StatML LAB-1

Import numpy and pandas for operations

```
import numpy as np
import pandas as pd
```

'np.array()' function produces an array as shown

```
lst = [1,2,3] array1 =
np.array(lst) array1
```

```
array([1, 2, 3])
```

And as for the given functions 'np' produces a set of zeros, ones and range of numbers between certain numbers.

0	abc	12	70
1	xyz	13	80

2 hij 14 90

```
print("A series of zeroes:",np.zeros(7)) print("A series of ones:",np.ones(9)) print("A series of numbers:",np.arange(5,16)) print("Numbers spaced apart by 2:",np.arange(0,11,2)) print("Numbers spaced apart by float:",np.arange(0,11,2.5)) print("Every 5th number from 30 in reverse order: ",np.arange(30,-1,print("11 linearly spaced numbers between 1 and 5: ",np.linspace(1,5,
```

```
A series of zeroes: [0. 0. 0. 0. 0. 0. 0.]
A series of ones: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
A series of numbers: [ 5  6  7  8  9 10 11 12 13 14 15]
Numbers spaced apart by 2: [ 0  2  4  6  8 10]
Numbers spaced apart by float: [ 0.   2.5  5.   7.5 10. ]
Every 5th number from 30 in reverse order:  [30 25 20 15 10  5  0]
11 linearly spaced numbers between 1 and 5:  [1.  1.4 1.8 2.2 2.6 3.  3.4 3.8 4.2 4.6 5. ]
```

'pd.read_csv' function reads a CSV le and syntax as shown.

```
dataframe = pd.read_csv("wine.csv")
dataframe.head()
```

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
datacsv = pd.read_csv("createcsv.csv")
datacsv
```

name Location Marks

'pd.read_txt' function reads a text le and syntax as shown.

```
datatext = pd.read_table("pandatext.txt")
datatext
```

	Name Height Weight Hometown
0	0 Ashley 155 140 Palo Alto
1	1 Robin 145 122 Fremont
2	2 Priyanka 152 131 Santa Clara
3	3 Youngchul 167 148 Cupertino
4	4 Aziz 161 139 San Francisco
5	5 Zoey 181 190 Hayward

'pd.read_xlsx' function reads a excel le and syntax as shown.

```
dataxl = pd.read_excel("Height_weight.xlsx")
dataxl
```

	Name	Height	Weight
0	Ashton	155	135
1	Kate	125	140
2	Bruce	178	210
3	Tom	181	165
4	Bill	165	180

```
read_from_html = pd.read_html("")
```

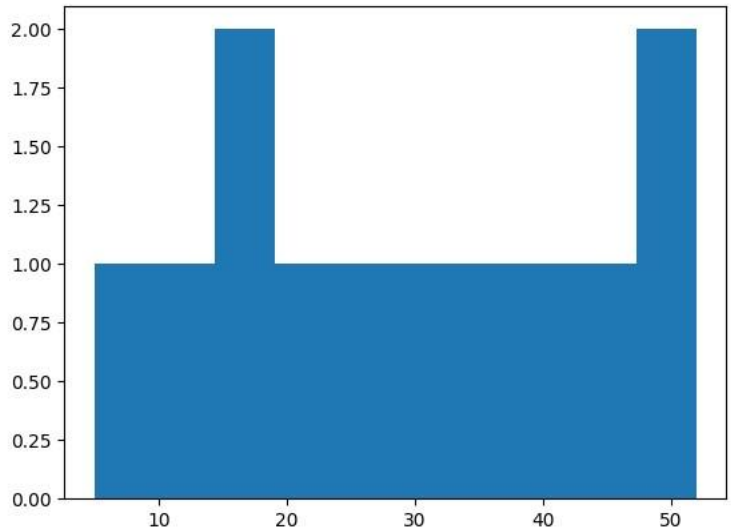
Now let's import 'matplotlib.pyplot' for graph operations. And given are the examples of this library like scatter, hist etc.

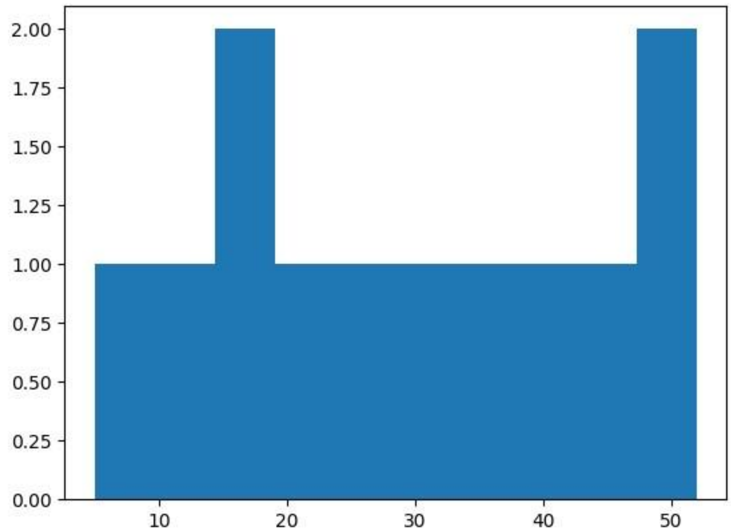
```
import matplotlib.pyplot as plt

people = ['Ann','Brandon','Chen','David','Emily','Farook',
'Gagan','Hamish','Imran','Julio','Katherine','Lily'] age =
[21,12,32,45,37,18,28,52,5,40,48,15] weight =
[55,35,77,68,70,60,72,69,18,65,82,48] height =
[160,135,170,165,173,168,175,159,105,171,155,158]
```

```
plt.scatter(age,weight)
plt.show()
```

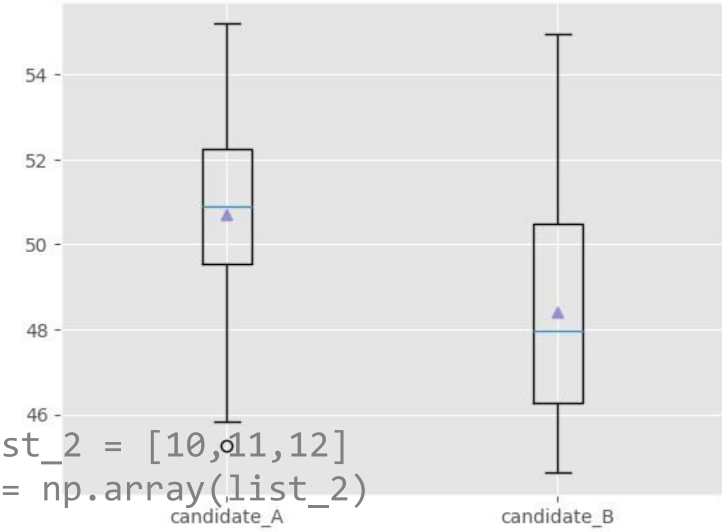
Double-click (or enter) to edit

```
plt.hist(age)
plt.show()
```





```
days = np.arange(1,31) candidate_A =
50+days*0.07+2*np.random.randn(30) candidate_B = 50-
days*0.1+3*np.random.randn(30) plt.style.use("ggplot")
plt.boxplot(x=[candidate_A,candidate_B],showmeans = True)
plt.grid(True) plt.xticks([1,2],["candidate_A","candidate_B"])
plt.show()
```




```
list_2 = [10,11,12]
n = np.array(list_2)
n
```

```
array([10, 11, 12])
```

```
print(f"Sum of arrays {lst} and {list_2} is {array1 + n}")
```

```
Sum of arrays [1, 2, 3] and [10, 11, 12] is [11 13 15]
```

```
#Operations on numpy arrays print(f"Multiplication of NUmpy arrays
{array1 * n}") print(f"Subtraction of NUmpy arrays {array1 - n}")
print(f"Division of NUmpy arrays {array1 / n}")
```

```
Multiplication of NUmpy arrays [10 22 36]
Subtraction of NUmpy arrays [-9 -9 -9]
Division of NUmpy arrays [0.1        0.18181818 0.25      ]
```

StatML LAB-2

Import required libraries

```
import numpy as np import matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D from scipy.stats import
multivariate_normal
```

First, you de ne a 4x4 covariance matrix named covariance. This matrix represents the covariances between four variables. The covariance matrix contains the variances of each variable on the diagonal and the covariances between variables off the diagonal.

You then use np.linalg.inv(covariance) to calculate the inverse of the covariance matrix. The np.linalg.inv function is a NumPy function that computes the matrix inverse.

Finally, you print the result, which is the precision matrix.

```
covariance = np.array([[0.14, -0.3, 0.0, 0.2],
[-0.3, 1.16, 0.2, -0.8],
[0.0, 0.2, 1.0, 1.0],                          [0.2, -
0.8, 1.0, 2.0]]) precision =
np.linalg.inv(covariance) print(precision)
```

```
[[ 60.   50.  -48.   38. ]
 [ 50.   50.  -50.   40. ]
 [-48.  -50.   52.4 -41.4]
 [ 38.   40.  -41.4  33.4]]
```

generate_pair() is a function that generates a pair of random values following a bivariate (two-dimensional) normal distribution. The parameters passed to np.random.multivariate_normal specify the mean vector and the covariance matrix for the distribution:

The mean vector [0.8, 0.8] indicates that the distribution is centered around the point (0.8, 0.8). This means that on average, the generated pairs will tend to be close to this point.

The covariance matrix [[0.1, -0.1], [-0.1, 0.12]] speci es how the two variables (in this case, x and y) are related. The values on the diagonal (0.1 and 0.12) represent the variances of the two variables, while the off-diagonal values (-0.1) represent the covariance between the variables. This matrix determines how spread out or correlated the generated pairs will be. Positive covariance values indicate that the variables tend to increase together, while negative values indicate that they tend to move in opposite directions.

When you call generate_pair(), it returns a pair of random values drawn from the speci ed bivariate normal distribution, and the values are stored in the variable mu_t.

Finally, you print the mu_t variable, which displays the pair of random values generated by the function.

```
def generate_pair():     return np.random.multivariate_normal([0.8,
0.8], [[0.1, -0.1],[-0

mu_t = generate_pair()
print(mu_t)
```

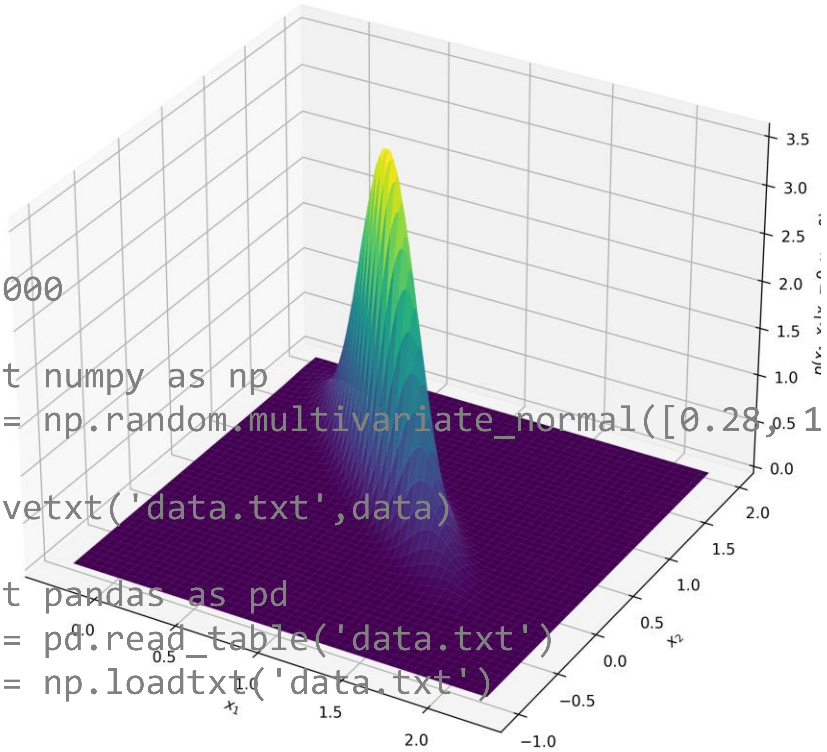
```
[0.93474884 0.80350139]
```

```
x, y = np.mgrid[-0.25:2.25:.01,-1:2:.01]
pos = np.empty(x.shape + (2,)) pos[:, :,
0] = x pos[:, :, 1] = y
```



```
mu p    [0 8
0 8]
```

```
mu_p = [0.8, 0.8] cov_p = [[0.1, -0.1], [-0.1, 0.12]] z =
multivariate_normal(mu_p, cov_p).pdf(pos)

fig = plt.figure(figsize=(10, 10), dpi=300)
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x, y, z, cmap=plt.cm.viridis)
plt.xlabel('$x_1$') plt.ylabel('$x_2$')
ax.set_zlabel('$p(x_1, x_2 | x_3=0, x_4=0)$')
plt.savefig('cond_mvg.png', bbox_inches='tight', dpi=300) plt.show()
```




```
N = 1000

import numpy as np
data = np.random.multivariate_normal([0.28, 1.18], [[2.0, 0.8], [0.8,

np.savetxt('data.txt',data)

import pandas as pd
data = pd.read_table('data.txt')
data = np.loadtxt('data.txt')
```

```
data
```

```
data
```

```
array([[-1.97416035,  0.18085742],
[ 1.89414551, -0.14292725],
       [ 2.44416774, -1.48606541],
       ...,
       [ 2.2292759 ,  0.26630475],
       [-1.50839992, -0.01755483],
       [-1.63801017,  2.59365018]])
```

```
mu_ml = data.mean(axis=0) x = data - mu_ml
cov_ml = np.dot(x.T, x) / N
cov_ml_unbiased = np.dot(x.T, x) / (N - 1)
print(mu_ml) print(cov_ml)
print(cov_ml_unbiased)
```

```
[0.32950325 1.24389006]
[[2.02971382 0.8557863 ]
 [0.8557863  4.23663802]]
[[2.03174557 0.85664294]
 [0.85664294 4.24087889]]
```

```
def seq_ml(data):     mus = [np.array([[0],
[0]])]     for i in range(N):         x_n =
data[i].reshape(2, 1)         mu_n = mus[-1] +
(x_n-mus[-1]) / (i + 1)         mus.append(mu_n)
return mus


mus_ml = seq_ml(data) print(mus_ml[-
1])
```

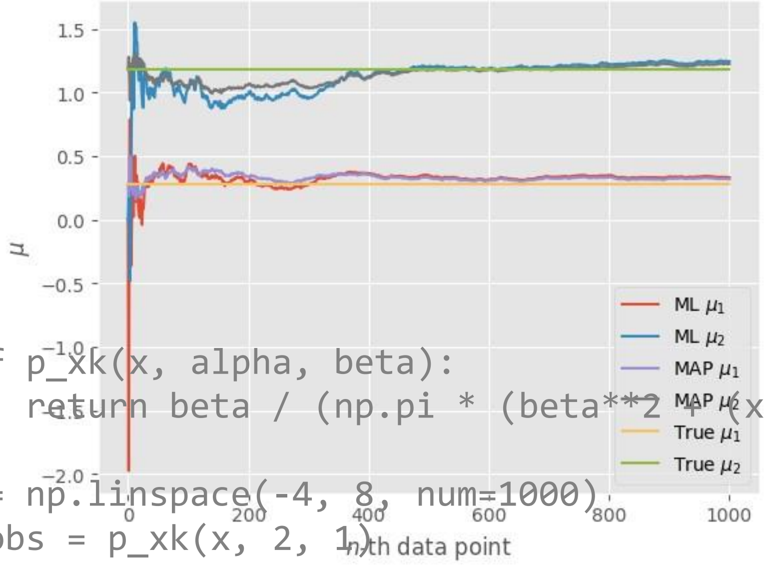
```
[[0.32950325]
 [1.24389006]]
```

```
mu_p = np.array([[0.28], [1.18]]) cov_p = np.array([[0.1, -0.1], [-
0.1, 0.12]]) cov_t = np.array([[2.0, 0.8], [0.8, 4.0]])

def seq_map(data, mu_p, cov_p, cov_t):
mus, covs = [mu_p], [cov_p]     for x
in data:         x_n = x.reshape(2, 1)
        cov_n = np.linalg.inv(np.linalg.inv(covs[-1]) + np.linalg.inv
mu_n = cov_n.dot(np.linalg.inv(cov_t).dot(x_n) + np.linalg.in
mus.append(mu_n)         covs.append(cov_n)     return mus, covs

mus_map, covs_map = seq_map(data, mu_p, cov_p, cov_t) print(mus_map[-
1])
```

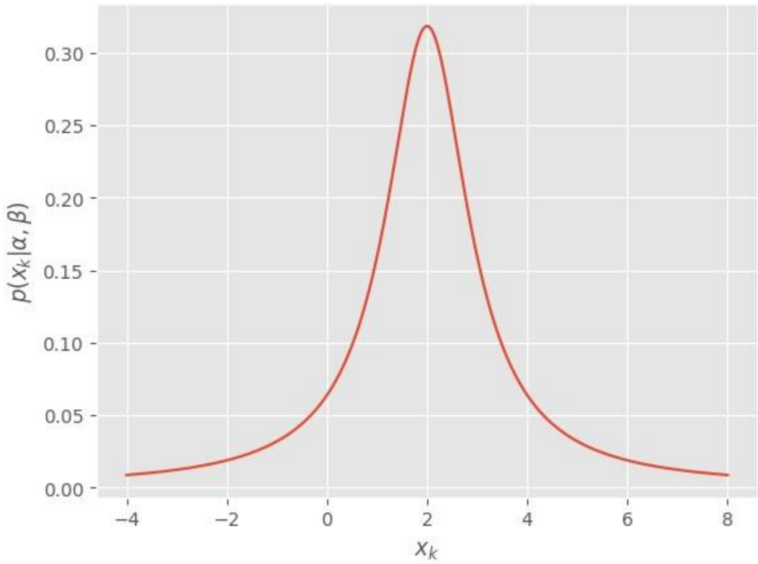
```
[[0.31735853]
 [1.22405254]]
```

```
X = np.arange(N+1)
mus1_ml = [mu[0] for mu in mus_ml]
mus2_ml = [mu[1] for mu in mus_ml]
mus1_map = [mu[0] for mu in mus_map]
mus2_map = [mu[1] for mu in mus_map]
mus1_t = [0.28] * (N+1) mus2_t =
[1.18] * (N+1)
plt.style.use('ggplot')
plt.plot(X, mus1_ml, label='ML $\mu_1$')
plt.plot(X, mus2_ml, label='ML $\mu_2$')
plt.plot(X, mus1_map, label='MAP $\mu_1$')
plt.plot(X, mus2_map, label='MAP $\mu_2$')
plt.plot(X, mus1_t, label='True $\mu_1$')
plt.plot(X, mus2_t, label='True $\mu_2$')
plt.xlabel('$n$-th data point')
plt.ylabel('$\mu$') plt.legend(loc=4)
```

```
plt.savefig('seq_learning.png', bbox_inches='tight', dpi=300)
plt.show()
```

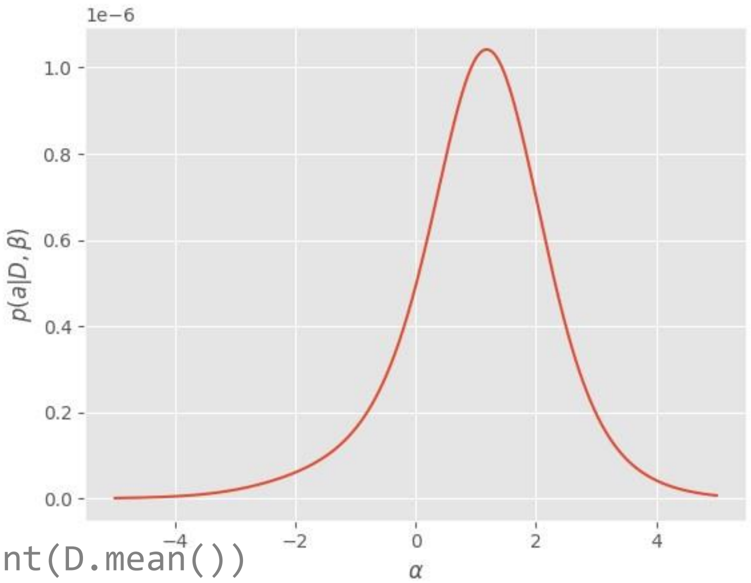



```
def p_xk(x, alpha, beta):
    return beta / (np.pi * (beta**2 + (x-alpha)**2))

x = np.linspace(-4, 8, num=1000)
probs = p_xk(x, 2, 1)

plt.plot(x, probs)
plt.xlabel('$x_k$')
plt.ylabel(r'$p(x_k | \alpha,
\beta)$') plt.savefig('prob_xk.png',
bbox_inches='tight', dpi=300)
plt.show()
```




```
def p_a(x, alpha, beta):     return np.product(beta / (np.pi *
beta**2 + (x-alpha)**2))
```

```
D = np.array([4.8, -2.7, 2.2, 1.1, 0.8, -7.3])
alphas = np.linspace(-5, 5, num=1000) beta = 1
likelihoods = [p_a(D, alpha, beta) for alpha
in alphas] plt.plot(alphas, likelihoods)
plt.xlabel(r'$\alpha$') plt.ylabel(r'$p(a | D,
\beta)$') plt.savefig('prob_a.png',
bbox_inches='tight', dpi=300) plt.show()
```




```
print(D.mean())

print(alphas[np.argmax(likelihoods)])
```

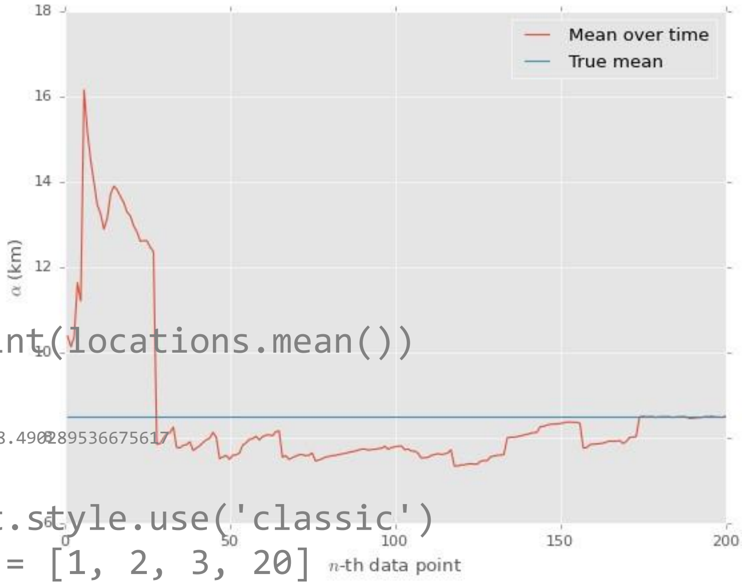
```
-0.18333333333333326
1.1761761761761758
```

```
alpha_t = np.random.uniform(0, 10)
beta_t = np.random.uniform(1, 2)
print(alpha_t, beta_t)
```

```
9.556577442054802 1.3712422301299707
```

```
def location(angle, alpha, beta):
return beta * np.tan(angle) + alpha

N = 200 angles = np.random.uniform(-np.pi/2, np.pi/2, N) locations =
np.array([location(angle, alpha_t, beta_t) for angle in a

mus = [locations[:i + 1].mean() for i in range(N)] mean =
[locations.mean()] * (N) X = np.arange(1, N + 1)
plt.style.use('ggplot')
plt.plot(X, mus, label='Mean over time')
plt.plot(X, mean, label='True mean')
plt.xlabel('$n$-th data point')
plt.ylabel(r'$\alpha$ (km)')
plt.legend()
```

```
plt.savefig('mean_x.png', bbox_inches='tight', dpi=300)
plt.show()
```



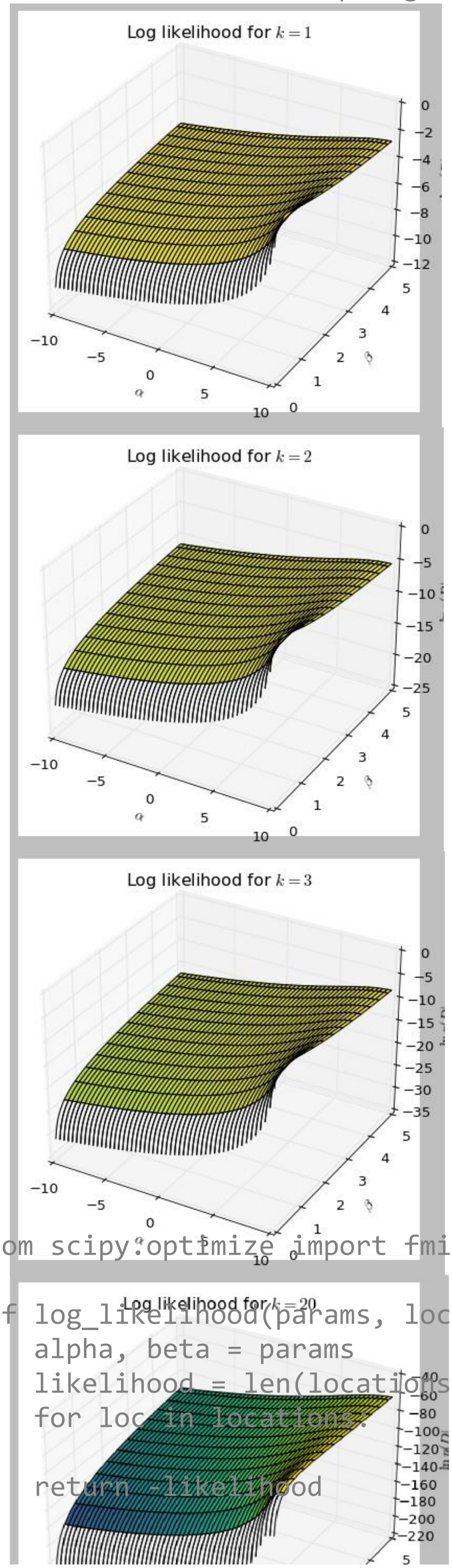

```
print(locations.mean())
```

```
8.490289536675617
```

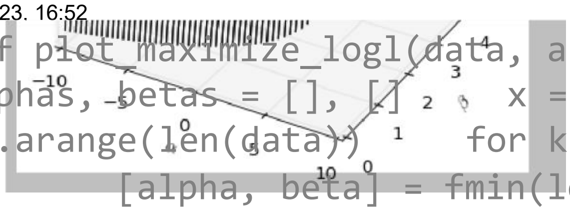
```
plt.style.use('classic')
ks = [1, 2, 3, 20]
alphas, betas = np.mgrid[-10:10:0.04, 0:5:0.04] # alphas, betas =
np.meshgrid(np.linspace(-10, 10, num=500), np.linsp for k in ks:
x = locations[:k]     # We only have to calculate the constant once
likelihood = k * np.log(betas/np.pi)     for loc in x:
likelihood -= np.log(betas**2 + (loc - alphas)**2)

    fig = plt.figure()     ax = fig.add_subplot(projection='3d')
ax.plot_surface(alphas, betas, likelihood, cmap=plt.cm.viridis, v
plt.xlabel(r'$\alpha$')     plt.ylabel(r'$\beta$')
ax.set_zlabel('$\ln p(D | \alpha, \beta)$')     plt.title('Log
likelihood for $k = {}$'.format(k))
plt.savefig('logl_{}.png'.format(k), bbox_inches='tight', dpi=300
plt.show()
```

```
<ipython-input-49-f58e94a921d6>:8: RuntimeWarning: divide by zero encountered in log
likelihood = k * np.log(betas/np.pi)
/usr/local/lib/python3.10/dist-packages/mpl_toolkits/mplot3d/proj3d.py:180: RuntimeWarning: invalid value encountered in true_divide
txs, tys, tzs = vecw[0]/w, vecw[1]/w, vecw[2]/w
```

```
        likelihood -= np.log
```




```
from scipy.optimize import fmin

def log_likelihood(params, locations):
    alpha, beta = params
    likelihood = len(locations) * np.log(beta/np.pi)
    for loc in locations:

    return -likelihood
                                                    (beta**2 + (loc -
alpha)**2)
```



```
def plot_maximize_logl(data, alpha_t, beta_t):
alphas, betas = [], []    x =
np.arange(len(data))     for k in x:
        [alpha, beta] = fmin(log_likelihood, (0, 1), args=(data[:k],)
alphas.append(alpha)         betas.append(beta)

    plt.style.use('ggplot')     plt.plot(x, alphas,
label=r'$\alpha$')     plt.plot(x ,betas, label=r'$\beta$')
plt.plot(x, [alpha_t]*len(data), label=r'$\alpha_t$')     plt.plot(x,
[beta_t]*len(data), label=r'$\beta_t$')     plt.xlabel('$k$')
plt.ylabel('location (km)')     plt.legend()
plt.savefig('plots/min_logl.png', bbox_inches='tight', dpi=300)
plt.show()     print(alphas[-1], betas[-1])


plot_maximize_logl(locations, alpha_t, beta_t)
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 10
         Function evaluations: 39
Optimization terminated successfully.
         Current function value: 0.898559
         Iterations: 104
         Function evaluations: 197
Optimization terminated successfully.
         Current function value: 2.181169
         Iterations: 86
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 7.644875
         Iterations: 85
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 9.410418
         Iterations: 87
         Function evaluations: 164
Optimization terminated successfully.
         Current function value: 17.834954
         Iterations: 86
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 19.446122
         Iterations: 94
         Function evaluations: 177
Optimization terminated successfully.
         Current function value: 20.190702
         Iterations: 94
         Function evaluations: 171
Optimization terminated successfully.
         Current function value: 20.627525
         Iterations: 90
         Function evaluations: 172
Optimization terminated successfully.
         Current function value: 22.856079
         Iterations: 77
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 25.308925
         Iterations: 89
         Function evaluations: 168
<ipython-input-45-6a1108c4c769>:5: RuntimeWarning: invalid value encountered in log
likelihood = len(locations) * np.log(beta/np.pi)
<ipython-input-45-6a1108c4c769>:14: RuntimeWarning: Maximum number of function evaluations has been exceeded.
  [alpha, beta] = fmin(log_likelihood, (0, 1), args=(data[:k],))
Optimization terminated successfully.
         Current function value: 27.014560
         Iterations: 80
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 32.265839
         Iterations: 85
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 38.483444
         Iterations: 87
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 43.508164
         Iterations: 86
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 46.596925
         Iterations: 98
         Function evaluations: 186
Optimization terminated successfully.
         Current function value: 48.454299
         Iterations: 83
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 50.244967
         Iterations: 83
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 52.133760
         Iterations: 84
         Function evaluations: 158
Optimization terminated successfully.
         Current function value: 54.230665
         Iterations: 82
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 56.810668
         Iterations: 90
         Function evaluations: 168
Optimization terminated successfully.
```

```
         Current function value: 58.207053
         Iterations: 89
         Function evaluations: 168
Optimization terminated successfully.
         Current function value: 61.041140
         Iterations: 87
         Function evaluations: 166
Optimization terminated successfully.
         Current function value: 64.232456
         Iterations: 84
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 67.123962
         Iterations: 81
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 69.299918
         Iterations: 85
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 70.903428
         Iterations: 81
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 81.448277
         Iterations: 89
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 83.804360
         Iterations: 83
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 86.149082
         Iterations: 82
         Function evaluations: 160
Optimization terminated successfully.
         Current function value: 88.074644
         Iterations: 87
         Function evaluations: 166
Optimization terminated successfully.
         Current function value: 89.930853
         Iterations: 79
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 92.681328
         Iterations: 89
         Function evaluations: 167
Optimization terminated successfully.
         Current function value: 99.286250
         Iterations: 92
         Function evaluations: 173
Optimization terminated successfully.
         Current function value: 102.287242
         Iterations: 84
         Function evaluations: 160
Optimization terminated successfully.
         Current function value: 103.799865
         Iterations: 80
         Function evaluations: 154
Optimization terminated successfully.
         Current function value: 106.080888
         Iterations: 80
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 107.605565
         Iterations: 79
         Function evaluations: 153
Optimization terminated successfully.
         Current function value: 113.058277
         Iterations: 93
         Function evaluations: 178
Optimization terminated successfully.
         Current function value: 114.505336
         Iterations: 82
         Function evaluations: 154
Optimization terminated successfully.
         Current function value: 115.913102
         Iterations: 96
         Function evaluations: 182
Optimization terminated successfully.
         Current function value: 117.751392
         Iterations: 79
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 119.367589
         Iterations: 77
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 120.875940
```

```
         Iterations: 88
         Function evaluations: 165
Optimization terminated successfully
Optimization terminated successfully.
         Current function value: 124.737112
         Iterations: 85
         Function evaluations: 165
Optimization terminated successfully.
         Current function value: 129.670497
         Iterations: 79
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 137.022564
         Iterations: 84
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 138.590052
         Iterations: 85
         Function evaluations: 165
Optimization terminated successfully.
         Current function value: 140.275050
         Iterations: 77
         Function evaluations: 146
Optimization terminated successfully.
         Current function value: 145.009137
         Iterations: 77
         Function evaluations: 147
Optimization terminated successfully.
         Current function value: 147.811478
         Iterations: 84
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 150.321737
         Iterations: 78
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 151.758095
         Iterations: 81
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 156.753771
         Iterations: 75
         Function evaluations: 146
Optimization terminated successfully.
         Current function value: 158.454630
         Iterations: 79
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 161.544737
         Iterations: 81
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 163.644613
         Iterations: 80
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 165.661547
         Iterations: 80
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 170.372727
         Iterations: 75
         Function evaluations: 144
Optimization terminated successfully.
         Current function value: 173.232661
         Iterations: 83
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 174.725255
         Iterations: 78
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 176.658109
         Iterations: 76
         Function evaluations: 148
Optimization terminated successfully.
         Current function value: 180.110992
         Iterations: 82
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 183.914366
         Iterations: 78
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 185.635596
         Iterations: 76
         Function evaluations: 148
Optimization terminated successfully.
         Current function value: 193.930732
```

```
         Iterations: 75
         Function evaluations: 147
Optimization terminated successfully.
         Current function value: 195.423769
         Iterations: 84
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 200.483857
         Iterations: 82
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 202.094245
         Iterations: 86
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 203.595402
         Iterations: 83
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 205.049057
         Iterations: 82
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 207.860953
         Iterations: 76
         Function evaluations: 144
Optimization terminated successfully.
         Current function value: 211.460461
         Iterations: 81
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 214.029520
         Iterations: 81
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 216.645916
         Iterations: 77
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 223.085923
         Iterations: 77
         Function evaluations: 148
Optimization terminated successfully.
         Current function value: 224.701822
         Iterations: 75
         Function evaluations: 146
Optimization terminated successfully.
         Current function value: 226.255690
         Iterations: 81
         Function evaluations: 158
Optimization terminated successfully.
         Current function value: 228.090479
         Iterations: 75
         Function evaluations: 146
Optimization terminated successfully.
         Current function value: 230.369384
         Iterations: 85
         Function evaluations: 158
Optimization terminated successfully.
         Current function value: 231.806920
         Iterations: 78
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 234.619301
         Iterations: 74
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 236.252667
         Iterations: 74
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 237.772496
         Iterations: 80
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 239.912695
         Iterations: 78
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 241.298176
         Iterations: 77
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 243.055085
         Iterations: 78
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 244.628091
         Iterations: 84
```

```
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 246.030436
         Iterations: 85
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 247 410376
         Current function value: 247.410376
         Iterations: 86
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 249.646775
         Iterations: 80
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 253.717529
         Iterations: 74
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 255.068693
         Iterations: 75
         Function evaluations: 145
Optimization terminated successfully.
         Current function value: 257.714717
         Iterations: 80
         Function evaluations: 154
Optimization terminated successfully.
         Current function value: 259.017250
         Iterations: 80
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 260.625274
         Iterations: 87
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 263.544657
         Iterations: 79
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 268.885058
         Iterations: 80
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 271.699869
         Iterations: 79
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 273.029173
         Iterations: 78
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 274.690860
         Iterations: 80
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 276.182424
         Iterations: 80
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 282.041240
         Iterations: 78
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 283.331904
         Iterations: 81
         Function evaluations: 154
Optimization terminated successfully.
         Current function value: 288.222701
         Iterations: 80
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 290.788539
         Iterations: 86
         Function evaluations: 158
Optimization terminated successfully.
         Current function value: 295.751863
         Iterations: 85
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 302.032656
         Iterations: 76
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 303.756361
         Iterations: 72
         Function evaluations: 140
Optimization terminated successfully.
         Current function value: 306.104533
         Iterations: 72
```

```
         Function evaluations: 142
Optimization terminated successfully.
         Current function value: 309.645428
         Iterations: 78
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 310.998849
         Iterations: 76
         Function evaluations: 148
Optimization terminated successfully.
         Current function value: 312.741050
         Iterations: 82
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 316.785037
         Iterations: 72
         Function evaluations: 140
Optimization terminated successfully.
         Current function value: 318.246655
         Iterations: 72
         Function evaluations: 141
Optimization terminated successfully.
         Current function value: 319.990919
         Iterations: 73
         Function evaluations: 142
Optimization terminated successfully.
         Current function value: 324.825810
         Iterations: 84
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 333.450131
         Iterations: 75
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 335.966797
         Iterations: 79
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 337.402793
         Iterations: 76
         Function evaluations: 146
Optimization terminated successfully.
         Current function value: 340.020481
         Iterations: 90
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 341.502172
         Iterations: 89
         Function evaluations: 164
Optimization terminated successfully.
         Current function value: 343.701537
         Iterations: 86
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 347.106162
         Iterations: 88
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 349.540678
         Iterations: 81
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 354.252588
         Iterations: 84
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 355.900194
         Iterations: 103
         Function evaluations: 192
Optimization terminated successfully.
         Current function value: 358.386896
         Iterations: 88
         Function evaluations: 170
Optimization terminated successfully.
         Current function value: 363.858308
         Iterations: 79
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 365.286566
         Iterations: 77
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 366.900914
         Iterations: 74
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 368.898213
         Iterations: 89
```

```
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 370.262323
         Iterations: 85
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 379.016732
         Iterations: 79
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 380.842308
         Iterations: 81
         Iterations: 81
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 383.228490
         Iterations: 84
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 384.785304
         Iterations: 80
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 386.200290
         Iterations: 78
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 388.861944
         Iterations: 79
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 390.290678
         Iterations: 76
         Function evaluations: 145
Optimization terminated successfully.
         Current function value: 393.179383
         Iterations: 85
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 394.721305
         Iterations: 84
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 396.150884
         Iterations: 82
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 402.837239
         Iterations: 75
         Function evaluations: 145
Optimization terminated successfully.
         Current function value: 404.373868
         Iterations: 81
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 407.602372
         Iterations: 85
         Function evaluations: 156
Optimization terminated successfully.
         Current function value: 410.008722
         Iterations: 84
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 411.399714
         Iterations: 74
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 413.591007
         Iterations: 84
         Function evaluations: 154
Optimization terminated successfully.
         Current function value: 415.467720
         Iterations: 86
         Function evaluations: 158
Optimization terminated successfully.
         Current function value: 416.925867
         Iterations: 84
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 419.522106
         Iterations: 91
         Function evaluations: 168
Optimization terminated successfully.
         Current function value: 421.936961
         Iterations: 84
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 424.538795
         Iterations: 75
```

```
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 426.327458
         Iterations: 80
         Function evaluations: 154
Optimization terminated successfully.
         Current function value: 430.445037
         Iterations: 77
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 440.428913
         Iterations: 73
         Function evaluations: 143
Optimization terminated successfully.
         Current function value: 441.784183
         Iterations: 76
         Function evaluations: 146
Optimization terminated successfully.
         Current function value: 447.248164
         Iterations: 85
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 448.826613
         Iterations: 81
         Function evaluations: 153
Optimization terminated successfully.
         Current function value: 450.261298
         Iterations: 85
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 451.674732
         Iterations: 84
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 453.009800
         Iterations: 83
         Function evaluations: 157
Optimization terminated successfully.
         Current function value: 456.343129
         Iterations: 87
         Function evaluations: 166
Optimization terminated successfully.
         Current function value: 459.355816
         Iterations: 75
         Function evaluations: 145
Optimization terminated successfully.
         Current function value: 462.712419
         Iterations: 86
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 464.638954
         Iterations: 83
         Function evaluations: 159
Optimization terminated successfully.
         Current function value: 466.793784
         Iterations: 86
         Function evaluations: 162
Optimization terminated successfully.
         Current function value: 472.890358
         Iterations: 77
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 477.349859
         Iterations: 91
         Function evaluations: 168
Optimization terminated successfully.
         Current function value: 483.603146
         Iterations: 77
         Function evaluations: 148
Optimization terminated successfully.
         Current function value: 485.003674
         Iterations: 79
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 486.664846
         Iterations: 78
         Function evaluations: 149
Optimization terminated successfully.
         Current function value: 496.317641
         Iterations: 77
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 498.521531
         Iterations: 79
         Function evaluations: 153
Optimization terminated successfully.
         Current function value: 501.882182
         Iterations: 77
```

```
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 503.383302
         Iterations: 77
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 504.991433
         Iterations: 78
         Function evaluations: 152
Optimization terminated successfully.
         Current function value: 509.052355
         Iterations: 81
         Function evaluations: 153
Optimization terminated successfully.
         Current function value: 511.747097
         Iterations: 81
         Function evaluations: 153
         Function evaluations: 153
Optimization terminated successfully.
         Current function value: 513.830133
         Iterations: 82
         Function evaluations: 155
Optimization terminated successfully.
         Current function value: 515.676904
         Iterations: 79
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 517.201032
         Iterations: 86
         Function evaluations: 163
Optimization terminated successfully.
         Current function value: 521.005817
         Iterations: 85
         Function evaluations: 160
Optimization terminated successfully.
         Current function value: 522.718773
         Iterations: 89
         Function evaluations: 172
Optimization terminated successfully.
         Current function value: 524.126741
         Iterations: 90
         Function evaluations: 172
Optimization terminated successfully.
         Current function value: 526.233875
         Iterations: 84
         Function evaluations: 158
Optimization terminated successfully.
         Current function value: 527.867788
         Iterations: 88
         Function evaluations: 161
Optimization terminated successfully.
         Current function value: 533.017629
         Iterations: 77
         Function evaluations: 148
Optimization terminated successfully.
         Current function value: 535.002388
         Iterations: 76
         Function evaluations: 147
Optimization terminated successfully.
         Current function value: 536.971966
         Iterations: 77
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 538.406220
         Iterations: 78
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 539.799025
         Iterations: 77
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 543.127376
         Iterations: 78
         Function evaluations: 150
Optimization terminated successfully.
         Current function value: 546.051274
         Iterations: 78
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 548.515494
         Iterations: 79
         Function evaluations: 153
Optimization terminated successfully.
         Current function value: 552.751043
         Iterations: 79
         Function evaluations: 151
Optimization terminated successfully.
         Current function value: 554.131287
         Iterations: 79
```

```
        Function evaluations: 149
Optimization terminated successfully.
        Current function value: 557.188955
        Iterations: 85
        Function evaluations: 157
---------------------------------------------------------------------------FileNotFoundError
Traceback (most recent call last)
<ipython-input-45-6a1108c4c769> in <cell line: 32>()
```

```
import  30 pandas  as pd
     31
import---> 32 numpy  plot_maximize_loglas np(locations, alpha_t, beta_t)
a=pd.read_csv("train.csv"8 frames ,sep=';')
print/usr/local/lib/python3.10/dist-packages/PIL/Image.py(a) in save(self, fp, format, **params)
     2235            fp = builtins.open(filename, "r+b")
p=a ['hour'2236            ]  else:
-> 2237            fp = builtins.open(filename, "w+b")
m=np.mean  2238 (p)
     2239       try:
sd=np.std(p)#sigma value
var=np.var(p)#sigma square value
m, sd, var
```

```
         id  year  hour  season  holiday  workingday  weather   temp   atemp  \
0         3  2012    23       3        0           0        2  23.78  27.275
1         4  2011     8       3        0           0        1  27.88  31.820
2         5  2012     2       1        0           1        1  20.50  24.240
3         7  2011    20       3        0           1        3  25.42  28.790
4         8  2011    17       3        0           1        3  26.24  28.790
...     ...   ...   ...     ...      ...         ...      ...    ...     ...
7684  10882  2012    18       1        0           1        1  13.94  15.150
7685  10883  2012     3       1        0           1        1   9.02  11.365
7686  10884  2012    15       2        0           0        1  21.32  25.000
7687  10885  2011    19       4        0           1        1  12.30  14.395
7688  10886  2012    21       3        0           1        1  30.34  34.850

      humidity  windspeed  count
0           73    11.0014    133
1           57     0.0000    132
2           59     0.0000     19
3           83    19.9995     58
4           89     0.0000    285
...        ...        ...    ...
7684        42    22.0028    457
7685        51    11.0014      1
7686        19    27.9993    626
7687        45    15.0013    217
7688        66     7.0015    381

[7689 rows x 12 columns]
(11.56535310183379, 6.915326938018648, 47.82174665968637)
```

```
t=np.array(a['temp'])
tm=np.mean(t) tsd=np.std(t)#sigma
value/std.dev
tvar=np.var(t)#variance x=29
l=np.log(np.sqrt(2*3.14))
e=np.log(tsd) #std.dev
f=(x-tm)**2 #mean
g=2*(tvar**2) #variance
h=f/g i=-l-e print(i-h)
```

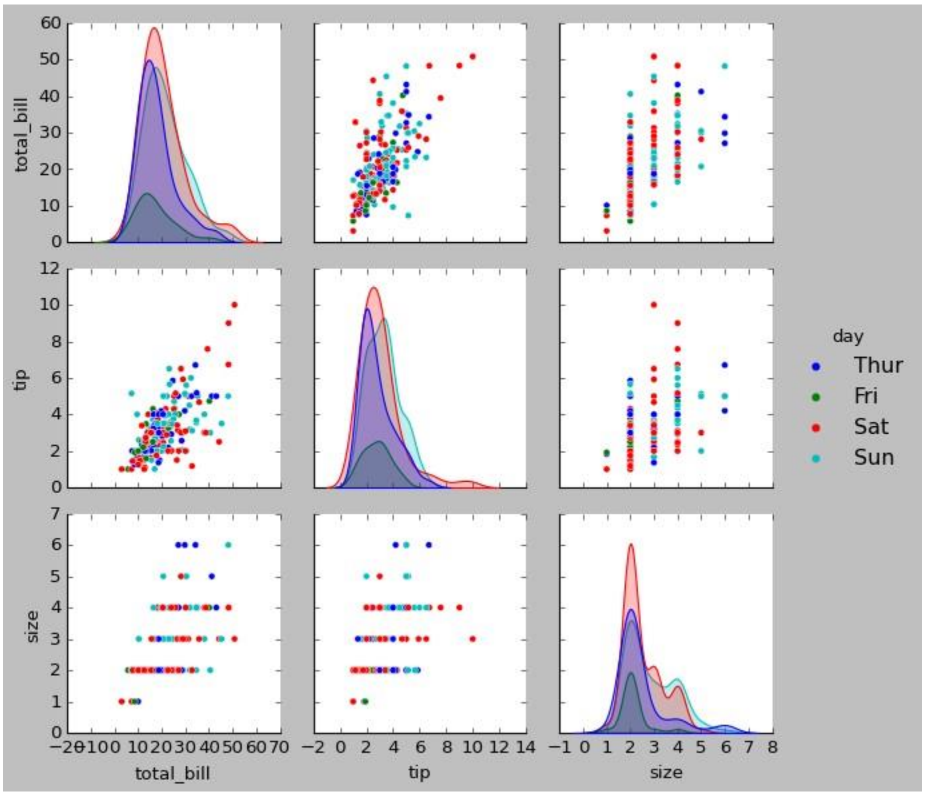
```
-2.9860025235468406
```

```
import pandas as pd import numpy
as np
a=pd.read_csv("test.csv",sep=';')
```

```
print(a) p=a['hour'] m=np.mean(p)
sd=np.std(p)#sigma value
var=np.var(p)#sigma square value
m, sd, var
```

```
      Unnamed: 0  year  hour  season  holiday  workingday  weather   temp  \
0              1  2012    21       3        0           0        1  29.52
1              2  2012     3       2        0           0        1  23.78
2              6  2011    10       1        0           1        3  16.40
3             14  2012    19       1        0           1        1  13.94
4             17  2011    23       3        0           1        2  26.24  ...          ...  ...  ...    ...     ...
             ...    ...   ...
3191       10868  2011     9       4        0           1        1  16.40
3192       10871  2011    12       4        0           1        3  18.04
3193       10872  2011    19       3        0           1        1  30.34
3194       10873  2012     0       4        0           1        2  13.12   3195       10874  2012    10       1        0           1
3  12.30

       atemp  humidity  windspeed
0     34.850        79     6.0032
1     27.275        83     0.0000
2     20.455         0    11.0014
3     15.150        46    19.9995
4     30.305        73    11.0014  ...      ...      ...       ...
3191  20.455        87     6.0032
3192  21.970       100     8.9981
3193  34.090        55    16.9979
3194  16.665        66     7.0015
3195  14.395        87    16.9979

[3196 rows x 11 columns] (11.482478097622028,
6.915772969192248, 47.82791576141015)
```

```
import seaborn import
matplotlib.pyplot as plt df =
seaborn.load_dataset('tips')
seaborn.pairplot(df, hue='day')
plt.show()
```



Linear Regrssion on US Housing Price

Linear Regrssion on US Housing Price

In statistics, linear regression is a linear approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

Linear regression models are often tted using the least squares approach, but they may also be tted in other ways, such as by minimizing the "lack of t" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares loss function as in ridge regression (L_2-norm penalty) and lasso (L_1-norm penalty). Conversely, the least squares approach can be used to t models that are not linear models. Thus, although the terms "least squares" and "linear model" are closely linked, they are not synonymous.

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

```
df = pd.read_csv("USA_Housing.csv")
df.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael F 674\nLaura

Check basic info on the data set

'info()' method to check the data types and number

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
0   Avg. Area Income              5000 non-null   float64
1   Avg. Area House Age           5000 non-null   float64
2   Avg. Area Number of Rooms     5000 non-null   float64
3   Avg. Area Number of Bedrooms  5000 non-null   float64
4   Area Population               5000 non-null   float64
5   Price                         5000 non-null   float64  6   Address                        5000 non-null   object dtypes:
    float64(6), object(1) memory usage: 273.6+ KB
```

'describe()' method to get the statistical summary of the various features of the data set

```
df.describe(percentiles=[0.1,0.25,0.5,0.75,0.9])
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.98133036163.516039		1.232073e+06

std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
10%	55047.633980	4.697755	5.681951	2.310000	23502.845262	7.720318e+05
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
90%	82081.188283	7.243978	8.274222	6.100000	48813.618633	1.684621e+06

'columns' method to get the names of the columns (features)

```
df.columns
```

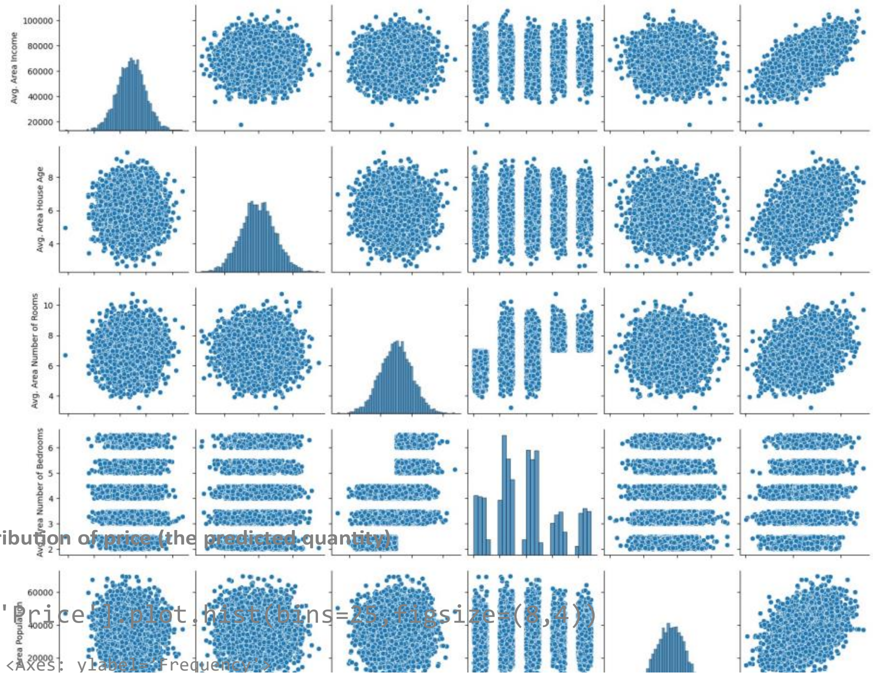
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Basic plotting and visualization on the data set

Pairplots using seaborn

```
sns.pairplot(df)
```

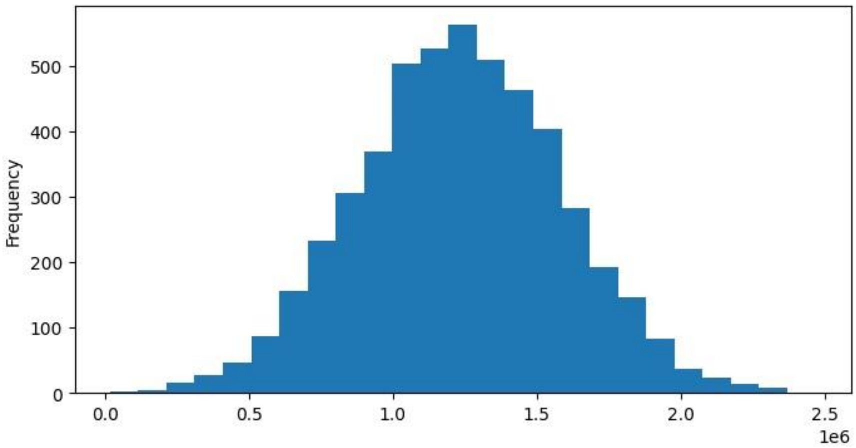
<seaborn.axisgrid.PairGrid at 0x7e90c862b340>



Distribution of price (the predicted quantity)

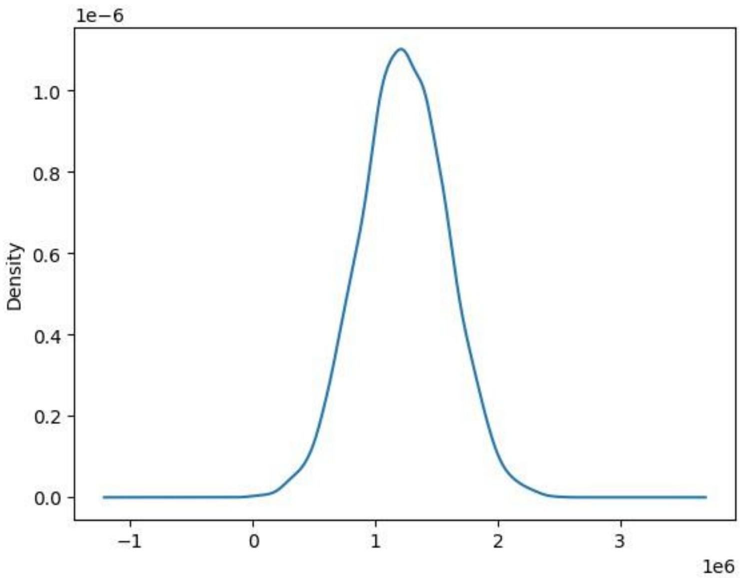
```
df['Price'].plot.hist(bins=25,figsize=(8,4))
```

<Axes: ylabel='Frequency'>




```
df['Price'].plot.density()
```

<Axes: ylabel='Density'>



Correlation matrix and heatmap

```
df.corr()
```

```
<ipython-input-9-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in
df.corr()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1.000000	-0.002007	-0.011032	0.019788	-0.016234	0.639734
Avg. Area House Age	-0.002007	1.000000	-0.009428	0.006149	-0.018743	0.452543
Avg. Area Number of Rooms	-0.011032	-0.009428	1.000000	0.462695	0.002040	0.335664

```
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(),annot=True,linewidths=2)
```

```
hon-input-10-73d88c5a3f1a>:2: FutureWarning: The default value of numeric_only i
.heatmap(df.corr(),annot=True,linewidths=2)
: >
```

Feature and variable sets

Make a list of data frame column names

```
l_column = list(df.columns) # Making a list out of column names
len_feature = len(l_column) # Length of column vector list
l_column
```

```
['Avg. Area Income',
 'Avg. Area House Age',
 'Avg. Area Number of Rooms',
```

```
 'Avg. Area Number of Bedrooms',
 'Area Population',
 'Price',
 'Address']
```

Put all the numerical features in X and Price in y, ignore Address which is string for linear regression

```
X = df[l_column[0:len_feature-2]]
y = df[l_column[len_feature-2]]

print("Feature set size:",X.shape)
print("Variable set size:",y.shape)
```

```
Feature set size: (5000, 5)
Variable set size: (5000,)
```

```
X.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
0	79545.458574	5.682861	7.009188	4.09	23086.800503
1	79248.642455	6.002900	6.730821	3.09	40173.072174
2	61287.067179	5.865890	8.512727	5.13	36882.159400
3	63345.240046	7.188236	5.586729	3.26	34310.242831

```
y.head()
```

```
0    1.059034e+06
1    1.505891e+06
2    1.058988e+06
3    1.260617e+06
4    6.309435e+05
Name: Price, dtype: float64
```

Test-train split

Import train_test_split function from scikit-learn

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

Create X and y train and test splits in one command using a split ratio and a random seed

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=123)
```

Check the size and shape of train/test splits (it should be in the ratio as per test_size parameter above)

```
print("Training feature set size:",X_train.shape)
print("Test feature set size:",X_test.shape)
print("Training variable set size:",y_train.shape)
print("Test variable set size:",y_test.shape)
```

```
Training feature set size: (3500, 5)
Test feature set size: (1500, 5)
Training variable set size: (3500,)
Test variable set size: (1500,)
```

Model t and training

Import linear regression model estimator from scikit-learn and instantiate

```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
lm = LinearRegression() # Creating a Linear Regression object 'lm'
```

Fit the model on to the instantiated object itself

```
lm.fit(X_train,y_train) # Fit the linear model on to the 'lm' object itself i.e. no need to set
```

```
▾LinearRegression
LinearRegression()
```

Check the intercept and coe cients and put them in a DataFrame

```
print("The intercept term of the linear model:", lm.intercept_)
```

```
The intercept term of the linear model: -2631028.9017454907
```

```
print("The coefficients of the linear model:", lm.coef_)
```

```
The coefficients of the linear model: [2.15976020e+01 1.65201105e+05 1.19061464e+05 3.21258561e+03
 1.52281212e+01]
```

```
#idict = {'Coefficients':lm.intercept_}
#idf = pd.DataFrame(data=idict,index=['Intercept'])
cdf = pd.DataFrame(data=lm.coef_, index=X_train.columns, columns=["Coefficients"])
#cdf=pd.concat([idf,cdf], axis=0)
cdf
```

	Coefficients
Avg. Area Income	21.597602
Avg. Area House Age	165201.104954
Avg. Area Number of Rooms	119061.463868
Avg. Area Number of Bedrooms	3212.585606
Area Population	15.228121

Calculation of standard errors and t-statistic for the coe cients

```
n=X_train.shape[0]
k=X_train.shape[1]
dfN = n-k
train_pred=lm.predict(X_train)
train_error = np.square(train_pred - y_train)
sum_error=np.sum(train_error) se=[0,0,0,0,0]
for i in range(k):
    r = (sum_error/dfN)
    r = r/np.sum(np.square(X_train[list(X_train.columns)[i]]-X_train[list(X_train.columns)[i]].m
se[i]=np.sqrt(r) cdf['Standard Error']=se
cdf['t-statistic']=cdf['Coefficients']/cdf['Standard Error']
cdf
```

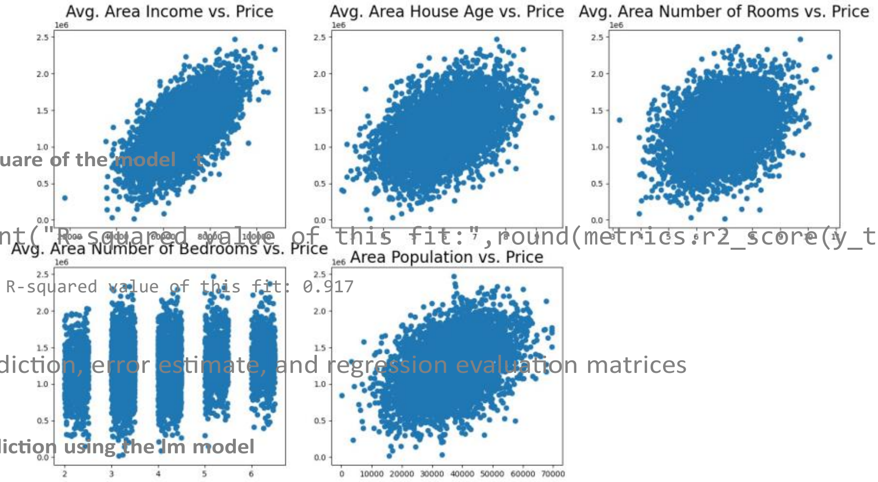
	Coefficients	Standard Error	t-statistic
Avg. Area Income	21.597602	0.160361	134.681505
Avg. Area House Age	165201.104954	1722.412068	95.912649
Avg. Area Number of Rooms	119061.463868	1696.546476	70.178722
Avg. Area Number of Bedrooms	3212.585606	1376.451759	2.333962
Area Population	15.228121	0.169882	89.639472

```
print("Therefore, features arranged in the order of importance for predicting the house price\n"
l=list(cdf.sort_values('t-statistic',ascending=False).index) print(' > \n'.join(l))
```

```
Therefore, features arranged in the order of importance for predicting the house price ---------------------------------------------
---------------------------------------------
Avg. Area Income >
Avg. Area House Age >
Area Population >
Avg. Area Number of Rooms >
Avg. Area Number of Bedrooms
```

```
l=list(cdf.index)
from matplotlib import gridspec
fig = plt.figure(figsize=(18, 10))
gs = gridspec.GridSpec(2,3)
#f, ax = plt.subplots(nrows=1,ncols=len(l), sharey=True)
ax0 = plt.subplot(gs[0])
ax0.scatter(df[l[0]],df['Price'])
ax0.set_title(l[0]+" vs. Price", fontdict={'fontsize':20})

ax1 = plt.subplot(gs[1])
ax1.scatter(df[l[1]],df['Price'])
ax1.set_title(l[1]+" vs. Price",fontdict={'fontsize':20})

ax2 = plt.subplot(gs[2])
ax2.scatter(df[l[2]],df['Price'])
ax2.set_title(l[2]+" vs. Price",fontdict={'fontsize':20})

ax3 = plt.subplot(gs[3])
ax3.scatter(df[l[3]],df['Price'])
ax3.set_title(l[3]+" vs. Price",fontdict={'fontsize':20})

ax4 = plt.subplot(gs[4])
ax4.scatter(df[l[4]],df['Price'])
ax4.set_title(l[4]+" vs. Price",fontdict={'fontsize':20})
```

```
Text(0.5, 1.0, 'Area Population vs. Price')
```



R-square of the model fit

```
print("R-squared value of this fit:",round(metrics.r2_score(y_train,train_pred),3))
```

```
R-squared value of this fit: 0.917
```

Prediction, error estimate, and regression evaluation matrices

Prediction using the lm model

```
predictions = lm.predict(X_test)
print ("Type of the predicted object:", type(predictions))
print ("Size of the predicted object:", predictions.shape)
```

```
Type of the predicted object: <class 'numpy.ndarray'>
Size of the predicted object: (1500,)
```

Scatter plot of predicted price and y_test set to see if the data fall on a 45 degree straight line

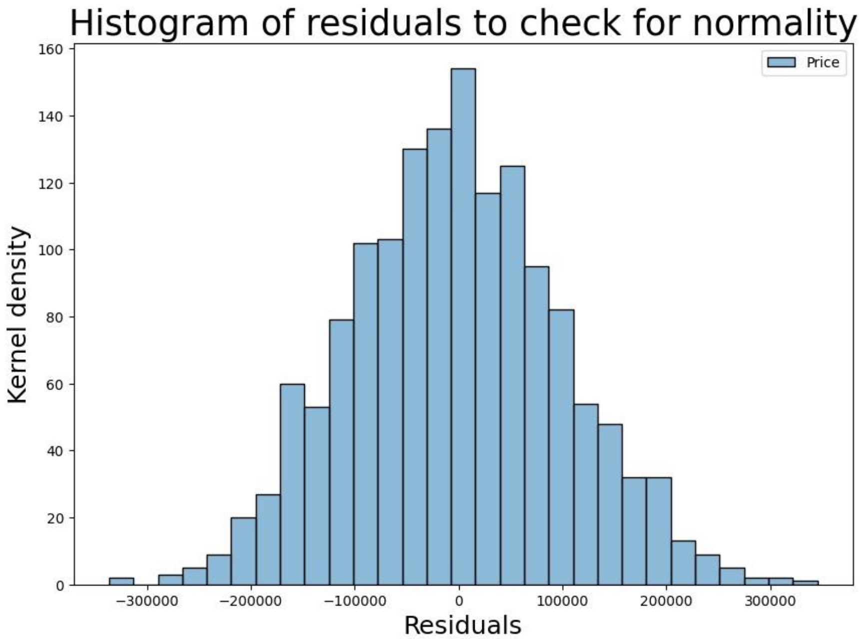
```
plt.figure(figsize=(10,7))
plt.title("Actual vs. predicted house prices",fontsize=25)
plt.xlabel("Actual test set house prices",fontsize=18)
plt.ylabel("Predicted house prices", fontsize=18)
plt.scatter(x=y_test,y=predictions)
```

```
<matplotlib.collections.PathCollection at 0x7e90bf811d20>
```

Plotting histogram of the residuals i.e. predicted errors (expect a normally distributed pattern)

```
plt.figure(figsize=(10,7))
plt.title("Histogram of residuals to check for normality",fontsize=25)
plt.xlabel("Residuals",fontsize=18) plt.ylabel("Kernel density", fontsize=18)
sns.histplot([y_test-predictions])
```

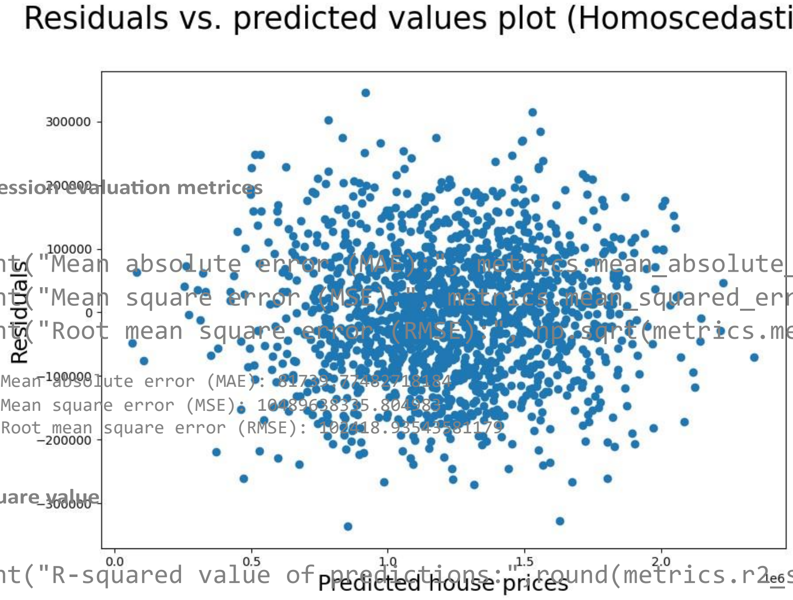
```
<Axes: title={'center': 'Histogram of residuals to check for normality'},
xlabel='Residuals', ylabel='Kernel density'>
```



Scatter plot of residuals and predicted values (Homoscedasticity)

```
plt.figure(figsize=(10,7))
plt.title("Residuals vs. predicted values plot (Homoscedasticity)\n",fontsize=25)
plt.xlabel("Predicted house prices",fontsize=18) plt.ylabel("Residuals",
fontsize=18) plt.scatter(x=predictions,y=y_test-predictions)
```

```
<matplotlib.collections.PathCollection at 0x7e90bf7309d0>
```



Regression evaluation metrices

```
print("Mean absolute error (MAE):", metrics.mean_absolute_error(y_test,predictions))
print("Mean square error (MSE):", metrics.mean_squared_error(y_test,predictions))
print("Root mean square error (RMSE):", np.sqrt(metrics.mean_squared_error(y_test,predictions)))
```

```
Mean absolute error (MAE): 81739.77482718184
Mean square error (MSE): 10489638335.804983
Root mean square error (RMSE): 102418.93543581179
```

R-square value

```
print("R-squared value of predictions:",round(metrics.r2_score(y_test,predictions),3))
```

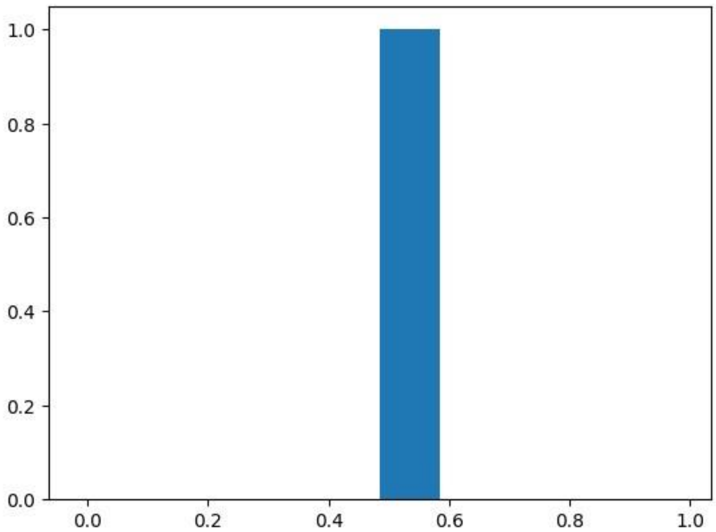
```
R-squared value of predictions: 0.919
```

```
#compute minmax value for observed price and expected price
import numpy as np min=np.min(predictions/6000)
max=np.max(predictions/12000) print(min, max)
```

```
10.57339854753646 195.14363973516853
```

```
#Compute MinMax value for Price=100
L = (100 - min)/(max - min)
L plt.hist(L)
```

```
(array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
 array([-0.01548743,  0.08451257,  0.18451257,  0.28451257,  0.38451257,
         0.48451257,  0.58451257,  0.68451257,  0.78451257,  0.88451257,
         0.98451257]),
 <BarContainer object of 10 artists>)
```



Logistic Regression with Titanic data set

Import packages and dataset

```
import pandas as pd import
seaborn as sns import
matplotlib.pyplot as plt import
seaborn as sns import nbconvert
```

```
dataframe = pd.read_csv("titanic_train.csv")
dataframe.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2

Check basic info about the data set including missing value

```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
0   PassengerId  891 non-null    int64
1   Survived     891 non-null    int64
2   Pclass       891 non-null    int64
3   Name         891 non-null    object
4   Sex          891 non-null    object
5   Age          714 non-null    float64
6   SibSp        891 non-null    int64
7   Parch        891 non-null    int64
8   Ticket       891 non-null    object
9   Fare         891 non-null    float64
10  Cabin        204 non-null    object  11  Embarked     889 non-null    object dtypes: float64(2), int64(5), object(5) memory
    usage: 83.7+ KB
```

```
d=dataframe.describe()
d
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	F
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329

Exploratory analysis and plots

Plot a bar diagram to check the number of numeric entries

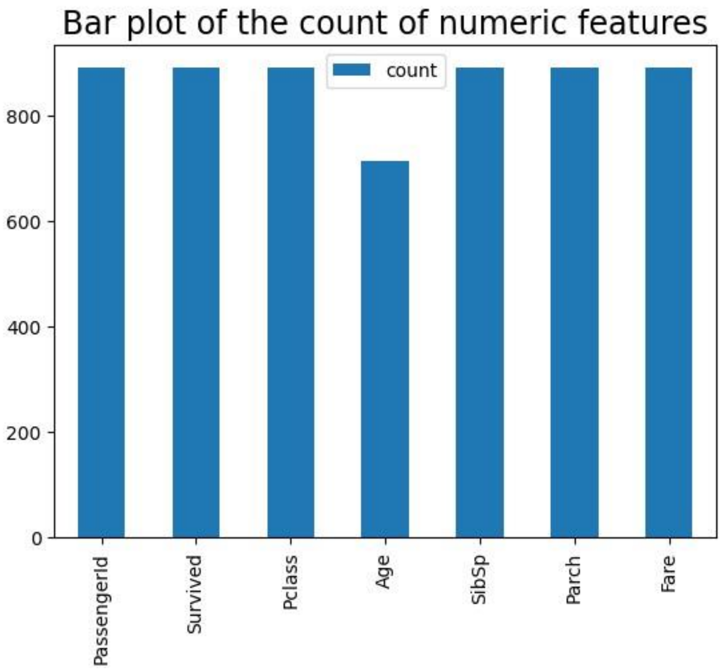
From the bar diagram, it shows that there are some age entries missing as the number of count for 'Age' is less than the other counts. We can do some impute/transformation of the data to ll-up the missing entries.

```
result = dataframe.dtypes
result
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

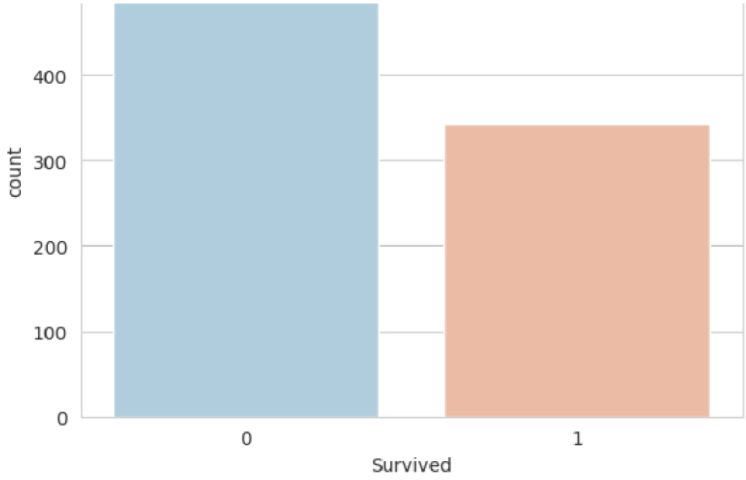
```
dT=d.T
dT.plot.bar(y='count') plt.title("Bar plot of the count of numeric
features",fontsize=17)
```

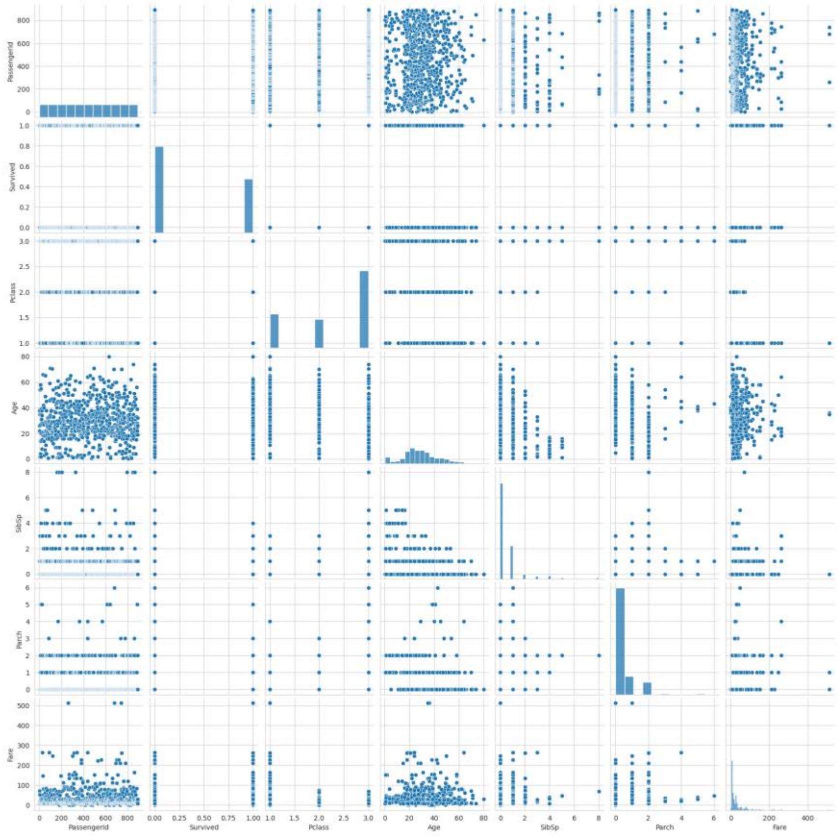
```
Text(0.5, 1.0, 'Bar plot of the count of numeric features')
```



Check the relative size of survived and not-survived

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=dataframe,palette='RdBu_r')
sns.pairplot(dataframe)
```



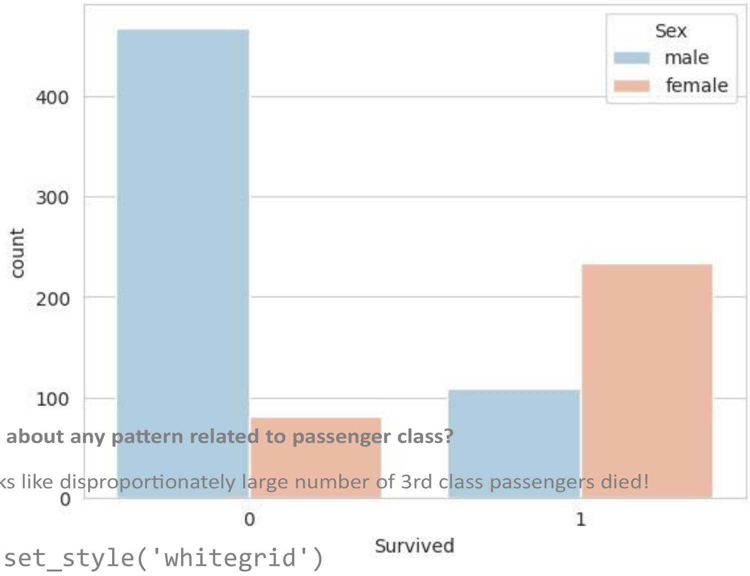


Is there a pattern for the survivability based on sex?

It looks like more female survived than males!

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=dataframe,palette='RdBu_r')
```

```
<Axes: xlabel='Survived', ylabel='count'>
```

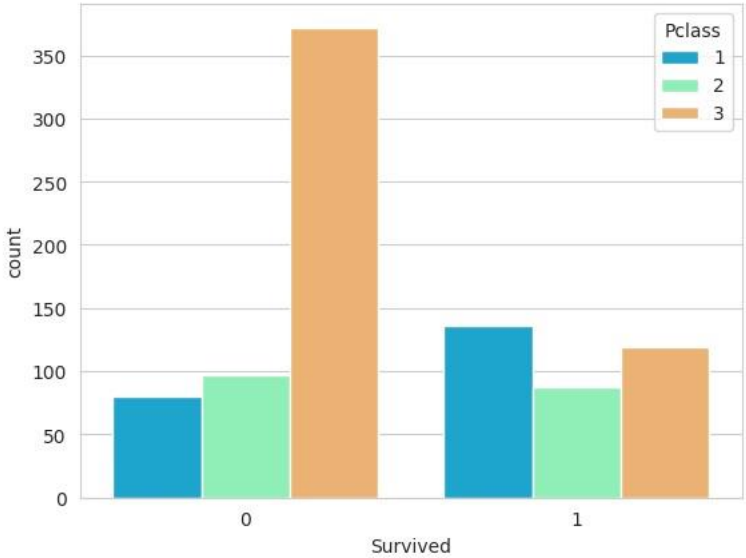


What about any pattern related to passenger class?

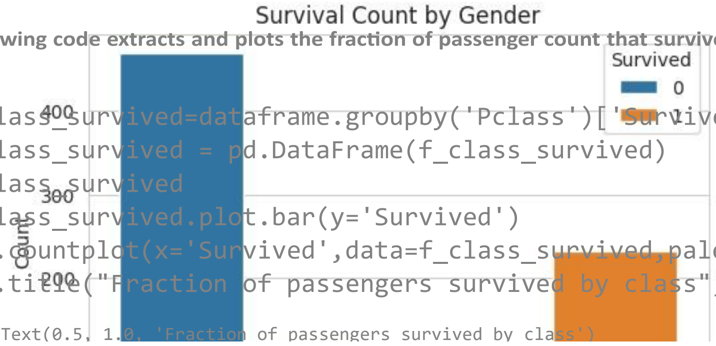
It looks like disproportionately large number of 3rd class passengers died!

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=dataframe,palette='rainbow')
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



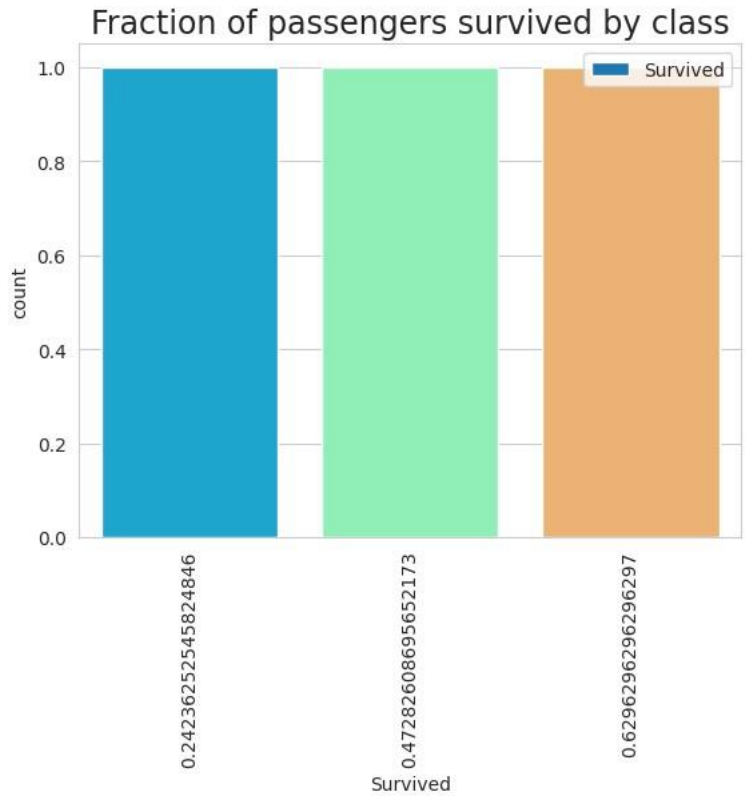

```
sns.set_style('whitegrid')
plt.figure(figsize=(6, 4))
sns.countplot(data=dataframe, x='Sex', hue='Survived')
plt.title("Survival Count by Gender")
plt.xlabel("Gender") plt.ylabel("Count") plt.show()
```



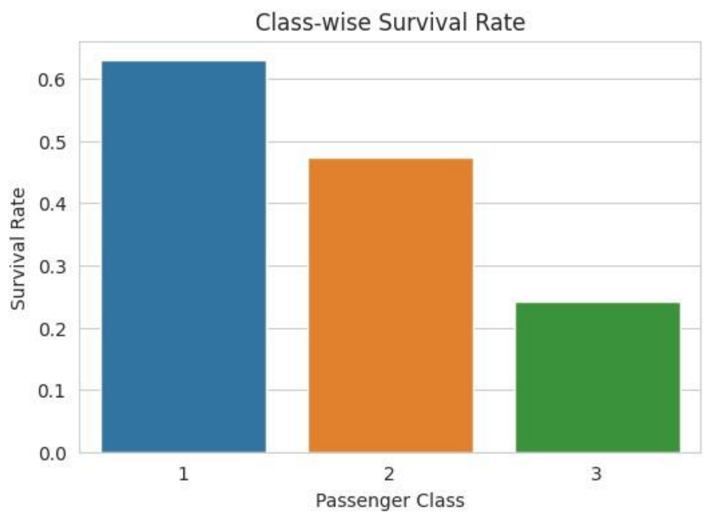
Following code extracts and plots the fraction of passenger count that survived, by each class

```
f_class_survived=dataframe.groupby('Pclass')['Survived'].mean()
f_class_survived = pd.DataFrame(f_class_survived)
f_class_survived
f_class_survived.plot.bar(y='Survived')
sns.countplot(x='Survived',data=f_class_survived,palette='rainbow')
plt.title("Fraction of passengers survived by class",fontsize=17)
```

```
Text(0.5, 1.0, 'Fraction of passengers survived by class')
```




```
class_survival = dataframe.groupby('Pclass')['Survived'].mean()
plt.figure(figsize=(6, 4))
sns.barplot(x=class_survival.index, y=class_survival.values)
plt.title("Class-wise Survival Rate") plt.xlabel("Passenger
Class") plt.ylabel("Survival Rate") plt.show()
```

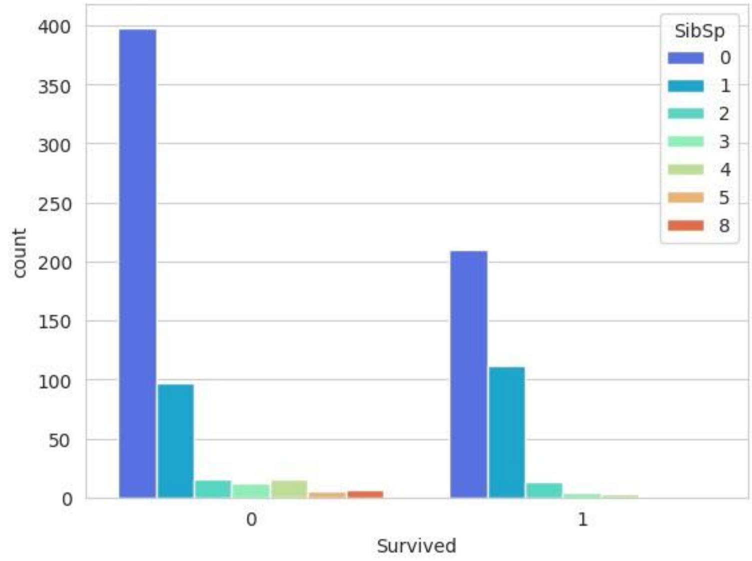


What about any pattern related to having sibling and spouse?

It looks like there is a weak trend that chance of survibility increased if there were more number of sibling or spouse

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='SibSp',data=dataframe,palette='rainbow')
```

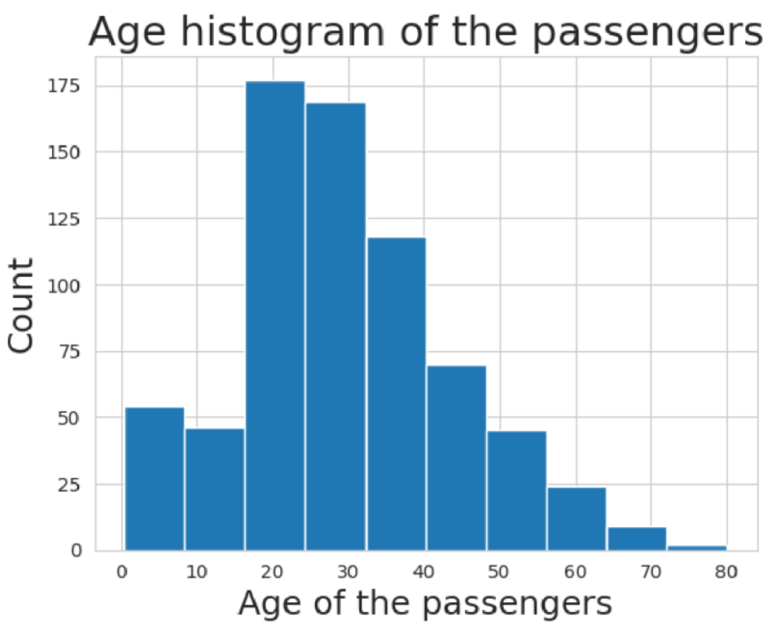
<Axes: xlabel='Survived', ylabel='count'>



How does the overall age distribution look like?

```
plt.xlabel("Age of the passengers",fontsize=18)
plt.ylabel("Count",fontsize=18)
plt.title("Age histogram of the passengers",fontsize=22)
#train['Age'].hist(bins=30,color='darkred',alpha=0.7,figsize=(10,6))
dataframe['Age'].hist()
```

<Axes: title={'center': 'Age histogram of the passengers'}, xlabel='Age of the passengers', ylabel='Count'>

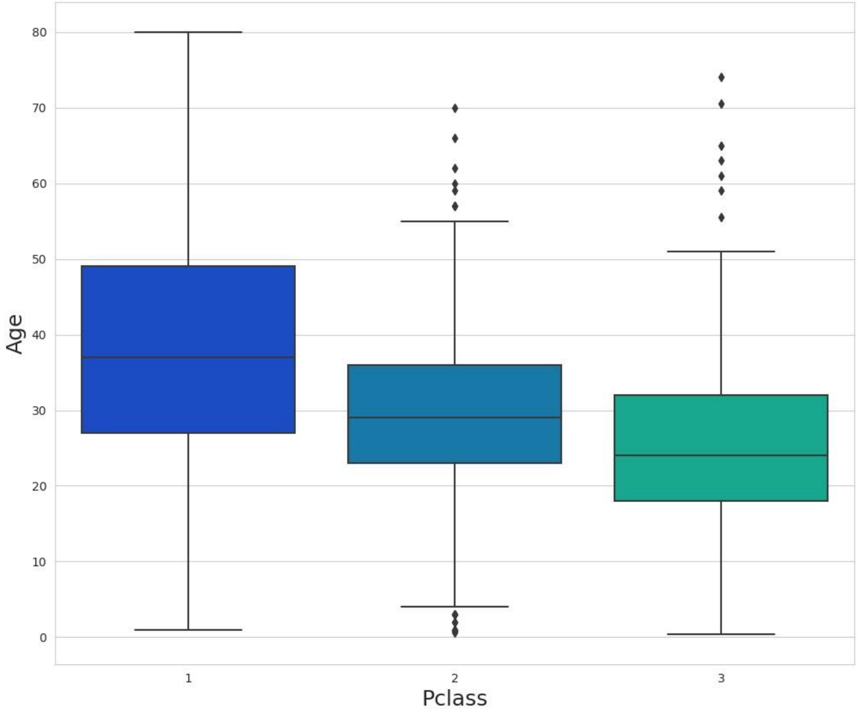


How does the age distribution look like across passenger class?

It looks like that the average age is different for three classes and it generally decreases from 1st class to 3rd class.

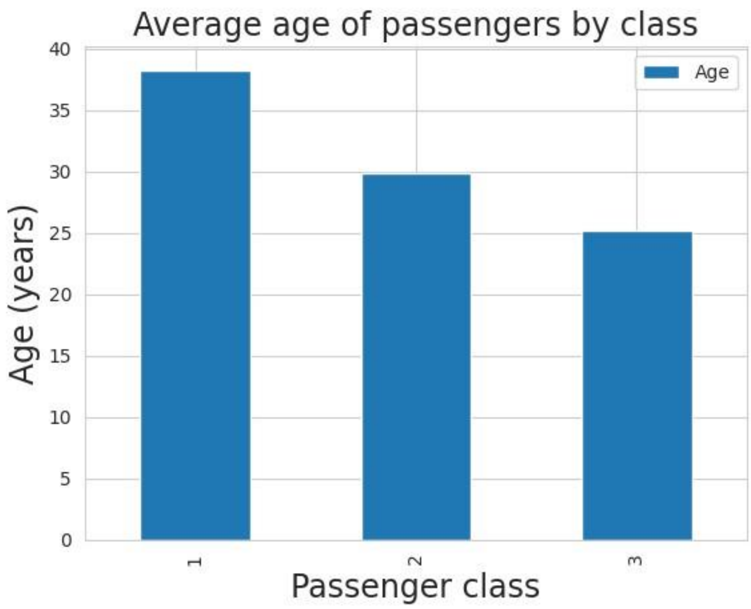
```
plt.figure(figsize=(12, 10))
plt.xlabel("Passenger Class",fontsize=18)
plt.ylabel("Age",fontsize=18)
sns.boxplot(x='Pclass',y='Age',data=dataframe,palette='winter')
```

<Axes: xlabel='Pclass', ylabel='Age'>




```
f_class_Age=dataframe.groupby('Pclass')['Age'].mean()
f_class_Age = pd.DataFrame(f_class_Age)
f_class_Age.plot.bar(y='Age')
plt.title("Average age of passengers by class",fontsize=17)
plt.ylabel("Age (years)", fontsize=17)
plt.xlabel("Passenger class", fontsize=17)
```

```
Text(0.5, 0, 'Passenger class')
```



Data wrangling (impute and drop)

- Impute age (by averaging)
- Drop unncessary features
- Convert categorical features to dummy variables

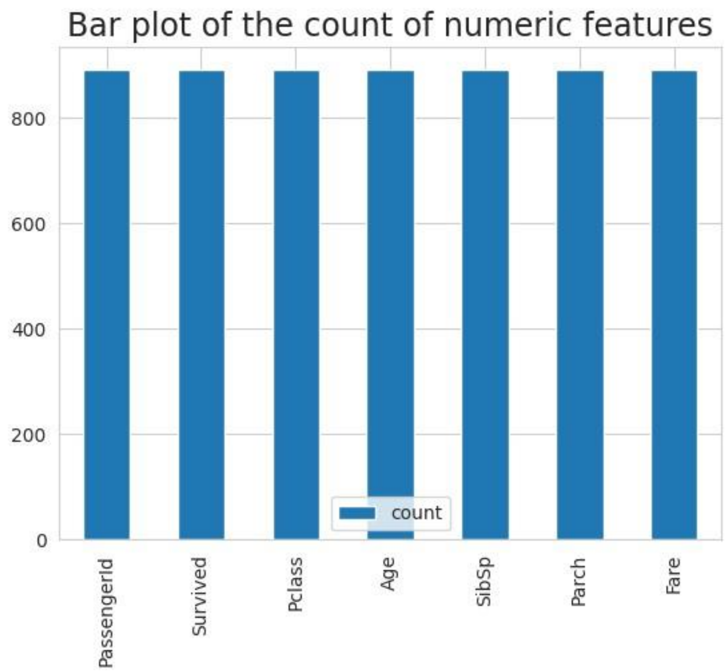
De ne a function to impute (ll-up missing values) age feature

```
a=list(dataframe['Age']);
def impute_age(cols):
Age = cols[0]   Pclass =
cols[1]   if
pd.isnull(Age):     if
Pclass == 1:       return
a[0]     elif Pclass ==
2:       return a[2]
else:
      return a[2]

  else:
return Age
```

Apply the above-de ned function and plot the count of numeric features

```
dataframe['Age'] = dataframe[['Age','Pclass']].apply(impute_age,axis=1)
d=dataframe.describe() dT=d.T dT.plot.bar(y='count') plt.title("Bar
plot of the count of numeric features",fontsize=17)
```

```
Text(0.5, 1.0, 'Bar plot of the count of numeric features')
```



Drop the 'Cabin' feature and any other null value

```
dataframe.drop('Cabin',axis=1,inplace=True)
dataframe.dropna(inplace=True)
dataframe.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2

1	2	1	1	Cumings, Mrs. John Bradley (Florence	female	38.0	1	0	PC 17599	71.2

Drop other unnecessary features like 'PassengerId', 'Name', 'Ticket'

```
dataframe.drop(['PassengerId','Name','Ticket'],axis=1,inplace=True)
dataframe.head()
```

Convert categorial feature like 'Sex' and 'Embarked' to dummy variables

Use pandas 'get_dummies()' function

```
sex = pd.get_dummies(dataframe['Sex'],drop_first=True) embark
= pd.get_dummies(dataframe['Embarked'],drop_first=True)
```

Now drop the 'Sex' and 'Embarked' columns and concatenate the new dummy variables

```
dataframe.drop(['Sex','Embarked'],axis=1,inplace=True)
dataframe = pd.concat([dataframe,sex,embark],axis=1)
dataframe.head()
```

This data set is now ready for logistic regression analysis!

Logistic Regression model t and prediction

Let's start by splitting our data into a training set and test set (there is another test.csv le that you can play around with in case you want to use all this data for training).

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dataframe.drop('Survived',axis=1),
dataframe['Survived'], test_size=0.30,
random_state=111)
```

F1-score as a fucntion of regularization (penalty) parameter

```
from sklearn.linear_model import LogisticRegression
 from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

nsimu = 201 penalty = [0]
* nsimu logmodel = [0] *
nsimu predictions = [0] *
nsimu class_report = [0] *
nsimu f1 = [0] * nsimu

for i in range(1, nsimu + 1):
    logmodel[i] = LogisticRegression(C=i / 1000, tol=1e-4, max_iter=int(1e6), n_jobs=4)
logmodel[i].fit(X_train, y_train)     predictions[i] = logmodel[i].predict(X_test)
    class_report[i] = classification_report(y_test, predictions[i])
l = class_report[i].split()     f1[i] = l[len(l) - 1]
penalty[i] = 1000 / i plt.scatter(penalty[1:len(penalty) - 2],
f1[1:len(f1) - 2]) plt.title("F1-score vs. regularization
parameter", fontsize=20) plt.xlabel("Penalty parameter",
fontsize=17) plt.ylabel("F1-score on test data", fontsize=17)
plt.show()
```

F1-score as a function of test set size (fraction)

```
nsimu=101
class_report = [0]*nsimu
f1=[0]*nsimu
test_fraction =[0]*nsimu
 for i in range(1,nsimu):
        X_train, X_test, y_train, y_test = train_test_split(dataframe.drop('Survived',axis=1),
dataframe['Survived'], test_size=0.1+(i-1)*0
random_state=111)         logmodel =(LogisticRegression(C=1,tol=1e-4, max_iter=1000,n_jobs=4))
logmodel.fit(X_train,y_train)         predictions = logmodel.predict(X_test)
        class_report[i] = classification_report(y_test,predictions)
l=class_report[i].split()         f1[i] = l[len(l)-2]
        test_fraction[i]=0.1+(i-1)*0.007

plt.plot(test_fraction[1:len(test_fraction)-2],f1[1:len(f1)-2])
plt.title("F1-score vs. test set size (fraction)",fontsize=20)
plt.xlabel("Test set size (fraction)",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17) plt.show()
```

F1-score as a function of random seed of test/train split

```
nsimu=101
class_report = [0]*nsimu
f1=[0]*nsimu random_init
=[0]*nsimu for i in
range(1,nsimu):
        X_train, X_test, y_train, y_test = train_test_split(dataframe.drop('Survived',axis=1),
dataframe['Survived'], test_size=0.3,
 random_state=i+100)         logmodel =(LogisticRegression(C=1,tol=1e-5,
max_iter=1000,n_jobs=4))         logmodel.fit(X_train,y_train)         predictions =
logmodel.predict(X_test)
        class_report[i] = classification_report(y_test,predictions)
l=class_report[i].split()         f1[i] = l[len(l)-2]
random_init[i]=i+100

plt.plot(random_init[1:len(random_init)-2],f1[1:len(f1)-2])
plt.title("F1-score vs. random initialization seed",fontsize=20)
plt.xlabel("Random initialization seed",fontsize=17)
plt.ylabel("F1-score on test data",fontsize=17) plt.show()
```

Implement Kernel Density Estimation for Feature Space

Import required libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns from
scipy import stats
```

We take 100 random samples and nd minium and maximum in them.

np.linspace(x_min, x_max, 100): This generates an array called x_axis that contains 100 evenly spaced values starting from x_min and ending at x_max. This is often used to create a range of x-values for plotting

```
dataset = np.random.randn(100)
x_min = dataset.min() - 2
x_max = dataset.max() + 2
x_axis = np.linspace(x_min,x_max,100)

print(x_min,x_max)
```

```
-4.53534782641586 4.623552856673909
```

Calculating the bandwidth for kernel density estimation. The bandwidth determines the smoothness of the estimated probability density function (PDF) when using kernel density estimation. The formula you are using for bandwidth appears to be based on a practical estimation method.

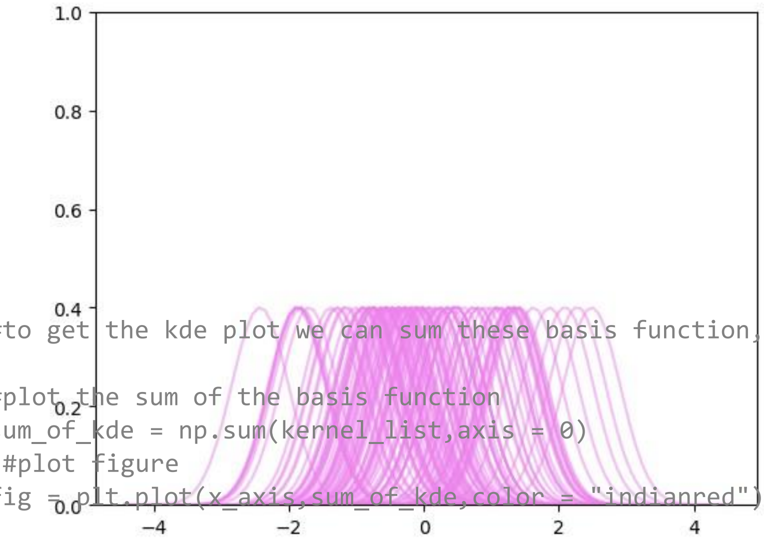
The bandwidth calculated in this way will be used in kernel density estimation to control the width of the kernel, which affects the smoothness of the estimated PDF. The bandwidth can have a signi cant impact on the appearance and accuracy of kernel density plots.

```
#set up the bandwidth, for info on this: url =
'http://en.wikipedia.org/wiki/Kernel\_density\_estimation\#Practical_estimation_of_the_ban
bandwidth = ((4*dataset.std()**-0.5)/(3*len(dataset)))**0.2 bandwidth
```

```
0.42199405201453
```

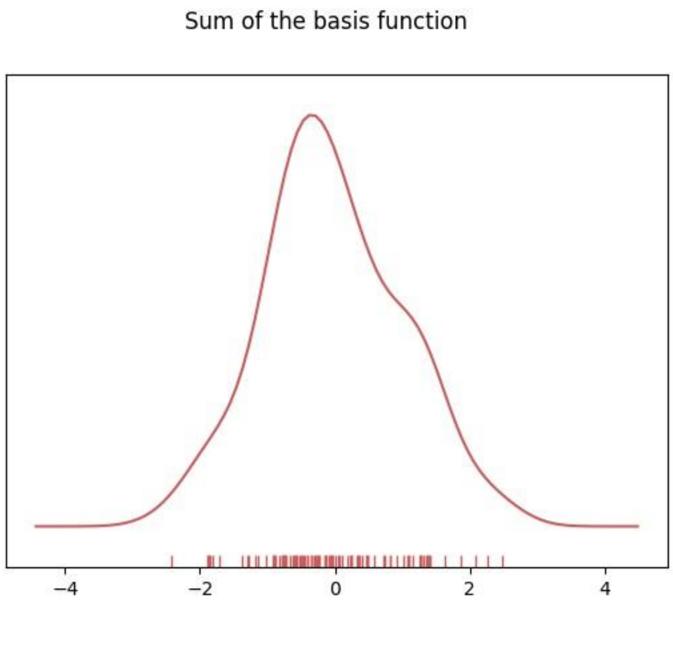
```
#create an empty kernel list
kernel_list = []

#Plot each basis function
for data_point in dataset:

  #create a ketnel for each point and append to list
kernel = stats.norm(data_point,bandwidth).pdf(x_axis)
kernel_list.append(kernel)

  #scale for plotting   kernel
= kernel / kernel.max()
kernel = kernel * .4
  plt.plot(x_axis,kernel,color = 'violet',alpha = 0.5)

plt.ylim(0,1)
```

```
(0.0, 1.0)
```




```
#to get the kde plot we can sum these basis function.

#plot the sum of the basis function
sum_of_kde = np.sum(kernel_list,axis = 0)
 #plot figure
fig = plt.plot(x_axis,sum_of_kde,color = "indianred")
                                                    #add the initial
rugplot
sns.rugplot(dataset,c = "indianred")
 #get rid of y-tick marks
plt.yticks([])

#set title plt.suptitle("Sum of the
basis function")
```

```
Text(0.5, 0.98, 'Sum of the basis function')
```



Implement Dimensionality Reduction using Principal Component Analysis (PCA)

Get required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns from sklearn
import preprocessing from
sklearn.decomposition import PCA
```

Rename the columns into desired names.

```
a = pd.read_csv("iris.data",names=['sepal length','sepal width','petal length','petal width','ta
print(a)
a.shape
```

```
       sepal length  sepal width  petal length  petal width          target
    0           5.1          3.5           1.4          0.2     Iris-setosa
    1           4.9          3.0           1.4          0.2     Iris-setosa
    2           4.7          3.2           1.3          0.2     Iris-setosa
    3           4.6          3.1           1.5          0.2     Iris-setosa
    4           5.0          3.6           1.4          0.2     Iris-setosa..          ...          ...          ...          ...
                ...
    145         6.7          3.0           5.2          2.3  Iris-virginica
    146         6.3          2.5           5.0          1.9  Iris-virginica
    147         6.5          3.0           5.2          2.0  Iris-virginica
    148         6.2          3.4           5.4          2.3  Iris-virginica
    149         5.9          3.0           5.1          1.8  Iris-virginica

    150         rows x 5 columns] (150, 5)
```

```
features = a.columns
features
```

```
Index(['sepal length', 'sepal width', 'petal length', 'petal width', 'target'], dtype='object')
```

```
from sklearn.preprocessing import StandardScaler
features = ['sepal length','sepal width','petal length','petal width']
x = a.loc[:,features].values y = a.loc[:,['target']].values x =
StandardScaler().fit_transform(x)

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDataframe = pd.DataFrame(data = principalComponents,columns = ['PC1','PC2'])

targetDataframe = a[['target']]
newDataframe = pd.concat([principalDataframe, targetDataframe], axis = 1)

newDataframe
```

	PC1	PC2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa

Creating a scatter plot of data points in a two-dimensional space de ned by the Principal Component 1 (PC1) and Principal Component 2 (PC2)**3** -2.304197 - 0.575368 Iris-setosa values. The plot is intended to visualize the relationship or distribution of data points in this reduced-dimensional space.**4** -2.388777 0.674767 Iris-setosa

You can see how data points are distributed in the PC1-PC2 space. It helps you understand the concentration of data points and whether they... ...

... ... form clusters or exhibit patterns.

145 1.870522 0.382822 Iris-virginica

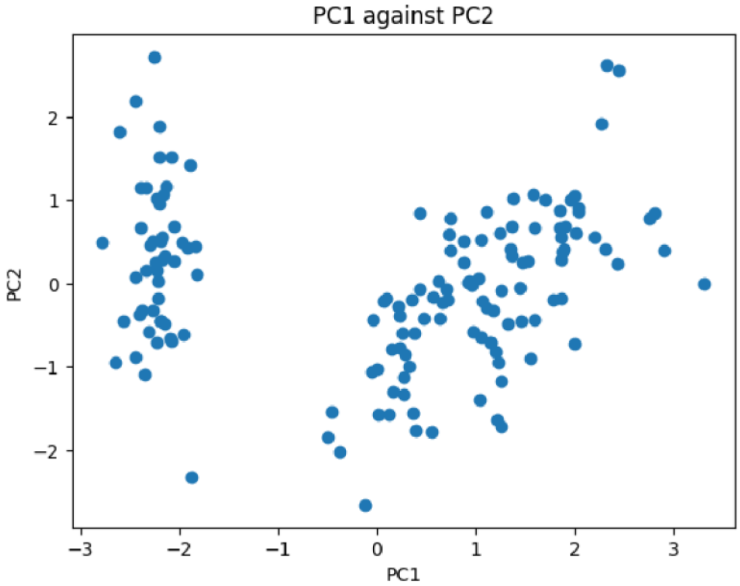
The scatter plot can show whether there is a correlation or relationship between PC1 and PC2. If the points tend to follow a linear trend or form**146** 1.558492 -0.905314 Iris-virginica a distinct shape, it indicates a correlation between the two components. **147** 1.520845 0.266795 Iris-virginica

Outliers, if present, may be visible as data points that deviate signi cantly from the general trend. These outliers can provide valuable
148 1.376391 1.016362 Iris-virginica
information about data anomalies.
149 0.959299 -0.022284 Iris-virginica

150 rows × 3 columns

```
plt.scatter(principalDataframe.PC1,principalDataframe.PC2)
plt.title("PC1 against PC2") plt.xlabel("PC1")
plt.ylabel("PC2")
```

```
Text(0, 0.5, 'PC2')
```




```
A scatter plot for data points in a two-dimensional space defined by Principal
Component 1 (PC1) and Principal Component 2 (PC2). This plot appears to be
specifically designed for visualizing the Iris dataset, which contains samples
of three different species: Iris-setosa, Iris-versicolor, and Iris-virginica.

You will see a scatter plot with data points from the Iris dataset displayed in
two dimensions (PC1 and PC2).

Data points belonging to different Iris species (setosa, versicolor, and
virginica) will be distinguished by different colors (red, green, and blue).

The plot visually shows how the Iris species are distributed in the PC1-PC2
space. It can help you observe patterns, clusters, or separations between these
species.

The legend will identify which color corresponds to each Iris species, making
it easy to interpret the plot.
```

A scatter plot for data points in a two-dimensional space de ned by Principal Component 1 (PC1) and Principal Component 2 (PC2). This plot appears to be speci cally designed for visualizing the Iris dataset, which contains samples of three different species: Iris-setosa, Iris-versicolor, and Iris-virginica.

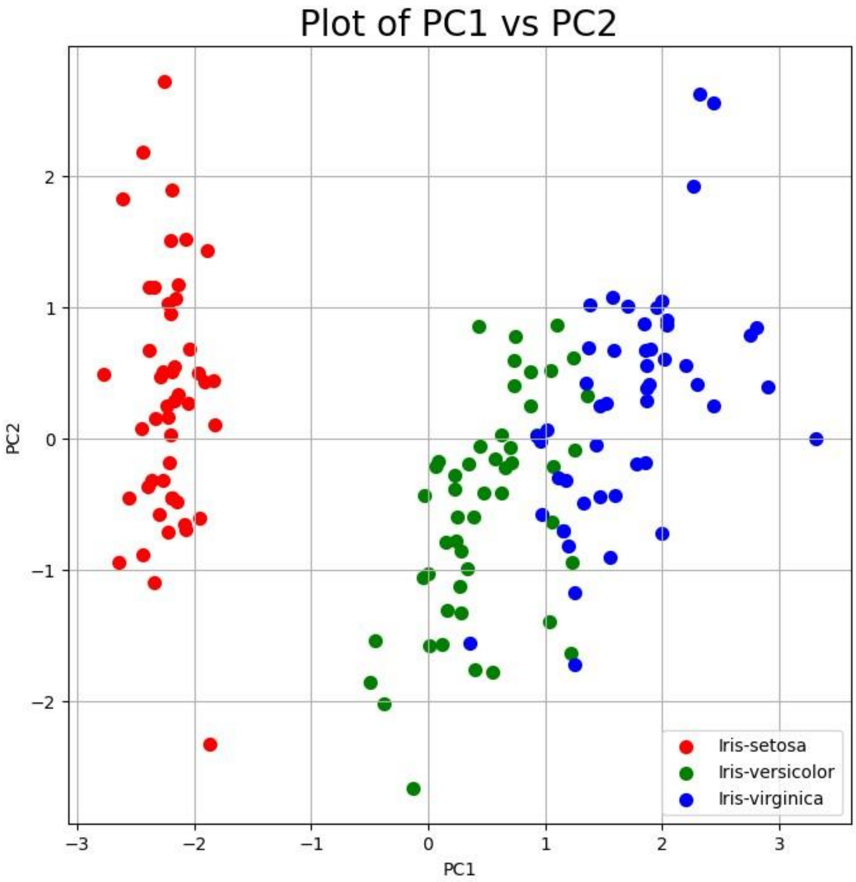
You will see a scatter plot with data points from the Iris dataset displayed in two dimensions (PC1 and PC2).

Data points belonging to different Iris species (setosa, versicolor, and virginica) will be distinguished by different colors (red, green, and blue).

The plot visually shows how the Iris species are distributed in the PC1-PC2 space. It can help you observe patterns, clusters, or separations between these species.

The legend will identify which color corresponds to each Iris species, making it easy to interpret the plot.

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_title("Plot of PC1 vs PC2",fontsize = 20) targets =
['Iris-setosa','Iris-versicolor','Iris-virginica'] colors =
['r','g','b'] for target,color in zip(targets,colors):
  indicesToKeep = newDataframe['target'] == target
  ax.scatter(newDataframe.loc[indicesToKeep,'PC1'],newDataframe.loc[indicesToKeep,'PC2'],c = col
ax.legend(targets)   ax.grid()
```




```
explained_variance_ratio = pca.explained_variance_ratio_
explained_variance_ratio
```

```
array([0.72770452, 0.23030523])
```

Implement K-Means Clustering using Synthetic Data from

Problem:

You have a multidimensional set of data (such as a set of hidden unit activations) and you want to see which points are closest to others. The k-means algorithm searches for a pre-determined number of clusters within an unlabeled multidimensional dataset. It accomplishes this using a simple conception of what the optimal clustering looks like:

• The "cluster center" is the arithmetic mean of all the points belonging to the cluster.

• Each point is closer to its own cluster center than to other cluster centers.

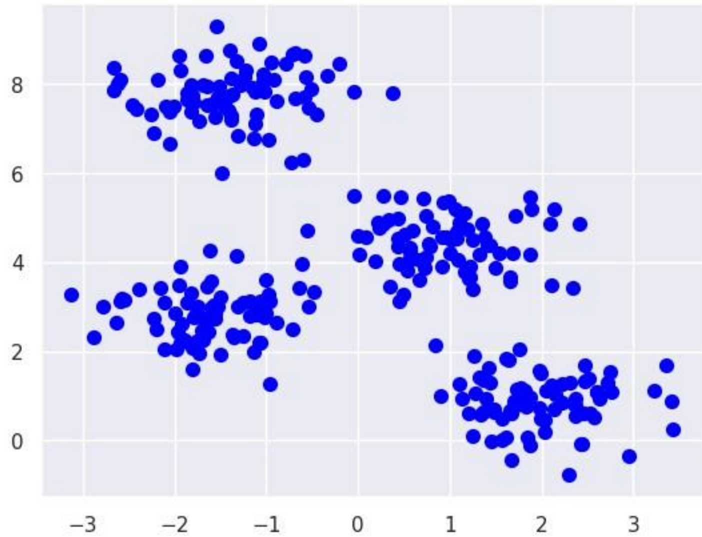
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()#plot styling import
numpy as np
```

Create synthetic dataset of unlabeled blobs

The dataset would be synthesized using sklearn.datasets.samples generator from the sklearn package. You will import binary large objectsblobs to form clusters from the synthetic dataset.

```
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300,centers = 4, cluster_std = 0.60, random_state = 0)
plt.scatter(X[:,0],X[:,1],s=50,color = 'blue')
```

```
<matplotlib.collections.PathCollection at 0x7f97c2206bf0>
```



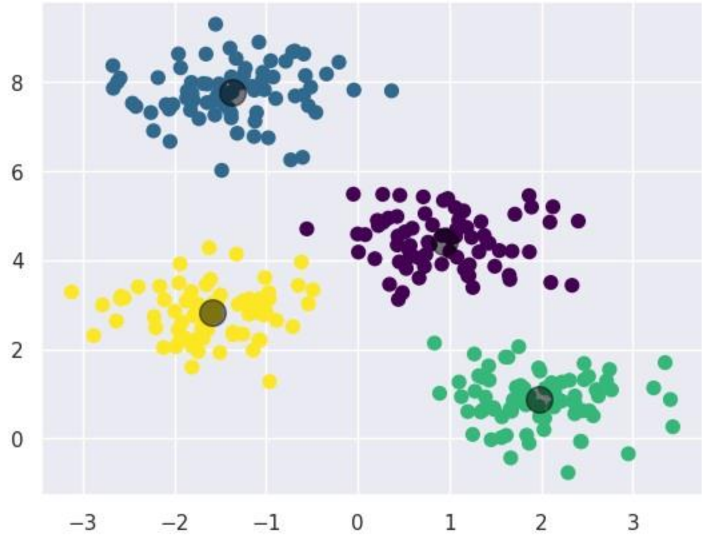
Import K-means from Sklearn and Fit the data

Verify the syntentic dataset and t the data to the K-Means model.

```
from sklearn.cluster import KMeans kmeans =
KMeans(n_clusters = 4, n_init = 10)
kmeans.fit(X) y_kmeans = kmeans.predict(X)
```

Visualize the tted data by coloring the blobs aby assigned label numbers Verify the syntentic dataset and t the data to the K-Means model.

```
plt.scatter(X[:,0],X[:,1],c = y_kmeans,s = 50, cmap = 'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='black',s = 200, alpha = .5);
```



How k-means is a special case of Expectation-maximization (EM) algorithm Expectation–maximization (EM) is a powerful algorithm that comes up in a variety of con- texts within data science. k-means is a particularly simple and special case of this more general algorithm. The basic algorithmic ow of k-means is to

• Guess some cluster center (initialization)

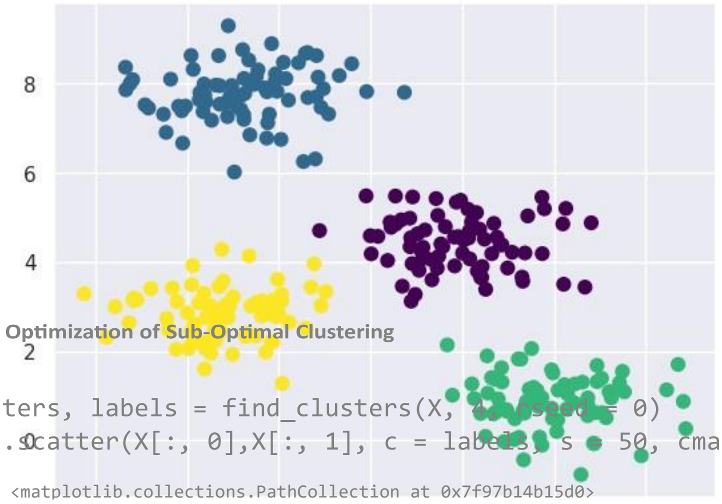
• Repeat following steps untill converged

E-step: assign points to the nearest cluster center

• M-Step: set the cluster centers to the mean

```
from sklearn.metrics import pairwise_distances_argmin

def find_clusters(X, n_clusters, rseed = 2):

  #1. Randomly choosed clusters   rng =
np.random.RandomState(rseed)   i =
rng.permutation(X.shape[0])[:n_clusters]
centers = X[i]

  while True:
    #2a. Assign labels based on closest center
labels = pairwise_distances_argmin(X, centers)

    #2b. Find new centers from means of points
new_centers = np.array([X[labels == i].mean(0)

    for i in range(n_clusters)])
#2c. Check for convergence     if
np.all(centers == new_centers):
      break
    centers = new_centers
return centers, labels

centers, labels = find_clusters(X , 4)
plt.scatter(X[:,0], X[:,1], c = labels, s = 50, cmap = "viridis")
```

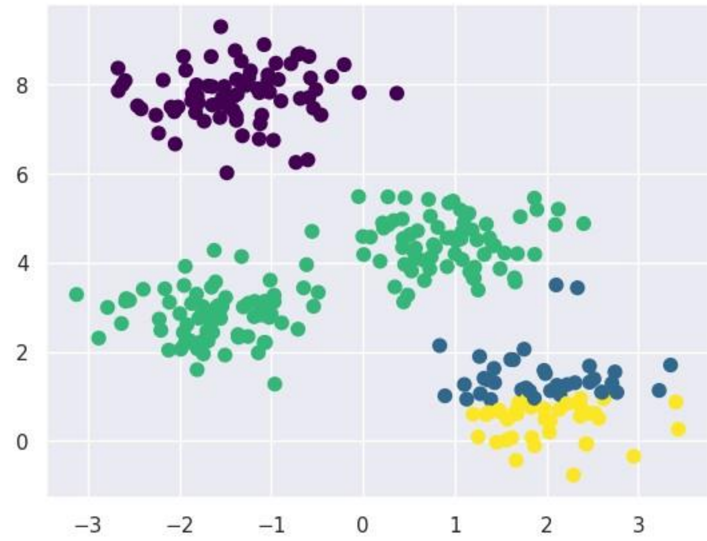
```
<matplotlib.collections.PathCollection at 0x7f97b1465510>
```



Bad Optimization of Sub-Optimal Clustering

```
centers, labels = find_clusters(X, 4, rseed = 0)
plt.scatter(X[:, 0],X[:, 1], c = labels, s = 50, cmap = 'viridis')
```

<matplotlib.collections.PathCollection at 0x7f97b14b15d0>

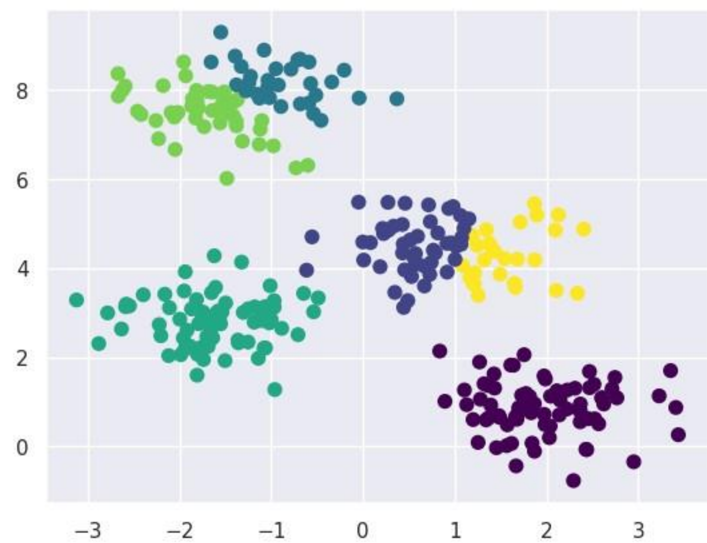


How many number of clusters

Plot the clusters formed using scatter plot

```
labels = KMeans(6, random_state = 0, n_init = 10).fit_predict(X)
plt.scatter(X[:, 0],X[:, 1], c = labels, s = 50, cmap = 'viridis')
```

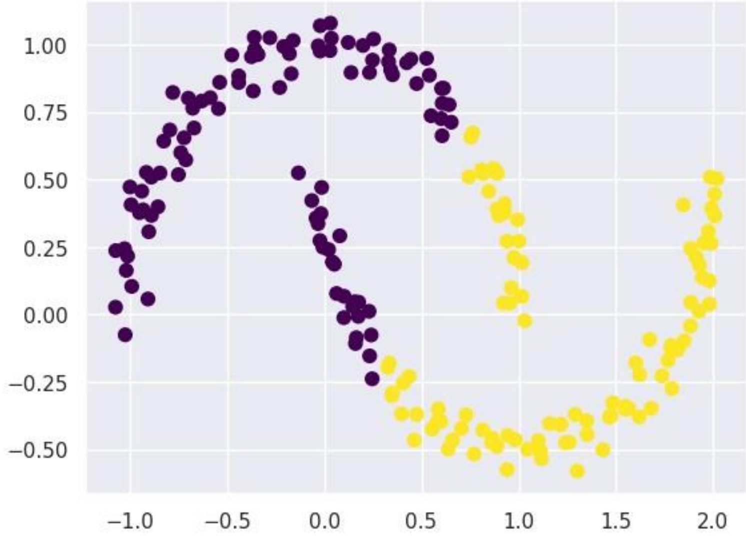
<matplotlib.collections.PathCollection at 0x7f97b0ed5690>



Limitation of K-Means Algorithm

```
from sklearn.datasets import make moons
from sklearn.datasets import make_moons
X, y = make_moons(200, noise = .05, random_state = 0)

labels = KMeans(2, random_state = 0, n_init = 10,).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1],c = labels, s = 50, cmap = 'viridis')
```

```
<matplotlib.collections.PathCollection at 0x7f97b0f46500>
```



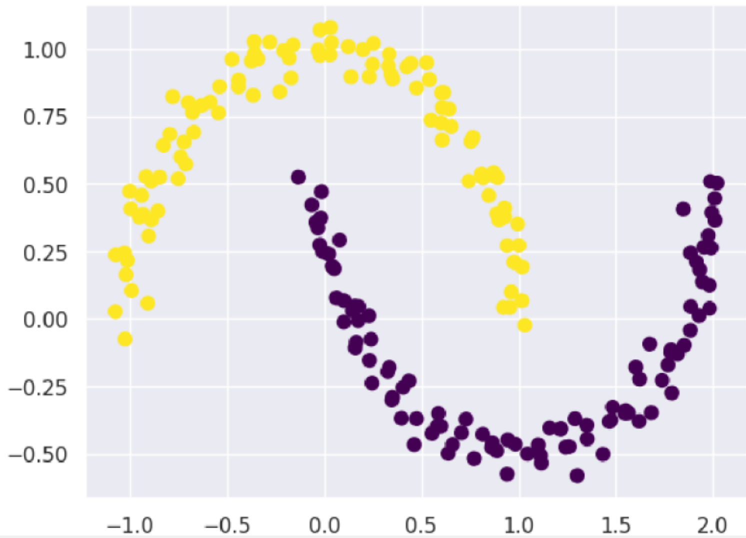
Kernel Transformation

The situation above is reminiscent of the Support Vector Machines, where we use a kernel transformation to project the data into a higher dimension where a linear separation is possible. We might imagine using the same trick to allow k-means to discover non-linear boundaries.

One version of this kernelized k-means is implemented in Scikit-Learn within the SpectralClustering estimator. It uses the graph of nearest neighbors to compute a higher- dimensional representation of the data, and then assigns labels using a k-means algorithm using the following code

```
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters = 2, affinity = 'nearest_neighbors', assign_labels = 'kmea
labels = model.fit_predict(X) plt
.scatter(X[:, 0], X[:, 1], c = labels, s = 50, cmap = 'viridis')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:274:
  warnings.warn(
<matplotlib.collections.PathCollection at 0x7f97b0fe8be0>
```



Implement Gaussian Mixture Model using Synthetic Dataset

Challenges in K-Means can be overcomed using:

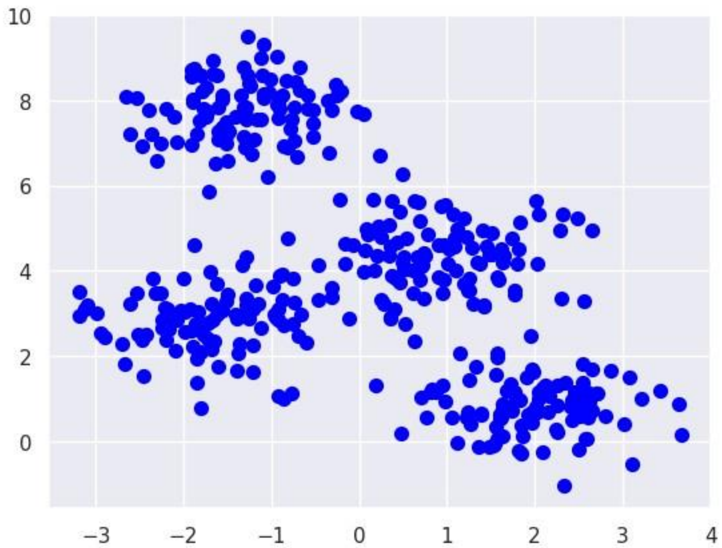
- You could measure uncertainty in cluster assignment by comparing the distances of each point to all cluster centers, rather than focusing onjust the closest.
- You might also imagine allowing the cluster boundaries to be ellipses rather than circles, so as to account for non-circular clusters.

```
import matplotlib.pyplot as plt
import seaborn as sns sns.set()
#plot styling import numpy as
np
```

Generate Synthetic Data using unlabeled blobs

```
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4,
cluster_std=0.7, random_state=0)
plt.scatter(X[:, 0],X[:, 1], s=50, color = 'blue')
```

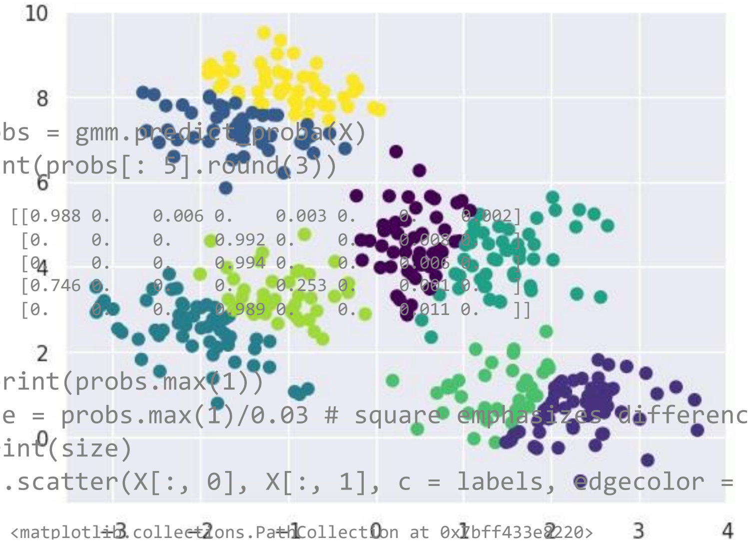
```
<matplotlib.collections.PathCollection at 0x7bff50fbac80>
```



Generalize to Gaussian Mixture Models

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=8).fit(X)
labels = gmm.predict(X)
plt.scatter(X[:, 0],X[:, 1], c=labels, s = 40, cmap = 'viridis')
```

```
<matplotlib.collections.PathCollection at 0x7bff48212f20>
```

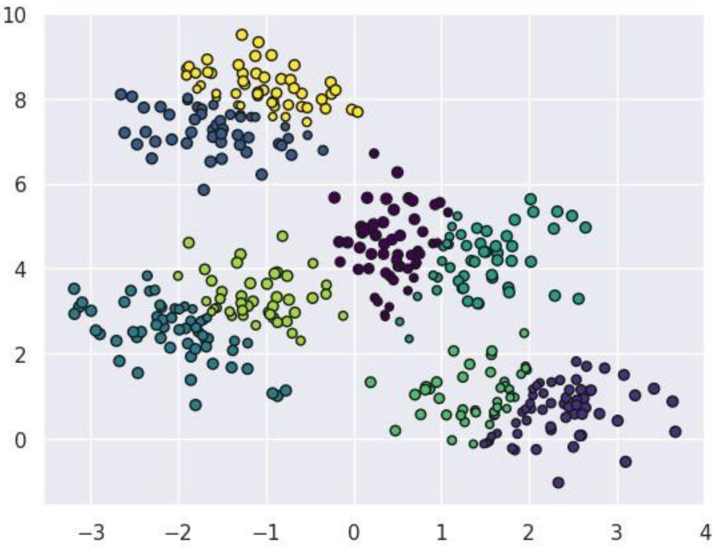


```
probs = gmm.predict_proba(X)
print(probs[: 5].round(3))
```

```
[[0.988 0.    0.006 0.    0.003 0.    0.    0.002]
 [0.    0.    0.    0.992 0.    0.    0.008 0.   ]
 [0.    0.    0.    0.994 0.    0.    0.006 0.   ]
 [0.746 0.    0.    0.    0.253 0.    0.001 0.   ]
 [0.    0.    0.    0.989 0.    0.    0.011 0.   ]]
```

```
# print(probs.max(1))
size = probs.max(1)/0.03 # square emphasizes differences
#print(size)
plt.scatter(X[:, 0], X[:, 1], c = labels, edgecolor = 'k',cmap = 'viridis', s=size)
```

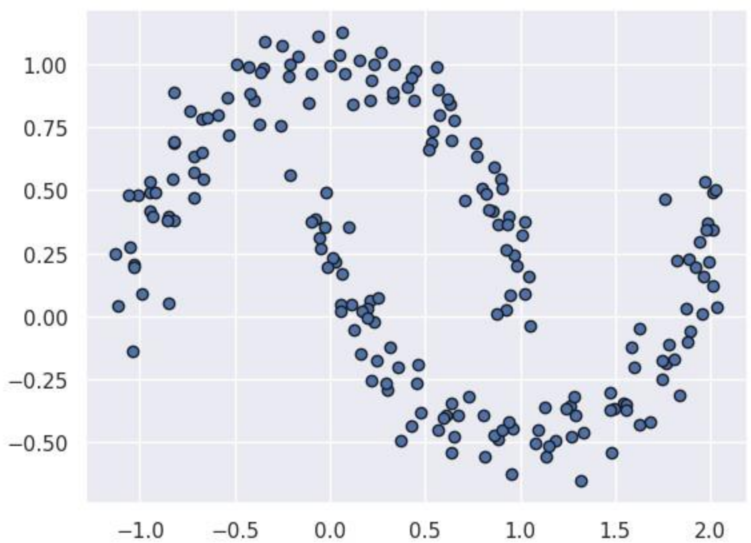
```
<matplotlib.collections.PathCollection at 0x7bff433e2220>
```



GMM as Density Estimation and Generative Model

```
from sklearn.datasets import make_moons
Xmoon, Ymoon = make_moons(200, noise = 0.08, random_state = 0)
plt.scatter(Xmoon[:, 0], Xmoon[:, 1], edgecolor = 'k')
```

```
<matplotlib.collections.PathCollection at 0x7bff434a2020>
```



The function rst ts the GMM to the data using the t method and then predicts cluster labels for each data point using the predict method. If label is True, it colors the data points based on their cluster labels using the 'viridis' colormap. It also plots ellipses representing the GMM's components using the draw_ellipse function, with the ellipses' properties being determined by the GMM's means, covariances, and weights.

The variable w_factor is used to adjust the transparency of the ellipses based on the weights of the GMM components. This makes the ellipses more transparent for components with lower weights.

The expected outcome when using these functions is a Matplotlib scatter plot where data points are colored according to their cluster assignments if ' label=True '. In addition, ellipses will be drawn to represent the shape, orientation, and size of each Gaussian component in the GMM. This allows you to visually understand the clustering and characteristics of the data based on the GMM model.

```
from matplotlib.patches import Ellipse def
draw_ellipse(position, covariance, ax=None, **kwargs):
"""Draw an ellipse with a given position and covariance"""
ax = ax or plt.gca()

    # Convert covariance to principal axes   if
covariance.shape == (2, 2):     U, s, Vt =
np.linalg.svd(covariance)     angle =
np.degrees(np.arctan2(U[1, 0], U[0, 0]))     width,
height = 2 * np.sqrt(s)   else:
    angle = 0
    width, height = 2 * np.sqrt(covariance)

  # Draw the Ellipse   for
nsig in range(1, 4):
    ax.add_patch(Ellipse(position, nsig * width, nsig * height,
angle, **kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
  ax = ax or plt.gca()   labels
= gmm.fit(X).predict(X)   if
label:
    ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis',
zorder=2,edgecolor='k')   else:
    ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2,cmap='viridis',edgecolor='k')
ax.axis('equal')

  w_factor = 0.2 / gmm.weights_.max()   for pos, covar, w in
zip(gmm.means_, gmm.covariances_, gmm.weights_):     draw_ellipse(pos,
covar, alpha=w * w_factor)
```

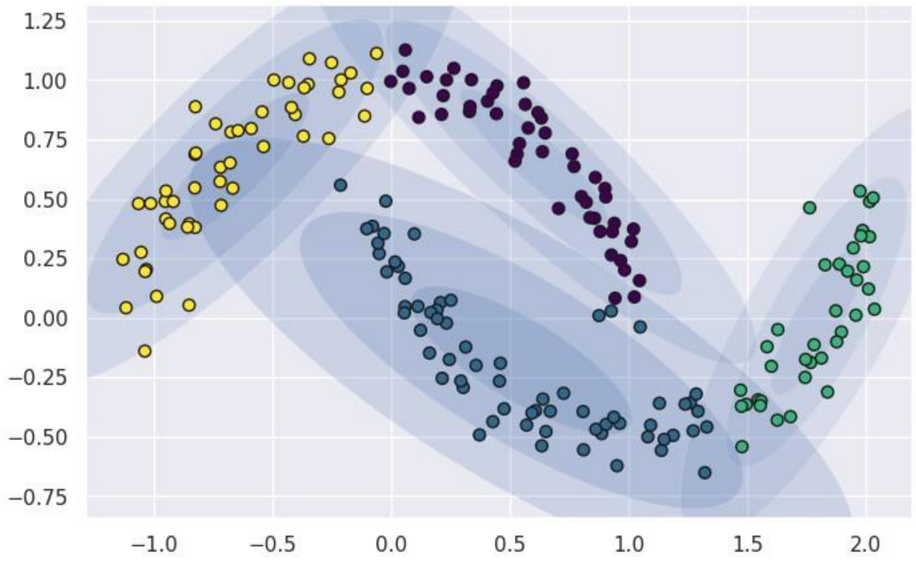
The outcome of this code will be a Matplotlib plot where:

Data points from Xmoon will be scattered on the plot, and each point will be colored according to its cluster assignment.

Ellipses will be drawn around the clusters to represent the estimated shapes, orientations, and sizes of the GMM components.

```
gmm2 = GaussianMixture(n_components=4, covariance_type='full',
random_state=42) plt.figure(figsize = (8,5)) plot_gmm(gmm2,
Xmoon)
```

```
<ipython-input-71-18cf7d486bb7>:17: MatplotlibDeprecationWarning: Passing the angle parameter of __init__() positionally is deprecat
ax.add_patch(Ellipse(position, nsig * width, nsig * height,
```


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Implement Support Vector Machine Classi cation using Breast Cancer Dataset

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classi cation and regression analysis.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. This gap is also called maximum margin and the SVM classi er is called maximum margin clasi er.

In addition to performing linear classi cation, SVMs can e ciently perform a non-linear classi cation using what is called the kernel trick, implicitly mapping their inputs into high- dimensional feature spaces.

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

Get the data

We'll use the built in breast cancer dataset from Sclkit learn. Note the load functionn:

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

The dataset is presented in a dictionary format

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
cancer[
    'feature_names'
]
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Set up dataframe

```
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
0    mean radius              569 non-null    float64
1    mean texture             569 non-null    float64
2    mean perimeter           569 non-null    float64
3    mean area                569 non-null    float64
4    mean smoothness          569 non-null    float64
5    mean compactness         569 non-null    float64
6    mean concavity           569 non-null    float64
7    mean concave points      569 non-null    float64
8    mean symmetry            569 non-null    float64
9    mean fractal dimension   569 non-null    float64
10   radius error             569 non-null    float64
11   texture error            569 non-null    float64
12   perimeter error          569 non-null    float64
```

```
 13   area error               569 non-null    float64
 14   smoothness error         569 non-null    float64
 15   compactness error        569 non-null    float64
 16   concavity error          569 non-null    float64
 17   concave points error     569 non-null    float64
 18   symmetry error           569 non-null    float64
 19   fractal dimension error  569 non-null    float64
 20   worst radius             569 non-null    float64
 21   worst texture            569 non-null    float64
 22   worst perimeter          569 non-null    float64
 23   worst area               569 non-null    float64
 24   worst smoothness         569 non-null    float64
 25   worst compactness        569 non-null    float64
 26   worst concavity          569 non-null    float64
 27   worst concave points     569 non-null    float64
 28   worst symmetry           569 non-null    float64 29  worst fractal dimension  569 non-null    float64 dtypes: float64(30)
memory usage: 133.5 KB
```

```
df.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	1
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	1
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	1
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	1
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	3

8 rows × 30 columns

```
np.sum(pd.isnull(df).sum()) #Sum of the count of the null objects in all
```

```
0
```

What are the 'target' data in the dataset?

```
cancer['target'].sum()
```

```
357
```

Adding the target data to DataFrame

```
df['Cancer'] = pd.DataFrame(cancer['target'])
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575

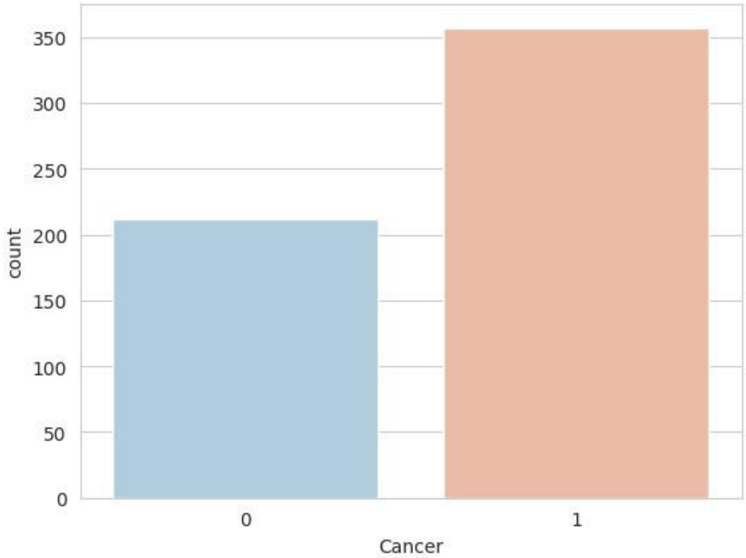
5 rows × 31 columns

Exploratory Data analysis

**Check the relative counts of benign (0) vs malignant (1) cases of*

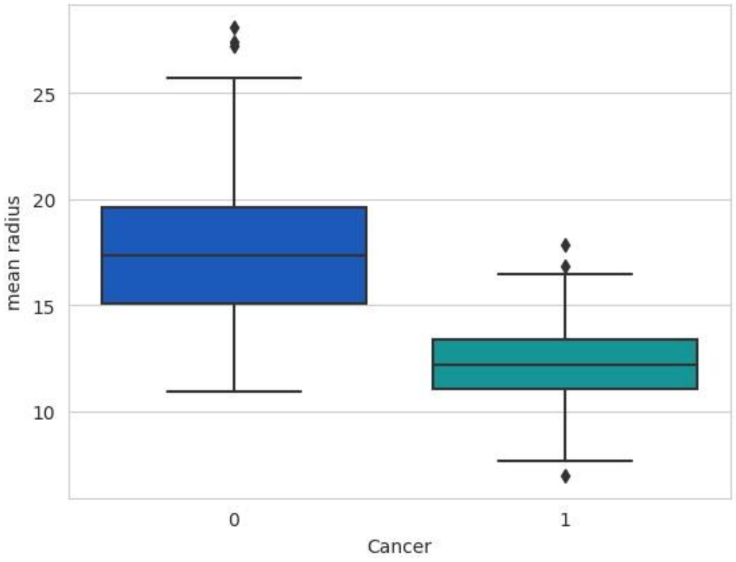
```
sns.set_style('whitegrid')
sns.countplot(x='Cancer', data=df, palette='RdBu_r')
```

<Axes: xlabel='Cancer', ylabel='count'>

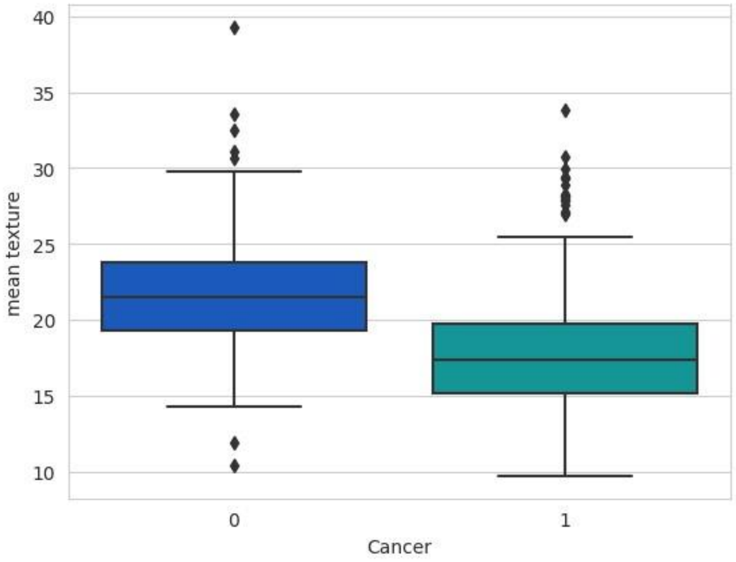


Draw boxplots of all the mean features(rst 10 columns) for '0' and '1' CANCER OF

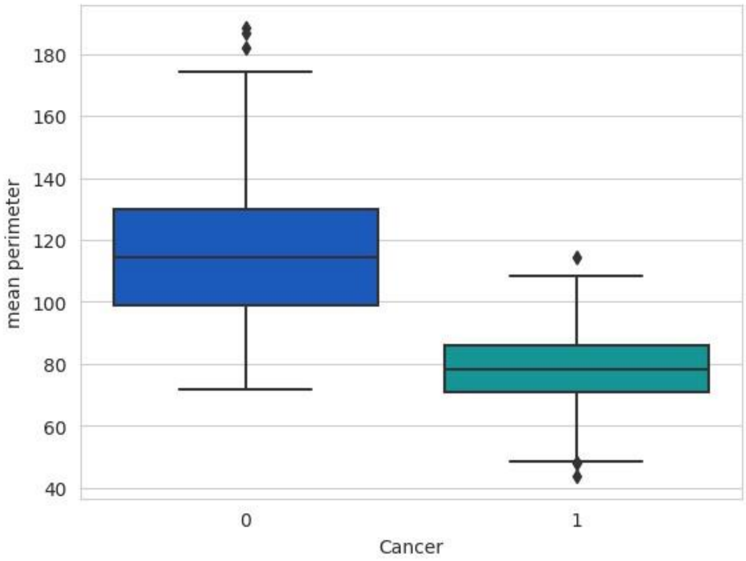
```
l = list(df.columns[0:10])
for i in range(len(l)-1):
  sns.boxplot(x='Cancer', y=l[i], data=df, palette='winter')
plt.figure()
```



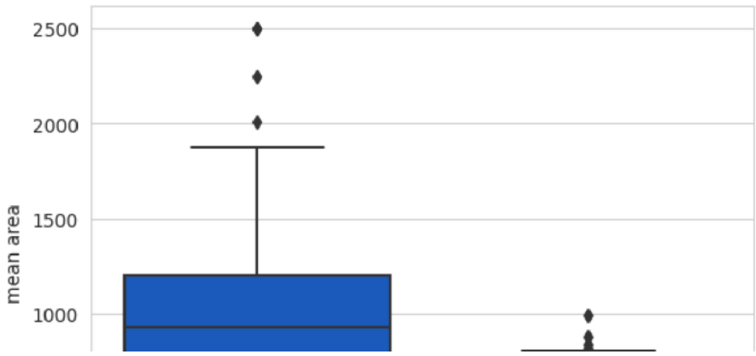
25
20
15
10
mean radius
0
1
Cancer



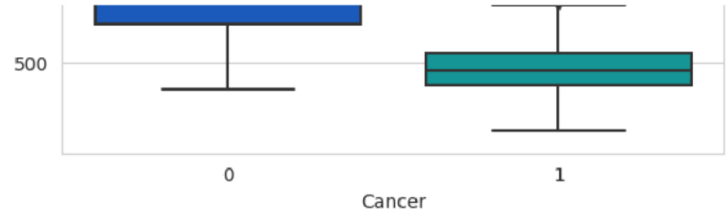
40
35
30
25
20
15
10
mean texture
0
1
Cancer



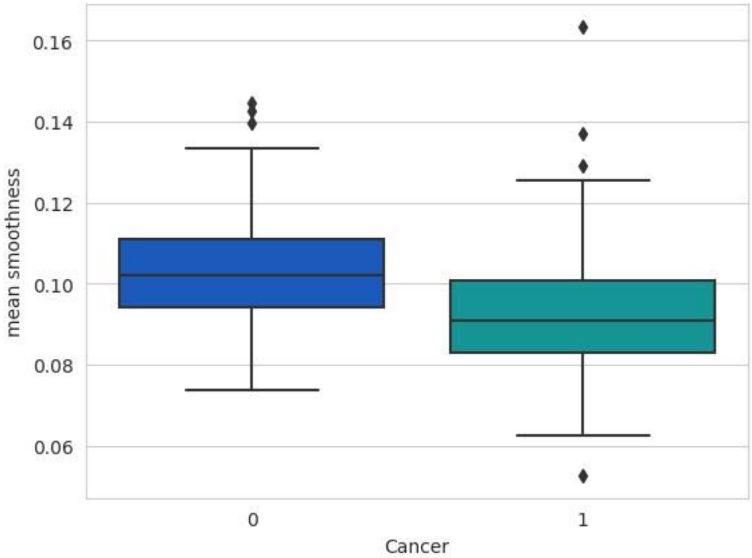
180
160
140
120
100
80
60
40
mean perimeter
0
1
Cancer



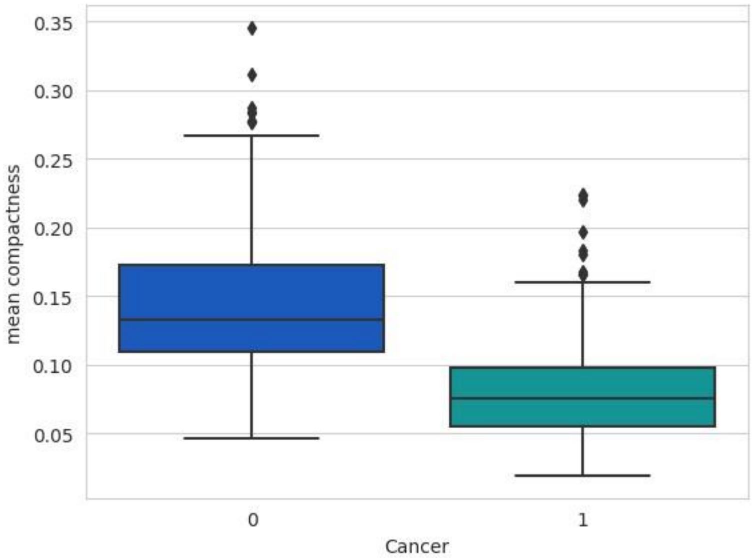
2500
2000
1500
1000
mean area



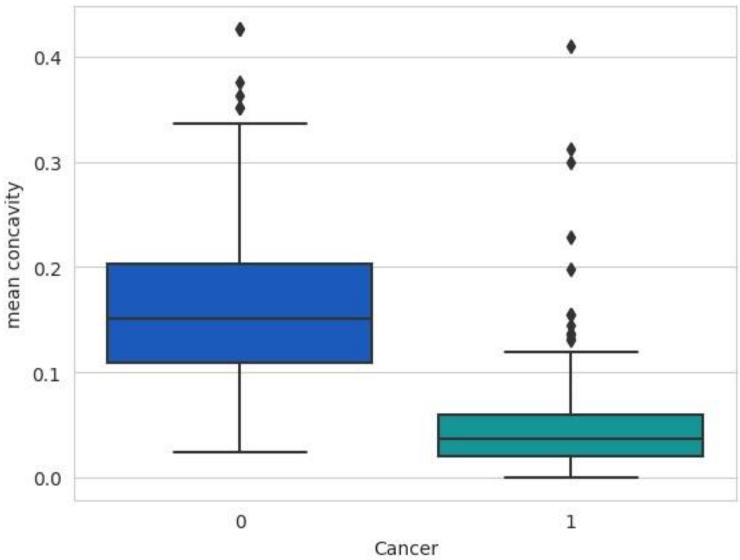
500
0
1
Cancer



mean smoothness
0.16
0.14
0.12
0.10
0.08
0.06
0
1
Cancer



mean compactness
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
1
Cancer

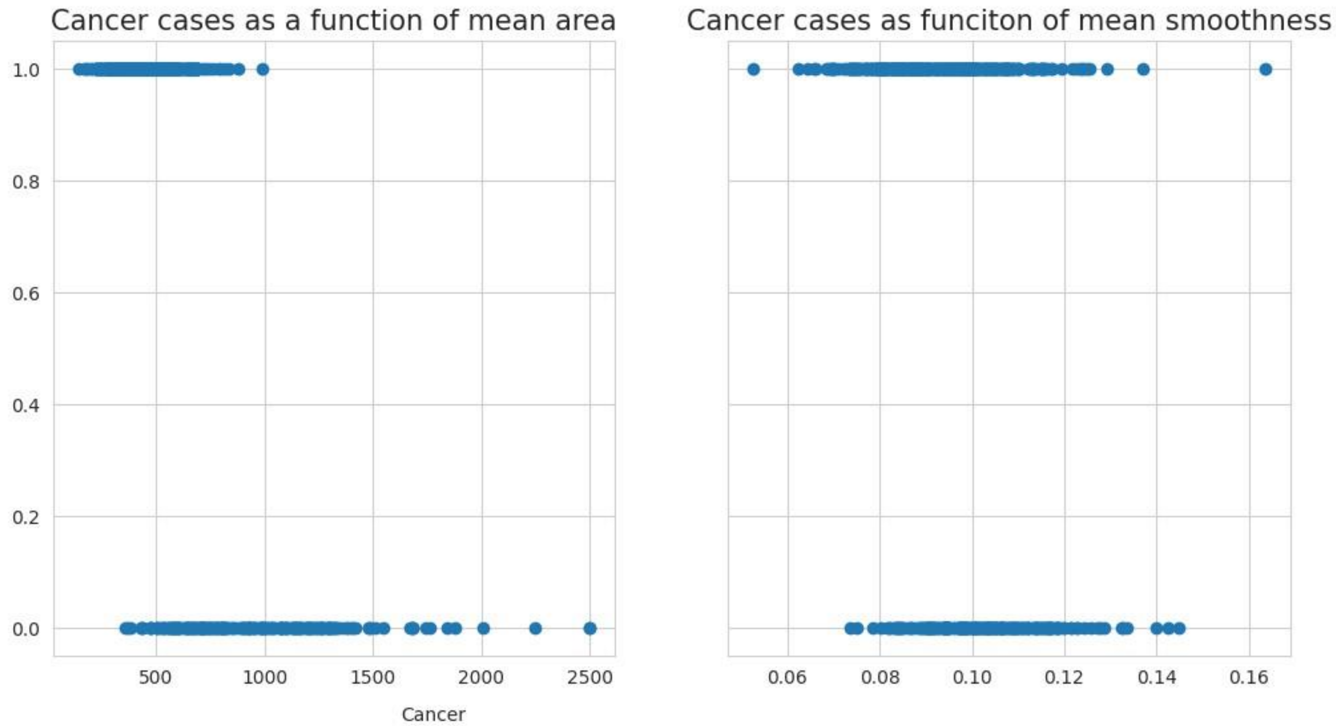


mean concavity
0.4
0.3
0.2
0.1
0.0
0
1
Cancer

0.200
0.175

```
f,(ax1,ax2) = plt.subplots(1,2, sharey=True, figsize=(12,6))
ax1.scatter(df['mean area'],df['Cancer'])
ax1.set_title("Cancer cases as a function of mean area", fontsize=15)

ax2.scatter(df['mean smoothness'], df['Cancer'])
ax2.set_title("Cancer cases as funciton of mean smoothness", fontsize=15)
```

```
Text(0.5, 1.0, 'Cancer cases as funciton of mean smoothness')
```




```
<Figure size 640x480 with 0 Axes>
```

Training and Prediction

Train Test Split

```
df_feat = df.drop('Cancer', axis=1)
df_feat.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	wor	worst texture	worst perimet
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152

5 rows × 30 columns

```
df_target = df['Cancer']
df_target.head()
```

```
0    0
1    0
2    0
3    0
4    0 Name: Cancer, dtype: int64
```

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

```
X_train, X_test, Y_train, Y_test = train_test_split(df_feat, df_target,test_size = .30,
                                                      random_state=101)
Y_train.head()
```

```
178    1
421    1
57     0
514    0
548    1
Name: Cancer, dtype: int64
```

Train the Support Vector Classi er

```
from sklearn.svm import SVC

model = SVC()

model.fit(X_train, Y_train)
```

```
▾SVC
SVC()
```

Predictions and Evaluations

```
predictions = model.predict(X_test) from sklearn.metrics import

classification_report, confusion_matrix
```

Notice that we are classifying everything into a single class! This means our model need to normalize the data

```
print(confusion_matrix(Y_test, predictions))
```

```
[[ 56  10]
 [  3 102]]
```

As expected the classi cation report card is bad

```
print(classification_report(Y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.85      0.90        66
1       0.91      0.97      0.94       105

    accuracy                           0.92       171
macro avg       0.93      0.91      0.92       171
weighted avg       0.93      0.92      0.92       171
```

```
param_grid = {
    'C': [0.1,1,10, 100, 1000], 'gamma': [1,0.1, 0.01, 0.001,0.0001], 'kernel': ['rbf']
} from sklearn.model_selection import GridSearchCV grid =

GridSearchCV(SVC(), param_grid, refit=True, verbose=1)

#May take awhile
grid.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```

```
▸ GridSearchCV
▸ estimator: SVC
▸ SVC
```

```
grid.best_params_
```

```
{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
grid.best_estimator_
```

```
▾         SVC
SVC(C=1, gamma=0.0001)
```

```
grid_predictions = grid.predict(X_test)
```

```
print(confusion_matrix(Y_test, grid_predictions))
```

```
[[ 59   7]
 [  4 101]]
```

```
print(classification_report(Y_test, grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.94      0.89      0.91        66
1       0.94      0.96      0.95       105

    accuracy                           0.94       171
macro avg       0.94      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```