In [220]:
```
import pandas as pd
import numpy as np
import urllib
import pymongo
from pymongo import MongoClient
from bs4 import BeautifulSoup
from bs4 import SoupStrainer
import lxml
import pickle
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

From experimentation, I found that www.pgatour.com stores the majority of their stats pages in the range of 101 - 300 in the path /stats/stat.NUMBER.YEAR.html - This makes it a great target for web scraping. Let's create a master dictionary of which number corresponds to which stat for 2018 to start

So, for instance I could load the dictionary and look for 'Driving Accuracy Percentage' and easily find which webpage to scrape in the range of 101 - 300

In [ ]:
```

```

In [2]:
```
##dont parse the whole page each time to gain a bit of extra performan
ce
stat_name_strainer = SoupStrainer("div", class_="breadcrumbs")
stat_read_dict = {}
stat_range_list = [] #create a list of stat pages that are valid on pg
atour.com to use later

for i in range(101, 300):
    stat_url = "https://www.pgatour.com/stats/stat.{0}.2018.html".form
at(i)
    print(i)
    try:
        web_page = urllib.request.urlopen(stat_url).read()
        soup = BeautifulSoup(web_page, "lxml", parse_only=stat_name_st
rainer)
        stat_name = soup.find("a", class_="current").text
        if stat_name != "":  #make sure we found data for that number
            stat_read_dict[stat_name] = i
            stat_range_list.append(i)
    except AttributeError:
        print("page number: {0} isn't there".format(i))
        continue
```

```
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
page number: 121 isn't there
122
123
page number: 123 isn't there
124
page number: 124 isn't there
125
page number: 125 isn't there
126
page number: 126 isn't there
127
128
page number: 128 isn't there
129
130
131
132
133
page number: 133 isn't there
134
page number: 134 isn't there
135
136
page number: 136 isn't there
137
138
139
140
141
```

```
page number: 141 isn't there
142
143
144
145
146
147
148
149
150
151
page number: 151 isn't there
152
153
154
155
156
157
page number: 157 isn't there
158
159
160
161
page number: 161 isn't there
162
page number: 162 isn't there
163
page number: 163 isn't there
164
page number: 164 isn't there
165
page number: 165 isn't there
166
page number: 166 isn't there
167
page number: 167 isn't there
168
page number: 168 isn't there
169
page number: 169 isn't there
170
page number: 170 isn't there
171
172
173
174
page number: 174 isn't there
175
page number: 175 isn't there
176
```

```
page number: 176 isn't there
177
page number: 177 isn't there
178
page number: 178 isn't there
179
page number: 179 isn't there
180
page number: 180 isn't there
181
page number: 181 isn't there
182
page number: 182 isn't there
183
page number: 183 isn't there
184
page number: 184 isn't there
185
page number: 185 isn't there
186
187
188
page number: 188 isn't there
189
page number: 189 isn't there
190
191
page number: 191 isn't there
192
page number: 192 isn't there
193
page number: 193 isn't there
194
195
page number: 195 isn't there
196
page number: 196 isn't there
197
page number: 197 isn't there
198
page number: 198 isn't there
199
200
page number: 200 isn't there
201
page number: 201 isn't there
202
page number: 202 isn't there
203
page number: 203 isn't there
```

```
204
page number: 204 isn't there
205
page number: 205 isn't there
206
page number: 206 isn't there
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
```

```
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
page number: 287 isn't there
288
page number: 288 isn't there
289
page number: 289 isn't there
290
page number: 290 isn't there
291
page number: 291 isn't there
292
293
294
295
```

```
296
297
298
299
```

Once we have retrieved which page number stores which stat, make sure we store the info locally (python's pickle library works perfectly here). Future versions of this project will use a database

**Note on database type: Because each stat page contains differently structured data, I most likely want to use a noSQL database to store and return entries. I can index by player name and year (which together are unique for each stat type)

In [4]:
```
##Save the stat dict
with open('golf_page_dict.pickle', 'wb') as handle:
    pickle.dump(stat_read_dict, handle, protocol=pickle.HIGHEST_PROTOC
OL)

##Save the list of valid page numbers for later searching
with open('golf_stat_list.pickle', 'wb') as handle:
    pickle.dump(stat_range_list, handle, protocol=pickle.HIGHEST_PROTO
COL)
```

In [191]:
```
##Open the golf page dict
golf_stat_dict = {}
with open('golf_page_dict.pickle', 'rb') as handle:
    golf_stat_dict = pickle.load(handle)
golf_stat_dict
```

Out[191]:
```
{"3-Putt Avoidance - 15-20'": 145,
 "3-Putt Avoidance - 20-25'": 146,
 "3-Putt Avoidance > 25'": 147,
 'All-Around Ranking': 127,
 'Back 9 Par 3 Scoring Average': 222,
 'Back 9 Par 4 Scoring Average': 230,
 'Back 9 Par 5 Scoring Average': 238,
 'Back 9 Round 1 Scoring Average': 246,
 'Back 9 Round 2 Scoring Average': 254,
 'Back 9 Round 3 Scoring Average': 262,
 'Back 9 Round 4 Scoring Average': 270,
 'Back 9 Round 5 Scoring Average': 278,
 'Back 9 Scoring Average': 208,
 'Ball Striking': 158,
 'Best YTD 1-Putt or Better Streak': 295,
 'Best YTD Streak w/o a 3-Putt': 294,
 'Birdie Average': 156,
 'Birdie or Better Conversion Percentage': 115,
```

```
'Bounce Back': 160,
'Career Money Leaders': 110,
'Consecutive Cuts': 122,
'Consecutive Fairways Hit': 297,
'Consecutive GIR': 298,
'Consecutive Sand Saves': 296,
'Current Par or Better Streak': 150,
'Driving Accuracy Percentage': 102,
'Driving Distance': 101,
'Driving Pct. 240-260 (All Drives)': 217,
'Driving Pct. 260-280 (All Drives)': 216,
'Driving Pct. 280-300 (All Drives)': 215,
'Driving Pct. 300+ (All Drives)': 214,
'Driving Pct. <= 240 (All Drives)': 218,
'Eagles (Holes per)': 155,
'Early Par 3 Scoring Average': 223,
'Early Par 4 Scoring Average': 231,
'Early Par 5 Scoring Average': 239,
'Early Round 1 Scoring Average': 247,
'Early Round 2 Scoring Average': 255,
'Early Round 3 Scoring Average': 263,
'Early Round 4 Scoring Average': 271,
'Early Round 5 Scoring Average': 279,
'Early Scoring Average': 292,
'Final Round Performance': 219,
'Final Round Scoring Average': 118,
'First Tee Early Par 3 Scoring Average': 225,
'First Tee Early Par 4 Scoring Average': 233,
'First Tee Early Par 5 Scoring Average': 241,
'First Tee Early Round 1 Scoring Average': 249,
'First Tee Early Round 2 Scoring Average': 257,
'First Tee Early Round 3 Scoring Average': 265,
'First Tee Early Round 4 Scoring Average': 273,
'First Tee Early Round 5 Scoring Average': 281,
'First Tee Early Scoring Average': 209,
'First Tee Late Par 3 Scoring Average': 227,
'First Tee Late Par 4 Scoring Average': 235,
'First Tee Late Par 5 Scoring Average': 243,
'First Tee Late Round 1 Scoring Average': 251,
'First Tee Late Round 2 Scoring Average': 259,
'First Tee Late Round 3 Scoring Average': 267,
'First Tee Late Round 4 Scoring Average': 275,
'First Tee Late Round 5 Scoring Average': 283,
'First Tee Late Scoring Average': 211,
'Front 9 Par 3 Scoring Average': 221,
'Front 9 Par 4 Scoring Average': 229,
'Front 9 Par 5 Scoring Average': 237,
'Front 9 Round 1 Scoring Average': 245,
'Front 9 Round 2 Scoring Average': 253,
'Front 9 Round 3 Scoring Average': 261,
```

```
            'Front 9 Round 4 Scoring Average': 269,
            'Front 9 Round 5 Scoring Average': 277,
            'Front 9 Scoring Average': 207,
            'GIR Percentage from Fairway': 190,
            'GIR Percentage from Other than Fairway': 199,
            'Greens in Regulation Percentage': 103,
            'Hit Fairway Percentage': 213,
            'Late Par 3 Scoring Average': 224,
            'Late Par 4 Scoring Average': 232,
            'Late Par 5 Scoring Average': 240,
            'Late Round 1 Scoring Average': 248,
            'Late Round 2 Scoring Average': 256,
            'Late Round 3 Scoring Average': 264,
            'Late Round 4 Scoring Average': 272,
            'Late Round 5 Scoring Average': 280,
            'Late Scoring Average': 293,
            'Longest Drives': 159,
            'Lowest Round': 299,
            'Money per Event Leaders': 154,
            'Non-member Earnings': 139,
            'Official Money': 109,
            'Official World Golf Ranking': 186,
            'PGA Championship Points': 132,
            'Par 3 Birdie or Better Leaders': 112,
            'Par 3 Performance': 171,
            'Par 3 Scoring Average': 142,
            'Par 4 Birdie or Better Leaders': 113,
            'Par 4 Performance': 172,
            'Par 4 Scoring Average': 143,
            'Par 5 Birdie or Better Leaders': 114,
            'Par 5 Performance': 173,
            'Par 5 Scoring Average': 144,
            'Par Breakers': 105,
            'Presidents Cup (International)': 187,
            'Presidents Cup (United States)': 140,
            'Putting Average': 104,
            'Putts Per Round': 119,
            'Putts made Distance': 135,
            'Round 1 Scoring Average': 148,
            'Round 2 Scoring Average': 149,
            'Round 3 Scoring Average': 117,
            'Round 4 Scoring Average': 285,
            'Round 5 Scoring Average': 286,
            "Rounds in the 60's": 152,
            'Ryder Cup Points': 131,
            'Sand Save Percentage': 111,
            'Scoring Average': 120,
            'Scoring Average (Actual)': 108,
            'Scoring Average Before Cut': 116,
            'Scrambling': 130,
```

```
'Sub-Par Rounds': 153,
'Tenth Tee Early Par 3 Scoring Average': 226,
'Tenth Tee Early Par 4 Scoring Average': 234,
'Tenth Tee Early Par 5 Scoring Average': 242,
'Tenth Tee Early Round 1 Scoring Average': 250,
'Tenth Tee Early Round 2 Scoring Average': 258,
'Tenth Tee Early Round 3 Scoring Average': 266,
'Tenth Tee Early Round 4 Scoring Average': 274,
'Tenth Tee Early Round 5 Scoring Average': 282,
'Tenth Tee Early Scoring Average': 210,
'Tenth Tee Late Par 3 Scoring Average': 228,
'Tenth Tee Late Par 4 Scoring Average': 236,
'Tenth Tee Late Par 5 Scoring Average': 244,
'Tenth Tee Late Round 1 Scoring Average': 252,
'Tenth Tee Late Round 2 Scoring Average': 260,
'Tenth Tee Late Round 3 Scoring Average': 268,
'Tenth Tee Late Round 4 Scoring Average': 276,
'Tenth Tee Late Round 5 Scoring Average': 284,
'Tenth Tee Late Scoring Average': 212,
'Top 10 Final Round Performance': 220,
'Top 10 Finishes': 138,
'Total Birdies': 107,
'Total Driving': 129,
'Total Eagles': 106,
'Total Money (Official and Unofficial)': 194,
'YTD Consecutive Cuts': 137}
```

So I can start doing some web scraping across different stats and years now. I have the information to find specific stats via the golf_stat_dict + URL and can select years easily as well

Let's try to figure out who made the most money per event from 2010 - 2018, something that the PGA tour's website doesn't let you easily do.

'Money per Event Leaders' is the stat that we want to look at

```
In [ ]:
```

In [199]:
```
years_to_check = ['2018','2017', '2016', '2015', '2014', '2013', '2012
', '2011', '2010']
multiple_year_stat = []

def df_stat_retriever(stat_name):
    stat_num = golf_stat_dict[stat_name]

    for year in years_to_check:
        stat_url = "https://www.pgatour.com/stats/stat.{0}.{1}.html".f
ormat(stat_num, year)
        search_dict = {'id' : 'statsTable'} #Each page has a statsTabl
e ID on the table header
        stat_data = pd.read_html(stat_url, attrs = search_dict) #panda
s has a nifty way to scrape an html table directly off of a page
        stat_data[0]['YEAR'] = year
        multiple_year_stat.append(stat_data[0])


    return pd.concat(multiple_year_stat)
```

In [185]:
```
multiple_year_stat_dframe = df_stat_retriever('Money per Event Leaders
')
```

In [188]:
```
#Some data cleaning to get rid of commas, get rid of dollar signs and
convert to int
multiple_year_stat_dframe['MONEY PER EVENT'] = multiple_year_stat_dfra
me['MONEY PER EVENT'].replace({'\$':'', ',':''}, regex=True).astype(in
t)
multiple_year_stat_dframe['TOTAL MONEY'] = multiple_year_stat_dframe['
TOTAL MONEY'].replace({'\$':'', ',':''}, regex=True).astype(int)
```

In [189]:
```
#Average money made per event across the tour over the last 8 years:
print(multiple_year_stat_dframe['MONEY PER EVENT'].mean()) #--> $45,09
7.74

#Average total money made per year across the tour over the last 8 yea
rs:
print(multiple_year_stat_dframe['TOTAL MONEY'].mean()) #--> $837,846.6
1

#20 highest earning years for players
print(multiple_year_stat_dframe.nlargest(20, 'TOTAL MONEY'))

# 1) Jordan Speith, 2015, $12,030,464
# 2) Justin Thomas, 2017, $9,921,559
# 3) Jordan Spieth, 2017, $9,433,032
# Interestingly, Tiger woods shows up on this list once as number 8 (2
013),
# but has the highest money earned per event because he only played
# 16 events, tied for the lowest number
```

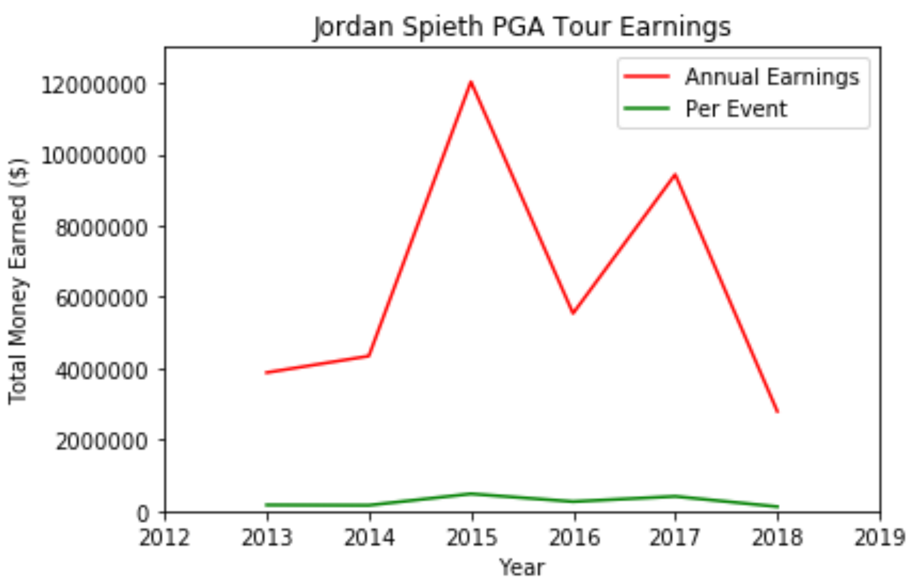
```
45097.740093603745
837846.6053042122
  RANK THIS WEEK RANK LAST WEEK        PLAYER NAME  EVENTS  MONEY PE
R EVENT  \
0              1              1      Jordan Spieth      25
481218
2              3            NaN      Justin Thomas      25
396862
1              2            NaN      Jordan Spieth      23
410131
1              2              2          Jason Day      20
470166
0              1              1     Dustin Johnson      22
425690
0              1            NaN     Dustin Johnson      20
436609
3              4              4      Justin Thomas      23
378035
0              1              1        Tiger Woods      16
534589
1              2              2     Dustin Johnson      20
422867
3              4            NaN   Hideki Matsuyama      22
380934
0              1              1       Rory McIlroy      17
487064
0              1              1        Justin Rose      18
451704
4              5              5  Bryson DeChambeau      26
```

