Peer-Assignment

- Pandas
- Pandas.read_sql
- SQLite3

Sections required in your report:

- **Brief description of the data set and a summary of its attributes**
- **Initial plan for data exploration**
- **Actions taken for data cleaning and feature engineering**
- **Key Findings and Insights, which synthesizes the results of Exploratory Data Analysis in an insightful and actionable manner**
- **Formulating at least 3 hypothesis about this data**
- **Conducting a formal significance test for one of the hypotheses and discuss the results**
- **Suggestions for next steps in analyzing this data**
- **A paragraph that summarizes the quality of this data set and a request for additional data if needed**

Brief Description of The Data Set and A Summary of Its Attributes

The title of the dataset is "euroleague". It was a fictional dataset I used in one of the Homeworks I did in Analysis of Algorithms I class in Istanbul Technical University. It is briefly a list on players' details from year 2000 to 2021.

- **Season: Years the player active in**
- **Name: Player's name**
- **Team: Player's affliated team**
- **Rebound: # of Rebound in the season**
- **Assist: # of Assist scored in the season**
- **Point: # of Points scored in the season**

Initial plan for DE

Reading the data

In [336]:

```
# Imports
import sqlite3 as sq3
import pandas.io.sql as pds
import pandas as pd
import os
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

sns.set()
%matplotlib inline
```

In [337]:

```
filepath = "data/euroleague.csv"
data = pd.read_csv(filepath)
data.head()
```

Out[337]:

Season	Name	Team	Rebound	Assist	Point

	Season	Name	Team	Rebound	Assist	Point
0	2000-2001	Ibrahim Kutluay	ATH	44	22	282
1	2000-2001	Andrew Betts	ATH	122	18	213
2	2000-2001	Vrbica Stefanov	ATH	46	41	179
3	2000-2001	Dimos Dikoudis	ATH	92	8	147
4	2000-2001	Martin Muursepp	ATH	83	12	147

In [338]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6873 entries, 0 to 6872
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Season   6873 non-null   object
 1   Name     6873 non-null   object
 2   Team     6873 non-null   object
 3   Rebound  6873 non-null   int64 
 4   Assist   6873 non-null   int64 
 5   Point    6873 non-null   int64 
dtypes: int64(3), object(3)
memory usage: 322.3+ KB
```

In [339]:

```
# check for missing values
data.isnull().sum()
```

Out[339]:

```
Season     0
Name       0
Team       0
Rebound    0
Assist     0
Point      0
dtype: int64
```

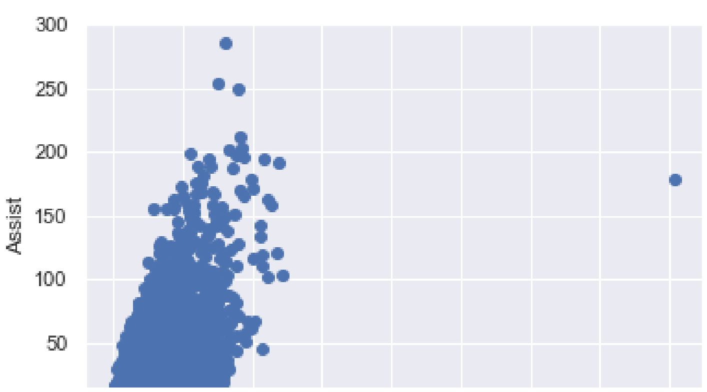
It can be seen there is no missing values.

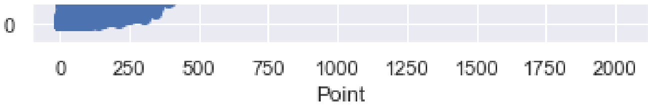
In [340]:

```
#Scatter Plot
ax = plt.axes()

ax.scatter(data.Point, data.Assist)

# Label the axes
ax.set(xlabel='Point',
       ylabel='Assist',)
```

Out[340]:

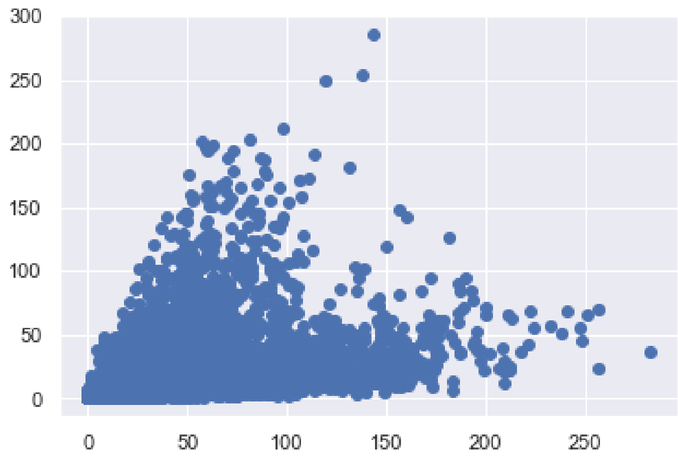
```
[Text(0.5, 0, 'Point'), Text(0, 0.5, 'Assist')]
```





In [341]:

```
x = data['Rebound']
y = data['Assist']
plt.scatter(x,y)
plt.show()
```



In [342]:

```
data.describe()
```

Out[342]:

	Rebound	Assist	Point
count	6873.000000	6873.000000	6873.000000
mean	40.483195	19.710025	103.885930
std	39.181647	26.210593	99.046325
min	0.000000	0.000000	0.000000
25%	10.000000	3.000000	24.000000
50%	29.000000	11.000000	79.000000
75%	58.000000	26.000000	156.000000
max	282.000000	286.000000	2017.000000

In [343]:

```
data['Point'].sort_values()
```

Out[343]:

```
508        0
1360       0
5666       0
5667       0
1361       0
        ...
5602     569
6357     590
6228     595
5757     609
5901    2017
Name: Point, Length: 6873, dtype: int64
```

In [344]:

```
data['Assist'].sort_values()
```

Out[344]:

```
1841      0
5435      0
3886      0
3888      0
5433      0
       ...
6135    203
6024    212
5960    249
6510    254
6258    286
Name: Assist, Length: 6873, dtype: int64
```

In [345]:

```
data['Point'].sort_values()
```

Out[345]:

```
508        0
1360       0
5666       0
5667       0
1361       0
        ...
5602     569
6357     590
6228     595
5757     609
5901    2017
Name: Point, Length: 6873, dtype: int64
```

It can be seen that one data point is considerable to be very high compared to the rest of data.

Which if we check:

In [346]:

```
df = pd.DataFrame(data)
df.iloc[[5901]]
```

Out[346]:

	Season	Name	Team	Rebound	Assist	Point
5901	2017-2018	Alexey Shved	KHI	89	178	2017

It can be seen that the Point the player scored within the season is actually the same as what year the season was in. This may mean an error on data entry where somebody accidentally copied the season year in the point column.

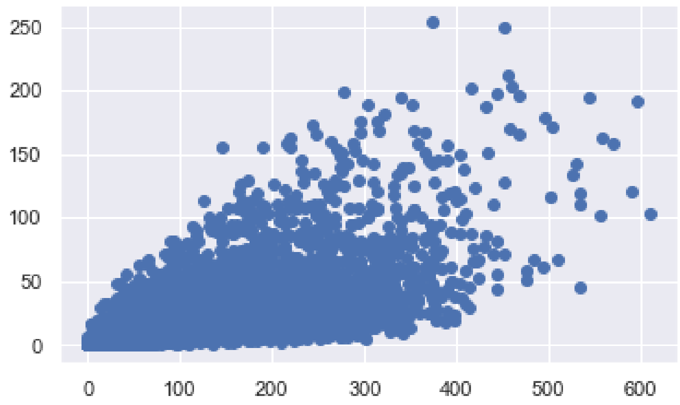
By dropping the outlier,

In [347]:

```
org_data = data.copy()
data = data.drop([5901])
```

In [348]:

```
x = data['Point']
y = data['Assist']
plt.scatter(x,y)
plt.show()
```



We can see now that the scatter plot looks better, even though it is more dense in between around approx. 0 and (450, 200), the graph isn't that distorted by the values above them.

Lastly, since in the plot of (Assist, Rebound) we couldn't see any outliers, we can safely say that the data is now cleaned.

One-hot encoding for dummy variables

In [349]:

```
# Get a Pd.Series consisting of all the string categoricals
one_hot_encode_cols = data.dtypes[data.dtypes == np.object]  # filtering by string categ
oricals
one_hot_encode_cols = one_hot_encode_cols.index.tolist()  # list of categorical fields

df[one_hot_encode_cols].head().T
```

Out[349]:

	0	1	2	3	4
Season	2000-2001	2000-2001	2000-2001	2000-2001	2000-2001
Name	Ibrahim Kutluay	Andrew Betts	Vrbica Stefanov	Dimos Dikoudis	Martin Muursepp
Team	ATH	ATH	ATH	ATH	ATH

In [350]:

```
# Do the one hot encoding
df = pd.get_dummies(data, columns=one_hot_encode_cols, drop_first=True)
df.describe().T
```

Out[350]:

	count	mean	std	min	25%	50%	75%	max
Rebound	6872.0	40.476135	39.180126	0.0	10.0	29.0	58.0	282.0
Assist	6872.0	19.686991	26.142840	0.0	3.0	11.0	26.0	286.0
Point	6872.0	103.607538	96.326789	0.0	24.0	79.0	156.0	609.0
Season_2001-2002	6872.0	0.063009	0.242997	0.0	0.0	0.0	0.0	1.0
Season_2002-2003	6872.0	0.049476	0.216876	0.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...
Team_ZAG	6872.0	0.002037	0.045093	0.0	0.0	0.0	0.0	1.0
Team_ZAL	6872.0	0.044383	0.205960	0.0	0.0	0.0	0.0	1.0
Team_ZAS	6872.0	0.003638	0.060210	0.0	0.0	0.0	0.0	1.0
Team_ZEN	6872.0	0.004511	0.067018	0.0	0.0	0.0	0.0	1.0
Team_ZVE	6872.0	0.016589	0.127735	0.0	0.0	0.0	0.0	1.0

2748 rows × 8 columns

Feature Engineering

Interaction Feature

Being a simple dataset, not many modifications may be needed. However, we can assume that in a fictional world, rebound counts 0.5, assist counts 0.75, and goals counts 1 points. With this in mind we can use Polynomial Features to add a new feature of Goals.

In [351]:

```
X = data.copy()
X['PtbyAssist'] = X['Assist']*0.75
X['PtbyRebound'] = X['Rebound']*0.5
X['Goals'] = round((X['Point'] - X['PtbyAssist'] - X['PtbyRebound'])/1)
X.head()
```

Out[351]:

	Season	Name	Team	Rebound	Assist	Point	PtbyAssist	PtbyRebound	Goals
0	2000-2001	Ibrahim Kutluay	ATH	44	22	282	16.50	22.0	244.0
1	2000-2001	Andrew Betts	ATH	122	18	213	13.50	61.0	138.0
2	2000-2001	Vrbica Stefanov	ATH	46	41	179	30.75	23.0	125.0
3	2000-2001	Dimos Dikoudis	ATH	92	8	147	6.00	46.0	95.0
4	2000-2001	Martin Muursepp	ATH	83	12	147	9.00	41.5	96.0

Categories and features derived from category aggregates

In [352]:

```
data['Name'].value_counts()
```

Out[352]:

```
Georgios Printezis      18
Paulius Jankunas        18
Nikos Zisis             17
Juan Carlos Navarro     17
Felipe Reyes            17
                        ..
Kamil Chanas             1
Jose Antonio Paraiso     1
Calvin Bowman            1
Bernard King             1
Ike Ofoegbu              1
Name: Name, Length: 2645, dtype: int64
```

Seeing that some players played in more than one seasons, we could combine all the data regarding to the player and unify them, however that may creates some outliers which we do not want.

In [353]:

```
pd.get_dummies(data['Name'], drop_first=True).head()
```

Out[353]:

	A.J. Ogilvy	A.J. Slaughter	Aaron Cel	Aaron Craft	Aaron Doornekamp	Aaron Harrison	Aaron Jackson	Aaron Miles	Aaron Mitchell	Aaron White	...	Zaza Pachulia	Zelimir Zagorac	Zoran Dragic	Z E
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

2	A.J Ogilvy	A.J Slaughter	Aaron Cel	Aaron Craft	Aaron Doornekamp	Aaron Harrison	Aaron Jackson	Aaron Miles	Aaron Mitchell	Aaron White	...	Zaza Pachulia	Zelimir Zagorac	Zoran Dragic	Z E
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

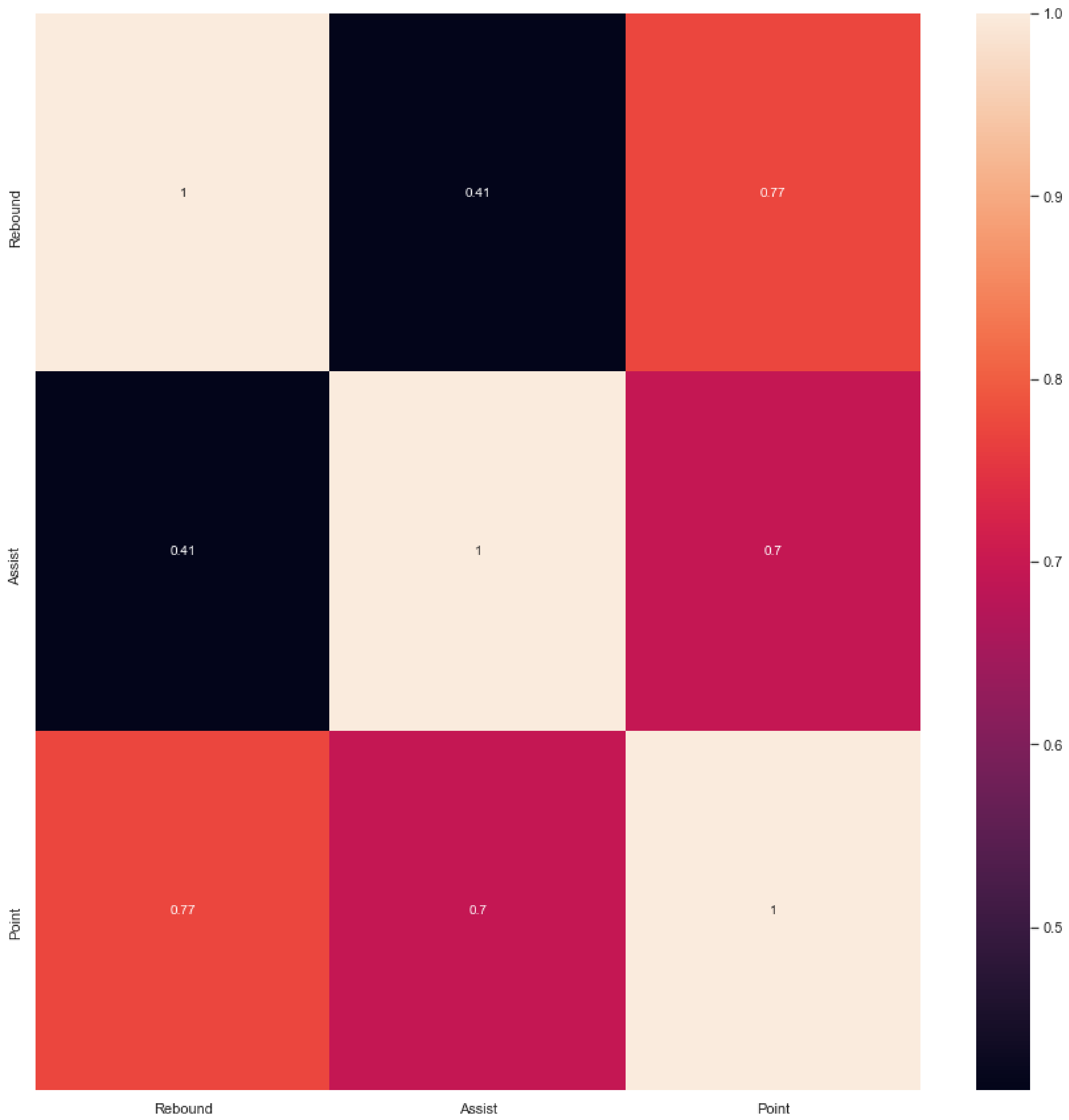
5 rows × 2644 columns

In [354]:

```
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(), color='b', annot=True)
```

Out[354]:

<AxesSubplot:>



Key Findings and Insights

To find key findings and insights, we can try to ask some questions to be framed from the data. Such as, To find key findings and insights, we can try to ask some questions to be framed from the data. Such as,

> 1. **What is the percentage of people who play in more than one season?**
> 2. **What is the percentage of people who play only in one season?**
> 3. **Which team has the most score of all time (assuming the games started being played in 2000s)**
> 4. **Which player is the top scorer (within a season) of all time?**

For the first question, first we may first count the unique combinations of columns and set it as dataone

In [355]:

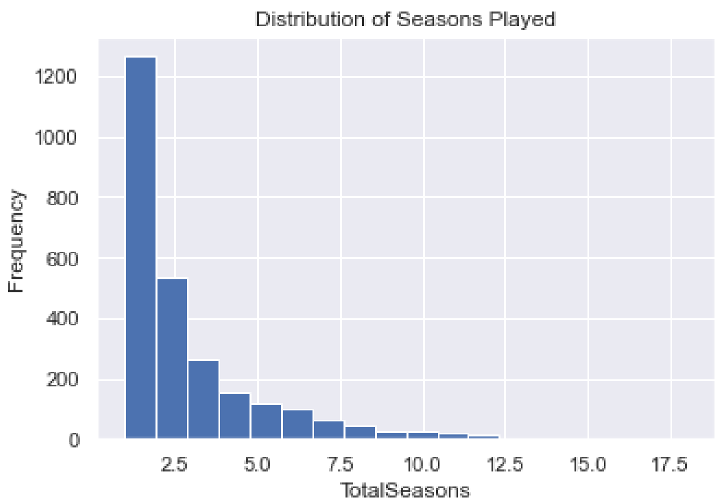
```
from pandas import DataFrame

count = data['Name'].value_counts()
dataone = [tuple((x, y)) for x, y in count.items()]
dataone = DataFrame (dataone, columns = ['Name','TotalSeasons'])
dataone.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2645 entries, 0 to 2644
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Name          2645 non-null   object
 1   TotalSeasons  2645 non-null   int64
dtypes: int64(1), object(1)
memory usage: 41.5+ KB
```

Here we just created a new dataframe of the total seasons every player has played.

In [356]:

```
ax = plt.axes()
ax.hist(dataone.TotalSeasons, bins=18);

ax.set(xlabel='TotalSeasons',
       ylabel='Frequency',
       title='Distribution of Seasons Played');
```



From the histogram it can be seen that the number of player that played less is so much higher than the ones who played frequently. So even though it is clear enough, in order to see the exact answer, we need to divide the data.

In [357]:

```
dataonex = dataone.loc[dataone['TotalSeasons'] <= 1,:]
dataoney = dataone.loc[dataone['TotalSeasons'] > 1, :]
dataonex.head()
```

Out[357]:

	Name	TotalSeasons
1379	Josh Akognon	1
1380	Ilya Popov	1
1381	Michael Kuebler	1
1382	Serguei Bazarevitch	1
1383	Mutlu Demir	1

In [358]:

```
dataoney.head()
```

Out[358]:

	Name	TotalSeasons
0	Georgios Printezis	18
1	Paulius Jankunas	18
2	Nikos Zisis	17
3	Juan Carlos Navarro	17
4	Felipe Reyes	17

In [359]:

```
percentage1 = (dataonex['TotalSeasons'].sum() / dataone['TotalSeasons'].sum()) * 100
percentage1
```

Out[359]:

```
18.42258440046566
```

In [360]:

```
percentage2 = (dataoney['TotalSeasons'].sum() / dataone['TotalSeasons'].sum()) * 100
percentage2
```

Out[360]:

```
81.57741559953435
```

In [361]:

```
P = {'MoreThanOne': [percentage1], 'One': [percentage2]}
dataP = pd.DataFrame(data=P)
dataP.head()
```

Out[361]:

	MoreThanOne	One
0	18.422584	81.577416

Which team has the most score of all time (assuming the games started being played in 2000s)?

We can directly directly count that using pandas count() method, though it is worth noting that other columns but Point should be neglected here.

In [362]:

```
countT = data.groupby(['Team']).count()
countT[['Point']]
```

Out[362]:

Team	Point
ARI	27
ATH	86
AVE	12
BAY	86
BBB	15
...	...
ZAG	14
ZAL	305
ZAS	25
ZEN	31
ZVE	114

82 rows × 1 columns

In [363]:

```
countT[['Point']].sort_values(by='Point', ascending=False)
```

Out[363]:

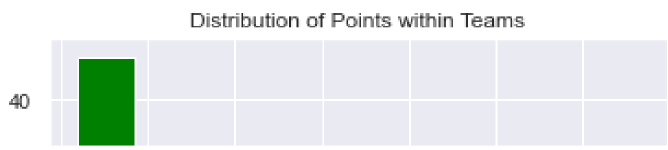
Team	Point
OLY	332
CAJ	311
ZAL	305
FCB	305
CSKA	302
...	...
LUG	12
VER	11
SPL	11
CHO	11
OOS	10

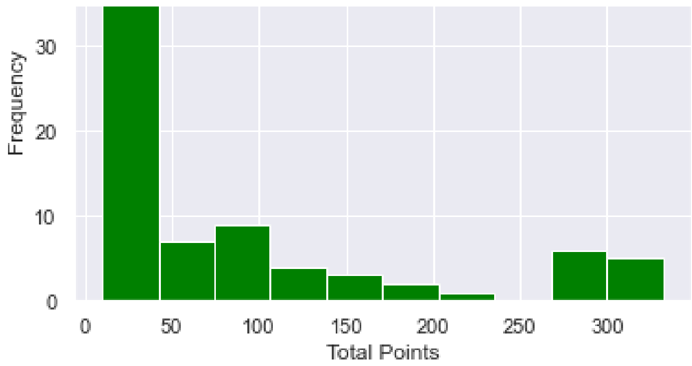
82 rows × 1 columns

And with sort_values method, we can see that the team with most point of all time is OLY, and with least point is OOS.

In [364]:

```
ax = plt.axes()
ax.hist(countT.Point, bins=10, color='green');

ax.set(xlabel='Total Points',
       ylabel='Frequency',
       title='Distribution of Points within Teams');
```





Which player is the top scorer (within a season) of all time?

In [365]:

```
countP = data.groupby(['Name']).count()
countP[['Point']]
```

Out[365]:

	Point
Name	
A.J. Guyton	2
A.J. Ogilvy	1
A.J. Slaughter	1
Aaron Cel	2
Aaron Craft	1
...	...
Zoran Planinic	10
Zoran Savic	2
Zoran Viskovic	1
Zoran Vrkic	1
Zygimantas Janavicius	2

2645 rows × 1 columns

In [366]:

```
countP[['Point']].sort_values(by='Point', ascending=False)
```

Out[366]:

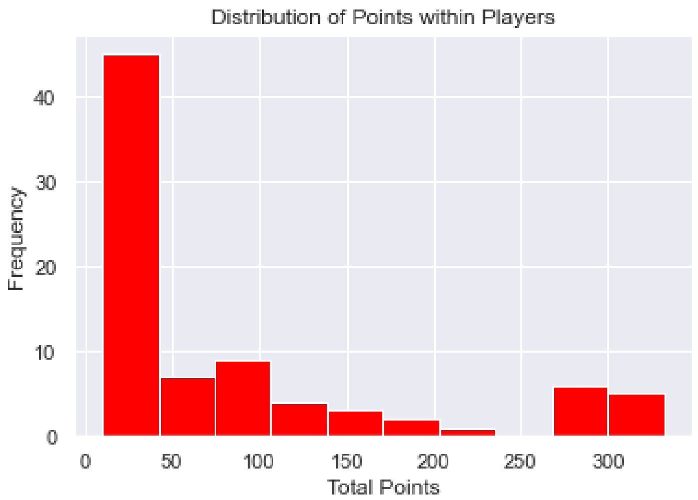
	Point
Name	
Paulius Jankunas	18
Georgios Printezis	18
Felipe Reyes	17
Nikos Zisis	17
Juan Carlos Navarro	17
...	...
Maxim Grigoryev	1
Maxim Barashkov	1
Mauro Sartori	1
Maurice Stuckey	1

Maurice Stuckey	1
	Point
Oleksiy Pecherov	1
Name	

2645 rows × 1 columns

In [367]:

```
ax = plt.axes()
ax.hist(countT.Point, bins=10, color='red');

ax.set(xlabel='Total Points',
       ylabel='Frequency',
       title='Distribution of Points within Players');
```



Formulating at least 3 hypothesis about this data

Looking back to the dataset,

In [368]:

```
X.head()
```

Out[368]:

	Season	Name	Team	Rebound	Assist	Point	PtbyAssist	PtbyRebound	Goals
0	2000-2001	Ibrahim Kutluay	ATH	44	22	282	16.50	22.0	244.0
1	2000-2001	Andrew Betts	ATH	122	18	213	13.50	61.0	138.0
2	2000-2001	Vrbica Stefanov	ATH	46	41	179	30.75	23.0	125.0
3	2000-2001	Dimos Dikoudis	ATH	92	8	147	6.00	46.0	95.0
4	2000-2001	Martin Muursepp	ATH	83	12	147	9.00	41.5	96.0

In [369]:

```
X.tail()
```

Out[369]:

	Season	Name	Team	Rebound	Assist	Point	PtbyAssist	PtbyRebound	Goals
6868	2020-2021	Marko Simonovic	ZVE	18	4	34	3.00	9.0	22.0
6869	2020-2021	Branko Lazic	ZVE	15	7	30	5.25	7.5	17.0
6870	2020-2021	Aleksa Uskokovic	ZVE	4	9	19	6.75	2.0	10.0
6871	2020-2021	Quim Colom	ZVE	1	0	5	0.00	0.5	4.0
6872	2020-2021	Borisa Simanic	ZVE	3	1	3	0.75	1.5	1.0

Since our dataset is consisting of data per-season, we may define/formulate a hypothesis such as
Null: The average point percentage increase in the most recent season from the first is the same as the first
Alternative: The average point percentage increase in the most recent season is higher than the first Also another hypothesis we can formulate is
Null: The average point percentage increase in the most recent season is is the same than in the first season
Alternative: The average point percentage increase in the second most recent season is lower than in the first season
And
Null: The average point percentage increase in the most recent season is the same with in the first season
Alternative: The average point percentage increase made in the second most recent season is lower than all season

Note: Since we haven't calculated the average points, the statistical shorthand notation for the hypothesis will be given next while conducting a formal significance test for one of the hypotheses.

Conducting a formal significance test for one of the hypotheses and discuss the results

We'll choose the first hypotheses. For that we have to take/calculate the average of both points. For that we may take the related the data values from the dataset to 2 different datasets for each season year.

In [370]:

```
data_1 = data[data['Season'] == '2020-2021']
data_2 = data[data['Season'] == '2019-2020']
data_3 = data[data['Season'] == '2000-2001']
data_4 = data[data['Season'] == '2001-2002']
```

After getting two datasets of the wanted season years, we can now calculate the average/mean values

In [371]:

```
data_1.mean()
```

Out[371]:

```
Rebound    30.756554
Assist     18.423221
Point      82.737828
dtype: float64
```

In [372]:

```
data_2.mean()
```

Out[372]:

```
Rebound     51.046823
Assist      29.287625
Point      135.769231
dtype: float64
```

In [373]:

```
data_3.mean()
```

Out[373]:

```
Rebound    31.930464
Assist     10.993377
Point      82.963576
dtype: float64
```

In [374]:

```
data_4.mean()
```

Out[374]:

```
Rebound    20.411085
Assist      8.170901
Point      54.346420
dtype: float64
```

In [375]:

```
a1 = 82.737828
a2 = 135.769231
b1 = 82.963576
b2 = 54.346420
```

In [383]:

```
a = (a2-a1)/a1*100
b = (b2-b1)/b1*100
a
```

Out[383]:

```
64.09571568642097
```

H0: b0 = 0.64 Ha: b < 0.64

The formula for test-statistic is (Best Estimate - Hypothesized Estimate)/Standard Error of Estimate.

First we calculate standard error with P0 according to the null hypothesis.

In [378]:

```
se = np.sqrt(0.42 * (1-0.64) / len(data_1))
se
```

Out[378]:

```
0.023796893386143094
```

Now we calculte the test-statistic z-score

In [379]:

```
#Best estimate
be = -0.34  #hypothesized estimate
he = 0.64
test_stat = (be - he)/se
test_stat
```

Out[379]:

```
-41.18184605435318
```

with z-value being -41, we can directly see from z-table or even guess that p-value is much less than .00001, hence it is much less than 0.05. Therefore, we can strongly reject the null hypothesis.
Furthermore, we can also see from the value of 'b' we calculate before, which is a percentage increase between 1st and 2nd season, that it is

In [385]:

```
b
```

Out[385]:

```
-34.493638509506866
```

Which not only significantly less than the increase between the last seasons, it also experienced decreasing in the average point.

Suggestions for next steps in analyzing this data

Regarding the balance of the dataset, we may not to conduct further techniques since the dataset itself as can be seen is particularly clean and have no missing values, which may be inevitable since this dataset is used for Algorithm class, which may be intentionally made balanced beforehand. However, in analyzing the data further, or maybe to tinker the data further some well-thought details may be add, since as can be seen above, some parts like how the points are divided are determined by me. Those details can be changed for better. Furthermore, the hypothesis that was tested gave an 'extreme' result of z-value. Some added features or values may make the dataset becomes much more complicated and affluent to explore and analyze.

A paragraph that summarizes the quality of this data set and a request for additional data if needed

As mentioned in the previous part, this dataset is considerably balanced due to the main nature of the purpose, being an algorithm course dataset. This doesn't mean it can't be analyzed, it still is, but some features may be added within Feature Engineering. The best to do is to add additional data values. Like injury condition, or characteristics of the player that may enrich the possibilites of the dataset analytics. Index may also be added, with the Season format showing only a year instead of two, to ease and further possibilities in cleaning the data.

In [ ]:

```

```