.0 | 1000000.0 | Free | 0.0 | Eve |
| **9** | Kids Paint Free - Drawing Fun | ART_AND_DESIGN | 4.7 | 121 | 3100000.0 | 10000.0 | Free | 0.0 | Eve |

## 4. Advanced Imputation

### Exercise 8: Analyzing Missing 'Rating'

- The `Rating` column has missing values.
- **Visualize** the distribution of Ratings using a Histogram or KDE plot to decide between Mean vs Median imputation.

In [12]:
```
plt.figure(figsize=(8,4))
sns.histplot(df["Rating"],kde=True,color='orange')
plt.xlabel("Ratings")
plt.ylabel("Counts")
plt.title("Distribution of App Ratings")
plt.show()
```




### Exercise 9: Group-Specific Imputation

- Fill missing `Rating` values with the **Median Rating** of the specific `Category` the app belongs to.
- *Example:* If a "Business" app is missing a rating, fill it with the median rating of all "Business" apps.

In [13]:
```
df['Rating'] = df.groupby('Category')['Rating'].transform(lambda x : x.fillna(x.med
#After this use transform Method
print(f"Missing Rating Afrter imputation : {df['Rating'].isnull().sum()}")
```

```
Missing Rating Afrter imputation : 0
```

### Exercise 10: Drop Remaining NaNs

- For the remaining columns with minimal missing data (like `Current Ver`), simply drop the rows containing NaNs to ensure a clean dataset for correlation.

In [14]:
```
# dropna
df.shape
```

Out[14]:
```
(10358, 13)
```

In [15]:
```
df.dropna(inplace=True)
df.shape
```

Out[15]:
```
(8821, 13)
```

## 5. Correlation & Visualization

### Exercise 11: Correlation Matrix

- Generate a correlation matrix for the numerical columns ( `Rating` , `Reviews` , `Size` , `Installs` , `Price` ).

In [16]:
```
# Hint: numeric_only Use CORR
corr_matrix = df.corr(numeric_only=True)
corr_matrix
```

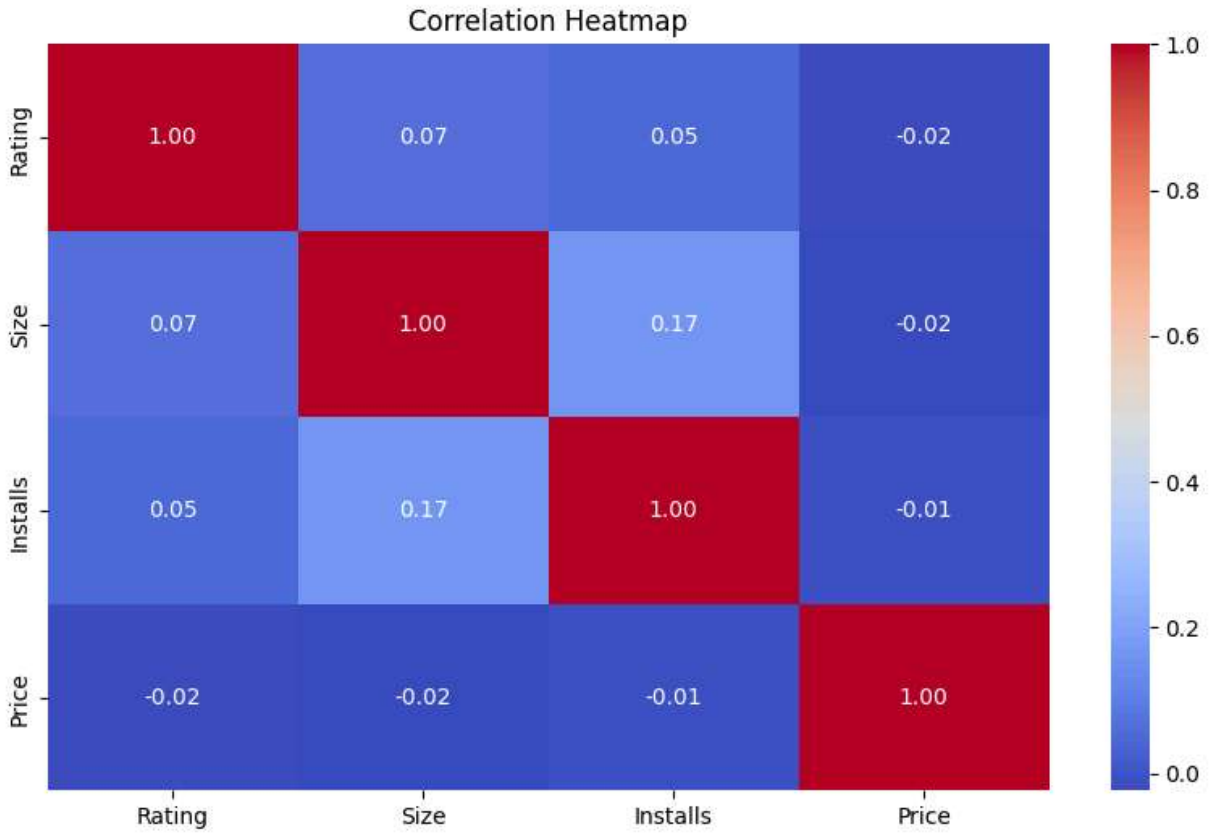
Out[16]:

| | Rating | Size | Installs | Price |
|---|---|---|---|---|
| **Rating** | 1.000000 | 0.065161 | 0.048168 | -0.019242 |
| **Size** | 0.065161 | 1.000000 | 0.168805 | -0.023818 |
| **Installs** | 0.048168 | 0.168805 | 1.000000 | -0.010290 |
| **Price** | -0.019242 | -0.023818 | -0.010290 | 1.000000 |

### Exercise 12: Heatmap Visualization

- Visualize the correlation matrix using a Seaborn Heatmap.
- Annotate the values.

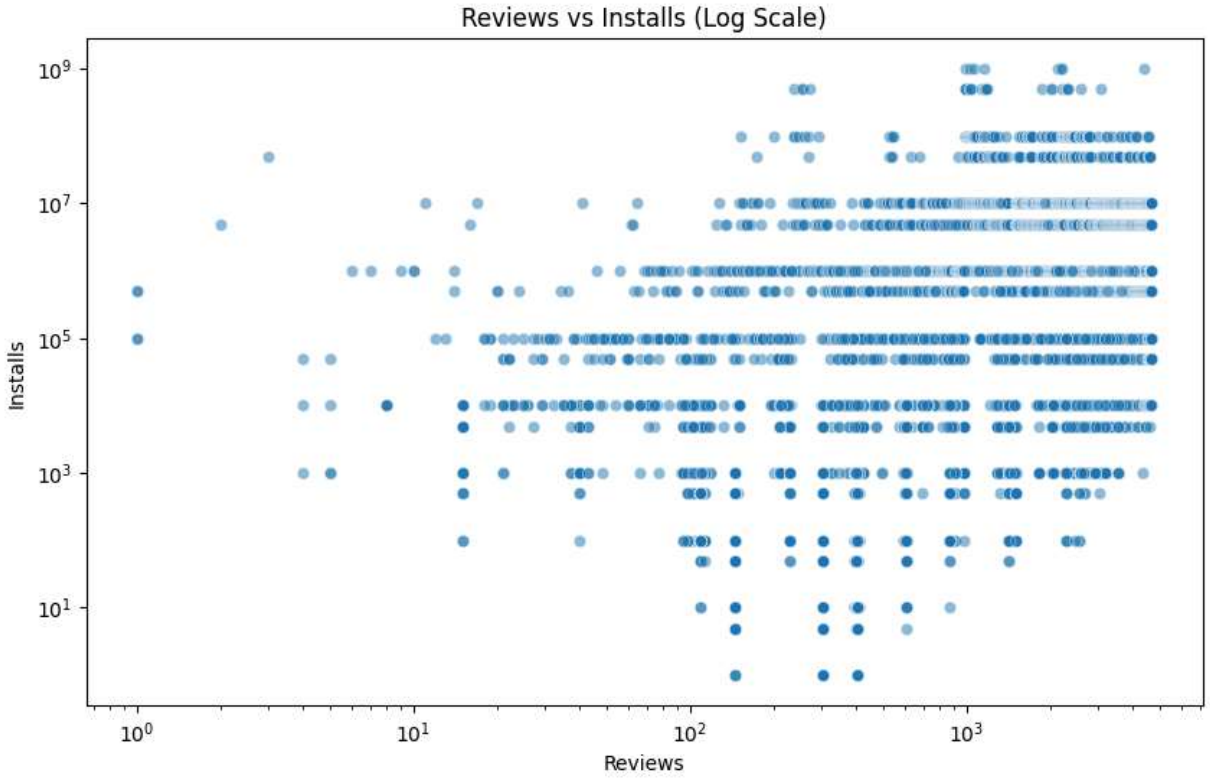
In [17]:
```
# Hint: sns.heatmap
plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix,annot=True,cmap='coolwarm',fmt='.2f')
plt.title("Correlation Heatmap")
plt.show()
```




### Exercise 13: Scatter Plot Analysis

- Create a Scatter Plot to analyze the relationship between `Reviews` and `Installs`.
- **Note:** You might need to use a log scale for the axes if the data is skewed.

In [18]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'Reviews',y = 'Installs',data=df,alpha=0.5)
# If needed
plt.xscale('log')
plt.yscale('log')
plt.title('Reviews vs Installs (Log Scale)')
plt.show()
```

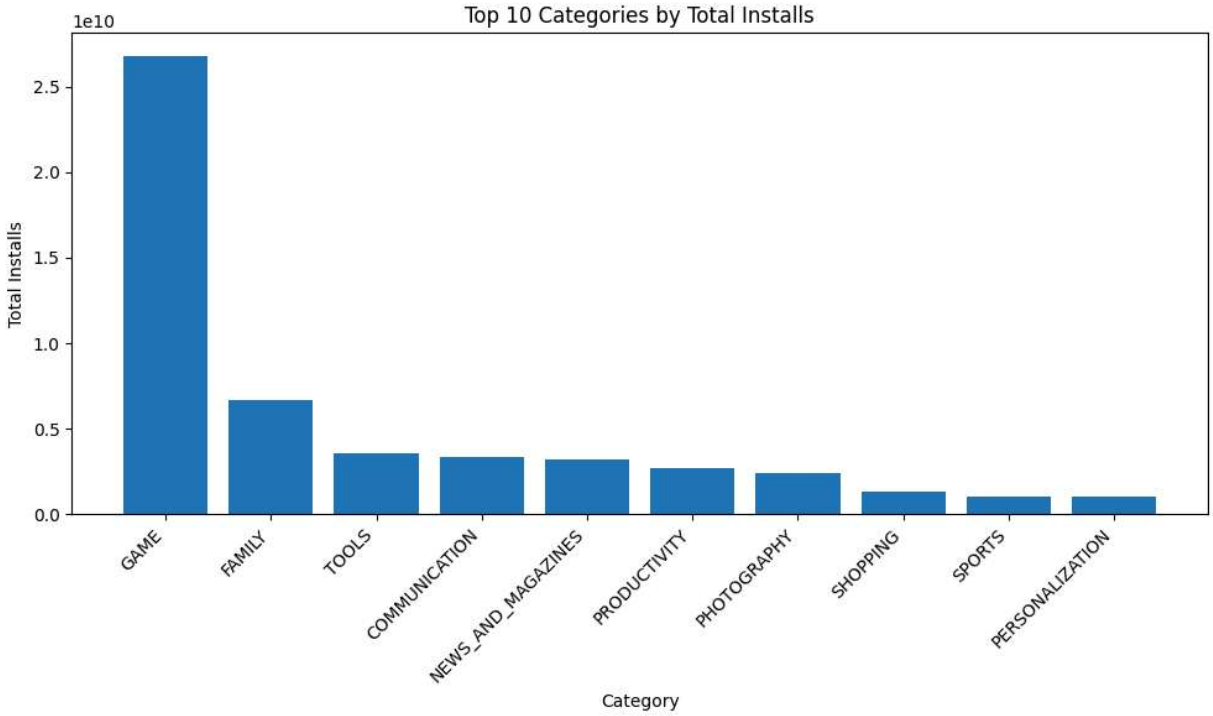



In [ ]:

### Exercise 14: Categorical Aggregation

- Create a Bar Plot showing the top 10 Categories by **Total Installs**.

In [23]:
```
# 1. Group by Category and aggregate installs
category_installs = df.groupby('Category')['Installs'].sum()

# 2. Sort and take Top 10
top10 = category_installs.sort_values(ascending=False).head(10)

plt.figure(figsize=(10, 6))
plt.bar(top10.index, top10.values)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Category")
plt.ylabel("Total Installs")
plt.title("Top 10 Categories by Total Installs")
plt.tight_layout()
plt.show()
```




## 6. Building a Sklearn Pipeline

**Exercise 15: Preprocessing Pipeline**

- Imagine you want to predict the `Rating` (Target).
- Create a Scikit-Learn `ColumnTransformer` that:
  1. StandardScales the numerical features ( `Reviews` , `Size` , `Installs` , `Price` ).
  2. OneHotEncodes the categorical feature ( `Category` , `Content Rating` ).
- Display the pipeline object.

# Without Pipline

In [28]:
```
# from sklearn.preprocessing import StandardScaler
```

In [30]:
```
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Numerical and Categorical Columns
numeric_features = ["Reviews", "Size", "Installs", "Price"]
categorical_features = ["Category", "Content Rating"]

# ----- Step 1: Scale Numerical Columns -----
scaler = StandardScaler()
scaled_numeric = scaler.fit_transform(df[numeric_features])

scaled_numeric_df = pd.DataFrame(
    scaled_numeric,
    columns=numeric_features
)
```

```
# ----- Step 2: One-Hot Encode Categorical Columns -----
encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
encoded_categorical = encoder.fit_transform(df[categorical_features])

encoded_cat_df = pd.DataFrame(
    encoded_categorical,
    columns=encoder.get_feature_names_out(categorical_features)
)

# ----- Step 3: Combine All Preprocessed Features -----
final_preprocessed_df = pd.concat(
    [scaled_numeric_df, encoded_cat_df],
    axis=1
)

final_preprocessed_df.head()
```

Out[30]:

| | Reviews | Size | Installs | Price | Category_ART_AND_DESIGN | Category_AUTO_ |
|---|---|---|---|---|---|---|
| **0** | -0.146291 | -0.102315 | -0.154446 | -0.066729 | 1.0 | |
| **1** | -0.145787 | -0.324102 | -0.142936 | -0.066729 | 1.0 | |
| **2** | -0.091790 | -0.559195 | -0.037234 | -0.066729 | 1.0 | |
| **3** | -0.011842 | 0.163829 | 1.019785 | -0.066729 | 1.0 | |
| **4** | -0.145787 | -0.820903 | -0.152332 | -0.066729 | 1.0 | |

5 rows × 43 columns

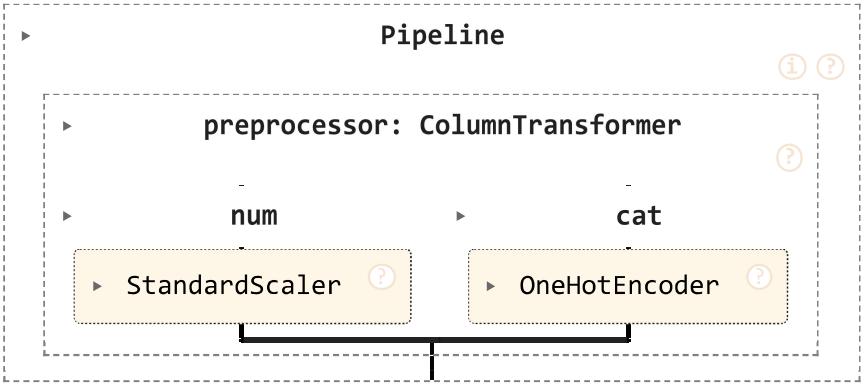
# With Pipline(Optional part)

In [26]:
```
# Hint: from sklearn.compose import ColumnTransformer
# Hint: from sklearn.preprocessing import StandardScaler, OneHotEncoder
# Hint: from sklearn.pipeline import Pipeline
```

In [31]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline

# ----- Step 1: Define feature groups -----

numeric_features = ["Reviews", "Size", "Installs", "Price"]
categorical_features = ["Category", "Content Rating"]

# ----- Step 2: Create the ColumnTransformer -----

preprocessor = ColumnTransformer(
```

```
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features)
    ]
)

# ----- Step 3: Build full Pipeline (only preprocessing for now) -----

pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor)
])

# ----- Step 4: Display the Pipeline -----

pipeline
```

Out[31]:




In [ ]: