COURSE OUTCOME-2

PROGRAM NO-1

Aim: Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm.

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

```
df = pd.read_csv('/content/sonar_csv.csv')
```

```
df.head()
```

```
   attribute_1  attribute_2  attribute_3  ...  attribute_59
attribute_60  Class
0       0.0200       0.0371       0.0428  ...        0.0090
0.0032   Rock
1       0.0453       0.0523       0.0843  ...        0.0052
0.0044   Rock
2       0.0262       0.0582       0.1099  ...        0.0095
0.0078   Rock
3       0.0100       0.0171       0.0623  ...        0.0040
0.0117   Rock
4       0.0762       0.0666       0.0481  ...        0.0107
0.0094   Rock

[5 rows x 61 columns]
```

```
from sklearn.preprocessing import StandardScaler
#Preprocessing- standardscaler
```

```
sc=StandardScaler()
```

```
sc.fit(df.drop('Class',axis=1))
```

```
StandardScaler()
```

```
s = sc.transform(df.drop('Class',axis=1))
```

```
s
```

```
array([[-0.39955135, -0.04064823, -0.02692565, ...,  0.06987027,
         0.17167808, -0.65894689],
       [ 0.70353822,  0.42163039,  1.05561832, ..., -0.47240644,
        -0.44455424, -0.41985233],
       [-0.12922901,  0.60106749,  1.72340448, ...,  1.30935987,
         0.25276128,  0.25758223],
       ...,
       [ 1.00438083,  0.16007801, -0.67384349, ...,  0.90652575,
```

```
        -0.03913824, -0.67887143],
       [ 0.04953255, -0.09539176,  0.13480381, ..., -0.00759783,
        -0.70402047, -0.34015415],
       [-0.13794908, -0.06497869, -0.78861924, ..., -0.6738235 ,
        -0.29860448,  0.99479044]])
```

```
a = pd.DataFrame(s,columns = df.columns[:-1])
```

```
a
```

```
     attribute_1  attribute_2  ...  attribute_59  attribute_60
0      -0.399551    -0.040648  ...      0.171678     -0.658947
1       0.703538     0.421630  ...     -0.444554     -0.419852
2      -0.129229     0.601067  ...      0.252761      0.257582
3      -0.835555    -0.648910  ...     -0.639154      1.034640
4       2.050790     0.856537  ...      0.447361      0.576375
..           ...          ...  ...           ...           ...
203    -0.456232    -0.116681  ...      1.841992      1.831621
204     0.136733    -0.861801  ...     -0.282388      0.038412
205     1.004381     0.160078  ...     -0.039138     -0.678871
206     0.049533    -0.095392  ...     -0.704020     -0.340154
207    -0.137949    -0.064979  ...     -0.298604      0.994790

[208 rows x 60 columns]
```

training and testing

```
from sklearn.model_selection import train_test_split
```

```
x = a
```

```
y = df['Class']
```

```
y
```

```
0      Rock
1      Rock
2      Rock
3      Rock
4      Rock
       ...
203    Mine
204    Mine
205    Mine
206    Mine
207    Mine
Name: Class, Length: 208, dtype: object
```

```
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.3,random_state= 42)
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=1)
```

```
pred = knn.predict(x_test)
```

```
pred
```

```
array(['Mine', 'Mine', 'Rock', 'Rock', 'Mine', 'Rock', 'Mine', 'Mine',
       'Rock', 'Rock', 'Mine', 'Rock', 'Mine', 'Mine', 'Mine', 'Mine',
       'Mine', 'Mine', 'Rock', 'Rock', 'Mine', 'Mine', 'Mine', 'Mine',
       'Rock', 'Rock', 'Rock', 'Rock', 'Mine', 'Mine', 'Mine', 'Rock',
       'Rock', 'Mine', 'Mine', 'Mine', 'Rock', 'Mine', 'Mine', 'Mine',
       'Rock', 'Mine', 'Rock', 'Mine', 'Rock', 'Rock', 'Mine', 'Mine',
       'Rock', 'Rock', 'Rock', 'Mine', 'Rock', 'Rock', 'Rock', 'Rock',
       'Mine', 'Mine', 'Rock', 'Mine', 'Rock', 'Mine', 'Rock'],
      dtype=object)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

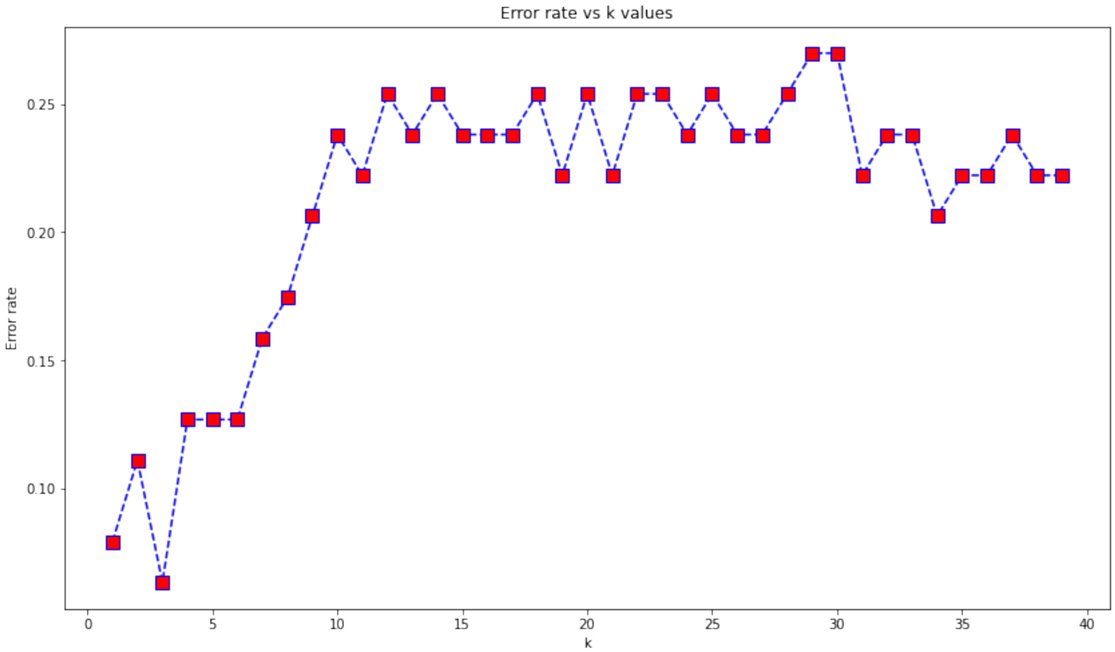
```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[32  3]
 [ 2 26]]
              precision    recall  f1-score   support

        Mine       0.94      0.91      0.93        35
        Rock       0.90      0.93      0.91        28

    accuracy                           0.92        63
   macro avg       0.92      0.92      0.92        63
weighted avg       0.92      0.92      0.92        63
```

```
#analysizing better k value through iterations
error_rate = []
for i in range(1,40):
  knn = KNeighborsClassifier(n_neighbors=i)
  knn.fit(x_train,y_train)
  pred_i=knn.predict(x_test)
  error_rate.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(14,8))
plt.plot(range(1,40),error_rate,color = 'blue',linestyle = 'dashed',
         marker= 's',markerfacecolor= 'red',markersize = 10)
plt.title('Error rate vs k values')
plt.xlabel('k')
plt.ylabel('Error rate')
```

```
Text(0, 0.5, 'Error rate')
```




```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)
pred = knn.predict(x_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[33  2]
 [ 2 26]]
              precision    recall  f1-score   support

        Mine       0.94      0.94      0.94        35
        Rock       0.93      0.93      0.93        28

    accuracy                           0.94        63
   macro avg       0.94      0.94      0.94        63
weighted avg       0.94      0.94      0.94        63
```

Result: The program is executed successfully and obtained the output.

PROGRAM NO-2

Aim: Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

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

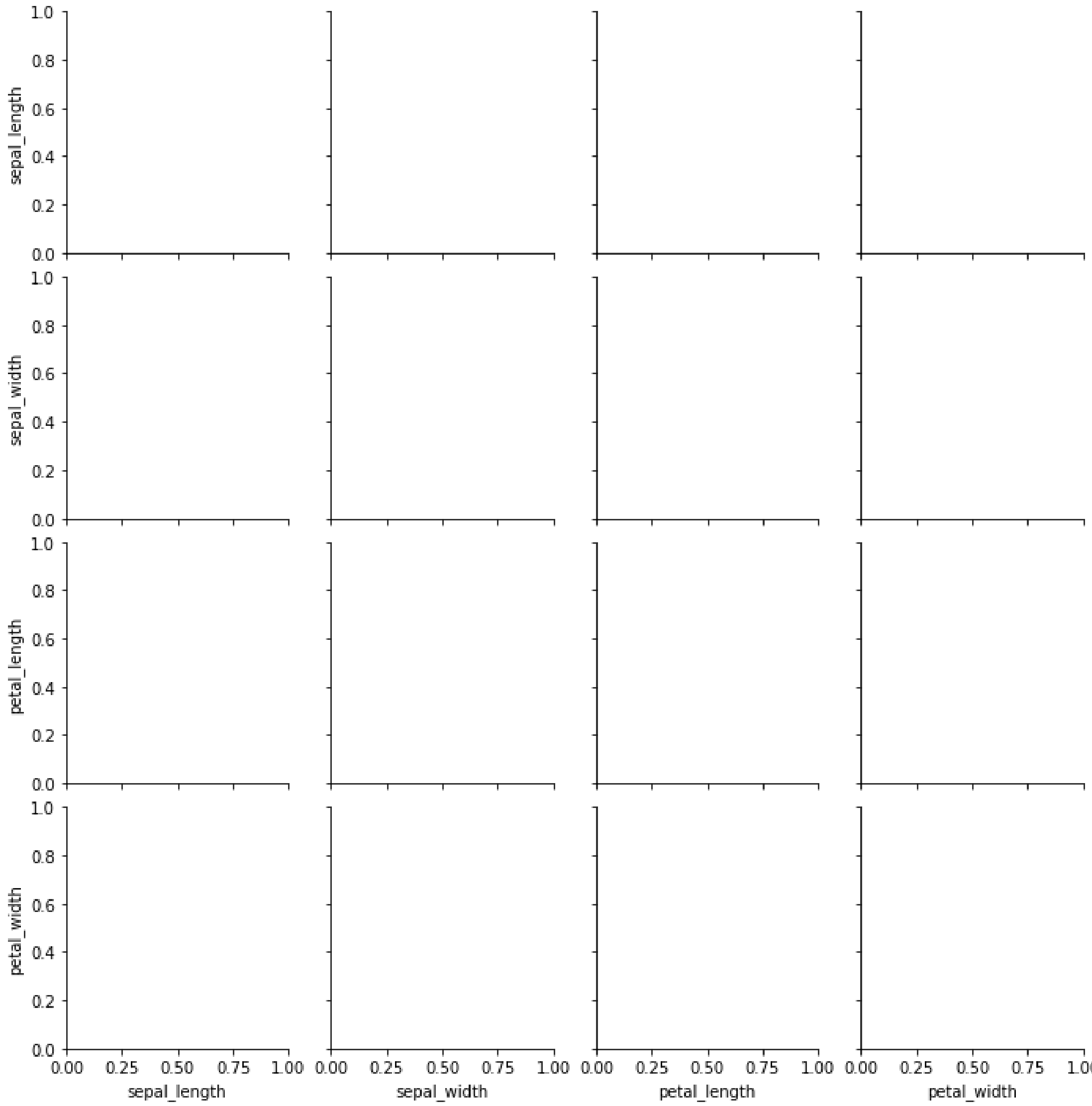
```
iris.head()
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Pairgrid

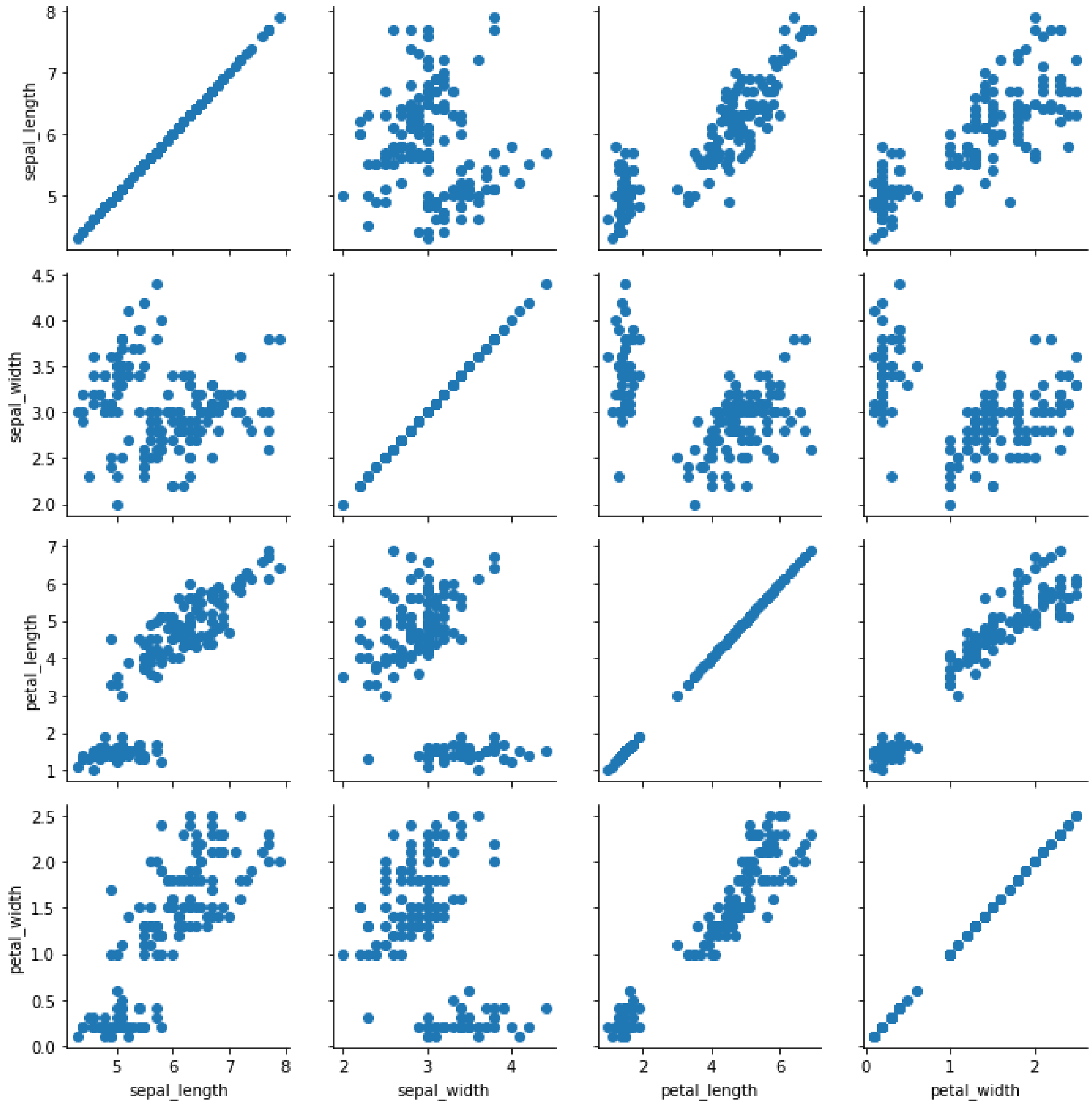
```
sns.PairGrid(iris)
```

```
<seaborn.axisgrid.PairGrid at 0x7f08b23f1810>
```



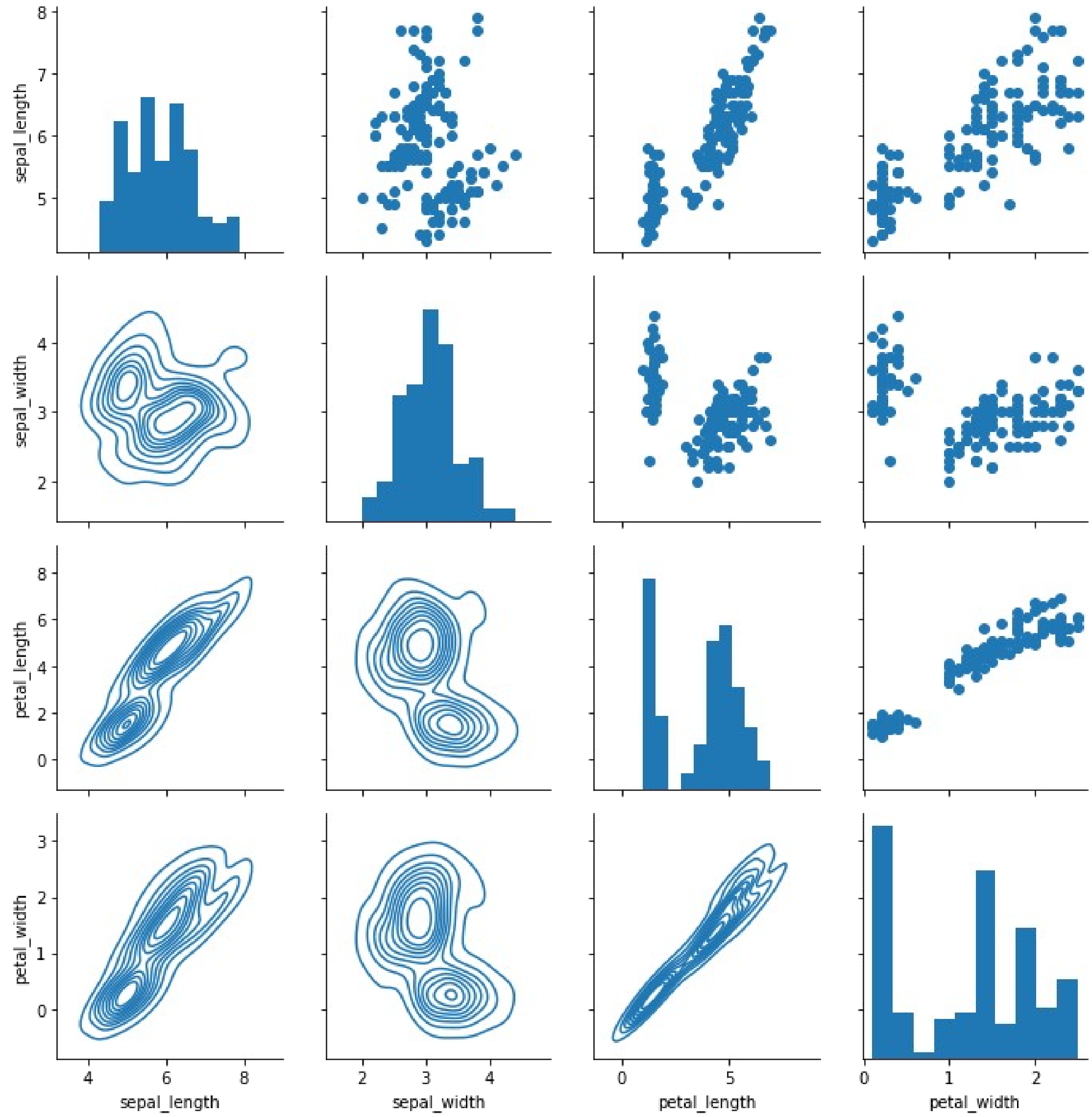

```
g = sns.PairGrid(iris)
g.map(plt.scatter)  # mapping to the grid
```

```
<seaborn.axisgrid.PairGrid at 0x7f08a91661d0>
```



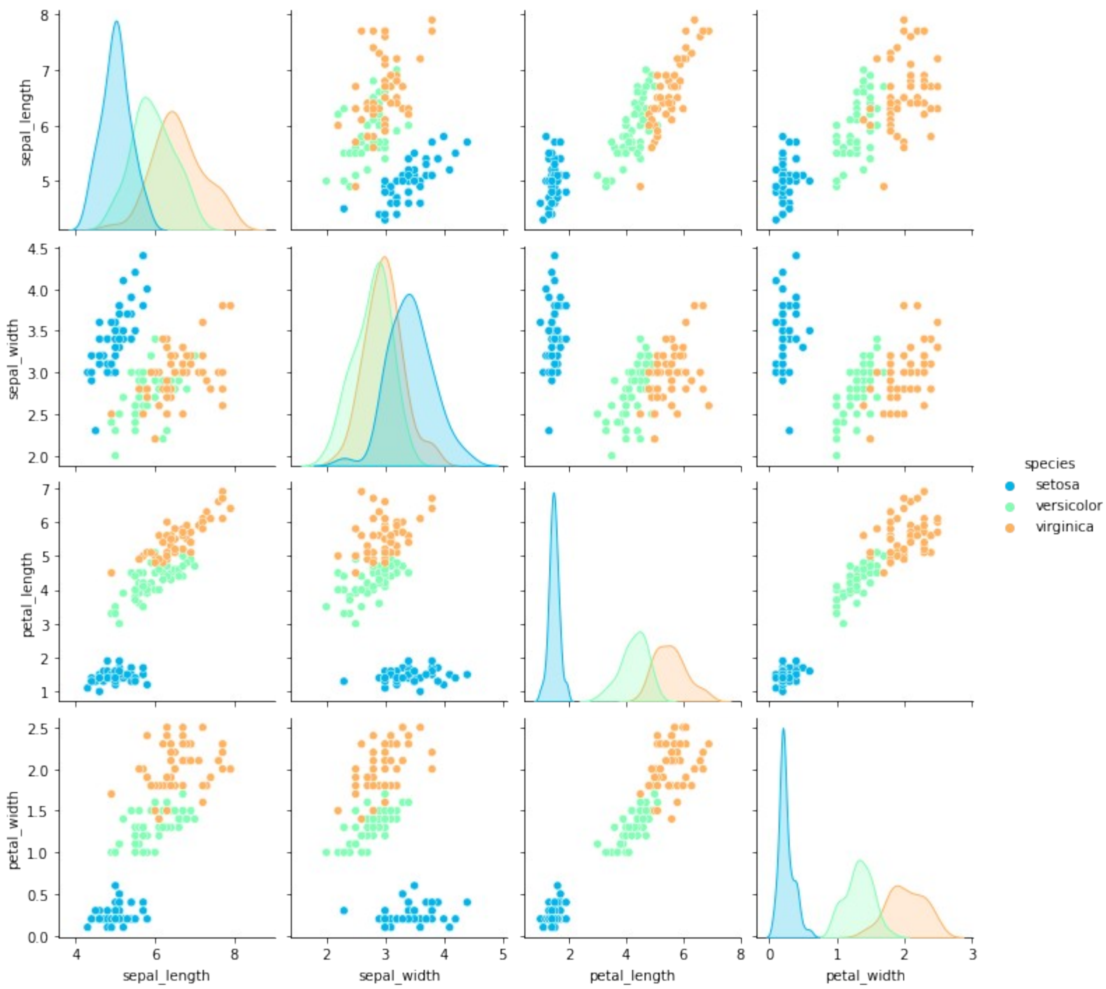

```
g = sns.PairGrid(iris)
g.map_diag(plt.hist)
g.map_upper(plt.scatter)
g.map_lower(sns.kdeplot)
```

```
<seaborn.axisgrid.PairGrid at 0x7f08b1ee85d0>
```




```
sns.pairplot(iris,hue='species',palette ='rainbow')
```

<seaborn.axisgrid.PairGrid at 0x7f08a3035090>

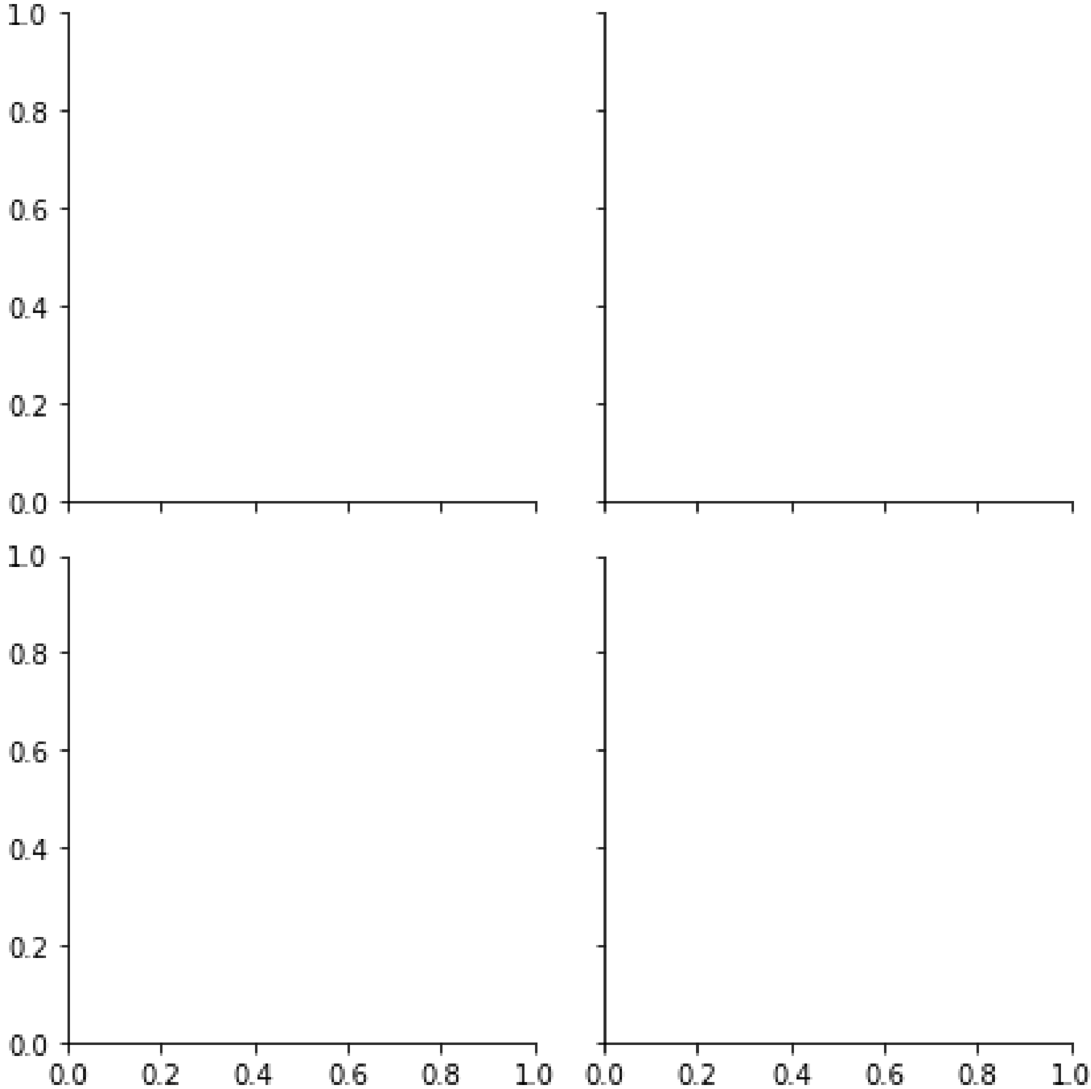


facetgrid

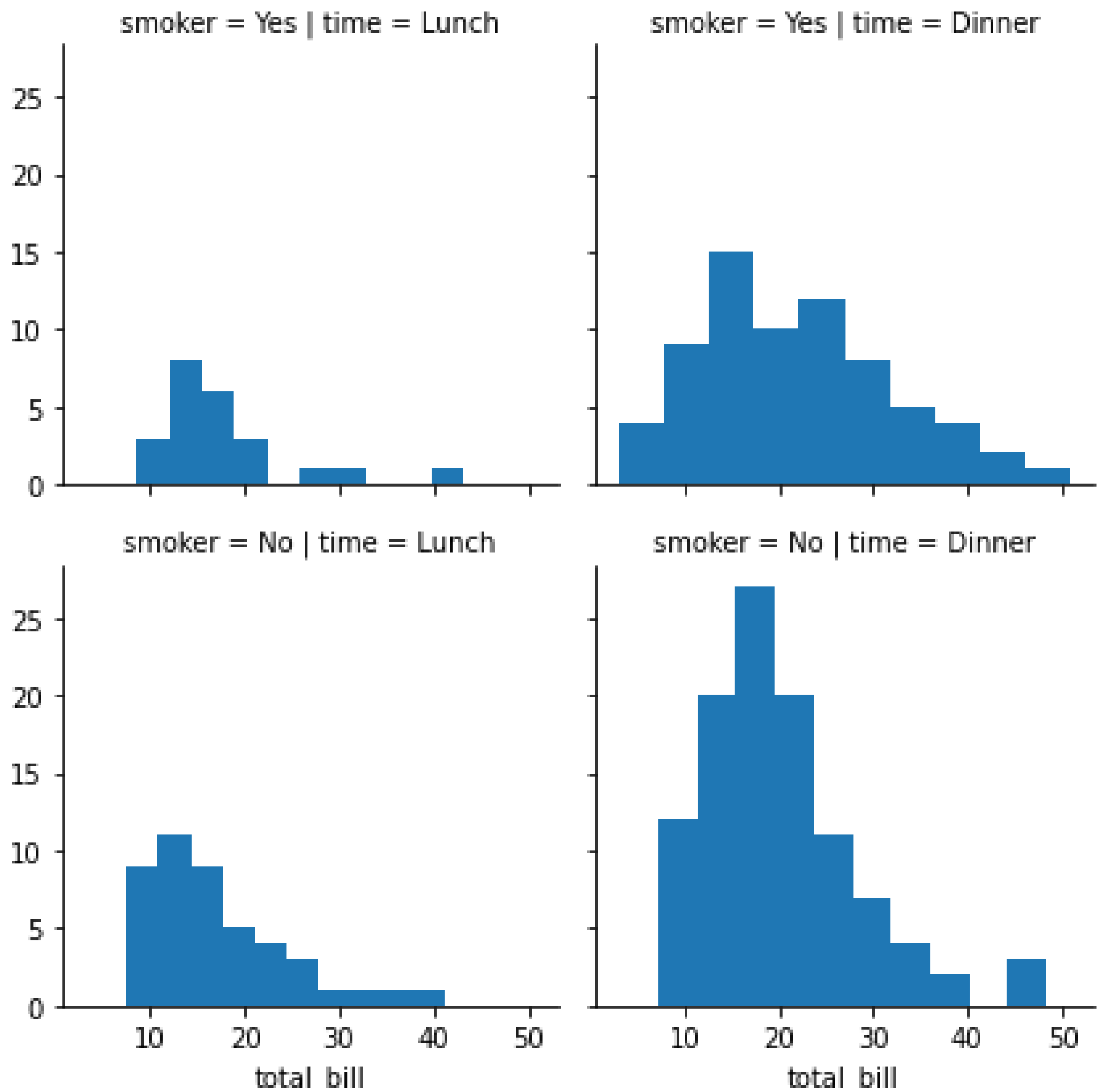
```
tips = sns.load_dataset('tips')
tips.head()
```

	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

```
g = sns.FacetGrid(tips, col= 'time',row='smoker')
```



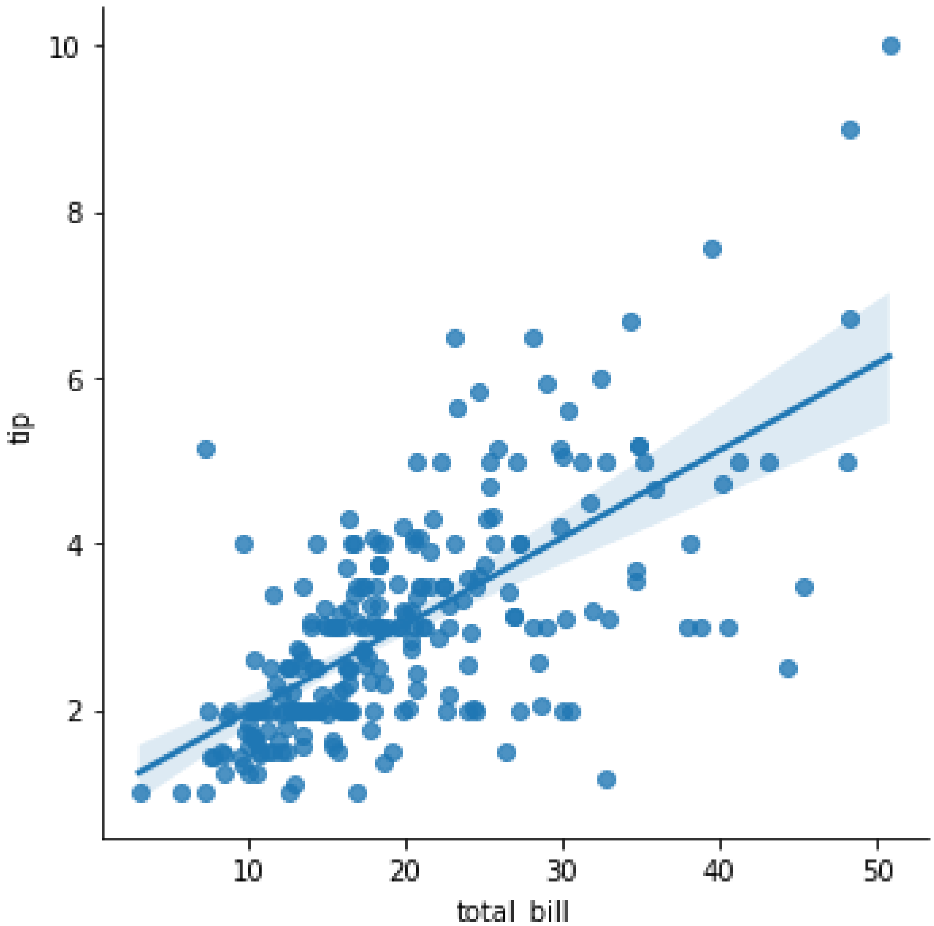

```
g = sns.FacetGrid(tips, col= 'time',row='smoker')
g =g.map(plt.hist,'total_bill')
```



Regression Plot

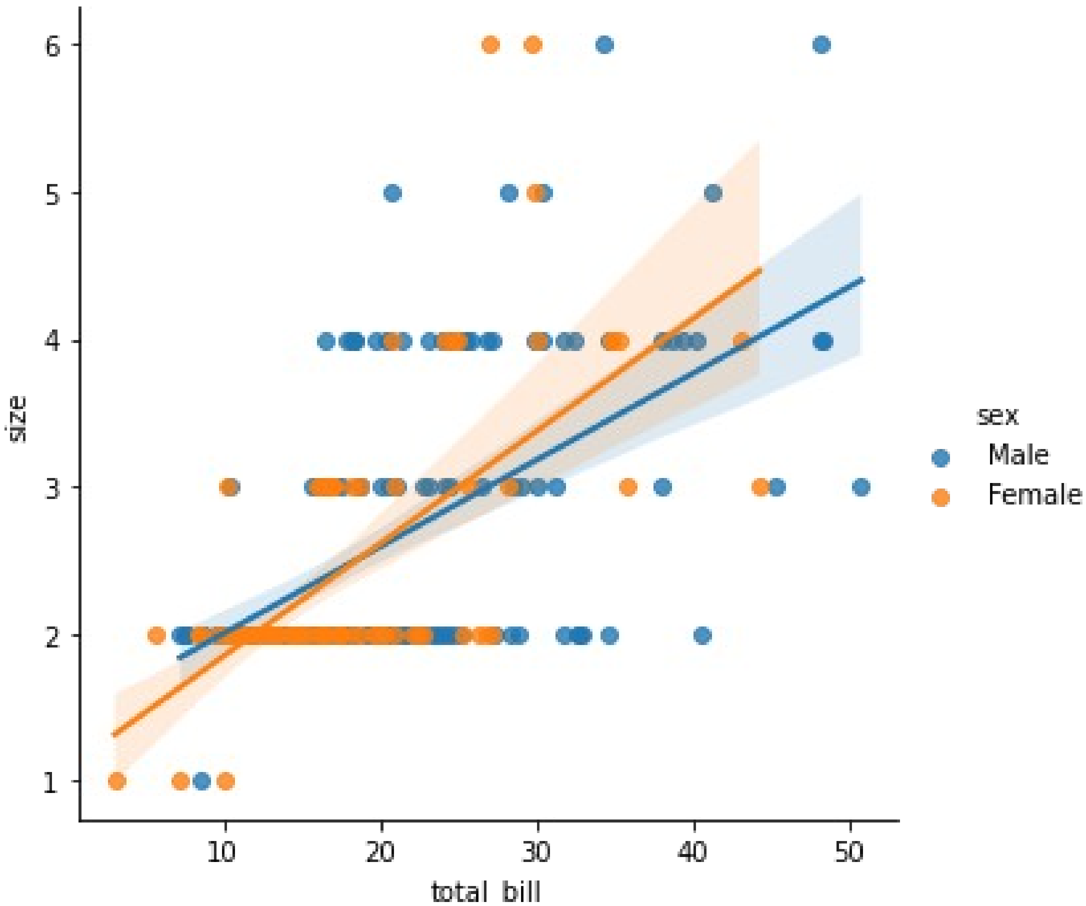
```
sns.lmplot(x ='total_bill',y='tip',data = tips)
```

<seaborn.axisgrid.FacetGrid at 0x7f08a249dd10>



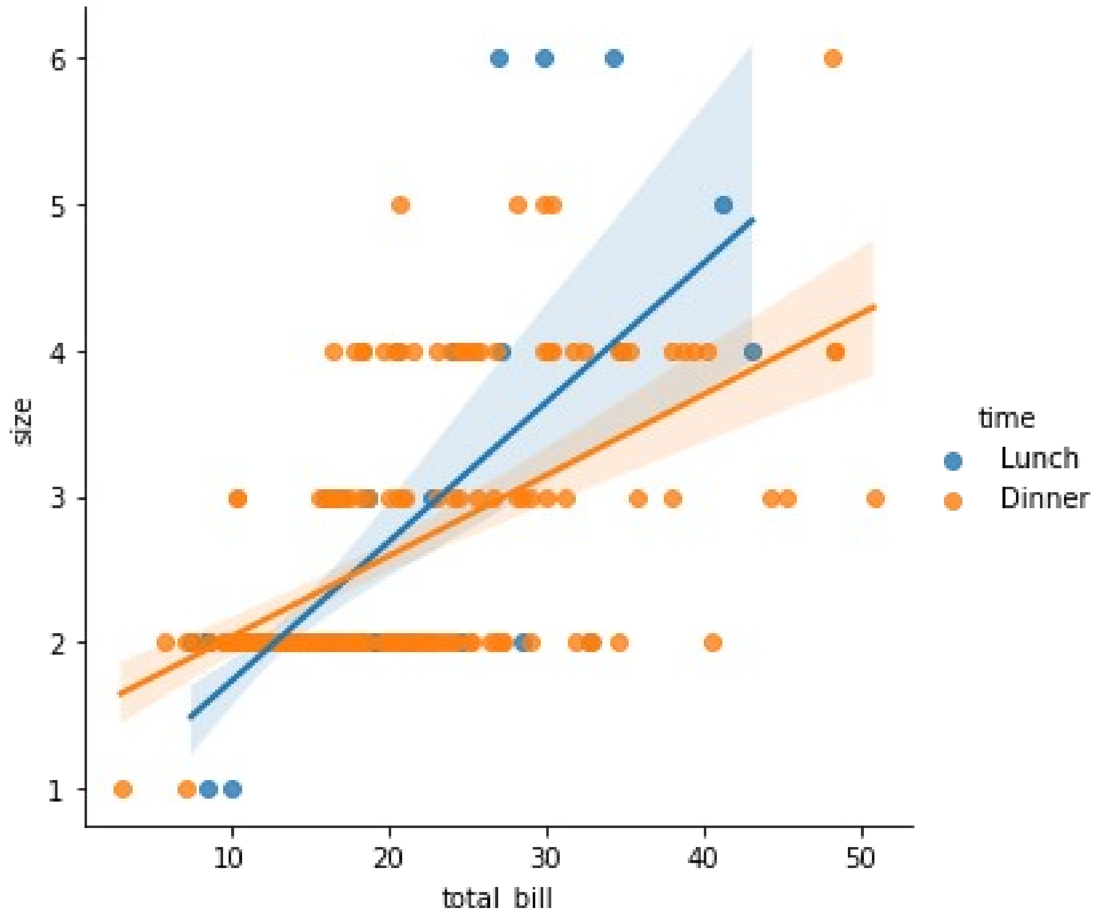

```
sns.lmplot(x ='total_bill',y='size',hue = 'sex',data = tips)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f08a2427a50>
```




```
sns.lmplot(x ='total_bill',y='size',hue = 'time',data = tips)
```

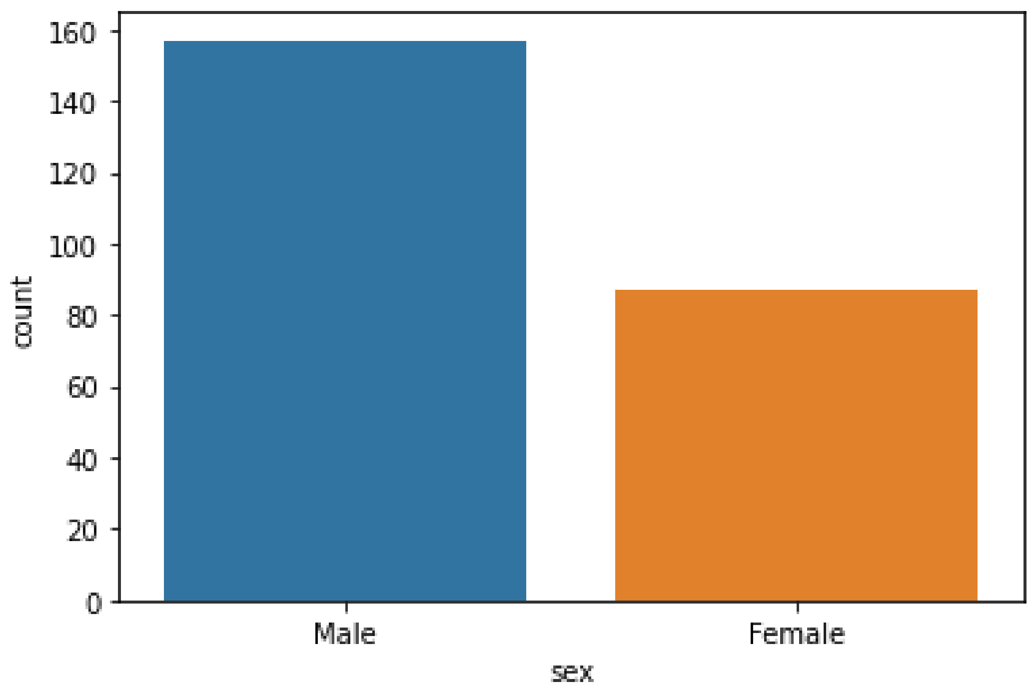
```
<seaborn.axisgrid.FacetGrid at 0x7f08a23b0ad0>
```



Style and color

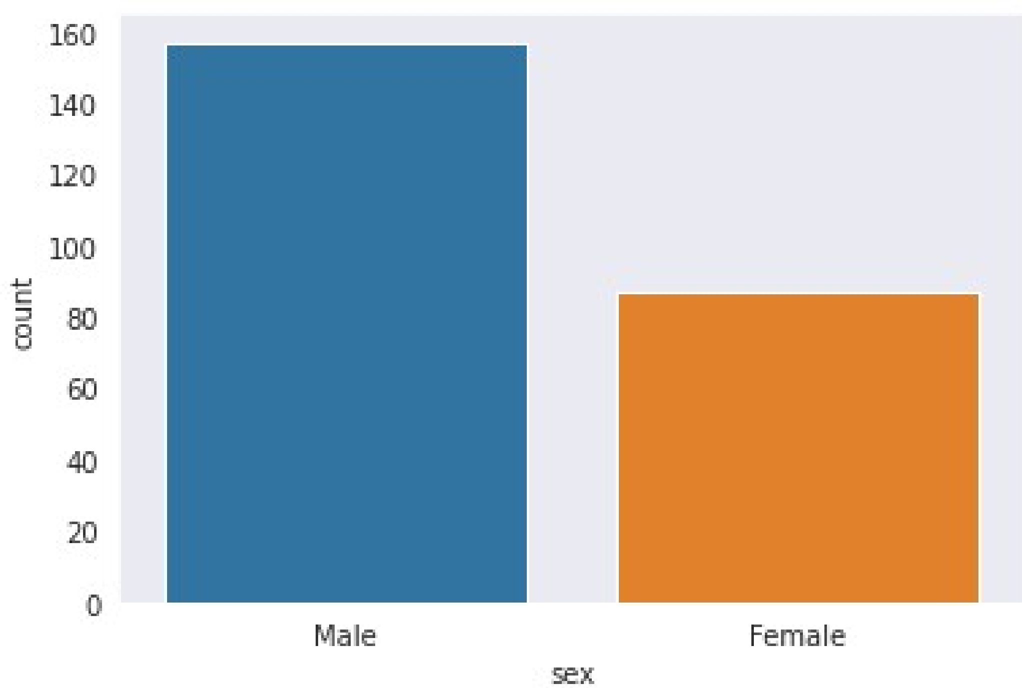
```
sns.countplot(x='sex',data = tips)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f08a234b990>



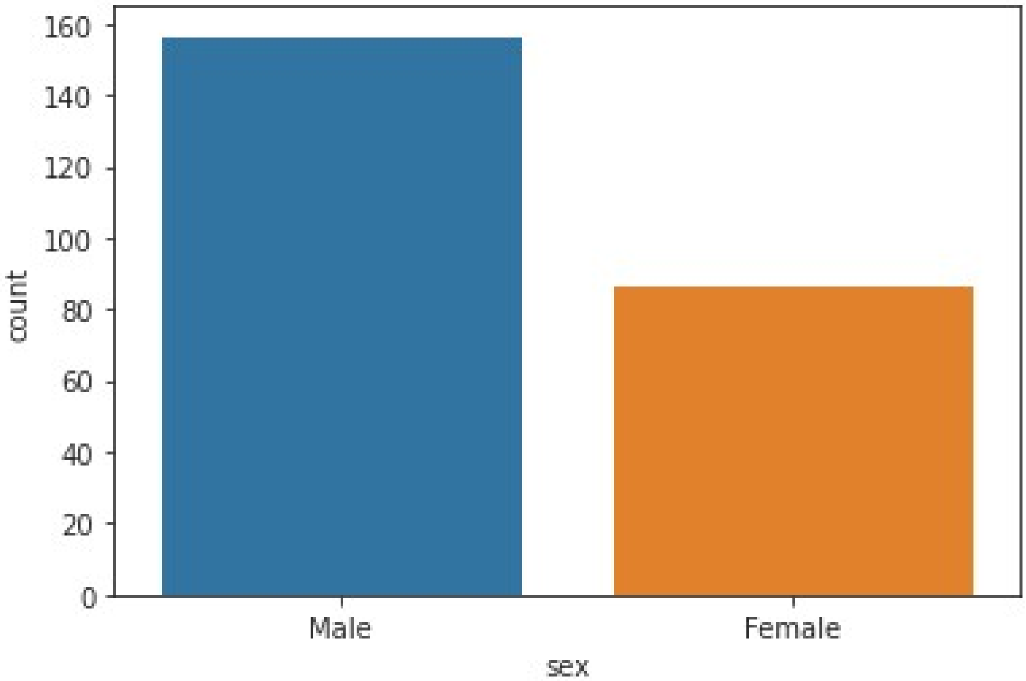

```
sns.set_style('dark')
sns.countplot(x='sex',data = tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08a2286bd0>
```

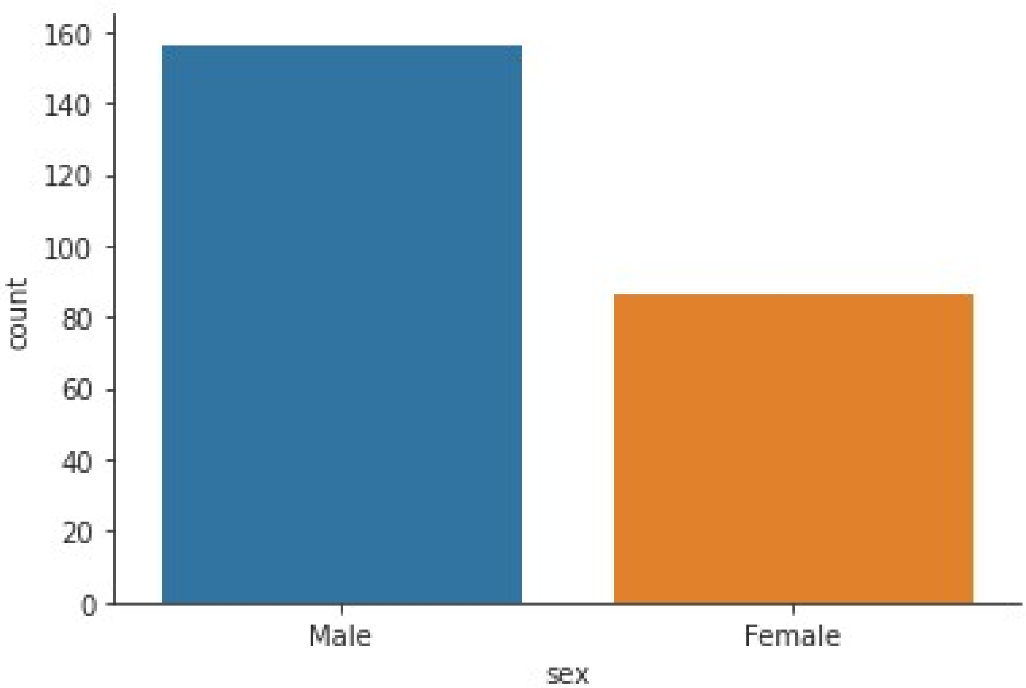



```
sns.set_style('ticks')
sns.countplot(x='sex',data = tips)
```

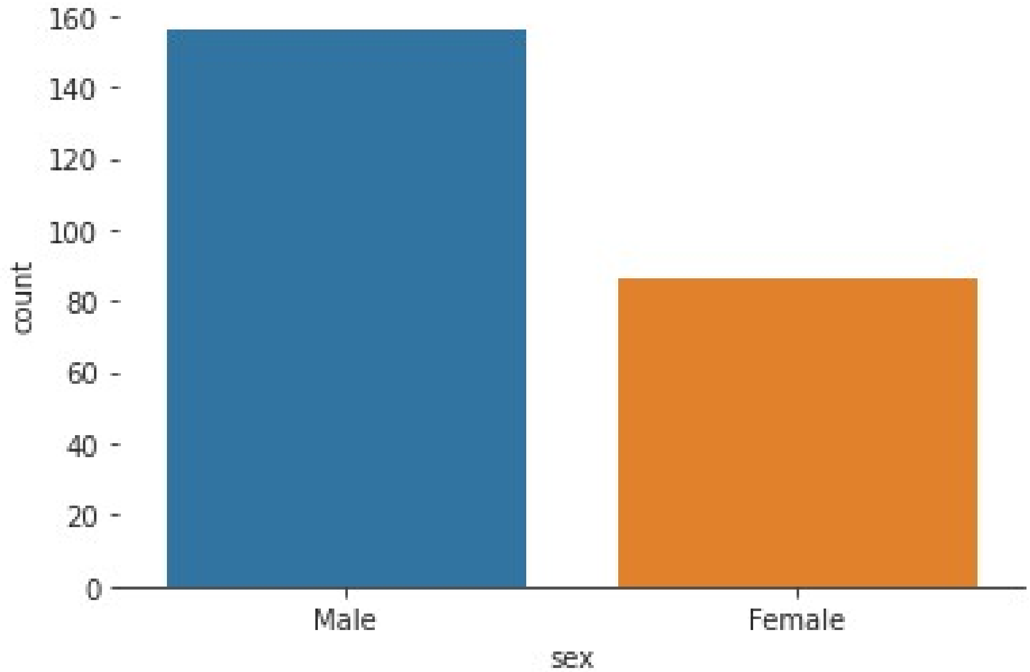
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08a226cad0>
```




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

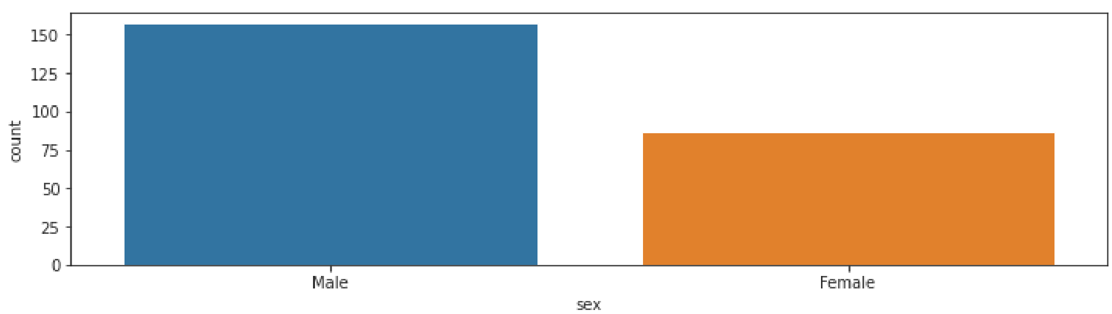



```
sns.countplot(x='sex',data = tips)
sns.despine(left = True)
```




```
plt.figure(figsize=(12,3))
resns.countplot(x='sex',data = tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f08a1fee450>
```



Result: The program is executed successfully and obtained the output.

PROGRAM NO-3

Aim: Program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance

```
#Simple Linear Regression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv('/content/sample_data/Salary_Data.csv')

df.head()

   YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0

x=df.iloc[:,:-1].values
y=df.iloc[:,:-1].values

# split the dataset into training set and testing set
from sklearn.model_selection import train_test_split
x_test,x_train,y_test,y_train=train_test_split(x,y,test_size=0.3,rando
m_state=30)

# fit the training dataset
from sklearn.linear_model import LinearRegression
regsr=LinearRegression()
regsr.fit(x_train,y_train)

LinearRegression()

y_pred=regsr.predict(x_test)

#visualize
plt.scatter(x_train,y_train,color='red')
plt.plot(x_train,regsr.predict(y_train),color='green')
plt.title("salary VS experience")
plt.xlabel("experience")
plt.ylabel("salary")
plt.show()
```


```
# visualizing the test results
plt.scatter(x_test,y_test,color='red')
plt.plot(x_test,regsr.predict(y_test),color='green')
plt.title("salary VS experience")
plt.xlabel("experience")
plt.ylabel("salary")
plt.show()
```


```
# Multiple Linear Regression

ds=pd.read_csv('/content/sample_data/combined cycle powerplant.csv')

ds.head()

      AT      V       AP     RH      PE
0   8.34  40.77  1010.84  90.01  480.48
1  23.64  58.49  1011.40  74.20  445.75
2  29.74  56.90  1007.15  41.91  438.76
3  19.07  49.69  1007.22  76.79  453.09
4  11.80  40.66  1017.13  97.20  464.43

u=ds.drop('PE',axis=1).values
v=ds['PE'].values

from sklearn.model_selection import train_test_split
u_train,u_test,v_train,v_test =
train_test_split(u,v,test_size=0.3,random_state=45)

#fitting
from sklearn.linear_model import LinearRegression
rgr=LinearRegression()
rgr.fit(u_train,v_train)

LinearRegression()

v_pred=rgr.predict(u_test)
print(v_pred)
```

```
[451.12275809 472.67973273 434.23317529 ... 479.20120415 470.80190333
 437.26990979]
```

```
#take the values of first row in u and compare our predicted v values
with actual v value
rgr.predict([[23.64   ,58.49,    1011.40,   74.20]])
```

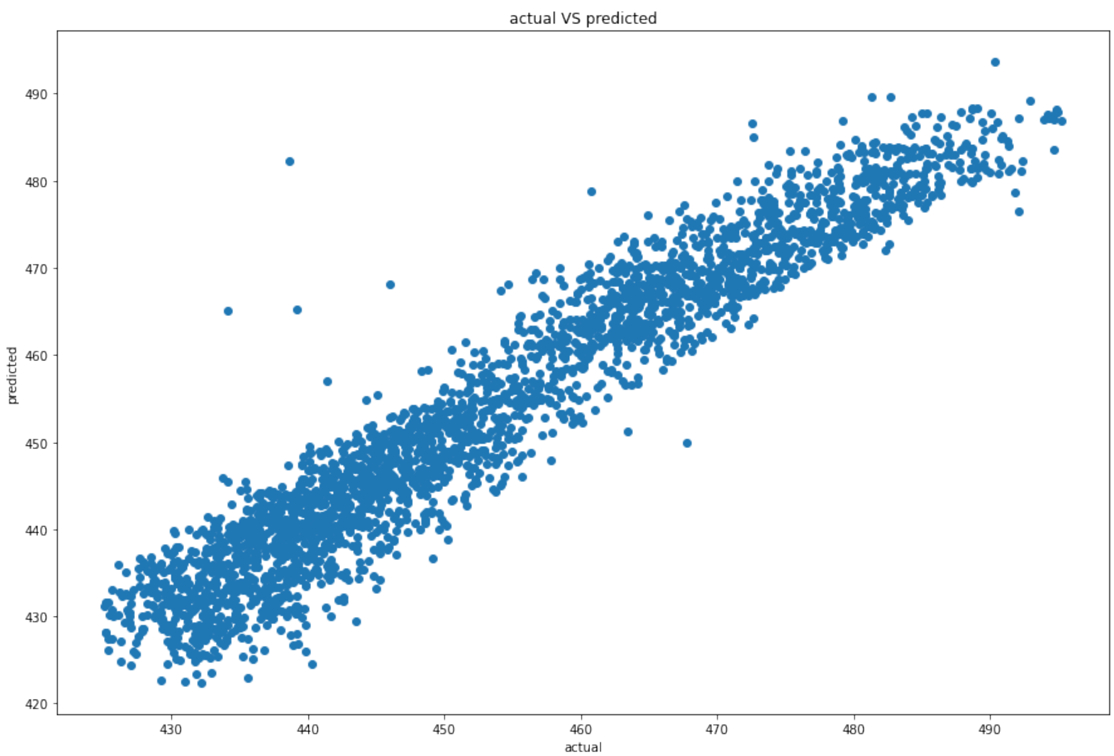
```
array([445.23162503])
```

```
from sklearn.metrics import r2_score
r2_score(v_test,v_pred)
```

```
0.9270483924843018
```

```
# visualize
plt.figure(figsize=(15,10))
plt.scatter(v_test,v_pred)
plt.title("actual VS predicted")
plt.xlabel("actual")
plt.ylabel("predicted")
```

```
Text(0, 0.5, 'predicted')
```




```
#print prdedicted values of our model
pred_ds=pd.DataFrame({'Actual value':v_test ,'Predicted
value':v_pred})
pred_ds
```

```
      Actual value  Predicted value
0           449.23       451.122758
1           474.70       472.679733
2           434.18       434.233175
3           436.70       442.479946
4           477.27       481.164400
...            ...              ...
2866        465.26       462.625918
2867        441.71       442.161640
2868        477.51       479.201204
2869        467.62       470.801903
2870        438.52       437.269910

[2871 rows x 2 columns]
```

Result: The program is executed successfully and obtained the output.