Introduction:

Try to see if plate discipline in baseball affected other stats including regular ones like BB%, K%, AVG, OBP, SLG, OPS, ISO, wOBA, wRC+, etc

Also see if plate discipline in baseball affected other stats like WAR per game to see how plate discipline affect winning

Methods:

Data was from FanGraphs

Use SQL Server and Python(Spyder)

Use linear regression

Plate Discipline defined as (ZSwing%-OSwing%)/Swing%

Find out the correlation coefficient, linear regression line, ratio and heritability, all of them were performed with pairs bootstrap as well, in order to analyze probabilistically and get the confidence interval.

In [1]:
```
import pandas as pd
```

In [2]:
```
import pyodbc
```

In [3]:
```
#connect with sql server, which already contains two tables from FanGra
phs
#1.https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat&lg=all&qual
=1000&type=8&season=2019&month=0&season1=2010&ind=0&team=0&rost=0&age=0
&filter=&players=0&startdate=&enddate=
#2.https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat&lg=all&qual
=1000&type=5&season=2019&month=0&season1=2010&ind=0&team=0&rost=0&age=0
&filter=&players=0&startdate=2010-01-01&enddate=2019-12-31
```

```
sql_conn = pyodbc.connect('''DRIVER={ODBC Driver 13 for SQL Server};
                             SERVER=ALLENHO\MSSQLSERVER002;
                             DATABASE=Plate discipline and winning corre
lation;
                             Trusted_Connection=yes''') # Make scatter p
lot
_ = plt.plot(df_new['plated'], df_new['per_war'], marker='.',
             linestyle='none', color='blue', alpha=0.5)

# Label axes and make legend
_ = plt.xlabel('plate discipline')
_ = plt.ylabel('per game war')
```

In [4]:
```
#sql query to grab data, including players from 2010-2019 with over 100
0 PA.
#Define plate discipline as [Z-Swing%]-[O-Swing%])/[Swing%]
#Also grab data from that corresponding player's BB%, K%, AVG, OBP, IS
O, wOBA, wRC+, WAR/PA for that period
query = '''
SELECT p.name, ([Z-Swing%]-[O-Swing%])/[Swing%] as plated, [BB%], [K%],
 AVG, OBP, ISO, wOBA, [wRC+], (d.WAR/d.PA) as per_war
FROM ['plate discipline 2010-2019 1000$'] p
JOIN ['dashboard stats 2010-2019 1000P$'] d
on p.name = d.name
order by WAR desc;
'''
```

In [5]:
```
#convert the data into dataframe
df = pd.read_sql(query, sql_conn)
```

In [6]:
```
#convert columns' type into string
df.columns = df.columns.astype(str)
```

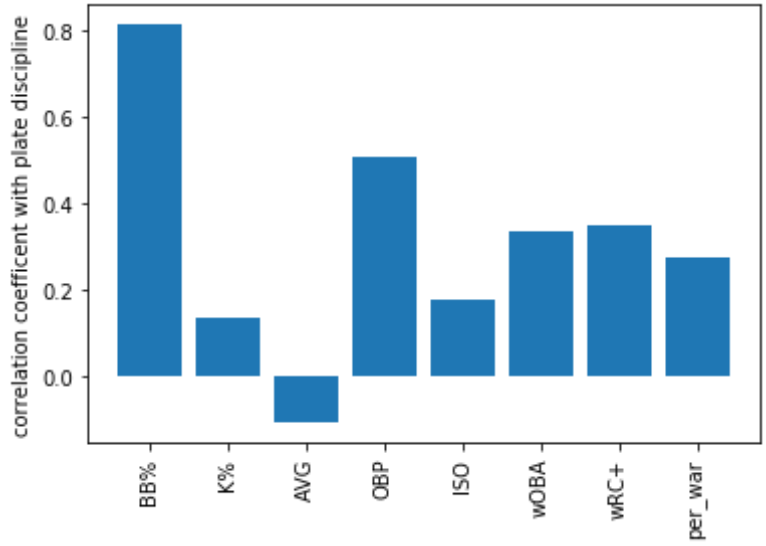
In [7]:
```
#slice the data into only columns from plated(respresent plate discipli
ne) to per_war(represent per game war)
df_new = df.loc[:, 'plated':'per_war']
```

In [8]:
```
#get the correlation table from df_new
df_corr = df_new.corr()
```

In [9]:
```
#slice the correlation table(df_corr) row from BB% to per game war
df_corr_new = df_corr.loc['BB%':'per_war',:]
```

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
```

In [11]:
```
fig, ax = plt.subplots()

#-----plot the correlation table as bar plot----------------------
ax.bar(df_corr_new.index, df_corr_new['plated'])

ax.set_xticklabels(df_corr_new.index, rotation=90)

ax.set_ylabel('correlation coefficent with plate discipline')

plt.show()
```



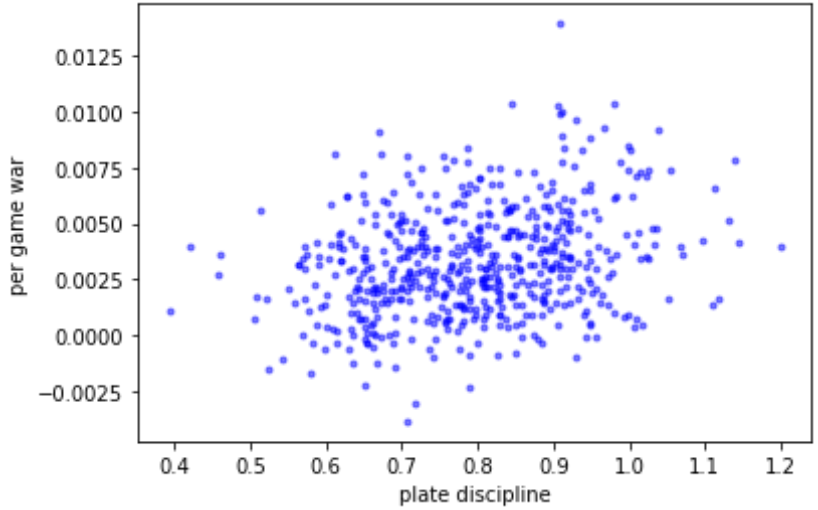
In [12]:
```
#-----plot linear regression line and pairs bootstraps and compute rati
o-------------------------------------
def draw_bs_pairs_linreg(x, y, size=1):
    """Perform pairs bootstrap for linear regression."""
    # Set up array of indices to sample from: inds
    inds = np.arange(len(x))
    # Initialize replicates: bs_slope_reps, bs_intercept_reps
    bs_slope_reps = np.empty(size)
    bs_intercept_reps = np.empty(size)
    # Generate replicates
    for i in range(size):
        bs_inds = np.random.choice(inds, size=len(inds))
        bs_x, bs_y = x[bs_inds], y[bs_inds]
        bs_slope_reps[i], bs_intercept_reps[i] = np.polyfit(bs_x, bs_y,
 1)
    return bs_slope_reps, bs_intercept_reps
```

In [13]:
```
# Compute the linear regressions
slope_perwar, intercept_perwar = np.polyfit(df_new['plated'], df_new['p
er_war'], 1)

# Perform pairs bootstrap for the linear regressions
bs_slope_reps_perwar, bs_intercept_reps_perwar = draw_bs_pairs_linreg(
                    df_new['plated'], df_new['per_war'], size=1000)

# Compute confidence intervals of slopes
slope_conf_int_perwar = np.percentile(bs_slope_reps_perwar, [2.5, 97.5
])

intercept_conf_int_perwar = np.percentile(
                            bs_intercept_reps_perwar, [2.5, 97.5])

# Print the results
print('perwar: slope =', slope_perwar,
      'conf int =', slope_conf_int_perwar)
print('perwar: intercept =', intercept_perwar,
      'conf int =', intercept_conf_int_perwar)
```

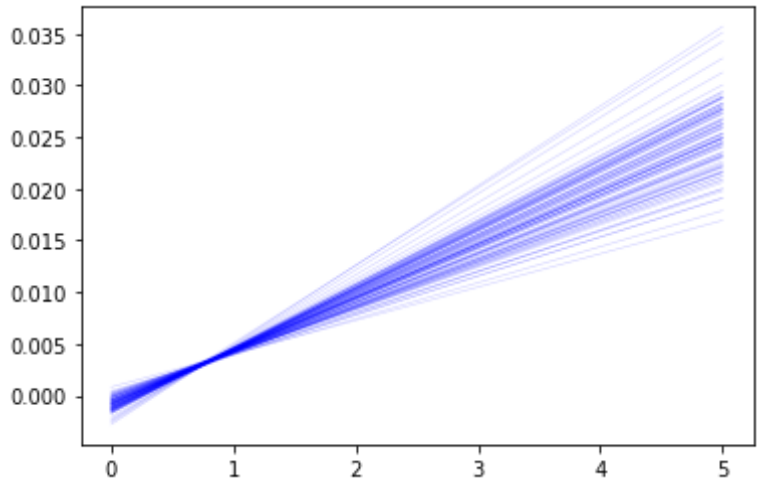
```
perwar: slope = 0.005104575299609698 conf int = [0.0036584  0.00656571]
perwar: intercept = -0.0007680167310351522 conf int = [-0.00193384  0.0
```

0035904]

In [14]:
```
# Make scatter plot
_ = plt.plot(df_new['plated'], df_new['per_war'], marker='.',
             linestyle='none', color='blue', alpha=0.5)

# Label axes and make legend
_ = plt.xlabel('plate discipline')
_ = plt.ylabel('per game war')
```



In [15]:
```
# Generate x-values for bootstrap lines: x
x = np.array([0, 5])
# Plot the bootstrap lines
for i in range(100):
    plt.plot(x, bs_slope_reps_perwar[i] * x + bs_intercept_reps_perwar[
i],
             linewidth=0.5, alpha=0.2, color='blue')

# Draw the plot again
plt.show()
```



In [16]:
```
# Compute ratios
ratio_perwar = df_new['per_war'] / df_new['plated']

# Compute means
mean_ratio_perwar = np.mean(ratio_perwar)
```

In [17]:
```
def bootstrap_replicate_1d(data, func):
    """Generate bootstrap replicate of 1D data."""
    bs_sample = np.random.choice(data, len(data))
    return func(bs_sample)

def draw_bs_reps(data, func, size=1):
    """Draw bootstrap replicates."""

    # Initialize array of replicates: bs_replicates
    bs_replicates = np.empty(size)

    # Generate replicates
    for i in range(size):
        bs_replicates[i] = bootstrap_replicate_1d(data, func)

    return bs_replicates
```

```
# Generate bootstrap replicates of the means
bs_replicates_perwar = draw_bs_reps(ratio_perwar, np.mean, size=10000)

# Compute the 99% confidence intervals
conf_int_perwar = np.percentile(bs_replicates_perwar, [0.5, 99.5])

# Print the results
print('war per game: mean ratio =', mean_ratio_perwar,
      'conf int =', conf_int_perwar)
```

```
war per game: mean ratio = 0.004112620026148256 conf int = [0.0037952
0.00444187]
```

In [18]:

```
#-----check correlation and heritability-----------------------------
----------------------------------------

def draw_bs_pairs(x, y, func, size=1):
    """Perform pairs bootstrap for a single statistic."""
    # Set up array of indices to sample from: inds
    inds = np.arange(len(x))
    # Initialize replicates: bs_replicates
    bs_replicates = np.empty(size)
    # Generate replicates
    for i in range(size):
        bs_inds = np.random.choice(inds, len(inds))
        bs_x, bs_y = x[bs_inds], y[bs_inds]
        bs_replicates[i] = func(bs_x, bs_y)
    return bs_replicates

def pearson_r(x, y):
    """Compute Pearson correlation coefficient between two arrays."""
    # Compute correlation matrix: corr_mat
    corr_mat = np.corrcoef(x, y)

    # Return entry [0,1]
    return corr_mat[0,1]

# Compute the Pearson correlation coefficients
```

```
r_perwar = pearson_r(df_new['plated'], df_new['per_war'])

# Acquire 1000 bootstrap replicates of Pearson r
bs_replicates_perwar = draw_bs_pairs(
        df_new['plated'], df_new['per_war'], pearson_r, size=1000)

# Compute 95% confidence intervals
conf_int_perwar = np.percentile(bs_replicates_perwar, [2.5, 97.5])

# Print results
print('per_war_corr_coef', r_perwar, conf_int_perwar)
```

```
per_war_corr_coef 0.274454416886356 [0.20076916 0.34203457]
```

In [19]:

```
def heritability(parents, offspring):
    """Compute the heritability from parent and offspring samples."""
    covariance_matrix = np.cov(parents, offspring)
    return covariance_matrix[0,1] / covariance_matrix[0,0]
# Compute the heritability
heritability_perwar = heritability(df_new['plated'],
                                   df_new['per_war'])

# Acquire 1000 bootstrap replicates of heritability
replicates_perwar = draw_bs_pairs(
        df_new['plated'], df_new['per_war'], heritability, size=1000)

# Compute 95% confidence intervals
conf_int_perwar = np.percentile(replicates_perwar, [2.5, 97.5])

# Print results
print('per_war_heritability', heritability_perwar, conf_int_perwar)

# Initialize array of replicates: perm_replicates
perm_replicates = np.empty(10000)

# Draw replicates
for i in range(10000):
    # Permute parent beak depths
    bd_parent_permuted = np.random.permutation(df_new['plated'])
```

```
    perm_replicates[i] = heritability(df_new['plated'],
                                      df_new['per_war'])

# Compute p-value: p
p = np.sum(perm_replicates >= heritability_perwar) / len(perm_replicate
s)

# Print the p-value
print('p-val =', p)

#-----above are all examples of relation between plate discipline and w
ar per game.
#-----we can also generate the same calculations and plot with BB%, K%,
 AVG, OBP, ISO, wOBA, wRC+
```

```
per_war_heritability 0.005104575299609694 [0.00369509 0.00662733]
p-val = 1.0
```

Results:

For results of relationship between plate discipline(plated) and per game war(pgw):

correlation coefficient = 0.274454416886356 [0.19710456 0.3450419]

linear regression slope = 0.005104575299609698 conf int = [0.00357109 0.00659355]

linear regression intercept = -0.0007680167310351522 conf int = [-0.00200369 0.00046052]

pgw/plated mean = 0.004112620026148256 conf int = [0.00379471 0.00442882]

heritability = 0.005104575299609694 [0.00357444 0.00679131]