# Linear Regression - Association of Tennis Professionals data

May 10, 2024

[8]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# load and investigate the data here:
df = pd.read_csv('./tennis_stats.csv')

# perform exploratory analysis here:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1721 entries, 0 to 1720
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Player                      1721 non-null   object
 1   Year                        1721 non-null   int64
 2   FirstServe                  1721 non-null   float64
 3   FirstServePointsWon         1721 non-null   float64
 4   FirstServeReturnPointsWon   1721 non-null   float64
 5   SecondServePointsWon        1721 non-null   float64
 6   SecondServeReturnPointsWon  1721 non-null   float64
 7   Aces                        1721 non-null   int64
 8   BreakPointsConverted        1721 non-null   float64
 9   BreakPointsFaced            1721 non-null   int64
 10  BreakPointsOpportunities    1721 non-null   int64
 11  BreakPointsSaved            1721 non-null   float64
 12  DoubleFaults                1721 non-null   int64
 13  ReturnGamesPlayed           1721 non-null   int64
 14  ReturnGamesWon              1721 non-null   float64
 15  ReturnPointsWon             1721 non-null   float64
 16  ServiceGamesPlayed          1721 non-null   int64
 17  ServiceGamesWon             1721 non-null   float64
 18  TotalPointsWon              1721 non-null   float64
 19  TotalServicePointsWon       1721 non-null   float64
 20  Wins                        1721 non-null   int64
 21  Losses                      1721 non-null   int64
 22  Winnings                    1721 non-null   int64
 23  Ranking                     1721 non-null   int64
dtypes: float64(12), int64(11), object(1)
memory usage: 322.8+ KB
```

[11]:

```
df.describe()
```

[11]:

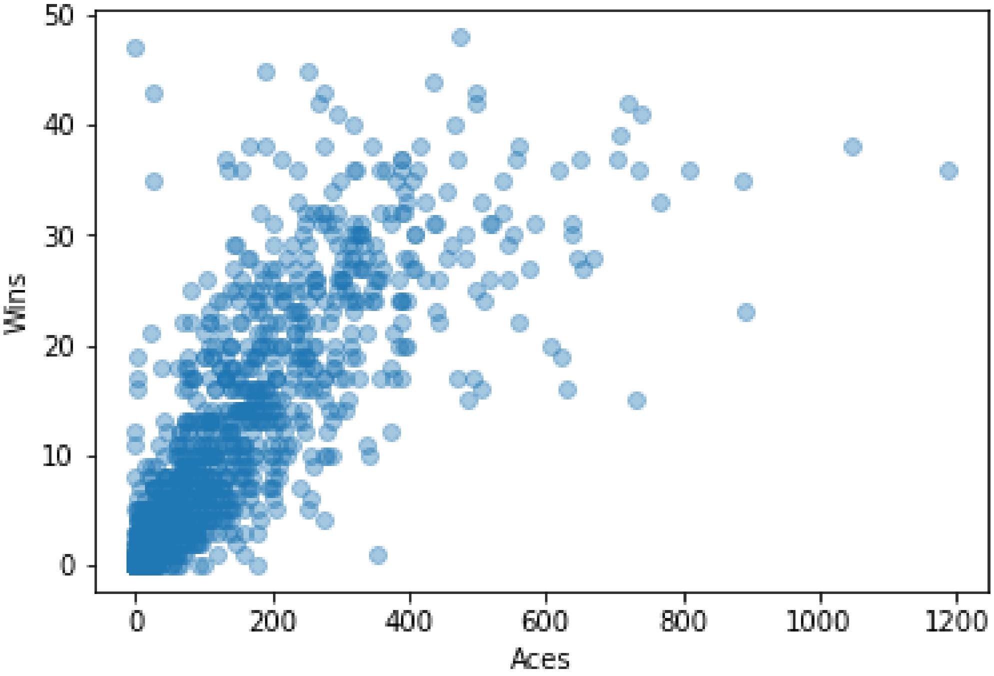
```
              Year   FirstServe  FirstServePointsWon  \
count  1721.000000  1721.000000          1721.000000
mean   2013.646717     0.598053             0.680738
std       2.488018     0.054533             0.070422
min    2009.000000     0.360000             0.270000
25%    2012.000000     0.570000             0.650000
50%    2014.000000     0.600000             0.690000
75%    2016.000000     0.630000             0.720000
max    2017.000000     0.880000             0.890000

       FirstServeReturnPointsWon  SecondServePointsWon  \
count                1721.000000           1721.000000
mean                    0.261673              0.479733
std                     0.056639              0.066902
min                     0.000000              0.060000
25%                     0.240000              0.460000
50%                     0.270000              0.490000
75%                     0.290000              0.520000
max                     0.480000              0.920000

       SecondServeReturnPointsWon         Aces  BreakPointsConverted  \
count                 1721.000000  1721.000000           1721.000000
mean                     0.466432    97.105171              0.369407
std                      0.068447   137.966077              0.162987
min                      0.000000     0.000000              0.000000
25%                      0.440000     7.000000              0.320000
50%                      0.480000    34.000000              0.380000
75%                      0.500000   140.000000              0.430000
max                      0.750000  1185.000000              1.000000

