Data Analyst Nanodegree

Project: Intro to Data Analysis

Index

1 Data

1.1 Which Dataset is used?

I used the Baseball Dataset.
Not all of the CSV-files are used. For analyzing following CSV-files are relevant:

- AllstarFull.csv
- HalloOfFame.csv
- Managers.csv
- Team.csv
- Salaries.csv
- Schools.csv
- CollegePlaying.csv
- SeriesPost.csv

1.2 Questions that will be analyzed

1. How is the Correlation of some Attributes?
2. How often will the salary change (avg)?
3. Compare the count of salary change with the change of salary.
4. What is the change average of salary over the years?
5. Which players start their career in school and can be found in the Hall of Fame?
6. Which states and cities have the most baseball colleges? (Show only Top 10)
7. Reject of retain the null-hypothesis for win and loss of Series post data?

1.3 Data Wrangling

The column name yearid in the HallOfFame.csv is written different than in other used .csv's. To have consistent column-names I used the rename()-function and changed the column yearid to yearID:

```
# Change yearid to YearID
HallOfFame.rename(columns={'yearid': 'yearID'}, inplace=True)
#print HallOfFame.head()
```

For a more comfortable analyzing of the school/college information I merged the schools.csv and the college_playing. Analyzing of one table is a lot of easier than two tables. Another point to merge these tables is that information of college_playing only have a schoolID, a yearID and a playerID. In the schools data exists also the schoolsID. It makes no sense to have more sources when they can be joined / merged. The schoolID can be used as the key for both tables, and the yearID and playerID can be joined on the schools table without losing important data / informations.

```
     schoolID                     name_full         city state country
0  abilchrist  Abilene Christian University      Abilene    TX     USA
1     adelphi            Adelphi University  Garden City    NY     USA
2    adrianmi                Adrian College       Adrian    MI     USA
3       akron           University of Akron        Akron    OH     USA
4     alabama         University of Alabama   Tuscaloosa    AL     USA
    playerID schoolID  yearID
0  aardsda01   pennst    2001
1  aardsda01     rice    2002
2  aardsda01     rice    2003
3   abadan01  gamiddl    1992
4   abadan01  gamiddl    1993
```

```
# schools and collage_playing can be merged to one Dataframe
def combine_dataframes(schools, college_playing):

    merge = college_playing.merge(schools, on =  ['schoolID'], how = 'left')
    return merge

print combine_dataframes(schools, college_playing)
```

If the data of college_playing and schools also needed in the future, it makes no sense (maybe only for now) to delete these two tables.

The new merged table will be saved in a new csv-file to analyze the new generated data in the next chapter.

```
# merged Dataframes save in a new CSV file
school_playing = pd.DataFrame(combine_dataframes(schools, college_playing))
school_playing.to_csv('schools_playing.csv', index = False)

school_playing = pd.read_csv('schools_playing.csv')
print school_playing.head()
```

2 Description and Results from analyzing baseball data-set

2.1 How is the Correlation of some Attributes?

Correlation describes the relationship between two variables. To test the correlation of two variables I created a function:

```
def correlation(x, y):
    x_std = (x - x.mean()) / x.std(ddof = 0)
    y_std = (y - y.mean()) / y.std(ddof = 0)

    return (x_std * y_std).mean()
```

In the next step I created the following variables to test the correlation between them:

```
salary_c = salary['salary']
year_c = salary['yearID']
ballots_c = HallOfFame['ballots']
rank_c = team['Rank']

print correlation(ballots_c, rank_c)
# -0.134767987156
print correlation(salary_c, year_c)
# 0.35173999336
print correlation(year_c, rank_c)
#-0.0415697039104
print correlation(salary_c, rank_c)
#-0.000443072942508
```

The result of the correlation test is that the variables I choose have no extreme positive or negative correlation. The result of ballots/rank, year/rank, salary/rank has no real positive or negative relationship because the result is close to 0. Only at the correlation test of salary and year the result is a little bit higher at 0.35 but not close enough to 1 to say there is an extreme correlation of those variables.

2.2 How often will the salary change (avg)?

The first interesting information when looking at the salary-data from baseball players is to analyze the average of how often the salary is change in a baseball-player career.

To get this information I used the groupby-function and grouped the playerID to get the count of how often data will be found for every player. The groupby-function count how often the

player is in the data-set of the salary. Only counting the playerID shows the counts for every column, but only the counts in the salary column are relevant for this analysis.

```
salary_data =  salary.groupby('playerID').count()['salary']
print salary_data
```

The result of this looks like the following part:

```
playerID
aardsda01     7
aasedo01      4
abadan01      1
abadfe01      5
abbotje01     4
abbotji01     9
```

But that is not the result of the question; there we need the average of how often it changed. The query above shows only the count of changes per player. To get the average the count must be summarized and divided by the distinct count of players. For this I used the mean-function:

```
salary_avg =  salary.groupby('playerID').count()['salary'].mean()
print salary_avg
```

The result of this code is 5.1267. This means that a baseball players salary normally changed five times in average during his career.

This result raised some new questions; one is to analyze how the count of salary changes compared to the salary change. This question will be answered in the next Chapter 2.3.

2.3 Compare the count of salary change with the change of salary.

More interesting then then average of salary change is to look at the player's salary and comparing them with the count of change per player.

In the first step for comparing I used the known code for the count:

```
salary_data =  salary.groupby('playerID').count()['salary']
print salary_data
```

```
playerID
aardsda01     7
aasedo01      4
abadan01      1
abadfe01      5
abbotje01     4
abbotji01     9
```

To get the information about the salary change I coded two other groupby-functions. The first groupby-function looks at the maximum salary per player and the second one at the minimum salary per player. These two functions is needed to calculate the delta or called the change.

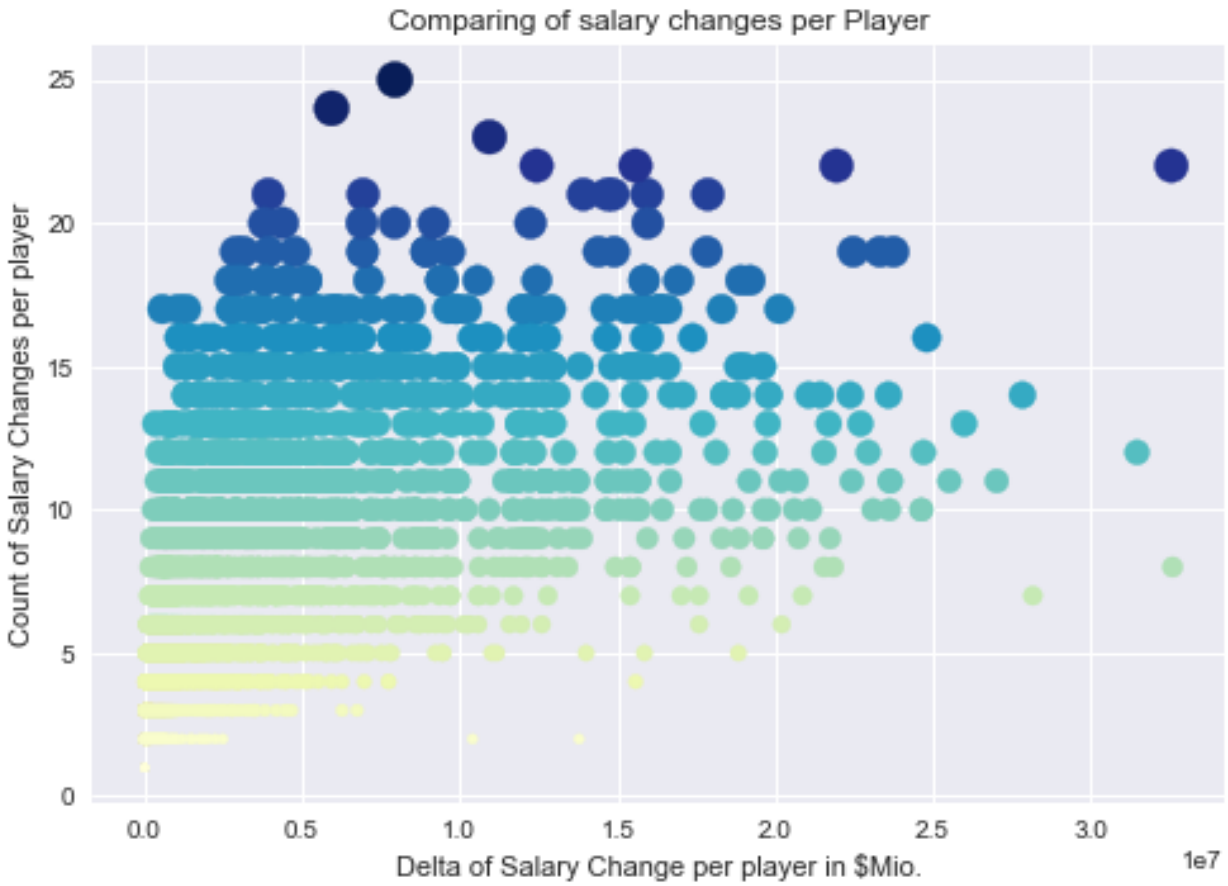
```
salary_max = salary.groupby('playerID').max()['salary']
#print salary_max
#playerID
#aardsda01     4500000
#aasedo01       675000
#abadan01       327000


salary_min = salary.groupby('playerID').min()['salary']
#print salary_min
#playerID
#aardsda01     300000
#aasedo01      400000
#abadan01      327000
```

In the next step the change has be calculated this will be done by subtracting the minimum from the maximum:

```
delta = (salary_max-salary_min)
#print delta
#playerID
#aardsda01     4200000
#aasedo01       275000
#abadan01            0
```

This long result-lists are not as meaningful as a visualization of the data. So I created a scatter-plot to show the result:

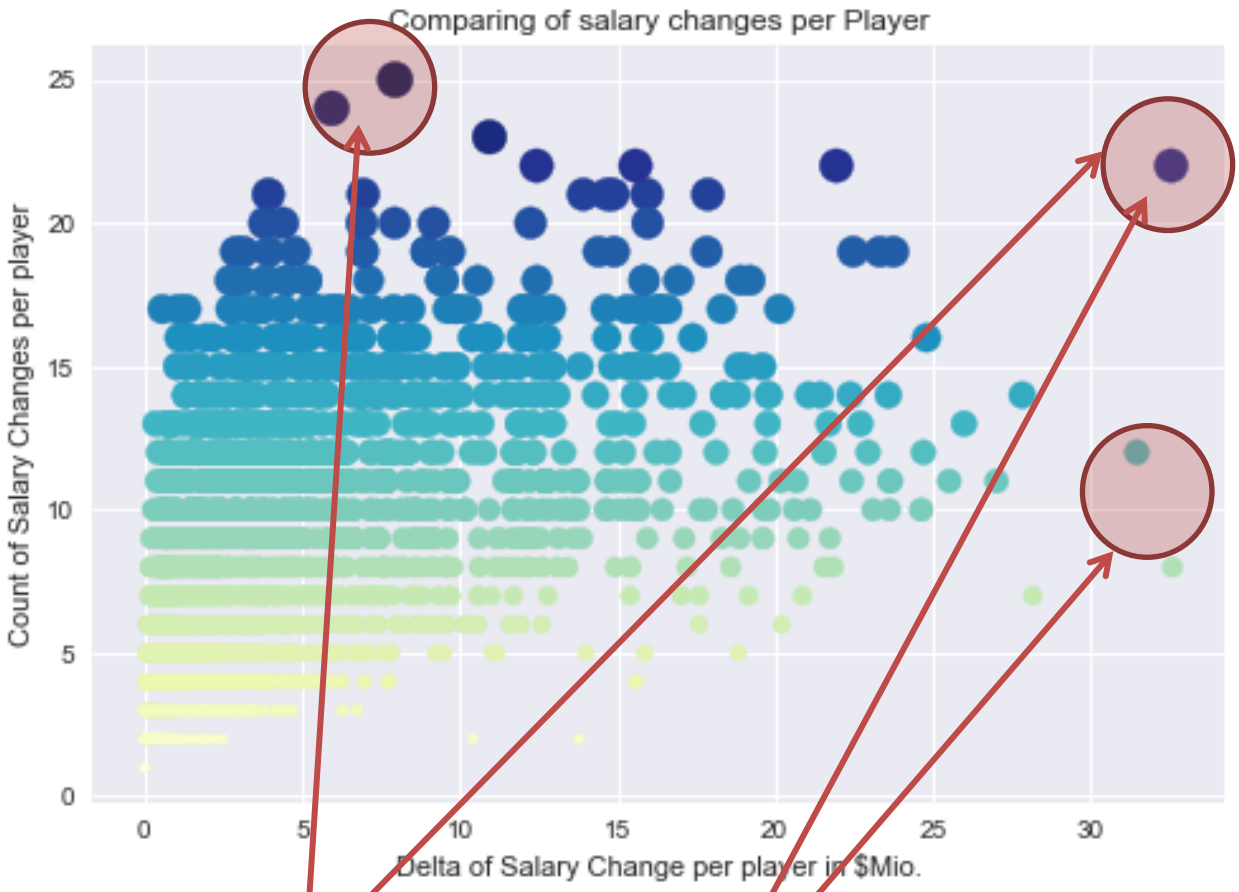
```
plt.scatter(delta, salary_data, s = salary_data*8, c = salary_data, cmap='YlGnBu')
plt.ylabel('Count of Salary Changes per player')
plt.xlabel('Delta of Salary Change per player in $Mio.')
plt.title('Comparing of salary changes per Player')
```



By looking at the x-axis it's not known what the values from 0.0 to 3.0 (1e7) mean. To solve this problem I divided the calculation of deltas by 1. Mio to get only the Million value of change:

```
delta = (salary_max-salary_min) / 1000000
print delta

#playerID after dividing
#aardsda01     4.200
#aasedo01      0.275
#abadan01      0.000
```

Now the visualization change and is readable:



This visualization raised a lot of new questions like:

- Which players have more than 20 salary changes?
- Which players get the lowest/highest salary?
- Why have some players a lot of changes and other not?
- Why get some players a lot of millions more than another player?
- What is the average of salary that a player normally gets?
- What are depending attributes that a players-salary change a lot or not?
- …

but they will not analyzed in the next chapters. There I look at some other interesting parts of the baseball dataset.

2.4 What is the change average of salary over the years?

Another interesting Question about the salary is how the average salary evolved over the years. It is pretty much the same average of salary, or is the salary increased/degreased over the years? Are years existing which have outliers?

In the first step I created a query that shows the average salary of players per year:

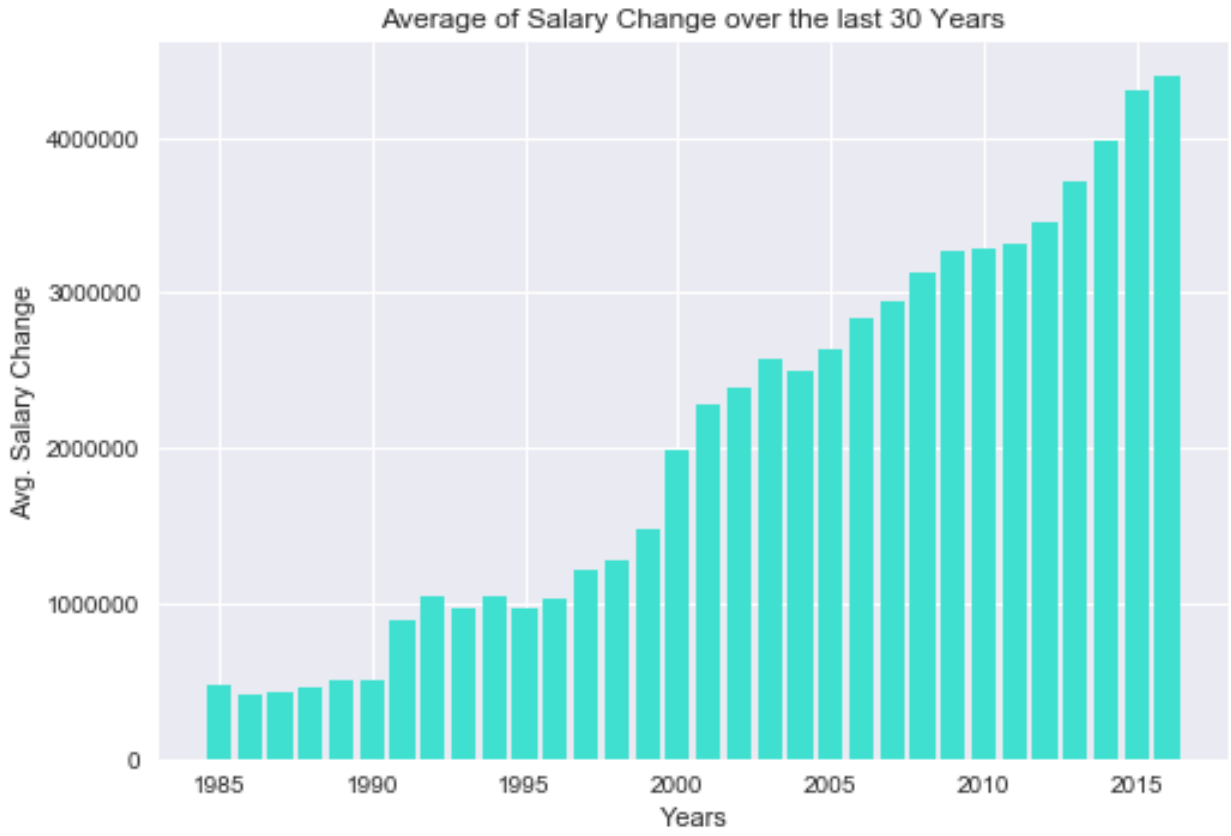
```
salary_mean = salary.groupby('yearID').mean()['salary']
print salary_mean
```

Result:

```
yearID
1985     476299
1986     417147
1987     434729
1988     453171
1989     506323
1990     511973
1991     894961
1992    1047520
1993     976966
1994    1049588
1995     964979
1996    1027909
1997    1218687
1998    1280844
1999    1485316
2000    1992984
2001    2279841
2002    2392526
2003    2573472
2004    2491776
2005    2633830
2006    2834520
2007    2941435
2008    3136517
2009    3277646
2010    3278746
2011    3318838
2012    3458421
2013    3723344
2014    3980445
2015    4301276
2016    4396409
```

Now the average value of salary in every year can be analyzed. But it will be easier to look at the values in visualization:

```
year = salary['yearID'].unique()

plt.bar(year, salary_mean, color = 'turquoise')
plt.xlabel('Years')
plt.ylabel('Avg. Salary Change')
plt.title('Average of Salary Change over the last 30 Years')
```



The result of analyzing the average salary over the years is that they are still increased. From this analyze can be derived a lot of other question, which will not analyzed in the following chapters, like:

- Why is the average increased over the years, why it is not uniform distribution?
- What is the reason for increasing of salary?
- Was a Player in the 1980s a worst player compared to a player of 2015, that there is so a big difference in the mean salary?
- Why had a baseball team now more salary-budget for every player than in the years before? What has changed that the salary budget increased?
- …

2.5 Which players start there career in school and can be found in the Hall of Fame?

In the first step is to analyze if the players from the school data set are completely equal to the players from the hall of fame dataset. For comparing the playerIDs from these two data sets I used the DataFrame.equals()[1]-function:

[1] http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.equals.html

```
#Compare playerID from College playern with playerID from HallofFame
compare = school_playing['playerID'].equals(HallOfFame['playerID'])
print compare
```

The result is False. This means that the playerIDs from both datasets are not completely equal. But it doesn't mean that not some ID's from the school data set can be found in the hall of fame dataset (also in the other direction Hall of Fame → School).

To get only the equal playerIDs from both data sets they have to be merged or joined. I used the merge-function and use the "inner-join" on the playerID. After this the result will be only the players and their data they can be found in both tables.

```
players = school_playing.merge(HallOfFame, how = 'inner', on ='playerID')
print players['playerID'].unique()
print len(players['playerID'].unique())
```

But that is not the end of the analyzing, because some Players can be found two-times or often in the Hall of Fame. To get every player only one time I used the unique-function(). And for getting the count of unique players the len()-function.

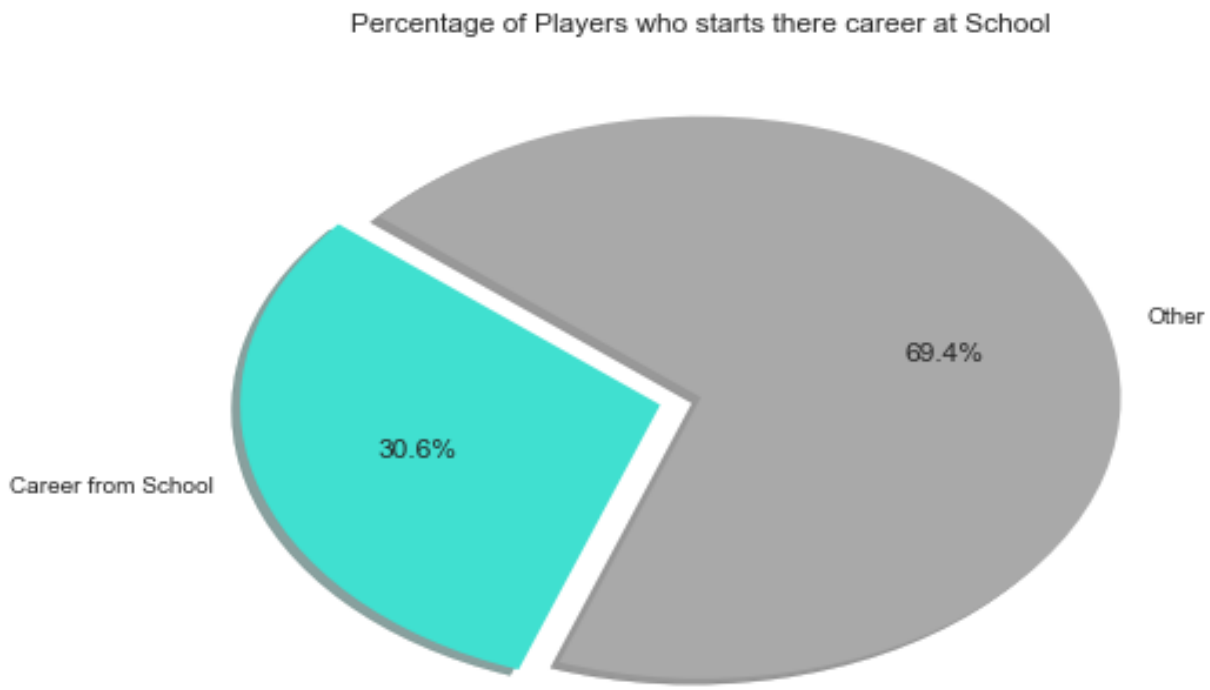
The result was that 386 players they started their career at school are in the hall of fame.

But this is result is not really meaningful, because it is not known how many "unique" players are in the hall of fame. To analyze that I calculated the percentage of the 386 players, and compared it with all the unique hall of fame players. After this I visualized the result in a pie-chart:

```
# Calculate Percentage
career_percentage = (float(len(players['playerID'].unique()))/float(len(HallOfFame['playerID'].unique())))*100
print career_percentage
#30.6349206349

labels = 'Career from School', 'Other'
data = [career_percentage, 100-career_percentage]
color = ['turquoise', 'darkgrey']
plt.pie(data, labels=labels, explode=[0.1,0], colors=color,autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Percentage of Players who starts there career at School')
```

For creating a pie-chart I looked at an example form a website[2]:

[2] https://pythonspot.com/en/matplotlib-pie-chart/



The pie-chart shows that more than 30% of players from the hall of fame start her career in school. This can be an important information for young people they want to start a career as a baseball player. This means that if someone trains a lot when he is young it will be more probable that he will work later as a baseball player.

Also this analysis brings new questions, which will not analyzed in the next chapters, like:

- Where have started the other 69.4 % of players her career? What have they done different?
- Which attributes can be important that a player will be in the Hall of Fame?
- …

2.6 Which states and cities have the most baseball colleges? (Show only Top 10)

In the first step I counted the states or cities with the groupby-function:

```
college_state = school_playing.groupby('state').count()['city']
print college_state
```

→ Gives the count of colleges per state

```
college_city = school_playing.groupby('city').count()['state']
print college_city
```

→ Gives the count of colleges per city

To get only the top 10 of States and cities a function is need that the data sorts and shows only the first top 10. The data must be sorted as descending.

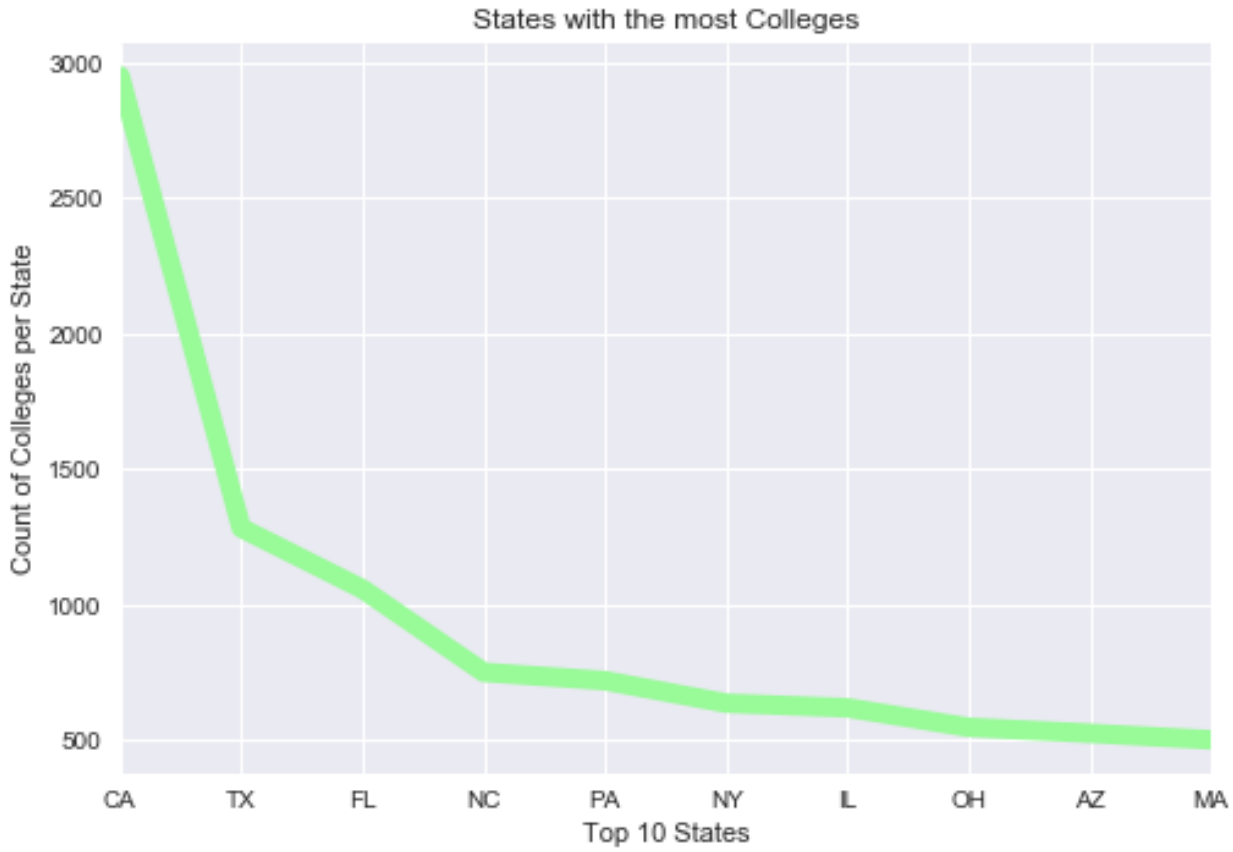
```
def sort_columns(column):
    sorted_column = column.sort_values(ascending = False)
    return sorted_column.iloc[0:10]
```

The iloc[0-10] gives the first 10 values of the sorted column.

To sort the college_state I transferd the data to a Pandas-Series to sort them in the next step.

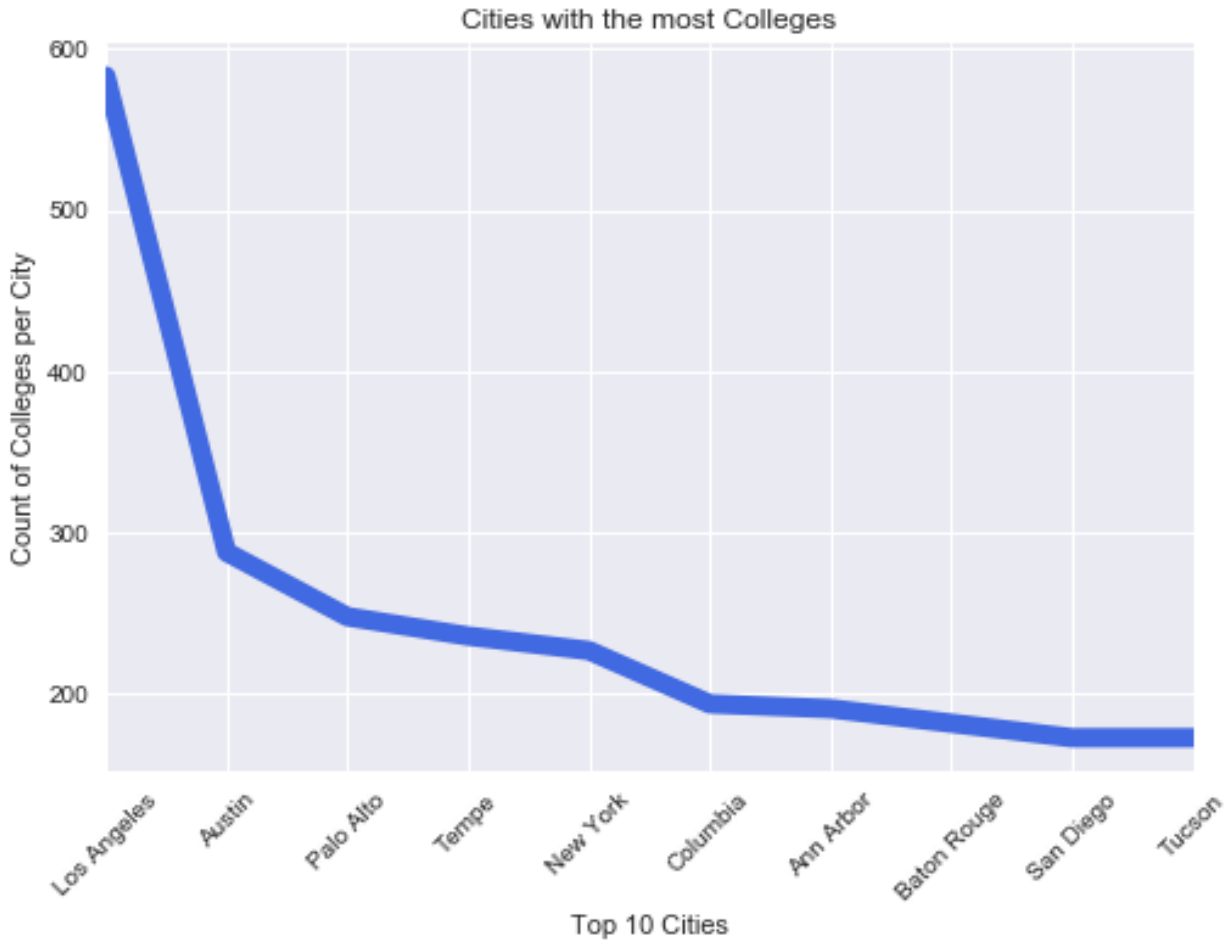
```
# Top 10 States
state_count = pd.Series(college_state)
top10state =  sort_columns(state_count)
print top10state
# Top10 of States
#CA    2948
#TX    1281
#FL    1056
#NC     749
#PA     718
#NY     635
#IL     618
#OH     546
#AZ     524
#MA     500
```

```
top10state.plot(color='PaleGreen', linewidth = 8)
plt.ylabel('Count of Colleges per State')
plt.xlabel('Top 10 States')
plt.title('States with the most Colleges')
```



The same procedure from the college_state will be done with the college_city:

```
# Top10 cities
city_count = pd.Series(college_city)
top10cities = sort_columns(city_count)
print top10cities

# Top 10 of Cities
#Los Angeles    583
#Austin         288
#Palo Alto      248
#Tempe          236
#New York       227
#Columbia       194
#Ann Arbor      191
#Baton Rouge    182
#San Diego      173
#Tucson         173

top10cities.plot(color='Royalblue', linewidth = 8)
plt.xticks(rotation=45)
plt.ylabel('Count of Colleges per City')
plt.xlabel('Top 10 Cities')
plt.title('Cities with the most Colleges')
```



Other options they are equal and also interested for players / managers and so an can be analyzes like that (not analyzed in the next Chapter):

- Which cities are in the Top 10 of a State, for example California?
- Which states or cities have the most successful colleges or teams?
- …

2.7 Reject or retain the null-hypothesis for win and loss of Series post data?

To analyze the win and loss of the Series post I will use a statistical test, to see if the results of a baseball game of the Seriespost.csv Data affect each other.

To check this, the following hypothesis will be looked at:

The null-hypothesis compares if the wins of the Series Post data are equal to the losses:

$$H_0: \mu W = \mu L$$

The alternative hypothesis checks if the wins are smaller than the losses:

$$H_a: \mu W < \mu L$$

The series post data set has a lot of data. To look at a smaller data set I choose the data from the World Series which is marked with “WS” in the round column.

To get this smaller data set I used the SQL-functionality from pandas:

Selected will be only the yearID, which is the unique identifier for the dataset, the round, wins and losses from the Series Post data. To get only the data from the World Series the dataset has to be restricted at the Where-clause.

```
# Create dataframe only with needed columns
from pandasql import sqldf
seriespost = pd.read_csv('SeriesPost.csv')

pysqldf = lambda q: sqldf(q, globals())

# Needed Columns are the yearID, round, wins and losses
# relevant data are only the results of the world series (WS)
ws = pysqldf("SELECT yearID, round, wins, losses FROM seriespost WHERE round == 'WS';")
print ws.head()
```

The pandasql-functionality wasn’t shown in the course; I used the following page to adapt the functionality to the code: https://pypi.python.org/pypi/pandasql

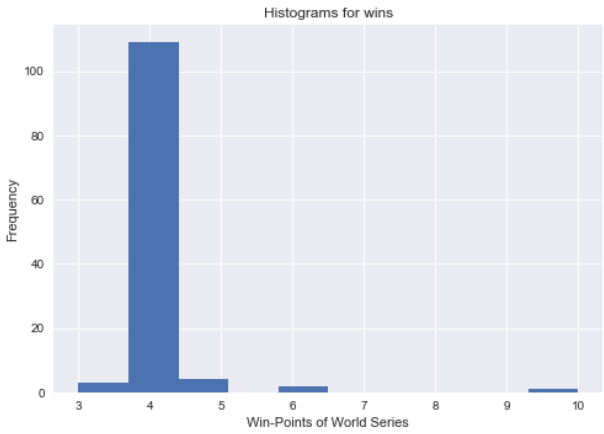
Result of the created variable WS is the following smaller dataset that will be analyzed in the next steps:

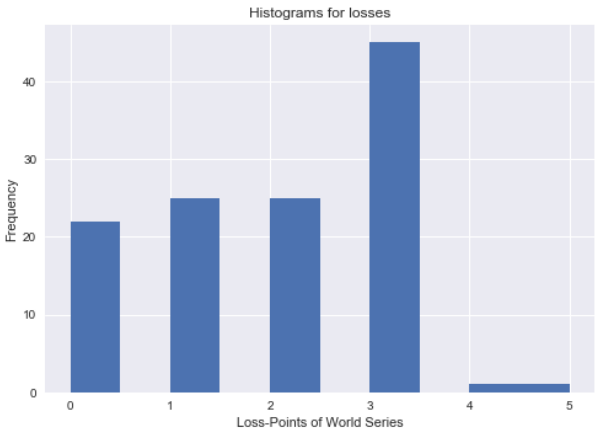
```
   yearID round  wins  losses
0    1884    WS     3       0
1    1885    WS     3       3
2    1886    WS     4       2
3    1887    WS    10       5
4    1888    WS     6       4
```

(only header is shown)

To get a better overview of data I visualized the wins and losses as histograms:

```
# Histograms for win and loss
plt.hist(ws['wins'])
plt.title('Histograms for wins')
plt.xlabel('Win-Points of World Series')
plt.ylabel('Frequency')


plt.hist(ws['losses'])
plt.title('Histograms for losses')
plt.xlabel('Loss-Points of World Series')
plt.ylabel('Frequency')
```

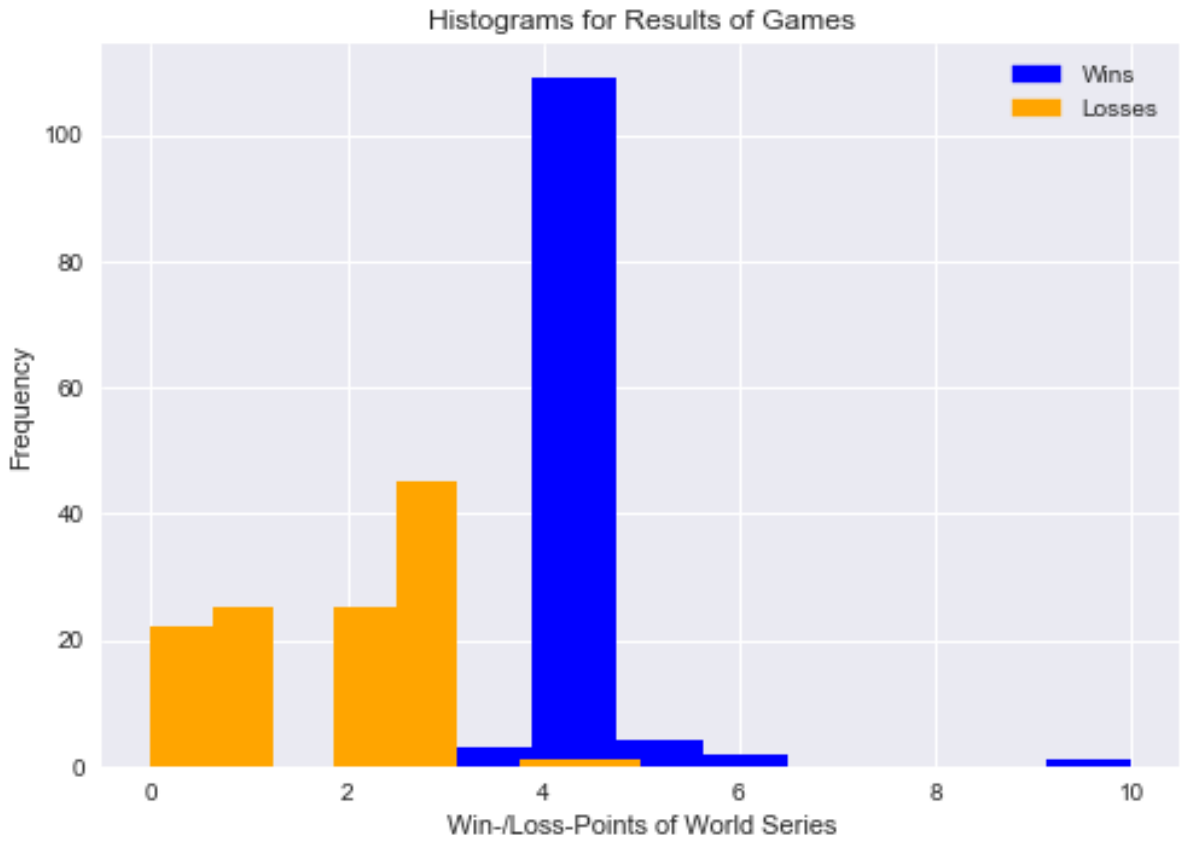




There is visible that the wins are normally distributed, but not in a typical way. Only in some years the World Series will be won with 3 or 5, in the most years (more than 100 years) the World Series was won with 4 points.

The loss-histogram shows a negative skew. Where the most games was lost with 3 points. This happens in more than 40 years.

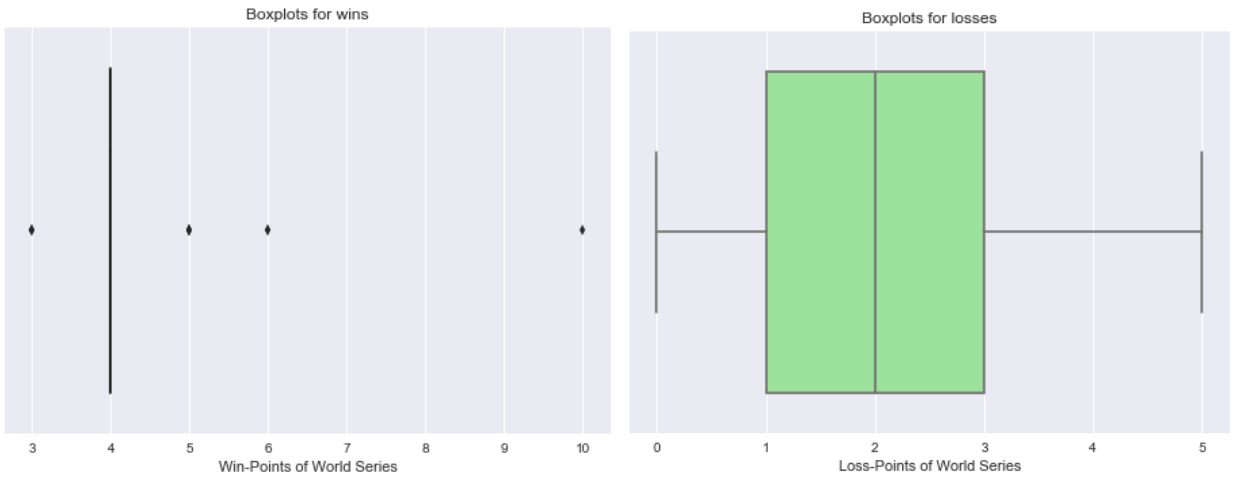
For comparing both distributions the following chart shows how the results differ in frequency of win and losses:



```
import matplotlib.patches as mpatches
#Comparing Histogram of win and losses
plt.hist(ws['wins'], color='blue', bins=8)
plt.hist(ws['losses'], color='orange', bins=8)
plt.xlabel('Win-/Loss-Points of World Series')
plt.ylabel('Frequency')
plt.title('Histograms for Results of Games')
blue = mpatches.Patch(color='blue', label='Wins')
orange = mpatches.Patch(color='orange', label='Losses')
plt.legend(handles=[blue, orange])
#https://matplotlib.org/users/legend_guide.html
```
[3]

To get another view on the data a boxplot can be also very interesting:

```
# boxplots for win and loss
sns.boxplot(ws['wins'], orient='h', color='purple')
plt.xlabel('Win-Points of World Series')
plt.title('Boxplots for wins')

sns.boxplot(ws['losses'], orient='h', color='lightgreen')
plt.xlabel('Loss-Points of World Series')
plt.title('Boxplots for losses')
```



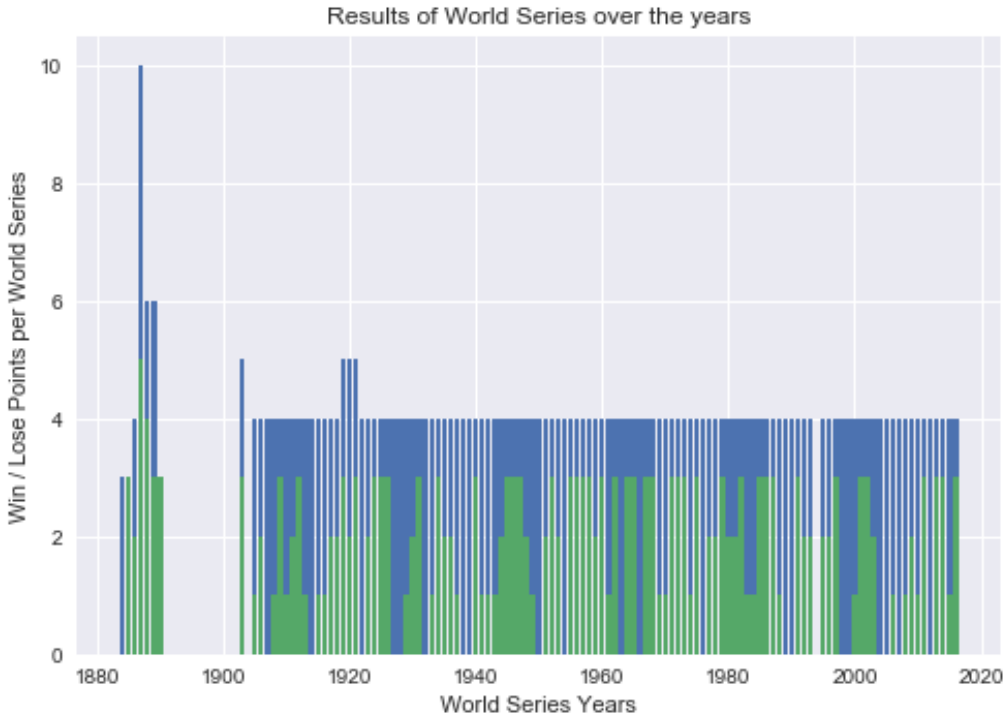
The-wins box-plot shows that the mean is at 4 and at 3, 5 and 10 points it is shown at outliers.

The losses have no outliers, they show that the mean is at 2 and the first quartile is at 1, the third quartile at 3.

To look at the data in a several way I created a bar chart with all world series results beginning in the 1880s and ends in the last year.

[3] Legend guide: https://matplotlib.org/users/legend_guide.html

```
plt.bar(ws['yearID'], ws['wins'])
plt.bar(ws['yearID'], ws['losses'])
plt.xlabel('World Series Years')
plt.ylabel('Win / Lose Points per World Series')
plt.title('Results of World Series over the years')
```



In the bar chart it is visible that the most World Series was won with 4 points only in the 19 century and in the beginning of the 20 century some games wasn't won with 4 points. The distribution of the wins could be names as uniform, because there is in the most years no change. A really interesting question that can be analyzed here is: Why is there no change in winning the World Series. Why it's so seldom to get more or less points than 4?

Also it is really probable that the World Series games in the future will be won with 4 points and loss with less than 4 points.

The curve distribution of the losses is bimodal; there are a lot of ups and downs. This distribution is always under the wins, which makes sense because a Game is only won if one team has more point than the other one.

In the next steps some important statistical values are calculated:

Statistical results:	**Wins**	**Losses**
Mean	4.09	1.84
Median	4.0	2.0
Variance	0.42	1.41
Standard deviation	0.65	1.19
Degrees of freedom	118	118

To check if the null-hypothesis will be rejecting or retained a statistical test is needed.

The alpha-level (α) will be at 0.1 at it is a two tailed test.

The t-value is at 1.646

The statistical t-test was also not shown in the course, I found the following page really helpful: http://hamelg.blogspot.de/2015/11/python-for-data-analysis-part-24.html

The t-test that has be to calculated will be a depending paired t-test because the result of wins of a world series game is depending on the losses, wins should be higher than loss, because a winner is needed to win a world series. And the variables are also paired if a team wins a World Series another team has to lose.

The statistical t-test function that will be used is the ttest_rel, which should be used for depending and paired t-Tests.

```
"""Calculating of depending paried t-Test_____"""

from scipy import stats as st
print st.ttest_rel(ws['wins'], ws['losses'])
# statistic=21.06929892563307, pvalue=8.4287409022462103e-42)
```

The t-test gives as the result that t is 21.07 and p is 8.43. t 21.07 is bigger than the t-value of 1.646 this means p is greater than 0.1 (8.43).

The conclusion of the statistical t-Test is that the null-hypothesis will be rejected.

2.8 Conclusion

The analysis shows that the salary from a baseball player increase over the years. The most salary changes (avg) per player are between 0-10 Mio. and the salary changed in most case 5 to 15 times.
More than 30% of the players that can be found in the Hall of Fame starts there career in school.
The analysis shows also the Top 10 of States and Cities with the most Colleges.

Interesting would be to analyze how the results of these three analyzes are depending from each other. Is a player from the Hall of Fame one of the top salary earners and was in one of the colleges that are in the top 10 of cities / states?

The statistical t-test shows that null-hypothesis will be rejected. This means that the mean of wins and losses of the World Series games are not equal. Losses should be always smaller than wins.

The scope of the analysis is only on the following CSV-Datafiles:

- AllstarFull.csv
- HalloOfFame.csv
- Managers.csv
- Team.csv
- Salaries.csv
- Schools.csv
- CollegePlaying.csv
- SeriesPost.csv

Salary.csv, HallOfFame.csv and Team.csv are used for the Correlation test. Where the chosen attributes shows no extreme correlation, but this doesn't mean that one attribute not causes another. For example the ballot can be influence the rank of a team, because if a team has a lot of good players (which can be found in the Hall of Fame) can influence the rank of a team.

To analyze the Top10 Colleges for States and Cities the Schools.csv and the CollegePlaying.csv was merged to schools_playing.csv. Joins / Merges are an important tool to build data models. They are needed to get the connection between different tables and makes analyzing data much more easier.

The Quality of the data is good; I found not lot things where data wrangling is needed to analyze the data. Some important information's that are missed in the data set are that information's of the players are not available / missing, like height, weight, left- or right-handed and so on. This information's can change the results of analysis a lot, because information's like these can influence the result of baseball games.

Analysis techniques that were used are Group-by functions to slice the data. The Result of Group-by functions shows only the relevant part of a table. This technique is used often in the salary and colleges analysis. With the result of a group-by function it is not possible to look at the data and attributes of the table only at the "selected" part. This can limit the analysis if more attributes getting relevant.
Another technique to analyze the data is done with pandasql where will be also selected a part of the data, but compared with a group by function it is possible to select more or join different data in the query.

Hypothesis test shows if a hypothesis will be statistical reject or not. But the problem of a hypothesis can be that hypothesis will be wrong and the result doesn’t show the needed or expected probability of the statistical test.

3 List of used Web sites

I used the following websites, when functions are needed they are not shown in the courses:

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.equals.html

https://pythonspot.com/en/matplotlib-pie-chart/

https://pypi.python.org/pypi/pandasql

http://hamelg.blogspot.de/2015/11/python-for-data-analysis-part-24.html

https://matplotlib.org/users/legend_guide.html