National Research University Higher School of Economics

Faculty of Computer Science

Homework Project 2018/2019

MSc Program “Data Science”:

Khodyreva Victoria

Shlykova Julia

Moscow 2018

Homework

Group:

- Khodyreva Victoria
- Shlykova Julia

Dataset statistical chart: Statistics For Major League Baseball Players USA 2018

Explanation of the choice:

Data Science and sports are working very closely together. Sport has become a great source of income and when it comes to money, statistics and data analysis become very helpful. Baseball can serve as a good example since first to discover the power of numbers were baseball lovers and this is not an accident. In baseball, the game breaks down into many distinct stages. This distinguishes it from many other team sports. It is much easier to take into account and evaluate events during a baseball game than to follow the chaotic movements of football players across the field. Baseball statistics has become very popular and even received a special name - "Seibermetric", formed from the abbreviated name of the American Baseball Research Society.

We have choosen this dataset, because we find it interesting and perspective to analyze key-features and results of baseball players.

Source: http://www.computerra.ru/86411/sports-bigdata/

Source link:

We copied the report from the link: https://legacy.baseballprospectus.com/sortable/index.php?cid=1931167

From the same website we also parsed some players salaries. They could be found here: https://legacy.baseballprospectus.com/compensation/?cyear=2018&team=&pos=SP

Size of the table: 476 x 19

Entities: Top baseball players of Major League (pitchers).

Features:

1. 'NAME' - Player name.
2. 'TEAM' - As used in most places (including the PECOTA cards), Team is the three letter abbreviation for a major league, minor league, or foreign team.
3. 'LG' - League.'AL' denotes American League. 'NL' denotes National League.
4. 'AGE' - Player's age.
5. 'G' - Games played pitched.
6. 'PITCHES' - Number of pitches thrown.
7. 'AB' - At-bats. Official plate appearances where the batter doesn't walk, get hit by a pitch, hit a recognized sacrifice or is interfered with by the catcher.
8. 'R' - Runs allowed.
9. 'ER' - Earned Runs.
10. 'H' - Hits, or hits allowed.
11. '1B' - Singles Allowed.
12. '2B' - Doubles Allowed.
13. '3B' - Triples Allowed
14. 'HR' - Home runs, or home runs allowed.
15. 'W' - Refers to a pitcher's wins.
16. 'L' - Refers to a pitcher's losses.
17. 'PVORP' - Value over Replacement Player as a pitcher.
18. 'Salary' - Yearly salary of a player.
19. 'Pos' - Player's position.

1-3, 19 – nominal features 4-18 - quantitative features

Examples of problems:

- **Dimentionality reduction:** We have various measurments of player's performance. If we could find a few features that can fully describe our dataset, we could get rid of highly correlated and excessive features.
- **Salary prediction:** We can try to predict salaries for the new players by analysing their performance.
- **Binary classification:** Similarly, we can predict the league (American/National) or player's position (SP/RP).
- **Clustering:** Clustering helps to divide players into smaller groups of similar players. This might help to understand the data better.

In [316]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Loading dataset

In [317]:
```python
data_ages = pd.read_csv('./data_ages.txt', sep='\t')
data_ages = data_ages[['NAME', 'TEAM', 'LG', 'AGE', 'G', 'PITCHES', 'AB', 'R', 'ER', 'H', '1B', '2B'
, '3B', 'HR', 'W', 'L', 'PVORP']]

for col in data_ages.columns[3:]:
    data_ages[col] = data_ages[col].apply(lambda x: x if not isinstance(x, str) else float(x.replace
(',', '.')))
```

In [318]:
```python
salaries = pd.read_csv('./salaries.txt', sep='\t', encoding='cp1251')
salaries.Player = salaries.Player.apply(lambda x: x.strip())
```

In [319]:
```python
data = pd.merge(data_ages, salaries, how='left', left_on='NAME', right_on='Player')

data = data[(~data.Player.isnull()) & (data.Salary>0)]
data = data[data.Pos.isin(['SP', 'RP'])].reset_index(drop=True)
data = data.drop(['Player', 'Team'], axis=1)
data.to_csv('baseball_sal.csv')

data['AVG'] = data['H'] / data['AB']
data['AVG'] = data['AVG'].fillna(0)
```

In [323]:
```python
data.head(10)
```

Out[323]:

	NAME	TEAM	LG	AGE	G	PITCHES	AB	R	ER	H	1B	2B	3B	HR	W	L	PVORP	Salary	Pos	AVG
0	Max Scherzer	WAS	NL	33	33	3503	797	66	62	150	86	36	5	23	18	7	74.1	22142857.0	SP	0.188206
1	Justin Verlander	HOU	AL	35	34	3427	781	63	60	156	92	31	5	28	16	9	70.9	28000000.0	SP	0.199744
2	Aaron Nola	PHI	NL	25	33	3221	756	57	56	149	99	31	2	17	17	6	63.9	573000.0	SP	0.197090
3	Gerrit Cole	HOU	AL	27	32	3257	723	68	64	143	86	36	2	19	15	5	61.5	6750000.0	SP	0.197787
4	Corey Kluber	CLE	AL	32	33	3181	801	75	69	179	118	32	4	25	20	7	58.9	10700000.0	SP	0.223471
5	Blake Snell	TBA	AL	25	31	2925	630	41	38	112	68	27	1	16	21	5	57.7	558200.0	SP	0.177778
6	Patrick Corbin	ARI	NL	28	33	3149	742	70	70	162	104	43	0	15	11	7	57.1	7500000.0	SP	0.218329
7	Trevor Bauer	CLE	AL	27	28	2853	645	51	43	134	93	30	2	9	12	6	54.9	6525000.0	SP	0.207752
8	Chris Sale	BOS	AL	29	27	2525	565	39	37	102	66	22	3	11	12	4	53.9	12500000.0	SP	0.180531
9	Luis Severino	NYA	AL	24	32	3152	726	76	72	173	112	39	3	19	19	8	53.7	604975.0	SP	0.238292

Task 2

Choose 3-6 features, Explain the choice, Apply K-means

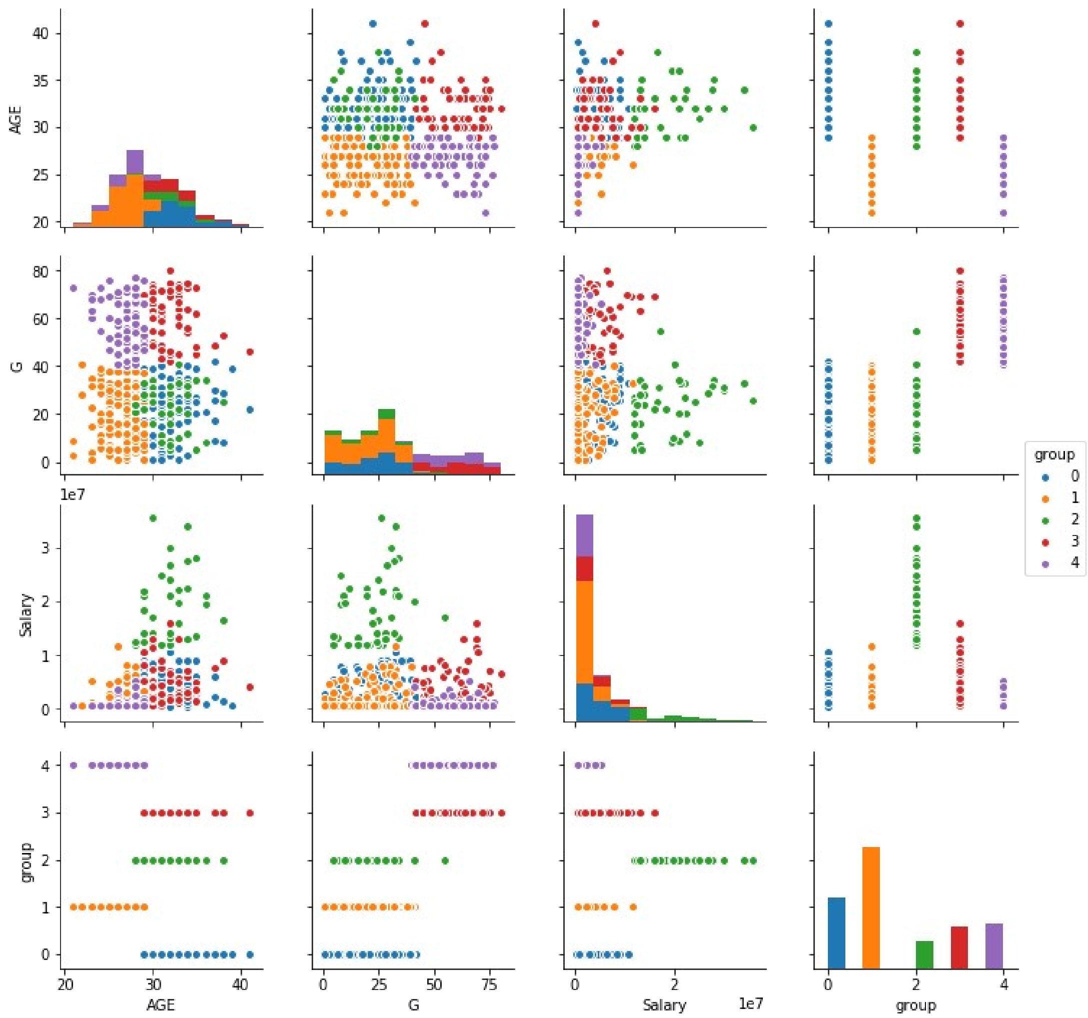
In [7]:
```python
from sklearn.cluster import KMeans
from sklearn import preprocessing
import seaborn as sns
```

In [68]:
```python
import warnings
warnings.filterwarnings("ignore")
```

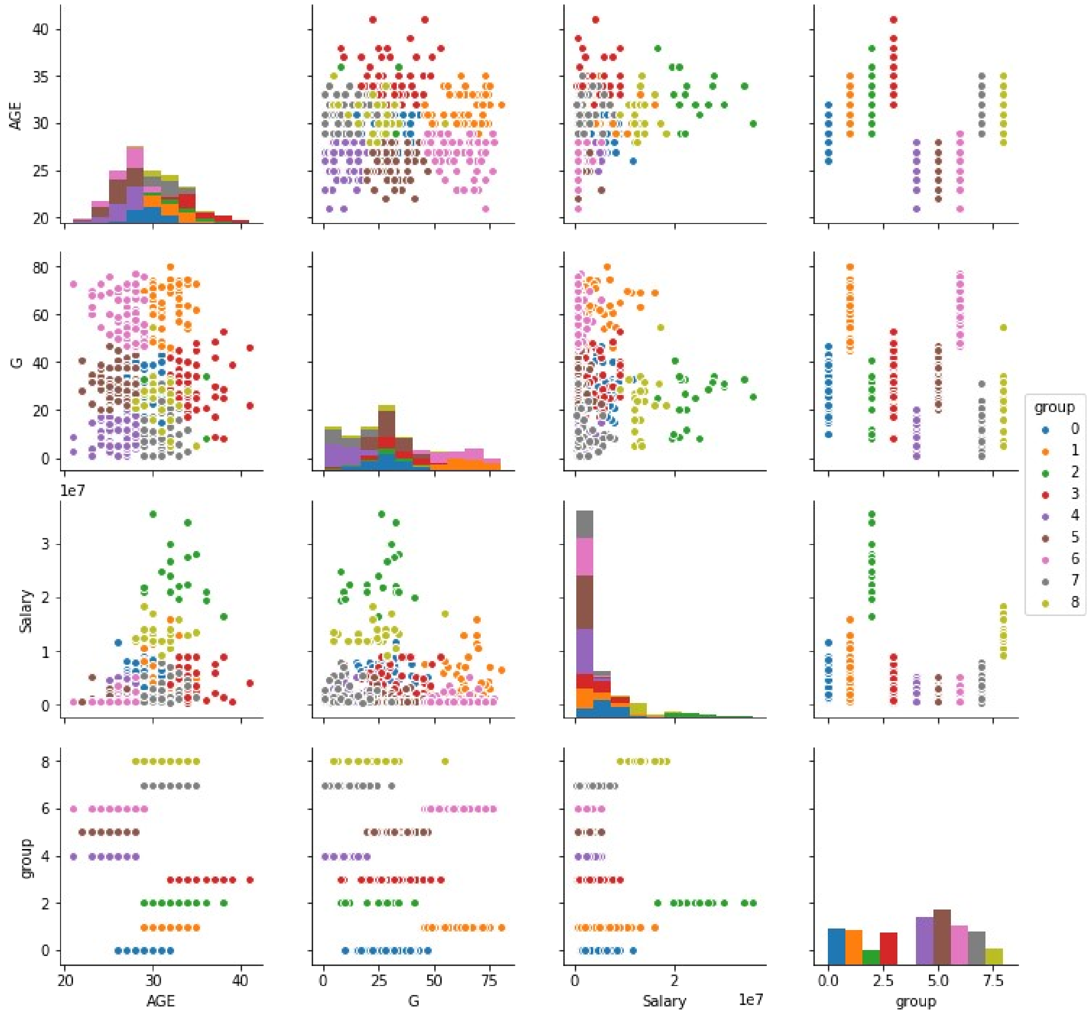
In [361]:

```python
X = data[['AGE', 'G', 'Salary']]
X_scaled = preprocessing.StandardScaler().fit_transform(X)
for n_cl in [5, 9]:
    list_of_scores = []
    possible_rs = range(100, 110)
    for rs in possible_rs:
        model = KMeans(n_clusters=n_cl, random_state=rs)
        model.fit(X_scaled)
        list_of_scores.append(model.score(X_scaled))
    km = KMeans(n_clusters=n_cl, random_state=possible_rs[np.argmax(list_of_scores)])
    clusters = km.fit_predict(X_scaled)
    print('Number of clusters = %s'%n_cl)
    print('Score = %s'%round(km.score(X_scaled), 2))
    X['group'] = clusters
    sns.pairplot(X, hue = 'group')
    plt.show()
```

```
Number of clusters = 5
Score = -427.56
```




```
Number of clusters = 9
Score = -254.64
```



Interpret each found partition by using features from the data table. Explain why you consider one of them better than the other in this perspective.

When the number of clusters is 5, we can see very well clustered groups on the scatter plots. The intuition behind it is simple: first split divides all players into 2 age groups: the younger (<30 y.o.) and the older players (>30 y.o.), second split is over the number of games played during the season: less than 40 and greater than 40 games. Finally, there is a green cluster of old players with little games but high salaries. This cluster is espessially interesting as it shows that experienced players with high salaries usually dont play many games. The result of applying k-means looks informative and clear.

On the other hand, when we tried number of clusters equaled to 9, applying differe[illegible]ndom seeds didn't help to fing good clusters. Most likely 3 features if not enough for that many clusters. However, some groups can still be distinguished, for example, rich players, young players with 0-20 games, older players with many games, ect.

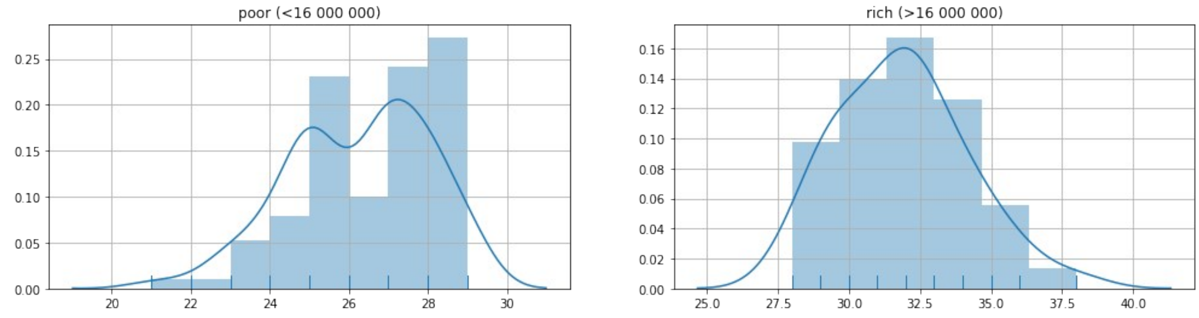
Take one of the partitions

We decided to keep 5 clusters and use green and orange clusters for comparizon. The goal is to test the hypothesis of whether or not the age differes between those groups.

In [560]:
```
poor = np.array(X[X.group == 1]['AGE'])
rich = np.array(X[X.group == 2]['AGE'])
```

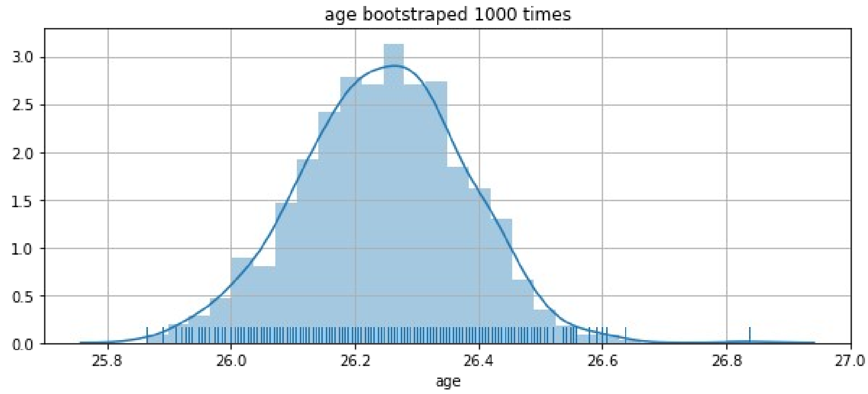
Compare one of the features between two clusters with using bootstrap

In [561]:
```
fig, axis = plt.subplots(1, 2, figsize=(17, 4))
sns.distplot(poor, ax=axis[0], rug=True)
axis[0].set_title('poor (<16 000 000)')
sns.distplot(rich, ax=axis[1], rug=True)
axis[1].set_title('rich (>16 000 000)')
axis[0].grid()
axis[1].grid()
plt.show()
```



The distribution looks more or less normal. This assumption will let us use pivotal bootstrab. Lest apply both pivotal and non-pivotal bootstrap.

In [562]:
```
poor_random = np.random.choice(poor, (len(poor), 1000))
poor_mean = np.mean(poor_random, axis=0)
plt.figure(figsize=(10, 4))
ax = sns.distplot(poor_mean, rug=True)
ax.set_title('age bootstraped 1000 times')
ax.set_xlabel('age')
ax.grid()
```



Next we find the mean of boostraped means and std of boostraped means and compute 95% confidence interval using formula

$$[\mu - 1.965 * \sigma; \mu + 1.965 * \sigma]$$

each application of bootstrap should be done in both, pivotal and non-pivotal, versions

In [563]:
```
def pivotal_boostrap(x):
    print('Using pivotal boostrap:')
    mean = x.mean()
    print('mean = {}'.format(round(mean,3)))
    std = x.std()
    print('std = {}'.format(round(std,3)))
    left = mean - 1.965 * std
    right = mean + 1.965 * std
    print('Confidence interval (95%) = [{};{}]'.format(round(left,3), round(right,3)))

pivotal_boostrap(poor_mean)
```

```
Using pivotal boostrap:
mean = 26.246
std = 0.132
Confidence interval (95%) = [25.987;26.506]
```

Non-pivotal boostrap doesn't use any assumptions about the nature of means. Let's cut 2,5% quantiles to get the confidence interval.

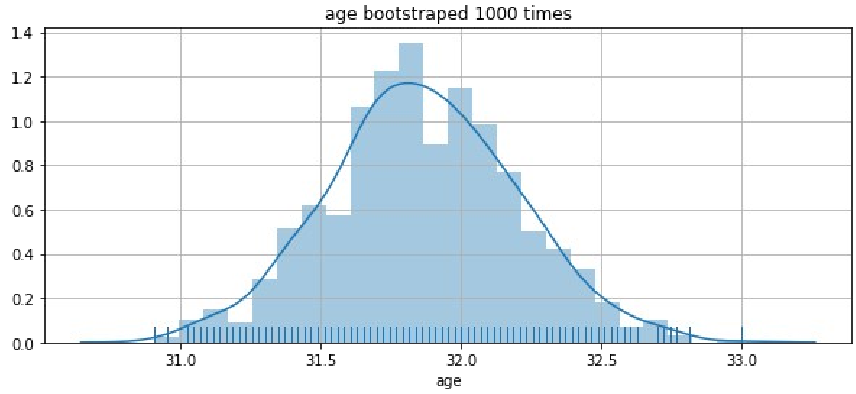
In [564]:
```
def non_pivotal_boostrap(x):
    print('Using non-pivotal boostrap:')
    means_sorted = sorted(x)
    left = means_sorted[round(len(x) * 0.025)]
    right =means_sorted[round(len(x) - len(x) * 0.025)]
    print('Confidence interval (95%) = [{};{}]'.format(round(left,3), round(right,3)))

non_pivotal_boostrap(poor_mean)
```

```
Using non-pivotal boostrap:
Confidence interval (95%) = [25.984;26.495]
```

Pivotal and non-pivotal intervals are very similar.

Let's repeat the computations for rich players

In [565]:
```
rich_random = np.random.choice(rich, (len(rich), 1000))
rich_mean = np.mean(rich_random, axis=0)
plt.figure(figsize=(10, 4))
ax = sns.distplot(rich_mean, rug=True)
ax.set_title('age bootstraped 1000 times')
ax.set_xlabel('age')
ax.grid()
```



In [566]:
```
pivotal_boostrap(rich_mean)
print('-'*10)
non_pivotal_boostrap(rich_mean)
```

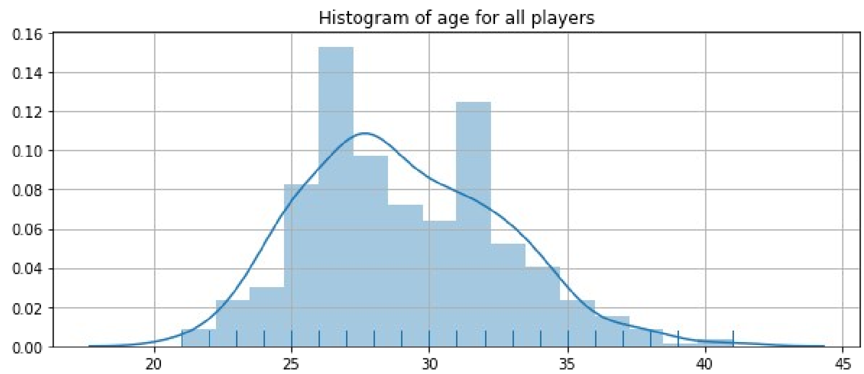
```
Using pivotal boostrap:
mean = 31.863
std = 0.335
Confidence interval (95%) = [31.204;32.522]
----------
Using non-pivotal boostrap:
Confidence interval (95%) = [31.186;32.558]
```

In both pivotal and non-pivotal cases confidence intervals do not intersect. We can conclude that mean age differs for players in clusters 1 and 2.

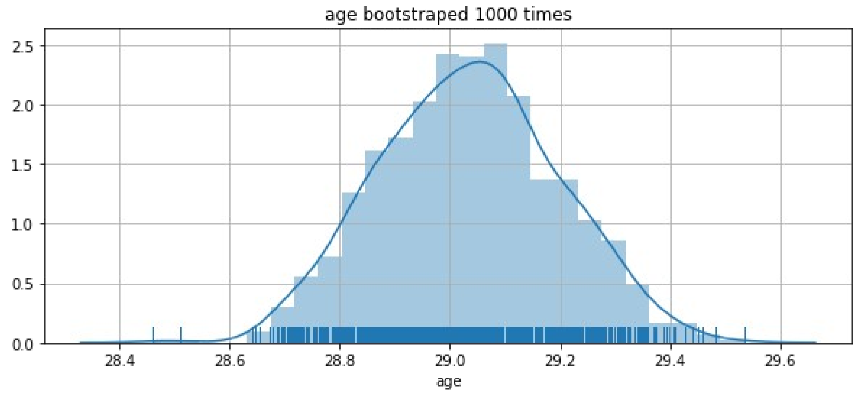
Take a feature, find the 95% confidence interval for its grand mean by using bootstrap

Grand mean:

In [567]:
```
all_ages = np.array(X['AGE'])

plt.figure(figsize=(10, 4))
sns.distplot(all_ages, rug=True)
plt.title('Histogram of age for all players')
plt.grid()
plt.show()
```



In [568]:
```
all_random = np.random.choice(all_ages, (len(all_ages), 1000))
all_mean = np.mean(all_random, axis=0)
plt.figure(figsize=(10, 4))
ax = sns.distplot(all_mean, rug=True)
ax.set_title('age bootstraped 1000 times')
ax.set_xlabel('age')
ax.grid()
```



In [569]:
```
pivotal_boostrap(all_mean)
non_pivotal_boostrap(all_mean)
```

```
Using pivotal boostrap:
mean = 29.034
std = 0.162
Confidence interval (95%) = [28.715;29.353]
Using non-pivotal boostrap:
Confidence interval (95%) = [28.723;29.349]
```

Take a cluster, and compare the grand mean with the within-cluster mean for the feature by using bootstrap

Now we will compare the grand mean with the within-cluster mean of 2nd cluster. This time lets test the hypothesis that means are equal. Lets build the confidence interval for the difference:

$$[\mu_1 - \mu_2 - z_\alpha\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}};\mu_1 - \mu_2 + z_\alpha\sqrt{\frac{\sigma_1^2}{n_1}+\frac{\sigma_2^2}{n_2}}]$$

In [528]:
```
def conf_for_diff(x_1, x_2):
    left = (x_1.mean() - x_2.mean() - 1.965 * np.sqrt(x_1.std() ** 2 / len(x_1) +
                                                       x_2.std() ** 2 / len(x_2)))
    right = (x_1.mean() - x_2.mean() + 1.965 * np.sqrt(x_1.std() ** 2 / len(x_1) +
                                                       x_2.std() ** 2 / len(x_2)))
    print("Confidence interval (95%) = [{};{}]".format(round(left,3), round(right,3)))
```

In [570]:
```
conf_for_diff(all_mean, rich)
```

```
Confidence interval (95%) = [-3.5;-2.153]
```

As 0 doesn't fall into the confidence interval, we reject null hypothesis. Thus, two means are not equal.

Task 4

In your data set, select a subset of 3-6 features related to the same aspect and explain your choice

In this assignment we desided to select features related to the experience of the playres. Those are number of games in 2018, number of pitches, runs and earned runs. We expect that the number of games correlates with the number of all other actions. If player participated in many games he had more chances to complete a run, for example.

In [273]:
```
features = ['G', 'PITCHES', 'R', 'ER']
```

In [274]:
```
X = data[features]
```

Standardize the selected subset; compute its data scatter and determine contributions of all the principal components to the data scatter, naturally and per cent

Next we wil apply standardization with two versions of normalization: over ranges and over std.

In [275]:
```
from sklearn.preprocessing import minmax_scale, StandardScaler
def standartize_over_range(X):
    return pd.DataFrame(minmax_scale(X), columns=X.columns)

def standartize_over_std(X):
    return pd.DataFrame(StandardScaler().fit_transform(X), columns=X.columns)
```

In [276]:
```
X_sc_over_range = standartize_over_range(X)
X_sc_over_std = standartize_over_std(X)
```

Finding the first singular triplet:

In [277]:
```
from numpy.linalg import svd
u, s, vh = np.linalg.svd(X_sc_over_std, full_matrices=True)
s_vector_1 = -u[:, 0]
s_value_1 = s[0]
loadings = -vh[0, :]
print('first singular vector: {} ...'.format(s_vector_1[:5]))
print('loadings: {}'.format(loadings))
```

```
first singular vector: [0.07964623 0.07557086 0.06597482 0.0777303  0.08378534] ...
loadings: [0.13701329 0.56611666 0.57486897 0.57468683]
```

In [278]:
```
data_scatter = np.sum(np.sum(s.dot(s.T)))
print('data scatter: {}'.format(data_scatter))
```

```
data scatter: 1903.9999999999998
```

In [279]:
```
print('Contribution to the data scatter: {:.2f}%'
      .format(100 * s_value_1 ** 2 / data_scatter))
```

```
Contribution to the data scatter: 73.54%
```

Contribution of all components naturally and per cent:

In [280]:
```
for component in range(len(s)):
    s_val = s[component]
    print('Component {}: s_val={:.2f}, contribution={:.2f}%'
          .format(component, s_val, 100 * s_val ** 2 / data_scatter))
```

```
Component 0: s_val=37.42, contribution=73.54%
Component 1: s_val=21.50, contribution=24.28%
Component 2: s_val=6.34, contribution=2.11%
Component 3: s_val=1.23, contribution=0.08%
```

Visualize the data with these features using standardization with two versions of normalization: over ranges and over standard deviations. At these visualizations, use a distinct shape/colour for points representing a pre-specified by you group of objects. Also, apply the conventional PCA for the visualization and see if there is any difference. Comment on which of the normalizations is better and why.

Visualization

Lest add an additional feature for groupping. We have a valiable Pos which stands for position of a player.

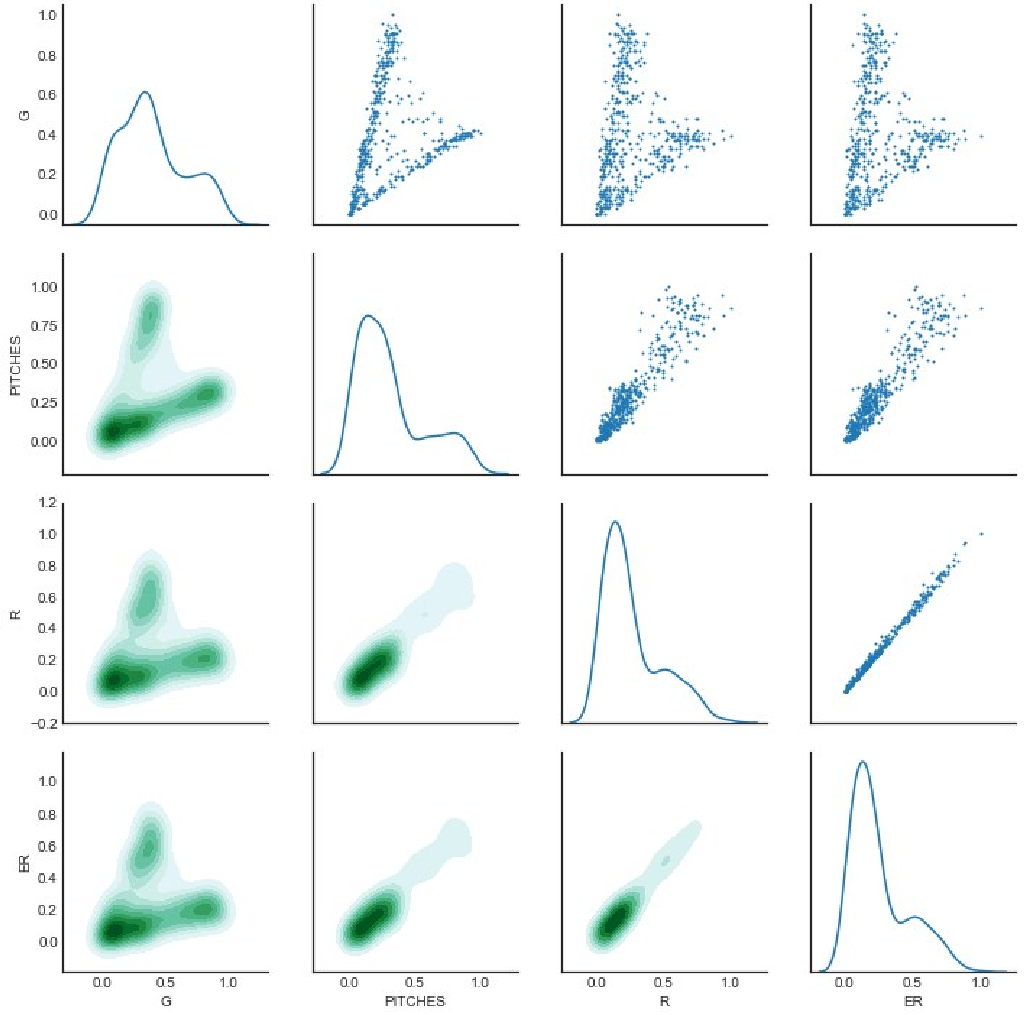
SP = Starting Pitcher. A starting pitcher is a player who first pitches the game and usually pitches the most innings in a game. A starting pitcher also pitches once every five days.

RP = Relief Pitcher. A relief pitcher is a pitcher that relives the pitcher and pitches after the starting pitcher.

In [281]:
```
Pos = data['Pos']

X = data[features]
X_sc_over_range = standartize_over_range(X)
X_sc_over_std = standartize_over_std(X)
```

In [282]:
```
g = sns.PairGrid(X_sc_over_range)
g.map_upper(plt.scatter, s=1)
g.map_lower(sns.kdeplot, shade = True, shade_lowest = False)
g.map_diag(sns.kdeplot)
plt.show()
```

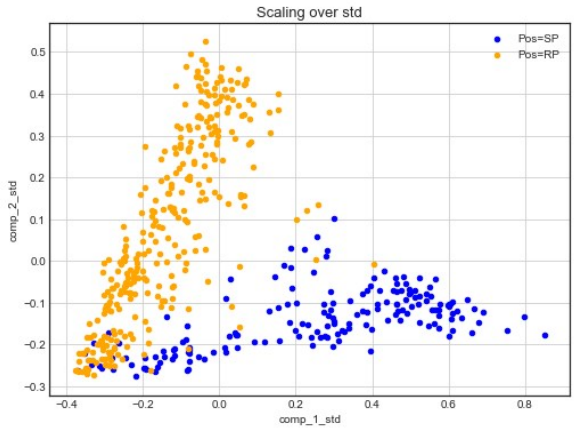
```
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
```

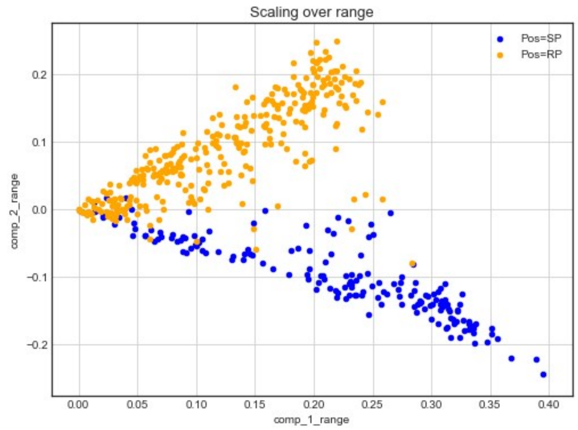


For visualization in 2D we need to compute two first singular triplets.

In [304]:
```
u, s, vh = np.linalg.svd(X_sc_over_std, full_matrices=True)
s_vector_1 = -u[:, 0]
s_vector_2 = -u[:, 1]
s_value_1 = s[0]
s_value_2 = s[1]
z1 = s_vector_1 * np.sqrt(s_value_1)
z2 = s_vector_2 * np.sqrt(s_value_2)
X['comp_1_std'] = z1
X['comp_2_std'] = z2

u, s, vh = np.linalg.svd(X_sc_over_range, full_matrices=True)
s_vector_1 = -u[:, 0]
s_vector_2 = -u[:, 1]
s_value_1 = s[0]
s_value_2 = s[1]
z1 = s_vector_1 * np.sqrt(s_value_1)
z2 = s_vector_2 * np.sqrt(s_value_2)
X['comp_1_range'] = z1
X['comp_2_range'] = z2
X['Pos'] = Pos
```

In [309]:
```
fig, axis = plt.subplots(1, 2, figsize=(18, 6))
X[X.Pos == 'SP'].plot.scatter(x='comp_1_std', y='comp_2_std', color='Blue', label='Pos=SP', ax=axis[
0])
X[X.Pos != 'SP'].plot.scatter(x='comp_1_std', y='comp_2_std', color='Orange', label='Pos=RP', ax=axi
s[0])
axis[0].grid()
axis[0].set_title('Scaling over std', fontsize=13)
X[X.Pos == 'SP'].plot.scatter(x='comp_1_range', y='comp_2_range', color='Blue', label='Pos=SP', ax=a
xis[1])
X[X.Pos != 'SP'].plot.scatter(x='comp_1_range', y='comp_2_range', color='Orange', label='Pos=RP', ax
=axis[1])
axis[1].grid()
axis[1].set_title('Scaling over range', fontsize=13)
plt.show()
```



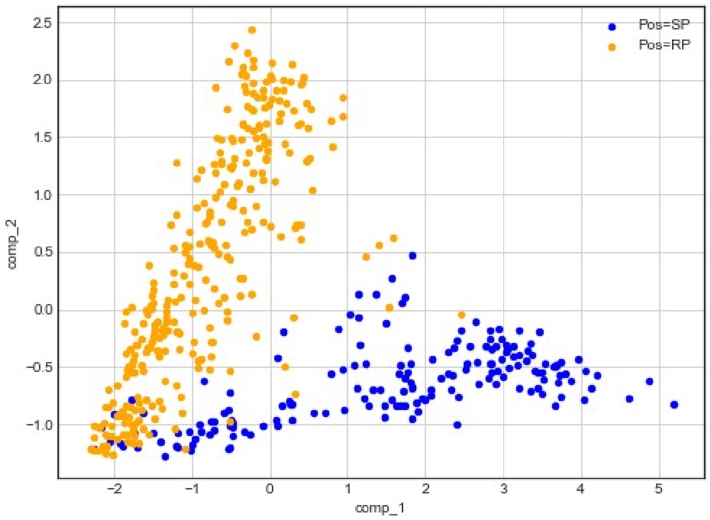


At the first glance there are no noticable differences between two methods of standartization except the positioning. They both help to account for the amount of variabitity in data. Scaling over range is simple to calculate, however, it is less robust than scaling over std since if there are outliers, the distribution will be biased. The std, on the other hand, takes into account to what extent every value is far from the mean.

In our case both methods are sutable for visualization.

Aproach using PCA from sklearn

In [315]:
```
from sklearn.decomposition import PCA

X = data[features]
X_scaled = StandardScaler().fit_transform(X)
pca = PCA(n_components=2)
pca.fit(X_scaled)
X_pca = pd.DataFrame(columns = ['comp_1', 'como_2', 'Pos'])
X_pca['comp_1'], X_pca['comp_2'] = pca.transform(X_scaled).T
X_pca['Pos'] = data['Pos']


ax = X_pca[X_pca.Pos == 'SP'].plot.scatter(x='comp_1', y='comp_2', color='Blue', label='Pos=SP', fig
size=(8, 6))
X_pca[X_pca.Pos != 'SP'].plot.scatter(x='comp_1', y='comp_2', color='Orange', label='Pos=RP', ax=ax)
plt.grid()
plt.show()
```



And again we get a very similar result.

Compute and interpret a hidden factor behind the selected features. The factor should be expressed in a 0-100 rank scale (as well as the features)

Lets rescale all features to the 0-100 scale.

In [209]:
```
X = data[features]
for col in X.columns:
    X[col] = X[col] / X[col].max() * 100
```

In [210]:
```
u, s, vh = np.linalg.svd(X, full_matrices=True)
loadings = -vh[0, :]
print('loadings: {}'.format(loadings))
```

```
loadings: [0.56772582 0.53979628 0.44664979 0.43221677]
```

Determining α (scaling factor):

In [211]:
```
alpha = 1 / np.sum(loadings)
print('scaling factor: {:.3f}'.format(alpha))
```

```
scaling factor: 0.503
```

the PCA hidden factor score vector:

In [212]:
```
hidden_factor = X.values.dot(loadings) * alpha
print('hidden factor: {} ...'.format(hidden_factor[:10]))
```

```
hidden factor: [62.46239195 61.31285432 57.52309037 60.93118397 62.90052082 48.26822735
 61.92263086 49.38798164 43.18614467 63.05429339] ...
```

In [213]:
```
X['hidden_factor'] = hidden_factor
X.head()
```

Out[213]:

	G	PITCHES	R	ER	hidden_factor
0	41.25	100.000000	53.658537	52.542373	62.462392
1	42.50	97.830431	51.219512	50.847458	61.312854
2	41.25	91.949757	46.341463	47.457627	57.523090
3	40.00	92.977448	55.284553	54.237288	60.931184
4	41.25	90.807879	60.975610	58.474576	62.900521

In [272]:
```
np.array([100, 100, 100, 100]).dot(loadings) * alpha
```

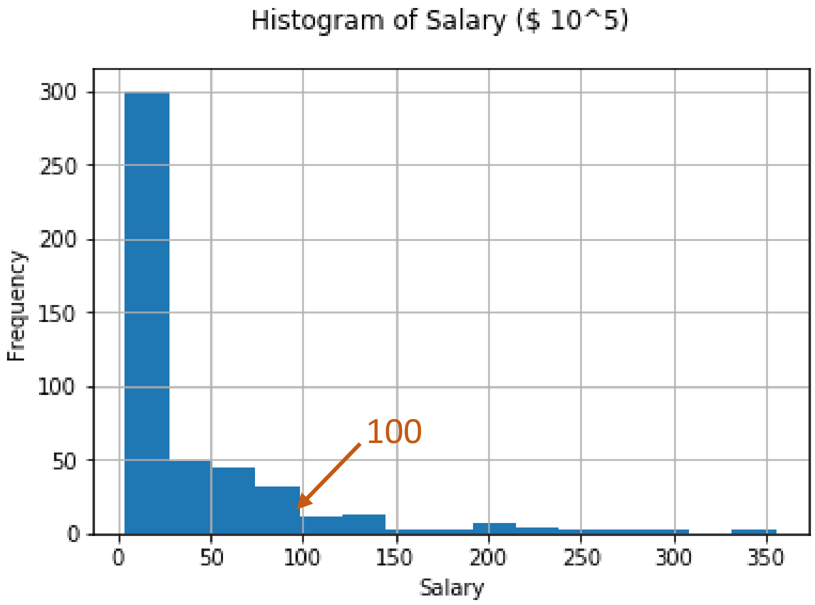
Out[272]:
```
100.0
```

Task 3. Contingency Table

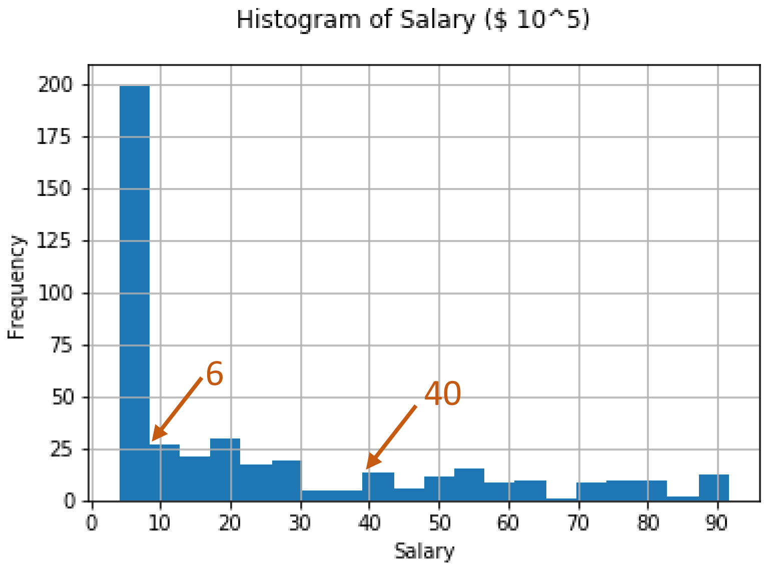
For the tasks 3 and 5 code will be represented in Appendix.

The first nominal feature will be taken from our data – player's position, which determine either Starting Pitcher (SP) or Relief Pitcher (RP).

For choosing another two nominal features let's at first consider histogram of feature "Salary":



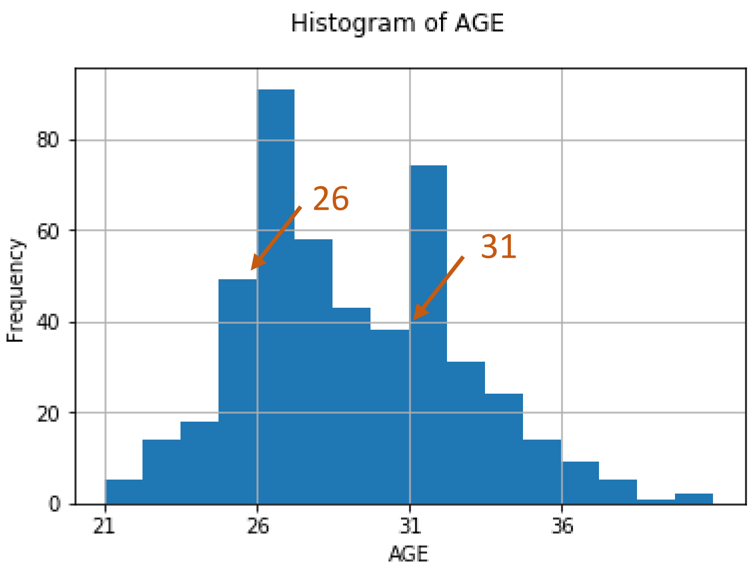
According to the histogram there is a marked incomes differentiation. It will be wise to consider one of the category as very rich (>= $10,000,000). Furthermore, it seems that the most part of our data is below $10,000,000. Let's see a bit closer to histogram for this part:



There is also the incomes differentiation: we can select our first category as poor (<\$600,000). The average of players' salaries is \$4,000,000: so we can define the second and the third categories of our data. The table of salary categories is presented below:

Conditions for Salary selecting category	**nom_Sal**
Salary < \$600,000	6-
\$600,000 <= Salary < \$4,000,000	6+
\$4,000,000 <= Salary < \$10,000,000	40+
\$10,000,000<= Salary	100+

It will be interesting to see what influence of age is on Salary and Position. So for the third nominal feature let's consider a feature 'AGE':



We may consider our bounds as points before picks: {26,31}. We obtain the next categories for the feature:

Conditions for Age selecting category	**nom_AGE**
Age < 26	26-
26 <= Age < 31	26+
31 <= Age	31+

Thus, we have three nominal features: Pos, nom_Sal, nom_AGE. Our task is to build two contingency tables over them: conditional frequency table and Quetelet relative index table.

Quetlet index: $q_{kv}=p(v|k)/p_v -1 = \frac{p_{vk}}{p_k p_v} - 1$ - shows the relative change of probability of v under condition of S_k (from the average).

It would be interesting what influence Position of a player exerts on Salary. Thus, we consider conditional probability of such level of Salary, given players' position.

Salary – Position

Conditional frequency table P(Salary|Pos), P(Age|Pos),

nom_Sal	RP	SP	All
100+	0,042484	0,217647	0,105042
40+	0,183007	0,264706	0,212185
6+	0,369281	0,170588	0,298319
6-	0,405229	0,347059	0,384454
All	1	1	1

Quetelet relative index

nom_Sal	RP	SP	P(Salary)
100+	-0,59556	1,072	0,105042
40+	-0,13751	0,247525	0,212185
6+	0,237872	-0,42817	0,298319
6-	0,054038	-0,09727	0,384454

Age – Position

nom_AGE	RP	SP	All
26+	0,496732	0,458824	0,483193
26-	0,140523	0,252941	0,180672
31+	0,362745	0,288235	0,336134
All	1	1	1

nom_AGE	RP	SP	P(Age)
26+	0,028019	-0,0504	0,483193
26-	-0,22222	0,4	0,180672
31+	0,079167	-0,1425	0,336134

According to the conditional frequency table we may conclude that players at position RP more frequent have Salary below $600,000 and are at age between 26 and 31 years. For SP it's also more frequent salary below $600,000 and age between 26 and 31 years.

According to the "Quetelet relative index table" it seems that Salary (100+), given a position Starting Pitcher, is 107% more frequent than average. This event also may be seen in the articles, where the highest-paid players are at a position 'Starting Pitcher'. [https://www.businessinsider.com/chart-mlbs-highest-paid-positions-2014-7]. We didn't see it at Conditional frequency table. It's also surprising that Age below 26 years (26-), given SP is 40% more frequent than average.

Summary Quetelet index is computed by the next formula:

$$Q = \sum_{k=1}^{K} \sum_{l=1}^{L} p(Hk \cap Gl) q(Hk|Gl) = \sum_{k=1}^{K} \sum_{l=1}^{L} \frac{p2(Hk \cap Gl)}{p(Hk)p(Gl)} - 1$$

Thus, Summary Quetelet index is the inner product of two tables, that of co-occurrence and Quetelet. In our case, Q = 0.1067, what means that on average knowledge of Position "adds" 10.67% to frequency of Salary. It seems that our features are not independent. Let's check the rule for independence: Two features are independent if and only if for all k,l: $p(Hk \cap Gl) = p(Hk)P(Gl)$

$p(nom\_Sal \cap Pos)$

nom_Sal	RP	SP	All
100+	0,027311	0,077731	0,105042
40+	0,117647	0,094538	0,212185
6+	0,237395	0,060924	0,298319
6-	0,260504	0,12395	0,384454
All	0,642857	0,357143	1

$P(nom\_Sal)P(Pos)$

nom_Sal	RP	SP	All
100+	0,067527	0,037515	0,105042
40+	0,136405	0,07578	0,212185
6+	0,191777	0,106543	0,298319
6-	0,247149	0,137305	0,384454
All	0,642857	0,357143	1

$P(nom\_AGE \cap Pos)$

nom_AGE	RP	SP	All
26+	0,319328	0,163866	0,483193
26-	0,090336	0,090336	0,180672
31+	0,233193	0,102941	0,336134
All	0,642857	0,357143	1

$P(nom\_AGE)P(Pos)$

nom_AGE	RP	SP	All
26+	0,310624	0,172569	0,483193
26-	0,116146	0,064526	0,180672
31+	0,216086	0,120048	0,336134
All	0,642857	0,357143	1

Difference 0.237 - 0.192=0.045, is high; Salary '6+' & RP co-occur more frequently than when being independent, we can suppose a positive relation. It's difficult to conclude about how strong correlation is: 4 entries differ by more than 0.04 (highlighted). We should check a hypothesis that the features (Salary and Position) are independent in the population.

However, it isn't seen a really big difference between P(nom_AGE∩Pos) and P(nom_AGE)P(Pos). These values differ by more than even 0,03. So, we can suppose that features (Age and Position) are independent.

Pearson's Chi-Squared is conducted by the next formula:

$$X^2 = \sum_{k=1}^{K} \sum_{l=1}^{L} \frac{\left(p(Hk \cap Gl) - p(Hk)p(Gl)\right)^2}{p(Hk)p(Gl)}$$

Pearson's Chi-Squared coincides with Q : $X^2 = Q$. Under the hypothesis that the features are independent in the population, and entity sampling has been done randomly and independently, the density function of random variable NX^2 tends to distribution χ^2 with f=(K-1)(L-1) degrees of freedom.

In our case, K = 4, L = 2 therefore f = 8. There is a 5% chance that the NX^2 value will be greater than 15.51, if the hypothesis of independence is true. Q = 0.1067, N = 476, then NX^2 = 50.7892 > 15.51. The hypothesis of independence of features 'Salary' and 'Position' has to be rejected with 95% confidence and even 99% confidence since 50.7892 > 20.1.

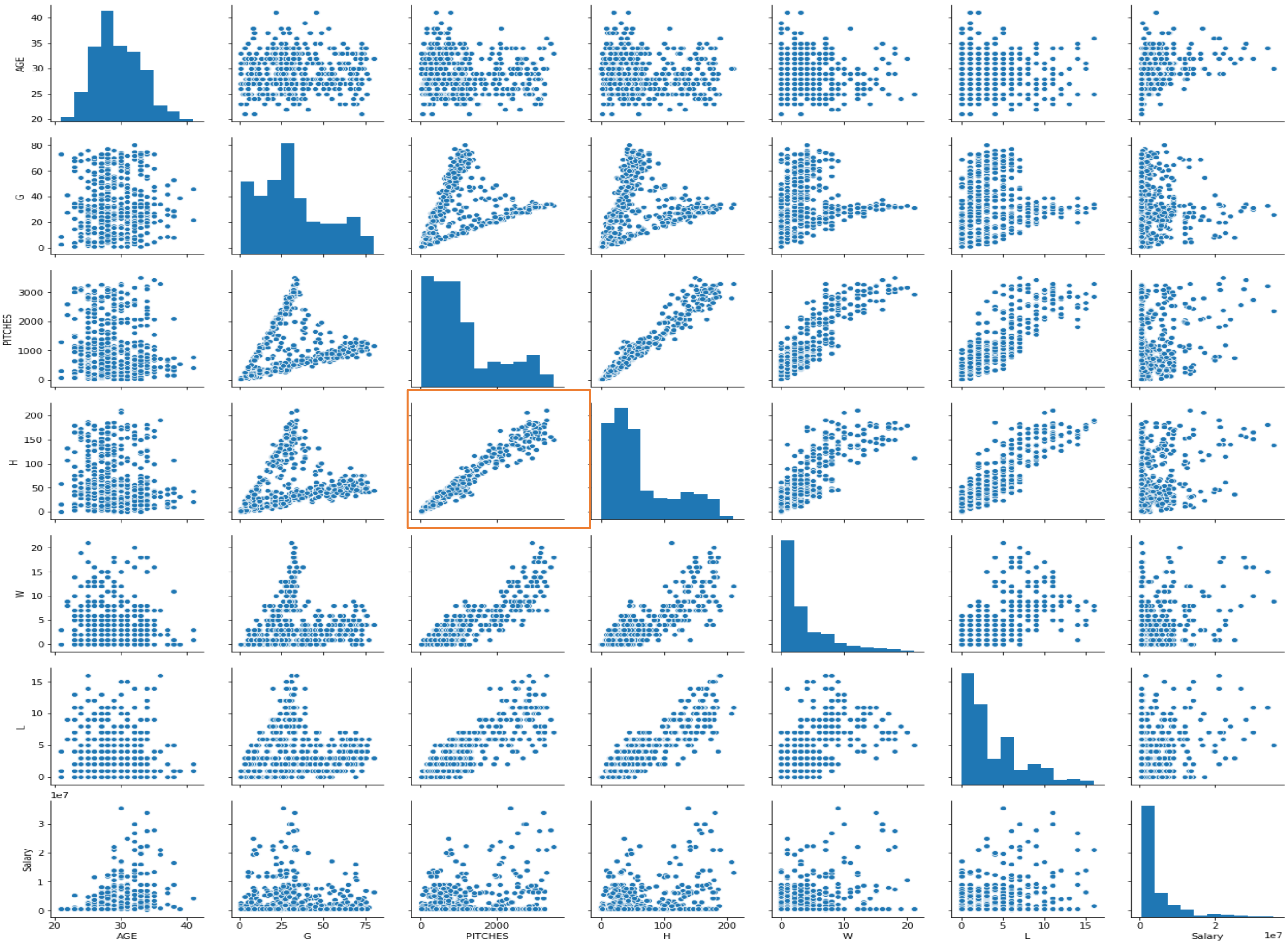
For Age and Position: K = 3, L = 2 then f = 6. There is a 5% chance that the NX^2 value will be greater than 12.592, if the hypothesis of independence is true. Q = 0.0205, N = 476, then NX^2 = 9.7744 < 12.592. The hypothesis of independence of features 'Age' and 'Position' shouldn't be rejected with 95% confidence and of course with 99% confidence since 9.7744 < 16.812. Our suppositions were correct.

Numbers of observations would suffice to see the features as associated at 95% confidence level will be: N > 15.51/0.1067= 145.36. That means it's sufficient for N to be at least 146 observations; For 99% confidence level: N > 20.1/0.1067= 188.379. Then N = 189.

For independence of Age and Position: N <12.592/9.7744= 1.3 at 95% confidence level. And N < 1.72. N =1.

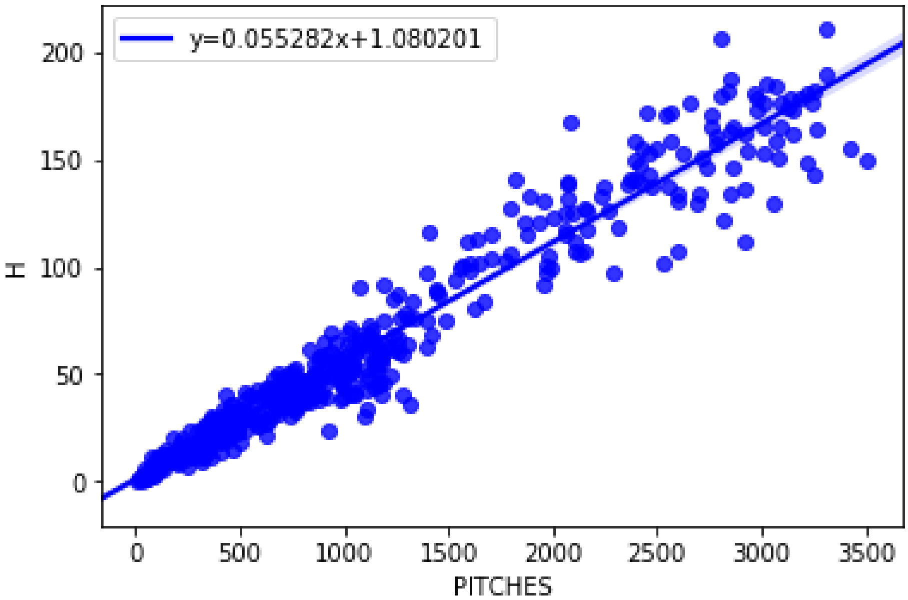
Task 5. Regression

1. Find two features in your dataset with more or less “linear-like” scatterplot.



It’s clearly seen that the most appropriate “linear-like” scatterplot is scatterplot between Pitches (number of pitches thrown) and H (hits allowed by pitcher). This dependence can be explained very easily: the more pitches thrown – the more hits can be made by a batter. We will analyze the dependence of allowed hits on number of pitches. So dependent variable will be Y = H and a factor variable X = Pitches.

2-3. Display the scatter-plot. Build a linear regression of one of the features over the other. Make a comment on the meaning of the slope.



The slope a = 0.055282 > 0 what means positive dependence (it's well seen in the figure). On average if a pitcher throws 100 balls (X increase by 100) then a batter will make 5 hits (Y will increase by 5).

4. Find the correlation and determinacy coefficients, and comment on the meaning of the latter.

The correlation coefficient is determined by the formula:

$\boldsymbol{\rho_{vw}} = \frac{1}{\sigma_v \sigma_w} \sum_{i=1}^{N} (x_{iv} - \overline{x_v})(x_{iw} - \overline{x_w}) / N$, where $\boldsymbol{\sigma_v}$, $\boldsymbol{\sigma_w}$ - standard deviations

This coefficient measures the strength and the direction of a linear relationship between two variables.

In our case ***corr(Pitches, H) = 0.97153***. The value is really close to 1 which means a strong positive linear relationship between chosen features.

The determinacy coefficient is the squared correlation coefficient: $\boldsymbol{\rho^2}$. This coefficient gives the information about goodness of fit for the observations (how good our model explains data). The higher coefficient – the more points the line passes through. More specifically, R2 indicates the proportion of the variance in the dependent variable (Y) explained by linear regression and the independent variable (X).

R^2 = 0.94387 indicates that 94% of the variation in the H is predictable from Pitches. 5.6% left unexplained.

5. Make a prediction of the target values for given two or three predictors' values; make a comment.

For a prediction of the target values we use our linear model: y= 0.055282*x+ 1.080201 and compute prediction error: $\frac{y* - (ax+b)}{y*}$

Table of prediction

n	PITCHES	Observed Hits	Predicted Hits	Error,%
1	3427.0	156.0	190.530844	22.135156
10	3221.0	181.0	179.142798	-1.026078
100	903.0	57.0	50.999644	-10.526941

Despite of the high level of the prediction ability of the model shown in the previous task. There are still big errors. And the average error: 11.23 %. It's not too high but above acceptable level (10%). However, number of our observations is high, and this little part may be predicted badly.

6. Compare the mean relative absolute error of the regression on all points of your set and the determinacy coefficient and make comments

Relative error: $\Delta(i)=|y_i-ax_i-b|/|y_i|$. Mean relative absolute error: $\Sigma_i\Delta(i)/N$

The mean relative absolute error,%	**The determinacy coefficient**
0.0095	***0.94387***

As we may see MRAE is small and in the same time R^2 is very high. We can conclude that our model perfectly describes the dependence of Pitches on allowed Hits.

homework_3

December 2, 2018

#
Appendix
#
Homework 3: Contingency Table

In [69]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
data = pd.DataFrame.from_csv('baseball_sal.csv')
data.head()
```

Out[69]:
```
                NAME TEAM  LG  AGE   G  PITCHES   AB   R  ER    H   1B  2B  3B  \
0       Max Scherzer  WAS  NL   33  33     3503  797  66  62  150   86  36   5
1   Justin Verlander  HOU  AL   35  34     3427  781  63  60  156   92  31   5
2         Aaron Nola  PHI  NL   25  33     3221  756  57  56  149   99  31   2
3        Gerrit Cole  HOU  AL   27  32     3257  723  68  64  143   86  36   2
4       Corey Kluber  CLE  AL   32  33     3181  801  75  69  179  118  32   4

   HR   W  L  PVORP      Salary Pos
0  23  18  7   74.1  22142857.0  SP
1  28  16  9   70.9  28000000.0  SP
2  17  17  6   63.9    573000.0  SP
3  19  15  5   61.5   6750000.0  SP
4  25  20  7   58.9  10700000.0  SP
```

In [7]:
```
data['Salary/10^5'] = data['Salary']/100000
```

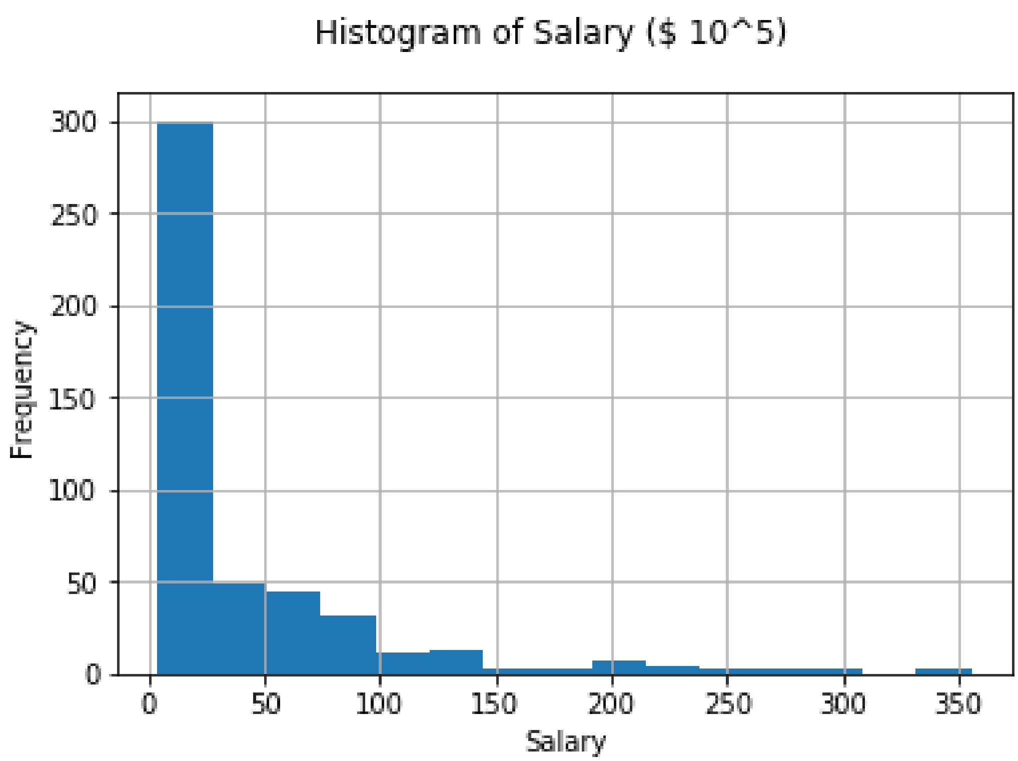
In [8]:
```
data['Salary/10^5'].describe()
```

Out[8]:
```
count    476.000000
mean      40.127554
std       57.100599
min        4.032260
25%        5.565000
50%       15.000000
75%       53.000000
max      355.714290
Name: Salary/10^5, dtype: float64
```

```
In [66]: hist_sal = np.histogram(data['Salary/10^5'], bins = 15)
         print(hist_sal)
         import pylab as pl
         axarr = data['Salary/10^5'].hist(bins = 15)
         pl.suptitle("Histogram of Salary ($ 10^5)")
         axarr.set_xlabel("Salary")
         axarr.set_ylabel("Frequency")
         pl.xticks(np.arange(0, 375, 50))
```

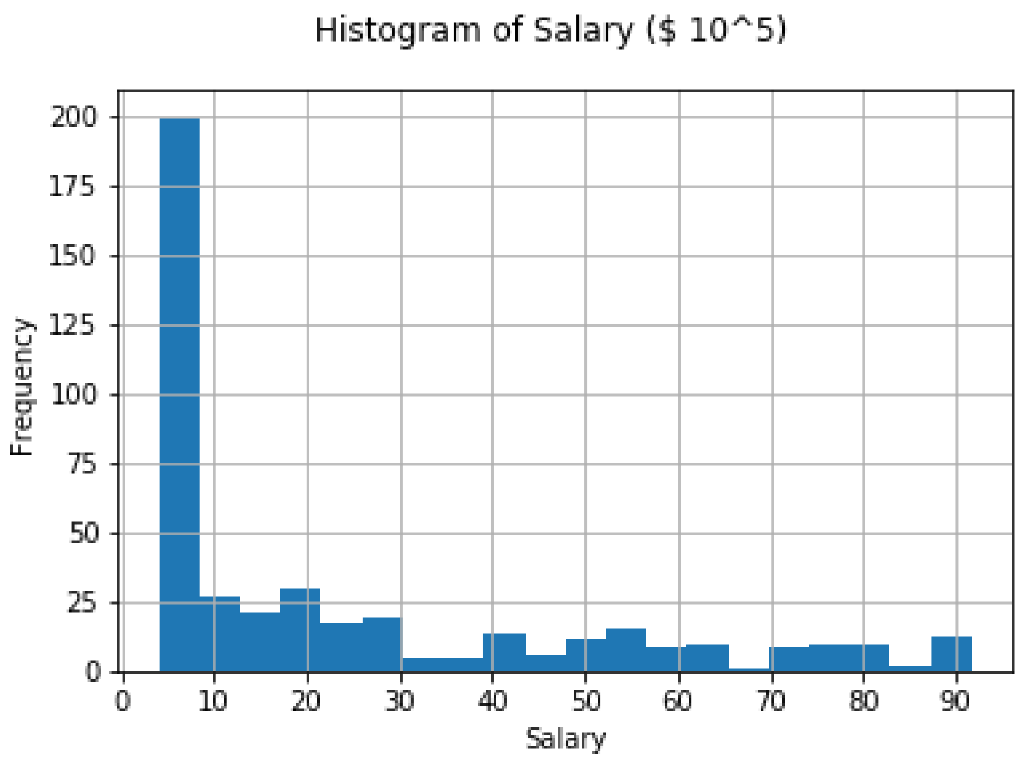
```
(array([300,  49,  45,  32,  11,  13,   3,   3,   7,   4,   2,   3,   2,
          0,   2], dtype=int64), array([  4.03226   ,  27.47772867,  50.92319733,  74.368666  ,
        97.81413467, 121.25960333, 144.705072  , 168.15054067,
       191.59600933, 215.041478  , 238.48694667, 261.93241533,
       285.377884  , 308.82335267, 332.26882133, 355.71429   ]))
```

```
Out[66]: ([<matplotlib.axis.XTick at 0x53769b0>,
           <matplotlib.axis.XTick at 0x5376320>,
           <matplotlib.axis.XTick at 0x5376208>,
           <matplotlib.axis.XTick at 0x53ba0f0>,
           <matplotlib.axis.XTick at 0x53ba518>,
           <matplotlib.axis.XTick at 0x53baa58>,
           <matplotlib.axis.XTick at 0x53bae80>,
           <matplotlib.axis.XTick at 0x7dc2518>],
          <a list of 8 Text xticklabel objects>)
```



In [13]:
```
data_sal_less100 = data[data['Salary/10^5']<100]
hist_sal_less100 = np.histogram(data_sal_less100['Salary/10^5'], bins =20)
axarr_less100 = data_sal_less100['Salary/10^5'].hist(bins = 20)
pl.suptitle("Histogram of Salary ($ 10^5)")
axarr_less100.set_xlabel("Salary")
axarr_less100.set_ylabel("Frequency")
pl.xticks(np.arange(0, 100, 10))
```

Out[13]:
```
([<matplotlib.axis.XTick at 0x98b9ac8>,
  <matplotlib.axis.XTick at 0x98b9400>,
  <matplotlib.axis.XTick at 0x98b92e8>,
  <matplotlib.axis.XTick at 0x9609a90>,
  <matplotlib.axis.XTick at 0x9615128>,
  <matplotlib.axis.XTick at 0x9615588>,
  <matplotlib.axis.XTick at 0x9615a58>,
  <matplotlib.axis.XTick at 0x9615f28>,
  <matplotlib.axis.XTick at 0x961a438>,
  <matplotlib.axis.XTick at 0x961a908>],
 <a list of 10 Text xticklabel objects>)
```



```
In [14]: mean_sal = data['Salary'].mean()
         print(mean_sal)
```

```
4012755.4285714286
```

```
In [15]: k1 = 6*10**5
         k2 = 40*10**5
         k3 = 10**7
```

```
In [16]: def categ(x):
             if x < k1:
                 return '6-'
             if k1 <= x < k2:
                 return '6+'
             if k2 <= x < k3:
                 return '40+'
             else:
                 return '100+'
         data['nom_Sal'] = data['Salary'].apply(categ)
```

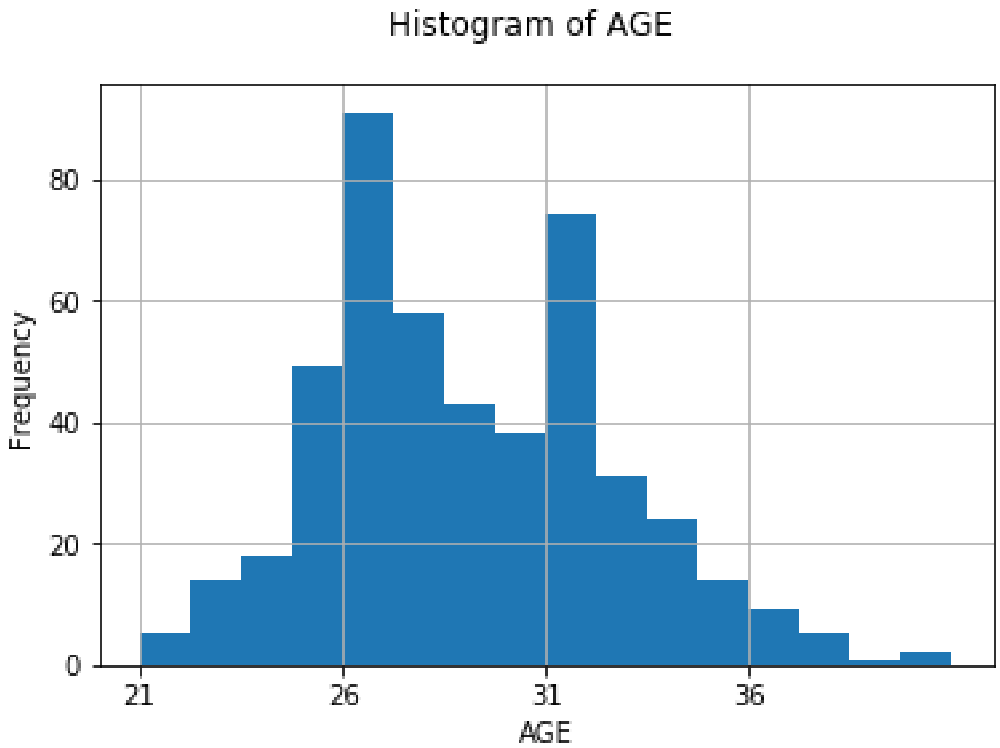
```
In [17]: data['nom_Sal'].value_counts()
```

```
Out[17]: 6-      183
         6+      142
         40+     101
         100+     50
         Name: nom_Sal, dtype: int64
```

```
In [18]: hist_age = np.histogram(data['AGE'], bins =16)
         print(hist_age)
         axarr_age = data['AGE'].hist(bins = 16)
         pl.suptitle("Histogram of AGE")
         axarr_age.set_xlabel("AGE")
         axarr_age.set_ylabel("Frequency")
         pl.xticks(np.arange(21, 41, 5))
```

```
(array([ 5, 14, 18, 49, 91, 58, 43, 38, 74, 31, 24, 14,  9,  5,  1,  2],
      dtype=int64), array([21.  , 22.25, 23.5 , 24.75, 26.  , 27.25, 28.5 , 29.75, 31.  ,
       32.25, 33.5 , 34.75, 36.  , 37.25, 38.5 , 39.75, 41.  ]))
```

```
Out[18]: ([<matplotlib.axis.XTick at 0x983ee80>,
           <matplotlib.axis.XTick at 0x983e7b8>,
           <matplotlib.axis.XTick at 0x983e518>,
           <matplotlib.axis.XTick at 0x96629e8>],
          <a list of 4 Text xticklabel objects>)
```




```
In [19]: def categ_age(x):
             if x < 26:
                 return '26-'
             if 26 <= x < 31:
                 return '26+'
             else:
                 return '31+'
         data['nom_AGE'] = data['AGE'].apply(categ_age)
         data['nom_AGE'].value_counts()

Out[19]: 26+    230
         31+    160
         26-     86
         Name: nom_AGE, dtype: int64
```

#
Salary - Position

```
In [21]: #Contingency (Co-Occurrence) table for Salary and Position
         crtable_sal_pos = pd.crosstab(index=data['nom_Sal'], columns=data['Pos'], margins=Tru
         crtable_sal_pos

Out[21]: Pos      RP   SP  All
         nom_Sal
```

```
100+       13   37   50
40+        56   45  101
6+        113   29  142
6-        124   59  183
All       306  170  476
```

In [22]:
```
#Conditional Probability(Pos|Salary)
crtable_sal_pos.div(crtable_sal_pos["All"], axis = 0)
```

Out[22]:
```
Pos            RP        SP  All
nom_Sal
100+     0.260000  0.740000  1.0
40+      0.554455  0.445545  1.0
6+       0.795775  0.204225  1.0
6-       0.677596  0.322404  1.0
All      0.642857  0.357143  1.0
```

In [24]:
```
#Conditional Probability(Salary|Pos)
Cond_sal_pos1 = crtable_sal_pos/crtable_sal_pos.loc["All"]
Cond_sal_pos1.drop(index = 'All', columns = "All")
```

Out[24]:
```
Pos            RP        SP
nom_Sal
100+     0.042484  0.217647
40+      0.183007  0.264706
6+       0.369281  0.170588
6-       0.405229  0.347059
```

In [26]:
```
#Probability of Salary
crtable_sal = pd.crosstab(index=data['nom_Sal'], columns='Percent', margins = True)
crtable_sal1 = crtable_sal/crtable_sal.values[4,0]
crtable_sal1
```

Out[26]:
```
col_0     Percent       All
nom_Sal
100+     0.105042  0.105042
40+      0.212185  0.212185
6+       0.298319  0.298319
6-       0.384454  0.384454
All      1.000000  1.000000
```

In [27]:
```
Cond_sal_pos1.values[1,0]/crtable_sal1.values[1,0]
```

Out[27]:
```
0.8624862486248625
```

In [29]:
```
#Probability of Position
crtable_pos = pd.crosstab(index=data['Pos'], columns='Percent', margins = True)
crtable_pos1 = crtable_pos/crtable_pos.values[2,0]
crtable_pos1
```

```
Out[29]: col_0    Percent       All
         Pos
         RP      0.642857  0.642857
         SP      0.357143  0.357143
         All     1.000000  1.000000
```

```
In [31]: #P(nom_Sal)*P(Pos)
         newdf = pd.DataFrame(index = crtable_sal1.index, columns = crtable_pos1.index)
         for i in range(5):
             for k in range(3):
                 newdf.iloc[i,k] = crtable_sal1.iloc[i,0]*crtable_pos1.iloc[k,0]
         newdf
```

```
Out[31]: Pos            RP         SP       All
         nom_Sal
         100+     0.067527   0.037515  0.105042
         40+      0.136405  0.0757803  0.212185
         6+       0.191777   0.106543  0.298319
         6-       0.247149   0.137305  0.384454
         All      0.642857   0.357143         1
```

```
In [32]: #Quetelet relative index table Salary to the Position)
         Quetlet_table_sal_pos = (Cond_sal_pos1.div(crtable_sal1.iloc[:,0], axis='index')-1)
         Quetlet_table_sal_pos = Quetlet_table_sal_pos.drop(index = 'All', columns ='All')
         Quetlet_table_sal_pos['P(Salary)'] = crtable_sal1['All']
         Quetlet_table_sal_pos
```

```
Out[32]: Pos            RP        SP  P(Salary)
         nom_Sal
         100+    -0.595556  1.072000   0.105042
         40+     -0.137514  0.247525   0.212185
         6+       0.237872 -0.428169   0.298319
         6-       0.054038 -0.097268   0.384454
```

According to the "Quetelet relative index table" it seems that Salary (100+), given a position Starting Pitcher, is 107% more frequent than average. This event also may be seen in the articles, where the highest-paid players are at a position 'Starting Pitcher'. [https://www.businessinsider.com/chart-mlbs-highest-paid-positions-2014-7]

```
In [33]: #Contingency (Co-Occurrence) table for Salary and Position in percetage
         crtable_sal_pos_perc = pd.crosstab(index=data['nom_Sal'], columns=data['Pos'],margins
         crtable_sal_pos_perc
```

```
Out[33]: Pos            RP        SP       All
         nom_Sal
         100+     0.027311  0.077731  0.105042
         40+      0.117647  0.094538  0.212185
         6+       0.237395  0.060924  0.298319
         6-       0.260504  0.123950  0.384454
         All      0.642857  0.357143  1.000000
```

In [34]:
```
writer = pd.ExcelWriter('salary - position.xlsx', engine='xlsxwriter')
crtable_sal_pos.to_excel(writer, sheet_name='Co-Occurrence')
crtable_sal_pos_perc.to_excel(writer, sheet_name='Co-Occurrence_perc')
Cond_sal_pos1.to_excel(writer, sheet_name='P(Salary|Pos)')
Quetlet_table_sal_pos.to_excel(writer, sheet_name='Quetlet')
newdf.to_excel(writer, sheet_name='P(nom_Sal)P(Pos)')
writer.save()
```

The chi-square-summary_Quetelet_index

In [35]:
```
Q_ind = 0
for i in range(4):
    for k in range(2):
        Q_ind+=crtable_sal_pos_perc.iloc[i,k]*Quetlet_table_sal_pos.iloc[i,k]
Q_ind
```

Out[35]:
```
0.10668926352525418
```

Age - Position

In [44]:
```
#Contingency (Co-Occurrence) table for Age and Salary
crtable_age_pos = pd.crosstab(index=data['nom_AGE'], columns=data['Pos'], margins=Tru
crtable_age_pos
```

Out[44]:
```
Pos       RP   SP  All
nom_AGE
26+      152   78  230
26-       43   43   86
31+      111   49  160
All      306  170  476
```

In [45]:
```
#Conditional Probability(Age|Position)
Cond_age_pos1 = crtable_age_pos/crtable_age_pos.loc["All"]
Cond_age_pos1.drop(index = 'All', columns = "All")
```

Out[45]:
```
Pos            RP        SP
nom_AGE
26+      0.496732  0.458824
26-      0.140523  0.252941
31+      0.362745  0.288235
```

In [47]:
```
#Probability of Position
crtable_age = pd.crosstab(index=data['nom_AGE'], columns='Percent', margins = True)
crtable_age1 = crtable_age/crtable_age.values[3,0]
crtable_age1
```

Out[47]:
```
col_0     Percent       All
nom_AGE
26+      0.483193  0.483193
26-      0.180672  0.180672
31+      0.336134  0.336134
All      1.000000  1.000000
```

In [53]:
```
#Quetelet relative index table Salary to the Age
Quetlet_table_age_pos = (Cond_age_pos1.div(crtable_age1.iloc[:,0], axis='index')-1)
Quetlet_table_age_pos = Quetlet_table_age_pos.drop(index = 'All', columns ='All')
Quetlet_table_age_pos['P(Age)'] = crtable_age1['All']
Quetlet_table_age_pos
```

Out[53]:
```
Pos            RP        SP    P(Age)
nom_AGE
26+      0.028019 -0.050435  0.483193
26-     -0.222222  0.400000  0.180672
31+      0.079167 -0.142500  0.336134
```

In [54]:
```
#Contingency (Co-Occurrence) table for Age and Position in percetage
crtable_age_pos_perc = pd.crosstab(index=data['nom_AGE'], columns=data['Pos'],margins
crtable_age_pos_perc
```

Out[54]:
```
Pos            RP        SP       All
nom_AGE
26+      0.319328  0.163866  0.483193
26-      0.090336  0.090336  0.180672
31+      0.233193  0.102941  0.336134
All      0.642857  0.357143  1.000000
```

In [55]:
```
#P(nom_AGE)*P(Pos)
newdf_age = pd.DataFrame(index = crtable_age1.index, columns = crtable_pos1.index)
for i in range(4):
    for k in range(3):
        newdf_age.iloc[i,k] = crtable_age1.iloc[i,0]*crtable_pos1.iloc[k,0]
newdf_age
```

Out[55]:
```
Pos            RP         SP       All
nom_AGE
26+      0.310624   0.172569  0.483193
26-      0.116146  0.0645258  0.180672
31+      0.216086   0.120048  0.336134
All      0.642857   0.357143         1
```

In [56]:
```
writer2 = pd.ExcelWriter('age - position.xlsx', engine='xlsxwriter')
crtable_age_pos.to_excel(writer2, sheet_name='Co-Occurrence')
crtable_age_pos_perc.to_excel(writer2, sheet_name='Co-Occurrence_perc')
Cond_age_pos1.to_excel(writer2, sheet_name='P(Salary|Pos)')
Quetlet_table_age_pos.to_excel(writer2, sheet_name='Quetlet')
newdf_age.to_excel(writer2, sheet_name='P(nom_AGE)P(Pos)')
writer2.save()
```

The chi-square-summary_Quetelet_index

In [57]:
```
Q_ind_age = 0
for i in range(3):
```

```
        for k in range(2):
            Q_ind_age+=crtable_age_pos_perc.iloc[i,k]*Quetlet_table_age_pos.iloc[i,k]
    Q_ind_age
```

Out[57]: 0.020534597897129823

In [58]:
```
Q_ind_age*len(data)
```

Out[58]: 9.774468599033796

homework 5

December 2, 2018

#
Homework 5: Regression

```
In [70]: import warnings
         warnings.filterwarnings('ignore')
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         from scipy import stats
         data = pd.DataFrame.from_csv('baseball_sal.csv')
         data.head()
```

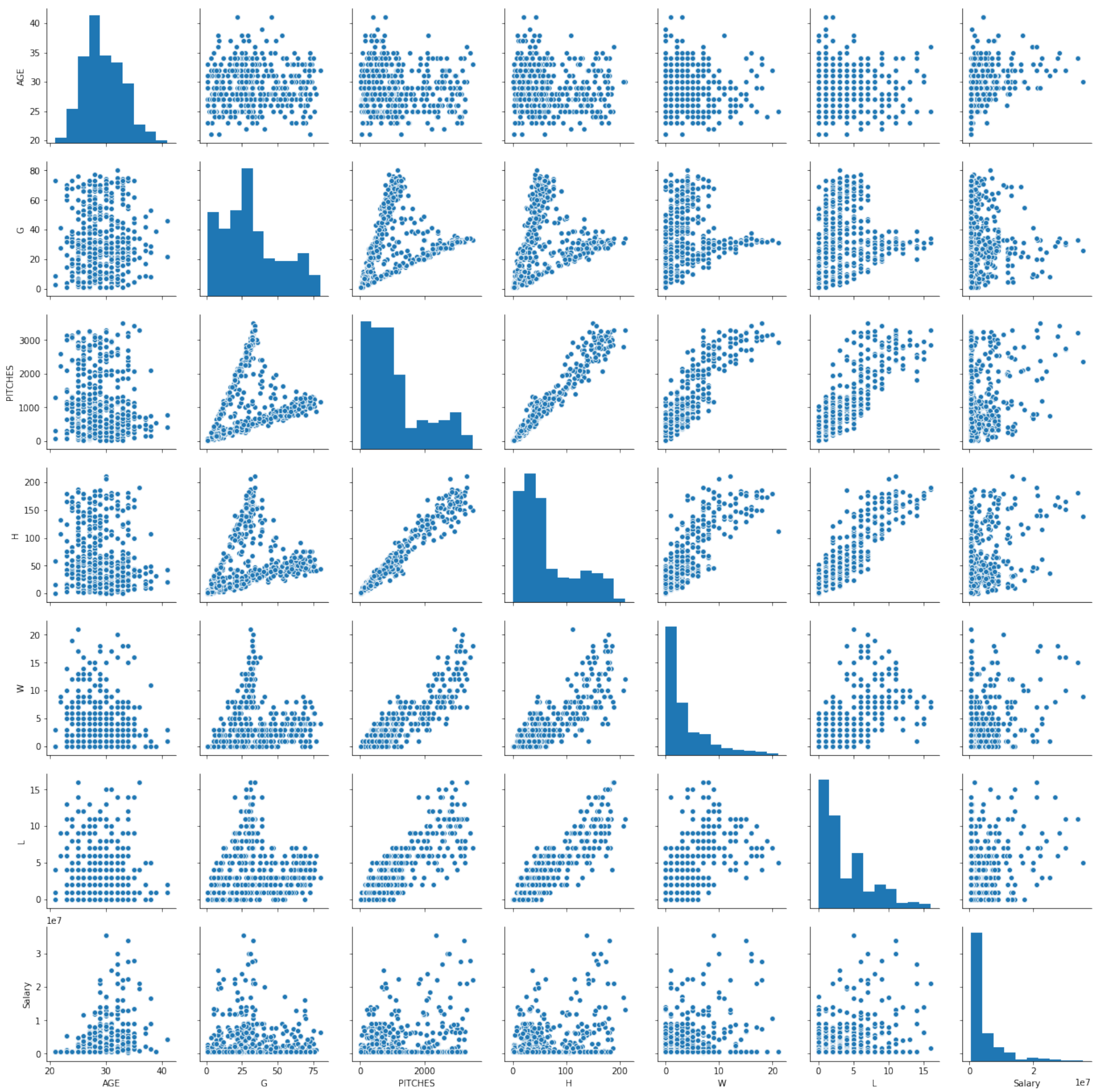
```
Out[70]:                 NAME TEAM  LG  AGE   G  PITCHES   AB   R  ER    H   1B  2B  3B  \
         0      Max Scherzer  WAS  NL   33  33     3503  797  66  62  150   86  36   5
         1  Justin Verlander  HOU  AL   35  34     3427  781  63  60  156   92  31   5
         2        Aaron Nola  PHI  NL   25  33     3221  756  57  56  149   99  31   2
         3       Gerrit Cole  HOU  AL   27  32     3257  723  68  64  143   86  36   2
         4     Corey Kluber  CLE  AL   32  33     3181  801  75  69  179  118  32   4

            HR   W  L  PVORP      Salary Pos
         0  23  18  7   74.1  22142857.0  SP
         1  28  16  9   70.9  28000000.0  SP
         2  17  17  6   63.9    573000.0  SP
         3  19  15  5   61.5   6750000.0  SP
         4  25  20  7   58.9  10700000.0  SP
```

```
In [71]: data.columns
```

```
Out[71]: Index(['NAME', 'TEAM', 'LG', 'AGE', 'G', 'PITCHES', 'AB', 'R', 'ER', 'H', '1B',
               '2B', '3B', 'HR', 'W', 'L', 'PVORP', 'Salary', 'Pos'],
              dtype='object')
```

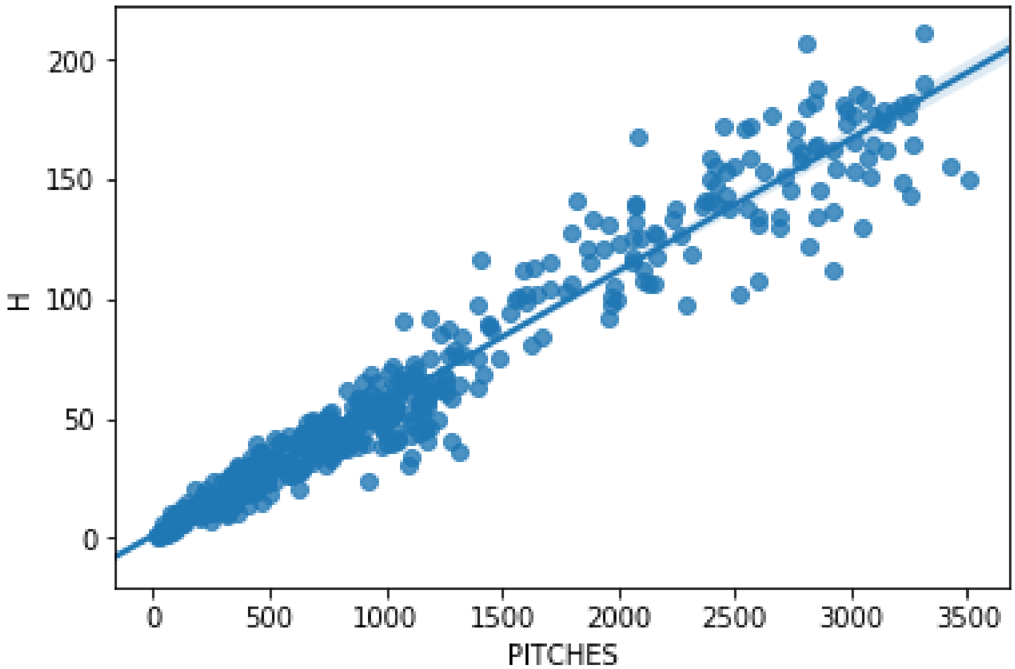
```
In [72]: # Seaborn visualization library
         import seaborn as sns
         # Create the default pairplot
         sns.pairplot(data.loc[:,['AGE', 'G', 'PITCHES', 'H', 'W', 'L', 'Salary']])
```

```
Out[72]: <seaborn.axisgrid.PairGrid at 0x31ecc518>
```

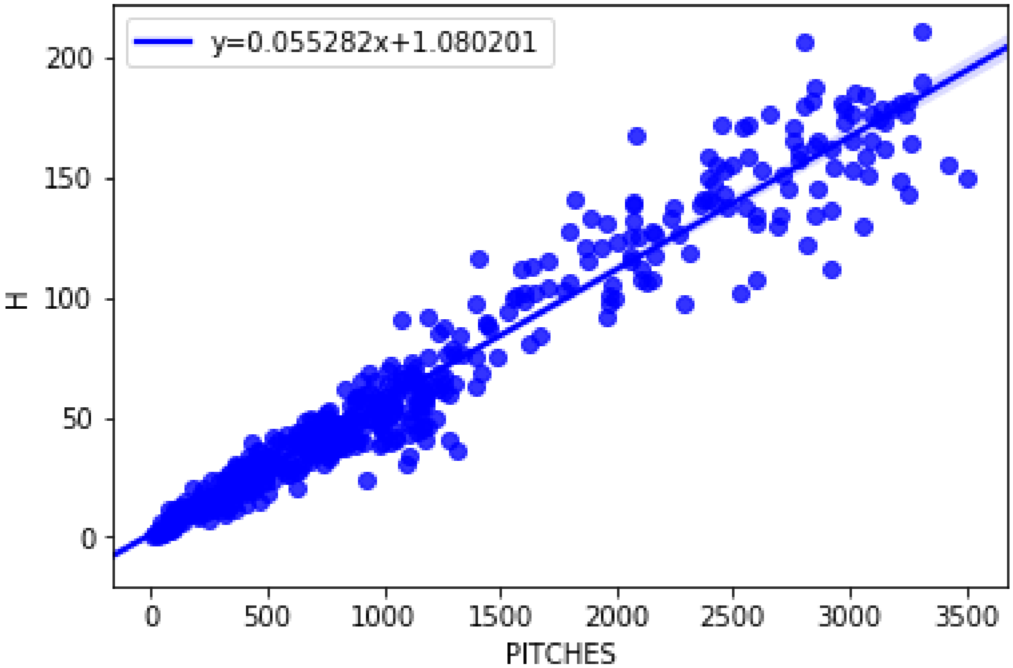



```
In [73]: sns.regplot( y = 'H', x = 'PITCHES', data = data)
```

```
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x331f9da0>
```




```
In [74]: # get coeffs of linear fit
         slope, intercept, r_value, p_value, std_err = stats.linregress(data['PITCHES'],data['I

         # use line_kws to set line label for legend
         ax = sns.regplot(x="PITCHES", y="H", data=data, color='b',
          line_kws={'label':"y={0:f}x+{1:f}".format(slope,intercept)})

         # plot legend
         ax.legend()

         plt.show()
         print("y=",slope,"*x+",intercept)
```




```
y= 0.05528177495482146 *x+ 1.0802010289457087
```

```
In [75]: print("r-squared:", r_value**2)
         print("corr:", r_value)
```

```
r-squared: 0.9438735514385752
corr: 0.9715315493789047
```

```
In [76]: y=0
         def predict(x):
             y = slope*x+intercept
             return y
         predict(data.loc[1,'PITCHES'])
```

```
Out[76]: 190.53084379911883
```

```
In [77]: prediction_dataframe = pd.DataFrame()
         n = (1,10,100)
         for i in n:
             prediction_dataframe.loc[i,'PITCHES'] = data.loc[i,'PITCHES']
             prediction_dataframe.loc[i,'OBS_H'] = data.loc[i,'H']
             prediction_dataframe.loc[i,'PRED_H'] = predict(data.loc[i,'PITCHES'])
             prediction_dataframe.loc[i,'Error,%'] = (predict(data.loc[i,'PITCHES']) - data.loc
         prediction_dataframe
```

```
Out[77]:       PITCHES  OBS_H      PRED_H    Error,%
         1      3427.0  156.0  190.530844  22.135156
         10     3221.0  181.0  179.142798  -1.026078
         100     903.0   57.0   50.999644 -10.526941
```

```
In [78]: prediction_dataframe['Error,%'].abs().sum()/len(n)
```

```
Out[78]: 11.229391775227418
```

```
In [79]: len(data)
```

```
Out[79]: 476
```

```
In [80]: error_mod = 0
         for i in range(len(data)):
             error_mod = abs(data.loc[i,'H']-predict(data.loc[i,'PITCHES']))/abs(data.loc[i,'H
         error_mod*100/len(data)
```

```
Out[80]: 0.009513901967643849
```

```
In [ ]:
```