Probability Assignment

To get full credit in this assignment you need to use `numpy`, `scipy` and `pandas` libraries. Sometimes you need to type equations - type equations in Latex math notation. To produce the plots you can use any plotting library you need.

PS1: We run the assignment through chatGPT the questions and you will be referred to the Dean if we find that a robot answered your questions.

PS2: We are also monitoring solution websites and we will take action against anyone that uploads this to a solution website.

Problem 1 (80 points)

A surgeon analyzes surgical videos and models events that occur. He describes the problem statement in here. Your job is to replicate the solution in Python and demonstrate your understanding of the steps performed by including adequate explanation of the code in either markdown cells or inline to the code. You can insert as many markdown or code cells you need to perform the analysis.

Question 1a (10 points)

Write the code for generating the `gs` variable. This is the simplest random variable of the problem and can be generated independent of the others.

```
[283] import numpy as np
      import matplotlib.pyplot as plot
      import pandas as pd
      import seaborn as sns
      import scipy.stats as sc_stats
```

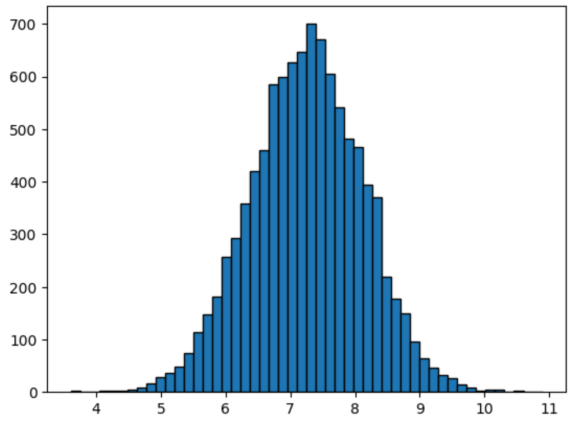
1a. Generate an array of 10^4 with given mean =7.25 and standard deviation = 0.875 that will follow normal distribution as per the case study rnorm function

```
[284] gs = np.random.normal(loc=7.25, scale=0.875, size=10000)
      gs.shape
```

```
(10000,)
```

```
plot.hist(gs, bins=50, edgecolor='black')
```

```
(array([  2.,   1.,   0.,   2.,   2.,   3.,   4.,   8.,  17.,  28.,  37.,
         49.,  74., 113., 148., 182., 257., 292., 359., 421., 460., 585.,
        600., 626., 647., 700., 670., 606., 542., 482., 465., 395., 370.,
        220., 178., 150.,  97.,  65.,  47.,  32.,  26.,  14.,   9.,   3.,
          4.,   4.,   0.,   3.,   0.,   1.]),
 array([ 3.60498819,  3.75076836,  3.89654853,  4.0423287 ,  4.18810887,
         4.33388904,  4.47966921,  4.62544938,  4.77122955,  4.91700972,
         5.06278989,  5.20857006,  5.35435023,  5.5001304 ,  5.64591057,
         5.79169074,  5.93747091,  6.08325108,  6.22903125,  6.37481142,
         6.52059159,  6.66637176,  6.81215193,  6.9579321 ,  7.10371227,
         7.24949244,  7.39527261,  7.54105278,  7.68683295,  7.83261312,
         7.97839329,  8.12417347,  8.26995364,  8.41573381,  8.56151398,
         8.70729415,  8.85307432,  8.99885449,  9.14463466,  9.29041483,
         9.436195  ,  9.58197517,  9.72775534,  9.87353551, 10.01931568,
        10.16509585, 10.31087602, 10.45665619, 10.60243636, 10.74821653,
        10.8939967 ]),
 <BarContainer object of 50 artists>)
```



Question 1b (20 points)

We have three variables, `ak`, `pp`, and `ptime`. Write the code for generating these variables from Multivate Gaussian distribution and replicate the associated plots.

1b. Genrating the variables ak, pp, ptime from Multivate Gaussian distribution and plotting the correlation of the associated plots.

```
[286] sigma = np.matrix('1 0.6 -0.9; 0.6 1 -0.5; -0.9 -0.5 1')
      sigma

      matrix([[ 1. ,  0.6, -0.9],
              [ 0.6,  1. , -0.5],
              [-0.9, -0.5,  1. ]])
```

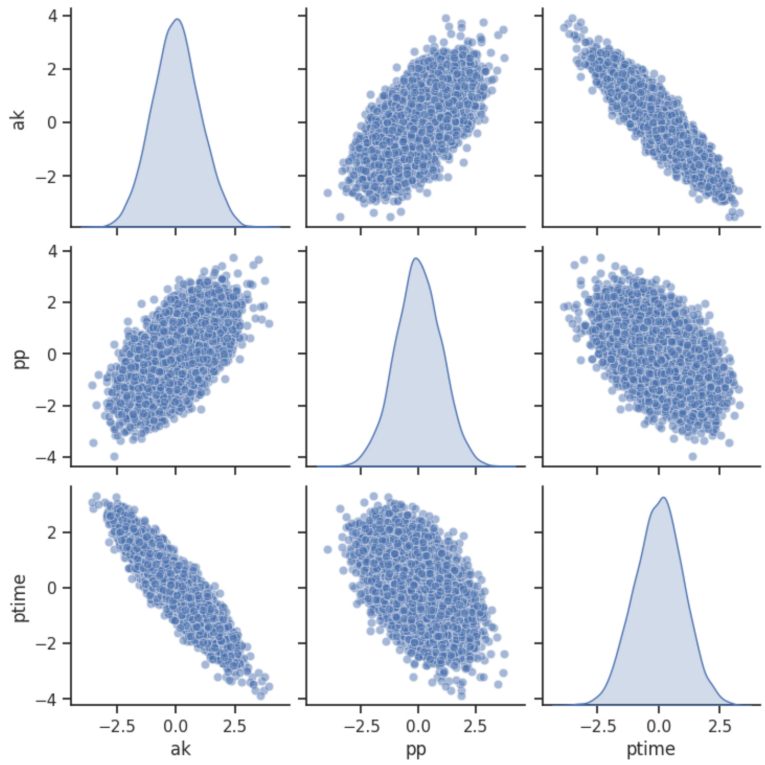
```
[287] mean = np.zeros(3)
      mean

      array([0., 0., 0.])
```

```
[288] np.random.seed(1234)
      APT = np.random.multivariate_normal(mean, sigma, size=10000)
      APT

      array([[-0.36127836,  0.34968669,  1.11819525],
             [ 0.35719837,  0.66903158,  0.12974387],
             [-0.94087986, -0.25111819,  1.00707132],
             ...,
             [-0.24770166, -0.23731461,  0.09608915],
             [ 0.01497189, -1.38883417, -0.2708827 ],
             [-1.18425028, -1.33327231,  0.84025677]])
```

```
[289] corr = np.corrcoef(APT, rowvar = False)
      corr

      array([[ 1.        ,  0.6048432 , -0.90063772],
             [ 0.6048432 ,  1.        , -0.51155345],
             [-0.90063772, -0.51155345,  1.        ]])
```

```
[290] df = pd.DataFrame(APT, columns=['ak', 'pp', 'ptime'])
```

```
[291] sns.set(style="ticks")
      sns.pairplot(df, diag_kind='kde', markers="o", plot_kws={'alpha': 0.5})
      plot.show()
```



Question 1c (20 points)

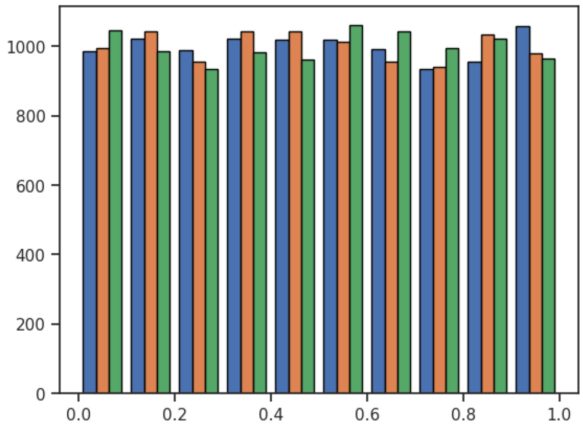
Perform the probability inrtegral transform and replicate the associated plots.

1c. For normal distribution, we have to consider the mean as 0 and standard deviation as 1 to yield a uniform distribution because we have created 3 correlated normal distribution(0,1) above. This is the Cumulatiove Distribution function and it is implemented using scioy's stats module's cdf -> Probility Integral Transform as follows.

```
[292] U = sc_stats.norm.cdf(APT, loc=0, scale=1)
```

```
[293] plot.hist(U, bins=10, edgecolor='black')

      (array([[ 987., 1022.,  990., 1021., 1020., 1019.,  992.,  934.,  956.,
               1059.],
              [ 995., 1044.,  955., 1042., 1042., 1013.,  956.,  940., 1033.,
                980.],
```

```
       [1047.,  985.,  935.,  984.,  961., 1062., 1044.,  994., 1023.,
         965.]]),
array([3.58997761e-05, 1.00027443e-01, 2.00018987e-01, 3.00010530e-01,
       4.00002073e-01, 4.99993617e-01, 5.99985160e-01, 6.99976703e-01,
       7.99968247e-01, 8.99959790e-01, 9.99951334e-01]),
<a list of 3 BarContainer objects>)
```



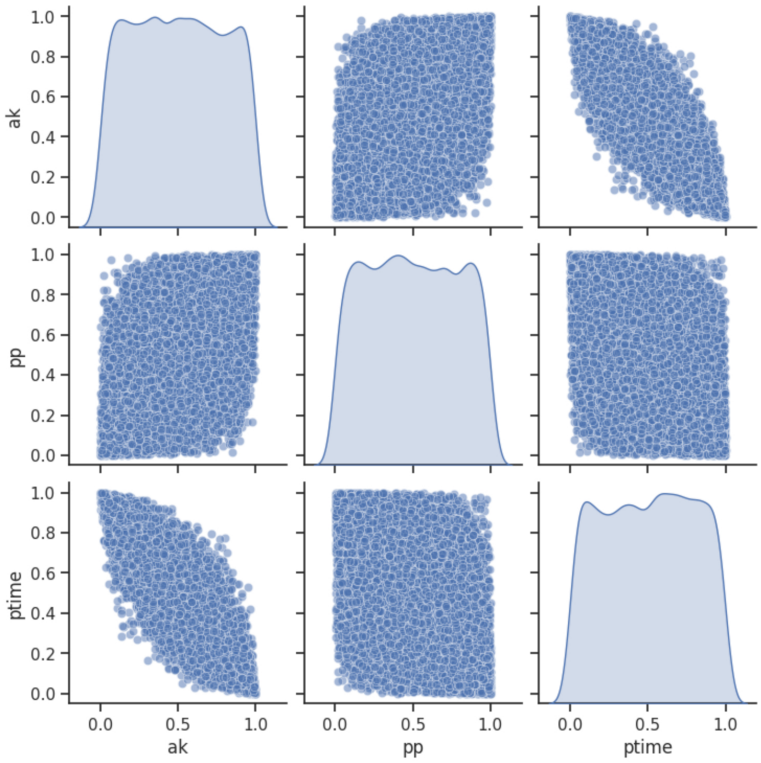
Therefore, as expected, the above graph represents a uniform distribution from 0 to 1. Now let's check the effect on the correlation above:

```
[294] corr_new = np.corrcoef(U, rowvar = False)
      corr_new
```

```
array([[ 1.        ,  0.58632699, -0.89178402],
       [ 0.58632699,  1.        , -0.49527681],
       [-0.89178402, -0.49527681,  1.        ]])
```

```
[295] df1 = pd.DataFrame(U, columns=['ak', 'pp', 'ptime'])
```

```
[296] sns.set(style='ticks')
      sns.pairplot(df1, diag_kind="kde", markers="o", plot_kws={'alpha': 0.5})
      plot.show()
```



Question 1d (20 points)

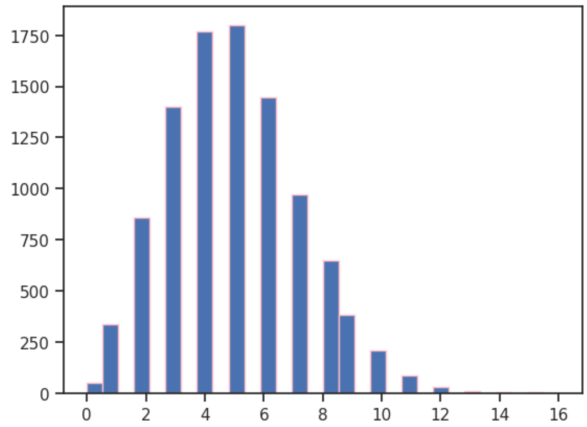
Perform the inverse transform sampling.

1d. ak: Our air knot variable, ak is in the first column of U. We will make it a Poisson distribution with an average number of air knots of 5 per case.

```
[297] ak = sc_stats.poisson.ppf(U[:, 0], 5)
      plot.hist(ak, bins=30, edgecolor='pink')
```

```
(array([4.800e+01, 3.380e+02, 0.000e+00, 8.580e+02, 0.000e+00, 1.403e+03,
        0.000e+00, 1.768e+03, 0.000e+00, 1.801e+03, 0.000e+00, 1.445e+03,
```

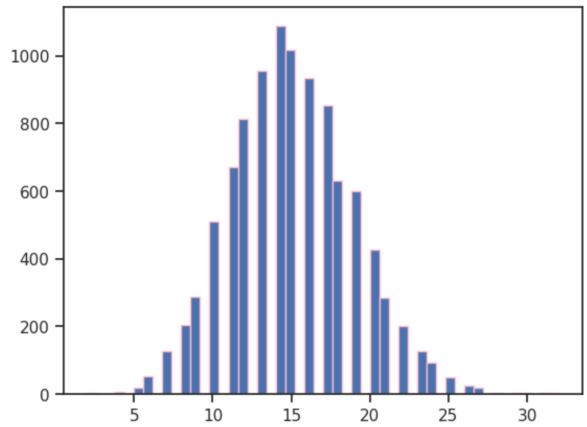
```
       0.000e+00, 9.730e+02, 0.000e+00, 6.470e+02, 3.820e+02, 0.000e+00,
       2.080e+02, 0.000e+00, 8.400e+01, 0.000e+00, 2.800e+01, 0.000e+00,
       7.000e+00, 0.000e+00, 5.000e+00, 0.000e+00, 4.000e+00, 1.000e+00]),
array([ 0.        ,  0.53333333,  1.06666667,  1.6       ,  2.13333333,
        2.66666667,  3.2       ,  3.73333333,  4.26666667,  4.8       ,
        5.33333333,  5.86666667,  6.4       ,  6.93333333,  7.46666667,
        8.        ,  8.53333333,  9.06666667,  9.6       , 10.13333333,
       10.66666667, 11.2       , 11.73333333, 12.26666667, 12.8       ,
       13.33333333, 13.86666667, 14.4       , 14.93333333, 15.46666667,
       16.        ]),
<BarContainer object of 30 artists>)
```



1d. pp: Our passes point variable, pp is in the second column of U. We will make it a Poisson distribution with a mean of 15 “passing point” happening per case.

```
[298] pp = sc_stats.poisson.ppf(U[:, 1] , 15)
      plot.hist(pp, bins=50, edgecolor='pink')
```

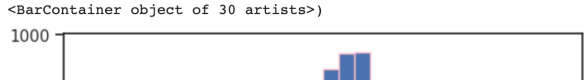
```
(array([1.000e+00, 0.000e+00, 0.000e+00, 7.000e+00, 0.000e+00, 1.900e+01,
       5.300e+01, 0.000e+00, 1.250e+02, 0.000e+00, 2.030e+02, 2.880e+02,
       0.000e+00, 5.100e+02, 0.000e+00, 6.710e+02, 8.130e+02, 0.000e+00,
       9.560e+02, 0.000e+00, 1.089e+03, 1.017e+03, 0.000e+00, 9.340e+02,
       0.000e+00, 8.540e+02, 6.310e+02, 0.000e+00, 5.990e+02, 0.000e+00,
       4.260e+02, 2.850e+02, 0.000e+00, 2.010e+02, 0.000e+00, 1.250e+02,
       9.100e+01, 0.000e+00, 4.800e+01, 0.000e+00, 2.300e+01, 1.900e+01,
       0.000e+00, 4.000e+00, 0.000e+00, 4.000e+00, 2.000e+00, 0.000e+00,
       1.000e+00, 1.000e+00]),
array([ 2. ,  2.6,  3.2,  3.8,  4.4,  5. ,  5.6,  6.2,  6.8,  7.4,  8. ,
        8.6,  9.2,  9.8, 10.4, 11. , 11.6, 12.2, 12.8, 13.4, 14. , 14.6,
       15.2, 15.8, 16.4, 17. , 17.6, 18.2, 18.8, 19.4, 20. , 20.6, 21.2,
       21.8, 22.4, 23. , 23.6, 24.2, 24.8, 25.4, 26. , 26.6, 27.2, 27.8,
       28.4, 29. , 29.6, 30.2, 30.8, 31.4, 32. ]),
<BarContainer object of 50 artists>)
```

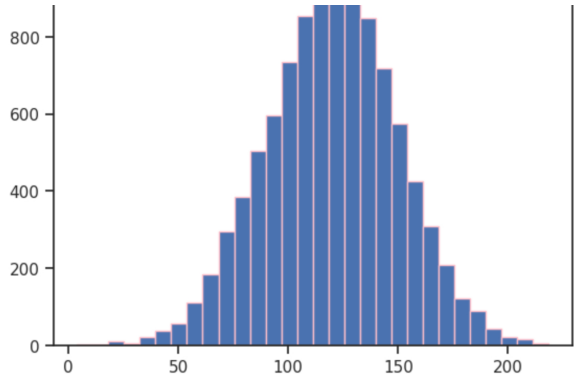


1d. ptime: The practice time variable, ptime, is in the third column of U. It represents the amount of time, in minutes, the resident spent practicing that week. We will make it a Normal distribution with a mean of 120 and standard deviation of 30.

```
[299] ptime = sc_stats.norm.ppf(U[:, 2], loc = 120, scale = 30)
      plot.hist(ptime, bins=30, edgecolor='pink')
```

```
(array([  3.,   3.,  10.,   6.,  20.,  37.,  57., 111., 184., 296., 386.,
       503., 595., 734., 855., 910., 952., 955., 849., 719., 574., 426.,
       309., 209., 122.,  90.,  44.,  22.,  15.,   4.]),
array([  3.87698164,  11.0299395 ,  18.18289736,  25.33585523,
        32.48881309,  39.64177095,  46.79472881,  53.94768667,
        61.10064454,  68.2536024 ,  75.40656026,  82.55951812,
        89.71247598,  96.86543385, 104.01839171, 111.17134957,
       118.32430743, 125.47726529, 132.63022316, 139.78318102,
       146.93613888, 154.08909674, 161.2420546 , 168.39501247,
       175.54797033, 182.70092819, 189.85388605, 197.00684391,
       204.15980178, 211.31275964, 218.4657175 ]),
<BarContainer object of 30 artists>)
```





Calculating the correlation coefficient between ak and pp

```
[300] corr_ak_pp = np.corrcoef(ak, pp, rowvar=False)[0,1]
      corr_ak_pp
```

```
0.596653543585595
```

Calculating the correlation coefficient between ak and ptime

```
[301] corr_ak_ptime = np.corrcoef(ak, ptime, rowvar=False)[0,1]
      corr_ak_ptime
```

```
-0.8893928092846783
```

Calculating the correlation coefficient between pp and ptime

```
[302] corr_pp_ptime = np.corrcoef(pp, ptime, rowvar=False)[0,1]
      corr_pp_ptime
```

```
-0.5101952523479483
```

Calculating the correlation coefficient between gs and ak

```
      corr_ak_gs = np.corrcoef(gs, ak, rowvar=False)[0,1]
      corr_ak_gs
```

```
0.012847937403571213
```

Calculating the correlation coefficient between gs and pp

```
[304] corr_pp_gs = np.corrcoef(gs, pp, rowvar=False)[0,1]
      corr_pp_gs
```

```
-0.006691918687535003
```

Calculating the correlation coefficient between gs and ptime

```
[305] corr_ptime_gs= np.corrcoef(gs, ptime, rowvar=False)[0,1]
      corr_ptime_gs
```

```
-0.015095159038861721
```

Question 1e (10 points)

Replicate the final plot showcasing the correlations between the variables.

1e. Finally correlations between all the variables as shown in the blow graph:

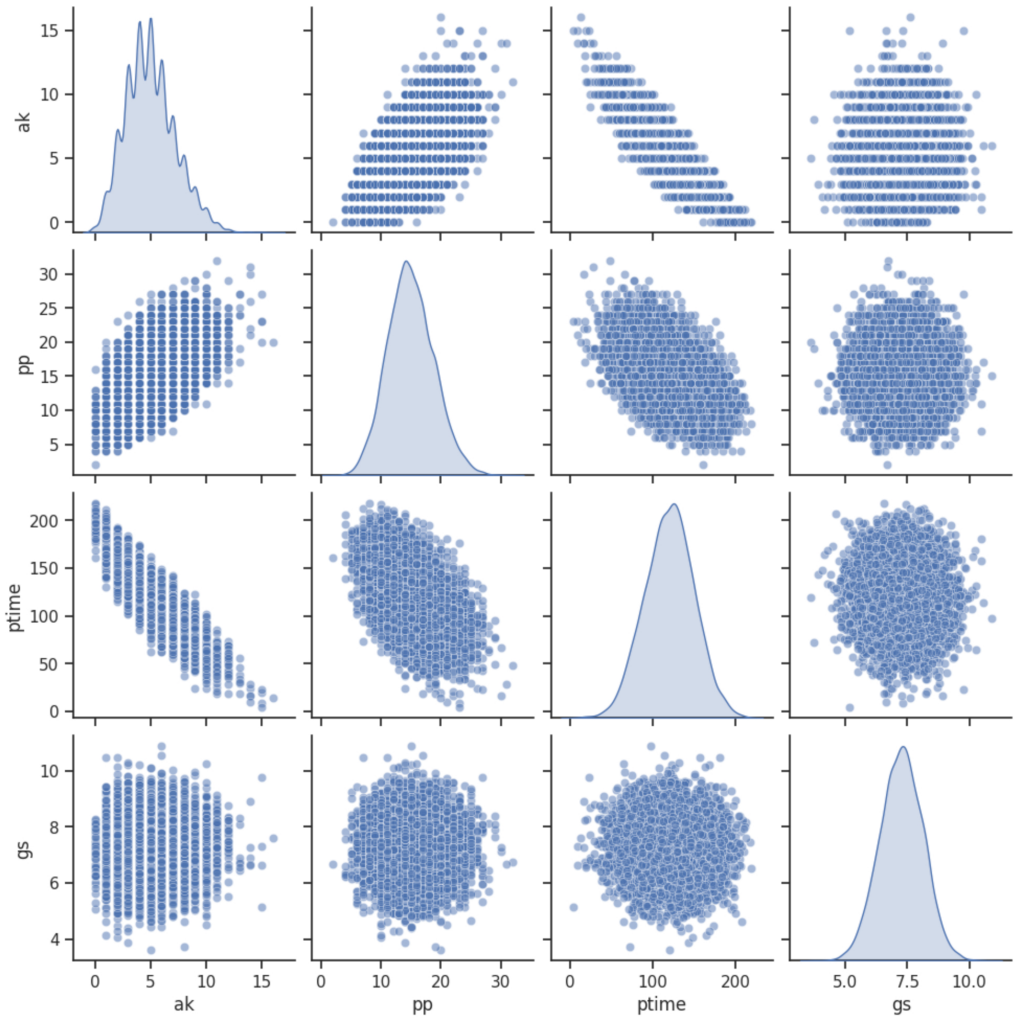
```
[306] df_all_new = pd.DataFrame({'ak':ak,'pp':pp,'ptime':ptime,'gs':gs })
      df_all_new
```

	ak	pp	ptime	gs
0	4.0	16.0	153.545858	7.274173
1	6.0	17.0	123.892316	7.149567
2	3.0	14.0	150.212140	5.939511
3	12.0	19.0	52.490476	7.043470
4	2.0	17.0	164.529162	7.508623
...	...	...	...	...
9995	4.0	11.0	116.674015	7.144330
9996	3.0	12.0	132.963977	8.993854
9997	4.0	14.0	122.882674	7.661322
9998	5.0	10.0	111.873519	7.464461

9999	2.0	10.0	145.207703	7.301731

10000 rows × 4 columns

```
[307] sns.set(style='ticks')
      sns.pairplot(df_all_new, diag_kind="kde", markers='o', plot_kws={'alpha' : 0.5})
      plot.show()
```



Problem 2 (20 points)

You now pretend that the $n = 4$ dimensional data you generated in Problem 1 arrive sequentially one at a time (the co-called **online** learning setting). Introduce the index i to represent the ith arriving data sample $\mathbf{x}_i$.

1. Write the expression of the *sample* correlation matrix (5 points)
2. Write the expression of the sample correlation matrix that can be estimated recursively and plot the elements of the sample correlation matrix from $i = 1$ to $i = 100$ (15 points)

2a. *Sample correlation matrix for a 4-dimensional data can be derived as follows*

2. Let x_i i.e $\{x_1, x_2, x_3, \ldots x_i\}$ be the incoming data samples, where each data sample x_i is a 4-dimensional vector i.e $[x_{i1}, x_{i2}, x_{i3}, x_{i4}]$
   - Calculating the sample 4-dimensional Mean vector "μ $(1 \times 4$ vector)"
     - $\mu_{ij} = \frac{1}{i} \sum_{k=1}^{i} x_{kj}$ where j ranges from 1 to dimensions (in this case j =1 .. 4)
       - μ_{ij} is the jth component of the mean
   - Centering the data : achieved by subtracting the mean vector from each data sample
     - $\hat{x_i} = x_i - \mu_i$
   - Sample co-variance matrix [4 x 4 matrix in this case]:
     - $S_i = \frac{1}{i-1} \sum_{k=1}^{i} \hat{x_k} \cdot (\hat{x_k})^T$
   - Now, Sample corelation matrix for the new set of data x_{i+1} as follows:
     - $S_{i+1} = \frac{1}{i-1+1} \sum_{k=1}^{i+1} \hat{x_k} \cdot (\hat{x_k})^T$, therefore:
     - $S_{i+1} = \frac{1}{i} \sum_{k=1}^{i+1} \hat{x_k} \cdot (\hat{x_k})^T$

2b. the sample correlation matrix that can be estimated recursively

- Mean of x_i : $\mu_{i+1j} = \frac{1}{i+1} \sum_{k=1}^{i+1} x_{kj}$
- Centralizing Data: $\hat{x_{i+1}} = x_{i+1} - \mu_{i+1j}$
- From 2a, $S_{i+1} = \frac{1}{i} \sum_{k=1}^{i+1} \hat{x_k} \cdot (\hat{x_k})^T$
- Simplifying and expanding the above:
  - $S_{i+1} = \frac{1}{i}[\sum_{k=1}^{i} (x_k - \mu_{i+1j}) \cdot (x_k - \mu_{i+1j})^T + (x_{i+1} - \mu_{i+1j}) \cdot (x_{i+1} - \mu_{i+1j})^T]$
  - $S_{i+1} = \frac{1}{i}[(i-1).S_i + (x_{i+1} - \mu_{i+1j}) \cdot (x_{i+1} - \mu_{i+1j})^T]$
  - $S_{i+1} = \frac{i-1}{i} S_i + \frac{1}{i}(x_{i+1} - \mu_{i+1j}) \cdot (x_{i+1} - \mu_{i+1j})^T]$
- Further Simplifying,
  - $S_{i+1} = (1 - \frac{1}{i}) * S_i + \frac{1}{i} * (\hat{x_{i+1}}) \cdot (\hat{x_{i+1}})^T)$

Here,

- $\frac{1}{i}$ is the learning rate for standardizing the incoming stream of data
- $((x_i - \mu_i) * (x_i - \mu_i)^T)$ represented by $\hat{x_i}$ is the sample corelation matrix before standardizing/normalising as per the calculation shown in the above cell. Hence, for the simplicity of the code below, we are using numpy's correlation function directly to calculate this part.
- S_i is the correlation matrix of the i-1 th dataset seen before the current datastream

Implementation 1

```
[270] # 2b. Creating the covariance matrix using pandas' cov() function
      new_sigma = df_all_new.cov()
      new_sigma
      # new_sigma.columns
```

	ak	pp	ptime	gs
ak	4.995698	5.160670	-59.411720	0.017366
pp	5.160670	14.975164	-59.006875	0.040435
ptime	-59.411720	-59.006875	893.224928	-0.214238
gs	0.017366	0.040435	-0.214238	0.767787

```
      new_mean = df_all_new.mean()
      new_mean = np.mean(new_sigma, axis = 0)
      new_mean
      # df_all_new["ak"].mean()
```

```
ak       -12.309496
pp        -9.707652
ptime    193.648024
gs         0.152838
dtype: float64
```

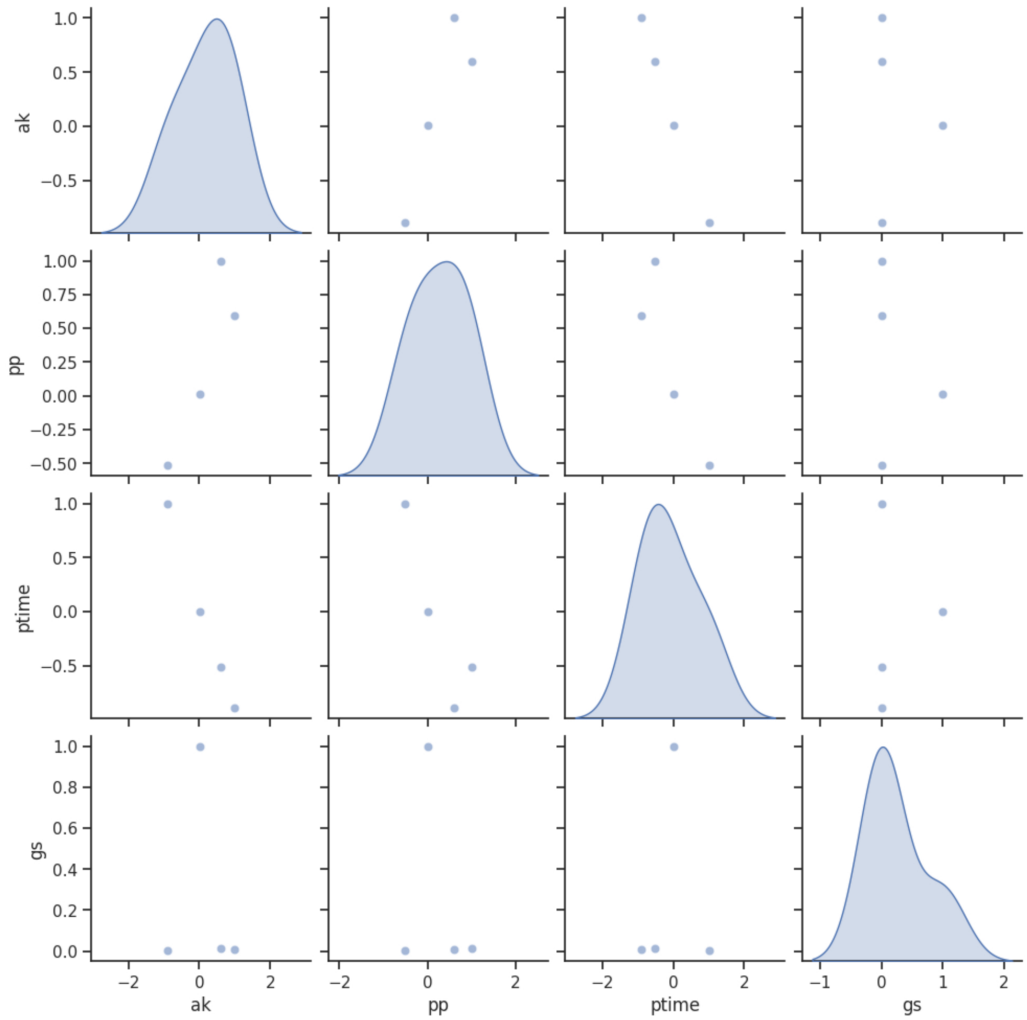
```
[272] n = 100
      sample_corr_mat = np.identity(4)

      for i in range(1, n + 1):
        # np.random.seed(1234)
        global sample_corr_mat
        global new_cov
        xi = np.random.multivariate_normal(new_mean, new_sigma, size = n)

        df_xi = pd.DataFrame(xi, columns=new_sigma.columns)

        # Append the new data to the streaming data frame
        new_cov = new_sigma.append(df_xi, ignore_index=True)
        # new_sigma = pd.concat([new_sigma, df_xi], ignore_index=True)

        # condidering the most recent set of data
        if len(new_cov) > n:
            new_cov = new_cov.tail(n)

        # correlation matrix for the new set
        new_correlation_matrix = new_cov.corr()

        # new_correlation_matrix
        # Si = (i-1)/i * S{i-1} + (1/i) * ((xi - mean(xi)) * (xi - mean(xi))^T)
        # sample_corr_mat = ((i - 1) / i) * new_correlation_matrix + (1 / i) * np.outer(xi - np.mean(xi), xi - np.mean(xi))

        # Updating the global correlation matrix using a recursive formula as per the derived recusrsive formula in #2b above
        learning_rate = 1 / i # 0.01
        # print(learning_rate)
        # print(learning_rate, new_correlation_matrix )
        sample_corr_mat = (1 - learning_rate) * sample_corr_mat + learning_rate * new_correlation_matrix
```

```
<ipython-input-214-f5e5955f612a>:13: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat
  new_cov = new_sigma.append(df_xi, ignore_index=True)
```

```
[273] # df_recursive = pd.DataFrame(new_cov)
      df_recursive = pd.DataFrame(sample_corr_mat)
      df_recursive
```

	ak	pp	ptime	gs
ak	1.000000	0.593912	-0.892120	0.008643
pp	0.593912	1.000000	-0.515326	0.013345
ptime	-0.892120	-0.515326	1.000000	0.000959
gs	0.008643	0.013345	0.000959	1.000000

```
sns.set(style='ticks')
sns.pairplot(df_recursive, diag_kind="kde", markers="o", plot_kws={'alpha': 0.5})
plot.show()
```



Implementation 2

```
[275] # 2b. Creating the covariance matrix using pandas' cov() function
      S0 = df_all_new.cov()
      S0
      # new_sigma.columns
```

	ak	pp	ptime	gs
ak	4.995698	5.160670	-59.411720	0.017366
pp	5.160670	14.975164	-59.006875	0.040435
ptime	-59.411720	-59.006875	893.224928	-0.214238
gs	0.017366	0.040435	-0.214238	0.767787

```
[276] curr_mean = np.mean(S0, axis=0)
      curr_mean
```

```
ak       -12.309496
pp        -9.707652
ptime    193.648024
gs         0.152838
dtype: float64
```

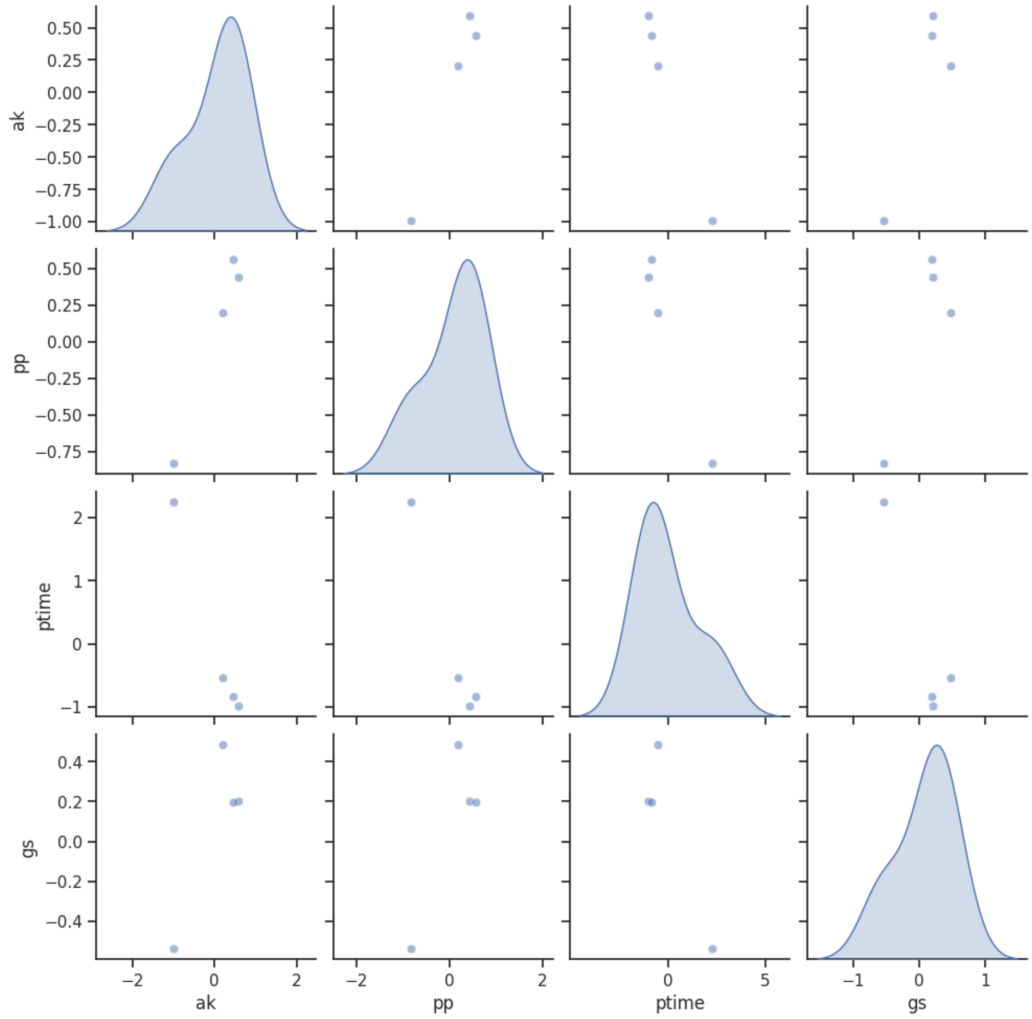
```
[280] n = 100

      for i in range(1, n + 1):

          # Mean vector for the multivariate normal distribution
```

```
        xi = np.random.multivariate_normal(curr_mean, S0, size = n)  # Adjust size as needed

        # Calculate the mean vector μ for the current data points observed up to i
        mu_xi = np.mean(xi)

        delta_xi = xi - mu_xi
        outer_product = np.outer(delta_xi, delta_xi)[:4,:4]
        new_correlation_matrix = outer_product / np.var(xi) # Calculate the correlation matrix for the new data


        # new_correlation_matrix
        # Si = (i-1)/i * S{i-1} + (1/i) * ((xi - mean(xi)) * (xi - mean(xi))^T)
        # new_Si = (1 - (1 / i)) * new_correlation_matrix + (1 / i) * np.outer(xi - np.mean(xi), xi - np.mean(xi))
        # Updating the sample correlation matrix as per the derived recusrsive formula in #2b above
        learning_rate = 1 / i
        sample_corr_mat = (1 - learning_rate) * sample_corr_mat + learning_rate * new_correlation_matrix
```

[281]
```
df_Si = pd.DataFrame(sample_corr_mat, columns=['ak','pp','ptime','gs'])
df_Si
```

	ak	pp	ptime	gs
ak	0.589558	0.442379	-0.992901	0.204070
pp	0.442379	0.565621	-0.833418	0.195773
ptime	-0.992901	-0.833418	2.257471	-0.540027
gs	0.204070	0.195773	-0.540027	0.487920

[282]
```
sns.set(style='ticks')
sns.pairplot(df_Si, diag_kind = 'kde', markers='o', plot_kws={'alpha': 0.5})
plot.show()
```



Implentation 3

```
n = 100

array = df_all_new.to_numpy
new_sigma2 = np.cov(df_all_new, rowvar=False)

new_mean2 = np.mean(new_sigma2, axis =0)

for i in range(1, n + 1):
  np.random.seed(1234)
```

```
    global sample_corr_mat1
    global new_cov1
    xi = np.random.multivariate_normal(new_mean2, new_sigma2, size = n)

    # df_xi = pd.DataFrame(xi, columns=new_sigma.columns)

    # Append the new data to the streaming data frame
    new_sigma2 = np.append(new_sigma2, xi, axis = 0)
    # new_sigma = pd.concat([new_sigma, df_xi], ignore_index=True)
    # print(new_sigma2.shape)

    # condidering the most recent set of data
    if len(new_sigma2) > n:
        new_sigma2 = new_sigma2[-n:]
    # print(new_sigma2.shape)

    # new_sigma2 = np.cov(new_cov.reshape(-1, 4), rowvar=False)
    # new_mean2 = np.mean(new_sigma2.reshape(-1, 4), axis=0)

    new_sigma2 = np.cov(new_sigma2, rowvar=False)
    new_mean2 = np.mean(new_sigma2, axis=0)
    # print(new_mean2, new_sigma2, new_sigma2.shape)

    # correlation matrix for the new set
    df_corr = pd.DataFrame(new_sigma2)
    # new_correlation_matrix = new_cov.corr()
    new_correlation_df_matrix = df_corr.corr()
    new_correlation_matrix = new_correlation_df_matrix.to_numpy()

    # new_correlation_matrix
    # the correlation matrix using the recursive formula
      # Si = (i-1)/i * S{i-1} + (1/i) * ((xi - mean(xi)) * (xi - mean(xi))^T)
    # new_Si = ((i - 1) / i) * new_correlation_matrix + (1 / i) * np.outer(xi - np.mean(xi), xi - np.mean(xi))

    # Updating the global correlation matrix using a recursive formula
    learning_rate = 1 / i # 0.01
    # print(learning_rate)
    # print(learning_rate, new_correlation_matrix )
    sample_corr_mat = (1 - learning_rate) * sample_corr_mat + learning_rate * new_correlation_matrix
```

```
[ ] df_recursive_new = pd.DataFrame(sample_corr_mat)
    # df_recursive_new = pd.DataFrame(new_sigma2)
    df_recursive_new
```

	ak	pp	ptime	gs
ak	1.000000	0.819018	-0.800732	-0.623606
pp	0.819018	1.000000	-0.331677	-0.902716
ptime	-0.800732	-0.331677	1.000000	0.092722
gs	-0.623606	-0.902716	0.092722	1.000000

```
[ ] sns.set(style='ticks')
    sns.pairplot(df_recursive_new, diag_kind = 'kde', markers='o', plot_kws={'alpha': 0.5})
    plot.show()
```