       BreakPointsFaced  BreakPointsOpportunities  ...  ReturnGamesWon  \
count       1721.000000               1721.000000  ...     1721.000000
mean         112.003486                102.918071  ...        0.173823
std          119.247651                122.761670  ...        0.080880
min            1.000000                  0.000000  ...        0.000000
25%           15.000000                  9.000000  ...        0.130000
50%           55.000000                 41.000000  ...        0.180000
75%          201.000000                172.000000  ...        0.220000
max          507.000000                573.000000  ...        0.560000
```

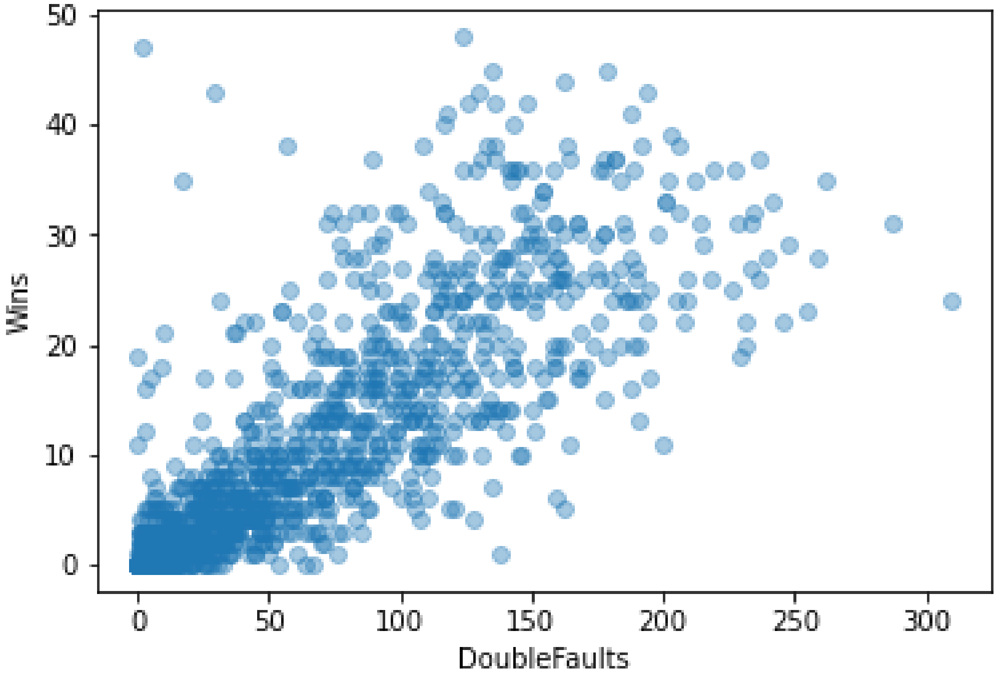
```
       ReturnPointsWon  ServiceGamesPlayed  ServiceGamesWon  TotalPointsWon  \
count      1721.000000         1721.000000      1721.000000     1721.000000
mean          0.342208          197.650203         0.715590        0.473155
std           0.049369          221.208703         0.123287        0.037139
min           0.000000            0.000000         0.000000        0.220000
25%           0.320000           22.000000         0.670000        0.460000
50%           0.350000           86.000000         0.750000        0.480000
75%           0.370000          348.000000         0.790000        0.500000
max           0.510000          916.000000         1.000000        0.820000

       TotalServicePointsWon         Wins       Losses      Winnings  \
count            1721.000000  1721.000000  1721.000000  1.721000e+03
mean                0.599245     7.876816     9.278908  2.344928e+05
std                 0.057718    10.183716     8.996450  2.530537e+05
min                 0.250000     0.000000     0.000000  1.080000e+02
25%                 0.570000     0.000000     2.000000  4.931100e+04
50%                 0.610000     3.000000     5.000000  1.252120e+05
75%                 0.630000    13.000000    17.000000  3.500750e+05
max                 0.820000    48.000000    36.000000  1.074562e+06

           Ranking
count  1721.000000
mean    269.610691
std     277.341947
min       3.000000
25%      83.000000
50%     166.000000
75%     333.000000
max    1443.000000

