ECE 475 Project 6: Market Basket Analysis

Tiffany Yu, Jonathan Lam, Harris Paspuleti

The goal of this project is to use market basket analysis to draw interesting inferences (association rules) about some dataset.

In [ ]:

```
# mlxtend has a priori algorithm implementation
!pip install mlxtend

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

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

Dataset

[FIFA 18 Player Statistics](#)

This dataset contains the attributes for every player that is registered in the latest edition of FIFA 18 database; it can be used for soccer/videogame analysis as it contains attributes such as skill moves, overall, potential, position, etc. While none of us know much about soccer, we wanted to see if there were any interesting assocations we could learn from the data even without knowing much about the sport.

> ### Content
>
> - **Every player featuring in FIFA 18**
> - **70+ attributes**
> - **Player and Flag Images**
> - **Playing Position Data**
> - **Attributes based on actual data of the latest EA's FIFA 18 game**
> - **Attributes include on all player style statistics like Dribbling, Aggression, - GK Skills etc.**
> - **Player personal data like Nationality, Photo, Club, Age, Wage, Salary etc.**
>
> ### Data Source
>
> **The data is scraped from the website https://sofifa.com by extracting the Player personal data and Player Ids and then the playing and style statistics.**

In [ ]:

```
# grab the data
raw_data = 'https://raw.githubusercontent.com/4m4n5/fifa18-all-player-statistics/master/2019/data.csv'
dataframe = pd.read_csv(raw_data, sep=',', header='infer', error_bad_lines=False)
```

Preprocessing

Preprocessing Step 1: Drop unusable columns and rows

Some columns were extraneous, included data that we didn't know how to interpret, or included data that was unique to a player. For example, there was a column for the player's index, the player photo, and some fields like "Real Face" that we weren't familiar with. Additionally, there were many rows with acronyms (e.g., "LS", "ST", etc.) with values that we weren't sure how to interpret.

Some rows also had missing data; we dropped this using `pd::dropna()`.

The dataframe after this initial preprocessing step is shown below.

In [ ]:

```
# preprocessing the data by removing unnecessary columns
dataframe = dataframe.drop(columns=[
    'Unnamed: 0','ID','Photo','Flag','Club Logo','Loaned From','Real Face','LS',
    'ST','RS','LW','LF','CF','RF','RW','LAM','CAM','RAM','LM','LCM','CM','RCM',
    'RM','LWB','LDM','LB','LCB','CB','RCB','RB','CDM','RDM','RWB'
    ]).dropna()
dataframe
```

Out[ ]:

	Name	Age	Nationality	Overall	Potential	Club	Value	Wage	Special	Preferred Foot	International Reputation	We Fo
0	L. Messi	31	Argentina	94	94	FC Barcelona	€110.5M	€565K	2202	Left	5.0	
1	Cristiano Ronaldo	33	Portugal	94	94	Juventus	€77M	€405K	2228	Right	5.0	
2	Neymar Jr	26	Brazil	92	93	Paris Saint-Germain	€118.5M	€290K	2143	Right	5.0	
3	De Gea	27	Spain	91	93	Manchester United	€72M	€260K	1471	Right	4.0	
4	K. De Bruyne	27	Belgium	91	92	Manchester City	€102M	€355K	2281	Right	4.0	
...	...	...	...	...	...	...	...	...	...	...	...	
18202	J. Lundstram	19	England	47	65	Crewe Alexandra	€60K	€1K	1307	Right	1.0	
18203	N. Christoffersson	19	Sweden	47	63	Trelleborgs FF	€60K	€1K	1098	Right	1.0	
18204	B. Worman	16	England	47	67	Cambridge United	€60K	€1K	1189	Right	1.0	
18205	D. Walker-Rice	17	England	47	66	Tranmere Rovers	€60K	€1K	1228	Right	1.0	
18206	G. Nugent	16	England	46	66	Tranmere Rovers	€60K	€1K	1321	Right	1.0	

16643 rows × 56 columns

Preprocessing Step 2: Determining bins

Before binning features, we plotted histograms of some of the quantitative fields so that we could see what the distributions look like.

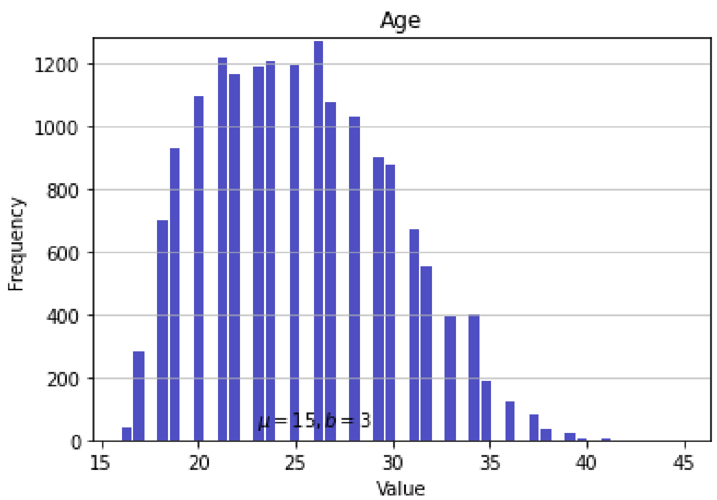
Since this is a videogame, much of the numerical ratings were presented as numbers out of 100, so we decided that binning most fields into quartiles was good enough for our purposes. (Using larger bins was also problematic because of the amount of time it took to run the algorithm with more items.)

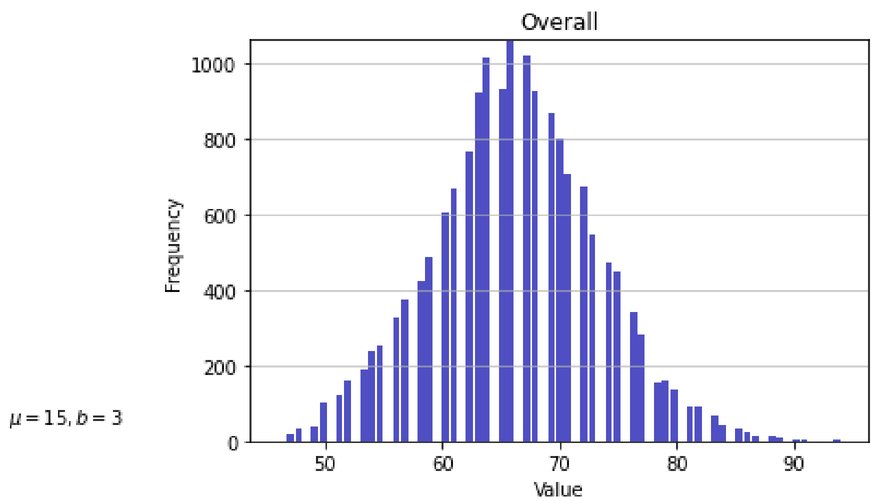
In [ ]:

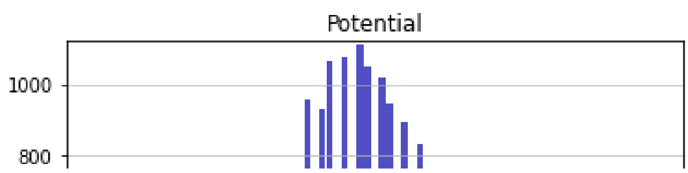
In [ ]:

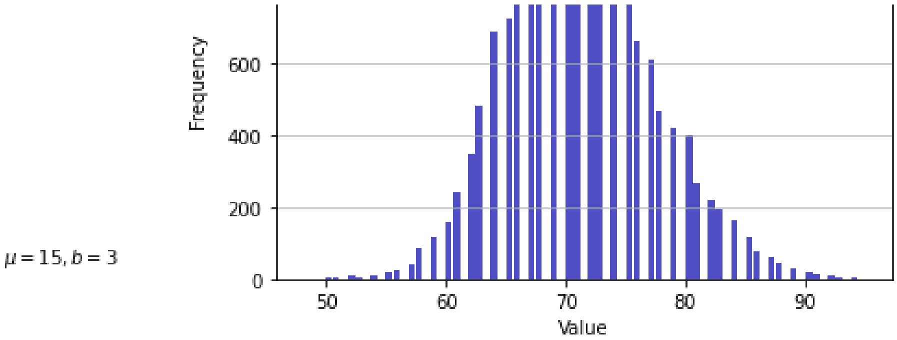
```
# list of numerical fields that can be binned
to_bin = ['Age', 'Overall', 'Potential', 'Value', 'Wage', 'Release Clause',
    'Jersey Number', 'FKAccuracy', 'LongPassing', 'BallControl', 'Acceleration',
    'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower', 'Jumping',
    'Stamina', 'Strength', 'LongShots', 'Aggression', 'Interceptions',
    'Positioning', 'Vision', 'Penalties', 'Composure', 'Marking',
    'StandingTackle', 'SlidingTackle', 'GKDiving', 'GKHandling', 'GKKicking',
    'GKPositioning', 'GKReflexes'
]

# histograms; see: https://realpython.com/python-histograms/
for col in to_bin:
    # An "interface" to matplotlib.axes.Axes.hist() method
    n, bins, patches = plt.hist(x=dataframe.loc[:, col], bins='auto', color='#0504aa',
                                alpha=0.7, rwidth=0.85)
    plt.title(col)
    plt.grid(axis='y', alpha=0.75)
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.text(23, 45, r'$\mu=15, b=3$')
    maxfreq = n.max()
    # Set a clean upper y-axis limit.
    plt.ylim(ymax=np.ceil(maxfreq / 10) * 10 if maxfreq % 10 else maxfreq + 10)
    plt.figure()
```

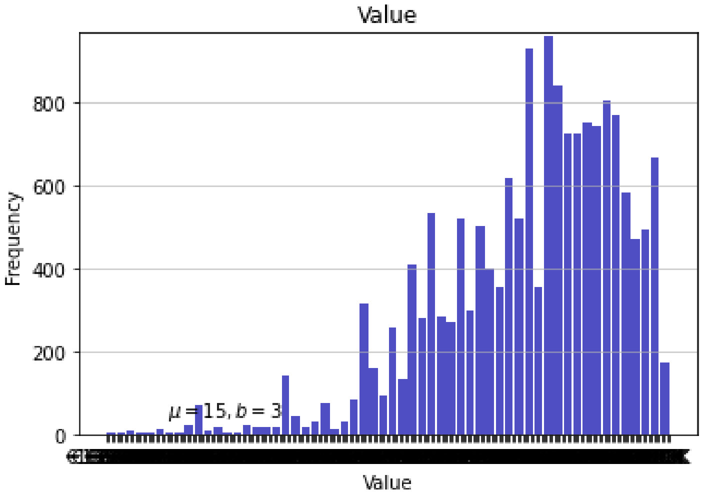
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: RuntimeWarning: More tha
n 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.
pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To
control this warning, see the rcParam `figure.max_open_warning`).
  from ipykernel import kernelapp as app
```

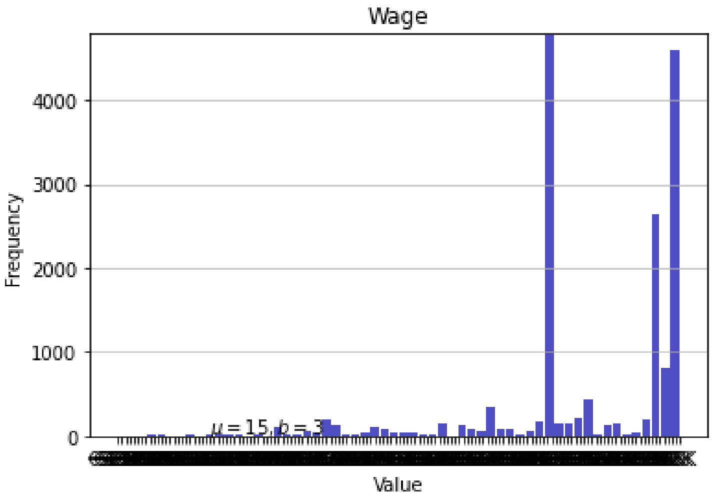


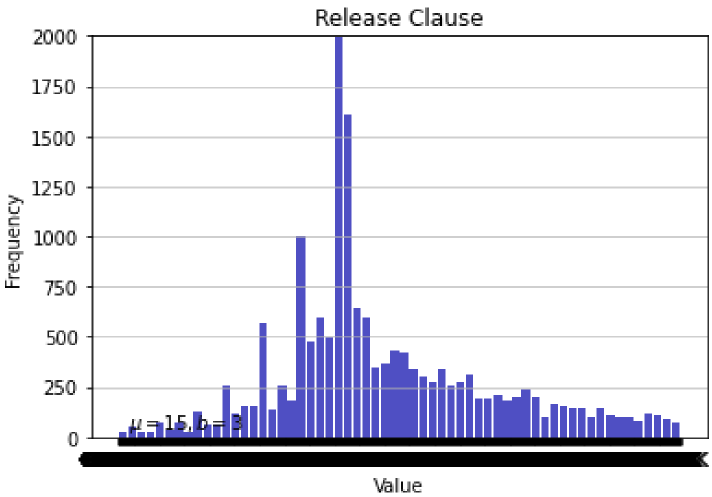


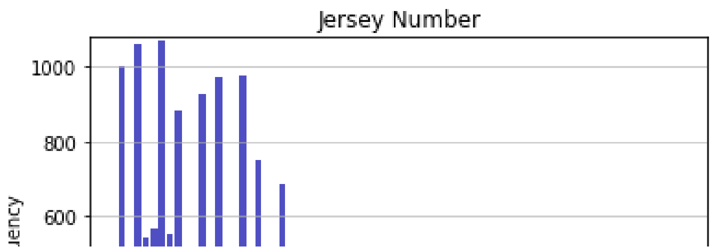


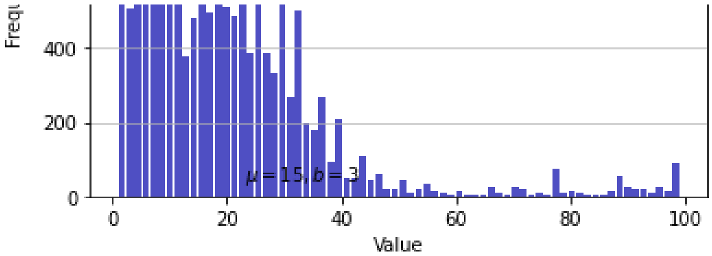

Frequency
600
400
200
0
50
60
70
80
90
Value
$\mu = 15, b = 3$


Value
800
600
400
200
0
Frequency
Value
$\mu = 15, b = 3$

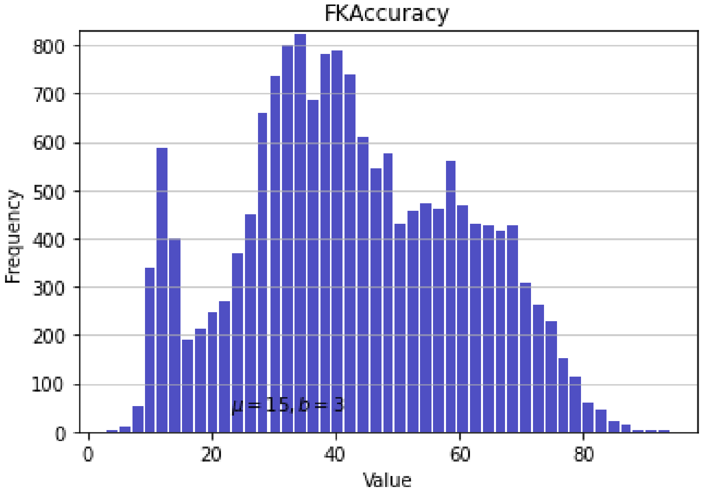

Wage
4000
3000
2000
1000
0
Frequency
Value
$\mu = 15, b = 3$


Release Clause
2000
1750
1500
1250
1000
750
500
250
0
Frequency
Value
$\mu = 15, b = 3$

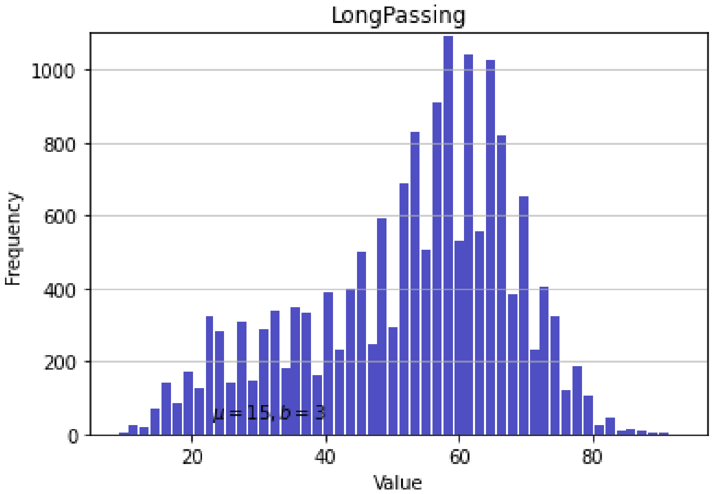

Jersey Number
1000
800
600



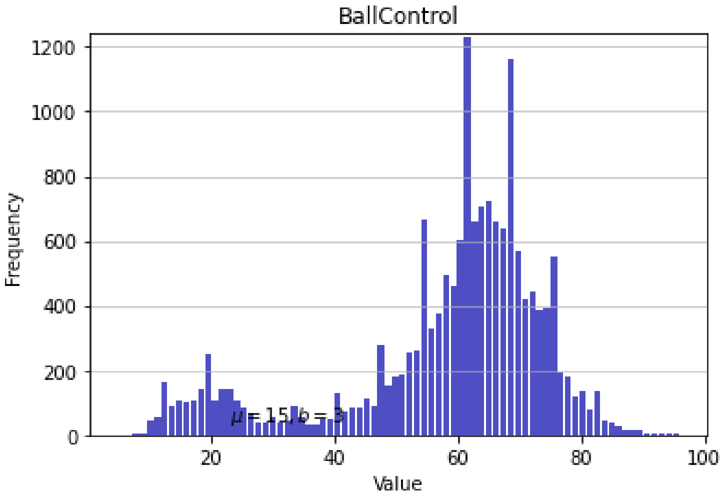
Freq
400
200
0
0
20
40
60
80
100
Value
μ = 15, b = 3



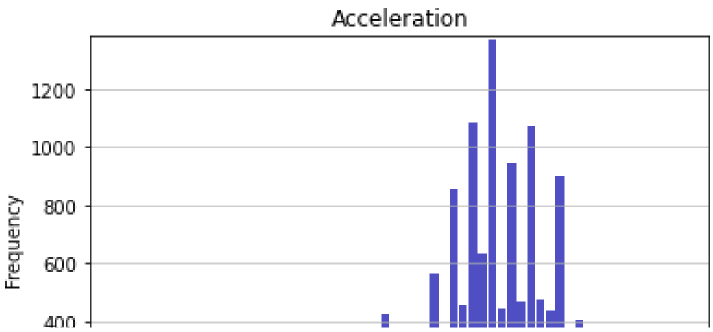
FKAccuracy
Frequency
800
700
600
500
400
300
200
100
0
0
20
40
60
80
Value
μ = 15, b = 3



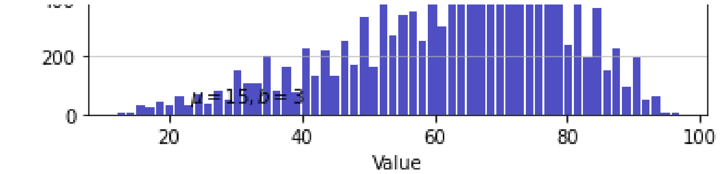
LongPassing
Frequency
1000
800
600
400
200
0
20
40
60
80
Value
μ = 15, b = 3

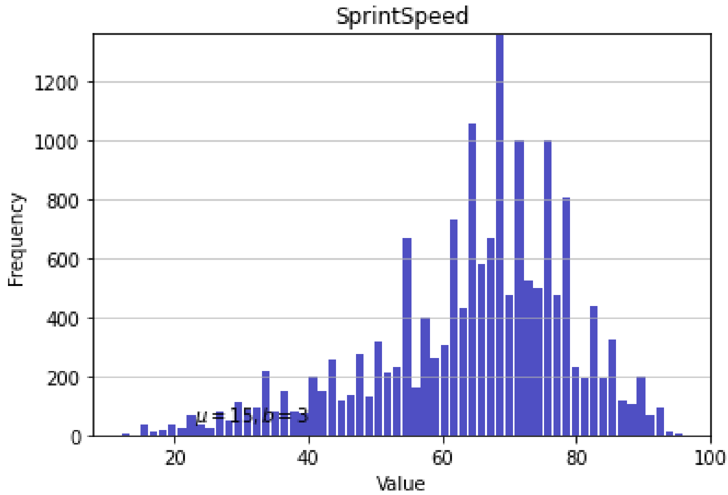


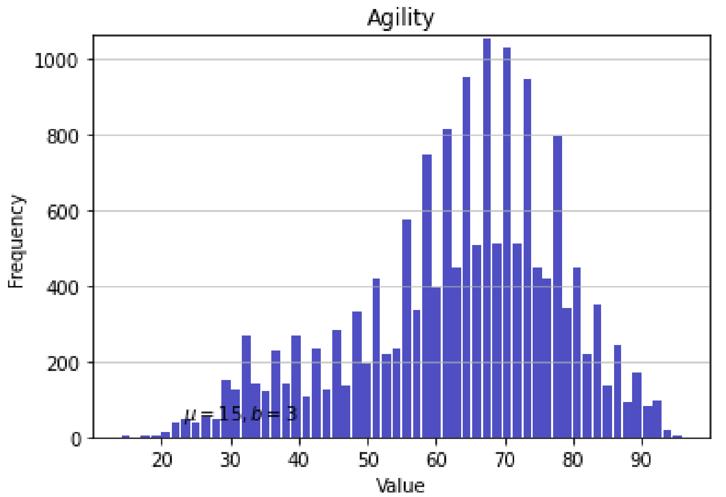
BallControl
Frequency
1200
1000
800
600
400
200
0
20
40
60
80
100
Value
μ = 15, b = 3

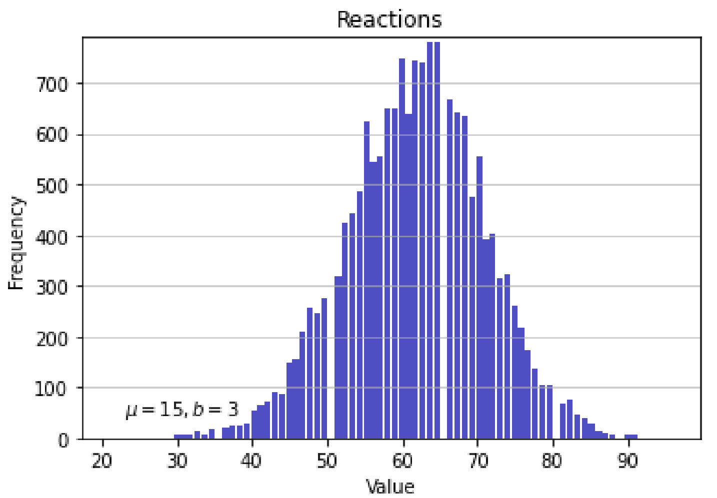


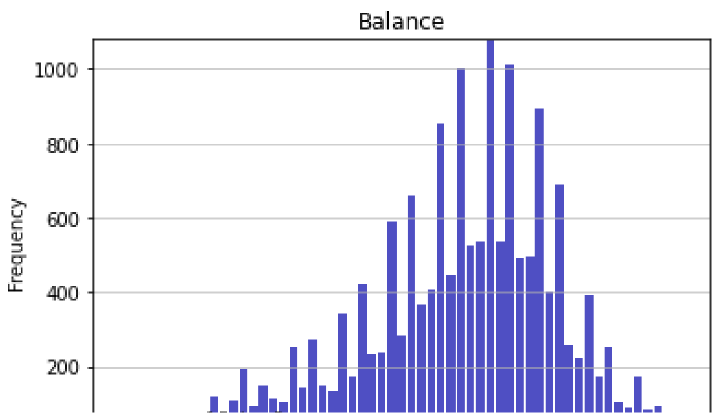
Acceleration
Frequency
1200
1000
800
600
400

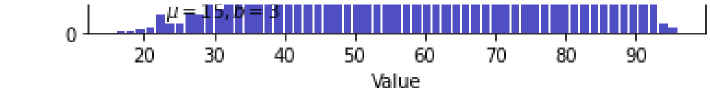

μ = 15, b = 3
Value

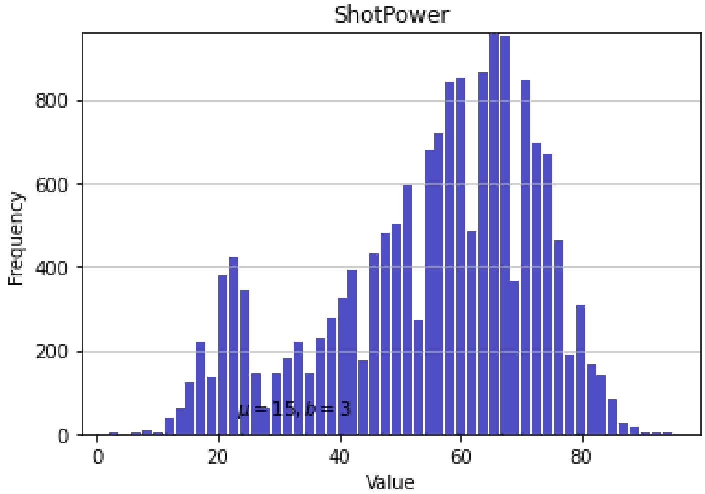

SprintSpeed
Frequency
μ = 15, b = 3
Value

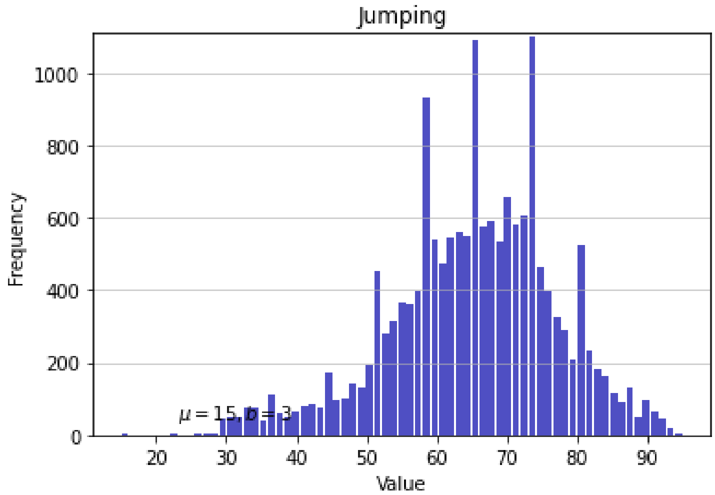

Agility
Frequency
μ = 15, b = 3
Value

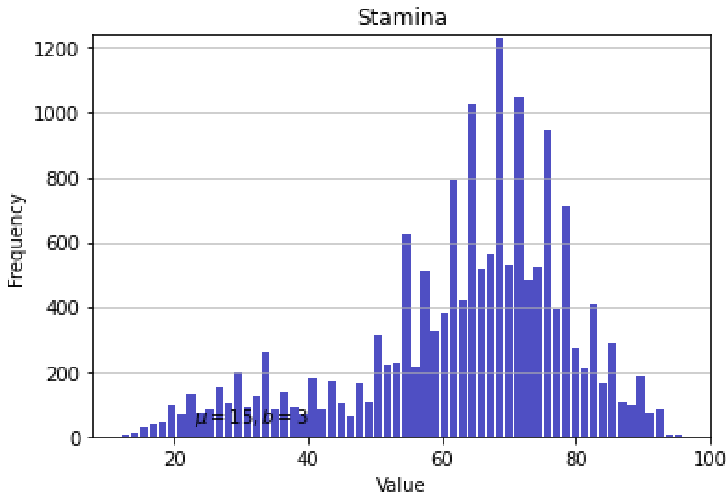

Reactions
Frequency
μ = 15, b = 3
Value

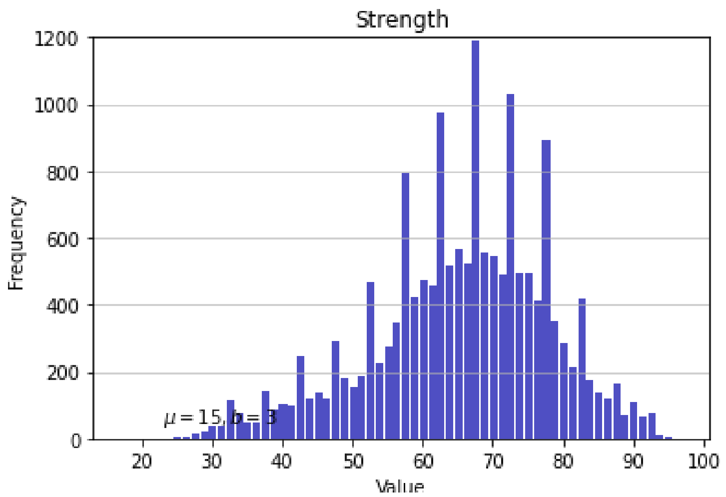

Balance
Frequency

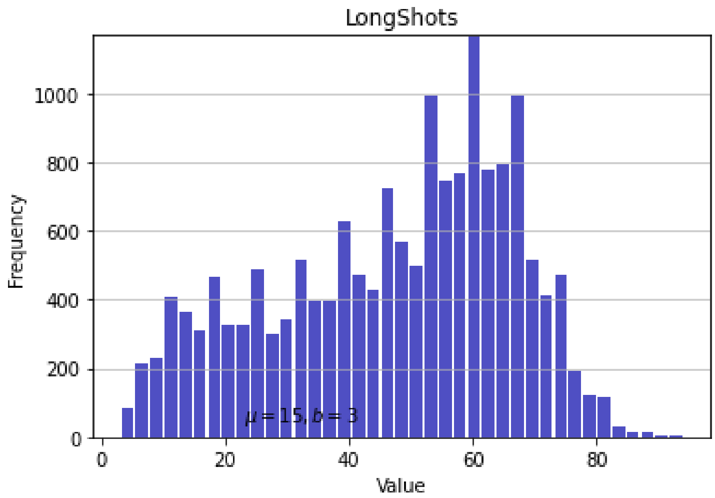

μ = 15, b = 3
0
20
30
40
50
60
70
80
90
Value

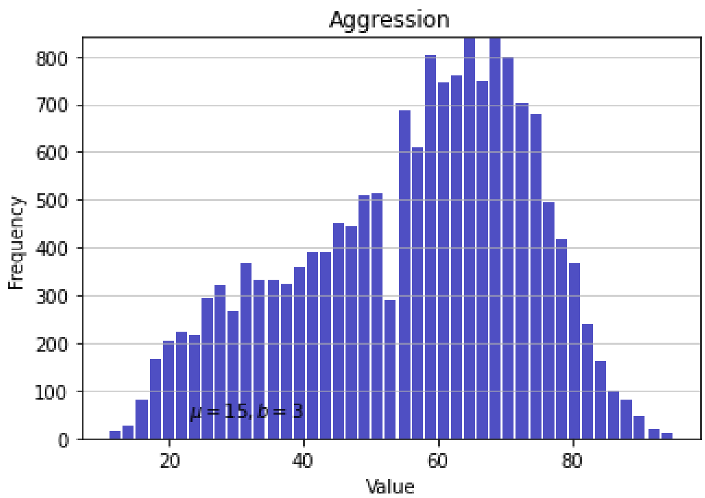

ShotPower
Frequency
0
200
400
600
800
μ = 15, b = 3
0
20
40
60
80
Value

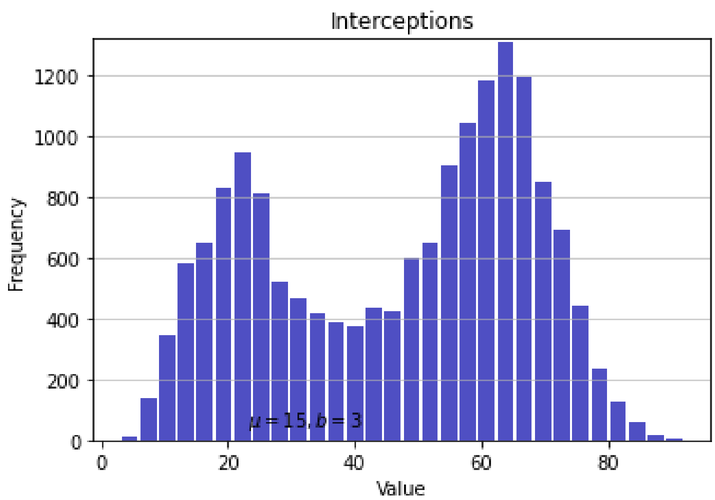

Jumping
Frequency
0
200
400
600
800
1000
μ = 15, b = 3
20
30
40
50
60
70
80
90
Value

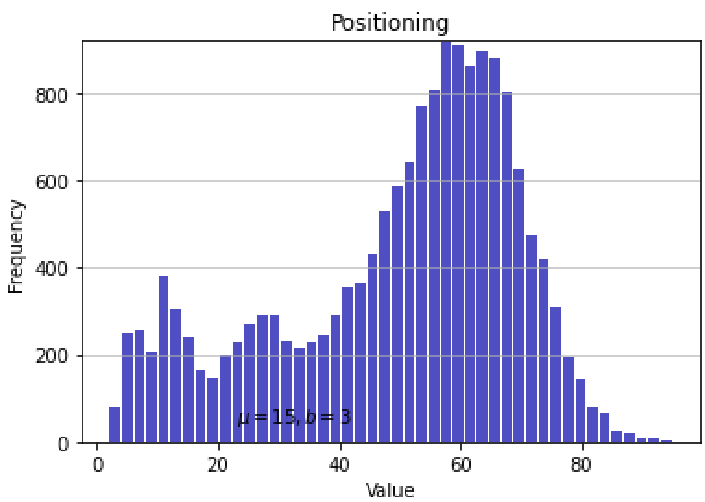

Stamina
Frequency
0
200
400
600
800
1000
1200
μ = 15, b = 3
20
40
60
80
100
Value


Strength
Frequency
0
200
400
600
800
1000
1200
μ = 15, b = 3
20
30
40
50
60
70
80
90
100
Value

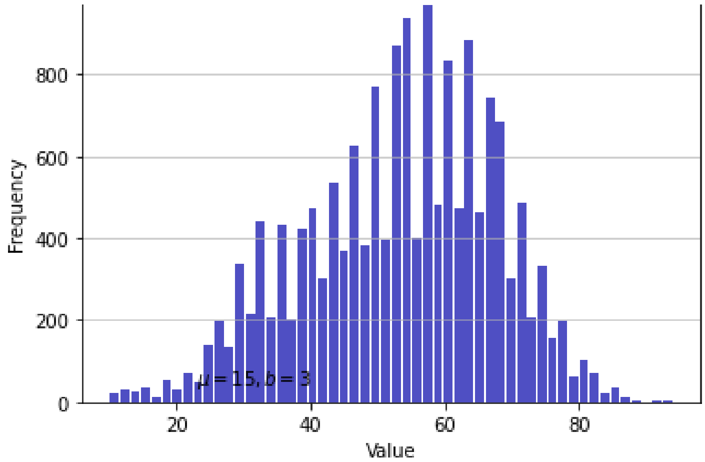

LongShots
Frequency
Value
$\mu = 15, b = 3$


Aggression
Frequency
Value
$\mu = 15, b = 3$

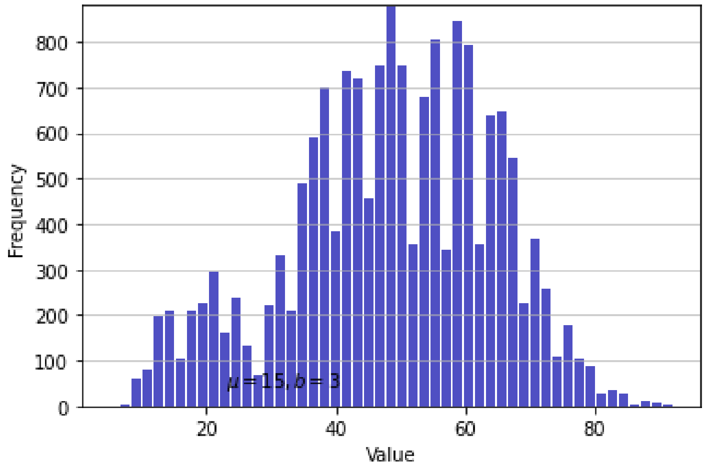

Interceptions
Frequency
Value
$\mu = 15, b = 3$


Positioning
Frequency
Value
$\mu = 15, b = 3$

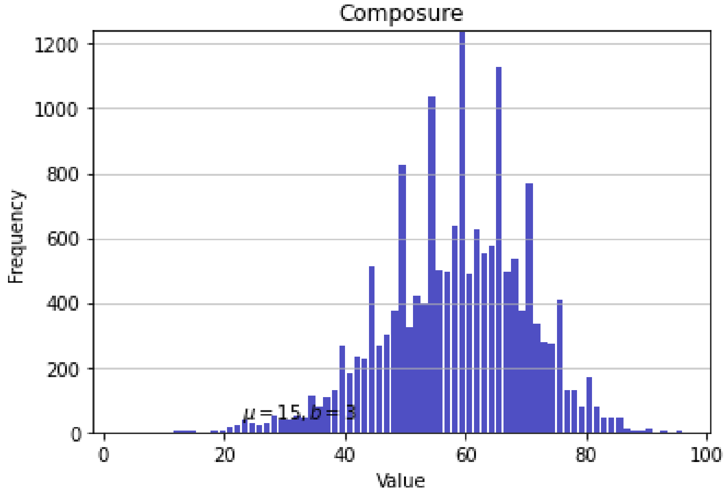

Vision



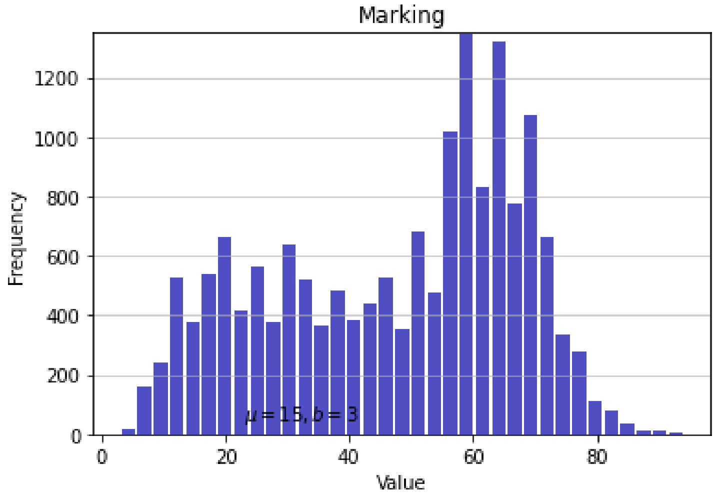
Penalties



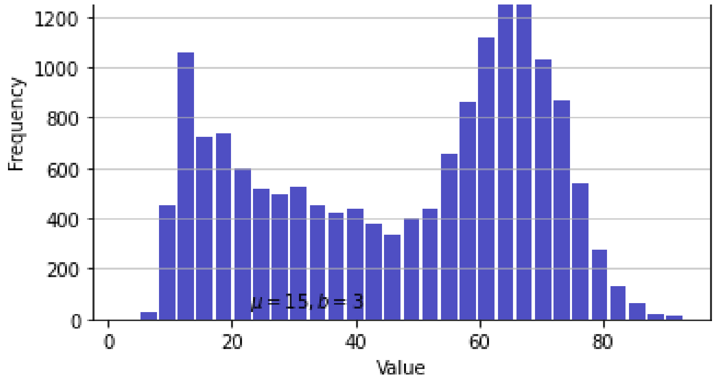
Composure

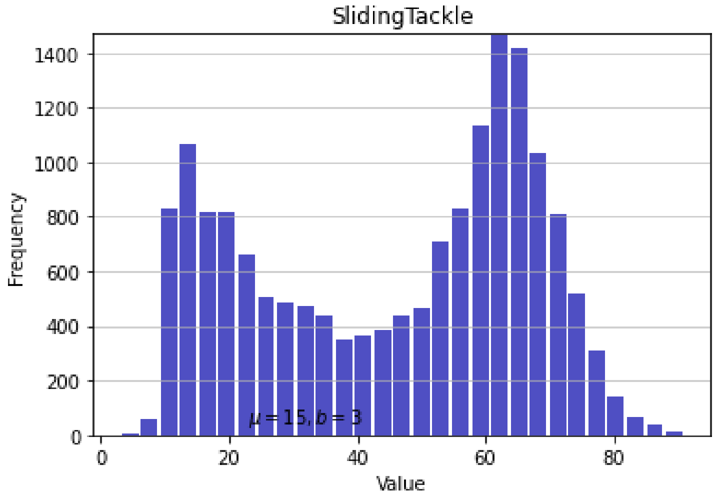


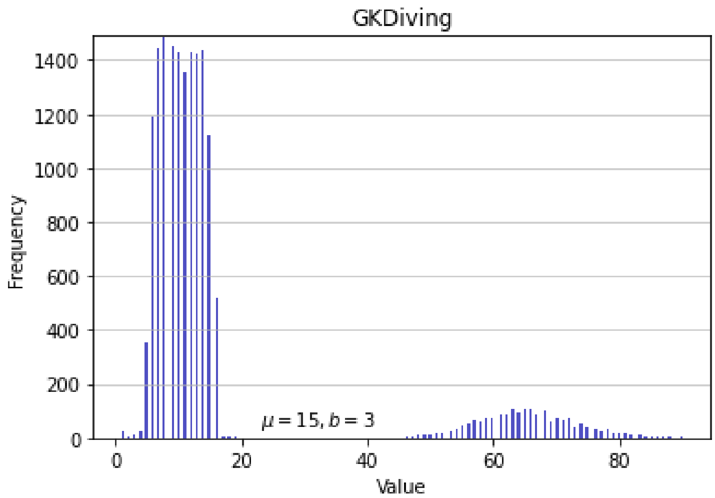
Marking

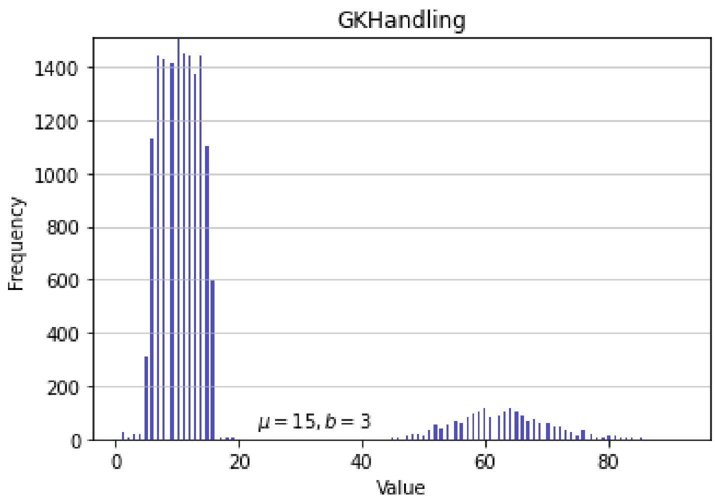


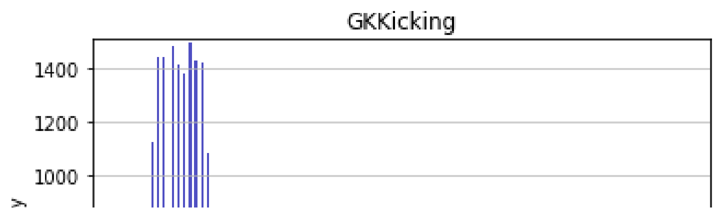
StandingTackle

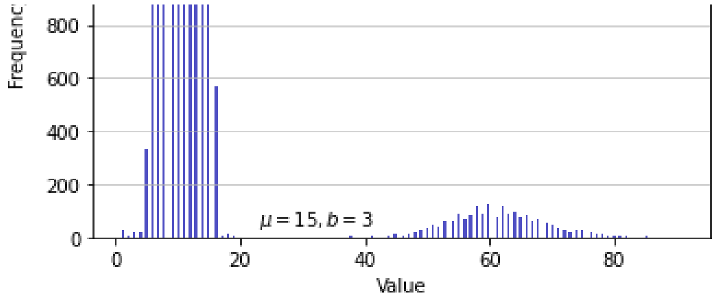

Frequency
Value
μ = 15, b = 3

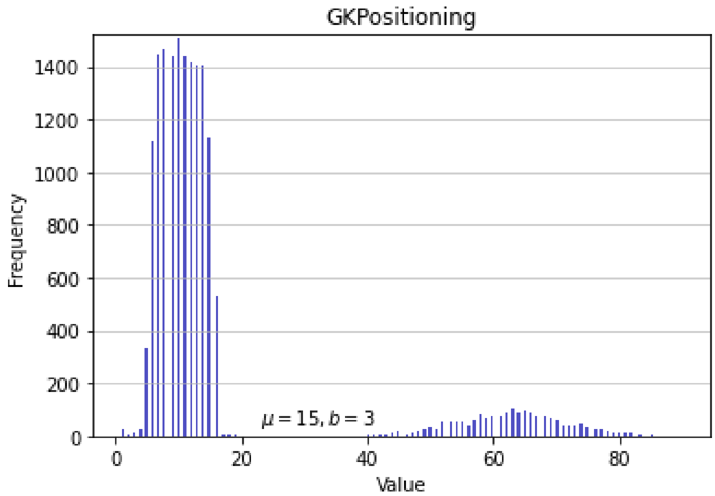

SlidingTackle
Frequency
Value
μ = 15, b = 3

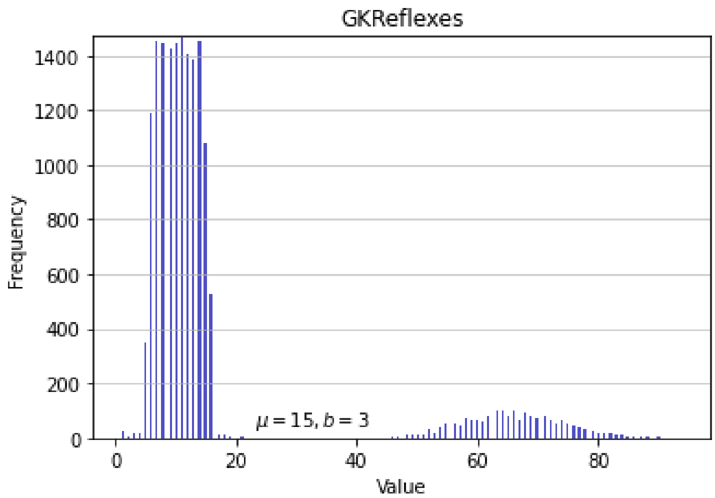

GKDiving
Frequency
Value
μ = 15, b = 3


GKHandling
Frequency
Value
μ = 15, b = 3


GKKicking








```
<Figure size 432x288 with 0 Axes>
```

Preprocessing Step 3: Extracting numbers and binning

Some of the numerical fields were formatted, and the numerical value had to be extracted first before binning. For example, money values were written with a euro sign prefix and the magnitude was indicated with a "M" or "K" postfix (indicating millions or thousands).

We decided to bin the players' ages into bins of age 10, and monetary values (e.g., "Wage," "Release Clause," "Value") into logarithmic bins. Most of the player stats are values between 0 and 100, and we decided to bin these into quartiles (larger bins would be noninformative, and smaller bins would make the algorithm take too long).

The dataframe of the binned values is shown below the code.

In [ ]:

```
# binning quantitative values
def bin(col, bins, col_name):
    new_col = col.copy()

    for i, bin_min in enumerate(bins):
        bin_max = float('Inf') if i==len(bins)-1 else bins[i+1]
        new_col[(col >= bin_min) & (col < bin_max)] = f'{bin_min}<={col_name}<{bin_max}'
```

```
    return new_col

# make a money column a number
def money_to_number(df_col):
    col = df_col.copy()

    for i, val in enumerate(col):
        if val[-1] == 'M':
            val = int(float(val[1:-1]) * 1000000)
        elif val[-1] == 'K':
            val = int(float(val[1:-1]) * 1000)
        else:
            val = int(float(val[1:]))
        col.iloc[i] = val

    return col.astype('int32')

# create a copy so we don't modify the original dataframe in place
dataframe_binned = dataframe.copy()

dataframe_binned.loc[:, 'Age'] = bin(dataframe.loc[:, 'Age'],
                                     [0, 10, 20, 30, 40, 50], 'Age')
dataframe_binned.loc[:, 'Overall'] = bin(dataframe.loc[:, 'Overall'],
                                         [0, 25, 50, 75, 100], 'Overall')
dataframe_binned.loc[:, 'Potential'] = bin(dataframe.loc[:, 'Potential'],
                                           [25, 50, 75, 100], 'Potential')
dataframe_binned.loc[:, 'Value'] = bin(money_to_number(dataframe.loc[:, 'Value']),
                                       [0, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9], 'Value')
dataframe_binned.loc[:, 'Wage'] = bin(money_to_number(dataframe.loc[:, 'Wage']),
                                      [0, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9], 'Wage')
dataframe_binned.loc[:, 'Release Clause'] = bin(money_to_number(dataframe.loc[:, 'Release
Clause']),
                                                [0, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9],
'Release Clause')

to_bin = [
    'Jersey Number', 'FKAccuracy', 'LongPassing', 'BallControl',
    'Acceleration', 'SprintSpeed', 'Agility', 'Reactions', 'Balance',
    'ShotPower', 'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression',
    'Interceptions', 'Positioning', 'Vision', 'Penalties', 'Composure',
    'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving', 'GKHandling',
    'GKKicking', 'GKPositioning', 'GKReflexes']

for col in to_bin:
    dataframe_binned.loc[:, col] = bin(dataframe.loc[:, col], [0, 25, 50, 75, 100], col)
dataframe_binned
```

Out[ ]:

	Name	Age	Nationality	Overall	Potential	Club	
0	L. Messi	30<=Age<40	Argentina	75<=Overall<100	75<=Potential<100	FC Barcelona	100000000.0<=Value<100000
1	Cristiano Ronaldo	30<=Age<40	Portugal	75<=Overall<100	75<=Potential<100	Juventus	10000000.0<=Value<10000
2	Neymar Jr	20<=Age<30	Brazil	75<=Overall<100	75<=Potential<100	Paris Saint-Germain	100000000.0<=Value<100000
3	De Gea	20<=Age<30	Spain	75<=Overall<100	75<=Potential<100	Manchester United	10000000.0<=Value<10000
4	K. De Bruyne	20<=Age<30	Belgium	75<=Overall<100	75<=Potential<100	Manchester City	100000000.0<=Value<100000
...	...	...	...	...	...	...	
18202	J. Lundstram	10<=Age<20	England	25<=Overall<50	50<=Potential<75	Crewe Alexandra	10000.0<=Value<10

	Name	Age	Nationality	Overall	Potential	Club	
18203	N. Christoffersson	10<=Age<20	Sweden	25<=Overall<50	50<=Potential<75	Trelleborgs FF	10000.0<=Value<10
18204	B. Worman	10<=Age<20	England	25<=Overall<50	50<=Potential<75	Cambridge United	10000.0<=Value<10
18205	D. Walker-Rice	10<=Age<20	England	25<=Overall<50	50<=Potential<75	Tranmere Rovers	10000.0<=Value<10
18206	G. Nugent	10<=Age<20	England	25<=Overall<50	50<=Potential<75	Tranmere Rovers	10000.0<=Value<10

16643 rows × 56 columns

Preprocessing Step 4: Choosing features

It is clear that the table becomes very wide at this point. If we included all of the features from the previous section (categorical and binned quantitative features), the a priori algorithm took way too long to run. We experimented with a few different combinations of which features to include. The following is one possible combination of features to perform an analysis on (an explanation of this choice will follow in a later section).

In [ ]:

```
# didn't include most of the rows in the analysis, because the a priori
# algorithm takes too long
cols = [
    'Preferred Foot', 'Age', 'Aggression', 'Nationality',
    'Body Type', 'Reactions','Position', 'Balance',
    'Contract Valid Until', 'Jersey Number', 'Penalties', 'Vision',
    # 'SprintSpeed', 'Agility', 'Reactions', 'ShotPower', 'Jumping', 'Stamina',
    # 'Strength', 'LongShots', 'Aggression','Composure', 'Agility',
    # 'Interceptions', 'Positioning' , 'Composure', 'Marking', 'StandingTackle',
    # 'SlidingTackle', 'GKDiving','International Reputation','Body Type',
    # 'GKHandling', 'GKKicking', 'GKPositioning', 'GKReflexes','Overall',
    # 'Value', 'Wage', 'FKAccuracy', 'LongPassing', 'BallControl', 'Aceleration'
]
```

Preprocessing Step 5: One-hot encoding items

Now that all of the data is binned, it is one-hot encoded. This is the necessary data input format for the a priori algorithm.

In [ ]:

```
# format data for the apriori algorithm
def one_hot_encode_column(df, col):
  items = np.unique(np.array(df.loc[:,col]))
  items_onehot = df.loc[:,col][:, np.newaxis] == items[np.newaxis, :]
  return  pd.DataFrame(columns=[col+' '+str(item) for item in items],
                       data=items_onehot,
                       dtype=np.int32)

basket_sets = pd.concat([
    one_hot_encode_column(dataframe_binned, col) for col in cols
], axis=1)
basket_sets
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarning: Support fo
r multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a
future version.  Convert to a numpy array before indexing instead.
  after removing the cwd from sys.path.
```

Out[ ]:

Preferred	Preferred Foot	Age	Age	Age	Age	Aggression	Aggression

	Foot Left Preferred	Foot Right Preferred Foot	10<=Age<20 Age	20<=Age<30 Age	30<=Age<40 Age	40<=Age<50 Age	0<=Aggression<25 Aggression	25<=Aggression<50 Aggression	50<
0	1	0	0	0	1	0	0	1	
1	0	1	0	0	1	0	0	0	
2	0	1	0	1	0	0	0	0	
3	0	1	0	1	0	0	0	1	
4	0	1	0	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	
16638	0	1	1	0	0	0	0	1	
16639	0	1	1	0	0	0	0	1	
16640	0	1	1	0	0	0	0	1	
16641	0	1	1	0	0	0	0	1	
16642	0	1	1	0	0	0	0	0	

16643 rows × 237 columns

Performing Market Basket Analysis

Finding itemsets

Now that the data is correctly formatted, we can run the a priori market basket analysis. We use mlxetend's a priori implementation to find itemsets with a minimum support of 0.05.

A preview of some of the itemsets with the highest support are shown below. The supports of the visible itemsets make sense. E.g., most of the players are right-footed, most of them are in their twenties, and the itemsets with small support are more specific.

(Note that, with the current feature set, this implementation takes a few minutes to complete this step. Using all of the features, this algorithm did not even complete overnight.)

In [ ]:

```
# for usage of apriori() see: https://pbpython.com/market-basket-analysis.html
frequent_itemsets = apriori(basket_sets, min_support=0.05, use_colnames=True)
frequent_itemsets
```

Out[ ]:

	support	itemsets
0	0.229526	(Preferred Foot Left)
1	0.770474	(Preferred Foot Right)
2	0.117287	(Age 10<=Age<20)
3	0.681307	(Age 20<=Age<30)
4	0.200745	(Age 30<=Age<40)
...	...	...
2829	0.064472	(Aggression 50<=Aggression<75, Jersey Number 0...
2830	0.054197	(Aggression 50<=Aggression<75, Penalties 50<=P...
2831	0.063330	(Aggression 50<=Aggression<75, Penalties 50<=P...
2832	0.059725	(Penalties 50<=Penalties<75, Jersey Number 0<=...
2833	0.054858	(Aggression 50<=Aggression<75, Penalties 50<=P...

2834 rows × 2 columns

Finding association rules

We grab a list of the association rules from the itemsets where the confidence is greater than 0.8, and the lift is greater than 7. A preview of these is shown below.

In [ ]:

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

pd.set_option('max_colwidth', 100)

rules[(rules['lift'] >= 7) & (rules['confidence'] >= 0.8)]
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
95	(Aggression 0<=Aggression<25)	(Position GK)	0.056180	0.114162	0.053957	0.960428	8.412842	0.047543	22.385363
199	(Position GK)	(Penalties 0<=Penalties<25)	0.114162	0.100703	0.093553	0.819474	8.137530	0.082056	4.981529
200	(Penalties 0<=Penalties<25)	(Position GK)	0.100703	0.114162	0.093553	0.928998	8.137530	0.082056	12.476171
665	(Preferred Foot Right, Position GK)	(Penalties 0<=Penalties<25)	0.102566	0.100703	0.084720	0.826011	8.202442	0.074392	5.168687
666	(Preferred Foot Right, Penalties 0<=Penalties<25)	(Position GK)	0.090308	0.114162	0.084720	0.938124	8.217470	0.074411	14.316283
669	(Penalties 0<=Penalties<25)	(Preferred Foot Right, Position GK)	0.100703	0.102566	0.084720	0.841289	8.202442	0.074392	5.654511
963	(Age 20<=Age<30, Position GK)	(Penalties 0<=Penalties<25)	0.069819	0.100703	0.056360	0.807229	8.015937	0.049329	4.665103
964	(Age 20<=Age<30, Penalties 0<=Penalties<25)	(Position GK)	0.060987	0.114162	0.056360	0.924138	8.094962	0.049398	11.676954
1448	(Body Type Normal, Position GK)	(Penalties 0<=Penalties<25)	0.081536	0.100703	0.067055	0.822402	8.166612	0.058844	5.063676
1449	(Body Type Normal, Penalties 0<=Penalties<25)	(Position GK)	0.071622	0.114162	0.067055	0.936242	8.200984	0.058879	13.893668
1552	(Reactions 50<=Reactions<75, Penalties 0<=Penalties<25)	(Position GK)	0.074926	0.114162	0.068978	0.920609	8.064054	0.060424	11.157978
1687	(Balance 25<=Balance<50, Position GK)	(Penalties 0<=Penalties<25)	0.074746	0.100703	0.063330	0.847267	8.413522	0.055803	5.888029
1688	(Balance 25<=Balance<50, Penalties 0<=Penalties<25)	(Position GK)	0.065673	0.114162	0.063330	0.964318	8.446922	0.055833	24.826175
1696	(Jersey Number 0<=Jersey Number<25, Penalties 0<=Penalties<25)	(Position GK)	0.059304	0.114162	0.054197	0.913880	8.005112	0.047427	10.286141
1700	(Position GK, Vision 25<=Vision<50)	(Penalties 0<=Penalties<25)	0.075708	0.100703	0.063811	0.842857	8.369732	0.056187	5.722799
1701	(Penalties 0<=Penalties<25, Vision	(Position GK)	0.068497	0.114162	0.063811	0.931579	8.160141	0.055991	12.946861

25<=Vision<50)

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2278	(Age 20<=Age<30, Preferred Foot Right, Position GK)	(Penalties 0<=Penalties<25)	0.062789	0.100703	0.050952	0.811483	8.058184	0.044629	4.770383
2279	(Age 20<=Age<30, Preferred Foot Right, Penalties 0<=Penalties<25)	(Position GK)	0.054618	0.114162	0.050952	0.932893	8.171654	0.044717	13.200437
2283	(Age 20<=Age<30, Penalties 0<=Penalties<25)	(Preferred Foot Right, Position GK)	0.060987	0.102566	0.050952	0.835468	8.145690	0.044697	5.454466
3018	(Preferred Foot Right, Body Type Normal, Position GK)	(Penalties 0<=Penalties<25)	0.072883	0.100703	0.060626	0.831822	8.260151	0.053287	5.347291
3019	(Preferred Foot Right, Body Type Normal, Penalties 0<=Penalties<25)	(Position GK)	0.064351	0.114162	0.060626	0.942110	8.252389	0.053280	15.302135
3025	(Body Type Normal, Penalties 0<=Penalties<25)	(Preferred Foot Right, Position GK)	0.071622	0.102566	0.060626	0.846477	8.253022	0.053280	5.845583
3164	(Preferred Foot Right, Reactions 50<=Reactions<75, Penalties 0<=Penalties<25)	(Position GK)	0.067055	0.114162	0.062489	0.931900	8.162950	0.054834	13.007830
3170	(Reactions 50<=Reactions<75, Penalties 0<=Penalties<25)	(Preferred Foot Right, Position GK)	0.074926	0.102566	0.062489	0.834002	8.131393	0.054804	5.406283
3403	(Preferred Foot Right, Balance 25<=Balance<50, Position GK)	(Penalties 0<=Penalties<25)	0.066755	0.100703	0.057081	0.855086	8.491162	0.050359	6.205708
3404	(Preferred Foot Right, Balance 25<=Balance<50, Penalties 0<=Penalties<25)	(Position GK)	0.059064	0.114162	0.057081	0.966429	8.465412	0.050338	26.387232
3410	(Balance 25<=Balance<50, Penalties 0<=Penalties<25)	(Preferred Foot Right, Position GK)	0.065673	0.102566	0.057081	0.869167	8.474255	0.050345	6.859411
3415	(Preferred Foot Right, Position GK, Vision 25<=Vision<50)	(Penalties 0<=Penalties<25)	0.068197	0.100703	0.057922	0.849339	8.434100	0.051055	5.969018
3416	(Preferred Foot Right, Penalties 0<=Penalties<25, Vision 25<=Vision<50)	(Position GK)	0.061407	0.114162	0.057922	0.943249	8.262361	0.050912	15.609075
3422	(Penalties 0<=Penalties<25, Vision 25<=Vision<50)	(Preferred Foot Right, Position GK)	0.068497	0.102566	0.057922	0.845614	8.244613	0.050897	5.812927
5132	(Body Type Normal, Reactions 50<=Reactions<75, Penalties 0<=Penalties<25)	(Position GK)	0.054798	0.114162	0.050652	0.924342	8.096750	0.044396	11.708466

Discussion

The table shown above is for associations with high confidence (> 0.8) and lift (> 7).

The first thing to notice is that many of the association rules shown seem to be centered around goalkeepers. It seems that goalkeepers have very distinctive attributes, e.g.:

- They are not very aggressive (0 <= aggression < 25)
- They are not prone to causing penalties (0 <= penalties < 25)
- They have a "normal" body type, have medium reactions, and have few penalties.

There are a lot of repeated associations in this list, but far and large they are mostly about goalkeepers. This is likely related to the bimodal distributions in the earlier histograms, where goalkeepers clearly stood out from the rest; most likely the goalkeepers are very specialized while the other players are more diverse, and therefore all of these selected associations involve goalkeepers.

These results are specific to this choice of features. When we tried different set of features (e.g., including values such as "International Reputation," "Value," "Wage," "Overall," etc.), there were many less-interesting correlations. Many of the associations we saw were that the highest-paid players were those with the highest international reputation, and who had the highest overall and potential scores.

We also did not include many of the player statistics because the algorithm takes too long to run otherwise. If we knew more about soccer or FIFA 18, we could probably draw much more interesting conclusions by using the most relevant or interesting statistics by choosing different sets of features.