```
# Do NOT modify this block of code

import numpy as np
import numpy.typing as npt
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns


# Do NOT modify this block of code

# Dataset 1
x1 = np.array([10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0])
y1 = np.array([8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68])

# Dataset 2
x2 = np.array([10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0])
y2 = np.array([9.14, 8.14, 8.74, 8.77, 9.26, 8.1, 6.13, 3.1, 9.13, 7.26, 4.74])

# Dataset 3
x3 = np.array([10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0])
y3 = np.array([7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73])

# Dataset 4
x4 = np.array([8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 19.0, 8.0, 8.0, 8.0])
y4 = np.array([6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.5, 5.56, 7.91, 6.89])

anscombe_quartet_df = pd.DataFrame([x1, y1, x2, y2, x3, y3, x4, y4], index=['x1', 'y1', 'x2', 'y2', 'x3', 'y3', 'x4', 'y4']).T
anscombe_quartet_df
```

	x1	y1	x2	y2	x3	y3	x4	y4
0	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
1	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
2	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
3	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
4	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
5	14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
7	4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
8	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
9	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
10	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

```
# Do NOT modify this block of code

def fit_regression_line(x: npt.NDArray[np.float64], y: npt.NDArray[np.float64]) -> npt.NDArray[np.float64]:
    """ function to fit a regression line on a data.
        args:
            x is the independent variable,
            y is the dependent variable.
        return: a 1-D numpy array of length 2 such that its first element is beta_0 and its second element is beta_1
    """
    beta_1 = (((x - x.mean()) * (y - y.mean())).sum()) / ((x - x.mean())**2).sum()

    beta_0 = y.mean() - beta_1 * x.mean()

    return np.array([beta_0, beta_1])


# Do NOT modify this block of code
def r_squared(x: npt.NDArray[np.float64], y: npt.NDArray[np.float64]) -> np.float64:
    """ function to the r-square score of a fitted regression line.
        args:
            x is the independent variable,
            y is the dependent variable.
        return: the r-squared score
    """
    beta_0, beta_1 = fit_regression_line(x, y)
    y_hat = beta_1 * x + beta_0

    return (np.corrcoef(y_hat, y, rowvar=False)[0, 1])**2
```

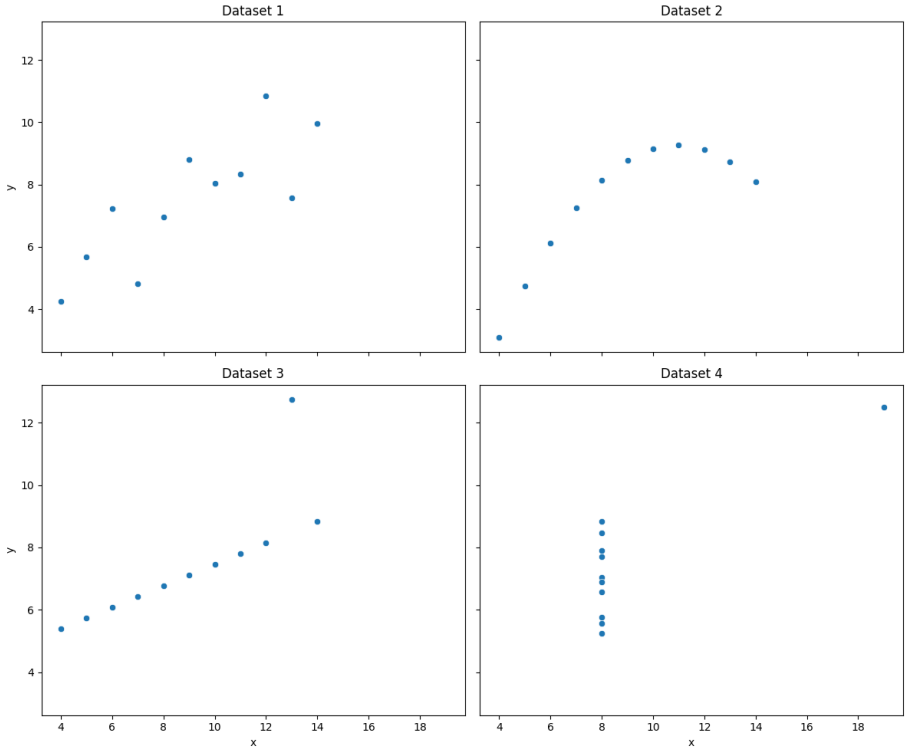
```
# Do NOT modify this block of code

def summary_statistics(data_df: pd.DataFrame) -> npt.NDArray[np.float64]:
    """ function to calculate some summary statistics.
    args:
        data_df is the input dataframe.
    return:
        a pandas dataframe showing the summary statistics all pairs of x_i and y_i
    """

    arr_result = np.zeros((8, data_df.shape[1] // 2))

    for i in range(0, data_df.shape[1], 2):
        x = data_df.iloc[:, i]
        y = data_df.iloc[:, i+1]

        arr_result[0, i//2] = np.mean(x) # calculate mean of x
        arr_result[1, i//2] = np.std(x) # calculate the standard deviation of x
        arr_result[2, i//2] = np.mean(y) # calculate mean of y
        arr_result[3, i//2] = np.std(y) # calculate the standard deviation of y
        arr_result[4, i//2] = np.corrcoef(x, y)[0, 1]

        beta_0, beta_1 = fit_regression_line(x, y)

        arr_result[5, i//2] = beta_0 # calculate beta_0 of regression line
        arr_result[6, i//2] = beta_1 # calculate beta_1 of regression line

        arr_result[7, i//2] = r_squared(x, y) # calculates the r-squared score

    result_df = pd.DataFrame(np.round(arr_result, 2),
                             index=['mean_x', 'std_x', 'mean_y', 'std_y', 'corrcoef_x_y', 'beta_0', 'beta_1', 'R^2'],
                             columns=[f"dataset_{i+1}" for i in range(arr_result.shape[1])]
        )

    return result_df


# Do NOT modify this block of code
summary_statistics(anscombe_quartet_df) # some summary statistics per dataset
```

	dataset_1	dataset_2	dataset_3	dataset_4
mean_x	9.00	9.00	9.00	9.00
std_x	3.16	3.16	3.16	3.16
mean_y	7.50	7.50	7.50	7.50
std_y	1.94	1.94	1.94	1.94
corrcoef_x_y	0.82	0.82	0.82	0.82
beta_0	3.00	3.00	3.00	3.00
beta_1	0.50	0.50	0.50	0.50
R^2	0.67	0.67	0.67	0.67

Q1a

```
## Your code starts here ##

#setting up matplotlib fig
#fig, axes unpacks the tuple returned by matplotlib
#(2,2) returns a 2x2 plot
#width/height of the fig
#share makes sures that all the subplots share the same axis ticks and range
fig, axes = plt.subplots(2, 2, figsize=(12, 10), sharex=True, sharey=True)

#labeling
label = ['Dataset 1', 'Dataset 2', 'Dataset 3', 'Dataset 4']

#dataset for iteration
dataset = [(x1, y1), (x2, y2), (x3, y3), (x4, y4)]

#zip fn combines multiple dictionaries into one and iterates over them
for ax, (x, y), label in zip(axes.flatten(), dataset, label):
  sns.scatterplot(x=x, y=y, ax=ax)
  ax.set_title(label)
  ax.set_xlabel('x')
  ax.set_ylabel('y')

#fixes spacing
plt.tight_layout()
#display the plot
plt.show

## end ##
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

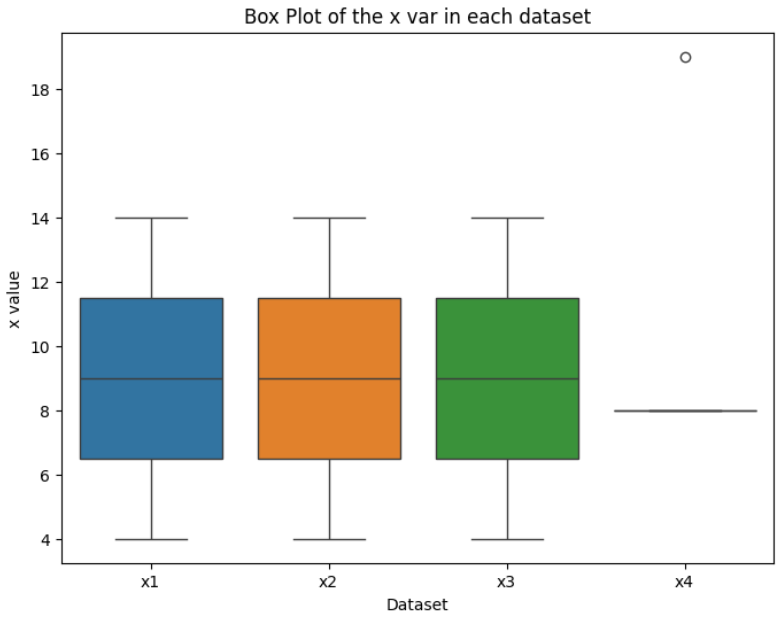


Your explanation goes here:

Ans: Although they are similar in statistical properties, their distributions and trends are different.

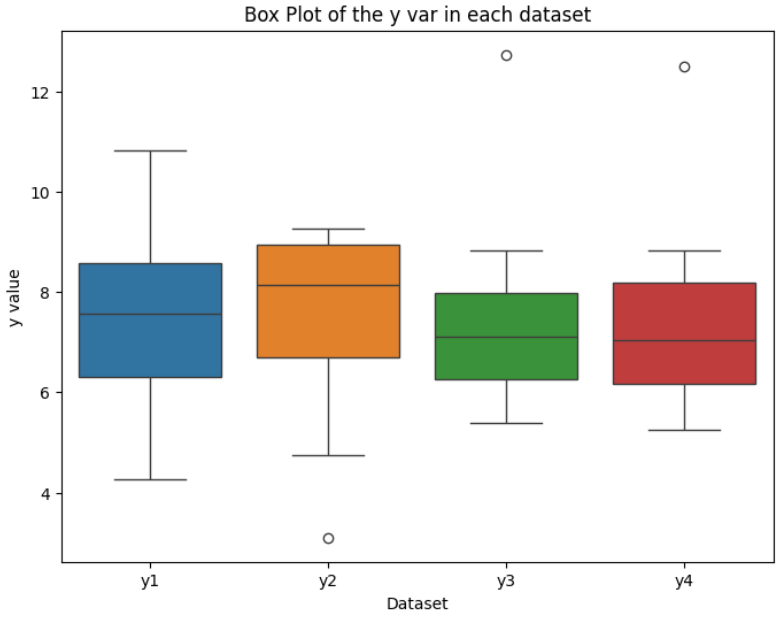
Q1b

```
## Your code starts here ##

#setting up matplotlib for box plots
plt.figure(figsize=(8, 6))

#getting data ready for box plot
boxplt_data = [anscombe_quartet_df[f'x{i+1}'] for i in range(4)]

#plotting; ticks are position on axes marked by lines and labels
sns.boxplot(data=boxplt_data)
plt.title('Box Plot of the x var in each dataset')
plt.xlabel('Dataset')
plt.ylabel('x value')
plt.xticks(ticks=range(4), label=label)
plt.show()

## end ##
```



Q1c

```
## Your code starts here ##

#setting up matplotlib for box plots
plt.figure(figsize=(8, 6))

#getting data ready for box plot, using a list where f'x{i+1} is f-string that allows for embedding expressions inside strings
boxplt_y_data = [anscombe_quartet_df[f'y{i+1}'] for i in range(4)]

#plotting
sns.boxplot(data=boxplt_y_data)
plt.title('Box Plot of the y var in each dataset')
plt.xlabel('Dataset')
plt.ylabel('y value')
plt.xticks(ticks=range(4), label=label)
plt.show()

## end ##
```



Loading in the data for Q2

```
# Do NOT modify this block of code

# loading in the data
google_playstore_df = pd.read_csv("https://raw.githubusercontent.com/MIE223-2024/course-datasets/main/googleplaystore.csv")
```

```
google_playstore_df = pd.read_csv( https://raw.githubusercontent.com/MIE223-2024/course-datasets/main/googleplaystore.csv )

print(google_playstore_df.shape)
google_playstore_df.head()
```

```
(10841, 13)
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design

Cleaning the Data

```
# Do NOT modify this block of code
google_playstore_df[google_playstore_df['Category'] == '1.9'] # corrupted row
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated

```
# Do NOT modify this block of code
# Dropping the corrupted row
google_playstore_df = google_playstore_df.drop(10472, axis=0)
google_playstore_df.shape
```

```
(10840, 13)
```

```
# Do NOT modify this block of code
google_playstore_df.isnull().sum() / google_playstore_df.shape[0] # % of null values per column
```

```
App               0.000000
Category          0.000000
Rating            0.135978
Reviews           0.000000
Size              0.000000
Installs          0.000000
Type              0.000092
Price             0.000000
Content Rating    0.000000
Genres            0.000000
Last Updated      0.000000
Current Ver       0.000738
Android Ver       0.000185
dtype: float64
```

```
# Do NOT modify this block of code

# Dropping rows with null values
google_playstore_df = google_playstore_df.dropna(axis=0)

print(f"Number of missing values is: {google_playstore_df.isnull().sum().sum()}, shape of data is: {google_playstore_df.shape}")
google_playstore_df.head()
```

```
Number of missing values is: 0, shape of data is: (9360, 13)
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design

```
# Do NOT modify this block of code
google_playstore_df.dtypes # checking the data types
```

```
App               object
Category          object
Rating           float64
Reviews           object
Size              object
Installs          object
Type              object
Price             object
Content Rating    object
Genres            object
Last Updated      object
Current Ver       object
Android Ver       object
dtype: object
```

```
# Do NOT modify this block of code
google_playstore_df.nunique() # checking the number of unique values per feature
```

```
App               8190
Category            33
Rating              39
Reviews           5990
Size               413
Installs            19
Type                 2
Price               73
Content Rating       6
Genres             115
Last Updated      1299
Current Ver       2638
Android Ver         31
dtype: int64
```

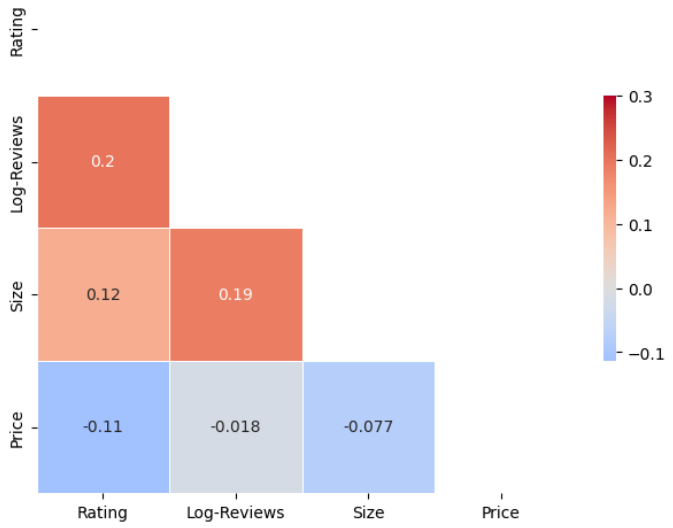
```
# Do NOT modify this block of code
def format_number(num: str) -> float:
    """Function to format a number by converting the place value from string to number e.g 1k to 1000, 1M to 1000000
    """
    num = num.lower() # convert to lowercase

    # if str number can be converted to float without further cleanup, convert it and return it
    try:
        return float(num.strip())
    except ValueError:
        pass

    # if after relacing the place value with number, num is still not convertible to float, return Nan e.g 'Varies with device' in Size column
    try:
        float(num[:-1].strip())
    except ValueError:
        return np.nan

    # else, replace the str place value by multiplying by the appropriate multiple of 10
    suffix_mapper = {'k': 1E3, 'm': 1E6, 'g': 1E9}

    return float(num[:-1]) * suffix_mapper[num[-1]]


# Do NOT modify this block of code
def format_place_value(num: str) -> str:
    """Function to format a number by converting it to its abbreviated place value e.g 1000 to 1k, 1000000 to 1M
    """
    num = int(num.strip("+").strip().replace(",", ""))

    if num >= 1_000_000_000:
        return f"{num // 1_000_000_000}G+"

    if num >= 1_000_000:
        return f"{num // 1_000_000}M+"

    if num >= 1000:
        return f"{num // 1000}k+"

    return f"{num}+"


# Do NOT modify this block of code
google_playstore_df['Log-Reviews'] = np.log(google_playstore_df.loc[:, 'Reviews'].astype(int)) # Converting 'reviews' column to float and taking the log


# Do NOT modify this block of code
google_playstore_df['Price'] = google_playstore_df['Price'].apply(lambda x: float(x.strip("$"))) # stripping $ and converting price to float


# Do NOT modify this block of code
google_playstore_df['Last Updated'] = pd.to_datetime(google_playstore_df['Last Updated']) # converting 'Last Updated' to datetime object


# Do NOT modify this block of code
google_playstore_df['Size'] = google_playstore_df['Size'].apply(format_number) # formating 'Size'. Note that after formatting, Nan values actually
                                                                               # indicate that the 'Size' varies with device.


# Do NOT modify this block of code
google_playstore_df['Installs'] = google_playstore_df['Installs'].apply(format_place_value) # formatting 'Installs'


# Do NOT modify this block of code
google_playstore_df['is_good_rating'] = google_playstore_df.loc[:, 'Rating'].apply(lambda x: 1 if x >= 4 else 0) # creating rating category


# Do NOT modify this block of code
google_playstore_df.head() # checking the head of the cleaned data
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Gen
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19000000.0	10k+	Free	0.0	Everyone	Art & De
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14000000.0	500k+	Free	0.0	Everyone	Design;Pret
	U Launcher Lite –									

Q2a

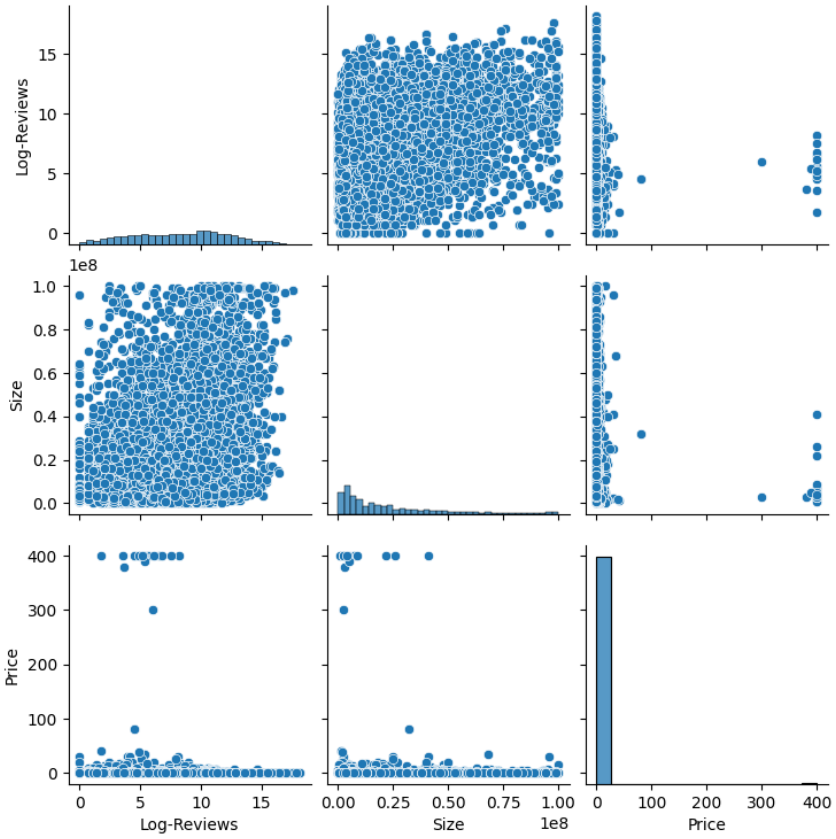
```
continuous_cols = ['Rating', 'Log-Reviews', 'Size', 'Price']

## Your code starts here ##

#filtering df for paid apps
paid_apps_df = google_playstore_df[google_playstore_df['Type'] == 'Paid']

#finding out the corr matrix for the cont features
corr_matrix = paid_apps_df[continuous_cols].corr()

#making a mask for the upper triangle, so that only one half of the symmetric corr matrix is displayed
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

#setting up matplotlib fig
plt.figure(figsize=(8, 6))

#drawing heat map w/ mask and correct aspect ratio
sns.heatmap(corr_matrix, mask=mask, cmap='coolwarm', vmax=.3, center=0, square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

## end ##
```

```
<Axes: >
```



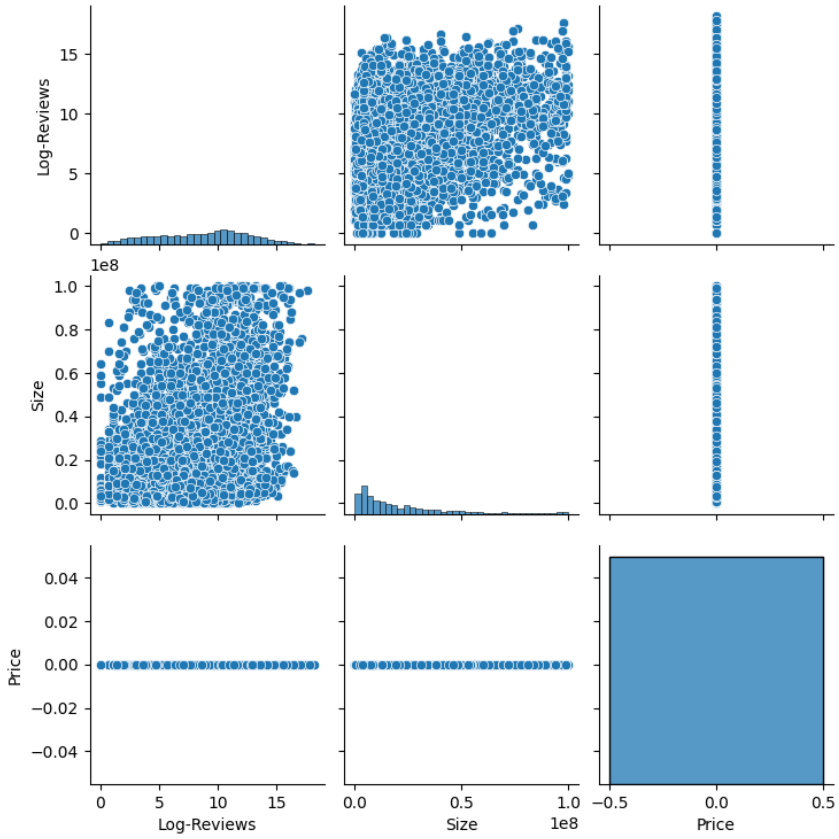
Q2b

```
## Your code starts here ##

#selecting cont features excl Rating
continuous_cols_excl_rating = [col for col in continuous_cols if col != 'Rating']

#pairplot for the data above
sns.pairplot(google_playstore_df[continuous_cols_excl_rating])
plt.show()

## end ##
```



Q2c

```
## Your code starts here ##

#filtering df for free apps
free_apps_df = google_playstore_df[google_playstore_df['Type'] == 'Free']

#pairplot for the data above
sns.pairplot(free_apps_df[continuous_cols_excl_rating])
plt.show()

## end ##
```



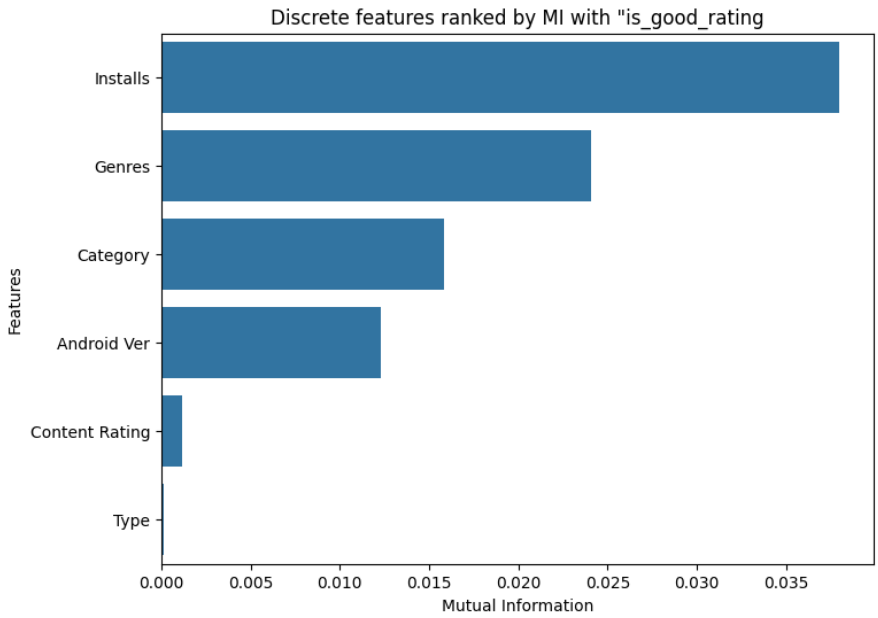
Q2d

```
# Do NOT modify this block of code
google_playstore_df.nunique() # checking the number of unique values per column
```

```
App               8190
Category            33
Rating              39
Reviews           5990
Size               411
Installs            19
Type                 2
Price               73
Content Rating       6
Genres             115
Last Updated      1299
Current Ver       2638
Android Ver         31
Log-Reviews       5990
is_good_rating       2
dtype: int64
```

```
# Do NOT modify this block of code

def cal_PMI(var: str, target_var: str = "is_good_rating", data: pd.DataFrame = google_playstore_df) -> pd.DataFrame:
    """function to calculate PMI"""
    contingency_table = pd.crosstab(index=data[var], columns=data[target_var]) # create a contingency table

    n = contingency_table.sum().sum() # total number of data points

    p_of_x_y =  contingency_table / n # calculate joint probability P(x,y)
    p_of_x_y = p_of_x_y.replace(0, 1E-5) # replace zeros because log(0) is undefined

    p_of_x = contingency_table.sum(axis=1) / n # calculate marginal probability P(x)
    p_of_y = contingency_table.sum(axis=0) / n # calculate marginal probability P(y)

    p_of_y_given_x = p_of_x_y.div(p_of_x, axis=0) # calculate conditional probability P(y|x)
    pmi_df = np.log(p_of_y_given_x.div(p_of_y, axis=1)) # calculate PMI for all pairs of x and y

    return pmi_df
```

```
# Do NOT modify this block of code
def cal_MI(var: str, target_var: str = "is_good_rating", data: pd.DataFrame = google_playstore_df, per_X: bool = False) -> float|pd.Series:
    """function to calculate MI"""
    contingency_table = pd.crosstab(index=data[var], columns=data[target_var]) # create a contingency table

    n = contingency_table.sum().sum() # total number of data points

    p_of_x_y =  contingency_table / n # calculate joint probability P(x,y)

    pmi_df = cal_PMI(var, target_var=target_var, data=data) # calculate PMI for all pairs of x and y

    if per_X: # if True, return MI of each category in x with target variable
        return (p_of_x_y * pmi_df).sum(axis=1)

    return (p_of_x_y * pmi_df).sum().sum() # else return MI of x and y


discrete_cols = ['Category', 'Installs', 'Type', 'Content Rating', 'Genres', 'Android Ver']

## Your code starts here ##

#calc MI for each discr feature
mi_score = {col: cal_MI(col) for col in discrete_cols}

#sort by MI score; idx 1 is the val in key-val pair
sorted_mi = sorted(mi_score.items(), key=lambda x: x[1], reverse=True)

#data for bar plot; * unpacks the list for it to pass through zip to pass them as separate arguments
features, scores = zip(*sorted_mi)

#bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x=scores, y=features)
plt.title('Discrete features ranked by MI with "is_good_rating')
plt.xlabel('Mutual Information')
plt.ylabel('Features')
plt.show()

## end ##
```



Q2e

```
## Your code starts here ##

#setting up matplotlib
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

#freq analysis subplot; indx 0 indicates that it is the first subplot
install_count = google_playstore_df['Installs'].value_counts()
sns.barplot(x=install_count.values, y=install_count.index, ax=axes[0])
axes[0].set_title('Frequency of Installs')
axes[0].set_xlabel('Frequency')
axes[0].set_ylabel('Installs')

#MI analysis subplot; per_X makes sure that MI is calculated for each Installs; inplace updates the original df
mi_score_installs = cal_MI('Installs', per_X=True)
mi_score_installs.sort_values(ascending=False, inplace=True)
sns.barplot(x=mi_score_installs.values, y=mi_score_installs.index, ax=axes[1])
```