[8 rows x 23 columns]
```

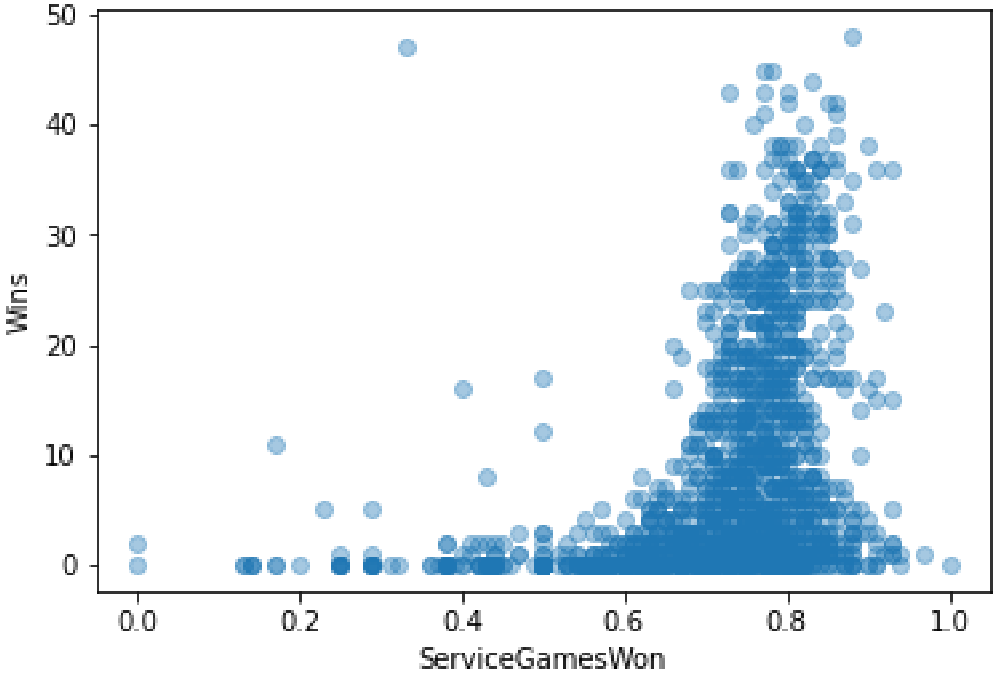
[34]:

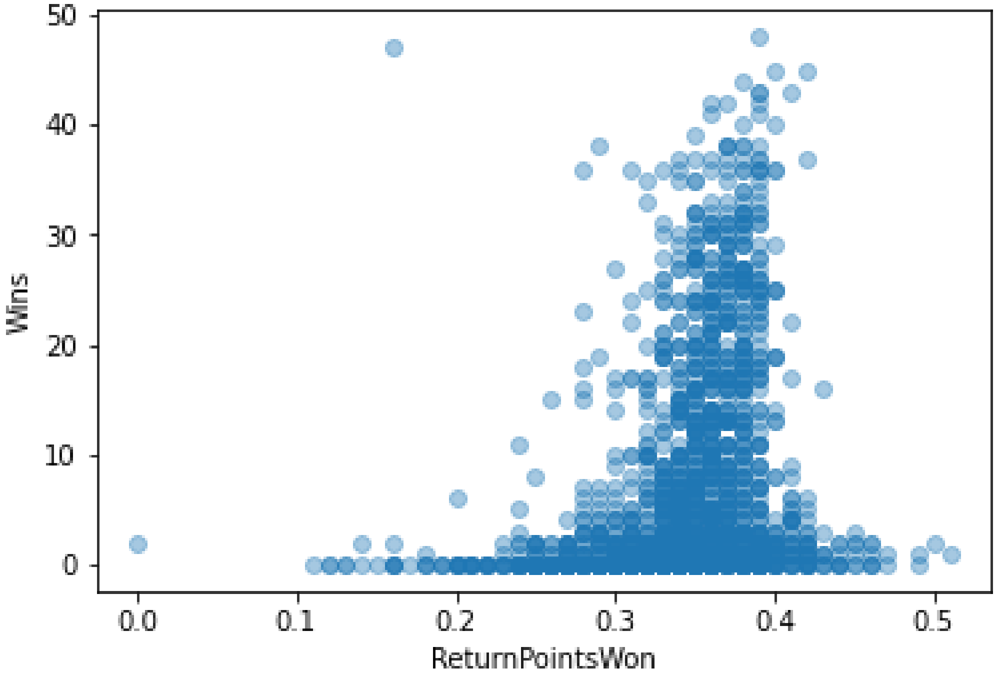
```
# Explore features visually
plt.plot(df['Aces'], df['Wins'], 'o', alpha=0.4) # good feature
plt.xlabel("Aces")
plt.ylabel("Wins")
plt.show()
plt.plot(df['DoubleFaults'], df['Wins'], 'o', alpha=0.4) # good feature
plt.xlabel("DoubleFaults")
plt.ylabel("Wins")
plt.show()
plt.plot(df['ServiceGamesWon'], df['Wins'], 'o', alpha=0.4)
plt.xlabel("ServiceGamesWon")
plt.ylabel("Wins")
plt.show()
plt.plot(df['ReturnPointsWon'], df['Wins'], 'o', alpha=0.4)
plt.xlabel("ReturnPointsWon")
```

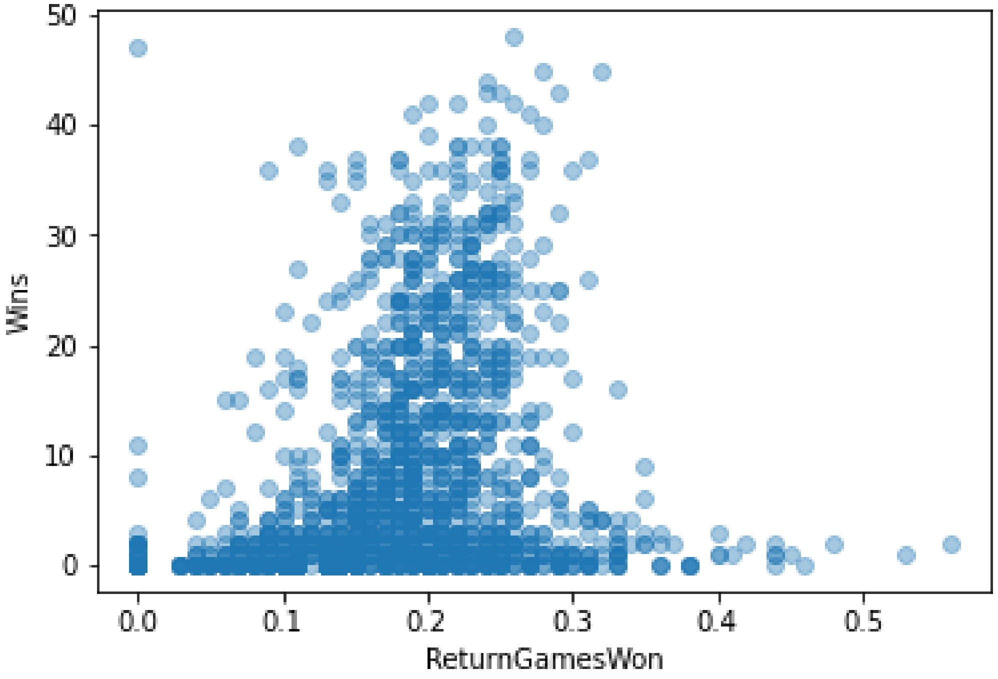
```python
plt.ylabel("Wins")
plt.show()
plt.plot(df['ReturnGamesWon'], df['Wins'], 'o', alpha=0.4)
plt.xlabel("ReturnGamesWon")
plt.ylabel("Wins")
plt.show()
plt.plot(df['TotalPointsWon'], df['Wins'], 'o', alpha=0.4)
plt.xlabel("TotalPointsWon")
plt.ylabel("Wins")
plt.show()
plt.plot(df['FirstServe'], df['Wins'], 'o', alpha=0.4)
plt.xlabel("FirstServe")
plt.ylabel("Wins")
plt.show()
plt.plot(df['FirstServeReturnPointsWon'], df['Wins'], 'o', alpha=0.4)
plt.xlabel("FirstServeReturnPointsWon")
plt.ylabel("Wins")
plt.show()
plt.plot(df['BreakPointsFaced'], df['Wins'], 'o', alpha=0.4) #good feature
plt.xlabel("BreakPointsFaced")
plt.ylabel("Wins")
plt.show()
plt.plot(df['BreakPointsConverted'], df['Wins'], 'o', alpha=0.4) #good feature
plt.xlabel("BreakPointsConverted")
plt.ylabel("Wins")
plt.show()
plt.plot(df['BreakPointsOpportunities'], df['Wins'], 'o', alpha=0.4) #good feature
plt.xlabel("BreakPointsOpportunities")
plt.ylabel("Wins")
plt.show()
```


50
40
30
20
10
0
Wins
0
200
400
600
800
1000
1200
Aces

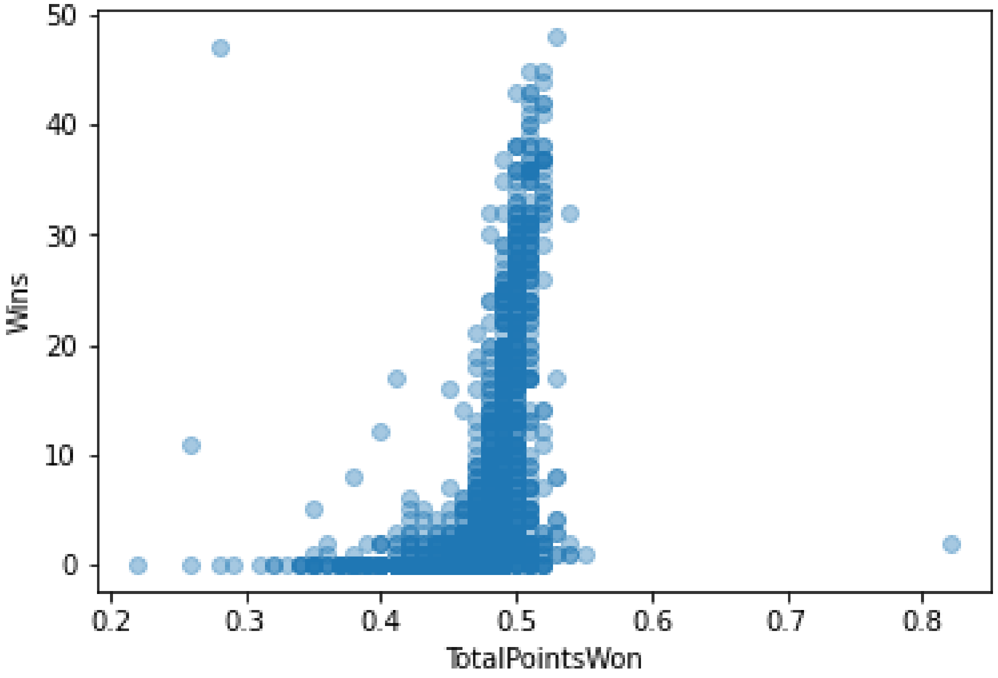

50
40
30
20
10
0
Wins
0
50
100
150
200
250
300
DoubleFaults


Wins
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
ServiceGamesWon

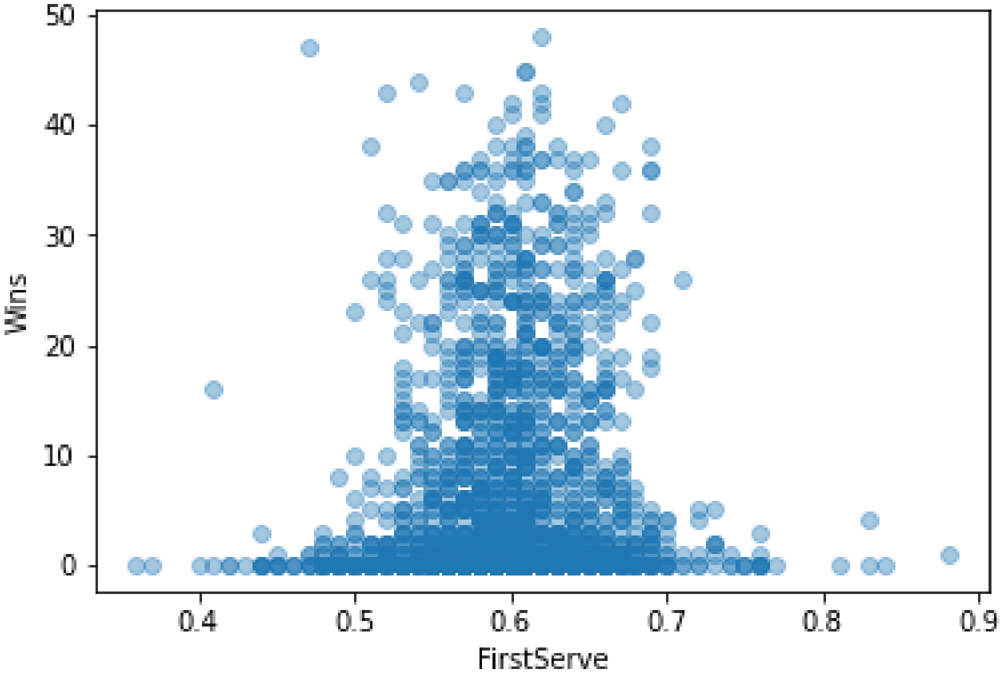

Wins
50
40
30
20
10
0
0.0
0.1
0.2
0.3
0.4
0.5
ReturnPointsWon

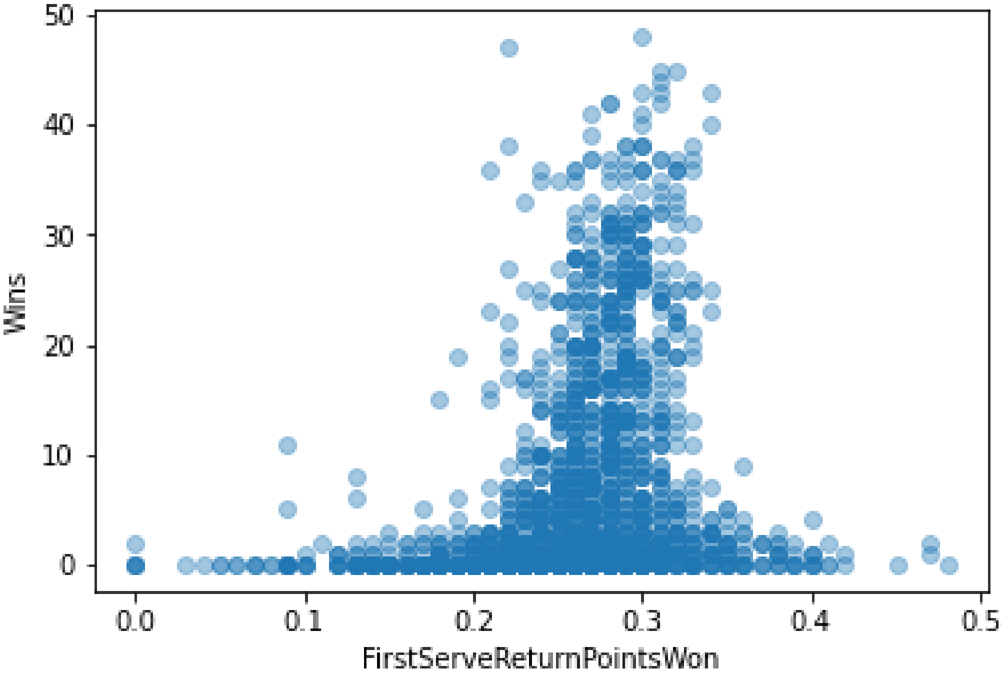


