Day 16 of Training at Ansh Info Tech

Topics Covered

Seaborn

- **Seaborn Plots**
  - `countplot` : Shows the counts of observations in each categorical bin using bars.
  - `kdeplot` : Plots the Kernel Density Estimation of the data.
  - `jointplot` : Draws a plot of two variables with bivariate and univariate graphs.
  - `distplot` : Displays a histogram with a line on it.
  - `heatmap` : Displays data in a matrix form using different colors.
  - `histplot` : Plots a histogram of the data.
  - `lmplot` : Draws a linear regression model.
  - `boxplot` : Displays the distribution of data based on a five-number summary.
  - `pairplot` : Creates a grid of Axes such that each numeric variable in the data will be shared across the y-axes across a single row and the x-axes across a single column.

Intro to OpenCV

- **Grayscale an Image**: Grayscaling is the process of converting an image from other color spaces (RGB, CMYK, etc.) to shades of gray.
- **Edge Detection**: Edge detection is a technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness.
- **Object Detection with Haar Cascades**: Haar cascades are classifiers used to detect objects for which they have been trained, such as faces or eyes. They are trained from a lot of positive and negative images before they can be used for object detection.

Summary

Seaborn

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Intro to OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It contains more than 2500 optimized algorithms for computer vision and machine learning.

```
Start coding or generate with AI.
```

```
#seaborn
#it is an data visualization library
#it is based on matplotlib
```

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

```
sns.get_dataset_names()
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri',
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'taxis',
 'tips',
 'titanic']
```

```
tips=sns.load_dataset('tips')
```

```
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
tips.tail()
```

	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

```
tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

```
tips.describe()
```

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

```
tips.columns
```

```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'], dtype='object')
```

```
tips['day'].unique()
```

```
['Sun', 'Sat', 'Thur', 'Fri']
Categories (4, object): ['Thur', 'Fri', 'Sat', 'Sun']
```

```
tips['total_bill'].unique()
```

```
array([16.99, 10.34, 21.01, 23.68, 24.59, 25.29,  8.77, 26.88, 15.04,
       14.78, 10.27, 35.26, 15.42, 18.43, 14.83, 21.58, 10.33, 16.29,
       16.97, 20.65, 17.92, 20.29, 15.77, 39.42, 19.82, 17.81, 13.37,
       12.69, 21.7 , 19.65,  9.55, 18.35, 15.06, 20.69, 17.78, 24.06,
       16.31, 16.93, 18.69, 31.27, 16.04, 17.46, 13.94,  9.68, 30.4 ,
       18.29, 22.23, 32.4 , 28.55, 18.04, 12.54, 10.29, 34.81,  9.94,
       25.56, 19.49, 38.01, 26.41, 11.24, 48.27, 13.81, 11.02, 17.59,
       20.08, 16.45,  3.07, 20.23, 15.01, 12.02, 17.07, 26.86, 25.28,
       14.73, 10.51, 27.2 , 22.76, 17.29, 19.44, 16.66, 10.07, 32.68,
       15.98, 34.83, 13.03, 18.28, 24.71, 21.16, 28.97, 22.49,  5.75,
       16.32, 22.75, 40.17, 27.28, 12.03, 12.46, 11.35, 15.38, 44.3 ,
       22.42, 20.92, 15.36, 20.49, 25.21, 18.24, 14.31, 14.  ,  7.25,
       38.07, 23.95, 25.71, 17.31, 29.93, 10.65, 12.43, 24.08, 11.69,
       13.42, 14.26, 15.95, 12.48, 29.8 ,  8.52, 14.52, 11.38, 22.82,
       19.08, 20.27, 11.17, 12.26, 18.26,  8.51, 14.15, 16.  , 13.16,
       17.47, 34.3 , 41.19, 27.05, 16.43,  8.35, 18.64, 11.87,  9.78,
        7.51, 14.07, 13.13, 17.26, 24.55, 19.77, 29.85, 48.17, 25.  ,
       13.39, 16.49, 21.5 , 12.66, 16.21, 17.51, 24.52, 20.76, 31.71,
       10.59, 10.63, 50.81, 15.81, 31.85, 16.82, 32.9 , 17.89, 14.48,
        9.6 , 34.63, 34.65, 23.33, 45.35, 23.17, 40.55, 20.9 , 30.46,
       18.15, 23.1 , 15.69, 19.81, 28.44, 15.48, 16.58,  7.56, 43.11,
       13.  , 13.51, 18.71, 12.74, 16.4 , 20.53, 16.47, 26.59, 38.73,
       24.27, 12.76, 30.06, 25.89, 48.33, 13.27, 28.17, 12.9 , 28.15,
       11.59,  7.74, 30.14, 12.16,  8.58, 16.27, 10.09, 20.45, 13.28,
```

```
       22.12, 24.01, 11.61, 10.77, 15.53, 12.6 , 32.83, 35.83, 29.03,
       27.18, 22.67, 17.82, 18.78])
```

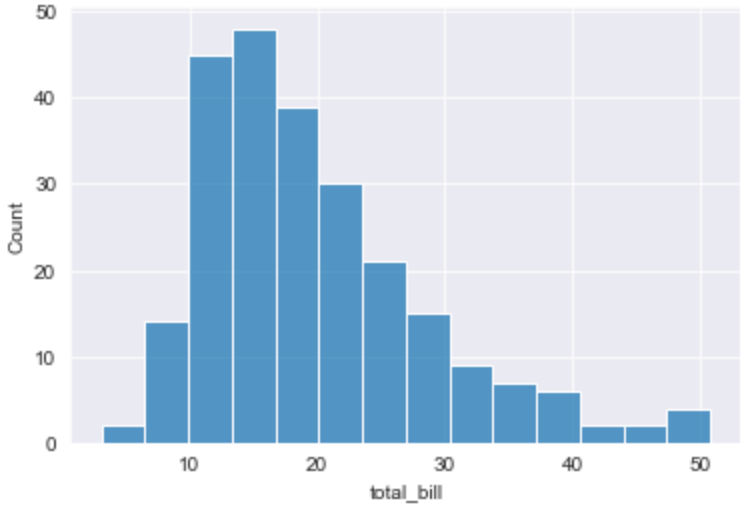
distribution plots

Univariate

```
sns.set_style('darkgrid')
```

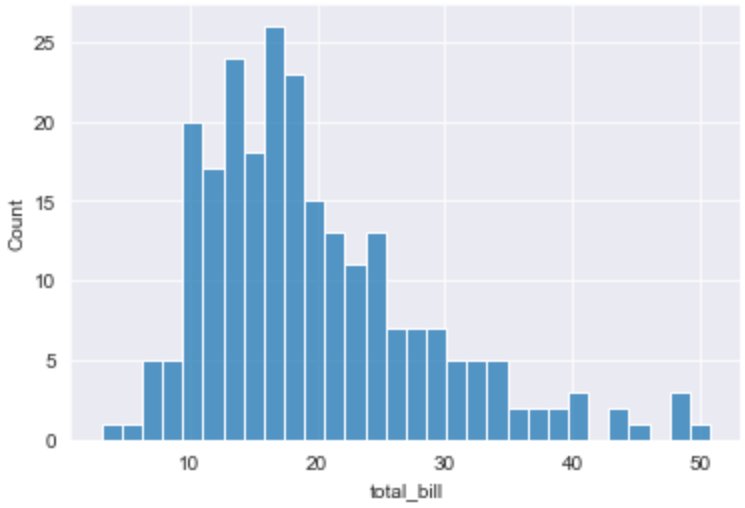
```
sns.histplot(x='total_bill',data=tips)
```

```
<AxesSubplot:xlabel='total_bill', ylabel='Count'>
```



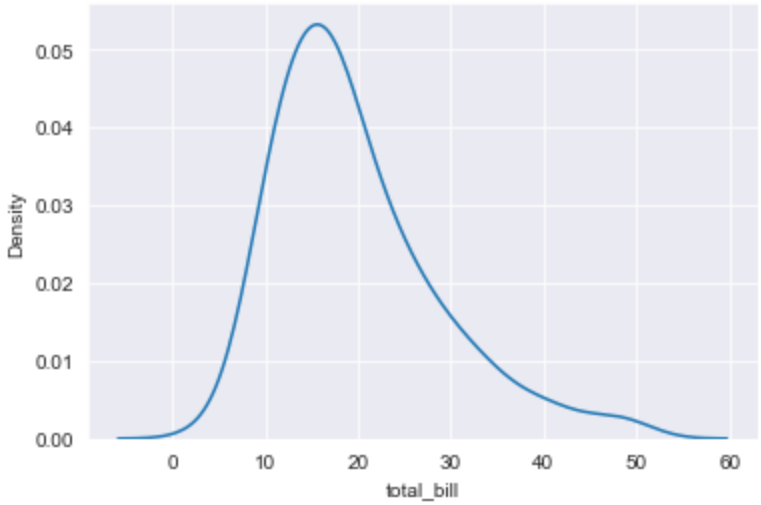

```
sns.histplot(x='total_bill',data=tips,bins=30)
```

```
<AxesSubplot:xlabel='total_bill', ylabel='Count'>
```



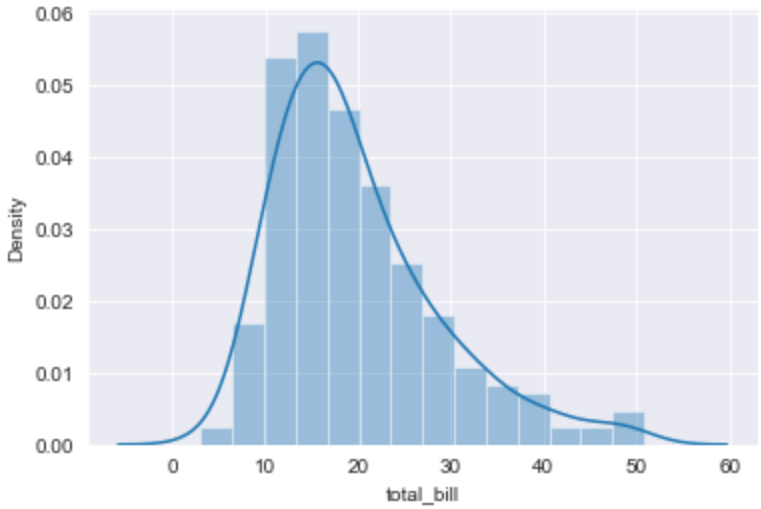

```
sns.kdeplot(x='total_bill', data=tips)
```

```
<AxesSubplot:xlabel='total_bill', ylabel='Density'>
```



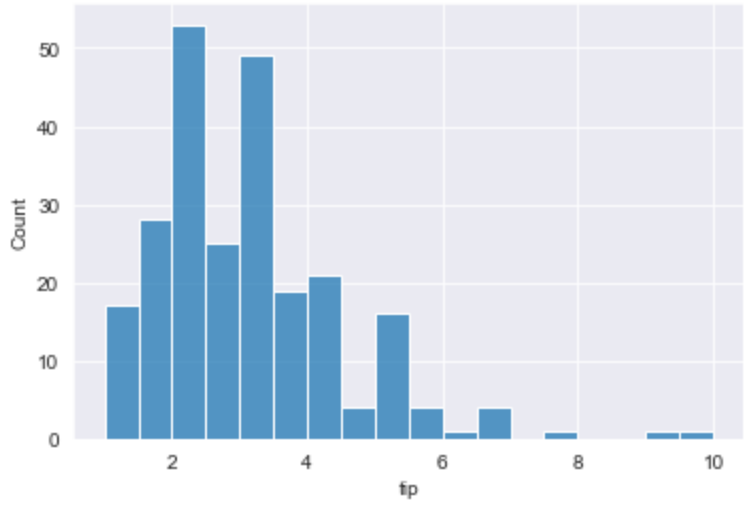

```
sns.distplot(tips['total_bill'])
```

```
/Volumes/Kriti-1/Applications/anaconda3/lib/python3.9/site-packages/seaborn/distributior
  warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='total_bill', ylabel='Density'>
```



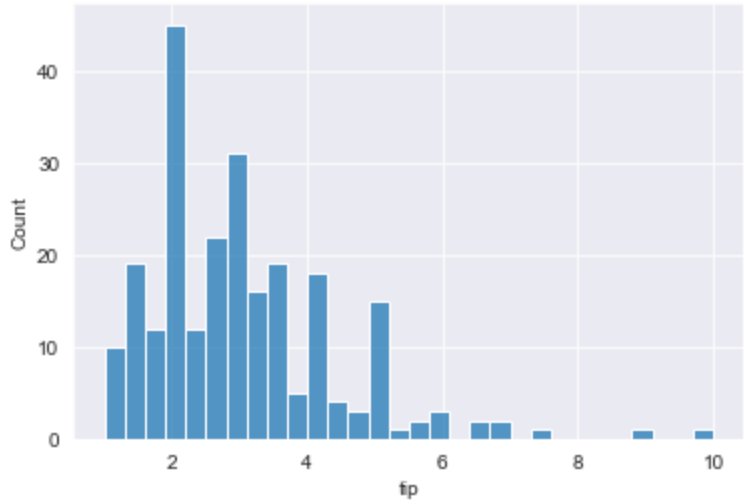

```
sns.histplot(x='tip',data=tips)
```

<AxesSubplot:xlabel='tip', ylabel='Count'>



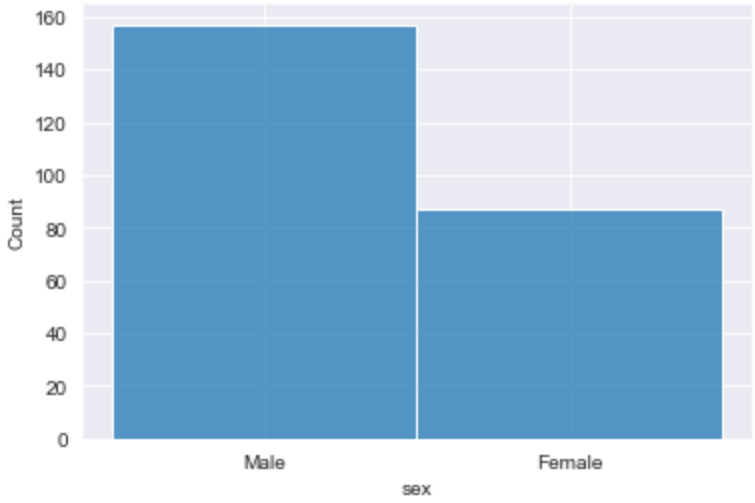

```
sns.histplot(x='tip',data=tips,bins=30)
```

<AxesSubplot:xlabel='tip', ylabel='Count'>



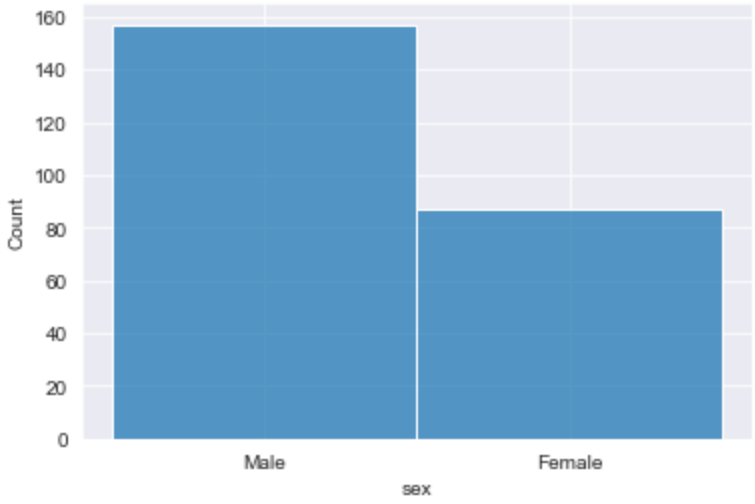

```
sns.histplot(x='sex',data=tips)
```

<AxesSubplot:xlabel='sex', ylabel='Count'>



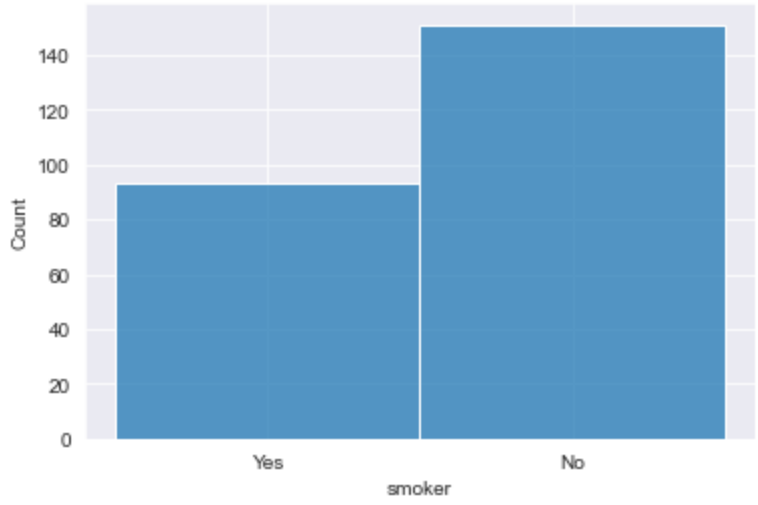

```
sns.histplot(x='sex',data=tips,bins=10)
```

<AxesSubplot:xlabel='sex', ylabel='Count'>



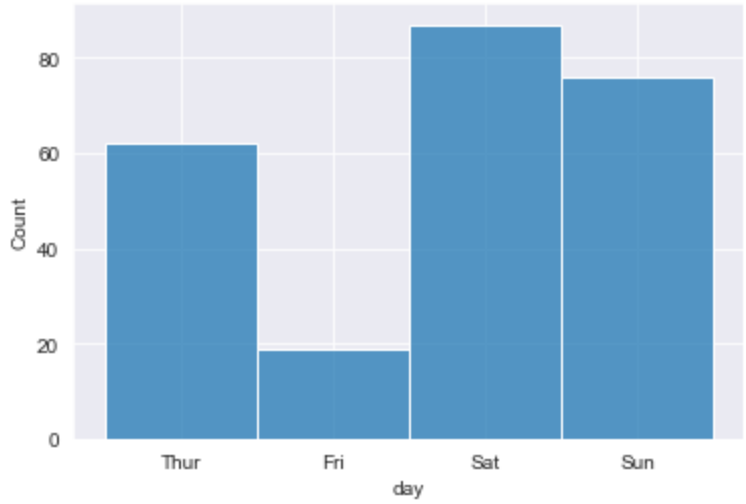

```
sns.histplot(x='smoker',data=tips)
```

```
<AxesSubplot:xlabel='smoker', ylabel='Count'>
```



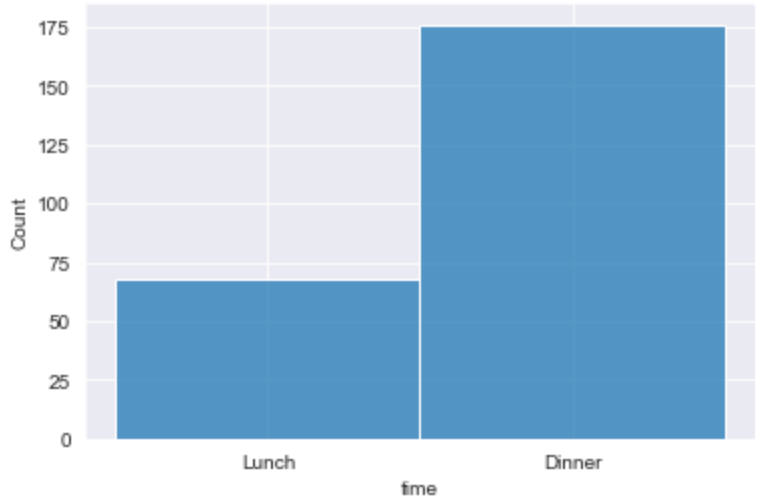

```
sns.histplot(x='day',data=tips)
```

```
<AxesSubplot:xlabel='day', ylabel='Count'>
```



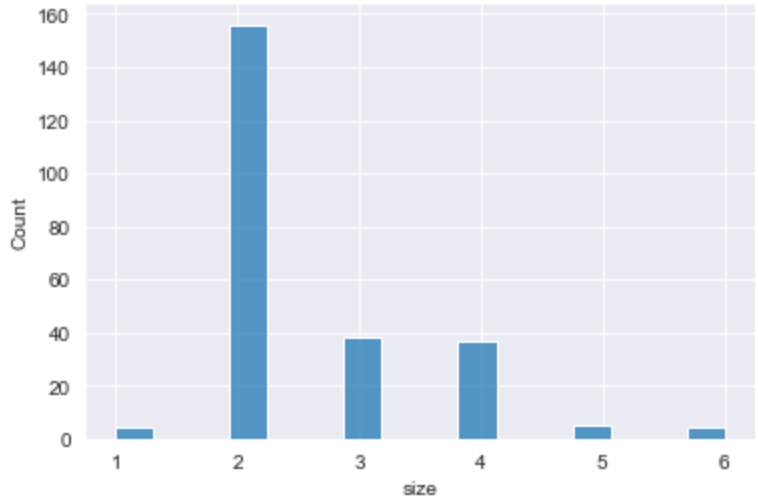

```
sns.histplot(x='time',data=tips)
```

<AxesSubplot:xlabel='time', ylabel='Count'>



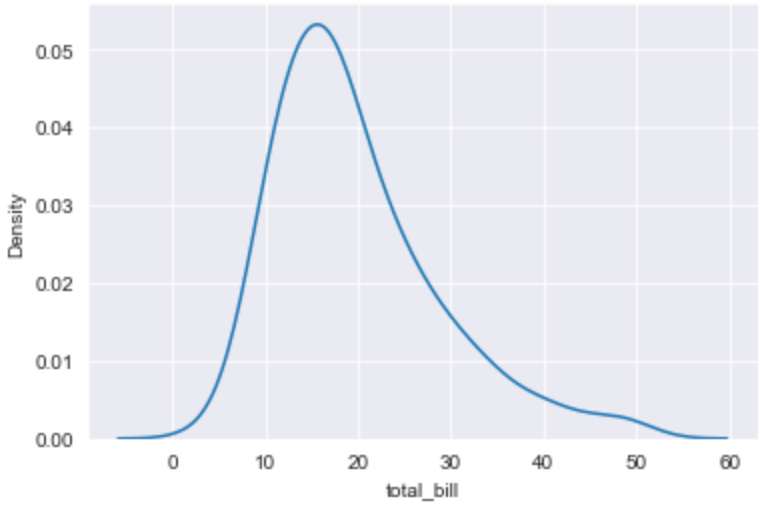

```
sns.histplot(x='size',data=tips)
```

<AxesSubplot:xlabel='size', ylabel='Count'>



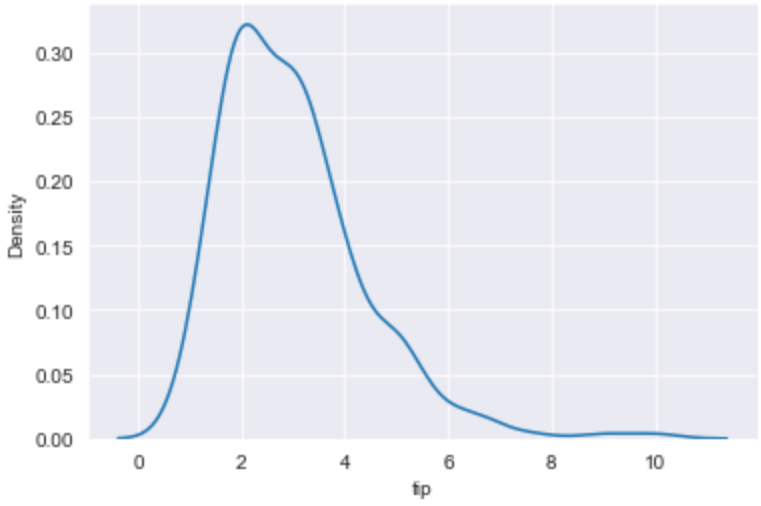

```
sns.kdeplot(x='total_bill',data=tips)
```

<AxesSubplot:xlabel='total_bill', ylabel='Density'>



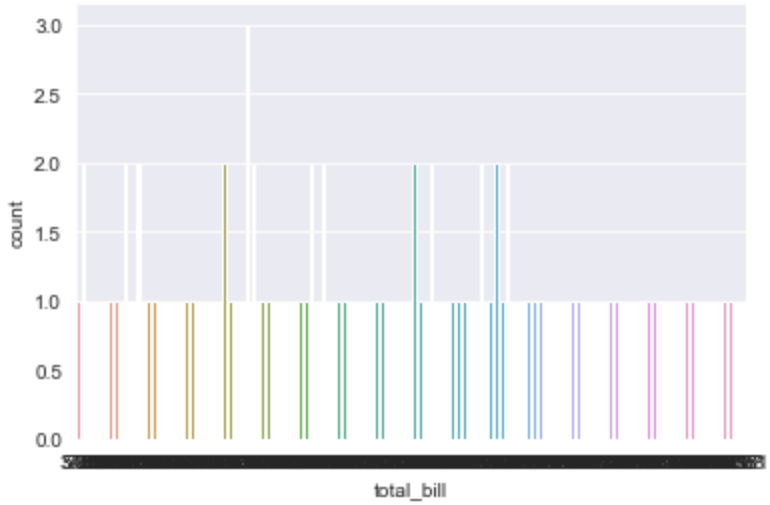

```
sns.kdeplot(x='tip',data=tips)
```

<AxesSubplot:xlabel='tip', ylabel='Density'>



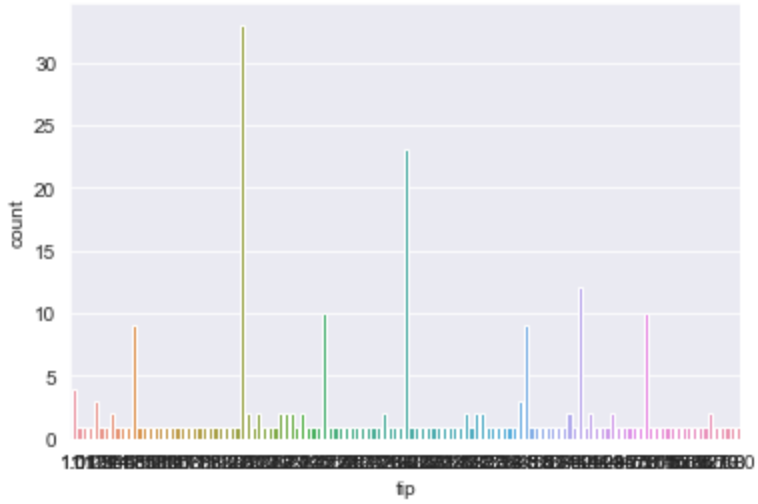

```
sns.countplot(x='total_bill',data=tips)
```

<AxesSubplot:xlabel='total_bill', ylabel='count'>



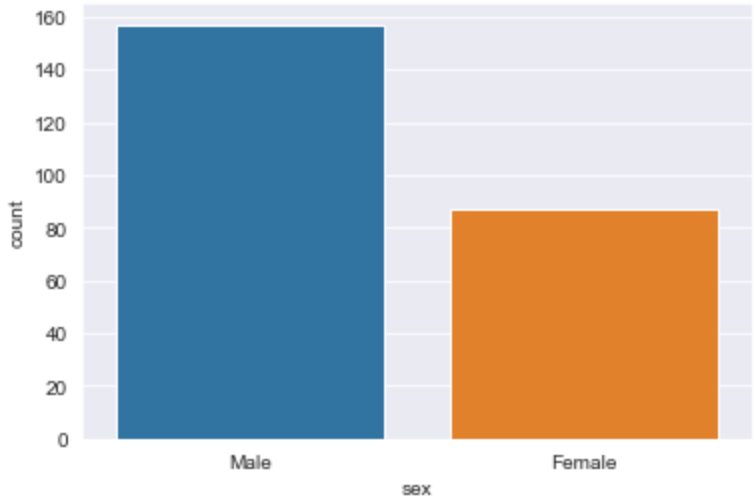

```
sns.countplot(x='tip',data=tips)
```

<AxesSubplot:xlabel='tip', ylabel='count'>



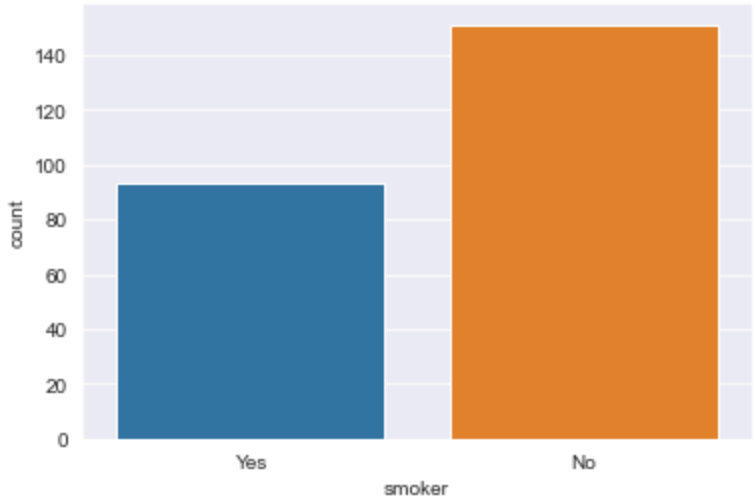

```
sns.countplot(x='sex',data=tips)
```

<AxesSubplot:xlabel='sex', ylabel='count'>



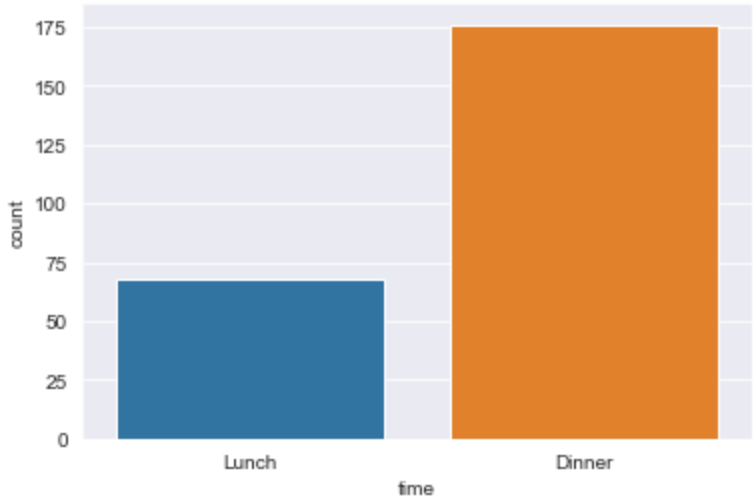

```
sns.countplot(x='smoker',data=tips)
```

<AxesSubplot:xlabel='smoker', ylabel='count'>



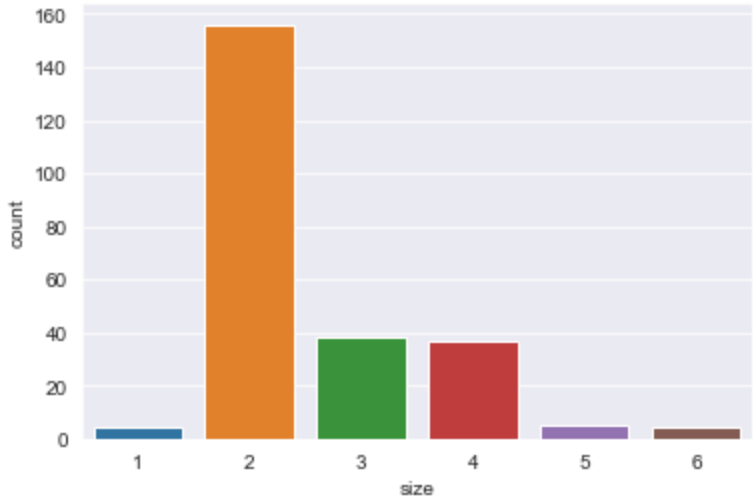

```
sns.countplot(x='time',data=tips)
```

<AxesSubplot:xlabel='time', ylabel='count'>



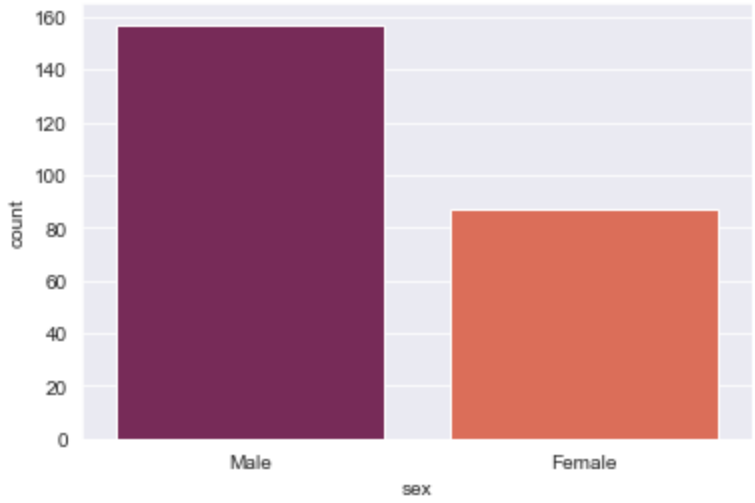

```
sns.countplot(x='size',data=tips)
```

<AxesSubplot:xlabel='size', ylabel='count'>



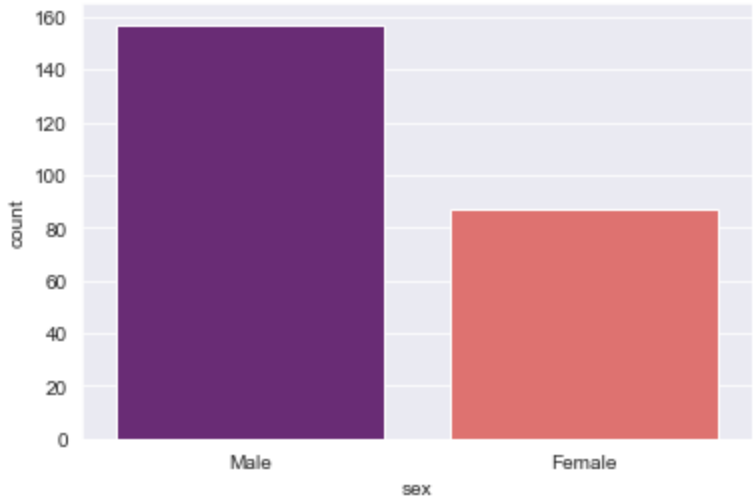

```
sns.countplot(x='sex',data=tips,palette='rocket')
```

<AxesSubplot:xlabel='sex', ylabel='count'>




```
sns.countplot(x='sex',data=tips,palette='magma')
```

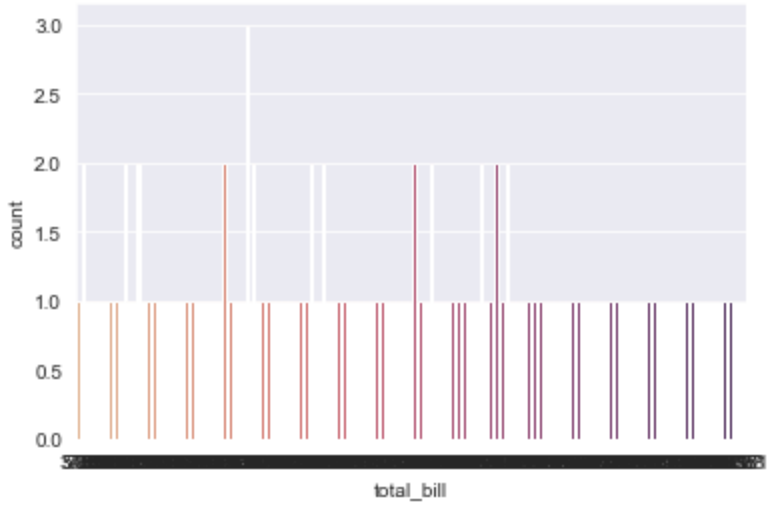
<AxesSubplot:xlabel='sex', ylabel='count'>




```
#flare
#magma
#crest
#viridis
#cubehelix
```

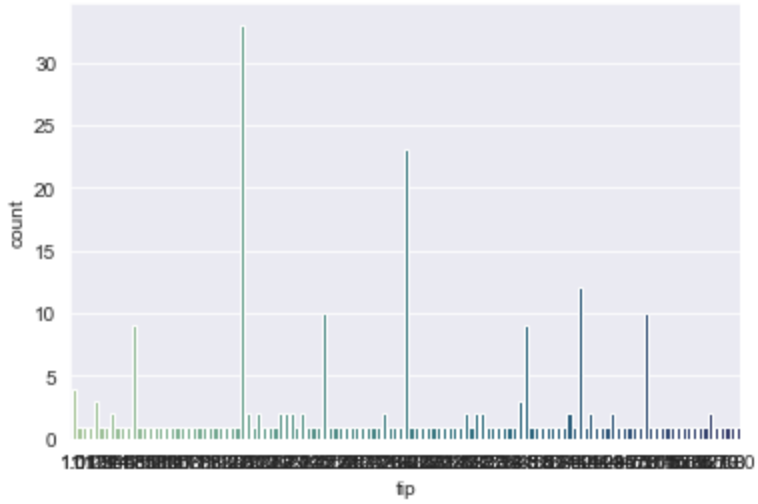
```
sns.countplot(x='total_bill',data=tips,palette='flare')
```

<AxesSubplot:xlabel='total_bill', ylabel='count'>



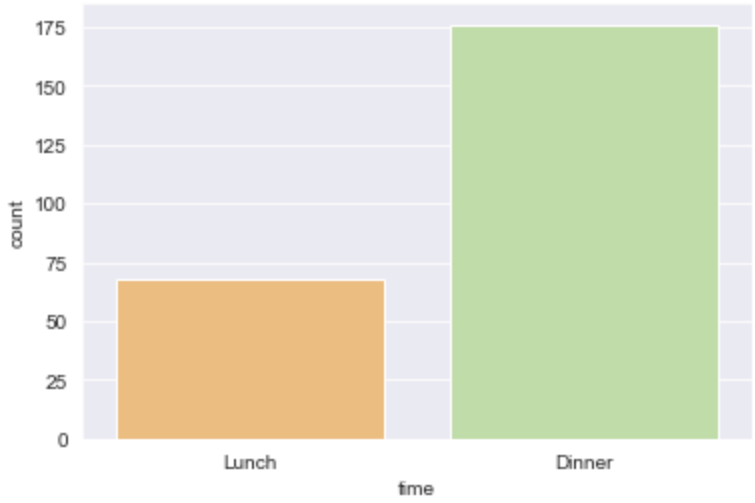

```
sns.countplot(x='tip',data=tips,palette='crest')
```

<AxesSubplot:xlabel='tip', ylabel='count'>



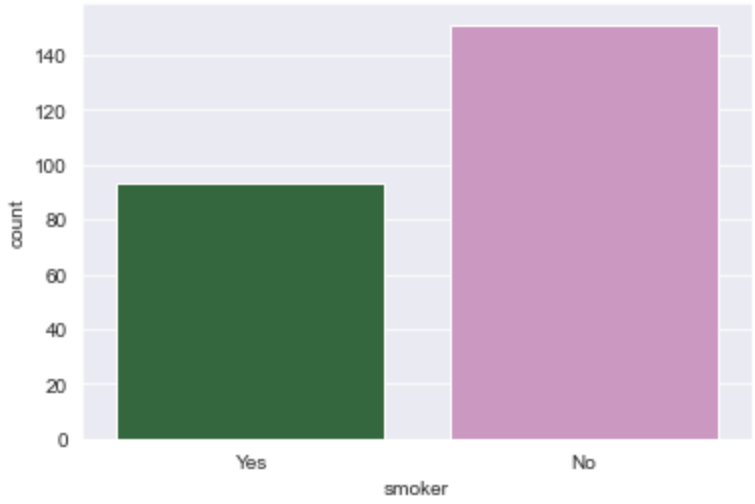

```
sns.countplot(x='time',data=tips,palette='Spectral')
```

```
<AxesSubplot:xlabel='time', ylabel='count'>
```




```
sns.countplot(x='smoker',data=tips,palette='cubehelix')
```

```
<AxesSubplot:xlabel='smoker', ylabel='count'>
```

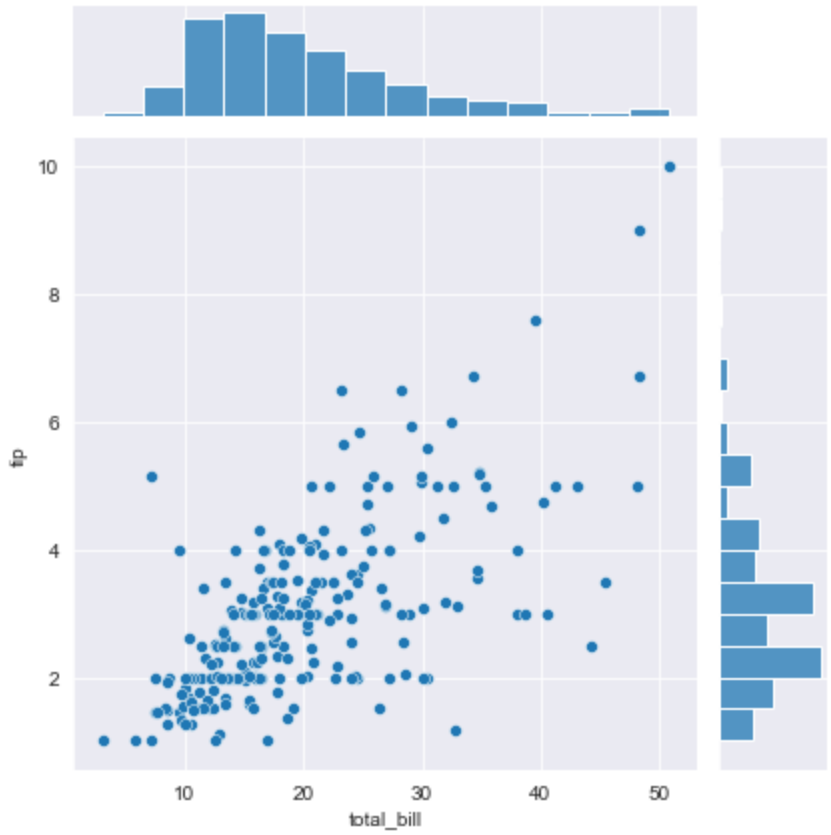


relational plots

Bivariate

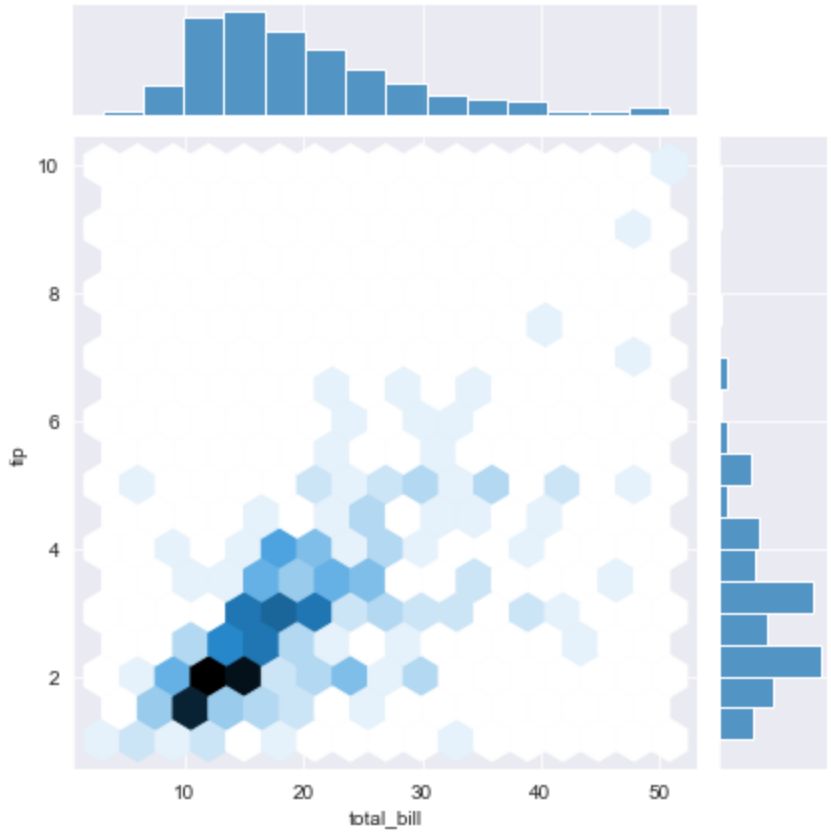
```
sns.jointplot(x='total_bill',y='tip',data=tips)
```

<seaborn.axisgrid.JointGrid at 0x7fbfe52dcc40>



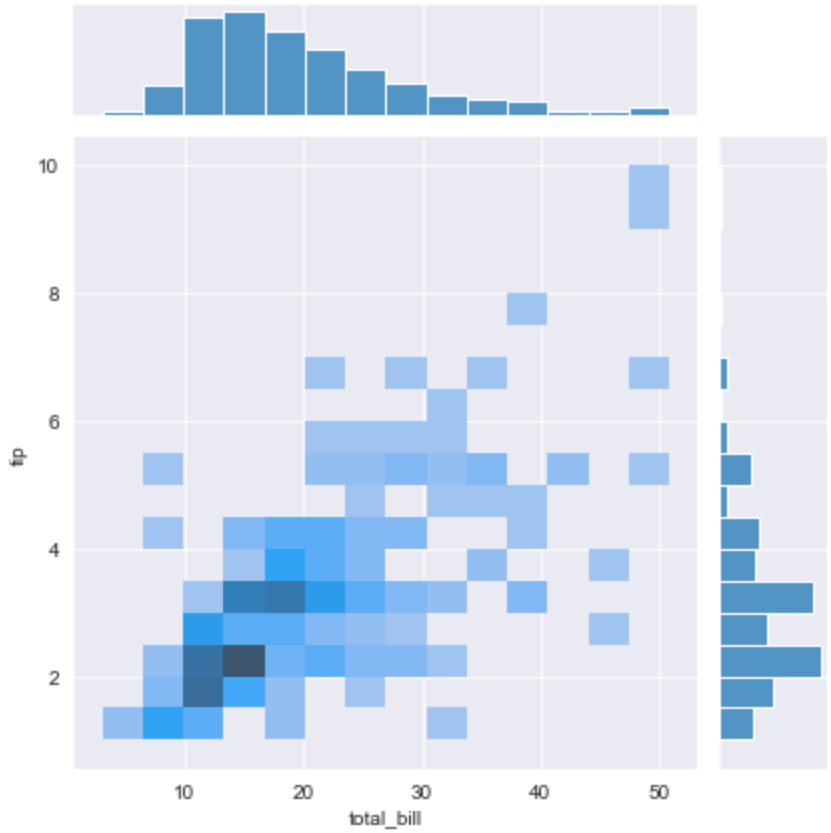

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind="hex")
```

<seaborn.axisgrid.JointGrid at 0x7fbfe5d46b20>



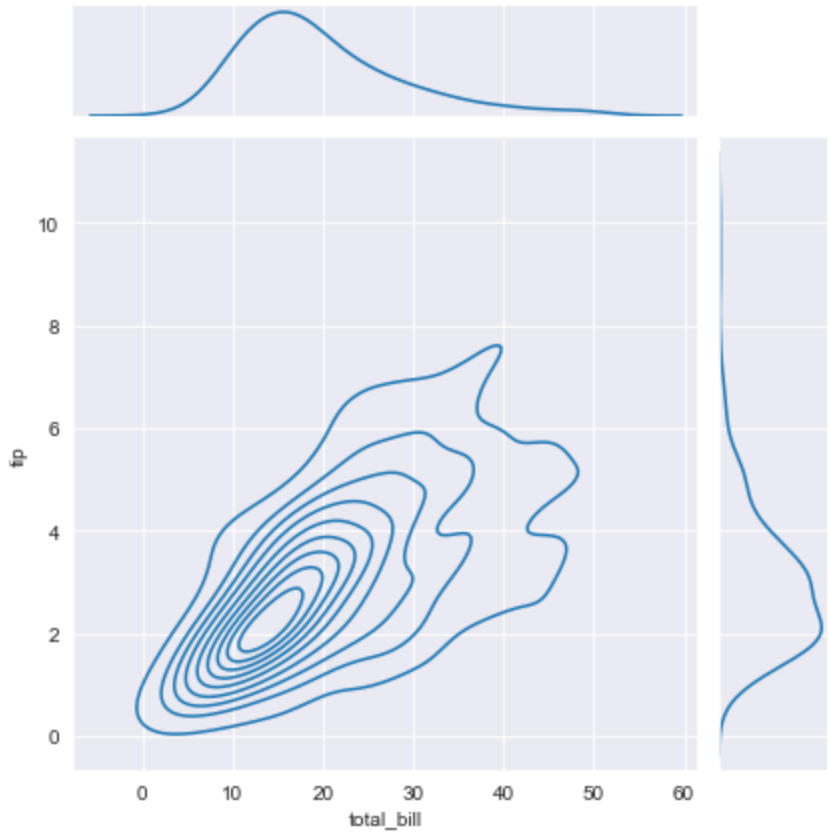
```
sns.jointplot(x='total_bill',y='tip',data=tips,kind="hist")
```

```
<seaborn.axisgrid.JointGrid at 0x7fbfe5bff460>
```



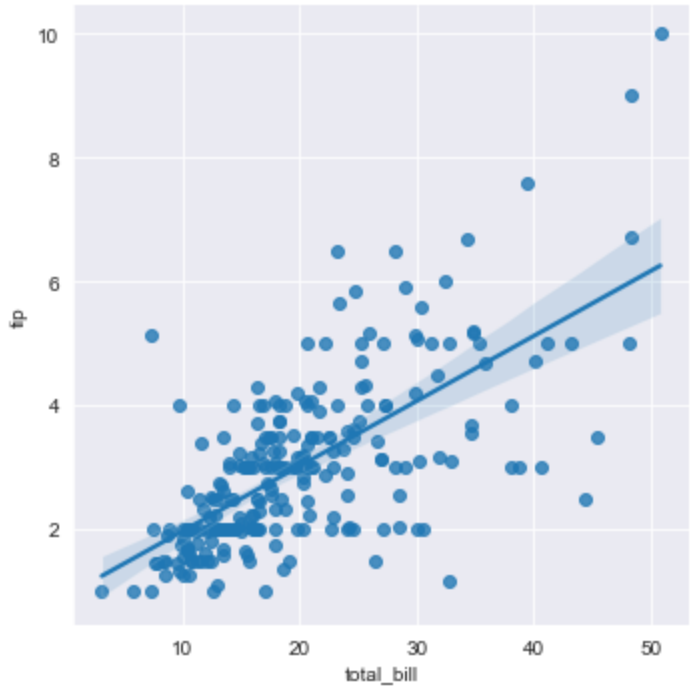

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind="kde")
```

```
<seaborn.axisgrid.JointGrid at 0x7fbfe6225cd0>
```



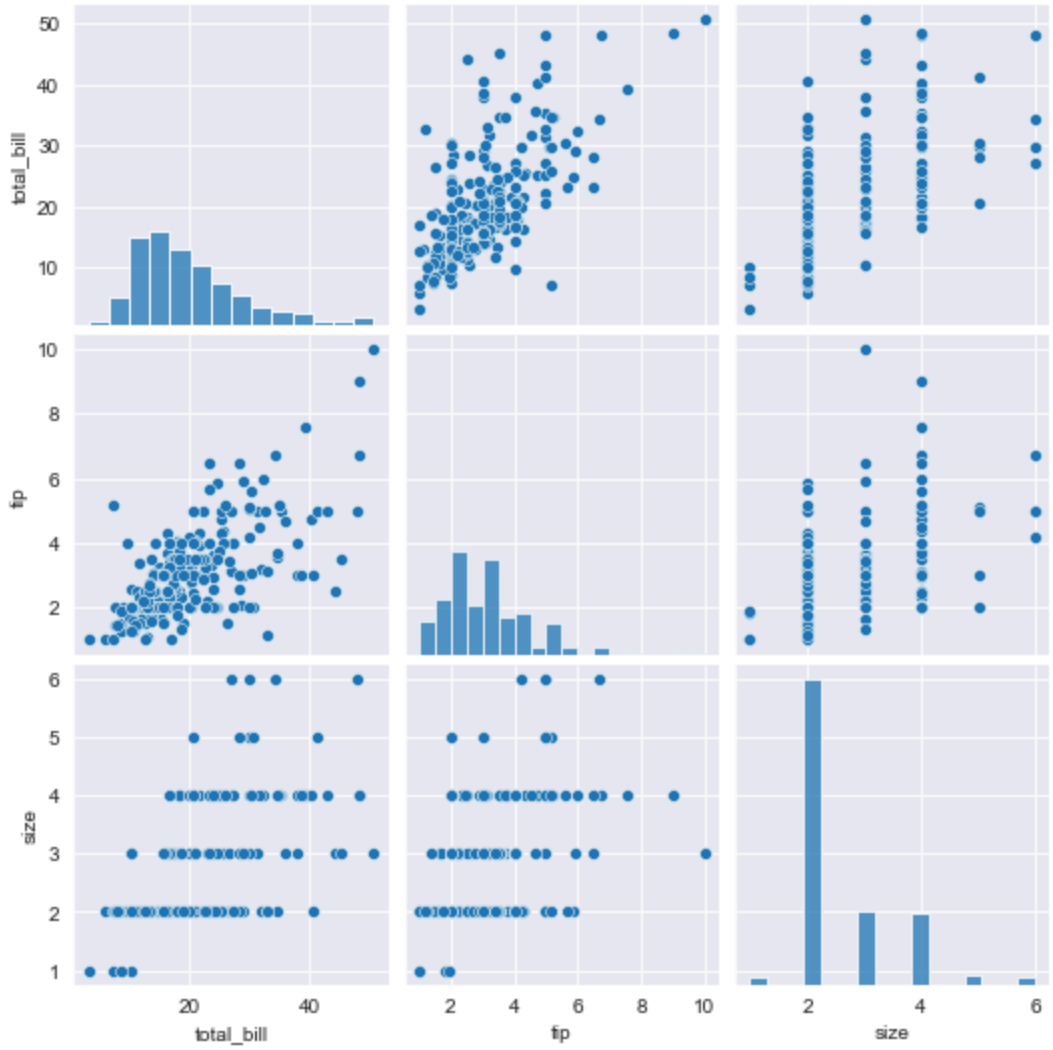

```
sns.lmplot(x='total_bill',y='tip',data=tips)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fbfe64fb910>
```



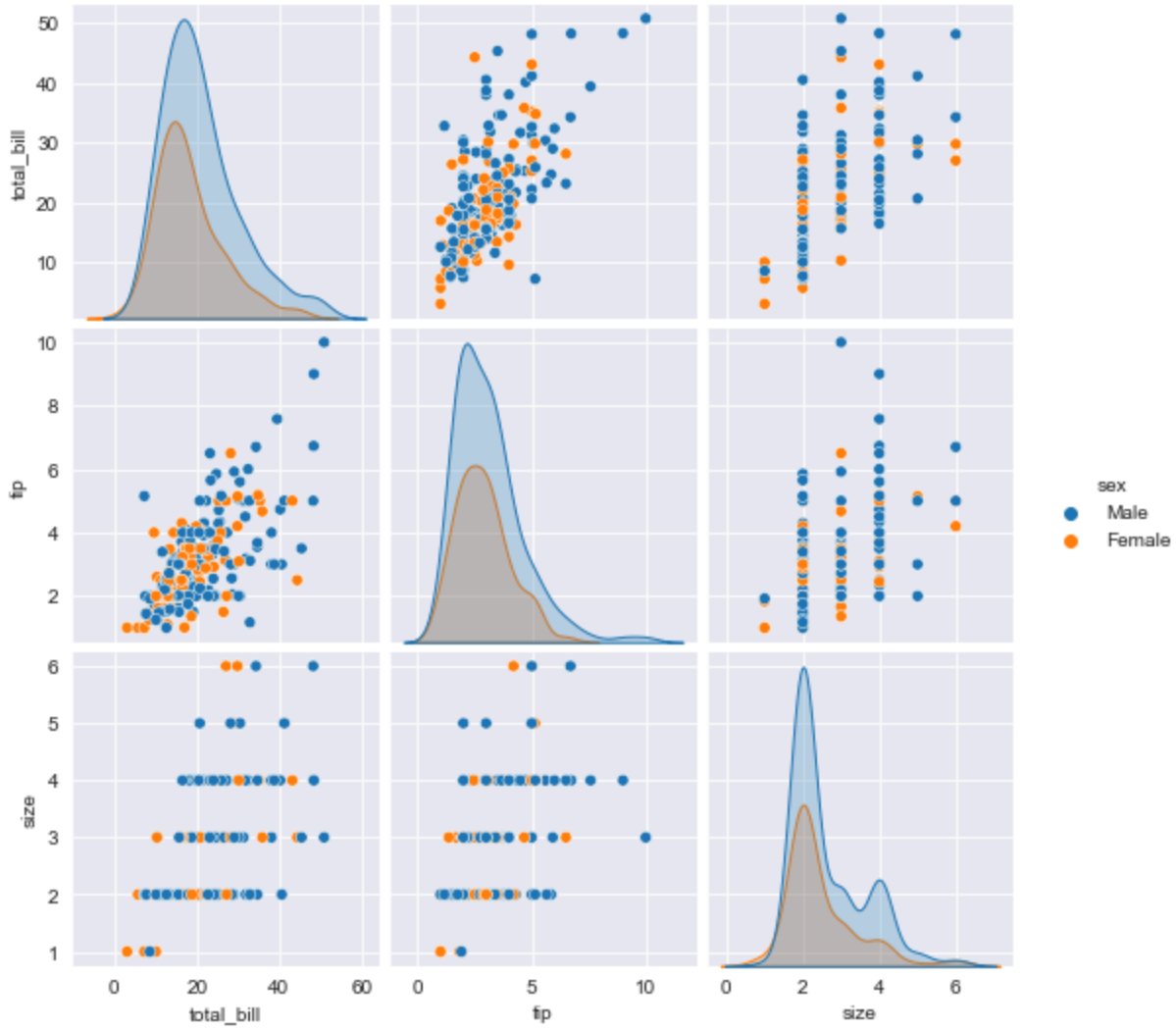

```
sns.pairplot(tips)
```

<seaborn.axisgrid.PairGrid at 0x7fbfe6532670>



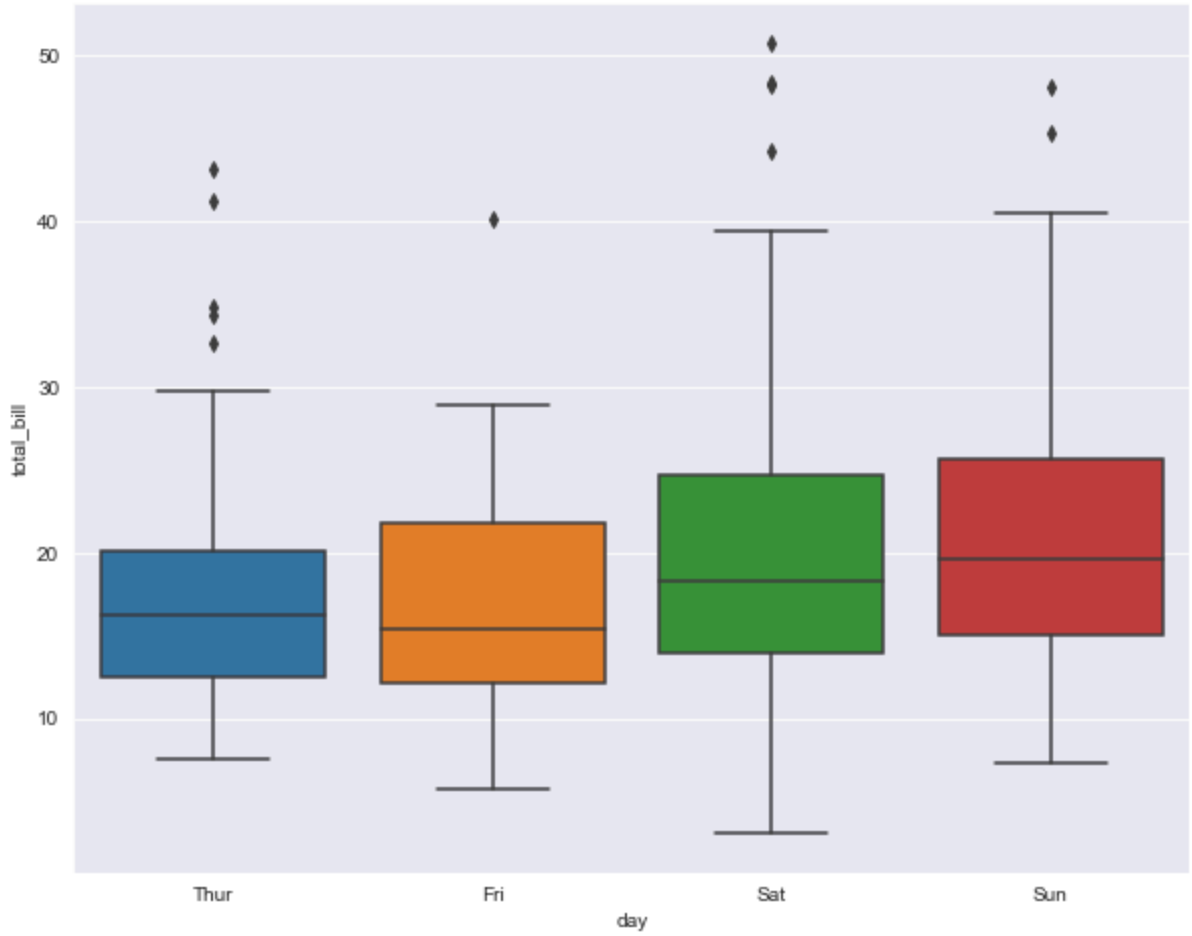

```
sns.pairplot(tips, hue='sex')
```

```
<seaborn.axisgrid.PairGrid at 0x1ef54affe50>
```




```
plt.figure(figsize=(10, 8))
sns.boxplot(x='day', y='total_bill', data=tips)
```

<AxesSubplot:xlabel='day', ylabel='total_bill'>



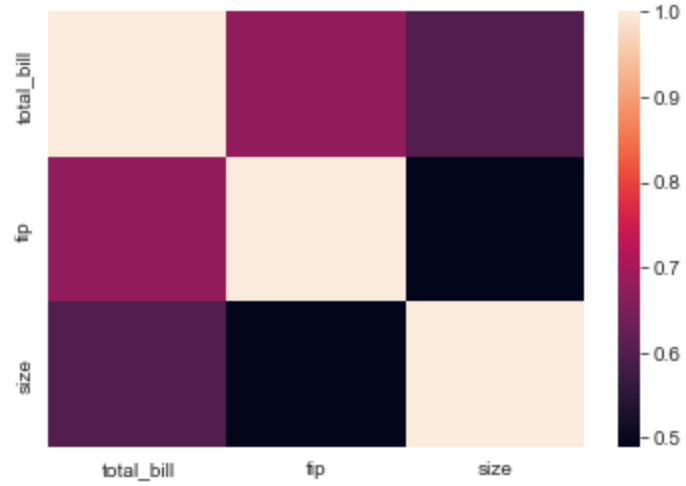
matrix plots

```
tips.corr()
```

	total_bill	tip	size
total_bill	1.000000	0.675734	0.598315
tip	0.675734	1.000000	0.489299
size	0.598315	0.489299	1.000000

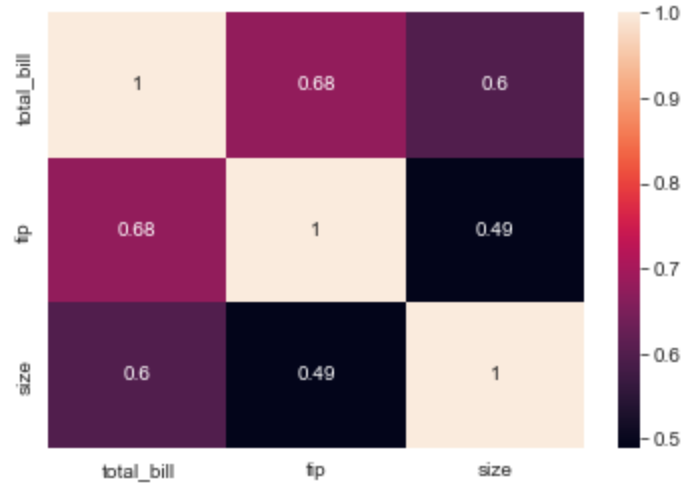
```
sns.heatmap(tips.corr())
```

```
<AxesSubplot:>
```




```
sns.heatmap(tips.corr(), annot=True)
```

```
<AxesSubplot:>
```



"OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision."

"Originally developed by Intel, it was later supported by Willow Garage and is now maintained by Itseez."

"OpenCV has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS."

"OpenCV leans mostly towards real-time image processing, so processing speed is a primary design concern."

"Some of the key features of OpenCV include:"

"* Face detection and recognition"

"* Feature matching"

"* Object identification"

"* Motion tracking"

"* Image segmentation"

"* 3D point cloud processing"

"* Machine learning"

"OpenCV is used in a wide variety of applications, including:"

"* Security and surveillance"

"* Medical imaging"

"* Robotics"

"* Automotive"

"* Augmented reality"

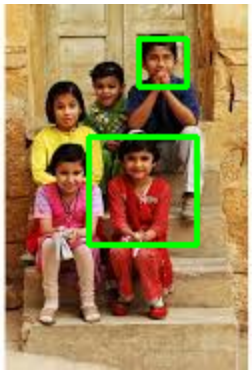
"* Virtual reality"

"* Games"

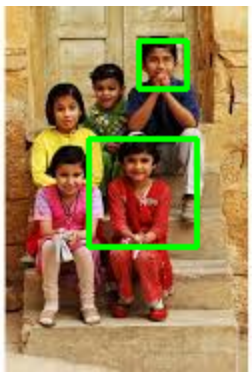
"OpenCV is a powerful library that can be used to solve a wide variety of computer vision problems."

"It is free and open source, making it a great choice for both commercial and non-commercial projects."

```
import cv2
from google.colab.patches import cv2_imshow

# Read an image
image = cv2.imread('image2.jpeg')

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = cv2.CascadeClassifier('haarcascade_frontalface_default.xml').detectMultiScale(gray)

# Draw a rectangle around each face
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the image with the detected faces
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



```
import cv2
import os
from google.colab.patches import cv2_imshow

# Path to the Haar cascade file
cascade_path = 'haarcascade_frontalface_default.xml'

# Check if the Haar cascade file exists
if not os.path.exists(cascade_path):
    raise FileNotFoundError(f"Haar cascade file '{cascade_path}' not found. Please download

# Read an image
image = cv2.imread('image2.jpeg')

# Check if the image is loaded successfully
if image is None:
    raise FileNotFoundError("Image file 'image2.jpeg' not found or could not be loaded.")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Load the Haar cascade classifier
face_cascade = cv2.CascadeClassifier(cascade_path)

# Check if the cascade classifier is loaded successfully
if face_cascade.empty():
    raise ValueError(f"Failed to load Haar cascade classifier from '{cascade_path}'.")

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray)

# Draw a rectangle around each face
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the image with the detected faces
cv2_imshow(image)
```



```
import cv2
import os
from google.colab.patches import cv2_imshow

# Path to the Haar cascade file
cascade_path = 'haarcascade_frontalface_default.xml'

# Check if the Haar cascade file exists
if not os.path.exists(cascade_path):
    raise FileNotFoundError(f"Haar cascade file '{cascade_path}' not found. Please download

# Read an image
image = cv2.imread('image2.jpeg')

# Check if the image is loaded successfully
if image is None:
    raise FileNotFoundError("Image file 'image2.jpeg' not found or could not be loaded.")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Load the Haar cascade classifier
face_cascade = cv2.CascadeClassifier(cascade_path)

# Check if the cascade classifier is loaded successfully
if face_cascade.empty():
    raise ValueError(f"Failed to load Haar cascade classifier from '{cascade_path}'.")

# Detect faces in the image with adjusted parameters
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3, minSize=(30, 30

# Draw a rectangle around each face
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the image with the detected faces
cv2_imshow(image)
```

```
import cv2
```