Dataset:
https://drive.google.com/file/d/1FuQr0wM8tczVQvtO8_Nn82C3E4gQTgYF/view

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")

%matplotlib inline


df=pd.read_csv("googleplaystore.csv")

df[~df['Reviews'].str.isnumeric()]

df_copy=df.copy()

df_copy=df_copy.drop(df_copy.index[10472])

df_copy["Reviews"]=df_copy["Reviews"].astype('int')

df_copy["Size"]=df_copy["Size"].str.replace('M','000')

df_copy["Size"]=df_copy["Size"].str.replace("Varies with device",
str(np.nan))

for i in df_copy['Size']:
    if i < 10:
        df_copy['Size']=df_copy['Size'].replace(i,i*1000)


chars_to_remove=['+',',','$']
cols_to_clean=['Installs',"Price"]
for item in chars_to_remove:
```

```
    for col in cols_to_clean:
        df_copy[col]=df_copy[col].str.replace(item,'')

df_copy["Installs"]=df_copy["Installs"].astype('int')

df_copy["Price"]=df_copy["Price"].astype('float')

df_copy["Last Updated"]=pd.to_datetime(df_copy["Last Updated"])

df_copy["day"]=df_copy["Last Updated"].dt.day
df_copy["month"]=df_copy["Last Updated"].dt.month
df_copy["year"]=df_copy["Last Updated"].dt.year

df_copy.to_csv("google_cleaned.csv",index=False)

pd.read_csv("google_cleaned.csv").info()
```

https://drive.google.com/file/d/1RqYkSLY1aw0hc2Jmvdmiju-0hYBFhmbW/view?usp=sharing

```
numeric_features = [feature for feature in df.columns if df[feature].dtype
!= 'O']
categorical_features = [feature for feature in df.columns if
df[feature].dtype == 'O']


plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)

for i in range(0, len(numeric_features)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[numeric_features[i]],shade=True, color='r')
    plt.xlabel(numeric_features[i])
```

```
    plt.tight_layout()


# categorical columns
plt.figure(figsize=(20, 15))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)
category = [ 'Type', 'Content Rating']
for i in range(0, len(category)):
    plt.subplot(2, 2, i+1)
    sns.countplot(x=df[category[i]],palette="Set2")
    plt.xlabel(category[i])
    plt.xticks(rotation=45)
    plt.tight_layout()


cat_df["Category"].value_counts().plot.pie(figsize=(20,20),autopct =
'%1.1f%%')

category=pd.DataFrame(cat_df["Category"].value_counts())

category.rename(columns={"Category":"Count"},inplace=True)

plt.figure(figsize=(20,20))
sns.barplot(x=category.index[:10],y="Count",data=category[:10])


df.groupby(['Category'])["Installs"].sum().sort_values(ascending=False)
```

```
df.groupby(['Category'])["Installs"].sum().nlargest(10).plot(kind='bar',figsize=(20,10))
```

how many apps are there on google play store which get 5 ratings?
does size of the application has any impact on its popularity?
what are the top 5 most installed apps in each popular category?
which category app users are reviewing the most?
which kind of app user are downloading the most free/paid?

<u>Additional things</u>
liner regression
num_df.head()
you need to create a model(linear regression) where all the features except Rating will be independent features and rating will be a dependent feature
create a model :
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

—--classs no-3

```
null_df = pd.DataFrame({'Null Values' : df.isna().sum().sort_values(ascending=False), 'Percentage Null Values' : (df.isna().sum().sort_values(ascending=False)) / (df.shape[0]) * (100)})
null_df
```

```
null_counts = df.isna().sum().sort_values(ascending=False)/len(df)
plt.figure(figsize=(16,8))
plt.xticks(np.arange(len(null_counts))+0.5,null_counts.index,rotation='vertical')
plt.ylabel('Fraction of rows with missing data')
plt.bar(np.arange(len(null_counts)),null_counts)
```

```
plt.show()


cols = [var for var in df_copy.columns if df_copy[var].isnull().mean()*100]
cols

drop_df = df_copy[cols].dropna()


drop_df


fig= plt.figure()

# density plot using seaborn library
fig, axs = plt.subplots(2, 2, figsize=(15, 7))

drop_df['Size'].plot.density(color='red',ax=axs[0,
0],alpha=0.5,label='Size')
df_copy['Size'].plot.density(color='green',ax=axs[0,
0],alpha=0.5,label='Size')
drop_df['Rating'].plot.density(color='red',ax=axs[0,
1],alpha=0.5,label='Rating')
df_copy['Rating'].plot.density(color='green',ax=axs[0,
1],alpha=0.5,label='Rating')
drop_df['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='red')
df_copy['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='green', alpha=0.8)
drop_df['Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='red')
df_copy['Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='green', alpha=0.8)

plt.show()

df_copy_me_mo[df_copy_me_mo['Size'].isnull()]
```

```
df_copy_me_mo['mean_Size'] =
df_copy_me_mo['Size'].fillna(df_copy_me_mo['Size'].mean())
df_copy_me_mo['median_Size'] =
df_copy_me_mo['Size'].fillna(df_copy_me_mo['Size'].median())
df_copy_me_mo['mean_Rating'] =
df_copy_me_mo['Rating'].fillna(df_copy_me_mo['Rating'].mean())
df_copy_me_mo['median_Rating'] =
df_copy_me_mo['Rating'].fillna(df_copy_me_mo['Rating'].median())


print('Original Rating Variance', df_copy_me_mo['Rating'].var())
print('Rating Variance After mean imputation',
df_copy_me_mo['mean_Rating'].var())
print('Rating Variance After median imputation',
df_copy_me_mo['median_Rating'].var())
```

—--

```
##plotting of the data for mean and meadian
fig= plt.figure()

# density plot using seaborn library
fig, axs = plt.subplots(2, 2, figsize=(15, 7))

df_copy_me_mo['Size'].plot.density(color='blue',ax=axs[0,
0],alpha=0.5,label='Size')
df_copy_me_mo['mean_Size'].plot.density(color='green',ax=axs[0,
0],alpha=0.5,label='mean_Size')
df_copy_me_mo['median_Size'].plot.density(color='red',ax=axs[0,
0],alpha=0.5,label='median_Size')

df_copy_me_mo['Rating'].plot.density(color='blue',ax=axs[0,
1],alpha=0.5,label='Rating')
df_copy_me_mo['mean_Rating'].plot.density(color='green',ax=axs[0,
1],alpha=0.5,label='mean_Rating')
```

```
df_copy_me_mo['median_Rating'].plot.density(color='red',ax=axs[0,
1],alpha=0.5,label='median_Rating')

df_copy_me_mo['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='blue')
df_copy_me_mo['mean_Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='red')
df_copy_me_mo['median_Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='green', alpha=0.8)

df_copy_me_mo['Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='blue')
df_copy_me_mo['mean_Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='red')
df_copy_me_mo['median_Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='green', alpha=0.8)


plt.show()
```

—--

```
def Random_Sample_imputation(feature):

random_sample=df_random[feature].dropna().sample(df_random[featur
e].isnull().sum())
   random_sample.index=df_random[df_random[feature].isnull()].index
   df_random.loc[df_random[feature].isnull(),feature]=random_sample


for col in df_random:
   Random_Sample_imputation(col)
```

—---

Plotting of df_random

```
fig= plt.figure()
```

```
# density plot using seaborn library
fig, axs = plt.subplots(2, 2, figsize=(15, 7))

df['Size'].plot.density(color='red',ax=axs[0, 0],alpha=0.5,label='Size')
df_random['Size'].plot.density(color='green',ax=axs[0,
0],alpha=0.5,label='Size')
df['Rating'].plot.density(color='red',ax=axs[0, 1],alpha=0.5,label='Rating')
df_random['Rating'].plot.density(color='green',ax=axs[0,
1],alpha=0.5,label='Rating')
df['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='red')
df_random['Size'].hist(bins=50,ax=axs[1,
0],density=True,figsize=(12,12),color='green', alpha=0.8)
df['Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='red')
df_random['Rating'].hist(bins=50,ax=axs[1,
1],density=True,figsize=(12,12),color='green', alpha=0.8)

plt.show()
```
—---

```
# Function to detect outliers
def outlier_thresholds(dataframe, variable):
   quartile1 = dataframe[variable].quantile(0.10)
   quartile3 = dataframe[variable].quantile(0.90)
   interquantile_range = quartile3 - quartile1
   up_limit = quartile3 + 1.5 * interquantile_range
   low_limit = quartile1 - 1.5 * interquantile_range
   return low_limit, up_limit

## function to remove outliers
def replace_with_thresholds(dataframe, numeric_columns):
   for variable in numeric_columns:
       low_limit, up_limit = outlier_thresholds(dataframe, variable)
       dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
       dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit
```
—--

```
plt.figure(figsize=(22,18))
for i,col in enumerate(num_df.columns):
    plt.subplot(4,9,i+1)
    sns.boxplot(num_df[col])
```