```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#1 call
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statistics as stc
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score


# In[2]:


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


# In[4]:


print("Count of null values in data")
df.isnull().sum()


# In[5]:


df.dropna(inplace=True)
print("Check for null values after removing nulls")
df.isnull().sum()


# In[6]:


df=df[-df['Size'].str.contains('Var')]


# In[7]:


df.loc[:,'SizeNum'] =df.Size.str.rstrip('Mk+')
df.SizeNum=pd.to_numeric(df['SizeNum'])
df.SizeNum.dtype
```

```
# In[8]:


df['SizeNum']=np.where(df.Size.str.contains('M'),df.SizeNum*1000, df.SizeNum)


# In[9]:


# Size no more needed, replace it with SizeNum and drop SizeNum
df.Size=df.SizeNum
df.drop('SizeNum',axis=1,inplace=True)
#df


# In[10]:


df.Reviews = pd.to_numeric(df.Reviews)


# In[11]:


df.Reviews.dtype


# In[12]:


df['Installs']=df.Installs.str.replace("+","")


# In[13]:


df.Installs=df.Installs.str.replace(",","")
df.Installs=pd.to_numeric(df.Installs)
df.Installs.dtype


# In[14]:


df.Price=df.Price.str.replace("$","")
df.Price=pd.to_numeric(df.Price)
df.Price.dtype


# In[15]:
```

```
df=df[(df.Rating>=1) & (df.Rating<=5) ]


# In[16]:


len(df.index)


# In[17]:


df.drop(df.index[df.Reviews>df.Installs],axis=0,inplace=True)
len(df.index)


# In[19]:


index_free_and_price_gt_0=df.index[((df.Type=='Free')&(df.Price>0))]
if len(index_free_and_price_gt_0)>0:
   print("Dropping following indices:",index_free_and_price_gt_0)
   df.drop(index_free_and_price_gt_0,axis=0,inplace=True)
else:
   print("There is no Free Apps with price >0")


# In[20]:


ax = sns.boxplot(x='Price', data=df)


# In[22]:


price_std=stc.stdev(df.Price)
price_std


# In[23]:


price_mean=stc.mean(df.Price)
price_mean


# In[24]:
```

```
price_outlier_uplimit=price_mean+3*price_std
price_outlier_uplimit


# In[25]:


#price_outlier_downlimit=price_mean-3*price_std
#price_outlier_downlimit
#df[df.Price>price_outlier_uplimit]
print("# of upper outliers is ",len(df[(df.Price>price_outlier_uplimit) ]))


# In[26]:


#df[df.Price<price_outlier_downlimit]
#print("# of lower outliers is ",len(df[df.Price<price_outlier_downlimit]))
sns.boxplot(x='Reviews',data=df)


# In[27]:


rev_std=stc.stdev(df.Reviews)
rev_std


# In[29]:


rev_mean=stc.mean(df.Reviews)
rev_mean


# In[30]:


rev_outlier_uplimit=rev_mean+3*rev_std
rev_outlier_uplimit


# In[31]:


rev_outlier_downlimit=rev_mean-3*rev_std
rev_outlier_downlimit


# In[32]:
```

```
#df[df.Reviews>rev_outlier_uplimit]
print("# of upper outliers is ",len(df[(df.Reviews>rev_outlier_uplimit) ]))


# In[33]:


# Since reviews cannot be less than 1, no need to check lower outliers
# remove outliers
#df.drop(df.index[(df.Reviews>rev_outlier_uplimit) ],inplace=True)
#len(df.index)


# In[34]:


#sns.boxplot(x='Rating',data=df)
sns.histplot(x='Rating',data=df)
#rating_std=stc.stdev(df.Rating)
#rating_std
#rating_mean=stc.mean(df.Rating)
#rating_mean
#rating_outlier_uplimit=rating_mean+3*rating_std
#rating_outlier_uplimit
#rating_outlier_downlimit=rating_mean-3*rating_std
#rating_outlier_downlimit
# Since max possible value of rating (5) is less than upper limit, no need to␣,→manage     ↩
upper outliers
#df[df.Rating<rating_outlier_downlimit]
#print("# of lower outliers is ",len(df[(df.Rating<rating_outlier_downlimit) ]))
#df.drop(df.index[(df.Rating<rating_outlier_downlimit)],inplace=True)
#len(df.index)


# In[35]:


# use log scale to make histogram more representable
sns.histplot(x='Size',data=df,log_scale=True)


# In[36]:


df[df.Price>=200]


# In[37]:
```

```
print("# of Apps with price >= 200 = ",len(df[(df.Price>=200) ]))


# In[38]:


df.drop(df.index[(df.Price>=200)], inplace=True)
len(df.index)


# In[39]:


df.drop(df.index[(df.Reviews>=2000000)], inplace=True)
len(df.index)


# In[40]:


install_10_perc=np.percentile(df.Installs, 10)
install_10_perc


# In[41]:


install_25_perc=np.percentile(df.Installs, 25)
install_25_perc


# In[42]:


install_50_perc=np.percentile(df.Installs, 50)
install_50_perc


# In[43]:


install_70_perc=np.percentile(df.Installs, 70)
install_70_perc


# In[44]:


install_90_perc=np.percentile(df.Installs,90)
```

```
install_90_perc


# In[45]:


install_95_perc=np.percentile(df.Installs,95)
install_95_perc


# In[46]:


install_99_perc=np.percentile(df.Installs,99)
install_99_perc


# In[47]:


sns.histplot(data=df,x='Installs',log_scale=True)


# In[48]:


print("As result, ",len(df[df.Installs >= install_99_perc])," will be dropped")


# In[49]:


df.drop(df.index[df.Installs >= install_99_perc],inplace=True)
len(df.index)


# In[50]:


sns.jointplot(data=df,y='Rating',x='Price')


# In[51]:


sns.jointplot(data=df,y='Rating',x='Size')


# In[52]:
```

```
sns.jointplot(data=df,y='Rating',x='Reviews')


# In[53]:


df['Content Rating'].unique()


# In[54]:


sns.boxplot(data=df,x='Rating',y='Content Rating')


# In[55]:


a4_dims = (11.7, 10.27)
fig, ax = plt.subplots(figsize=a4_dims)
sns.boxplot(data=df,x='Rating',y='Category',ax=ax)


# In[56]:


#8.1
inp1=df.copy()
inp1.Reviews=inp1.Reviews.apply(np.log1p)


# In[57]:


inp1.Installs=inp1.Installs.apply(np.log1p)


# In[60]:


#8.2
inp1.drop(columns=['App','Last Updated','Current Ver','Android Ver'],inplace=True)


# In[61]:


inp1.shape


# In[62]:
```

```
#8.3
inp2= pd.get_dummies(inp1)


# In[63]:


inp2.shape


# In[64]:


data = inp2.drop(columns='Rating')
data.shape


# In[65]:


target = pd.DataFrame(inp2.Rating)
target.shape


# In[68]:


x_train, x_test, y_train, y_test = train_test_split(data, target, test_size=0.3, random_state=3)
print("x_train shape is ", x_train.shape)
print("y_train shape is ", y_train.shape)
print("x_test shape is ", x_test.shape)
print("y_test shape is ", y_test.shape)


# In[69]:


model=LinearRegression()
model.fit(x_train, y_train)


# In[70]:


train_pred=model.predict(x_train)


# In[71]:
```

```
print("R2 value of the model(by train) is ", r2_score(y_train, train_pred))


# In[72]:


test_pred=model.predict(x_test)


# In[73]:


print("R2 value of the model(by test) is ", r2_score(y_test, test_pred))
```