Exploratory Data Analysis - Tennis

December 1, 2020

Load in required packages:

```
[1]: import pandas as pd
     import numpy as np
     from scipy import stats

     import seaborn as sns
     import matplotlib.pyplot as plt
     from IPython.display import set_matplotlib_formats

     %matplotlib inline
     set_matplotlib_formats('pdf', 'svg')
```

1 Dataset Summary

This project focuses on data from ATP matches in 2019 (obtained from https://github.com/JeffSackmann/tennis_atp). The dataset contains match statistics, including from both the match winner and loser, including age and height as well as aces, double faults, 1st serve and 2nd serve wins, and more.

The variables with integers (number of points/shots) for match stats will be converted to percentages to account for different match lengths. "Winner" is coded with the prefix "w" and "Loser" is coded with the prefix "l".

The overall aim is to explore which match factors are related to **a) the likelihood of winning a match** and **b) the win strength** (or the % of total points won).

Read data:

```
[2]: tennis_df = pd.read_csv('atp_matches_2019.csv')
     print('Raw data contains information on',tennis_df.shape[0],'tennis matches␣
      ↪with',tennis_df.shape[1],'variables\n')
     print('Columns =',tennis_df.columns.tolist())
```

```
Raw data contains information on 2781 tennis matches with 49 variables

Columns = ['tourney_id', 'tourney_name', 'surface', 'draw_size',
'tourney_level', 'tourney_date', 'match_num', 'winner_id', 'winner_seed',
'winner_entry', 'winner_name', 'winner_hand', 'winner_ht', 'winner_ioc',
```

'winner_age', 'loser_id', 'loser_seed', 'loser_entry', 'loser_name', 'loser_hand', 'loser_ht', 'loser_ioc', 'loser_age', 'score', 'best_of', 'round', 'minutes', 'w_ace', 'w_df', 'w_svpt', 'w_1stIn', 'w_1stWon', 'w_2ndWon', 'w_SvGms', 'w_bpSaved', 'w_bpFaced', 'l_ace', 'l_df', 'l_svpt', 'l_1stIn', 'l_1stWon', 'l_2ndWon', 'l_SvGms', 'l_bpSaved', 'l_bpFaced', 'winner_rank', 'winner_rank_points', 'loser_rank', 'loser_rank_points']

2 Plan for Data Exploration

To explore the data, I'm going to first filter by variables that I am interested in, rename them, and convert the data to long format so that each row refers to a player, not a match. Next, I will deal with missing values (NaNs) and plot histograms and boxplots of my variables to investigate their distribution and any potential outliers. Where needed, variables will be transformed to optimise model performance. Finally, I'm going to explore correlations between my variables with scatter plots and I will summarise their main attributes separately by match winners and losers.

3 Data Cleaning and Feature Engineering

3.1 Format variables of interest

Make sure that the data does not contain any duplicate rows:

[3]:
```
n_dup = sum(tennis_df.duplicated())
if n_dup == 0:
    print('No duplicate rows in dataset')
```

```
No duplicate rows in dataset
```

Filter by the variables of interest:

[4]:
```
# select variables of interest
my_vars = ['winner_name','winner_ht','winner_age',
           'loser_name','loser_ht','loser_age',
           'w_svpt','w_ace','w_df','w_1stIn','w_1stWon','w_2ndWon',
           'l_svpt','l_ace','l_df','l_1stIn','l_1stWon','l_2ndWon']
tennis_df = tennis_df[my_vars]

# rename some variables for consistency and for long formatting later:
tennis_df.rename(columns={"winner_name": "name_w", "winner_ht": "height_w",␣
 ↪"winner_age": "age_w",
                          "w_svpt": "svpt_w", "w_ace": "ace_w", "w_df": "df_w",
                          "w_1stIn": "1stIn_w", "w_1stWon": "1stWon_w",␣
 ↪"w_2ndWon": "2ndWon_w",
                          "loser_name": "name_l", "loser_ht": "height_l",␣
 ↪"loser_age": "age_l",
                          "l_svpt": "svpt_l", "l_ace": "ace_l", "l_df": "df_l",
                          "l_1stIn": "1stIn_l", "l_1stWon": "1stWon_l",␣
 ↪"l_2ndWon": "2ndWon_l",},
```

```
                inplace=True)
tennis_df.head()
```

[4]:
```
              name_w  height_w     age_w                name_l  height_l  \
0      Kei Nishikori     178.0  29.004791      Daniil Medvedev       NaN
1    Daniil Medvedev       NaN  22.885695   Jo-Wilfried Tsonga     188.0
2      Kei Nishikori     178.0  29.004791        Jeremy Chardy     188.0
3 Jo-Wilfried Tsonga     188.0  33.705681       Alex De Minaur       NaN
4    Daniil Medvedev       NaN  22.885695         Milos Raonic     196.0

      age_l  svpt_w  ace_w  df_w  1stIn_w  1stWon_w  2ndWon_w  svpt_l  ace_l  \
0 22.885695    77.0    3.0   3.0     44.0      31.0      17.0   100.0    8.0
1 33.705681    52.0   10.0   1.0     33.0      28.0      14.0    77.0   17.0
2 31.882272    47.0    2.0   2.0     33.0      26.0       9.0    46.0   10.0
3 19.868583    68.0   12.0   2.0     43.0      34.0      15.0    81.0    1.0
4 28.010951   105.0   12.0   3.0     68.0      48.0      25.0    94.0   29.0

   df_l  1stIn_l  1stWon_l  2ndWon_l
0   6.0     54.0      34.0      20.0
1   2.0     52.0      36.0       7.0
2   3.0     27.0      15.0       6.0
3   2.0     60.0      38.0       9.0
4   5.0     56.0      46.0      19.0
```

Convert integers to percentages using the number of service points per player (w/l_svpt):

[5]:
```
# winner
tennis_df['ace_per_w'] = (tennis_df['ace_w'] / tennis_df['svpt_w'].tolist()) *␣
 ↪100
tennis_df['df_per_w'] = (tennis_df['df_w'] / tennis_df['svpt_w'].tolist()) * 100
tennis_df['1stIn_per_w'] = (tennis_df['1stIn_w'] / tennis_df['svpt_w'].
 ↪tolist()) * 100
tennis_df['1stWon_per_w'] = (tennis_df['1stWon_w'] / tennis_df['1stIn_w'].
 ↪tolist()) * 100

# loser
tennis_df['ace_per_l'] = (tennis_df['ace_l'] / tennis_df['svpt_l'].tolist()) *␣
 ↪100
tennis_df['df_per_l'] = (tennis_df['df_l'] / tennis_df['svpt_l'].tolist()) * 100
tennis_df['1stIn_per_l'] = (tennis_df['1stIn_l'] / tennis_df['svpt_l'].
 ↪tolist()) * 100
tennis_df['1stWon_per_l'] = (tennis_df['1stWon_l'] / tennis_df['1stIn_l'].
 ↪tolist()) * 100
```

Add variable for the percentage of total points won by the winner (and loser):

```
[6]: total_points = tennis_df['svpt_w'] + tennis_df['svpt_l']
     # points won includes 1st and 2nd serve wins on their serve, and wins on other␣
      ↪serve
     w_points = tennis_df['1stWon_w'] + tennis_df['2ndWon_w'] + (tennis_df['svpt_l']␣
      ↪- tennis_df['1stWon_l'] - tennis_df['2ndWon_l'])
     l_points = tennis_df['1stWon_l'] + tennis_df['2ndWon_l'] + (tennis_df['svpt_w']␣
      ↪- tennis_df['1stWon_w'] - tennis_df['2ndWon_w'])

     tennis_df['pt_per_w'] = (w_points / total_points) * 100
     tennis_df['pt_per_l'] = (l_points / total_points) * 100
```

Filter the df again to include only those variables I want to explore:

```
[7]: tennis_df.drop(['svpt_w', 'ace_w', 'df_w', '1stIn_w', '1stWon_w', '2ndWon_w',
                     'svpt_l', 'ace_l', 'df_l', '1stIn_l', '1stWon_l', '2ndWon_l'],
                     axis=1, inplace=True)
```

3.2 Convert to long format

To ensure that the data is in the most useful format for further exploration, I'm going to convert it to code won/lost as a variable, so that each row represents a player's stats and not the match stats:

```
[8]: # add unique id per match:
     tennis_df["match_id"] = range(tennis_df.shape[0])

     # long format
     tennis_df_long = pd.wide_to_long(tennis_df,
                                      stubnames=["name","height","age",
                                                                          ␣
      ↪"ace_per","df_per","1stIn_per","1stWon_per","pt_per"],
                                      i="match_id", j="outcome",
                                      sep="_", suffix='\w+')
     tennis_df_long.reset_index(inplace=True)
     tennis_df_long.head()
```

```
[8]:    match_id outcome                name  height        age    ace_per  \
     0         0       w       Kei Nishikori   178.0  29.004791   3.896104
     1         1       w     Daniil Medvedev     NaN  22.885695  19.230769
     2         2       w       Kei Nishikori   178.0  29.004791   4.255319
     3         3       w  Jo-Wilfried Tsonga   188.0  33.705681  17.647059
     4         4       w     Daniil Medvedev     NaN  22.885695  11.428571

          df_per  1stIn_per  1stWon_per     pt_per
     0  3.896104  57.142857   70.454545  53.107345
     1  1.923077  63.461538   84.848485  58.914729
     2  4.255319  70.212766   78.787879  64.516129
```

```
3  2.941176  63.235294   79.069767  55.704698
4  2.857143  64.761905   70.588235  51.256281
```

Variable key:
* match_id = unique code for the match (so shared by two rows - one for winner, one for loser) * outcome = whether the player won (w) or lost (l) * ace_per = percentage of service points that were aces * df_per = percentage of service points that were double faults * 1stIn_per = percentage of service points where the first serve was in * 1stWon_per = percentage of 1st serves where the point was won * pt_per = percentage of total match points that the player won

3.3 Remove missing values

[9]:
```
# How many NaNs do we have in the data:
tennis_df_long.isna().sum()
```

[9]:
```
match_id         0
outcome          0
name             0
height        2527
age              2
ace_per        204
df_per         204
1stIn_per      204
1stWon_per     204
pt_per         204
dtype: int64
```

The above numbers reveal that "height" has a lot of missing data, and so will be a particularly poor variable to include, so I'm going to drop that from the analysis:

[10]:
```
tennis_df_long.drop(['height'], axis=1, inplace=True)
```

Luckily, there is a lot of data, so I'm going to simply remove any rows that contain a NaN for any variable:

[11]:
```
tennis_df_long.dropna(inplace=True)
```

[12]:
```
print('Data frame contains',len(tennis_df_long["match_id"].unique()),'matches␣
↪with complete data, including',tennis_df_long.shape[0],'players\n')
tennis_df_long.head()
```

```
Data frame contains 2679 matches with complete data, including 5358 players
```

[12]:
```
   match_id outcome              name        age    ace_per    df_per  \
0         0       w      Kei Nishikori  29.004791   3.896104  3.896104
1         1       w    Daniil Medvedev  22.885695  19.230769  1.923077
2         2       w      Kei Nishikori  29.004791   4.255319  4.255319
```

```
3          3       w  Jo-Wilfried Tsonga  33.705681  17.647059  2.941176
4          4       w     Daniil Medvedev  22.885695  11.428571  2.857143

   1stIn_per  1stWon_per     pt_per
0  57.142857   70.454545  53.107345
1  63.461538   84.848485  58.914729
2  70.212766   78.787879  64.516129
3  63.235294   79.069767  55.704698
4  64.761905   70.588235  51.256281
```

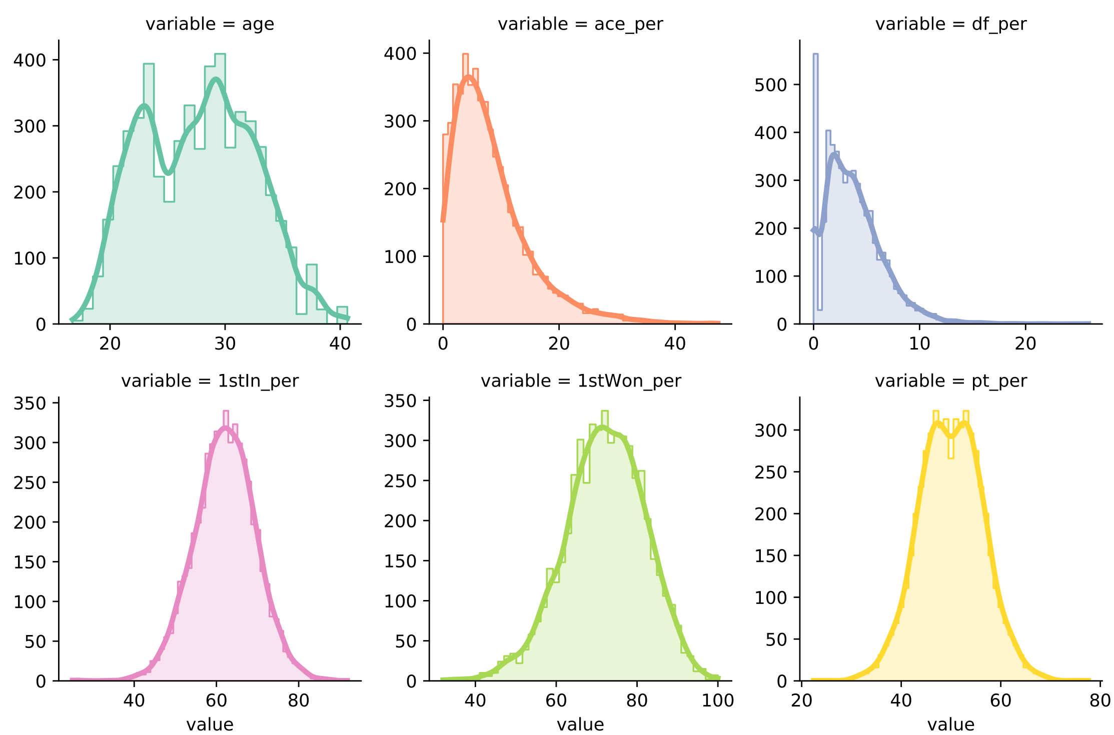
3.4 Distributions and data transformation

Now that the data is in a usable format, I'm going to inspect each variable to check for any outliers, or skewed distributions:

[13]:
```
melted_df = tennis_df_long.select_dtypes("float").melt()

grid = sns.FacetGrid(melted_df, col='variable', col_wrap=3,
                     hue='variable', palette='Set2',
                     height=3, aspect=1,
                     sharex = False, sharey = False)
grid.map(sns.histplot, "value", alpha=.25,
         kde=True, line_kws={"lw":3}, element="step")
```

[13]:
```
<seaborn.axisgrid.FacetGrid at 0x10a3a8fd0>
```



From the above plots, I can see that most of the variables are normally distributed. However, aces and double faults are substantially skewed, with most players making few of either in a match (a large number of players never hit a double fault). Therefore, I'm going to a) remove rows that have 0 aces or double faults and b) log-transform those variables:

[14]:
```
tennis_df_long = tennis_df_long[(tennis_df_long["ace_per"] > 0) &␣
↪(tennis_df_long["df_per"] > 0)].copy()
print('New data frame contains information for',tennis_df_long.
↪shape[0],'players')
```

```
New data frame contains information for 4567 players
```

[15]:
```
# return the skew of my variables (> |.75| indicates skew that may need␣
↪transforming)
skew_df = pd.DataFrame(stats.skew(tennis_df_long.select_dtypes("float")),
             index = tennis_df_long.select_dtypes("float").columns)
skew_df.rename(columns={0:"Skew"}, inplace=True)
skew_df.loc[skew_df["Skew"].abs() > .75,'Transform'] = '*'
skew_df
```

[15]:

	Skew	Transform
age	0.104424	NaN
ace_per	1.543465	*
df_per	1.438449	*
1stIn_per	-0.088851	NaN
1stWon_per	-0.211778	NaN
pt_per	0.006292	NaN

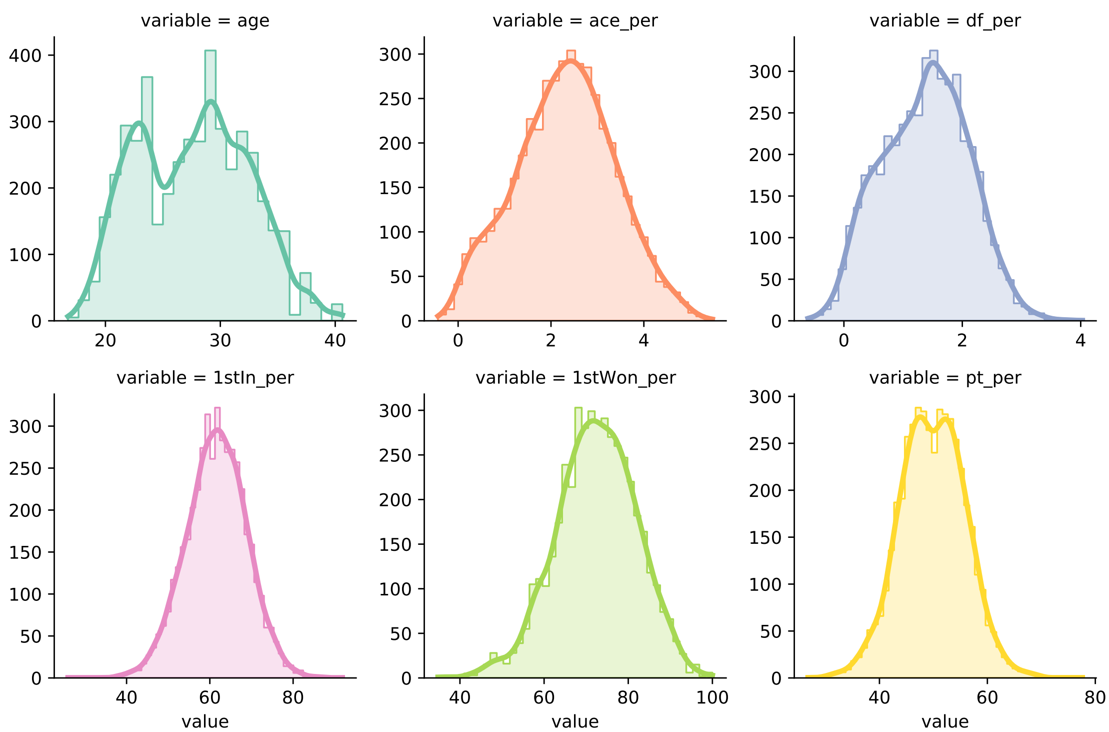
[16]:
```
# log transform:
log_vars = skew_df.index[skew_df["Transform"]=='*']
for v in log_vars:
    tennis_df_long[v] = stats.boxcox(tennis_df_long[v])[0]
```

Re-plot the log-transformed values:

[17]:
```
melted_df = tennis_df_long.select_dtypes("float").melt()

grid = sns.FacetGrid(melted_df, col='variable', col_wrap=3,
                     hue='variable', palette='Set2',
                     height=3, aspect=1,
                     sharex = False, sharey = False)
grid.map(sns.histplot, "value", alpha=.25,
         kde=True, line_kws={"lw":3}, element="step")
```

[17]: <seaborn.axisgrid.FacetGrid at 0x120922ba8>



Now all of the above variables appear to be normally distributed - great!

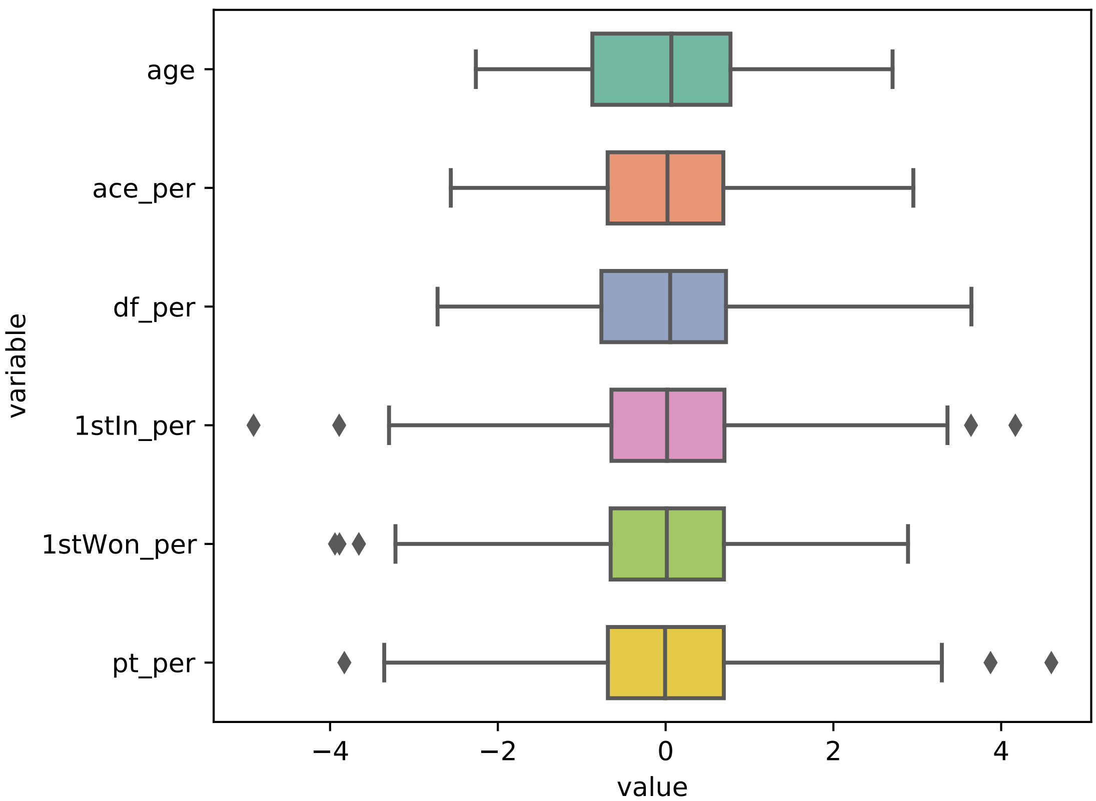
3.5 Outliers

Next, I am checking that there aren't any substantial outliers that might affect linear relationships. To do so, I'm first going to standardize my variables to put them all on an equivalent scale for plotting:

[18]:
```
# standardize each of my variables to have equivalent scales:
tennis_df_z = tennis_df_long.copy()
tennis_df_z.loc[:,tennis_df_z.dtypes == "float"] = stats.zscore(tennis_df_z.
 ↪loc[:,tennis_df_z.dtypes == "float"], axis=0)
```

To be a bit more liberal with exclusion criteria (so marking fewer data points as outliers), I'm extending the whiskers for the below box plots to 2 * IQR (typically 1.5):

[19]:
```
plt.figure(figsize=(6,5))
sns.boxplot(data=tennis_df_z.select_dtypes("float").melt(),
            y = 'variable', x = 'value', palette = "Set2",
            width=.6, whis=2)
plt.show()
```



Remove outliers from the data:

[20]:
```
# data = a pandas series, marks outliers as NaN
def mark_outliers(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    my_filter = (data < Q1 - 2 * IQR) | (data > Q3 + 2 * IQR)
    return my_filter
```

[21]:
```
my_vars = tennis_df_long.select_dtypes("float").columns.tolist()
for v in my_vars:
    filt = mark_outliers(tennis_df_long[v])
    tennis_df_long.loc[filt,v] = np.NaN
```

[22]:
```
# drop missing values (outliers):
tennis_df_long.dropna(inplace=True)

print('Cleaned data contains statistics for',tennis_df_long.
 ↪shape[0],'players\n')
```

```
tennis_df_long.head()
```

```
Cleaned data contains statistics for 4556 players
```

[22]:

```
   match_id outcome               name        age   ace_per    df_per  \
0         0       w       Kei Nishikori  29.004791  1.536686  1.484146
1         1       w     Daniil Medvedev  22.885695  3.879314  0.681774
2         2       w       Kei Nishikori  29.004791  1.649625  1.589506
3         3       w  Jo-Wilfried Tsonga  33.705681  3.735738  1.155998
4         4       w     Daniil Medvedev  22.885695  3.042286  1.122826

   1stIn_per  1stWon_per     pt_per
0  57.142857   70.454545  53.107345
1  63.461538   84.848485  58.914729
2  70.212766   78.787879  64.516129
3  63.235294   79.069767  55.704698
4  64.761905   70.588235  51.256281
```

4 Key Findings and Insights

In this section, I am using the cleaned data to provide some preliminary insight into how my player variables relate to the likelihood of winning a match as well as the total percentage of points won in a game.

4.1 Data for winners vs. losers

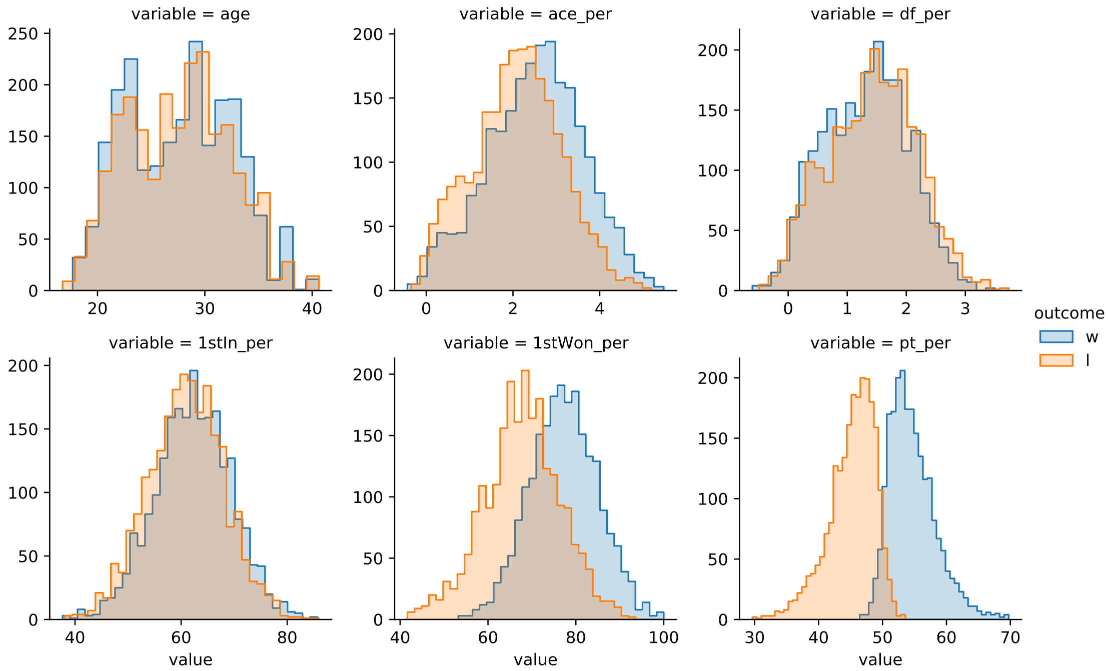
Here are the same histograms as I plotted above, but now divided by whether the player won or lost their match:

[23]:

```
melted_df = tennis_df_long.drop(['match_id','name'], axis=1).
 ↪melt(id_vars="outcome")

grid = sns.FacetGrid(melted_df, col='variable', col_wrap=3,
                     hue='outcome',
                     height=3, aspect=1,
                     sharex = False, sharey = False)
grid.map(sns.histplot, "value", alpha=.25,
         element="step")
grid.add_legend()
```

[23]: <seaborn.axisgrid.FacetGrid at 0x120e889e8>



Unsurprisingly, players who win the match tend to win a higher percentage of total points ("pt_per"). Visually, what appears to be most strongly associated with the probability of winning is the percentage of aces made in a match, as well as the percentage of 1st service points won. Double faults and percentage of 1st serves in appear to make a small contribution, and there appears to be minimal or no impact of age.

Here is a summary of descriptive statistics, grouped by winners and losers:

[24]:
```
tennis_df_long.drop('match_id', axis=1).groupby('outcome').
↪agg(['mean','std','min','max']).transpose()
```

[24]:

outcome		l	w
age	mean	27.551390	27.672662
	std	4.710009	4.912380
	min	16.739220	17.681040
	max	40.624230	40.624230
ace_per	mean	2.098201	2.528593
	std	1.011213	1.086299
	min	-0.346641	-0.433622
	max	5.189438	5.476191
df_per	mean	1.424274	1.319377
	std	0.741199	0.708457
	min	-0.490891	-0.606692
	max	3.734195	3.504947
1stIn_per	mean	60.900438	62.281636
	std	7.130305	7.198408

```
            min   38.028169   37.647059
            max   85.964912   85.714286
1stWon_per  mean  67.741135   77.268828
            std    8.667596    7.701484
            min   41.666667   53.246753
            max   93.617021  100.000000
pt_per      mean  45.207064   54.683643
            std    3.666534    3.628172
            min   29.629630   46.428571
            max   53.571429   69.736842
```

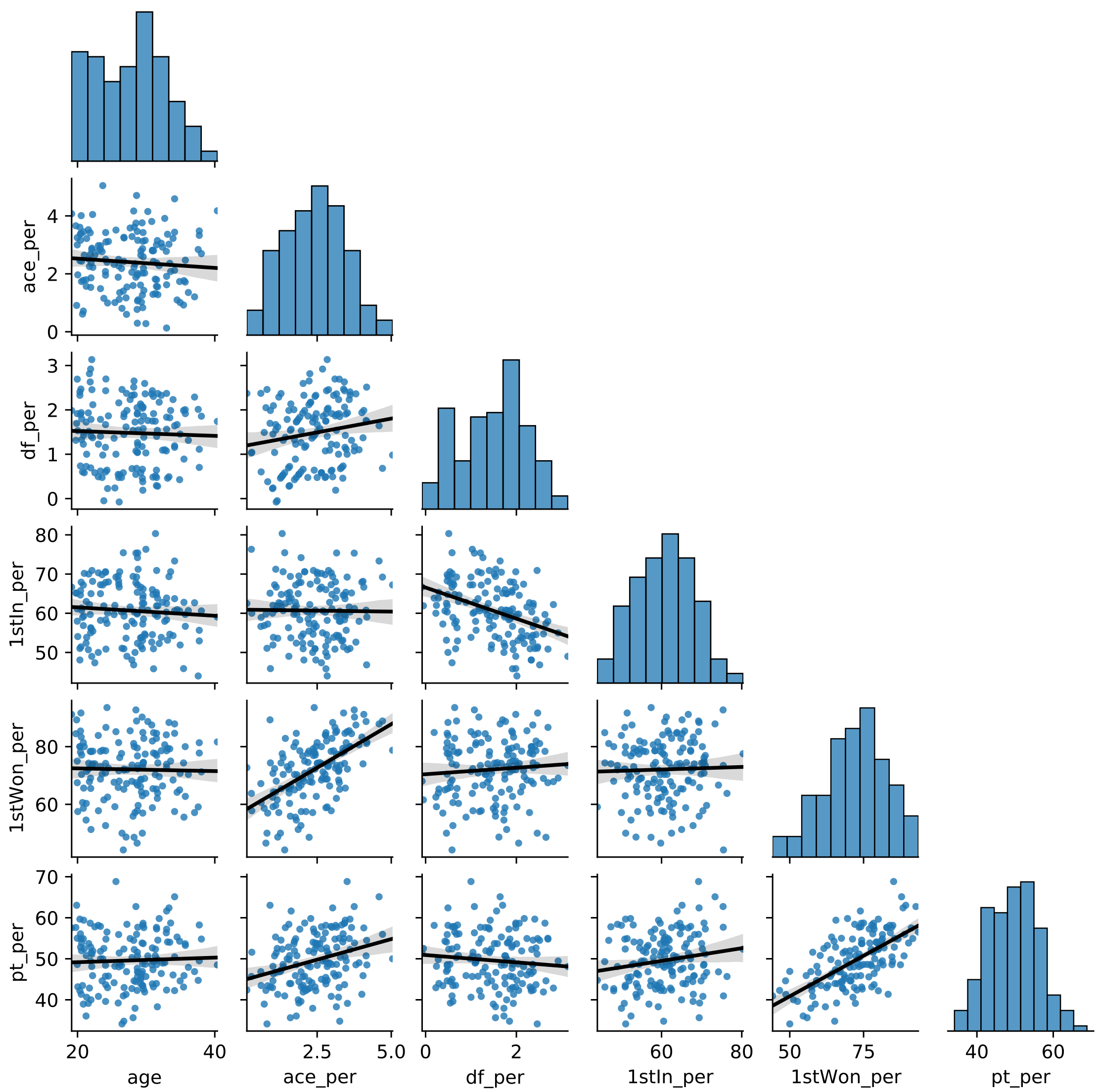
4.2 Correlations

Due to the amount of data, I'm randomly selecting a sub-sample to plot the correlations between my variables:

[25]:
```
print('Randomly sampling data of 150 players\n')
data_sample = tennis_df_long.sample(150, replace=False)

sns.pairplot(data_sample.select_dtypes("float"),
             kind="reg", height=1.4, corner=True,
             plot_kws={'scatter_kws':{'s':8},
                       'line_kws':{'color':'black',
                                   'lw':2}})
plt.show()
```

```
Randomly sampling data of 150 players
```



A couple of things to take away from the above plot:

- none of our feature variables (all apart from pt_per — percentage of game points won) show a particiularly strong correlation with one another, so no real issues with multicollinearity.
- looking at the bottom row, we confirm the observations above comparing winners and losers — a higher percentage of game points won (pt_per) is positively associated with percentage of 1st serves won, as well as slightly correlated with aces and percentage of 1st serves in, and is slightly negatively associated with the double faults.

5 Potential Hypotheses

1. Match winners make a higher percentage of 1st serves than match losers.
2. Match winners are younger than match losers.

3. A higher number of match points won is associated with a higher percentage of 1st serves in.

6 Significance Test: Hypothesis 1

To test the hypothesis that match winners make a higher percentage of 1st serves than match losers, I'm running an independent samples t-test with scipy:

```
[26]: grp1 = tennis_df_long.loc[tennis_df_long["outcome"] == "w","1stIn_per"]
      grp2 = tennis_df_long.loc[tennis_df_long["outcome"] == "l","1stIn_per"]
      stats.ttest_ind(grp1, grp2)
```

```
[26]: Ttest_indResult(statistic=6.506143450550752, pvalue=8.541338317664799e-11)
```

A p value of $< .001$ shows that winners make a significantly higher % of 1st serves (at alpha $= .05$) than losers.

7 Next Steps

The relationship between percentage of match points won and my continuous features — 1st serves, aces, double faults, etc. — could also be explored separately for winners and losers or based on age categories. For example, perhaps older players benefit (in terms of winning likelihood) from a higher number of aces than younger players? Additionally, you could use classification, such as logistic regression, to predict whether a player will win or lose based on their match statistics. In this case, you could train on half of the data and test on the other half of the data to test the generalisability of the model. Finally, I would be interested in looking at the differences betwen opponents (by match) rather than each player's individual performance — for example, perhaps age becomes important for winning likelihood only when there is a large difference in age between opponents.

8 Dataset Quality Summary

Overall, this dataset appears to be good quality. The only variable that had a significant amount of missing data was players' height, which was therefore excluded. Most of the variables are normally diistributed, and relatively few outliers appeared in the data, and there do not appear to be redundancies between the variables (none show a very high correlation). Some additional stats would be useful, including forehand and backhand winners, rally durations, etc., in order to build a comprehensive model of winning likelihood.