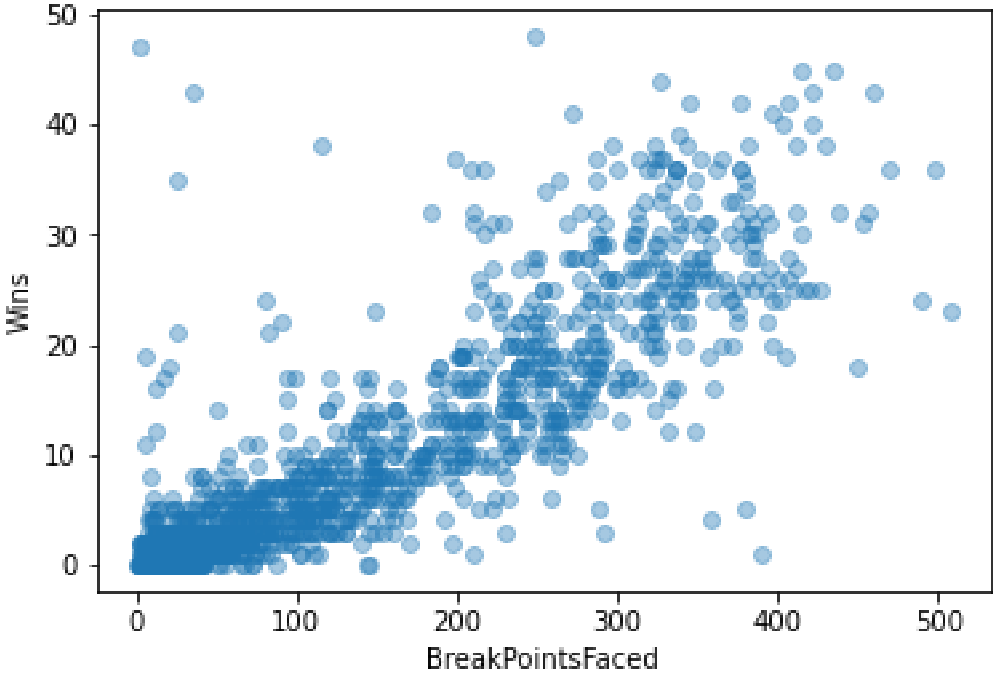
50
40
30
20
10
0
Wins
0.0
0.1
0.2
0.3
0.4
0.5
ReturnGamesWon

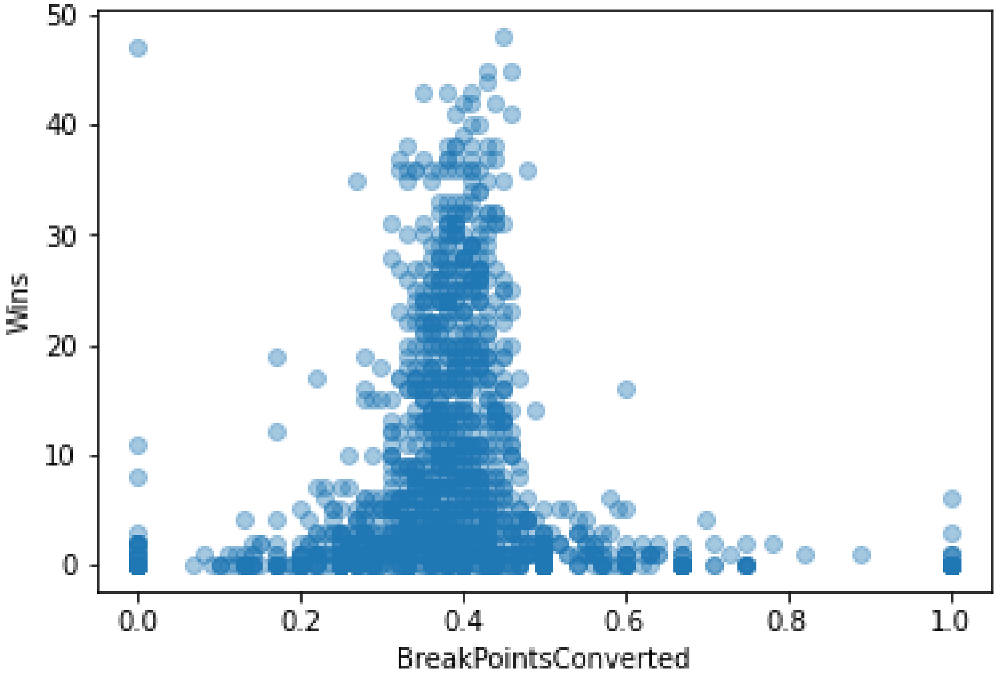


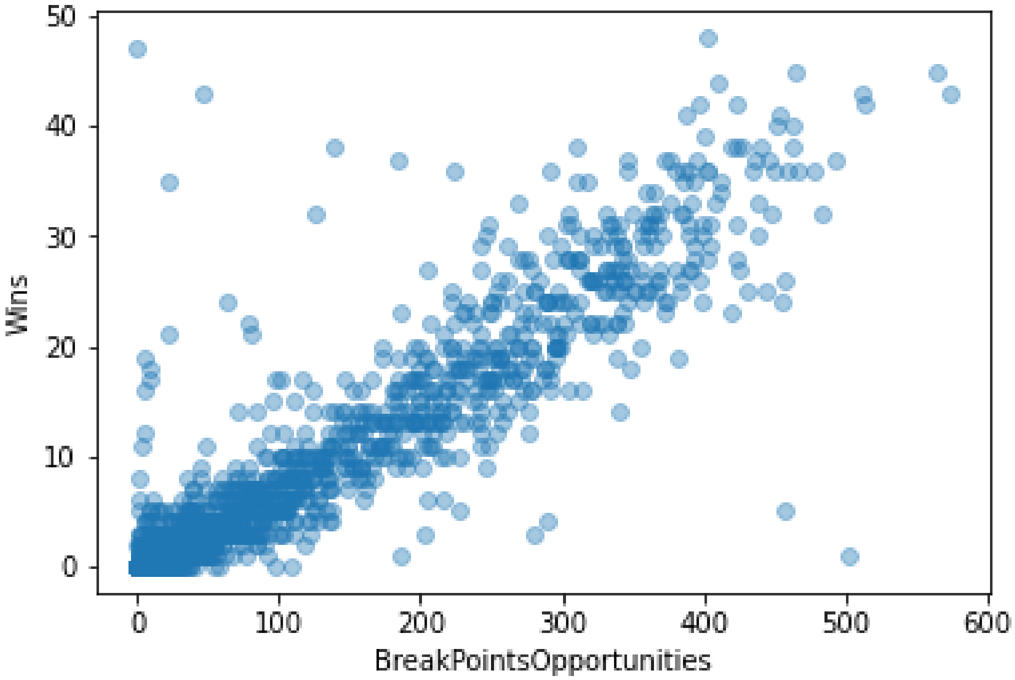
50
40
30
20
10
0
Wins
0.2
0.3
0.4
0.5
0.6
0.7
0.8
TotalPointsWon


Wins
0
10
20
30
40
50
0.4
0.5
0.6
0.7
0.8
0.9
FirstServe


Wins
0
10
20
30
40
50
0.0
0.1
0.2
0.3
0.4
0.5
FirstServeReturnPointsWon


Wins
50
40
30
20
10
0
0
100
200
300
400
500
BreakPointsFaced


Wins
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
BreakPointsConverted

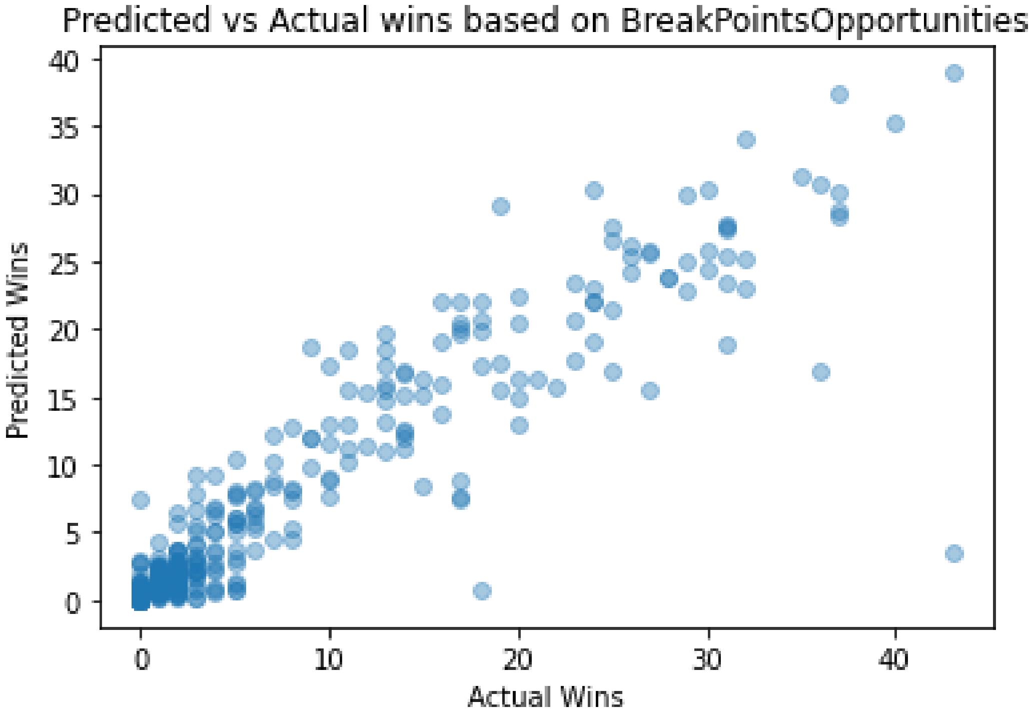


[35]:
```
# create train/test data split
features = df[['BreakPointsOpportunities']]
labels = df['Wins']

x_train, x_test, y_train, y_test = train_test_split(features, labels,␣
 ↪train_size=0.8)
```

[41]:
```
# Single feature linear regression
SingleLR = LinearRegression()
SingleLR.fit(x_train, y_train)
print("The line for BreakPoints Opportunities vs Wins has a slope of %.4f and␣
 ↪an intercept of %.4f" % (SingleLR.coef_, SingleLR.intercept_))
R_squared = SingleLR.score(x_test, y_test)
print("This model has an R^2 value of %.2f" % R_squared)
```

```
The line for BreakPoints Opportunities vs Wins has a slope of 0.0760 and an
intercept of -0.0094
This model has an R^2 value of 0.86
```

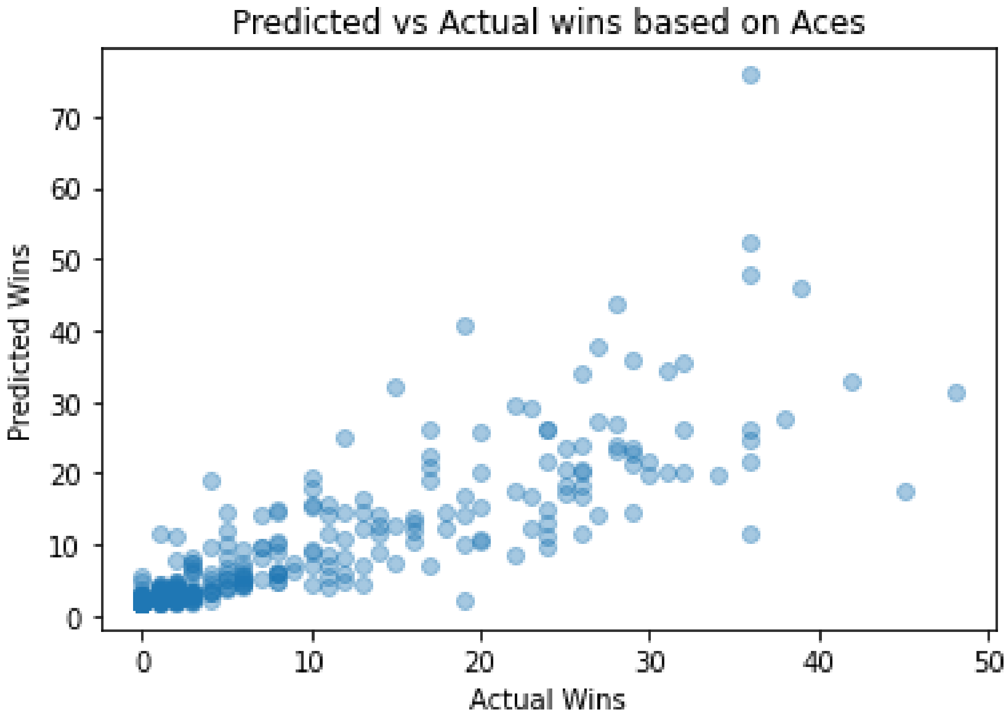
[45]:
```
prediction = SingleLR.predict(x_test)
plt.scatter(y_test, prediction, alpha=0.4)
plt.xlabel("Actual Wins")
plt.ylabel("Predicted Wins")
plt.title("Predicted vs Actual wins based on BreakPointsOpportunities")
plt.show()
```



[51]:

```
# New single feature linear regression
features = df[['Aces']]
x_train, x_test, y_train, y_test = train_test_split(features, labels,␣
 ↪train_size=0.8)
SingleLR = LinearRegression()
SingleLR.fit(x_train, y_train)
print("The line for Aces vs Wins has a slope of %.4f and an intercept of %.4f"␣
 ↪% (SingleLR.coef_, SingleLR.intercept_))
R_squared = SingleLR.score(x_test, y_test)
print("This model has an R^2 value of %.2f" % R_squared)
prediction = SingleLR.predict(x_test)
plt.scatter(y_test, prediction, alpha=0.4)
plt.xlabel("Actual Wins")
plt.ylabel("Predicted Wins")
plt.title("Predicted vs Actual wins based on Aces")
plt.show()
```

```
The line for Aces vs Wins has a slope of 0.0624 and an intercept of 1.8811
This model has an R^2 value of 0.70
```



[54]:

```
# Double feature linear regression
features = df[['BreakPointsOpportunities', 'BreakPointsFaced']]
labels = df[['Winnings']]
DoubleLR = LinearRegression()
x_train, x_test, y_train, y_test = train_test_split(features, labels,␣
 ↪train_size=0.8)
DoubleLR.fit(x_train, y_train)
r2 = DoubleLR.score(x_test, y_test)
print("R^2 for BreakPointsOpportunities and BreakPointsFaced vs Winnings is %.
 ↪2f" % r2)
features = df[['Aces', 'DoubleFaults']]
DoubleLR = LinearRegression()
x_train, x_test, y_train, y_test = train_test_split(features, labels,␣
 ↪train_size=0.8)
DoubleLR.fit(x_train, y_train)
r2 = DoubleLR.score(x_test, y_test)
print("R^2 for Aces and DoubleFaults vs Winnings is %.2f" % r2)
```

```
R^2 for BreakPointsOpportunities and BreakPointsFaced vs Winnings is 0.80
R^2 for Aces and DoubleFaults vs Winnings is 0.72
```

[56]:

```
print("BreakPoints Opportunities and BreakPoints Faced are better predictors␣
 ↪for Winnings than Aces and DoubleFaults")
```

BreakPoints Opportunities and BreakPoints Faced are better predictors for Winnings than Aces and DoubleFaults

[61]:
```
# multiple linear regression
from sklearn.preprocessing import StandardScaler
mlr = LinearRegression()
features = df.drop(columns=['Player', 'Wins', 'Losses', 'Winnings', 'Ranking'])
labels = df['Winnings']
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1721 entries, 0 to 1720
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Year                        1721 non-null   int64
 1   FirstServe                  1721 non-null   float64
 2   FirstServePointsWon         1721 non-null   float64
 3   FirstServeReturnPointsWon   1721 non-null   float64
 4   SecondServePointsWon        1721 non-null   float64
 5   SecondServeReturnPointsWon  1721 non-null   float64
 6   Aces                        1721 non-null   int64
 7   BreakPointsConverted        1721 non-null   float64
 8   BreakPointsFaced            1721 non-null   int64
 9   BreakPointsOpportunities    1721 non-null   int64
 10  BreakPointsSaved            1721 non-null   float64
 11  DoubleFaults                1721 non-null   int64
 12  ReturnGamesPlayed           1721 non-null   int64
 13  ReturnGamesWon              1721 non-null   float64
 14  ReturnPointsWon             1721 non-null   float64
 15  ServiceGamesPlayed          1721 non-null   int64
 16  ServiceGamesWon             1721 non-null   float64
 17  TotalPointsWon              1721 non-null   float64
 18  TotalServicePointsWon       1721 non-null   float64
dtypes: float64(12), int64(7)
memory usage: 255.6 KB
```

[65]:
```
scaler = StandardScaler()
scaledFeatures = scaler.fit_transform(features)
x_train, x_test, y_train, y_test = train_test_split(scaledFeatures, labels,␣
 ↪train_size=0.8)
mlr.fit(x_train, y_train)
r2 = mlr.score(x_test, y_test)
print("Using a standardized form of all numerical features, the R^2 value for␣
 ↪multiple linear regression predicting Winnings is %.2f" % r2)
```

```
Using a standardized form of all numerical features, the R^2 value for multiple
linear regression predicting Winnings is 0.82
```

[ ]: