CSCI4022 Homework 8; Filtering and Matrix Methods

Due Friday, April 22 at 11:59 pm to Canvas and Gradescope

Submit this file as a .ipynb with *all cells compiled and run* to the associated dropbox.

Your solutions to computational questions should include any specified Python code and results as well as written commentary on your conclusions. Remember that you are encouraged to discuss the problems with your classmates, but **you must write all code and solutions on your own**.

NOTES:

- Any relevant data sets should be available on Canvas. To make life easier on the graders if they need to run your code, do not change the relative path names here. Instead, move the files around on your computer.
- If you're not familiar with typesetting math directly into Markdown then by all means, do your work on paper first and then typeset it later. Here is a reference guide (https://math.meta.stackexchange.com/questions/5020/mathjax-basic-tutorial-and-quick-reference) linked on Canvas on writing math in Markdown. **All** of your written commentary, justifications and mathematical work should be in Markdown. I also recommend the wikibook (https://en.wikibooks.org/wiki/LaTeX) for LaTex.
- Because you can technically evaluate notebook cells is a non-linear order, it's a good idea to do **Kernel** → **Restart & Run All** as a check before submitting your solutions. That way if we need to run your code you will know that it will work as expected.
- It is **bad form** to make your reader interpret numerical output from your code. If a question asks you to compute some value from the data you should show your code output **AND** write a summary of the results in Markdown directly below your code.
- 45 points of this assignment are in problems. The remaining 5 are for neatness, style, and overall exposition of both code and text.
- This probably goes without saying, but... For any question that asks you to calculate something, you **must show all work and justify your answers to receive credit**. Sparse or nonexistent work will receive sparse or nonexistent credit.
- There is *not a prescribed API* for these problems. You may answer coding questions with whatever syntax or object typing you deem fit. Your evaluation will primarily live in the clarity of how well you present your final results, so don't skip over any interpretations! Your code should still be commented and readable to ensure you followed the given course algorithm.
- There are two ways to quickly make a .pdf out of this notebook for Gradescope submission. Either:
  - Use File -> Download as PDF via LaTeX. This will require your system path find a working install of a TeX compiler
  - Easier: Use File -> Print Preview, and then Right-Click -> Print using your default browser and "Print to PDF"

Shortcuts: Problem 1 | Problem 2 | Problem 3 |

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import networkx as nx
from sklearn.neighbors import NearestNeighbors
from sklearn.ensemble import IsolationForest
from PIL import Image, ImageOps
```

Back to top

Problem 1 (k-NN; 15 pts)

The Anage (https://genomics.senescence.info/species/index.html) database is a large repositiory of biological information, and includes various properties of various animals with an eye towards understanding the effects of aging in different species. Most of the columns should be self-explanatory, but for more information you can consult their overview, here (https://genomics.senescence.info/help.html#anage).

A couple of acronyms (https://genomics.senescence.info/software/demographic.html) in the columns: IMR stands for "initial mortality rate," and represents non-age related deaths. MRDT is the "mortality doubling rate," and is a measure for the age-related species deaths.

Because it has many missing values, we will be exploring this data set as a nearest-neighbor algorith, which gives us the option to **impute** missing data as we compute similarities.

The data is loaded in with a couple of preliminary explorations below.

In [2]:
```
df=pd.read_csv('./data/AnAge.csv', encoding='UTF-8')
print(df.columns) #information available: many numeric columns are blank for many
print(df.shape) #4224 species available
df.head(3)
```

```
Index(['HAGRID', 'Kingdom', 'Phylum', 'Class', 'Order', 'Family', 'Genus',
       'Species', 'Common name', 'Female maturity (days)',
       'Male maturity (days)', 'Gestation/Incubation (days)', 'Weaning (day
s)',
       'Litter/Clutch size', 'Litters/Clutches per year',
       'Inter-litter/Interbirth interval', 'Birth weight (g)',
       'Weaning weight (g)', 'Adult weight (g)', 'Growth rate (1/days)',
       'Maximum longevity (yrs)', 'Source', 'Specimen origin', 'Sample size',
       'Data quality', 'IMR (per yr)', 'MRDT (yrs)', 'Metabolic rate (W)',
       'Body mass (g)', 'Temperature (K)', 'References'],
      dtype='object')
(4224, 31)
```

Out[2]:

	HAGRID	Kingdom	Phylum	Class	Order	Family	Genus	Spec
0	3.0	Animalia	Arthropoda	Branchiopoda	Diplostraca	Daphniidae	Daphnia	pulic
1	5.0	Animalia	Arthropoda	Insecta	Diptera	Drosophilidae	Drosophila	melanoga

Part A: Processing

As you can see above, the data set uses null values (`np.nan` , after loading) to denote missing information. To set up using centered cosine similarity, you should:

- Subtract from each column the non-NaN mean for that column
- Replace any NaN values with 0's, so they don't affect cosine similarity.
- Drop any non-numeric columns or columns where the values don't correspond to physical properties of that species

In [27]:
```
cd = df.select_dtypes('number')
cd = cd.drop('References',1)


for i in cd.columns:
    cd[i] = cd[i]-(np.sum(cd[i])/cd[i].count())
    cd[i] = cd[i].replace(np.nan, 0)
cd.head()
```

Out[27]:

	HAGRID	Female maturity (days)	Male maturity (days)	Gestation/Incubation (days)	Weaning (days)	Litter/Clutch size	Litters/Clut per
0	-2119.987201	0.000000	0.000000	0.0	0.0	0.0	
1	-2117.987201	-1003.475686	-909.159624	0.0	0.0	0.0	
2	-2116.987201	0.000000	0.000000	0.0	0.0	0.0	
3	-2114.987201	0.000000	0.000000	0.0	0.0	0.0	
4	-2113.987201	0.000000	0.000000	0.0	0.0	0.0	

Part B: Finding a Recommendation

Zach's girlfriend thinks the `Common name` animal of `Slow loris` is the cutest creature in the world. Use sklearn's `NearestNeighbors` to suggest to her 5 other animals that might be similar to the slow loris. Use only the numeric data columns in the given data frame.

In [4]:
```
loris=df['Common name']=='Slow loris'
#here's the data:
df.loc[loris]
```

Out[4]:

	HAGRID	Kingdom	Phylum	Class	Order	Family	Genus	Species	Common name	n
2439	2456.0	Animalia	Chordata	Mammalia	Primates	Lorisidae	Nycticebus	coucang	Slow loris	

1 rows × 31 columns

In [89]:
```
neighbors = NearestNeighbors(n_neighbors=6)
neighbors.fit(cd)
similar = neighbors.kneighbors(np.reshape(cd.iloc[2439].to_list(),(1,-1)))
```

In [91]:
```
for i in similar[1]:
    print(df.iloc[i])
```

```
      HAGRID   Kingdom    Phylum     Class     Order         Family  \
2439  2456.0  Animalia  Chordata  Mammalia  Primates      Lorisidae
2284  2300.0  Animalia  Chordata  Mammalia  Primates        Aotidae
2402  2419.0  Animalia  Chordata  Mammalia  Primates      Galagidae
2471  2488.0  Animalia  Chordata  Mammalia  Rodentia    Capromyidae
2459  2476.0  Animalia  Chordata  Mammalia  Rodentia  Aplodontiidae
2742  2759.0  Animalia  Chordata  Mammalia  Rodentia      Sciuridae

           Genus         Species               Common name  \
2439   Nycticebus         coucang                Slow loris
2284        Aotus     trivirgatus     Northern night monkey
2402    Otolemur  crassicaudatus            Greater galago
2471  Geocapromys       ingrahami            Bahamian hutia
2459   Aplodontia           rufa           Mountain beaver
2742      Cynomys    ludovicianus  Black-tailed prairie dog

      Female maturity (days)  ...  Source  Specimen origin  Sample size  \
2439                   578.0  ...     671        captivity       medium
2284                   821.0  ...     671        captivity       medium
2402                   495.0  ...     671        captivity       medium
2471                   730.0  ...     NaN          unknown        small
2459                   730.0  ...     NaN          unknown        small
2742                   730.0  ...     671        captivity        large

      Data quality  IMR (per yr)  MRDT (yrs)  Metabolic rate (W)  \
2439    acceptable           NaN         NaN               1.504
2284    acceptable           NaN         NaN               2.499
2402    acceptable           NaN         NaN               2.595
2471           low           NaN         NaN               1.483
2459           low           NaN         NaN               1.892
2742          high           NaN         NaN               2.358

      Body mass (g)  Temperature (K)    References
2439         1128.6           308.55  3.642040e+32
2284          914.5           311.15  3.642040e+28
2402          993.5           309.75  3.659380e+37
2471          775.0           309.15  3.642040e+19
2459          706.8           311.15  3.642040e+23
2742         1112.3              NaN  3.644150e+19

[6 rows x 31 columns]
```

The 5 most similar animals to the slow loris in descending order are: Northern night monkey, greater galago, bahamian hutia, mountain beaver and the black-tailed prairie dog.

Part C: Imputing Missing Data

Zach watched a cool documentary with a segment about the `Arctic fox` and watched a very young fox that had just reached weaning go on its first hunt. He was wondering about how large that fox was, but unfortunately, the data for the `Weaning weight (g)` of the artic fox is missing!

Predict the value of this missing piece of information by:

- Finding the 10 most similar animals to the `Arctic fox` that have weaning weight data available
- Predicting the weaning weight of the arctic fox via a similarity-weighted mean of those 10 animals' weaning weights
- Remember that your data is currently **centered**, so you'll need to add the mean back in for prediction

Sanity check your answer by eyeballing the weaning weight for another fox such as the `Red fox`.

In [200]:
```
aFox=df['Common name']=='Arctic fox'
cdwW = cd.loc[cd["Weaning weight (g)"]!=0]
neighbors = NearestNeighbors(n_neighbors=10)
neighbors.fit(cd.loc[cd["Weaning weight (g)"]!=0])
similar = neighbors.kneighbors(np.reshape(cd.iloc[1735,:].to_list(),(1,-1)))
```

In [201]:
```
dfwW = cd["Weaning weight (g)"]!=0
dfwW = df[dfwW]
dfwW
```

Out[201]:

	HAGRID	Kingdom	Phylum	Class	Order	Family	Genus	Species
1530	1542.0	Animalia	Chordata	Mammalia	Afrosoricida	Tenrecidae	Echinops	telfair
1536	1548.0	Animalia	Chordata	Mammalia	Afrosoricida	Tenrecidae	Tenrec	ecaudatus
1537	1549.0	Animalia	Chordata	Mammalia	Artiodactyla	Antilocapridae	Antilocapra	americana
1539	1551.0	Animalia	Chordata	Mammalia	Artiodactyla	Bovidae	Aepyceros	melampus
1546	1558.0	Animalia	Chordata	Mammalia	Artiodactyla	Bovidae	Bison	bison
...	...	...	...	...	...	...	...	...
2851	2868.0	Animalia	Chordata	Mammalia	Soricomorpha	Soricidae	Suncus	murinus
2852	2869.0	Animalia	Chordata	Mammalia	Soricomorpha	Talpidae	Condylura	cristata
2855	2872.0	Animalia	Chordata	Mammalia	Soricomorpha	Talpidae	Parascalops	breweri
2857	2874.0	Animalia	Chordata	Mammalia	Soricomorpha	Talpidae	Talpa	europaea
2858	2875.0	Animalia	Chordata	Mammalia	Tubulidentata	Orycteropodidae	Orycteropus	afer

405 rows × 31 columns

In [202]:
```
sum_ = 0
mean = (np.sum(df["Weaning weight (g)"])/df["Weaning weight (g)"].count())
for i in similar[1][0]:
    sum_+=dfwW.iloc[i]["Weaning weight (g)"]
    print(dfwW.iloc[i]["Common name"],dfwW.iloc[i]["Weaning weight (g)"])
sum_ = sum_/10
print("Mean weining weight for 10 most similar animals to the Artic Fox:",sum_)
```

```
Finless porpoise 24500.0
Eastern roe deer 28750.0
Sika deer 28590.0
Brown hyena 27400.0
Dorcas gazelle 7000.0
Western gray kangaroo 11000.0
Sea otter 13425.0
Tammar wallaby 3000.0
Ocelot 3400.0
Black spider monkey 3790.0
Mean weining weight for 10 most similar animals to the Artic Fox: 15085.5
```

In [203]:
```
rFox=df['Common name']=='Red fox'
df[rFox]["Weaning weight (g)"]
```

Out[203]:
```
1740    1397.0
Name: Weaning weight (g), dtype: float64
```

After some sanity checking the returned animals do seem to be mostly alright considering those that weining data is avalible for but the mean weaning weight is off by a factor of 10

Back to top

Problem 2 (SVD/PCA; 15 pts)

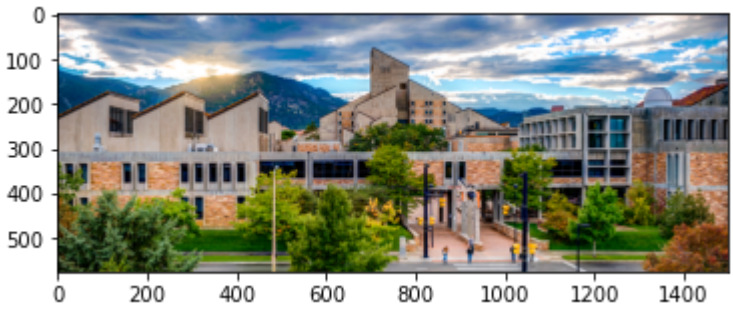
In this problem we'll explore the idea of using SVD as a memory saving tool. In particular, it's sometimes used for image compression: deciding how to take a larger image and save it using less information than all of the pixels.

Syntax: Image processing

Given is some code to load in an image of the engineering center. Play around with it and make sure you understand the dimensions of the rbg and grayscale matrices. Check out `plt.imshow` for documentation of the shapes of pixelated objects that can make sense to plot. **READ** the difference between inputting a single matrix and 3-D array: you'll need to think about rescaling/normalizing if you do part C!

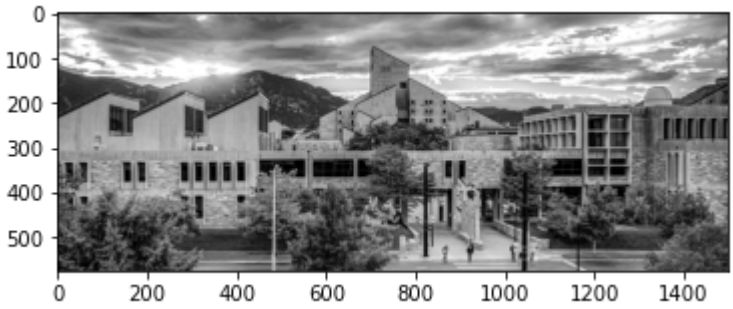
In [204]:
```
img = Image.open('./Data/cu_2012.png')
plt.imshow(img)

rbg=np.reshape(img.getdata(), (img.size[1],img.size[0],4)) #4 m x n arrays; R, B,
print(np.shape(rbg))
```

```
(575, 1500, 4)
```



In [226]:
```
grayscale=np.reshape(ImageOps.grayscale(img).getdata(), (img.size[1],img.size[0])
plt.imshow(grayscale, cmap='gray')
print(np.shape(grayscale))
```

```
(575, 1500)
```



Part A: SVD

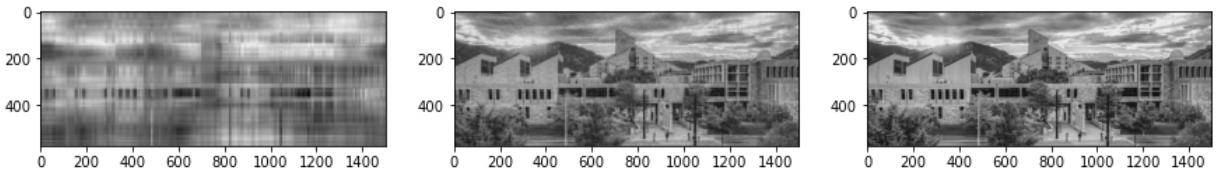
Make the singular value decomposition of the `grayscale` image matrix above. Then use it to reduce the dimension of the problem, and approximate greyscale by only the first k dimensions of the SVD. Then:

- Plot the 3 images corresponding to $k = 5, 50, 200$ in a single figure as side-by-side plots.
- For each such image, print out the exact proportion of the original number of floating point integers required to generate the image. In other words, how many elements **total** of U, Σ and V are you using relative to the original number?

In [227]:
```
U, Sig, VT = np.linalg.svd(grayscale)
Sig = np.diag(Sig)
Sig_full = np.zeros((U.shape[1], VT.shape[0]))
Sig_full[:Sig.shape[0], :Sig.shape[1]] = Sig
```

In [229]:
```
keep = 5
Uf_red = U[:,:keep]
Sigf_red = Sig_full[:keep,:keep]
VTf_red = VT[:keep,:]
Mf_red = np.matmul(np.matmul(Uf_red, Sigf_red), VTf_red)

keep = 50
Uft_red = U[:,:keep]
Sigft_red = Sig_full[:keep,:keep]
VTft_red = VT[:keep,:]
Mft_red = np.matmul(np.matmul(Uft_red, Sigft_red), VTft_red)

keep = 200
Ut_red = U[:,:keep]
Sigt_red = Sig_full[:keep,:keep]
VTt_red = VT[:keep,:]
Mt_red = np.matmul(np.matmul(Ut_red, Sigt_red), VTt_red)

fig,(ax1,ax2,ax3) = plt.subplots(1,3)
fig.set_figwidth(15)
fig.set_figheight(10)
ax1.imshow(Mf_red, cmap='gray')
ax2.imshow(Mft_red, cmap='gray')
ax3.imshow(Mt_red, cmap='gray')
```

Out[229]:
```
<matplotlib.image.AxesImage at 0x17b74e035b0>
```



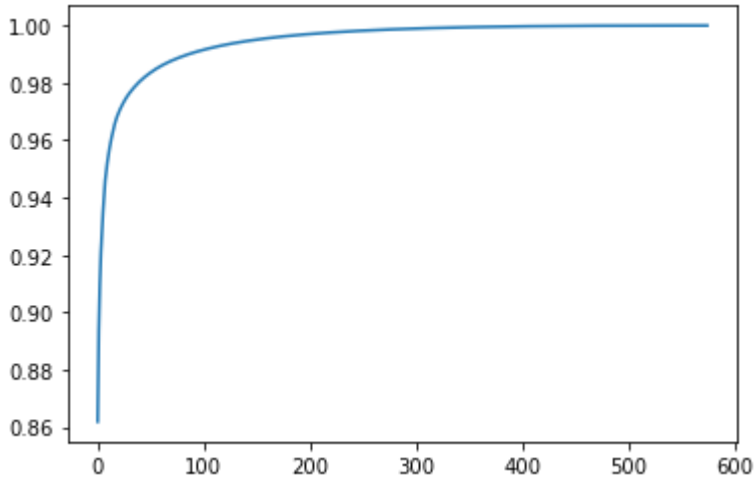
Part B: How many should we use?

In class we discussed the heuristic of using 80-90% of the *energy* of a system to recreate the reconstruction. How does that work for images? Create a list of the cumulative energy retained by using the first $k = 1, 2, 3, \ldots$ dimensions, and plot it.

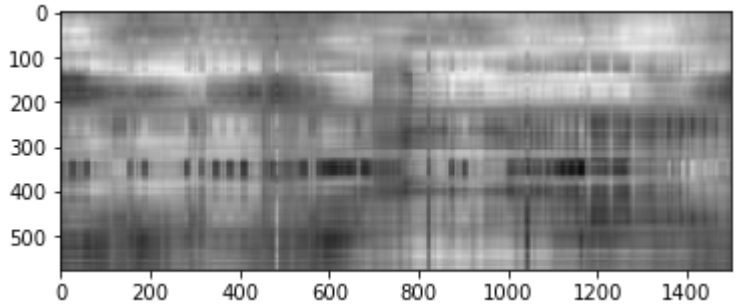
Then, for the first such index where we retain 90% of the energy of the original matrix, plot that image. Does 90% feel like a good baseline for images? If not, state what you think seems to be appropriate for a .png of this size.

In [237]:
```
total_energy = np.sum(Sig**2)
frac_energy_retained = np.zeros(len(Sig))
for k in range(1,len(frac_energy_retained)+1):
    frac_energy_retained[k-1] = np.sum(Sig[:k,:k]**2)/total_energy

plt.plot(frac_energy_retained)
```

Out[237]:
```
[<matplotlib.lines.Line2D at 0x17b79a23430>]
```



In [236]:
```
for i in range(len(frac_energy_retained)):
    if(frac_energy_retained[i] > .90):
        U_red = U[:,:i]
        Sig_red = Sig_full[:i,:i]
        VT_red = VT[:i,:]
        M_red = np.matmul(np.matmul(U_red, Sig_red), VT_red)
        plt.imshow(Mf_red, cmap='gray')
        break
```



Not it doesn't look good at all, somewhere around k=50 looks a lot better

It says we should use indices 0-2, for a total of 3 columns... but that image looks terrible! It's probably more like 98-99%, at least plotting things as small as they are on this notebook. That's

still pretty good savings!

Part C: Extra Credit (5 pts)

Our image was in color! Repeat the analysis in parts A/B to create a reduced dimenstional representation of **all 3** of the R, G, and B color values for each node. Start with the `rgb` object given, and perform 3 separate SVD decompositions for the 3 colors, discarding transparancy. At the end, create a 2x2 plot of images for using 60 singular values, including the 3 reconstructed R, G, and B matrix images and the fully colorized.

Play around with a few other values of k besides 60. Does it feel like you need more or less dimensions to be able to interpret SVDs of RBG data? Explain.

In [ ]:

Back to top

Problem 3 (Outlier Detection; 15 pts)

The file `nba2022.csv` includes all of the major basketball statistics from the 2021-2022 NBA season. We're going to use it to determine whether "outlierness" of players correlated with their value to the team. Load in the file and inspect it. A glossary for the columns is as follows below. We're going to use this set to do some outlier detection to explore how well we can extract some of the

For all future analysis, you should only use NBA players that played at least 1000 minutes `MIN` (around 1/4 of the seasons' possible total).

- GP Games Played
- W Wins
- L Losses
- MIN Minutes Played
- FGM Field Goals Made
- FGA Field Goals Attempted
- FG% Field Goal Percentage
- 3PM 3 Point Field Goals Made
- 3PA 3 Point Field Goals Attempted
- 3P% 3 Point Field Goals Percentage
- FTM Free Throws Made
- FTA Free Throws Attempted
- FT% Free Throw Percentage
- OREB Offensive Rebounds
- DREB Defensive Rebounds
- REB Rebounds
- AST Assists
- TOV Turnovers

- STL Steals
- BLK Blocks
- PF Personal Fouls
- FP Fantasy Points
- DD2 Double doubles
- TD3 Triple doubles
- PTS Points
- +/- Plus Minus

In [238]:
```
df=pd.read_csv('./Data/nba2022.csv')
df.head(8)
```

Out[238]:

	PLAYER	TEAM	AGE	GP	W	L	MIN	PTS	FGM	FGA	...	DREB	REB	AST	TOV	STL
0	Mikal Bridges	PHX	25	82	64	18	2854	1162	458	858	...	273	347	185	68	96
1	Miles Bridges	CHA	24	80	41	39	2837	1613	596	1214	...	469	559	300	150	74
2	DeMar DeRozan	CHI	32	76	43	33	2743	2118	774	1535	...	336	392	374	181	68
3	Jayson Tatum	BOS	24	76	49	27	2731	2046	708	1564	...	524	609	334	217	75
4	Saddiq Bey	DET	23	82	23	59	2704	1321	450	1136	...	334	441	233	96	73
5	Tyrese Haliburton	IND	22	77	26	51	2694	1181	430	909	...	250	311	628	199	134
6	Russell Westbrook	LAL	33	78	31	47	2678	1441	548	1233	...	470	580	550	295	75
7	Trae Young	ATL	23	76	40	36	2652	2155	711	1544	...	234	284	737	303	72

8 rows × 28 columns

In [240]:
```
df = df.loc[df["MIN"]>1000]
```

Part A: Outliers: Classical

Bi) In the older days of NBA statistics, the most common measure of a players' value were in the triple (Points per game, Rebounds per game, Assists per game). Create new columns in your data frame that house these 3 values. Use `GP` for games, not the theroetical max of 82.

Bii) Use your new columns in an $n \times 3$ object and calculate the **Mahalanobis' distance** for each player. Who are the largest 10 outliers?

Biii) Use the `decision_function` method from a fit of SKLearn's `IsolationForest` library in the import statement above. Who are the largest 10 outliers under an Isolation Forest?

In [283]:
```
df["PPG"] = df["PTS"]/df["GP"]
df["RPG"] = df["REB"]/df["GP"]
df["APG"] = df["AST"]/df["GP"]
malT = pd.DataFrame(np.transpose([df["PPG"],df["RPG"],df["APG"]]))
malT
```

Out[283]:

	0	1	2
0	14.170732	4.231707	2.256098
1	20.162500	6.987500	3.750000
2	27.868421	5.157895	4.921053
3	26.921053	8.013158	4.394737
4	16.109756	5.378049	2.841463
...	...	...	...
267	6.079365	2.603175	0.698413
268	10.142857	2.142857	0.979592
269	9.047619	4.047619	2.809524
270	5.081633	1.877551	0.734694
271	6.225352	1.887324	1.957746

272 rows × 3 columns

In [299]:
```
mT = sorted(enumerate(forest.decision_function(malT)), key=lambda i: i[1])
for i in range(10):
    print(df.iloc[mT[i][0]]["PLAYER"])
```

```
Nikola Jokic
Trae Young
Rudy Gobert
Joel Embiid
Luka Doncic
Chris Paul
Giannis Antetokounmpo
James Harden
Dejounte Murray
Domantas Sabonis
```

Part B Outliers: Rate

Ci) These days, we often get better date that measures how effective or ineffective a player is *when they're on the court*. These are known as *rate* statistics, and require us create "per minute" statistics for each player. Create new columns that are (Points per 36 minutes, Rebounds per 36 minutes, Assists per 36 minutes) for each player.

Cii) Use your new columns in an $n \times 3$ object and calculate the **Mahalanobis' distance** for each player. Who are the largest 10 outliers?

Ciii) Use the `decision_function` method from a fit of SKLearn's `IsolationForest` library in the import statement above. Who are the largest 10 outliers under an Isolation Forest?

In [300]:
```
df["PR"] = df["PTS"]/(df["MIN"]/36)
df["RR"] = df["REB"]/(df["MIN"]/36)
df["AR"] = df["AST"]/(df["MIN"]/36)
malR = pd.DataFrame(np.transpose([df["PR"],df["RR"],df["AR"]]))
malR
```

Out[300]:

	0	1	2
0	14.657323	4.377015	2.333567
1	20.468100	7.093409	3.806838
2	27.797302	5.144732	4.908494
3	26.970341	8.027829	4.402783
4	17.587278	5.871302	3.102071
...	...	...	...
267	13.584236	5.816749	1.560591
268	17.679842	3.735178	1.707510
269	13.611940	6.089552	4.226866
270	8.946108	3.305389	1.293413
271	15.896104	4.819181	4.999001

272 rows × 3 columns

In [302]:
```
mR = sorted(enumerate(forest.decision_function(malR)), key=lambda i: i[1])
for i in range(10):
    print(df.iloc[mR[i][0]]["PLAYER"])
```

```
Nikola Jokic
JaVale McGee
Hassan Whiteside
Giannis Antetokounmpo
Trae Young
Joel Embiid
Rudy Gobert
Clint Capela
Steven Adams
Andre Drummond
```

Part C: Outliers: Which is better?

For whether or not our methods are extracting players that are of value to the team, consider some other measures.

The League votes on the single "Most Valuable Player" in the coming days. The nominal favorites are Nikola Jokic, Joel Embiid, and Giannis Antetokounmpo. How do they appear on your lists?

Consider the top 20 by "win shares", an "advanced" stat listed here (https://www.basketball-reference.com/leagues/NBA_2022_advanced.html#advanced_stats::ws).

Modern NBA thinking suggests that rate statistics are more useful than simple aggregates. Do we seem to get a better top-10 list from rate statistics? Are any of your outliers coming being outliers in possibly **not** valuable ways?

Jokic appears first in both lists, Joel Embiid appears 4th in the triple list and 6th in the rate list, Giannis appears 7th in triple rate and 4th in the rate list.

Jokic, Embiid, and Giannis appear first, second, and third in the advanced stat list.

Not necissarily.v Yes in the rate list Jabale McGee is second but is far lower on the advanced list (76th)