problemsy

June 24, 2021

1 Assignment 2

By: Paul Vogler (4979420), Robin Xu, and Yiftach Kolb (5195763)

1.1 Problem 1

1.1.1 part A,B

load data and scatter plot. In the scatter plot matrix, we see that some pairs like 'vectors' and 'algebra' seem more correlated than others.

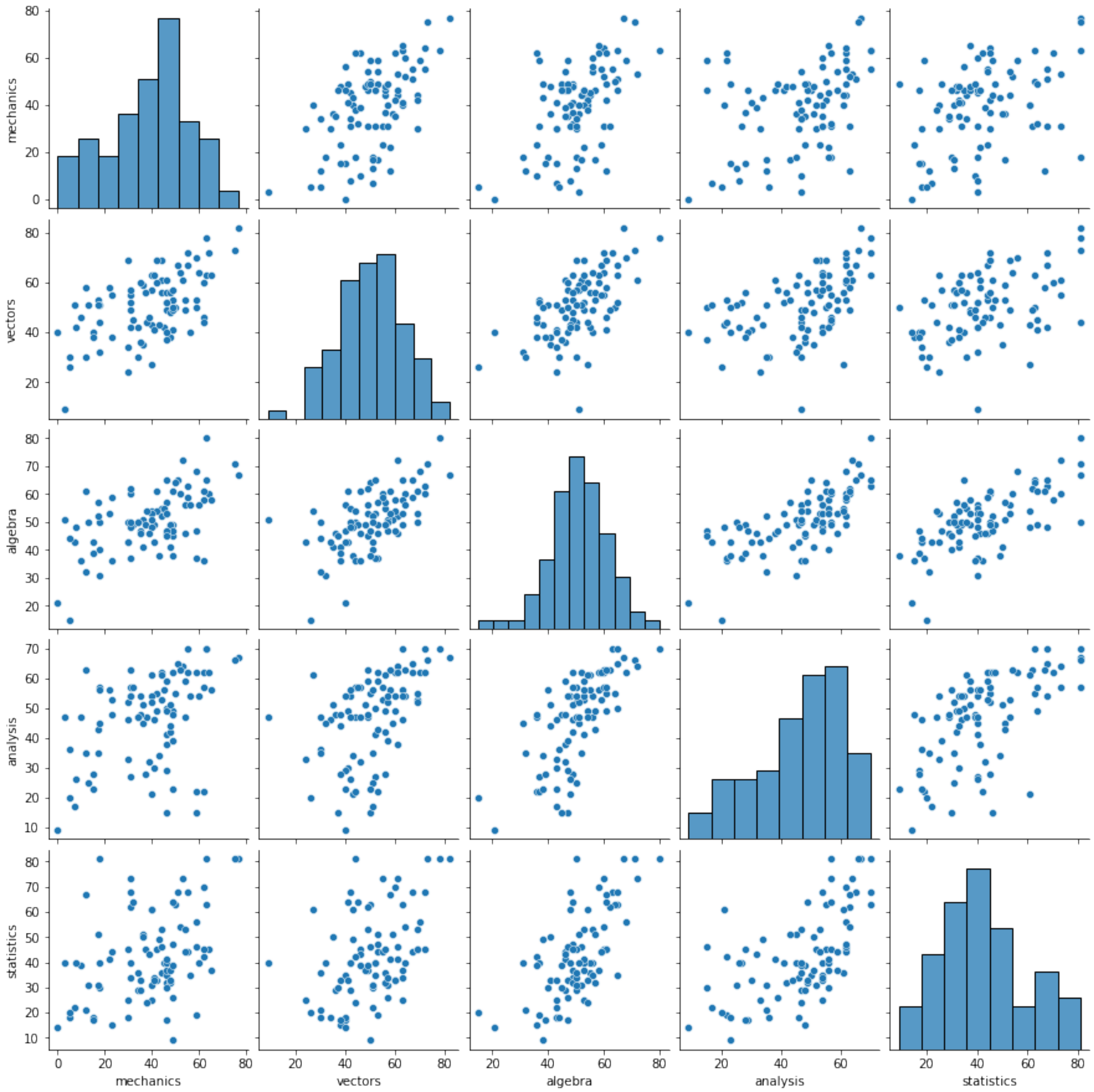
[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
import seaborn as sns
from scipy.spatial.distance import correlation
from statsmodels import datasets as moredatasets
from scipy.linalg import inv

import networkx as nx

from numpy.random import MT19937
from numpy.random import RandomState, SeedSequence
rs = RandomState(MT19937(SeedSequence(42)))

np.set_printoptions(precision=3)

df = pd.read_csv("./MathMarks.csv", header=0, sep=',')
df = df.drop(columns=['student_id'])
df.head()
```

[1]:

	mechanics	vectors	algebra	analysis	statistics
0	77	82	67	67	81
1	63	78	80	70	81
2	75	73	71	66	81
3	55	72	63	70	68
4	63	63	65	70	63

[2]:
```
sns.pairplot(df, )
```

[2]: <seaborn.axisgrid.PairGrid at 0x7fabf858fac0>

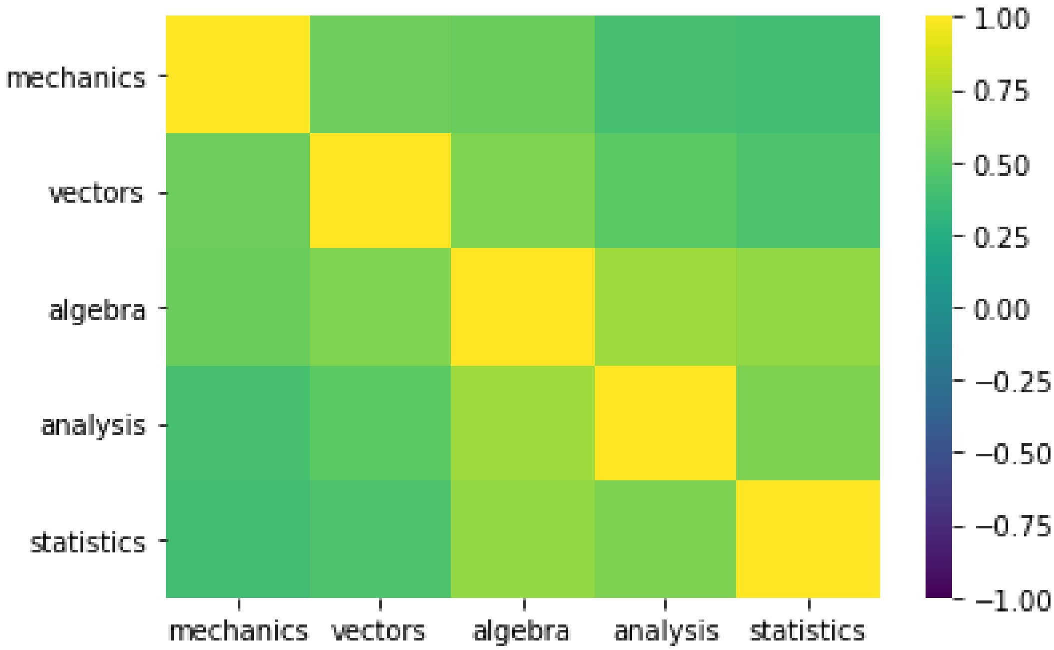


1.1.2 part C

Calculate the correlation matrix and plot it as a heat map.

[3]:
```
cormat = np.corrcoef(df, rowvar=False)
cormatdf = pd.DataFrame(cormat, columns=list(df.columns),
        index=list(df.columns))
sns.heatmap(cormatdf, center=0, vmin=-1, vmax=1,
        cmap='viridis')
```

[3]: <AxesSubplot:>



1.1.3 part D

Create the score matrix $S = (\frac{D_{ii}-1}{D_{ii}})$

[4]:
```
D = inv(cormat)
S = np.array([D[i,i] for i in range(5)])
S = (S - 1)/S
print(S)
```

```
[0.376 0.445 0.671 0.541 0.479]
```

1.1.4 Part E

Fitting one vs all models, getting the R^2 scores.

As we see below, the R^2 score isn't great. Nothing gets close to the perfect 1.

[5]:
```
r2scores = np.zeros(5)
j=0
for name in df.columns:
    y = df[[name]] # predict variable
    x = df.drop(columns=name) # explain variables
    regr = linear_model.LinearRegression()
    regr.fit(x, y)
    r2scores[j] = r2_score(regr.predict(x),y)
```

```
    j+=1

print(r2scores)
```

```
[-0.662 -0.248  0.51   0.151 -0.086]
```

1.1.5 Part F

calculate the covariance matrix Σ and the precision matrix P, and the normalized precision K.

[6]:
```
covmat = np.cov(df, rowvar=False)
p = inv(covmat)
scale = np.diagonal(p)
scale = np.sqrt(scale)
K = -p / scale
K = K.T
K = K / scale
K = K.T
print(K)
```

```
[[-1.000e+00  3.285e-01  2.292e-01 -7.122e-04  2.551e-02]
 [ 3.285e-01 -1.000e+00  2.816e-01  7.783e-02  1.997e-02]
 [ 2.292e-01  2.816e-01 -1.000e+00  4.318e-01  3.567e-01]
 [-7.122e-04  7.783e-02  4.318e-01 -1.000e+00  2.528e-01]
 [ 2.551e-02  1.997e-02  3.567e-01  2.528e-01 -1.000e+00]]
```

1.1.6 Part G, H

Fitting 1 vs 3 models and getting partial correlations

[7]:
```
## mechanics and vectors
ymech = df[['mechanics']]
yvect = df[['vectors']]
xtrain = df.drop(columns=["mechanics", "vectors"])
regr = linear_model.LinearRegression()
regr.fit(xtrain, ymech)
ymech_pred = regr.predict(xtrain)
regr.fit(xtrain, yvect)
yvect_pred = regr.predict(xtrain)
res_mech = ymech_pred - ymech
res_vect = yvect_pred - yvect
res_vect = res_vect.to_numpy().flatten()
res_mech = res_mech.to_numpy().flatten()
print(np.corrcoef(res_vect, res_mech),
    1 - correlation(res_vect, res_mech)) # correlation DISTANCE is 1 -
 ↪correlation!
```

```
[[1.    0.328]
 [0.328 1.   ]] 0.3284628168669589
```

[8]:
```
## mechanics and algebra
ymech = df[['mechanics']]
yvect = df[['algebra']]
xtrain = df.drop(columns=["mechanics", "algebra"])
regr = linear_model.LinearRegression()
regr.fit(xtrain, ymech)
ymech_pred = regr.predict(xtrain)
regr.fit(xtrain, yvect)
yvect_pred = regr.predict(xtrain)
res_mech = ymech_pred - ymech
res_vect = yvect_pred - yvect
res_vect = res_vect.to_numpy().flatten()
res_mech = res_mech.to_numpy().flatten()
print(np.corrcoef(res_vect, res_mech),
   1 - correlation(res_vect, res_mech)) # correlation DISTANCE is 1 -␣
 ↪correlation!
```

```
[[1.    0.229]
 [0.229 1.   ]] 0.2292441884913834
```

[9]:
```
## mechanics and analysis
ymech = df[['mechanics']]
yvect = df[['analysis']]
xtrain = df.drop(columns=["mechanics", "analysis"])
regr = linear_model.LinearRegression()
regr.fit(xtrain, ymech)
ymech_pred = regr.predict(xtrain)
regr.fit(xtrain, yvect)
yvect_pred = regr.predict(xtrain)
res_mech = ymech_pred - ymech
res_vect = yvect_pred - yvect
res_vect = res_vect.to_numpy().flatten()
res_mech = res_mech.to_numpy().flatten()
print(np.corrcoef(res_vect, res_mech),
   1 - correlation(res_vect, res_mech)) # correlation DISTANCE is 1 -␣
 ↪correlation!
```

```
[[ 1.000e+00 -7.122e-04]
 [-7.122e-04  1.000e+00]] -0.0007121818006188274
```

[10]:
```
## mechanics and analysis
ymech = df[['mechanics']]
yvect = df[['analysis']]
xtrain = df.drop(columns=["mechanics", "analysis"])
regr = linear_model.LinearRegression()
regr.fit(xtrain, ymech)
ymech_pred = regr.predict(xtrain)
regr.fit(xtrain, yvect)
```

```
yvect_pred = regr.predict(xtrain)
res_mech = ymech_pred - ymech
res_vect = yvect_pred - yvect
res_vect = res_vect.to_numpy().flatten()
res_mech = res_mech.to_numpy().flatten()
print(np.corrcoef(res_vect, res_mech),
   1 - correlation(res_vect, res_mech)) # correlation DISTANCE is 1 -␣
↪correlation!
```

```
[[ 1.000e+00 -7.122e-04]
 [-7.122e-04  1.000e+00]] -0.0007121818006188274
```

[11]:
```
print(K)
```

```
[[-1.000e+00  3.285e-01  2.292e-01 -7.122e-04  2.551e-02]
 [ 3.285e-01 -1.000e+00  2.816e-01  7.783e-02  1.997e-02]
 [ 2.292e-01  2.816e-01 -1.000e+00  4.318e-01  3.567e-01]
 [-7.122e-04  7.783e-02  4.318e-01 -1.000e+00  2.528e-01]
 [ 2.551e-02  1.997e-02  3.567e-01  2.528e-01 -1.000e+00]]
```

[12]:
```
print(K[:,0])
```

```
[-1.000e+00  3.285e-01  2.292e-01 -7.122e-04  2.551e-02]
```

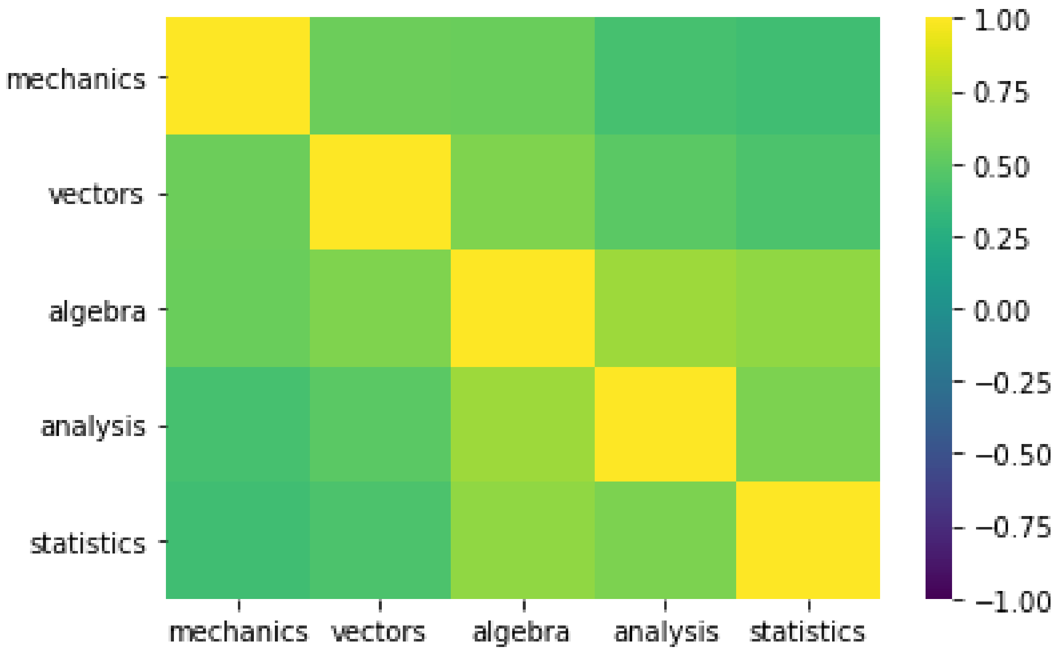
I think the first row of K is the partial correlations for “mechanics” which we have calculated above.

1.1.7 Part I

heatmaps

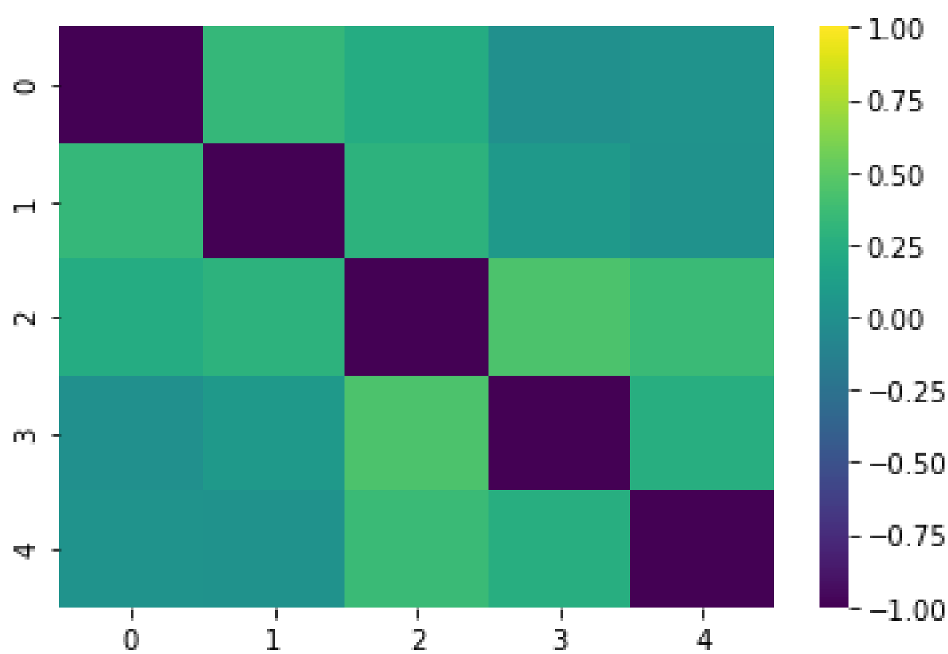
[13]:
```
sns.heatmap(cormatdf, center=0, vmin=-1, vmax=1,
        cmap='viridis')
```

[13]: <AxesSubplot:>



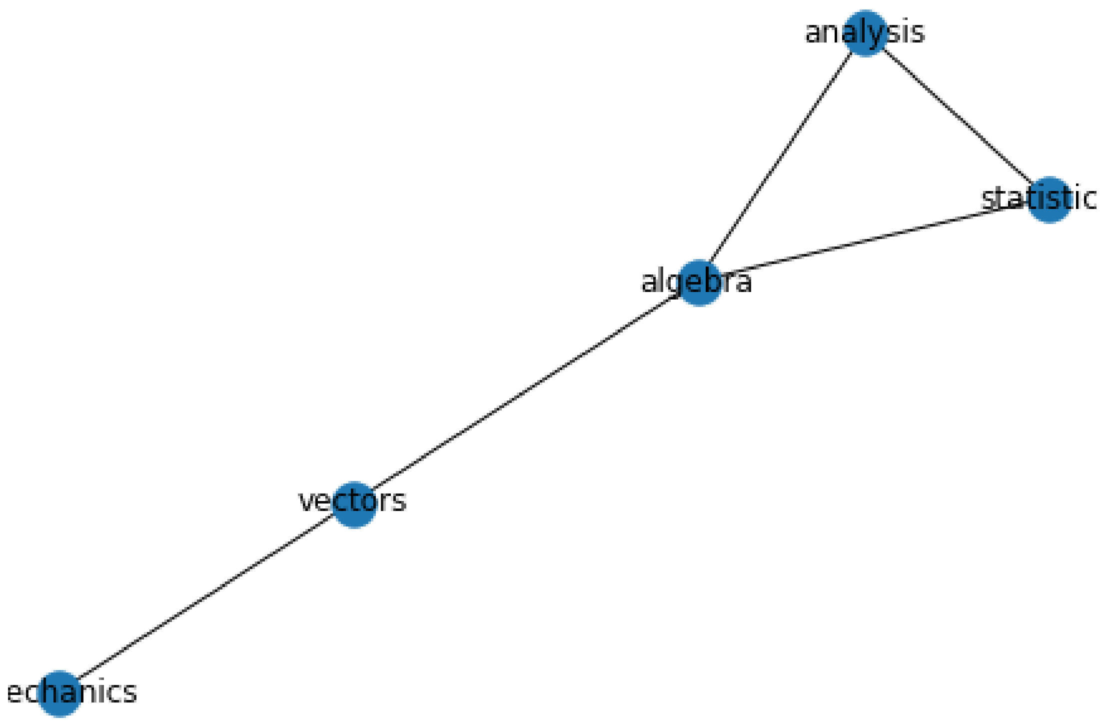
[14]:
```
sns.heatmap(K, center=0, vmin=-1, vmax=1,
          cmap='viridis')
```

[14]: <AxesSubplot:>



[15]:

```
g = nx.from_numpy_array(K > 0.25)

mylabels = list(df.columns)
mylabels = list(zip(range(5), mylabels))
mylabels = dict(mylabels)
mylabels

nx.draw_spring(g, labels=mylabels, )
```



1.2 Problem 2

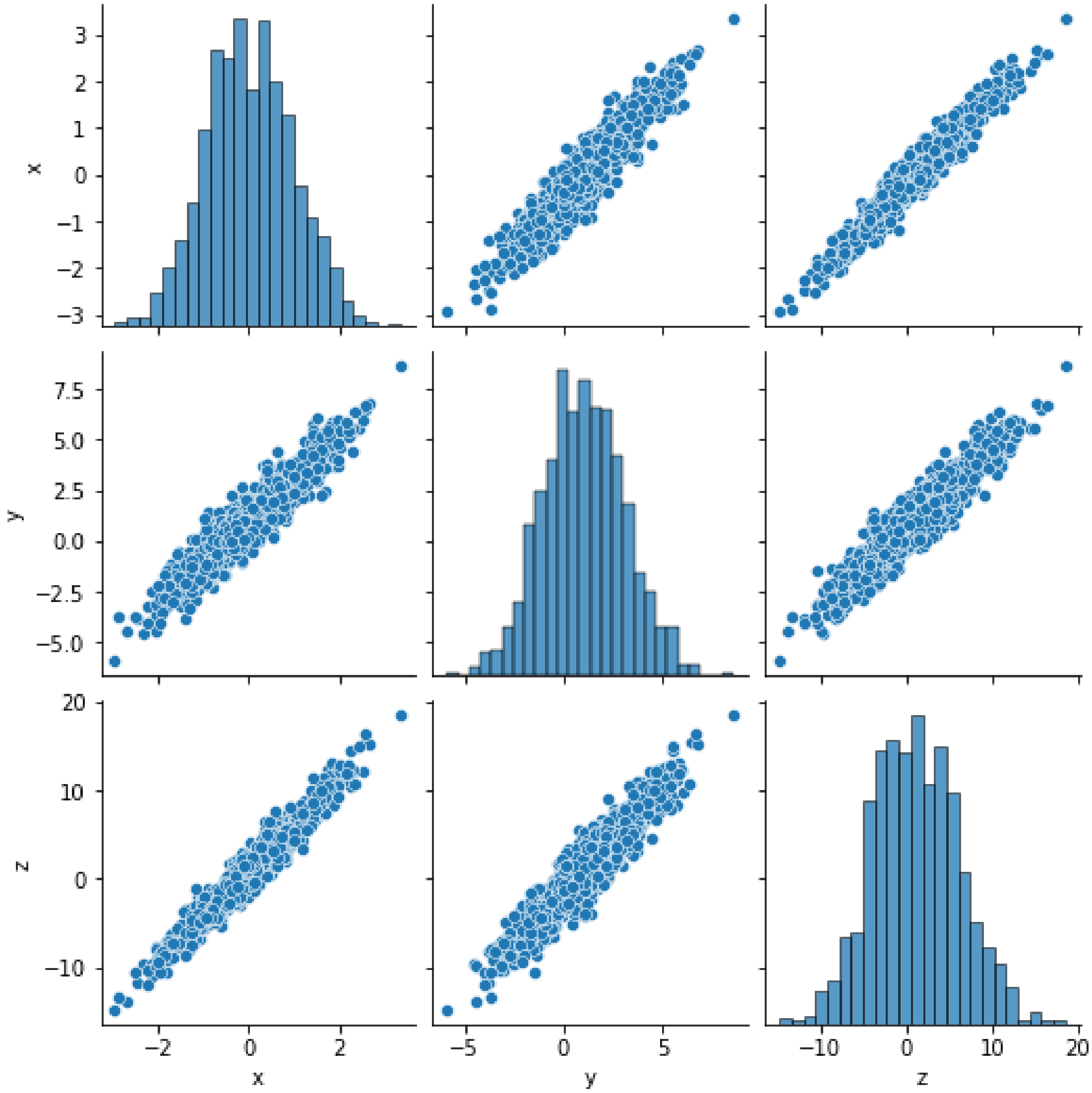
1.2.1 Part A

plots. x and y appear to be to most correlated, but all pairs appear to have correlation.

[16]:

```
x = np.random.normal(0,1,1000)
eps  = np.random.normal(0,0.5,1000)
y = np.random.normal(2*x + 1, 0.5) + eps
z = np.random.normal(5*x + 1, 1) + eps
mydata = {"x" : x, "y" : y, "z" : z}
mydata = pd.DataFrame(mydata)
sns.pairplot(mydata, )
```

[16]: <seaborn.axisgrid.PairGrid at 0x7fabc9691790>



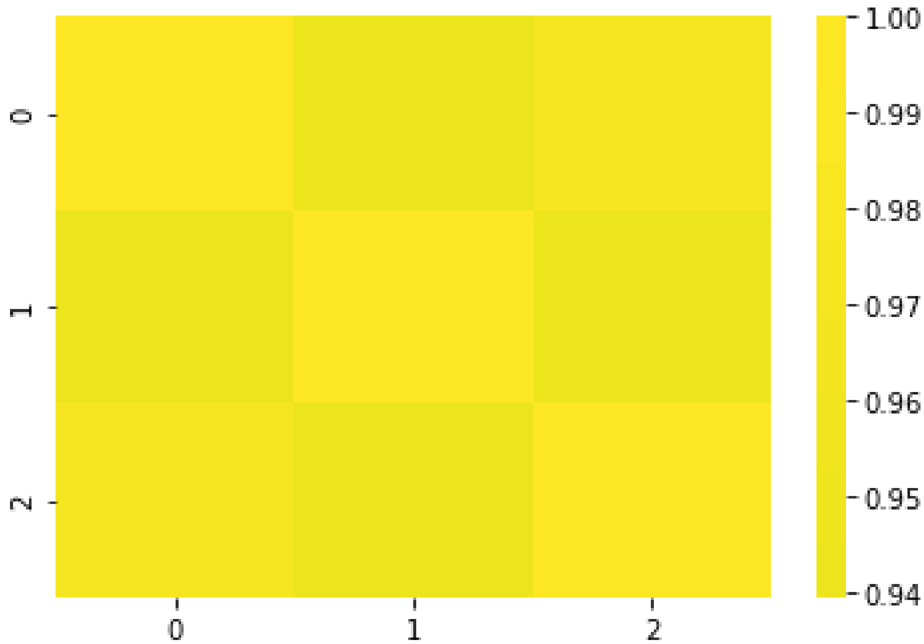
1.2.2 B

Correlation matrix and heatmap. All appear highly correlated.

[17]:
```
cormat= np.corrcoef(mydata, rowvar=False)
print(cormat)
sns.heatmap(cormat, center=0, cmap="viridis")
```

```
[[1.    0.94  0.974]
 [0.94  1.    0.942]
 [0.974 0.942 1.   ]]
```

[17]: <AxesSubplot:>



1.2.3 Part D Partial correlation matrix

we compute the matrix which according to the most beautiful theorem gives us the partial correlation.

[18]:
```
covmat = np.cov(mydata, rowvar=False)
p = inv(covmat)
scale = np.diagonal(p)
scale = np.sqrt(scale)
K = -p / scale
K = K.T
K = K / scale
K = K.T
print(K)
```

```
[[-1.     0.294  0.771]
 [ 0.294 -1.     0.346]
 [ 0.771  0.346 -1.   ]]
```

1.2.4 Part C Partial Correlation using the definition

Verify that it is indeed the same value as in the matrix above.

[19]:
```
regr = linear_model.LinearRegression()
regr.fit(X=mydata[["x"]], y=mydata[["z"]])
zhat = regr.predict(mydata[["x"]])
regr = linear_model.LinearRegression()
regr.fit(X=mydata[["x"]], y=mydata[["y"]])
yhat = regr.predict(mydata[["x"]])

zhat = zhat.flatten()
yhat = yhat.flatten()
np.corrcoef(z - zhat, y - yhat)
1 - correlation(y-yhat, z-zhat)
```

[19]: 0.34637490047500896

[20]:
```
regr = linear_model.LinearRegression()
regr.fit(X=mydata[["y"]], y=mydata[["z"]])
zhat = regr.predict(mydata[["y"]])

regr = linear_model.LinearRegression()
regr.fit(X=mydata[["y"]], y=mydata[["x"]])
xhat = regr.predict(mydata[["y"]])

zhat = zhat.flatten()
xhat = xhat.flatten()

1 - correlation(x-xhat, z-zhat)
```

[20]: 0.7711318924496935

[21]:
```
regr = linear_model.LinearRegression()
regr.fit(X=mydata[["z"]], y=mydata[["x"]])
xhat = regr.predict(mydata[["z"]])

regr = linear_model.LinearRegression()
regr.fit(X=mydata[["z"]], y=mydata[["y"]])
yhat = regr.predict(mydata[["z"]])

xhat = xhat.flatten()
yhat = yhat.flatten()

np.corrcoef(x - xhat, y - yhat)

1 - correlation(y-yhat, x-xhat)
```

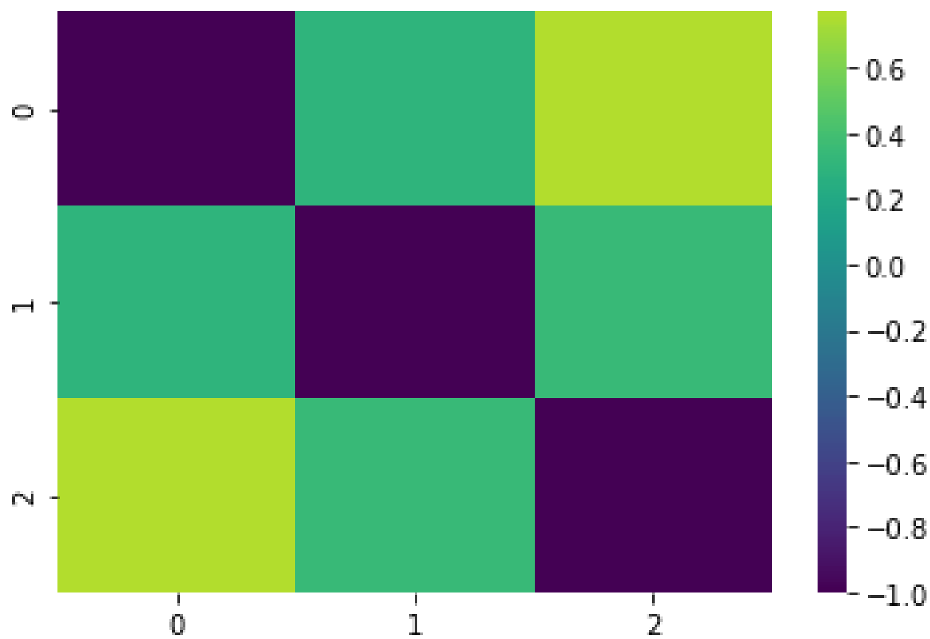
[21]: 0.2941011729114642

1.2.5 Part E

Heatmap of the partial correlation map. We observe that 0 x and z appear to be more partially correlated.

[22]:
```
sns.heatmap(K, center=0, cmap="viridis")

print(K)

#p @ covmat
```

```
[[-1.     0.294  0.771]
 [ 0.294 -1.     0.346]
 [ 0.771  0.346 -1.   ]]
```



1.3 Problem 3

1.3.1 Part A

Because the variance of each single variable is 1 the correlation coefficient matrix should approach covariance matrix if the sample is large enough.

[23]:
```
### Aifgabe 3
mu = np.zeros(2)
sig = np.array([[1,0.6], [0.6,1]])

Z = np.random.multivariate_normal(mu, sig, 10)
print("with n=10:\n",np.corrcoef(Z, rowvar=False))
```

```
Z = np.random.multivariate_normal(mu, sig, 100)
print("\n\n with n=100:\n",np.corrcoef(Z, rowvar=False))
Z = np.random.multivariate_normal(mu, sig, 1000)
print("\n\n with n=1000:\n",np.corrcoef(Z, rowvar=False))
```

```
with n=10:
 [[1.    0.651]
 [0.651 1.   ]]


 with n=100:
 [[1.    0.674]
 [0.674 1.   ]]


 with n=1000:
 [[1.    0.602]
 [0.602 1.   ]]
```

1.3.2 2 Part B

[24]:
```
x = np.random.normal(0,2,100)
y = np.random.normal(1,np.sqrt(3),100)
print(np.corrcoef(x,y))
```

```
[[ 1.    -0.044]
 [-0.044  1.   ]]
```

Since these two are Gaussian they are independt if and only iff their correlation is 0. As a matter of fact they are independent by design. The correlation above is an estimation of the true correlation (0) and it comes pretty close to 0.

[ ]:
```

```