```
311326
1             2              2        Rory McIlroy      16
502996
1             2              2           Jason Day      20
402255
2             3              3       Brooks Koepka      17
417296
3             4              4        Bubba Watson      19
361936
1             2              2         Luke Donald      19
351748
2             3              3          Adam Scott      20
323654
1             2              2      Henrik Stenson      18
354901

   TOTAL MONEY  YEAR
0     12030464  2015
2      9921559  2017
1      9433032  2017
1      9403330  2015
0      9365184  2016
0      8732193  2017
3      8694821  2018
0      8553438  2013
1      8457351  2018
3      8380569  2017
0      8280095  2014
0      8130677  2018
4      8094489  2018
1      8047951  2012
1      8045111  2016
2      7094047  2018
3      6876797  2015
1      6683214  2011
2      6473089  2016
1      6388229  2013
```

Simple visualization: How did total money and money earned per event fluctuate for Jordan Speith (arguably the most successful player of the last 5 years)?

In [180]:
```
spieth_dframe = multiple_year_stat_dframe.loc[multiple_year_stat_dframe['PLAYER NAME'] == 'Jordan Spieth'].sort_values('YEAR', ascending=False)
print(spieth_dframe)

plt.plot(spieth_dframe['YEAR'], spieth_dframe['TOTAL MONEY'], 'r-')
plt.plot(spieth_dframe['YEAR'], spieth_dframe['MONEY PER EVENT'], 'g-')
plt.axis([2012, 2019, 0, 13000000])
plt.ticklabel_format(style='plain', axis='y')
plt.ylabel('Total Money Earned ($)')
plt.xlabel('Year')
plt.title('Jordan Spieth PGA Tour Earnings')
plt.legend(['Annual Earnings', 'Per Event'])
plt.show()
```

```
   RANK THIS WEEK RANK LAST WEEK    PLAYER NAME  EVENTS  MONEY PER E
VENT  \
36             37             37  Jordan Spieth      23            12
1458
1               2            NaN  Jordan Spieth      23            41
0131
4               5              5  Jordan Spieth      21            26
3736
0               1              1  Jordan Spieth      25            48
1218
15             16             16  Jordan Spieth      27            16
0842
9              10             10  Jordan Spieth      23            16
8687

    TOTAL MONEY  YEAR
36      2793536  2018
1       9433032  2017
4       5538470  2016
0      12030464  2015
15      4342748  2014
9       3879819  2013
```



We can now see how Spieth's money earned has fluctuated over the last few years (his entire career). Obviously 2015 was a breakout year for him in terms of total money and money earned per event.

Lastly, I want to make a hypothesis about which statistic on the PGA tour accurately predicts total money earned for a given year. I'll use a simple linear regression.

I am curious about how impactful 'Greens in Regulation Percentage' as a statistic is on the PGA tour. Greens in Regulation Percentage is a statistic measuring how often a player makes it onto a green with a chance for a birdie or better

In [241]:

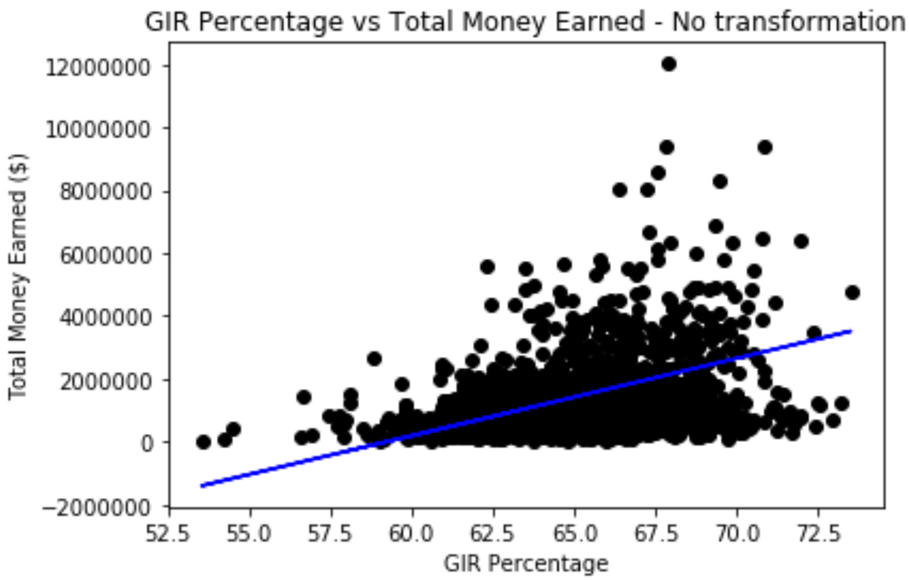
```
df_GIR_percentage = df_stat_retriever('Greens in Regulation Percentage
')
```

In [219]:

```
# To set up the side-by-side comparison of Greens in Regulation for a
given year vs Money Earned
# we have to essentially perform an inner join across the 2 dataframes
.  This can be done very
# easily with pandas.merge
merged_df = pd.merge(df_GIR_percentage, multiple_year_stat_dframe, on
= ['PLAYER NAME', 'YEAR'])
merged_df = merged_df[['PLAYER NAME', '%', 'YEAR', 'TOTAL MONEY']] #ge
t rid of the uneeded columns that are NaN now
```

In [240]:
```
# To train a linear model and see how well it predicts
# my data, I will split my dataset 20:80, 20% training
# data, 80% predictive data

#merged_df.describe --> 1678 total rows.  20% = 335, 80% = 1343
stats_train = merged_df.head(335)
stats_X_train = stats_train['%']
stats_Y_train = stats_train['TOTAL MONEY']

stats_test = merged_df.tail(1343)
stats_X_test = stats_test['%']
stats_Y_test = stats_test['TOTAL MONEY']

regr = linear_model.LinearRegression()
regr.fit(stats_X_train.values.reshape(-1, 1), stats_Y_train)

#Make a prediction using the testing set
stats_Y_pred = regr.predict(stats_X_test.values.reshape(-1, 1))

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(stats_Y_test, stats_Y_pred))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(stats_Y_test, stats_Y_pred))

plt.ticklabel_format(style='plain', axis='y')
plt.title('GIR Percentage vs Total Money Earned - No transformation')
plt.ylabel('Total Money Earned ($)')
plt.xlabel('GIR Percentage')
plt.scatter(stats_X_test, stats_Y_test, color='black')
plt.plot(stats_X_test, stats_Y_pred, color='blue')
plt.show()
```

```
Coefficients:
 [ 246105.57765697]
Mean squared error: 1683155039981.29
Variance score: 0.03
```



Coefficients: [246105.57765697], Mean squared error: 1683155039981.29, Variance score: 0.03

We can see that based on this model, Greens in Regulation Percentage is not a great predictor of Total Money Earned. From the variance score, only 3% of the variance in Total Money is explained by GIR percentage.

One way to possibly improve this model would be to transform the GIR percentage to spread the data out. Ie, instead of it being in a possible range of 0-100, we may expand the range and see some separation for our model to predict.

In [ ]: