

Анализ формирования ставок на теннисные матчи на букмекерском рынке

Анисимов Максим Андреевич, БЭК 151

Данная работа посвящена анализу ставок на теннисные матчи и статистики теннисных матчей в целом. Идея провести такое небольшое исследование пришла ко мне после долгих лет игры в теннис и небольшого увлечения ставками. Вопросы, которые я задал себе, состоят в следующем:

1. На что люди ориентируются, когда делают ставки на того или иного игрока?
2. Как выглядит статистика теннисных матчей в динамике, в зависимости от типа покрытия корта и прочее?
3. Может ли машинное обучение превзойти точность предсказаний букмекерских ставок?

Сразу стоит оговориться, что в своей работе я не пытаюсь проверить, можно ли обыграть букмекерские конторы, ведь они, подстраивая свои ставки под спрос и предложение, всегда находятся в выигрыше. Вернее сказать, что интерес представляет сравнение ожиданий от матча компьютера, настроенного мной, и людей, играющих на ставках и тем самым их формирующих. Но помимо машинного обучения я считаю необходимым посмотреть на визуализированные данные о теннисных матчах: из них определённо можно будет сделать выводы для оптимизации стратегии ставок. В данной работе будут проделаны следующие основные действия:

1. Обработка больших массивов данных
2. Визуализация данных
3. Сравнение доли качества предсказаний букмекерского рынка и предсказаний, полученных с помощью машинного обучения

Заранее стоит сказать, что в данной работе будут анализироваться матчи только мужского профессионального тенниса (ATP) за последние 16 лет. Это связано с тем, что:

- данные по мужским матчам более доступные
- женский теннис априори считается более непредсказуемым
- после 1990-х годов теннис претерпел определённые изменения в плане стиля игры

Теперь приступим к практике!

Для начала загрузим все необходимые для анализа данных библиотеки.

In [1] :

```
import scipy.stats as stats
import csv as csv
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.stats import kendalltau
import fnmatch
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
%matplotlib inline
import warnings
warnings.simplefilter('ignore')
%pylab inline
```

Populating the interactive namespace from numpy and matplotlib

Далее приступим к работе с данными.

Работа со статистическими данными о теннисных матчах

Откроем файл с данными о матчах за 2000-2015 годы и за 2016 год.

*Данные взяты с https://github.com/JeffSackmann/tennis_atp.

In [2]:

```
matches = pd.read_csv('all_matches_2000-2015.csv')
matches.head()
```

Out [2]:

	tourney_id	tourney_name	surface	draw_size	tourney_level	tourney_date	match_num	w
0	2000-717	Orlando	Clay	32	A	20000501	1	10
1	2000-717	Orlando	Clay	32	A	20000501	2	10
2	2000-717	Orlando	Clay	32	A	20000501	3	10
3	2000-717	Orlando	Clay	32	A	20000501	4	10
4	2000-717	Orlando	Clay	32	A	20000501	5	10

5 rows × 49 columns



In [3]:

```
tennis2016 = pd.read_csv('atp_matches_2016.csv')
```

Теперь объединим их и сделаем нумерацию от 0 до числа строк в новом датафрейме.

In [4]:

```
matches = pd.concat([matches, tennis2016])
matches.index = range(len(matches.index))
matches.tail(2)
```

Out [4]:

	tourney_id	tourney_name	surface	draw_size	tourney_level	tourney_date	match_num
53118	2016-M-DC-2016-WG-PO- SUI vs SUI	Davis Cup WG PO: SUI vs SUI	Clay	4	D	20160916	4

	SUI-UZB-tourney_id	UZB-tourney_name	surface	draw_size	tourney_level	tourney_date	match_num
53119	2016-M-DC-2016-WG-PO-SUI-UZB-01	Davis Cup WG PO: SUI vs UZB	Clay	4	D	20160916	5

2 rows × 49 columns



In [5] :

```
matches[['tourney_date', 'winner_name', 'loser_name', 'score']].head()
```

Out [5] :

	tourney_date	winner_name	loser_name	score
0	20000501	Antony Dupuis	Andrew Ilie	3-6 7-6(6) 7-6(4)
1	20000501	Fernando Gonzalez	Cecil Mamiit	6-2 7-5
2	20000501	Paradorn Srichaphan	Sebastien Lareau	6-1 6-3
3	20000501	Jan Siemerink	Justin Gimelstob	4-6 6-2 7-5
4	20000501	Jason Stoltenberg	Alex Lopez Moron	6-1 6-4

Посмотрим основную информацию о новом датафрейме 'matches':

In [6] :

```
matches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53120 entries, 0 to 53119
Data columns (total 49 columns):
tourney_id           53120 non-null object
tourney_name          53120 non-null object
surface              53002 non-null object
draw_size             53120 non-null int64
tourney_level         53120 non-null object
tourney_date          53120 non-null int64
match_num             53120 non-null int64
winner_id              53120 non-null int64
winner_seed            22156 non-null float64
winner_entry           6192 non-null object
winner_name            53120 non-null object
winner_hand             53108 non-null object
winner_ht              49751 non-null float64
winner_ioc              53120 non-null object
winner_age              53102 non-null float64
winner_rank             52057 non-null float64
winner_rank_points      52057 non-null float64
loser_id                53120 non-null int64
loser_seed              11899 non-null float64
loser_entry             10390 non-null object
loser_name              53120 non-null object
loser_hand              53093 non-null object
```

```
-----  
loser_ht          47607 non-null float64  
loser_ioc         53120 non-null object  
loser_age         53092 non-null float64  
loser_rank        51350 non-null float64  
loser_rank_points 51350 non-null float64  
score              53119 non-null object  
best_of            53120 non-null int64  
round               53120 non-null object  
minutes             46132 non-null float64  
w_ace              47366 non-null float64  
w_df                47366 non-null float64  
w_svpt              47366 non-null float64  
w_1stIn             47366 non-null float64  
w_1stWon             47366 non-null float64  
w_2ndWon             47366 non-null float64  
w_SvGms              47366 non-null float64  
w_bpSaved             47366 non-null float64  
w_bpFaced             47366 non-null float64  
l_ace              47366 non-null float64  
l_df                47366 non-null float64  
l_svpt              47366 non-null float64  
l_1stIn             47366 non-null float64  
l_1stWon             47366 non-null float64  
l_2ndWon             47366 non-null float64  
l_SvGms              47366 non-null float64  
l_bpSaved             47366 non-null float64  
l_bpFaced             47366 non-null float64  
dtypes: float64(29), int64(6), object(14)  
memory usage: 19.9+ MB
```

In [7]:

```
matches[['tourney_name', 'best_of']].head() #этот признак показывает, какое  
максимально число сетов могут сыграть игроки
```

Out[7]:

	tourney_name	best_of
0	Orlando	3
1	Orlando	3
2	Orlando	3
3	Orlando	3
4	Orlando	3

In [8]:

```
matches[matches['tourney_name'] == 'Roland Garros']['best_of'].head()
```

Out[8]:

```
2775      5  
2776      5  
2777      5  
2778      5  
2779      5  
Name: best_of, dtype: int64
```

In [9]:

```
matches[['winner_name', 'winner_ioc']].head() # а этот показывает, какую страну представляет игрок
```

Out [9]:

	winner_name	winner_ioc
0	Antony Dupuis	FRA
1	Fernando Gonzalez	CHI
2	Paradorn Srichaphan	THA
3	Jan Siemerink	NED
4	Jason Stoltenberg	AUS

Сделаем новые, более репрезентативные и информативные статистические показатели игроков за матч:

In [10]:

```
for pl, op in zip(['w', 'l'], ['l', 'w']):
    # Число выигранных очков на подаче:
    matches[pl + '_svptWon'] = matches[pl + '_1stWon'] + matches[pl + '_2ndWon']
    # Процент выигранных очков на подаче:
    matches[pl + '_PrctsvptWon'] = matches[pl + '_svptWon'] / matches[pl + '_svpt']
    # Число попаданий 2 подачей:
    matches[pl + '_2ndIn'] = matches[pl + '_svpt'] - matches[pl + '_1stIn'] - matches[pl + '_df']
    # Число совершённых подач:
    matches[pl + '_sv_number'] = matches[pl + '_1stIn'] + 2 * matches[pl + '_2ndIn'] + 2 * matches[pl + '_df']
    # Точность подачи:
    matches[pl + '_ServeAccuracy'] = (matches[pl + '_1stIn'] + matches[pl + '_2ndIn']) / matches[pl + '_sv_number']
    # Точность 1 подачи:
    matches[pl + '_Prct1stIn'] = matches[pl + '_1stIn'] / matches[pl + '_svpt']
    # Точность 2 подачи:
    matches[pl + '_Prct2ndIn'] = matches[pl + '_2ndIn'] / (matches[pl + '_svpt'] - matches[pl + '_1stIn'])
    # Процент выигранных розыгрышей на 1 подаче:
    matches[pl + '_Prct1stWon'] = matches[pl + '_1stWon'] / matches[pl + '_1stIn']
    # Процент выигранных розыгрышей на 2 подаче:
    matches[pl + '_Prct2ndWon'] = matches[pl + '_2ndWon'] / matches[pl + '_2ndIn']
    # Процент эйсов от числа очков, на которых игрок подавал:
    matches[pl + '_Prct_ace'] = matches[pl + '_ace'] / matches[pl + '_svpt']
    # Процент двойных ошибок от числа очков, на которых игрок подавал:
    matches[pl + '_Prct_df'] = matches[pl + '_df'] / matches[pl + '_svpt']
    # Число выигранных брейкоинтов:
    matches[pl + '_bpWon'] = matches[op + '_bpFaced'] - matches[op + '_bpSaved']
    # Число разыгранных мячей (когда подающий не совершил двойную ошибку пр
```

```

и подаче) на приёме 1 подачи:
matches[pl + '_1stRetpt'] = matches[op + '_1stIn']
# Число выигранных розыгрышей на приёме 1 подачи:
matches[pl + '_1stRetptWon'] = matches[op + '_1stIn'] - matches[op + '_1stInLost']

# Процент выигранных розыгрышей на приёме 1 подачи от общего числа розыгрышней на приёме 1 подачи:
matches[pl + '_Prct1stRetptWon'] = matches[pl + '_1stRetptWon'] / matches[op + '_1stIn']

for pl, op in zip(['w', 'l'], ['l', 'w']):
    # Число разыгранных розыгрышей на приёме 2 подачи:
    matches[pl + '_2ndRetpt'] = matches[op + '_2ndIn']
    # Число выигранных розыгрышей на приёме 2 подачи:
    matches[pl + '_2ndRetptWon'] = matches[op + '_2ndIn'] - matches[op + '_2ndInLost']

    # Процент выигранных розыгрышей на приёме 2 подачи от общего числа розыгрышней на приёме 2 подачи:
    matches[pl + '_Prct2ndRetptWon'] = matches[pl + '_2ndRetptWon'] / matches[op + '_2ndIn']

    # Число выигранных очков за матч:
    matches[pl + '_ptWon'] = matches[pl + '_1stWon'] + matches[pl + '_2ndWon'] + matches[pl + '_1stRetptWon'] + matches[pl + '_2ndRetptWon']

    # Процент выигранный очков за матч (может быть меньше 0.5 для победителя в силу особенностей правил игры в теннис):
    matches[pl + '_PrctptWon'] = matches[pl + '_ptWon'] / (matches[pl + '_svpt'] + matches[op + '_svpt'])

for pl, op in zip(['w', 'l'], ['l', 'w']):
    # Число выигранных очков на приёме
    matches[pl + '_RetptWon'] = matches[pl + '_1stRetptWon'] + matches[pl + '_2ndRetptWon']

    # Доля выигранных очков на приёме от числа розыгрышней на приёме
    matches[pl + '_PrctRetptWon'] = matches[pl + '_RetptWon'] / matches[op + '_svpt']

```

В данном датафрейме в колонках 'winner_name' и 'loser_name' можно увидеть полные имя и фамилию выигравшего и проигравшего игроков соответственно. Создадим новые колонки 'Winner' и 'Loser' с фамилией, первой буквой имени игрока и точкой после неё. В дальнейшем мы увидим, что это нам очень пригодится.

In [11]:

```

matches['Winner'] = matches['winner_name'].apply(lambda s: s.split()[-1] + ' + s[0] + '.')

matches['Loser'] = matches['loser_name'].apply(lambda s: s.split()[-1] + ' + s[0] + '.')

```

Посмотрим, как выглядят даты турниров в данном датафрейме:

In [12]:

```
matches['tourney_date'].head(5)
```

Out [12]:

```
0      20000501
```

```
1 20000501
2 20000501
3 20000501
4 20000501
Name: tourney_date, dtype: int64
```

Очень странный, неудобный и нечитаемый формат. Переделаем его в классический формат даты для работы в pandas 'datetime'. Это также поможет нам для анализа данных в дальнейшем. На всякий случай выделим день, месяц и год тоже.

Комментарий: важно понимать, что здесь указаны даты НАЧАЛА ТУРНИРОВ, а не самих матчей.

In [13]:

```
matches['Tournament Date'] = pd.to_datetime(matches['tourney_date'], format='%Y%m%d')

matches['Tournament Year'] = np.int_(matches['Tournament Date'].dt.year)
matches['Tournament Month'] = np.int_(matches['Tournament Date'].dt.month)
matches['Tournament Day'] = np.int_(matches['Tournament Date'].dt.day)

matches[['Tournament Date', 'Tournament Year', 'Tournament Month', 'Tournament Day']].head()
```

Out [13]:

	Tournament Date	Tournament Year	Tournament Month	Tournament Day
0	2000-05-01	2000	5	1
1	2000-05-01	2000	5	1
2	2000-05-01	2000	5	1
3	2000-05-01	2000	5	1
4	2000-05-01	2000	5	1

Ради интереса посмотрим, какие вообще раунды турниров представлены в нашем датафрейме:

In [14]:

```
matches['round'].value_counts()
```

Out [14]:

```
R32      16867
R16      8868
R64      8208
RR       5990
R128     5312
QF       4451
SF       2266
F        1153
BR       5
Name: round, dtype: int64
```

После проделанной работы датафрейм стал более читаемым и информативным.

Графики: matches

Визуализируем данные, которые есть в датафрейме matches

Чтобы не мешать переменные, удобные лишь для построения графиков, с переменными, которые будут далее использоваться при обучении, создадим новый датафрейм, и с помощью него построим некоторые диаграммы.

In [15] :

```
matches_gr = matches
```

In [16] :

```
# Создание статистических показателей в общем за матч

# Число эйсов за матч:
matches_gr['match_ace'] = matches_gr['w_ace'] + matches_gr['l_ace']
# Процент эйсов от общего числа разыгранных очков за матч:
matches_gr['match_Prct_ace'] = matches_gr['match_ace'] / (matches_gr['w_svpt'] + matches_gr['l_svpt'])
# Число двойных ошибок за матч:
matches_gr['match_df'] = matches_gr['w_df'] + matches_gr['l_df']
# Процент двойных ошибок от общего числа разыгранных очков за матч:
matches_gr['match_Prct_df'] = matches_gr['match_df'] / (matches_gr['w_svpt'] + matches_gr['l_svpt'])

for feature in ['ace', 'df', 'svpt', '1stIn', '1stWon', '2ndWon', 'svptWon',
'PrctsvptWon', 'SvGms', 'bpSaved', 'bpFaced',
'ServeAccuracy', 'sv_number', 'Prct1stIn', 'Prct1stWon', 'Prct_ace',
'Prct_df', 'bpWon', '1stRetpt',
'1stRetptWon', 'Prct1stRetptWon', 'RetptWon', 'PrctRetptWon']:
    # Средние показатели за матч:
    matches_gr['match_avg_' + feature] = (matches_gr['w_' + feature] +
matches_gr['l_' + feature]) / 2

for feature in ['age', 'rank_points', 'seed', 'ht']:
    # Средние показатели, связанные с предматчевой информацией:
    matches_gr['match_avg_' + feature] = (matches_gr['winner_' + feature] +
matches_gr['loser_' + feature]) / 2
```

Теперь перейдём к построению графиков.

Хотелось бы начать с проверки гипотезы о том, что чем больше средний рост игроков матча, тем больше они подают эйсов. Проверим её, построив гистограмму в осях "средний рост игроков матча - среднее число эйсов за матч для данного среднего роста игроков", и посмотрим, есть ли какая-либо тенденция.

In [17] :

```
plt.figure(figsize=(15, 5))
```

```
matches_gr.groupby('match_avg_ht')[ 'match_ace' ].mean().plot(kind='line')
plt.xlabel('Средний рост игроков матча')
plt.title('Число эйсов за матч', fontsize=15)
```

Out [17] :

```
<matplotlib.text.Text at 0x13b4d804be0>
```



Как можно видеть из тенденции на графике выше, чем больше средний рост игроков матча, тем в среднем они больше подают эйсов за матч. Но что, если высокие игроки просто в среднем дольше играют матчей, и поэтому подают больше эйсов? Проверим, какова тенденция, если брать средний процентов эйсов от общего числа разыгранных очков.

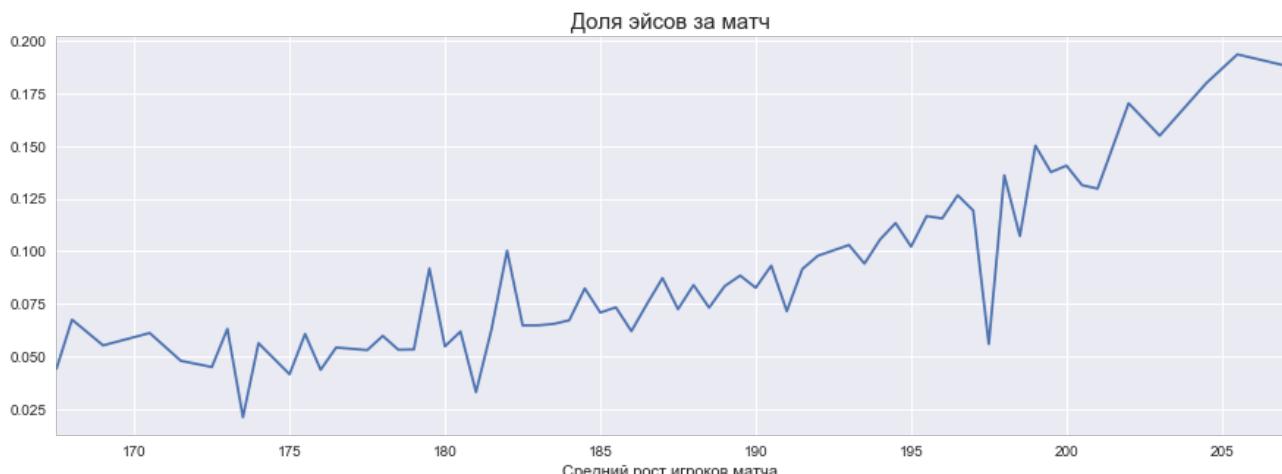
In [18] :

```
plt.figure(figsize=(15, 5))

matches_gr.groupby('match_avg_ht')[ 'match_Prct_ace' ].mean().plot(kind='line')
plt.xlabel('Средний рост игроков матча')
plt.title('Доля эйсов за матч', fontsize=15)
```

Out [18] :

```
<matplotlib.text.Text at 0x13b4d3135f8>
```



Тенденция сохраняется, и поэтому можно утверждать, что при одинаковом числе разыгранных мячей более высокие игроки в среднем подают больше эйсов за матч, чем более низкие.

интересно было бы также проверить, зависит ли число двойных ошибок от роста игроков. Их гистограммы ниже можно сказать, что очевидной связи нет.

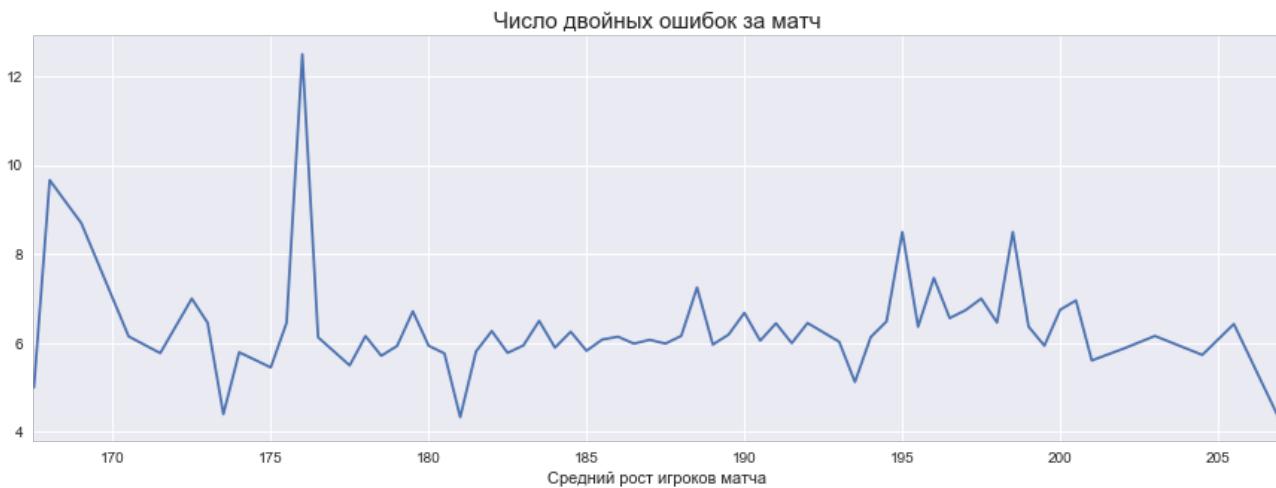
In [19] :

```
plt.figure(figsize=(15, 5))

matches_gr.groupby('match_avg_ht')[['match_df']].mean().plot(kind='line')
plt.xlabel('Средний рост игроков матча')
plt.title('Число двойных ошибок за матч', fontsize=15)
```

Out [19] :

<matplotlib.text.Text at 0x13b4d7674e0>



Визуализация данных о победителях и проигравших

Теперь посмотрим, какое распределение имеют различные признаки, имеющиеся в датафрейме, причём сравним распределения признаков победителя и проигравшего. Возможно, такое сравнение позволит понять, какие параметры являются ключевыми для победы в матче.

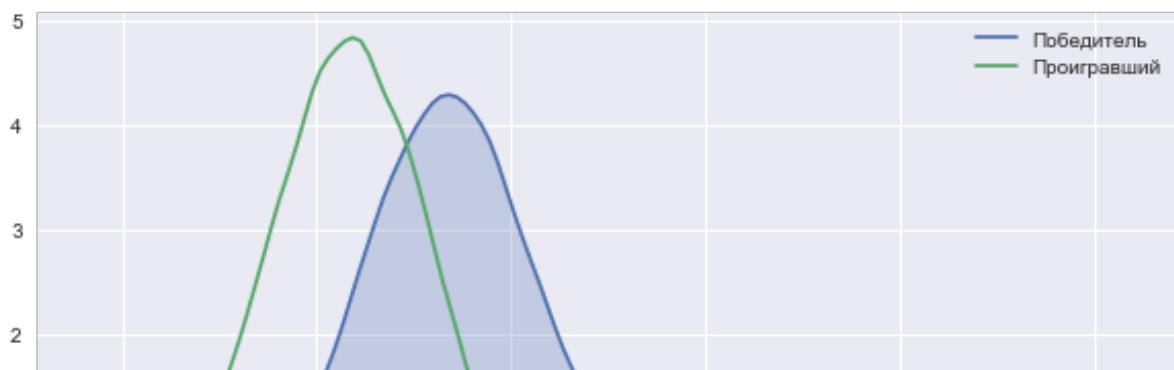
In [20] :

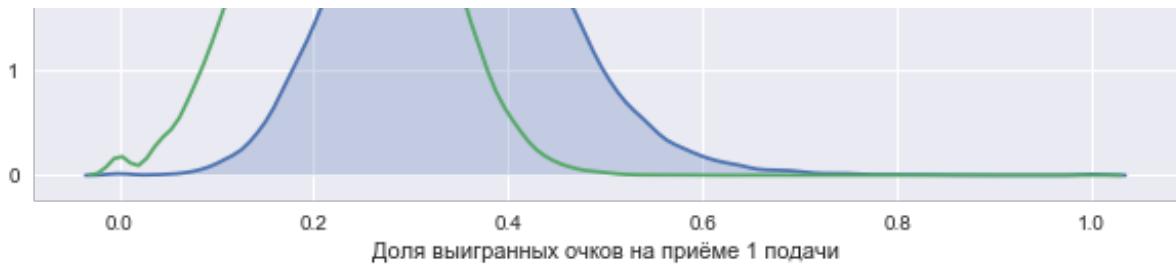
```
plt.figure(figsize=(10,5))

sns.kdeplot(matches['w_Prct1stRetptWon'], shade=True)
sns.kdeplot(matches['l_Prct1stRetptWon'])
plt.xlabel('Доля выигранных очков на приёме 1 подачи')
plt.legend(['Победитель', 'Проигравший'])
```

Out [20] :

<matplotlib.legend.Legend at 0x13b4dc9c4a8>





In [21]:

```

plt.figure(figsize=(25,20))

plt.subplot(3, 3, 1)
sns.kdeplot(matches['w_ServeAccuracy'], shade=True)
sns.kdeplot(matches['l_ServeAccuracy'])
plt.legend(['Победитель', 'Проигравший'])
plt.xlabel('Точность подачи')

plt.subplot(3, 3, 2)
sns.kdeplot(matches['w_Prct_df'], shade=True)
sns.kdeplot(matches['l_Prct_df'])
plt.xlabel('Процент двойных ошибок от очков на подаче')
plt.legend(['Победитель', 'Проигравший'])

plt.subplot(3, 3, 3)
sns.kdeplot(matches['w_Prct_ace'], shade=True)
sns.kdeplot(matches['l_Prct_ace'])
plt.xlabel('Процент эйсов от числа очков на подаче')
plt.legend(['Победитель', 'Проигравший'])

plt.subplot(3, 3, 4)
sns.kdeplot(matches['w_Prct2ndIn'], shade=True)
sns.kdeplot(matches['l_Prct2ndIn'])
plt.xlabel('Точность 2 подачи')
plt.legend(['Победитель', 'Проигравший'])

plt.subplot(3, 3, 5)
sns.kdeplot(matches['w_Prct1stWon'], shade=True)
sns.kdeplot(matches['l_Prct1stWon'])
plt.xlabel('Доля выигранных очков на 1 подаче')
plt.legend(['Победитель', 'Проигравший'])

plt.subplot(3, 3, 6)
sns.kdeplot(matches['w_Prct1stRetptWon'], shade=True)
sns.kdeplot(matches['l_Prct1stRetptWon'])
plt.xlabel('Доля выигранных очков на приёме 1 подачи')
plt.legend(['Победитель', 'Проигравший'])

plt.subplot(3, 3, 7)
sns.kdeplot(matches['winner_age'], shade=True)
sns.kdeplot(matches['loser_age'])
plt.xlabel('Возраст')
plt.legend(['Победитель', 'Проигравший'])

```

```

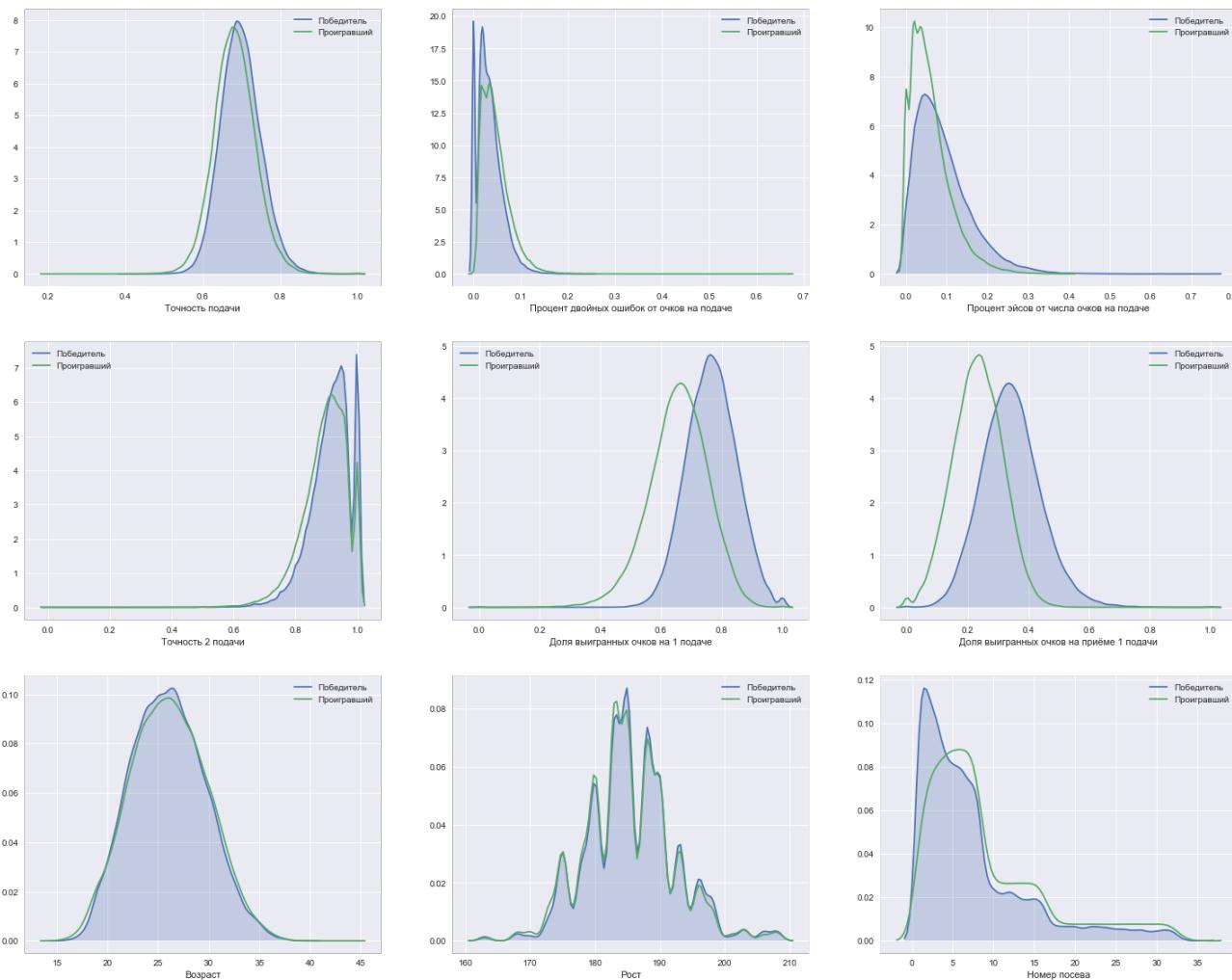
plt.subplot(3, 3, 8)
sns.kdeplot(matches['winner_ht'], shade=True)
sns.kdeplot(matches['loser_ht'])
plt.xlabel('Рост')
plt.legend(['Победитель', 'Проигравший'])

plt.subplot(3, 3, 9)
sns.kdeplot(matches['winner_seed'], shade=True)
sns.kdeplot(matches['loser_seed'])
plt.legend(['Победитель', 'Проигравший'])
plt.xlabel('Номер посева') # ранг игрока в конкретном турнире, исходя из мирового рейтинга теннисистов

```

Out [21] :

<matplotlib.text.Text at 0x13b5072f9b0>



Видим, что, например, распределения возраста очень близки, и, наверное, нельзя сказать, что с помощью знания о возрасте игроков можно с хорошей точностью предсказывать победу игроков. Хочется проверить гипотезу о равенстве математических ожиданий этих возрастов, но для этого сначала надо предположить, что эти случайные величины имеют какое-то распределение. Проверим их на нормальность!

In [22] :

```

stats.mstats.normaltest(matches['winner_age'].dropna())
# тест на нормальность

```

```
Out[22]:
```

```
NormaltestResult(statistic=707.45802689570121, pvalue=2.384741835081246e-14)
```

```
In [23]:
```

```
stats.mstats.normaltest(matches['loser_age'].dropna())
```

```
Out[23]:
```

```
NormaltestResult(statistic=512.51144682505628, pvalue=5.1233422168043402e-12)
```

К сожалению, для обоих векторов `pvalue` получились ничтожно малыми, и для любого разумного уровня доверия принять гипотезу о нормальность мы не можем.

Тогда сравним средний возраст игроков.

```
In [24]:
```

```
print('Средний возраст победителя: ', matches['winner_age'].mean(), '\nСредний возраст проигравшего: ',  
      matches['loser_age'].mean())
```

```
Средний возраст победителя: 26.01607666270934
```

```
Средний возраст проигравшего: 26.097519406732268
```

Как мы видим, среднее по выборке двух векторов не имеют существенных отличий, поэтому мы не будем считать на данном этапе возраст хорошим признаком для определения победителя и проигравшего.

Сравним среднее число эйсов и получим значимый результат: в среднем в матче победитель имеет в 1,5 раза больший процент эйсов, чем проигравший!

```
In [25]:
```

```
w_mean_prct_ace = mean(matches['w_Prct_ace'])  
l_mean_prct_ace = mean(matches['l_Prct_ace'])  
  
print('Средняя доля эйсов победителя = ', w_mean_prct_ace,  
      '\nСредняя доля эйсов проигравшего = ', l_mean_prct_ace)
```

```
Средняя доля эйсов победителя = 0.09079957973199616
```

```
Средняя доля эйсов проигравшего = 0.05989152674646107
```

Сравним также долю двойных ошибок и точной второй подачи:

```
In [26]:
```

```
w_mean_prct_df = mean(matches['w_Prct_df'])  
l_mean_prct_df = mean(matches['l_Prct_df'])  
  
print('Средняя доля двойных ошибок победителя = ', w_mean_prct_df,  
      '\nСредняя доля двойных ошибок проигравшего = ', l_mean_prct_df)
```

```
Средняя доля двойных ошибок победителя = 0.03334570315964894
```

```
Средняя доля двойных ошибок проигравшего = 0.04321600851874923
```

In [27]:

```
w_mean_prct_2ndIn = mean(matches['w_Prct2ndIn'])
l_mean_prct_2ndIn = mean(matches['l_Prct2ndIn'])

print('Средняя точность второй подачи победителя = ', w_mean_prct_2ndIn,
      '\nСредняя точность второй подачи проигравшего = ',
      l_mean_prct_2ndIn)
```

Средняя точность второй подачи победителя = 0.9146270357517253
Средняя точность второй подачи проигравшего = 0.8949424775818305

И здесь уже мы не видим значительной разницы между показателями победителя и проигравшего.

А теперь посмотрим, какие игроки за 2000-2016 гг. выиграли больше всех матчей:

In [28]:

```
best_winners = matches['winner_name'].value_counts()[:10]
print(best_winners)
```

Roger Federer	1075
Rafael Nadal	814
Novak Djokovic	761
David Ferrer	698
Andy Murray	638
Andy Roddick	614
Tomas Berdych	582
Lleyton Hewitt	564
Tommy Robredo	525
Nikolay Davydenko	484

Name: winner_name, dtype: int64

Конечно, данный список можно оформить в более красивом виде:

In [29]:

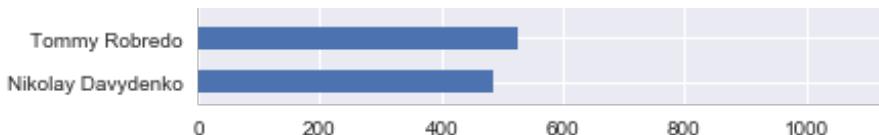
```
bw = pd.DataFrame(data=best_winners[::-1])
bw.plot.barch()

plt.legend('')
plt.title('Игроки: число выигранных матчей (2000-2016)')
```

Out [29]:

<matplotlib.text.Text at 0x13b506da8d0>





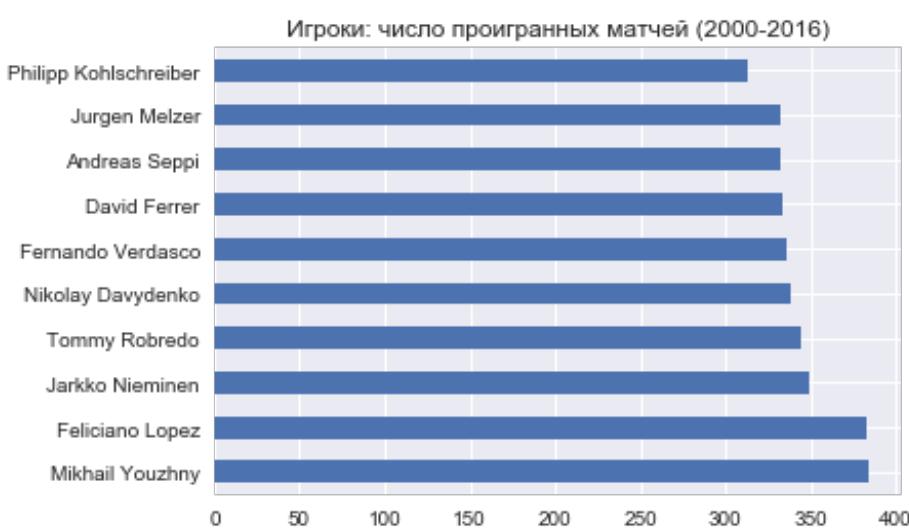
Теперь посмотрим на проигравших:

In [30] :

```
best_losers = matches['loser_name'].value_counts()[:10]
bl = pd.DataFrame(data=best_losers)
bl.plot.bart()
plt.legend('')
plt.title('Игроки: число проигранных матчей (2000-2016)')
```

Out [30] :

<matplotlib.text.Text at 0x13b50eb2978>



Далее посмотрим, представители каких стран чаще всех выигрывают и проигрывают матчи.

Вот список стран с наибольшим числом побед за последние \$16\$ лет. Россия шестая - не так уже плохо!

In [31] :

```
best_countries = matches['winner_ioc'].value_counts()
best_countries = best_countries[:10]

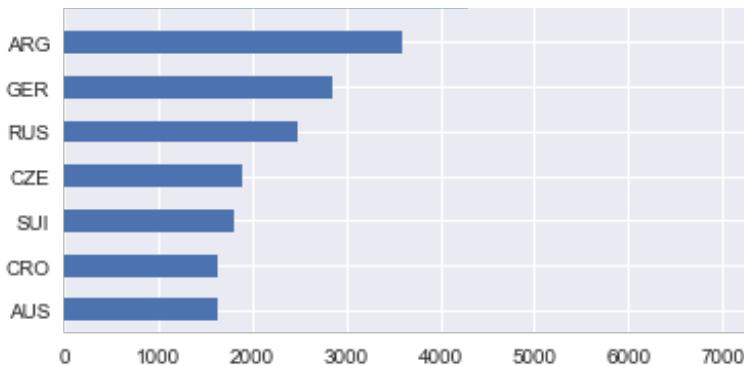
bc = pd.DataFrame(data=best_countries[::-1])
bc.plot.bart()

plt.legend('')
plt.title('Представители стран: число выигранных матчей (2000-2016)')
```

Out [31] :

<matplotlib.text.Text at 0x13b50f4d6d8>





Но и по числу поражений Россия - одна из лидеров...

In [32] :

```
worst_countries = matches['loser_ioc'].value_counts()
worst_countries = worst_countries[:10]

wc = pd.DataFrame(data=worst_countries)
wc.plot.barh()

plt.legend('')
plt.title('Представители стран: число проигранных матчей')
```

Out [32] :

<matplotlib.text.Text at 0x13b50fe1080>



Диаграммы рассеяния

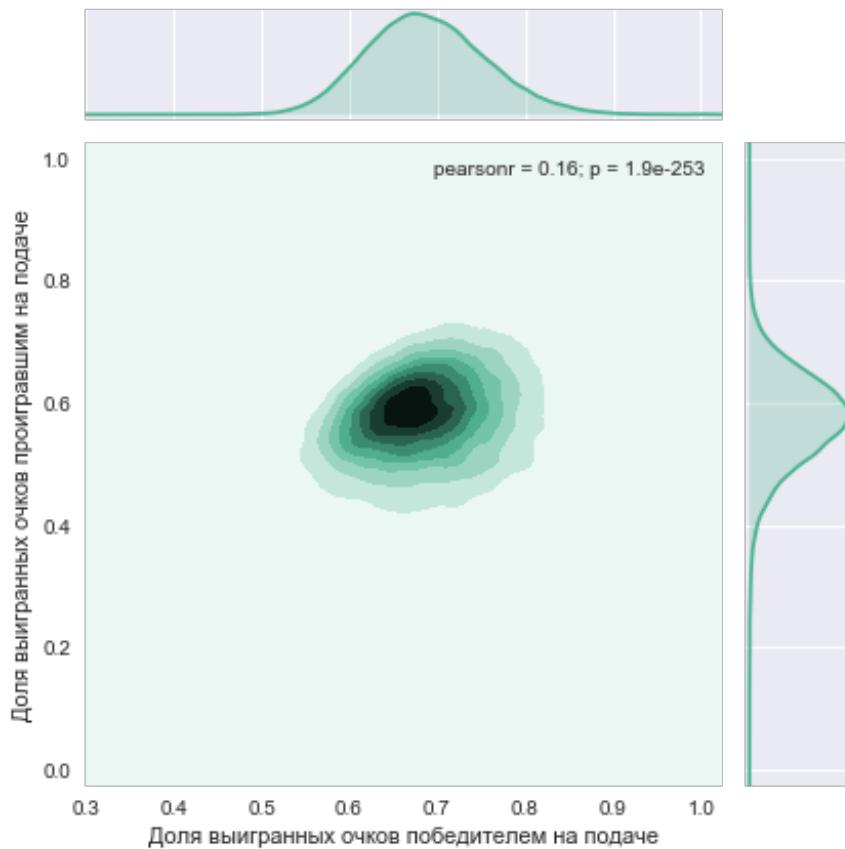
Посмотрим, как связаны доля выигранных мячей на своей подаче победителя и проигравшего:

In [33] :

```
x = matches.w_PrcsvptWon
y = matches.l_PrcsvptWon
sns.jointplot(x, y, kind="kde", color="#4CB391").set_axis_labels("Доля выигранных очков победителем на подаче",
                                                               "Доля выигранных очков проигравшим на подаче")
```

Out [33] :

<seaborn.axisgrid.JointGrid at 0x13b5102c470>



А теперь посмотрим на распределение точек в координатах "длительность матча - доля выигранных очков победителем":

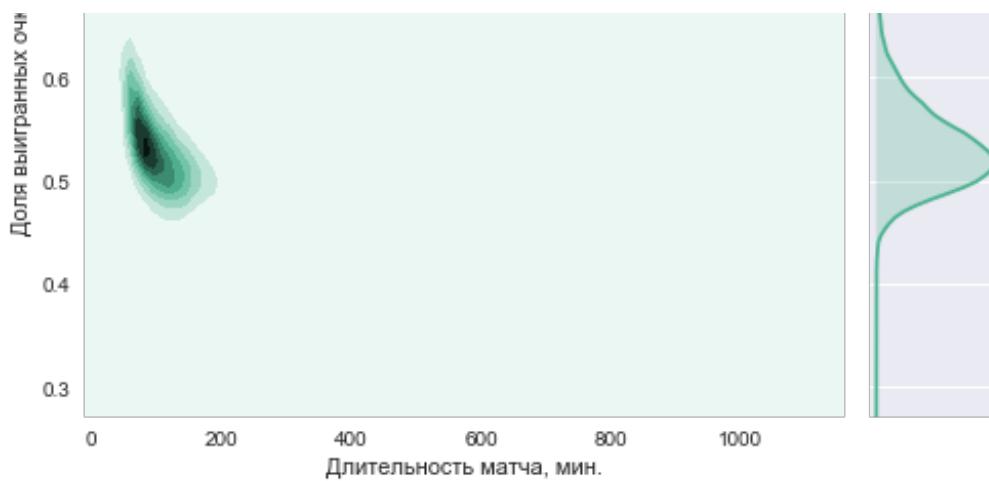
In [34] :

```
x = matches.minutes
y = matches.w_PrcptptWon
sns.jointplot(x, y, kind="kde", stat_func=kendalltau, color="#4CB391",
               size=7).set_axis_labels("Длительность матча, мин.", "Доля выигранных очков победителем")
```

Out [34] :

<seaborn.axisgrid.JointGrid at 0x13b51509908>





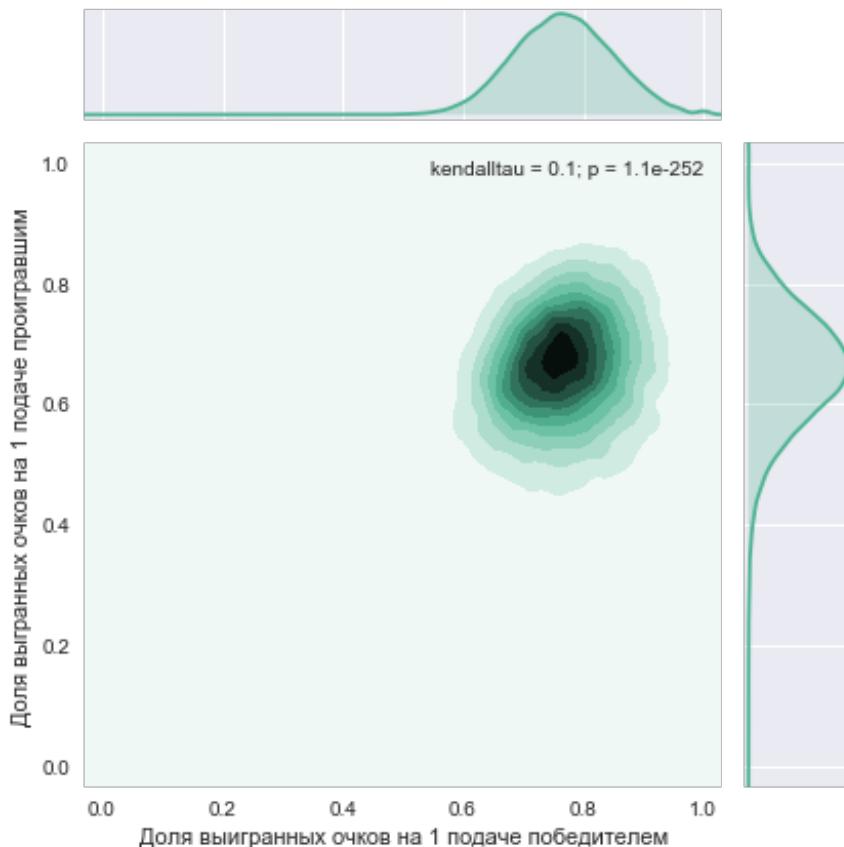
И на распределение данных о доле выигранных очков на 1 подаче победителя и проигравшего:

In [35] :

```
x = matches.w_Prc1stWon
y = matches.l_Prc1stWon
sns.jointplot(x, y, kind="kde", stat_func=kendalltau, color="#4CB391").set_
axis_labels("Доля выигранных очков на 1 подаче победителем",
"Доля выигранных очков на 1 подаче проигравшим")
```

Out [35] :

<seaborn.axisgrid.JointGrid at 0x13b5155e2e8>



Динамика основных показателей

Посмотрим, как менялись некоторые средние показатели за год в течение последних 16 лет.

In [36]:

```
plt.figure(figsize=(25, 10))
plt.suptitle('Средние показатели игроков за матч [1]', fontsize=15)

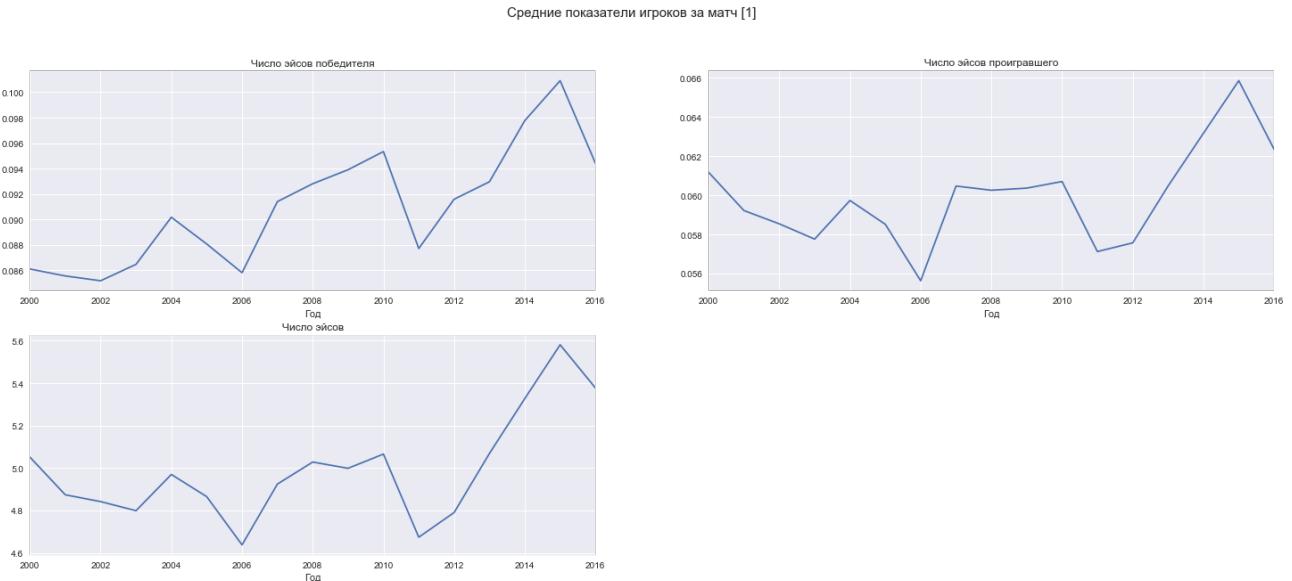
plt.subplot(2, 2, 1)
matches_gr.groupby('Tournament Year')['w_Prc Ace'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Число эйсов победителя')

plt.subplot(2, 2, 2)
matches_gr.groupby('Tournament Year')['l_Prc Ace'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Число эйсов проигравшего')

plt.subplot(2, 2, 3)
matches_gr.groupby('Tournament Year')['l Ace'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Число эйсов')
```

Out [36] :

<matplotlib.text.Text at 0x13b52ce8908>



Из графиков выше видно, что в целом число эйсов как победителя, так и проигравшего волатильны до 2011 года, затем следует стабильный рост, а в предыдущем году наблюдается сильное падение.

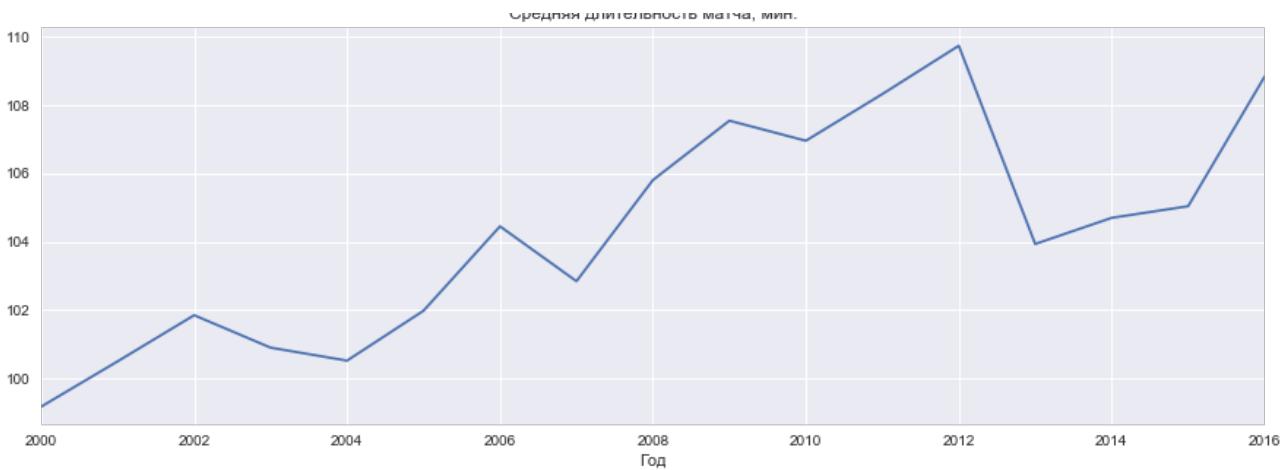
In [37] :

```
plt.figure(figsize=(15, 5))

matches_gr.groupby('Tournament Year')['minutes'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Средняя длительность матча, мин.')
```

Out [37] :

<matplotlib.text.Text at 0x13b53120320>



Средняя длительность матча в целом возрастает с трёхлетним провалом в 2013-2015 гг.

In [38]:

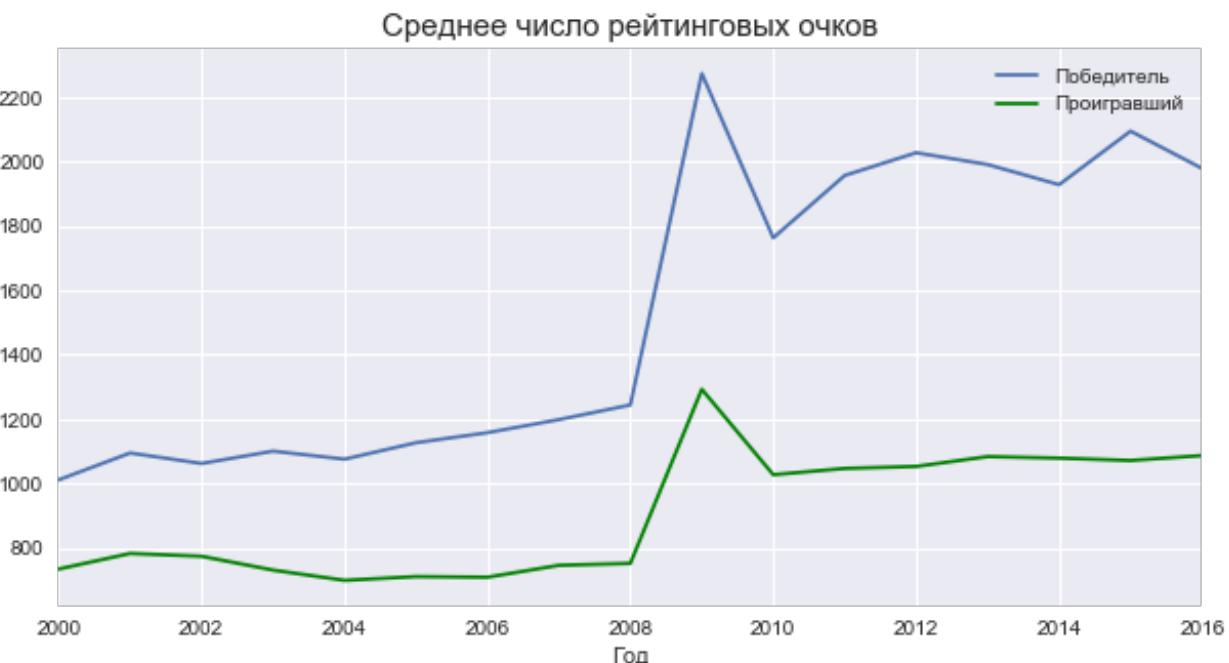
```
plt.figure(figsize=(10, 5))

matches_gr.groupby('Tournament Year')['winner_rank_points'].mean().plot(kind='line')
matches_gr.groupby('Tournament Year')['loser_rank_points'].mean().plot(kind='line', color='green')

plt.xlabel('Год')
plt.title('Среднее число рейтинговых очков', fontsize=15)
plt.legend(['Победитель', 'Проигравший'])
```

Out [38]:

<matplotlib.legend.Legend at 0x13b532c6978>



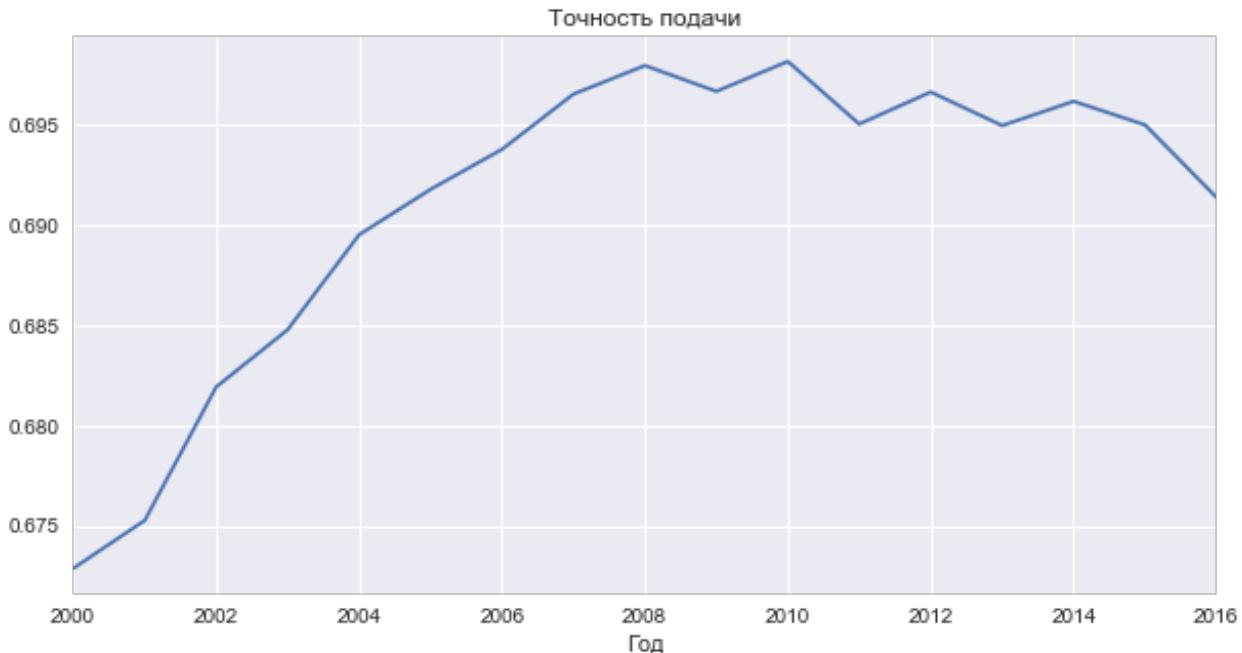
Среднее число рейтинговых очков и победителя, и проигравшего в целом растёт: то есть участники матчей ATP в среднем стали иметь больше рейтинговых очков, чем, например, в 2000 году.

In [39]:

```
plt.figure(figsize=(10, 5))
matches_gr.groupby('Tournament Year')['match_avg_ServeAccuracy'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Точность подачи')
```

Out [39] :

<matplotlib.text.Text at 0x13b535236a0>



In [40] :

```
plt.figure(figsize=(25, 10))
plt.suptitle('Средние показатели игроков за матч [2]', fontsize=15)

plt.subplot(2, 2, 1)
matches_gr.groupby('Tournament Year')['match_avg_ServeAccuracy'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Точность подачи')

plt.subplot(2, 2, 2)
matches_gr.groupby('Tournament Year')['match_avg_Prct_ace'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля эйсов')

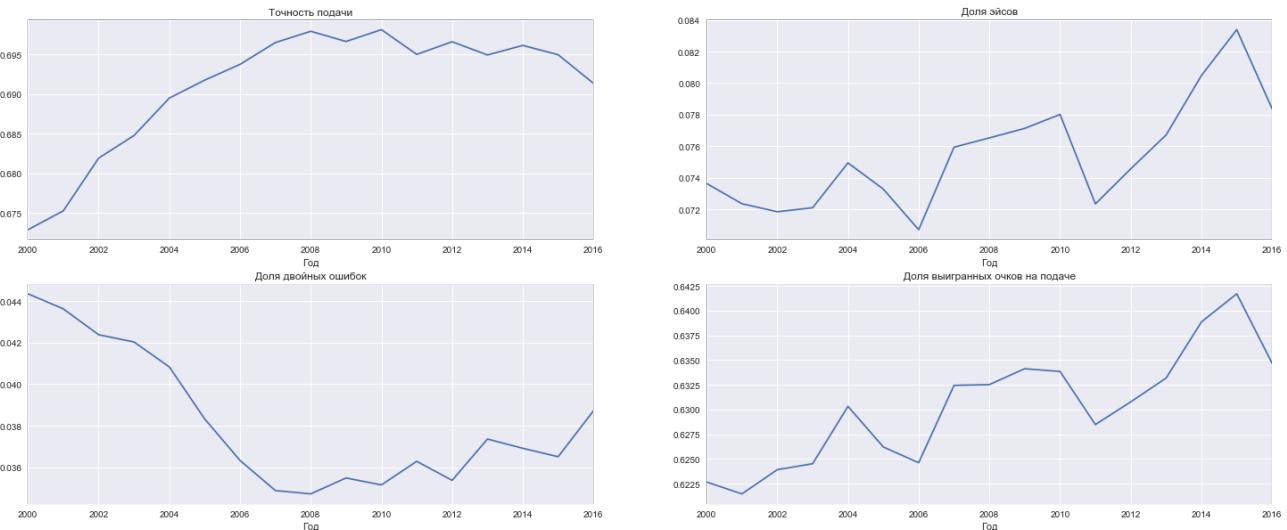
plt.subplot(2, 2, 3)
matches_gr.groupby('Tournament Year')['match_avg_Prct_df'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля двойных ошибок')

plt.subplot(2, 2, 4)
matches_gr.groupby('Tournament Year')['match_avg_PrctsvptWon'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля выигранных очков на подаче')
```

Out [40] :

```
<matplotlib.text.Text at 0x13b5345f780>
```

Средние показатели игроков за матч [2]



1. Точность подачи увеличивалась с \$2000\$ по \$2008\$ гг., затем колебалась, а с \$2014\$ года видна тенденция к её снижению.
2. При том, что точность подачи постепенно увеличивалась и с \$2014\$ постепенно уменьшалась, доля эйсов от числа розыгрышей на подаче имеет большую волатильность и не всегда движется однонаправленно с точностью подачи, что достаточно неожиданно.
3. Число двойных ошибок с \$2000\$ по \$2008\$ гг. неуклонно падало и в итоге снизилось с \$0.044\$ до \$0.035\$. С \$2009\$ по \$2016\$ наблюдается постепенных рост доли двойных ошибок.
4. График доли выигранных очков на подаче от общего числа разыгранных очков на подаче почти все 16 лет движется сонаправленно с графиком доли эйсов. Это неудивительно, ведь эйсы дают очки на подаче игрока.

In [41]:

```
plt.figure(figsize=(25, 10))
plt.suptitle('Средние показатели игроков за матч [3]', fontsize=15)

plt.subplot(2, 2, 1)
matches_gr.groupby('Tournament Year')['w_ServeAccuracy'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Точность подачи победителя')

plt.subplot(2, 2, 2)
matches_gr.groupby('Tournament Year')['l_ServeAccuracy'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Точность подачи проигравшего')

plt.subplot(2, 2, 3)
matches_gr.groupby('Tournament Year')['w_Prct_df'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля двойных ошибок победителя')

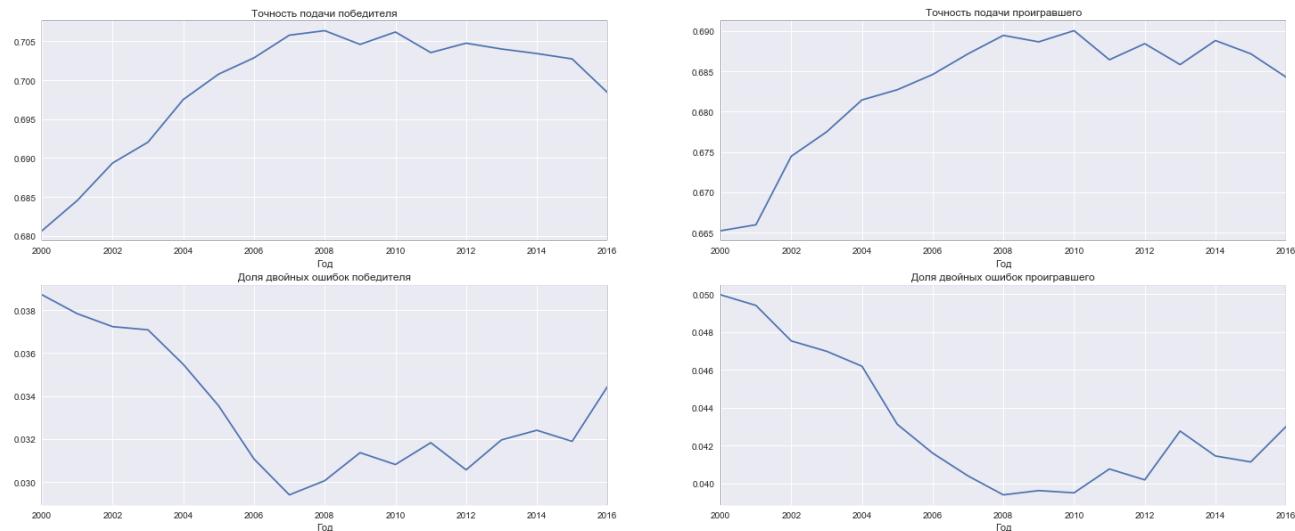
plt.subplot(2, 2, 4)
matches_gr.groupby('Tournament Year')['l_Prct_df'].mean().plot(kind='line')
plt.xlabel('Год')
```

```
plt.title('Доля двойных ошибок проигравшего')
```

Out[41]:

```
<matplotlib.text.Text at 0x13b53a16240>
```

Средние показатели игроков за матч [3]



Анализируемые выше показатели всегда лучше у победителей, что может говорить о важности этих показателей для победы в матче.

Стоит отметить, что тенденция к снижению и увеличению обоих показателей и даже резкость колебаний графиков очень похожи.

In [42]:

```
plt.figure(figsize=(25, 10))
plt.suptitle('Средние показатели игроков за матч [4]')

plt.subplot(2, 2, 1)
matches_gr.groupby('Tournament Year')['w_PrctsvptWon'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля выигранных очков победителем на подаче')

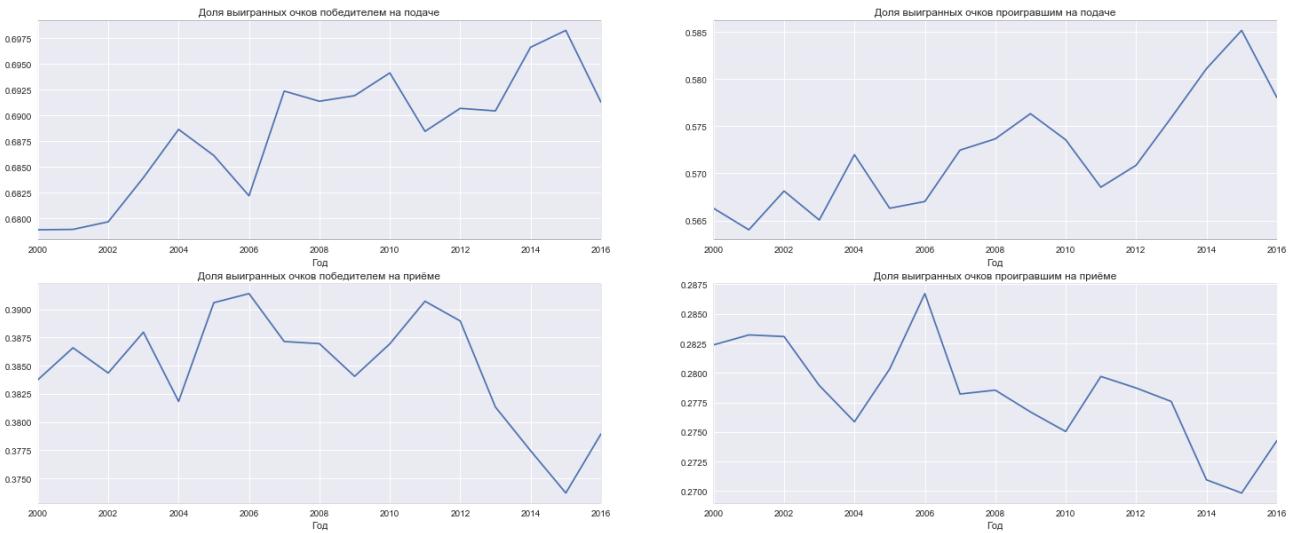
plt.subplot(2, 2, 2)
matches_gr.groupby('Tournament Year')['l_PrctsvptWon'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля выигранных очков проигравшим на подаче')

plt.subplot(2, 2, 3)
matches_gr.groupby('Tournament Year')['w_PrctRetptWon'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля выигранных очков победителем на приёме')

plt.subplot(2, 2, 4)
matches_gr.groupby('Tournament Year')['l_PrctRetptWon'].mean().plot(kind='line')
plt.xlabel('Год')
plt.title('Доля выигранных очков проигравшим на приёме')
```

Out[42]:

```
<matplotlib.text.Text at 0x13b53f32e10>
```



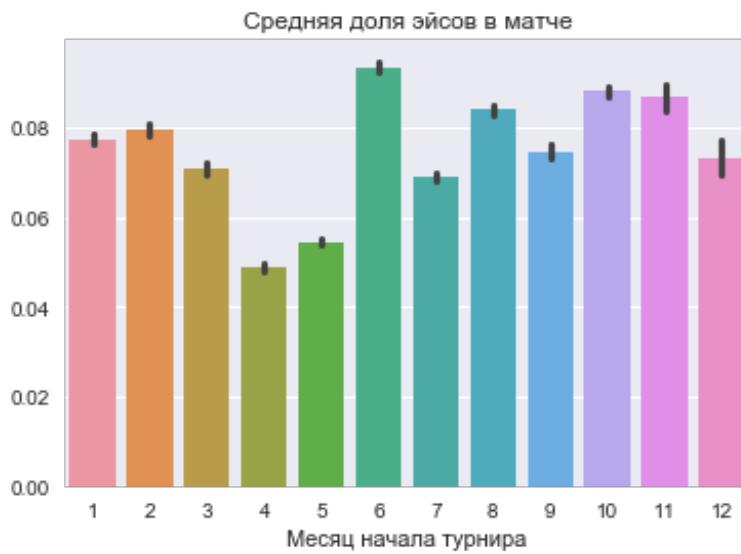
Видно, что доля выигранных очков на подаче и победителя, и проигравшего до 2016 года почти всегда росла и сильно упала в 2016 году. При этом опять показатели победителя остаются лучше, чем у проигравшего на протяжении всего времени.

In [43]:

```
sns.barplot(x="Tournament Month", y='match_Prct_ace', data=matches_gr)
plt.xlabel('Месяц начала турнира')
plt.ylabel('')
plt.title('Средняя доля эйсов в матче')
```

Out [43]:

<matplotlib.text.Text at 0x13b53fdd6a0>



На графике выше можно увидеть, что наибольшая доля эйсов за матч наблюдается в июне. Это не удивительно, так как в это время начинаются турниры на траве, а это покрытие считается самым быстрым. Как следствие, подачи сложнее принимать, и эйсов больше.

Тот факт, что мы обратили внимание на то, что в месяц, когда начинается большинство матчей на траве, самый высокий процент эйсов, намекает, что надо посмотреть на зависимость показателей игроков от типа покрытия.

Визуализация данных о типе покрытия

In [44] :

```
plt.figure(figsize=(25, 10))
plt.suptitle('Показатели за матч в зависимости от типа покрытия', fontsize=17)

plt.subplot(2, 2, 1)
sns.barplot(x='surface', y='match_ace', data=matches_gr)
plt.xlabel('Покрытие')
plt.ylabel('')
plt.title('Среднее число эйсов', fontsize=15)

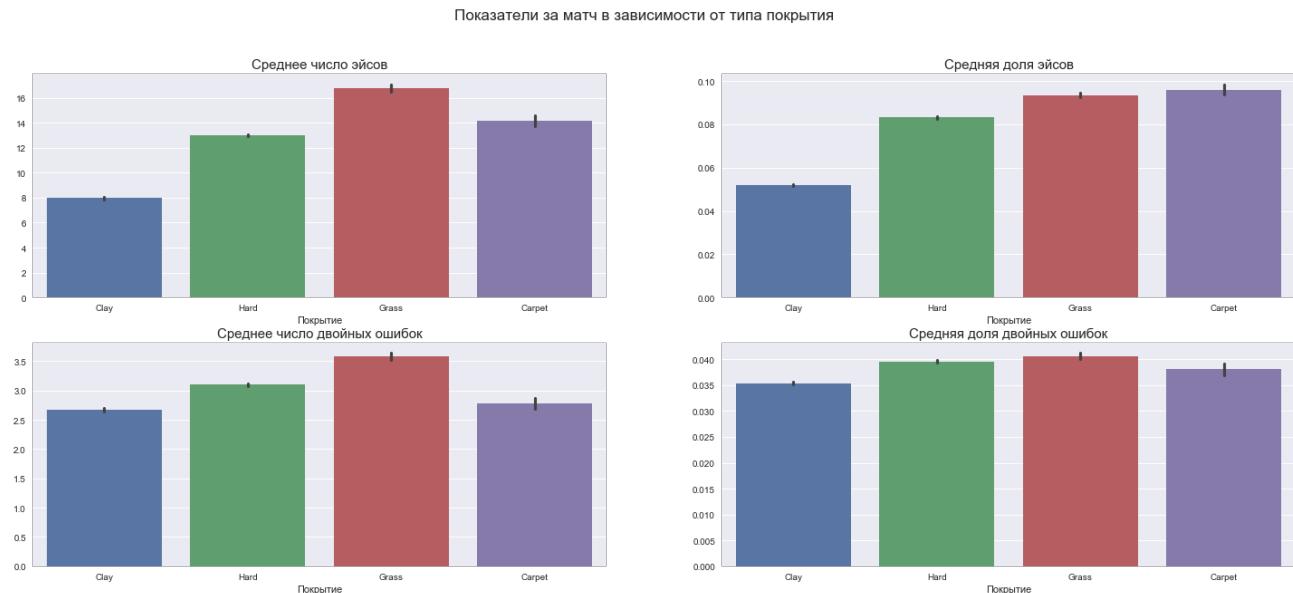
plt.subplot(2, 2, 2)
sns.barplot(x='surface', y='match_Prct_ace', data=matches_gr)
plt.xlabel('Покрытие')
plt.ylabel('')
plt.title('Средняя доля эйсов', fontsize=15)

plt.subplot(2, 2, 3)
sns.barplot(x="surface", y='match_avg_df', data=matches_gr)
plt.xlabel('Покрытие')
plt.ylabel('')
plt.title('Среднее число двойных ошибок', fontsize=15)

plt.subplot(2, 2, 4)
sns.barplot(x="surface", y='match_avg_Prct_df', data=matches_gr)
plt.xlabel('Покрытие')
plt.ylabel('')
plt.title('Средняя доля двойных ошибок', fontsize=15)
```

Out [44] :

<matplotlib.text.Text at 0x13b545ec6a0>



Примечание: матчи мужского профессионального тенниса с 2009 года на ковровом покрытии не проводятся.

1. Существует распространённое мнение, что на травяном покрытии удобно играть тем

игрокам, сильной стороной которых является подача. И если смотреть на вышерасположенную гистограмму среднего числа эйсов, можно с этим согласится. Но график средней доли эйсов говорит нам о том, что, на самом деле, именно в процентном выражении эйсов больше подавалось на ковровом покрытии! Такая разница возникает из-за того, что на траве проводится турнир Большого шлема Уимблдон, в котором игроки для победы должны взять 3 сета, а не 2, как в чемпионатах других серий. На ковре же турниры Большого шлема не проводятся: из-за этого в абсолютном выражении среднее число эйсов на траве было выше, чем на ковровом покрытии, но в процентном выражении ковёр показывают себя лучше. Стоит отметить, что ковровое покрытие также является достаточно быстрым.

2. Интересно, что на траве и число, и доля двойных ошибок были больше, чем на ковре. Какое-то объяснение этому придумать не удаётся.

Далее посмотрим на среднюю длительность матча и на закон распределения длительности матча на разных покрытиях.

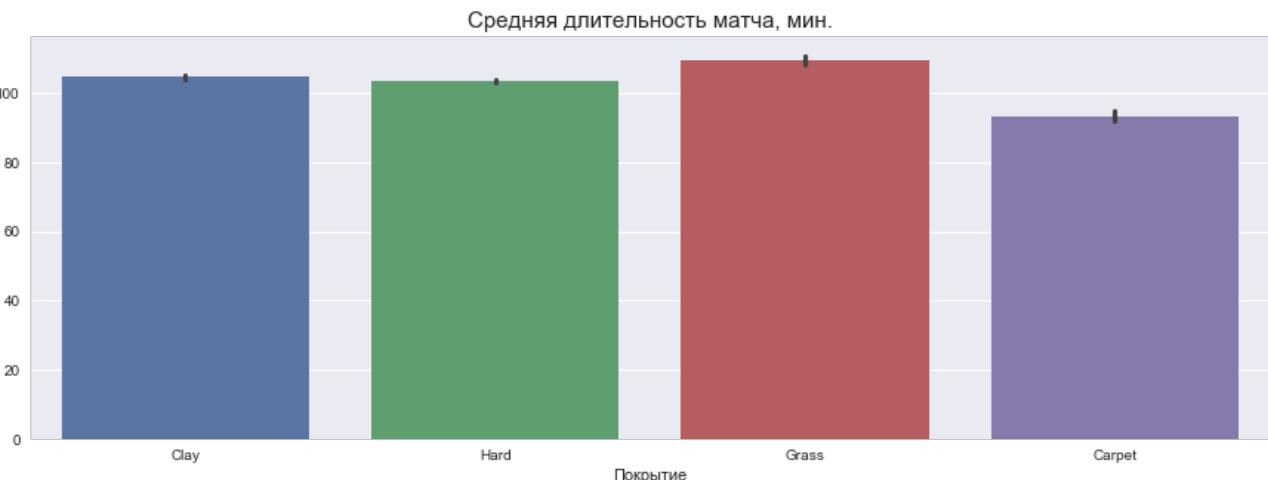
In [45] :

```
plt.figure(figsize=(15, 5))
sns.barplot(x="surface", y='minutes', data=matches)

plt.title('Средняя длительность матча, мин.', fontsize=15)
plt.xlabel('Покрытие')
plt.ylabel('')
```

Out [45] :

<matplotlib.text.Text at 0x13b544838d0>



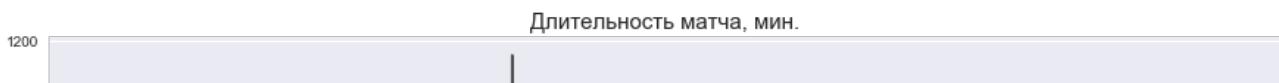
In [46] :

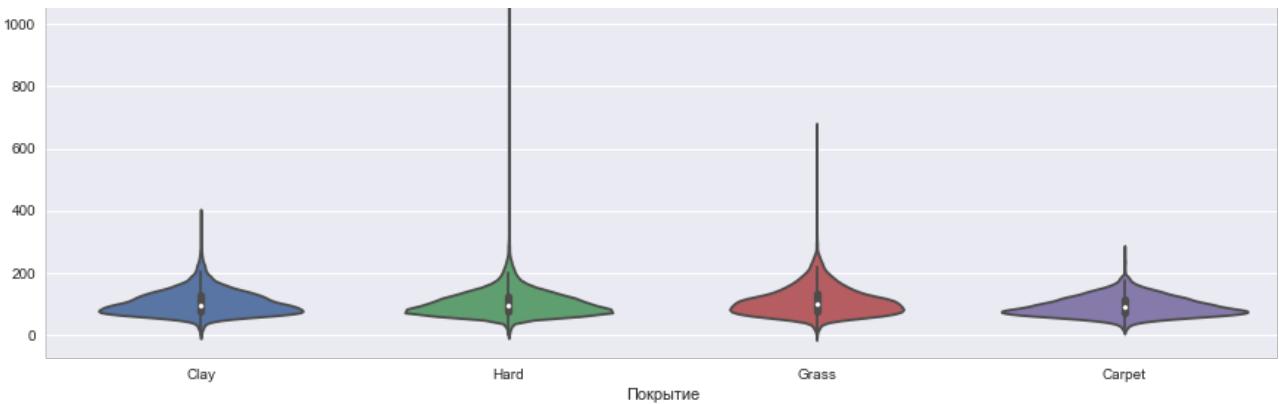
```
plt.figure(figsize=(15, 5))
sns.violinplot(x="surface", y='minutes', data=matches)

plt.title('Длительность матча, мин.', fontsize=15)
plt.xlabel('Покрытие')
plt.ylabel('')
```

Out [46] :

<matplotlib.text.Text at 0x13b54a7cb00>





Из первого графика видим, что матчи на ковре длились меньше всех, что неудивительно: как было сказано ранее, на нём не играются турниры Большого шлема. Самые длинные матчи проходят на траве, а длительность матча на грунте и харде почти одинакова. На втором графике можно увидеть, что самый большой разброс значений длительности матча наблюдается на харде. Объяснением может служить тот факт, что больше всего матчей в туре проводится именно на этом покрытии.

Визуализация данных о ведущей руке игрока

In [47] :

```
plt.figure(figsize=(15, 10))
plt.suptitle('Показатели за матч в зависимости от ведущей руки игрока')

plt.subplot(2, 2, 1)
sns.barplot(x="winner_hand", y='w_ace', data=matches)
plt.xlabel('Ведущая рука')
plt.ylabel('')
plt.title('Число эйсов победителя')

plt.subplot(2, 2, 2)
sns.barplot(x="loser_hand", y='l_ace', data=matches)
plt.xlabel('Ведущая рука')
plt.ylabel('')
plt.title('Число эйсов проигравшего')

plt.subplot(2, 2, 3)
sns.barplot(x="winner_hand", y='w_Prct_ace', data=matches)
plt.xlabel('Ведущая рука')
plt.ylabel('')
plt.title('Доля эйсов победителя')

plt.subplot(2, 2, 4)
sns.barplot(x="loser_hand", y='l_Prct_ace', data=matches)
plt.xlabel('Ведущая рука')
plt.ylabel('')
plt.title('Доля эйсов проигравшего')
```

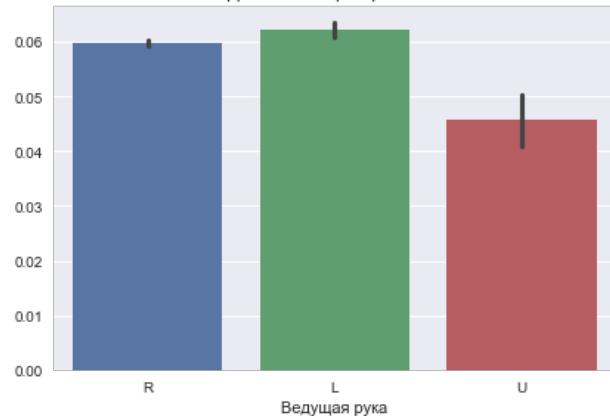
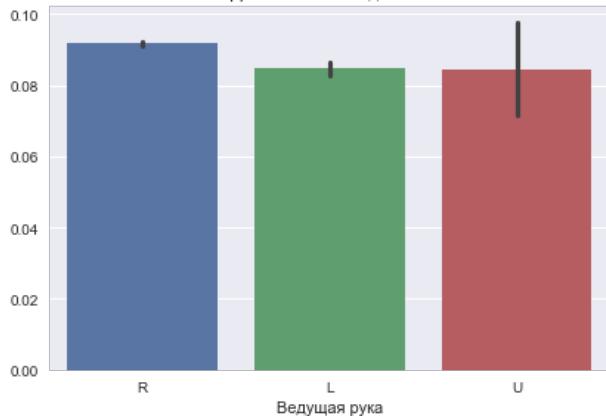
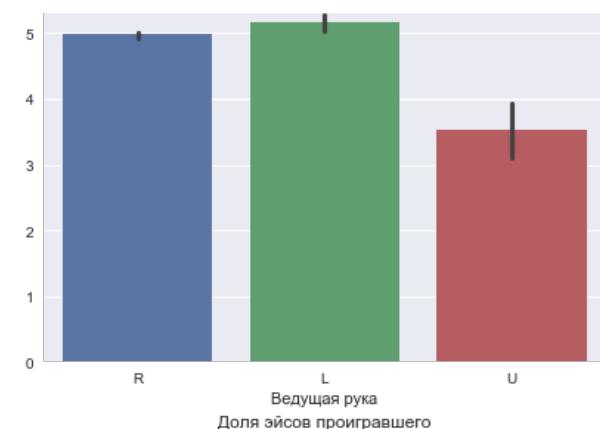
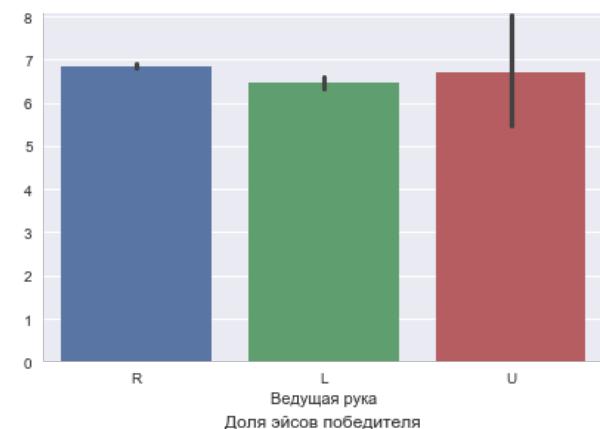
Out [47] :

<matplotlib.text.Text at 0x13b54e82a90>

Показатели за матч в зависимости от ведущей руки игрока

Число эйсов победителя

Число эйсов проигравшего



Примечание: U - 'unknown' (ведущая рука неизвестна).

Из гистограмм выше можно вынести странный факт: в среднем победитель-правша подаёт больше эйсов, чем победитель-левша, но при этом проигравший правша подаёт меньше эйсов, чем проигравший левша! Так что если заранее предполагаете, что левша выиграет правшу, ставьте на то, что он подаст меньше эйсов, чем соперник.

Работа с букмекерскими ставками на теннисные матчи

Прочитаем файлы со ставками за 2001-2016 года.

*Данные взяты с <http://www.tennis-data.co.uk/alldata.php>.

In [135]:

```
bets_2001 = pd.read_excel('2001.xls')
bets_2002 = pd.read_excel('2002.xls')
bets_2003 = pd.read_excel('2003.xls')
bets_2004 = pd.read_excel('2004.xls')
bets_2005 = pd.read_excel('2005.xls')
bets_2006 = pd.read_excel('2006.xls')
bets_2007 = pd.read_excel('2007.xls')
bets_2008 = pd.read_excel('2008.xls')
bets_2009 = pd.read_excel('2009.xls')
bets_2010 = pd.read_excel('2010.xls')
bets_2011 = pd.read_excel('2011.xls')
bets_2012 = pd.read_excel('2012.xls')
bets_2013 = pd.read_excel('2013.xls')
bets_2014 = pd.read_excel('2014.xls')
bets_2015 = pd.read_excel('2015.xls')
```

```
bets_2010 = pd.read_excel('2010.xls')
bets_2016 = pd.read_excel('2016.xls')
```

К сожалению выясняется, что данные о ставках за \$2010\$ и \$2011\$ годы отсутствуют в загруженных датафреймах по непонятным причинам и их учесть в анализе мы не сможем:

In [136]:

```
bets_2010.head()
```

Out [136]:

	ATP	Start Date	Tournament	Venue	Location	Series	Court	Surface	Players
0	1	2002-12-30	NaN	Adelaide	Australia	International Series	Outdoor	Hard	32
1	2	2002-12-30	NaN	Doha	Qatar	International Series	Outdoor	Hard	32
2	3	2002-12-30	NaN	Chennai	India	International Series	Outdoor	Hard	32
3	4	2003-01-06	NaN	Auckland	New Zealand	International Series	Outdoor	Hard	32
4	5	2003-01-06	NaN	Sydney	Australia	International Series	Outdoor	Hard	32

In [137]:

```
bets_2011.head()
```

Out [137]:

	ATP	Start Date	Tournament	Venue	Location	Series	Court	Surface	Players
0	1	2002-12-30	NaN	Adelaide	Australia	International Series	Outdoor	Hard	32
1	2	2002-12-30	NaN	Doha	Qatar	International Series	Outdoor	Hard	32
2	3	2002-12-30	NaN	Chennai	India	International Series	Outdoor	Hard	32
3	4	2003-01-06	NaN	Auckland	New Zealand	International Series	Outdoor	Hard	32
4	5	2003-01-06	NaN	Sydney	Australia	International Series	Outdoor	Hard	32

Объединим все датафреймы со ставками в один:

In [138]:

```
all_bets = pd.concat([bets_2001, bets_2002, bets_2003, bets_2004, bets_2005,
, bets_2006, bets_2007, bets_2008, bets_2009,
```

```
bets_2012, bets_2013, bets_2014, bets_2015, bets_2016  
)  
all_bets.tail()
```

Out [138] :

	ATP	AvgL	AvgW	B&WL	B&WW	B365L	B365W	Best of	CBL	CBW	...	Venue	W1	W2
2621	66	3.68	1.28	NaN	NaN	3.75	1.28	3.0	NaN	NaN	...	NaN	6.0	6
2622	66	1.56	2.41	NaN	NaN	1.57	2.37	3.0	NaN	NaN	...	NaN	3.0	6
2623	66	4.28	1.22	NaN	NaN	4.50	1.20	3.0	NaN	NaN	...	NaN	5.0	7
2624	66	4.20	1.23	NaN	NaN	4.33	1.22	3.0	NaN	NaN	...	NaN	6.0	6
2625	66	1.50	2.60	NaN	NaN	1.53	2.50	3.0	NaN	NaN	...	NaN	6.0	6

5 rows × 57 columns

In [139] :

```
len(all_bets.index)
```

Out [139] :

36173

Выше мы видим очень странную нумерацию: конец датафрейма заканчивается на номере 2625, хотя число строк в датафрейме равно 36311.

Чтобы исправить это недочёт, переиндексируем строки:

In [53] :

```
all_bets.index = range(len(all_bets.index))
```

In [54] :

```
all_bets.tail()
```

Out [54] :

	ATP	AvgL	AvgW	B&WL	B&WW	B365L	B365W	Best of	CBL	CBW	...	Venue	W1	W2
36168	66	3.68	1.28	NaN	NaN	3.75	1.28	3.0	NaN	NaN	...	NaN	6.0	6
36169	66	1.56	2.41	NaN	NaN	1.57	2.37	3.0	NaN	NaN	...	NaN	3.0	6
36170	66	4.28	1.22	NaN	NaN	4.50	1.20	3.0	NaN	NaN	...	NaN	5.0	7
36171	66	4.20	1.23	NaN	NaN	4.33	1.22	3.0	NaN	NaN	...	NaN	6.0	6

	ATP	AvgL	AvgW	B&WL	B&WW	B365L	B365W	Best of	CBL	CBW	...	Venue	W1	W2
36172	66	1.50	2.60	NaN	NaN	1.53	2.50	3.0	NaN	NaN	...	NaN	6.0	6

5 rows × 57 columns

In [55]:

```
all_bets[['Date', 'Winner', 'Loser', 'AvgW']].head()
```

Out[55]:

	Date	Winner	Loser	AvgW
0	2001-01-01	Clement A.	Gaudenzi A.	NaN
1	2001-01-01	Goldstein P.	Jones A.	NaN
2	2001-01-01	Haas T.	Smith L.	NaN
3	2001-01-01	Henman T.	Rusedski G.	NaN
4	2001-01-01	Hewitt L.	Arthurs W.	NaN

Так лучше!

Посмотрим информацию о датафрейме 'all_bets':

In [56]:

```
all_bets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36173 entries, 0 to 36172
Data columns (total 57 columns):
ATP            36173 non-null int64
AvgL           10480 non-null float64
AvgW           10480 non-null float64
B&WL          1097 non-null float64
B&WW          1097 non-null float64
B365L          30497 non-null float64
B365W          30493 non-null float64
Best of        36104 non-null float64
CBL            17507 non-null float64
CBW            17507 non-null float64
Comment         36104 non-null object
Court           36173 non-null object
Date            36104 non-null datetime64[ns]
EXL             26384 non-null float64
EXW             26379 non-null object
GBL             5055 non-null float64
GBW             5055 non-null float64
IWL             13358 non-null float64
IWW             13358 non-null float64
L1              35931 non-null float64
L2              35578 non-null object
L3              16643 non-null object
L4              3234 non-null float64
T               1215 non-null float64
```

```
    ...  
L5C          1245 non-null float64  
LBL          15722 non-null float64  
LBW          15719 non-null float64  
LPts         22821 non-null float64  
LRank        36044 non-null object  
Location     36173 non-null object  
Loser         36104 non-null object  
Lsets         35957 non-null object  
MaxL          10480 non-null float64  
MaxW          10480 non-null float64  
PSL           24202 non-null float64  
PSW           24202 non-null float64  
Players       69 non-null float64  
Round         36104 non-null object  
SBL           5424 non-null float64  
SBW           5424 non-null float64  
SJL           7747 non-null float64  
SJW           7747 non-null float64  
Series        36173 non-null object  
Start Date   69 non-null datetime64[ns]  
Surface       36173 non-null object  
Tournament    36104 non-null object  
UBL           10672 non-null float64  
UBW           10672 non-null float64  
Venue          69 non-null object  
W1            35931 non-null float64  
W2            35577 non-null object  
W3            16644 non-null object  
W4            3234 non-null float64  
W5            1245 non-null float64  
WPts          22864 non-null float64  
WRank         36091 non-null float64  
Winner        36104 non-null object  
Wsets         35956 non-null float64  
dtypes: datetime64[ns] (2), float64(37), int64(1), object(17)  
memory usage: 15.7+ MB
```

Узнаем ради любопытства, какие раунды представлены в этом датафрейме:

In [57]:

```
all_bets['Round'].value_counts() #поэтому датасеты нельзя объединить по roundам
```

Out [57]:

```
1st Round      16788  
2nd Round      9904  
Quarterfinals  3388  
3rd Round      2568  
Semifinals    1721  
The Final      859  
4th Round      600  
Round Robin    276  
Name: Round, dtype: int64
```

А теперь посмотрим на даты:

In [58]:

```
all_bets['Date'].head()
```

Out [58] :

```
0    2001-01-01  
1    2001-01-01  
2    2001-01-01  
3    2001-01-01  
4    2001-01-01  
Name: Date, dtype: datetime64[ns]
```

Ух ты! Они представлены ровно в том формате, в который мы переделывали даты из датафрейма 'matches' - в формате 'datetime'.

Создадим колонки с годом, месяцем и днём: они могут пригодиться.

In [59] :

```
all_bets['Year'] = np.int_(all_bets['Date'].dt.year)  
all_bets['Month'] = np.int_(all_bets['Date'].dt.month)  
all_bets['Day'] = np.int_(all_bets['Date'].dt.day)  
  
all_bets[['Date', 'Year', 'Month', 'Day']].head()
```

Out [59] :

	Date	Year	Month	Day
0	2001-01-01	2001	1	1
1	2001-01-01	2001	1	1
2	2001-01-01	2001	1	1
3	2001-01-01	2001	1	1
4	2001-01-01	2001	1	1

В информации о датасете можно было найти колонку 'Start Date'. Что за информация находится в ней? Дата начала турнира, как в датафрейме 'matches', или что-то еще? Посмотрим, много ли значений в ней есть вообще:

In [60] :

```
all_bets['Start Date'].describe()
```

Out [60] :

```
count                  69  
unique                 37  
top       2003-07-07 00:00:00  
freq                   3  
first      2002-12-30 00:00:00  
last       2003-11-10 00:00:00  
Name: Start Date, dtype: object
```

Всего 207 из 36310. Неинформативный признак, получается.

Посмотрим ещё на пару подозрительных признаков.

In [61]:

```
all_bets['Players'].describe()
```

Out[61]:

```
count      69.000000
mean       44.057971
std        24.909179
min        8.000000
25%       32.000000
50%       32.000000
75%       48.000000
max       128.000000
Name: Players, dtype: float64
```

In [62]:

```
all_bets['Venue'].describe()
```

Out[62]:

```
count          69
unique         69
top           Memphis
freq           1
Name: Venue, dtype: object
```

Колонки 'Players', 'Start Date' и 'Venue' практически не несут в себе никакой информации.

Удалим их из датасета:

In [63]:

```
all_bets = all_bets.drop(['Players', 'Start Date', 'Venue'], axis=1)
```

Теперь посмотрим на ещё один подозрительный признак: 'Court':

In [64]:

```
all_bets['Court'].value_counts()
```

Out[64]:

```
Outdoor     29685
Indoor      6488
Name: Court, dtype: int64
```

Его удалять не будем, так как он несёт информацию о каждом матче: был он сыгран на улице (outdoor) или в помещении (indoor).

Выделим в отдельный датасет ставки на победителей и проигравших в разных букмекерских конторах, чтобы поработать с ними:

In [65]:

```
winner_bets = all_bets[['B365W', 'CBW', 'GBW', 'IWW', 'SBW', 'B&WW', 'EXW', 'LBW', 'PSW', 'UBW', 'SJW']]
loser_bets = all_bets[['B365L', 'CBL', 'GBL', 'IWL', 'SBL', 'B&WL', 'EXL', 'L']]
```

```
BL', 'PSL', 'UBL', 'SJL']]
```

Сделаем полностью заполненные значениями максимальной котировки на победителя и проигравшего колонки:

In [66]:

```
all_bets['MaxW'] = winner_bets.max(axis=1)
all_bets['MaxL'] = loser_bets.max(axis=1)
```

Теперь вычислим среднюю котировку из представленных в датафрейме для победителя и проигравшего:

In [67]:

```
all_bets['AvgW'] = winner_bets.mean(axis=1) # вычисляем среднюю ставку на и
грука из коэффициентов имеющихся букмекерских контор
all_bets['AvgL'] = loser_bets.mean(axis=1)
```

Теперь вычислим такой статистических показатель, как стандартное отклонение котировок букмекерских контор друг от друга.

In [68]:

```
all_bets['StdW'] = winner_bets.std(axis=1)
all_bets['StdL'] = loser_bets.std(axis=1)
```

Посмотрим на новые данные в совокупности с некоторыми старыми:

In [69]:

```
all_bets[['Winner', 'Wsets', 'Lsets', 'Loser', 'AvgW', 'AvgL', 'StdW',
'StDl', 'Tournament', 'Location', 'Round', 'Year']].tail()
```

Out [69]:

	Winner	Wsets	Lsets	Loser	AvgW	AvgL	StdW	StdL	Tournament	Loc
36168	Murray A.	2.0	0	Wawrinka S.	1.293333	3.6525	0.015275	0.236696	Masters Cup	Lc
36169	Cilic M.	2.0	1	Nishikori K.	2.446667	1.5525	0.068069	0.047871	Masters Cup	Lc
36170	Murray A.	2.0	1	Raonic M.	1.230000	4.2525	0.026458	0.423901	Masters Cup	Lc
36171	Djokovic N.	2.0	0	Nishikori K.	1.230000	4.2400	0.017321	0.342734	Masters Cup	Lc
36172	Murray A.	2.0	0	Djokovic N.	2.633333	1.4825	0.125831	0.075000	Masters Cup	Lc

Посмотрим, как выглядят признаки 'Wsets' и 'Lsets':

```
In [70]:
```

```
all_bets[['Winner', 'Wsets', 'Lsets', 'Loser', 'Date', 'Year']].head()
```

```
Out[70]:
```

	Winner	Wsets	Lsets	Loser	Date	Year
0	Clement A.	2.0	1	Gaudenzi A.	2001-01-01	2001
1	Goldstein P.	2.0	0	Jones A.	2001-01-01	2001
2	Haas T.	2.0	0	Smith L.	2001-01-01	2001
3	Henman T.	2.0	0	Rusedski G.	2001-01-01	2001
4	Hewitt L.	2.0	0	Arthurs W.	2001-01-01	2001

Переведём значения выигранных победителем сетов и его место в рейтинге в целые числа:

```
In [71]:
```

```
all_bets['Wsets'] = np.int_(all_bets['Wsets'])
```

```
In [72]:
```

```
all_bets['WRank'] = np.int_(all_bets['WRank'])
```

Теперь датафрейм выглядит намного лучше.

Графики: all_bets

Далее будем предполагать, что:

- чем выше средняя ставка на проигравшего, тем точнее предсказывается исход матча
- чем ниже средняя ставка на победителя, тем точнее предсказывается исход матча

Зависимость точности ставок от этапа турнира

Посмотрим, как зависит точность ставок от раунда, а также изобразим распределение ставок на матчи разных этапов турнира.

```
In [73]:
```

```
plt.figure(figsize=(25, 10))

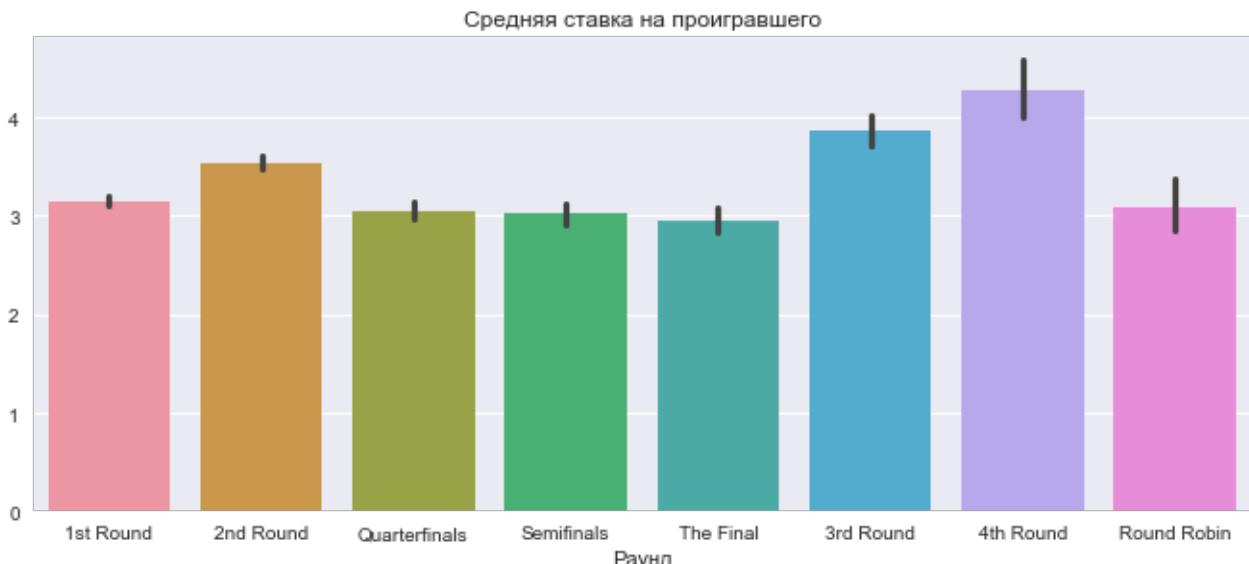
plt.subplot(2, 2, 1)
sns.barplot(x="Round", y='AvgL', data=all_bets)
plt.title('Средняя ставка на проигравшего')
plt.ylabel('')
plt.xlabel('Раунд')

plt.subplot(2, 2, 3)
sns.barplot(x="Round", y='AvgW', data=all_bets)
plt.title('Средняя ставка на победителя')
plt.ylabel('')
```

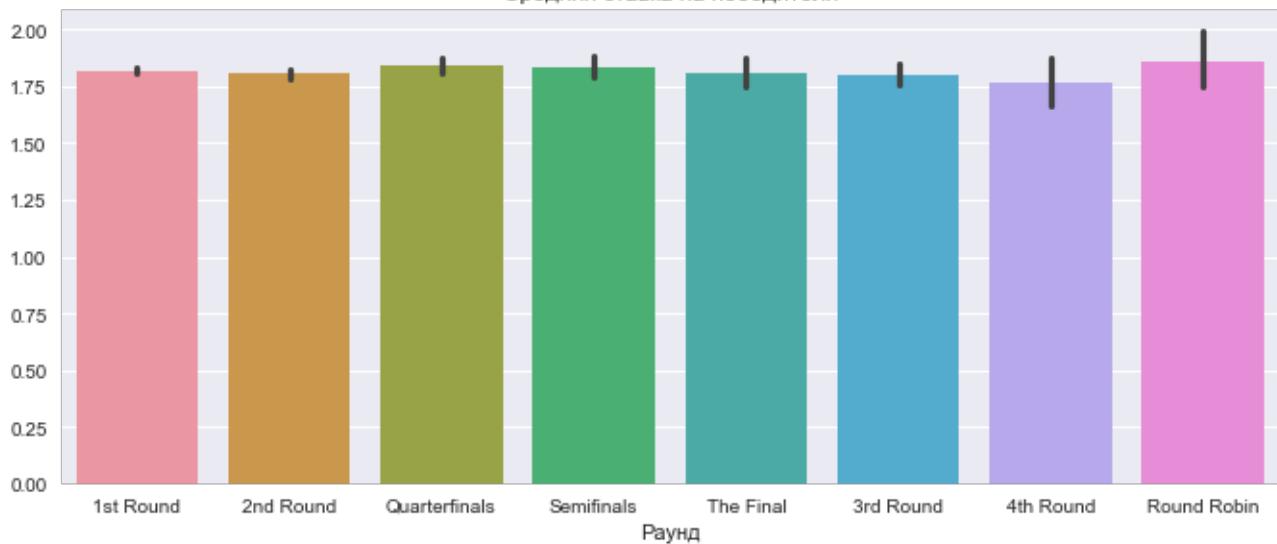
```
plt.xlabel('Раунд')
```

Out[73]:

```
<matplotlib.text.Text at 0x13b5ee20828>
```



Средняя ставка на победителя



In [74]:

```
plt.figure(figsize=(25, 10))

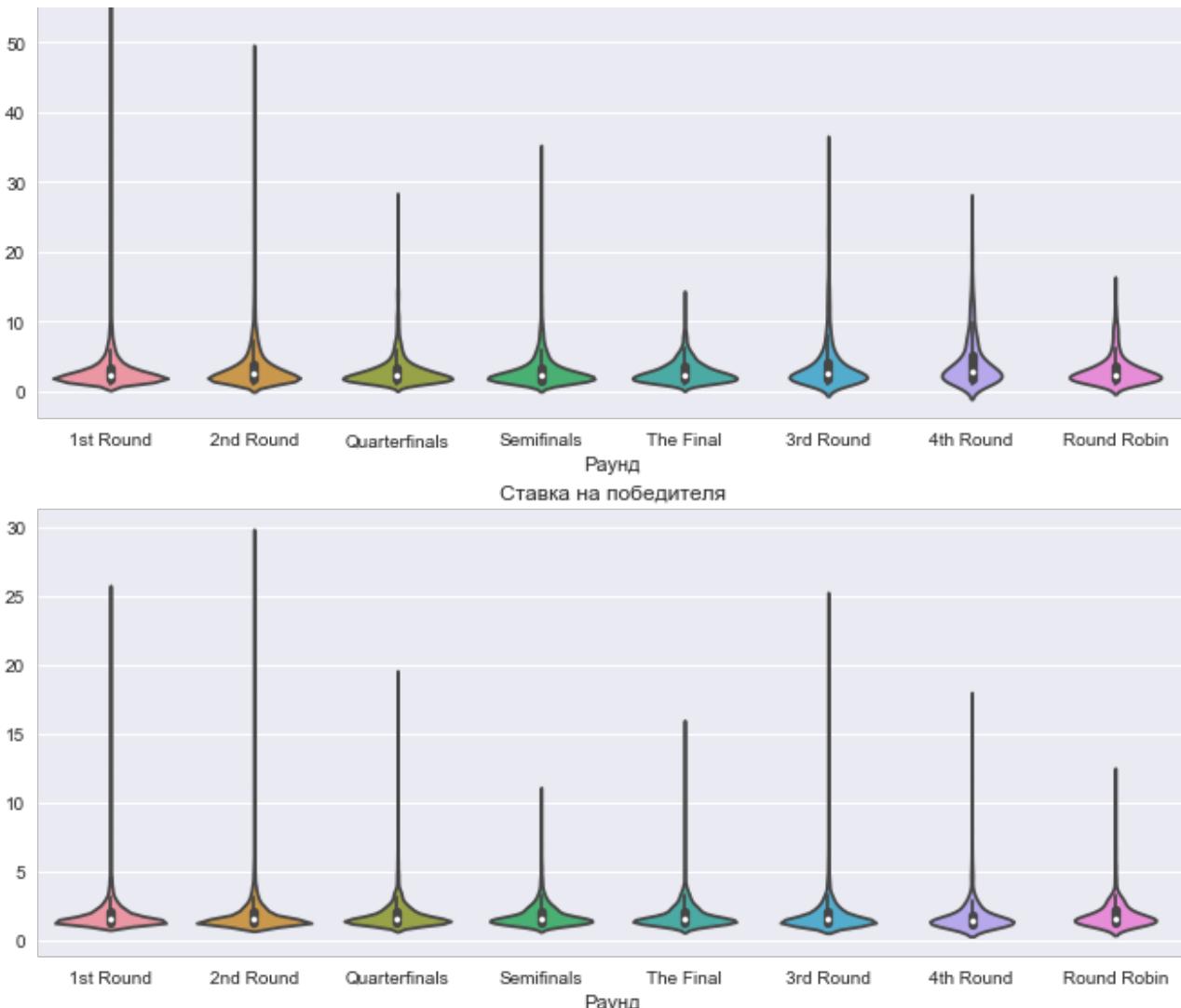
plt.subplot(2, 2, 1)
sns.violinplot(x="Round", y='AvgL', data=all_bets)
plt.title('Ставка на проигравшего')
plt.ylabel('')
plt.xlabel('Раунд')

plt.subplot(2, 2, 3)
sns.violinplot(x="Round", y='AvgW', data=all_bets)
plt.title('Ставка на победителя')
plt.ylabel('')
plt.xlabel('Раунд')
```

Out[74]:

```
<matplotlib.text.Text at 0x13b5d018160>
```

Ставка на проигравшего



Узнаем, что такое Round Robin.

In [75]:

```
robin = all_bets[all_bets['Round'] == 'Round Robin']
robin['Tournament'].value_counts()
```

Out [75]:

Masters Cup	156
Movistar Open	24
Copa Telmex	24
Channel Open	24
Next Generation Adelaide International	24
International Championships	24
Name: Tournament, dtype: int64	

Это матчи в основном групповых стадий Итогового турнира. Они немного испортят соединение датафреймов, так как в этих соревнованиях могут встречаться одни и те же игроки (но такое происходит очень редко).

Выводы из графиков:

- Хуже всего букмекерский рынок делает предсказания на групповом этапе (Round Robin)
- Лучше всего букмекерский рынок предсказывает исходы матчей 4 круга (4th Round)

3. Наибольший разброс значений ставок наблюдается в первых трёх раундах турнира

Динамика ставок

Далее посмотрим, как менялись показатели ставок с течение времени (без учёта 2010 и 2011 годов).

In [76]:

```
plt.figure(figsize=(15, 10))

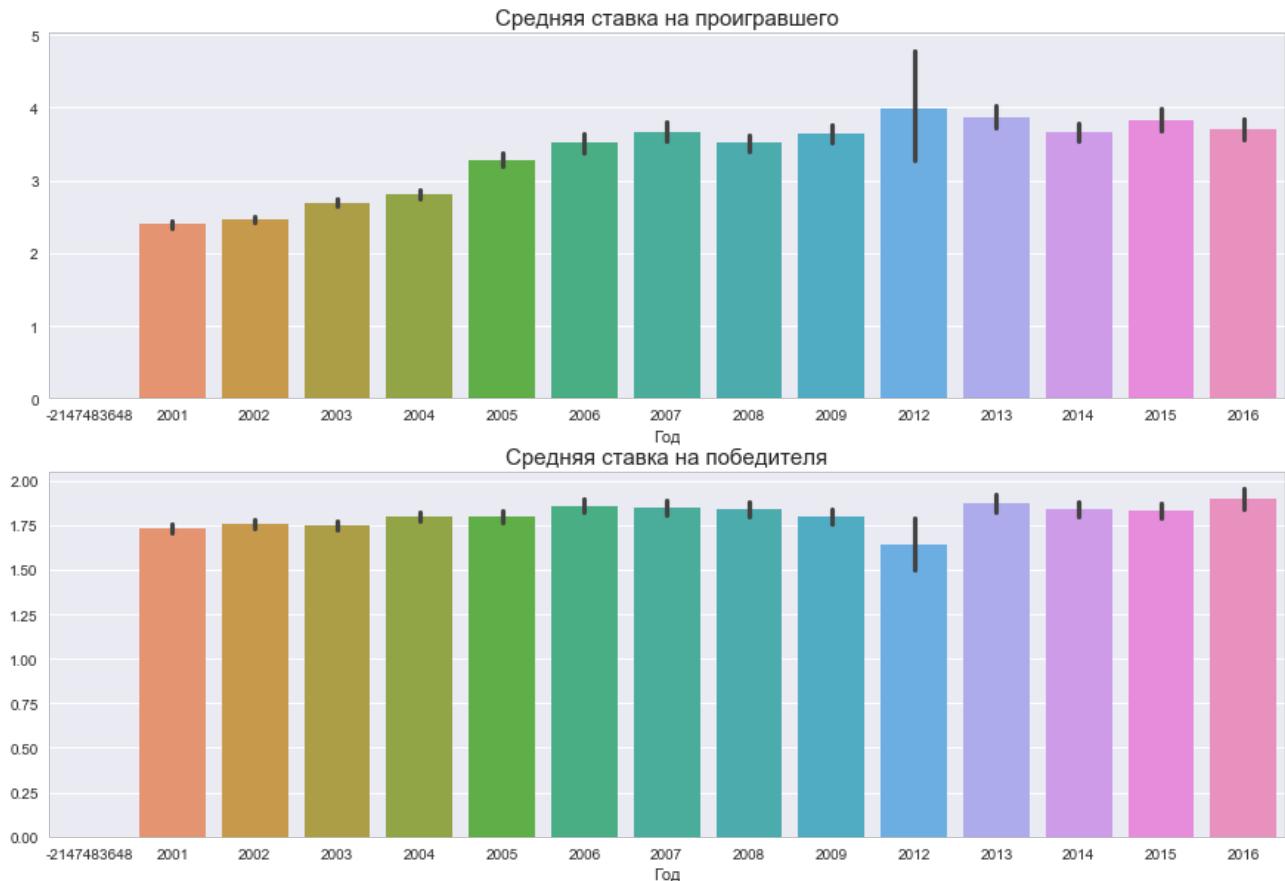
plt.subplot(2, 1, 1)
sns.barplot(x="Year", y='AvgL', data=all_bets)
plt.title('Средняя ставка на проигравшего', fontsize=15)
plt.ylabel('')
plt.xlabel('Год')

plt.subplot(2, 1, 2)
sns.barplot(x="Year", y='AvgW', data=all_bets)

plt.title('Средняя ставка на победителя', fontsize=15)
plt.ylabel('')
plt.xlabel('Год')
```

Out [76]:

```
<matplotlib.text.Text at 0x13b5e0d4550>
```



Из графика средней ставки на проигравшего лучше всего видно, что с \$2001\$ по \$2007\$ год точность предсказаний росла, а затем оставалась примерно на одном уровне.

Зависимость точности ставок от турнира Большого шлема

Далее посмотрим, на каких турнирах Большого шлема люди лучшего всего предсказывают исход матча.

In [77] :

```
GS = all_bets[all_bets['Series'] == 'Grand Slam']
```

In [78] :

```
plt.figure(figsize=(15, 10))

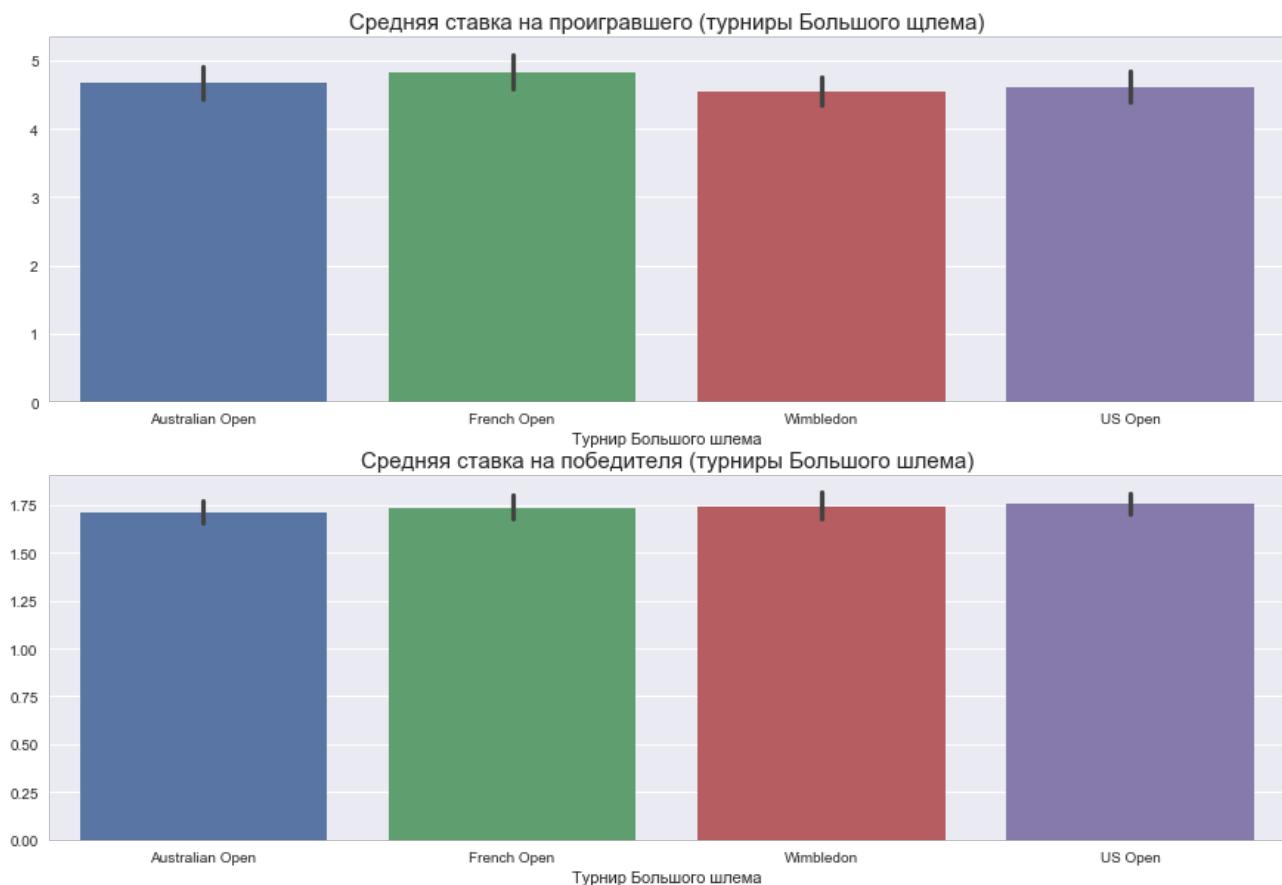
plt.subplot(2, 1, 1)
sns.barplot(x="Tournament", y='AvgL', data=GS)
plt.title('Средняя ставка на проигравшего (турниры Большого шлема)', fontsize=15)
plt.ylabel('')
plt.xlabel('Турнир Большого шлема')

plt.subplot(2, 1, 2)
sns.barplot(x="Tournament", y='AvgW', data=GS)

plt.title('Средняя ставка на победителя (турниры Большого шлема)', fontsize=15)
plt.ylabel('')
plt.xlabel('Турнир Большого шлема')
```

Out [78] :

```
<matplotlib.text.Text at 0x13b5e233e10>
```



Из графиков выше видим, что лучше всего букмекерский рынок предсказывает исходы матчей на Ролан Гарросе (French Open).

Зависимость точности ставок от типа покрытия

In [79]:

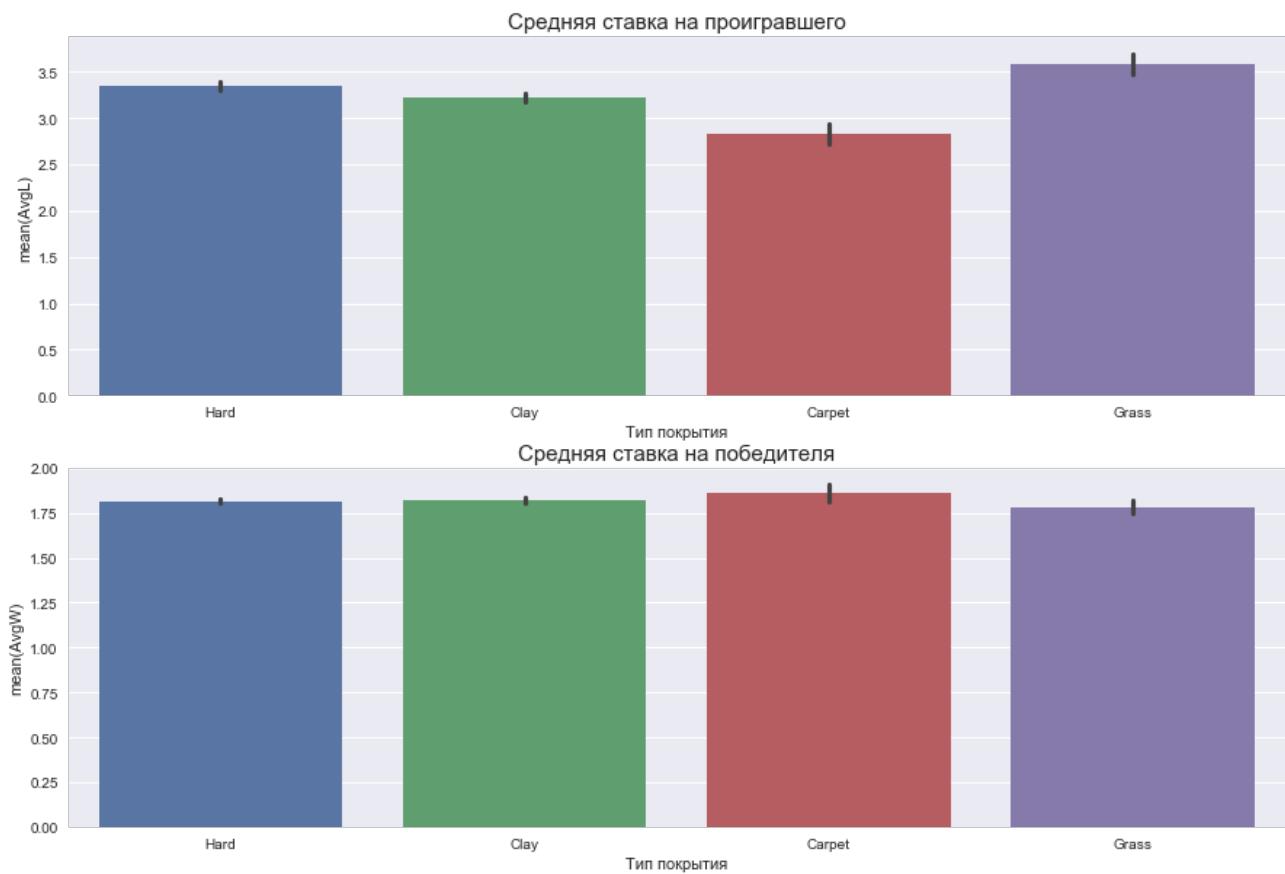
```
plt.figure(figsize=(15, 10))

plt.subplot(2, 1, 1)
sns.barplot(x="Surface", y='AvgL', data=all_bets)
plt.title('Средняя ставка на проигравшего', fontsize=15)
plt.xlabel('Тип покрытия')

plt.subplot(2, 1, 2)
sns.barplot(x="Surface", y='AvgW', data=all_bets)
plt.title('Средняя ставка на победителя', fontsize=15)
plt.xlabel('Тип покрытия')
```

Out [79]:

```
<matplotlib.text.Text at 0x13b5ae810f0>
```



Выводы:

1. Букмекерский рынок лучше всего предсказывает матчи на траве
2. Букмекерский рынок хуже всего предсказывал матчи на ковре (до того как их отменили в 2009 г.)

Соединение датафреймов со статистикой матчей и ставок

Чтобы сделать данное действие, стоит иметь в виду, что:

1. В одном турнире два игрока могут встретиться в одиночном разряде только один раз
2. Каждый турнир проходит лишь раз в году
3. Турниры Большого шлема делятся 2 недели, остальные - неделю

Таким образом, чтобы понять, что два набора данных относятся к одному матчу, нужно проверить лишь совпадение имен победителя ('Winner') и проигравшего ('Loser'), года и названия турнира. Но всё же стоит проверить, являются ли, например, названия одних и тех же турниров одинаковыми в обоих датафреймах?

Случайным образом выберем строку из датафрейма 'matches' - 25506.

In [80]:

```
Question = matches[25505:25507]
Question[['Winner', 'Loser', 'tourney_name', 'Tournament Year']]
```

Out [80]:

	Winner	Loser	tourney_name	Tournament Year
25505	Karlovic I.	Hartfield D.	Houston	2007
25506	Melzer J.	Hanescu V.	Houston	2007

Обратим внимание на название турнира - 'Houston'. Есть вероятность, что в датафрейме со ставками данный турнир назван каким-то более специфичным способом. Проверим:

In [81]:

```
Houston = all_bets[(all_bets['Winner'] == 'Melzer J.') & (all_bets['Loser'] == 'Hanescu V.') & (all_bets['Year'] == 2007)]
Houston['Tournament']
```

Out [81]:

```
18234    U.S. Men's Clay Court Championships
Name: Tournament, dtype: object
```

Оказывается, что так и есть. Посмотрим, а какие вообще названия турниров встречаются наиболее часто в обоих датафреймах. Предположим, что наиболее встречающиеся турниры в обоих датафреймах должны быть примерно одними и теми же, но могут быть названы по-разному.

In [82]:

```
matches['tourney_name'].value_counts()[:12]
```

Out [82]:

Roland Garros	2159
Wimbledon	2159
Australian Open	2159
US Open	2032
Miami Masters	1615

```
Indian Wells Masters      148 /  
Monte Carlo Masters     991  
Rome Masters             991  
Cincinnati Masters      991  
Canada Masters           983  
Barcelona                903  
Queen's Club              825  
Name: tourney_name, dtype: int64
```

In [83]:

```
all_bets['Tournament'].value_counts()[:12]
```

Out [83]:

```
US Open                  1651  
French Open               1651  
Wimbledon                 1651  
Australian Open           1651  
Monte Carlo Masters       763  
Sony Ericsson Open        665  
Western & Southern Financial Group Masters 637  
Pacific Life Open          601  
Hamburg TMS                488  
NASDAQ-100 Open            475  
BNP Paribas Open           475  
Mercedes Cup                463  
Name: Tournament, dtype: int64
```

Как мы видим, названия трёх турниров Большого шлема (Wimbledon, Australian Open и US Open) совпадают. В то же время, Roland Garros и French Open - это один и тот же турнир (четвёртый турнир Большого шлема), то есть соединить матчи этого соревнования не получится, что приведёт к значительной потере информации (1651 матч на грунте). Более того, можно заметить тенденцию, что в датафрейме 'matches' турниры часто носят названия городов, в которых они проводятся, тогда как в 'all_bets' в названиях чемпионатов можно увидеть имена их спонсоров: Sony Ericsson, BNP Paribas, Mercedes, что приводит к большому числу несовпадающих названий турниров.

Таким образом, равенство названия турниров - плохое условие для соединение датафреймов.

Можно было бы соединять матчи, пользуясь условием равенства дат матчей. Но, как уже можно было видеть, в датафрейме 'matches' есть только даты начала матчей турниров, тогда как в датафрейме 'all_bets' - даты самих матчей. Но, на самом деле, такие данные мы можем обратить в преимущество, воспользовавшись фактом №3 о теннисных матчах.

Соединение датафреймов с использованием имён победителя и проигравшего и дат датасетов

Сначала объединим датафреймы 'matches' и 'all_bets' с помощью функции `merge`. В её скобках записаны названия объединяемых датафреймов, равенство определяет признаки, на основе которых будет произведено объединение, `how` - какие образом (с помощью объединения, пересечения или других способов; используем `inner` - пересечение).

In [84]:

```
matches_bets = pd.merge(matches, all_bets, on='Tournament', how='inner')
```

```
matches_bets = pd.merge(matches, all_bets, on=['winner', 'loser'], how='inner')
matches_bets.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 77195 entries, 0 to 77194
Columns: 187 entries, tourney_id to StdL
dtypes: datetime64[ns](2), float64(140), int32(8), int64(7), object(30)
memory usage: 108.4+ MB
```

Теперь у нас есть датафрейм со всеми матчами, где совпали указания победителя и проигравшего в обоих датафреймах. Очевидно, что такой датафрейм не учитывает других данных о матче, что значительно снижает точность объединения.

Для улучшения датафрейма, который мы потом будет использовать для машинного обучения, сделаем следующие действия:

1. Создадим признак 'date_difference', показывающий разницу между датой турнирного матча и датой начала турнира
2. Создадим два дополнительных признака:
 - KEEP_IN_PLACE_14: равен \$1\$, если разность дат 'date_difference' лежит на отрезке \$[0;14]\$\$, и \$0\$\$, если иначе. Данный признак пригодится для турниров Большого шлема
 - KEEP_IN_PLACE_7: равен \$1\$, если разность дат 'date_difference' лежит на отрезке \$[0;7]\$\$, и \$0\$\$, если иначе. Данный признак пригодится для остальных турниров

Таким образом созданные признаки проверяют, входит ли матч с конкретной датой в сроки проведения турнира.

3. Далее преобразуем датафрейм 'matches_bets' так, чтобы разность между датой матча и датой начала турнира лежала на отрезке \$[0;14]\$ и матч входил в серию турниров Большого шлема или же разность между датой матча и датой начала турнира лежала на отрезке \$[0;7]\$ и матч принадлежал любой серии турниров.
4. Отсортируем получившийся датафрейм по дате в порядке возрастания (от старых матчей к всё более новым) для более удобной работы с ним
5. Отобразим получившийся датафрейм, продемонстрировав несколько выбранных признаков

In [85]:

```
matches_bets['Date'].head()
```

Out [85]:

```
0    2001-02-19
1    2001-02-19
2    2006-06-14
3    2006-06-14
4    2006-10-18
Name: Date, dtype: datetime64[ns]
```

In [86]:

```
matches_bets['date_difference'] = matches_bets['Date'] - matches_bets['Tournament Date']

matches_bets['KEEP_IN_PLACE_14'] = matches_bets['date_difference'].apply(lambda
```

```

mbda x: 1 if x.days<=14 and x.days>=0 else 0)
matches_bets['KEEP_IN_PLACE_7'] = matches_bets['date_difference'].apply(lambda
bda x: 1 if x.days<=7 and x.days>=0 else 0)

matches_bets = matches_bets.loc[
    ((matches_bets['KEEP_IN_PLACE_14'] == 1) & (matches_bets['Series'] == 'Grand Slam')) | (matches_bets['KEEP_IN_PLACE_7'] == 1)
]

matches_bets.sort_values('Date', ascending=True, inplace=True, kind='quicksort', na_position='last')

matches_bets[['winner_name', 'score', 'loser_name', 'w_Prct1stIn', 'AvgW',
'AvgL', 'StdW', 'StdL',
'tourney_name', 'Date']].head()

```

Out [86]:

	winner_name	score	loser_name	w_Prct1stIn	AvgW	AvgL	StdW	StdL	tourn
7724	Michal Tabara	6-3 6-3	Fernando Gonzalez	0.666667	NaN	NaN	NaN	NaN	Chen
7761	Kristian Pless	6-1 7-5	Adrian Voinea	0.585714	1.9250	1.6825	0.086603	0.135000	Chen
7667	Bohdan Ulihrach	6-2 6-3	Marc Rosset	0.675000	2.4125	1.4200	0.201556	0.061644	Doha
7760	Peter Wessels	6-0 6-4	Mikhail Youzhny	0.600000	1.9750	1.6325	0.125831	0.076757	Chen
1646	Alberto Martin	3-6 7-6(5) 7-5	Christophe Rochus	0.556604	NaN	NaN	NaN	NaN	Adela

На всякий случай проверим, действительно ли выполняются условия, на основе которых мы создали новый датафрейм, посмотрев на несовпадающие даты матча и начала турнира.
Ответ: выполняются!

In [87]:

```

condition_check = matches_bets[matches_bets.date_difference != '0 days']
condition_check[['Date', 'Tournament Date', 'date_difference',
'KEEP_IN_PLACE_7', 'KEEP_IN_PLACE_14']].tail()

```

Out [87]:

	Date	Tournament Date	date_difference	KEEP_IN_PLACE_7	KEEP_IN_PLACE_14
57087	2016-11-18	2016-11-14	4 days	1	1
60094	2016-11-18	2016-11-14	4 days	1	1
67929	2016-11-19	2016-11-14	5 days	1	1

		Tournament Date	date_difference	KEEP_IN_PLACE_7	KEEP_IN_PLACE_14
65215	Date-11-19	2016-11-14	5 days	1	1
56931	2016-11-20	2016-11-14	6 days	1	1

In [88]:

```
matches_bets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29507 entries, 7724 to 56931
Columns: 190 entries, tourney_id to KEEP_IN_PLACE_7
dtypes: datetime64[ns](2), float64(140), int32(8), int64(9), object(30), timedelta64[ns](1)
memory usage: 42.1+ MB
```

Из последних строк датафрейма и информации о нём видно, что индексация строк сделана неправильно. Исправим:

In [89]:

```
matches_bets.index = range(len(matches_bets.index))

matches_bets[['winner_name', 'score', 'loser_name', 'w_Prct1stIn', 'AvgW',
'AvgL', 'StdW', 'StdL',
'tourney_name', 'Date', 'Tournament Date']].tail()
```

Out [89]:

	winner_name	score	loser_name	w_Prct1stIn	AvgW	AvgL	StdW	StdL	to
29502	Andy Murray	6-4 6-2	Stanislas Wawrinka	0.588235	1.293333	3.6525	0.015275	0.236696	Lc
29503	Marin Cilic	3-6 6-2 6-3	Kei Nishikori	0.545455	2.446667	1.5525	0.068069	0.047871	Lc
29504	Andy Murray	5-7 7-6(5) 7-6(9)	Milos Raonic	0.609589	1.230000	4.2525	0.026458	0.423901	Lc
29505	Novak Djokovic	6-1 6-1	Kei Nishikori	0.636364	1.230000	4.2400	0.017321	0.342734	Lc
29506	Andy Murray	6-3 6-4	Novak Djokovic	0.542373	2.633333	1.4825	0.125831	0.075000	Lc

Теперь с индексацией всё в порядке.

Проверим, в какой доле матчей нет средней ставки на победителя и проигравшего, что не позволит связывать обучение и букмекерские котировки.

In [90]:

```
matches_bets['AvgW'].isnull().sum() / len(matches_bets)
```

Out [90] :

```
0.025993831972074423
```

Около \$2,6%. Не так уже много, но мы всё же отбросим матчи без ставок, чтобы сравнение потом полученных вероятностей с ними было релевантным.

In [91] :

```
matches_bets = matches_bets[np.isfinite(matches_bets['AvgW'])] # оставили только матчи с известными ставками
```

In [92] :

```
matches_bets['AvgL'].isnull().sum() # пропали NaN не только у ставок на победителя, но и на проигравшего
```

Out [92] :

```
0
```

In [93] :

```
print('Число матчей в итоговом датафрейме: ', len(matches_bets))
```

```
Число матчей в итоговом датафрейме: 28740
```

А теперь сравним два, по идее, одинаковых столбца с информацией о типе покрытия корта, на котором проходил матч.

In [94] :

```
print ('matches - отсутствует информация о покрытии: ', matches['surface'].isnull().sum(),
      '\nall_bets - отсутствует информация о покрытии: ', all_bets['Surface'].isnull().sum())
```

```
matches - отсутствует информация о покрытии: 118
```

```
all_bets - отсутствует информация о покрытии: 0
```

In [95] :

```
matches_bets.head()
```

Out [95] :

	tourney_id	tourney_name	surface	draw_size	tourney_level	tourney_date	match_num	w
1	2001-891	Chennai	Hard	32	A	20010101	25	10
2	2001-451	Doha	Hard	32	A	20010101	18	10
3	2001-891	Chennai	Hard	32	A	20010101	24	10
5	2001-891	Chennai	Hard	32	A	20010101	23	10
6	2001-891	Chennai	Hard	32	A	20010101	22	10

5 rows × 190 columns

Видим, что признак 'Surface' из 'all_bets' описан более полно, следовательно, его и будем далее брать как описывающий покрытие корта.

Создание нового датафрейма

Со случайным выбором результата игрока для релевантного обучения

Теперь перед нами стоит задача создать столбцы с именами игроков так, чтобы из названия самих столбцов не было понятно, кто победитель, а кто проигравший - это нужно для дальнейшего обучения. Для этого проведём "жеребёвку", сгенерировав псевдослучайные числа: 0 или 1 и поместив их в новый столбец датафрейма 'Random'.

*Зафиксировав ядро на 2016 (np.random.seed(2016)) мы обеспечили себе одинаковые псевдослучайные числа при каждой загрузке кода.

In [96]:

```
np.random.seed(2016)
matches_bets['Random'] = np.random.randint(0, high=1,
size=len(matches_bets.index))
matches_bets['Random'].value_counts()
```

Out [96]:

```
1    14472
0    14268
Name: Random, dtype: int64
```

Теперь создадим столбцы 'Player 1' и 'Player 2', записав в столбец 'Player 1' имя победителя матча, если в столбце 'Random' стоит число 1, и имя проигравшего, если 0. Для столбца 'Player 2' проделаем обратное.

После этого создадим столбец 'Player 1 is a winner', в который запишем 1, если это правда, и 0, если это ложь. Это и будет в дальнейшем столбом ответов при обучении.

In [97]:

```
matches_bets.loc[matches_bets['Random'] == 1, 'Player 1'] = matches_bets['Winner']
matches_bets.loc[matches_bets['Random'] == 1, 'Player 2'] = matches_bets['Loser']

matches_bets.loc[matches_bets['Random'] == 0, 'Player 1'] = matches_bets['Loser']
matches_bets.loc[matches_bets['Random'] == 0, 'Player 2'] = matches_bets['Winner']

matches_bets.loc[matches_bets['Player 1'] == matches_bets['Winner'], 'P1 is a winner'] = 1
matches_bets.loc[matches_bets['Player 1'] == matches_bets['Loser'], 'P1 is a winner'] = 0

matches_bets[['Player 1', 'Winner', 'Loser', 'Player 2', 'P1 is a winner', 'Year']].head()
```

Out[97]:

	Player 1	Winner	Loser	Player 2	P1 is a winner	Year
1	Pless K.	Pless K.	Voinea A.	Voinea A.	1.0	2001
2	Rosset M.	Ulihrach B.	Rosset M.	Ulihrach B.	0.0	2001
3	Youzhny M.	Wessels P.	Youzhny M.	Wessels P.	0.0	2001
5	Stoliarov A.	Stoliarov A.	Vinciguerra A.	Vinciguerra A.	1.0	2001
6	Saulnier C.	Saulnier C.	Black B.	Black B.	1.0	2001

Далее переименуем статистические показатели и букмекерские котировки так, чтобы они относились либо к 1, либо ко 2 игроку, а не к победителю или проигравшему.

In [98]:

```
winner_loser_lst = ['name', 'rank', 'id', 'seed', 'entry', 'hand', 'ht', 'io
c', 'age', 'rank_points']
```

In [99]:

```
for col in winner_loser_lst:

    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P1_'+col] =
matches_bets['winner_'+col]
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P2_'+col] =
matches_bets['loser_'+col]

    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P1_'+col] =
matches_bets['loser_'+col]
    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P2_'+col] =
matches_bets['winner_'+col]

matches_bets[['P1 is a winner', 'winner_name', 'loser_name', 'winner_age',
'P1_name', 'P1_age', 'P2_name', 'P2_age']].head()
```

Out[99]:

	P1 is a winner	winner_name	loser_name	winner_age	P1_name	P1_age	P2_name	P2_age
1	1.0	Kristian Pless	Adrian Voinea	19.893224	Kristian Pless	19.893224	Adrian Voinea	26.406571
2	0.0	Bohdan Ulihrach	Marc Rosset	25.856263	Marc Rosset	30.151951	Bohdan Ulihrach	25.856263
3	0.0	Peter Wessels	Mikhail Youzhny	22.655715	Mikhail Youzhny	18.521561	Peter Wessels	22.655715
5	1.0	Andrei Stoliarov	Andreas Vinciguerra	23.978097	Andrei Stoliarov	23.978097	Andreas Vinciguerra	19.865845
6	1.0	Cyril Saulnier	Byron Black	25.379877	Cyril Saulnier	25.379877	Byron Black	31.238877

In [100]:

```
w_l_lst = ['sv_number', 'ServeAccuracy', 'ace', 'df', 'svpt', '1stIn',  
'2ndIn', '1stWon', '2ndWon', 'svptWon', 'PrctsvptWon',  
'ptWon', 'PrctptWon', 'SvGms', 'bpSaved', 'bpFaced', 'Prct1stIn',  
'Prct1stWon', 'Prct2ndWon', 'Prct_ace', 'Prct_df',  
'bpWon', '1stRetpt', '2ndRetpt', '1stRetptWon', '2ndRetptWon', 'I  
rct1stRetptWon', 'Prct2ndRetptWon', 'RetptWon',  
'PrctRetptWon']
```

In [101]:

```
for col in w_l_lst:  
  
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P1_'+col] =  
    matches_bets['w_'+col]  
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P2_'+col] =  
    matches_bets['l_'+col]  
  
    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P1_'+col] =  
    matches_bets['l_'+col]  
    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P2_'+col] =  
    matches_bets['w_'+col]  
  
matches_bets[['P1 is a winner', 'score', 'P1_name', 'P1_ServeAccuracy', 'P1  
_Prct1stIn',  
             'P1_Prct1stWon', 'P1_Prct_ace', 'P1_PrctsvptWon', 'P1_1stRetpt',  
             'P1_1stRetptWon', 'P1_Prct1stRetptWon']].head()
```

Out [101]:

	P1 is a winner	score	P1_name	P1_ServeAccuracy	P1_Prct1stIn	P1_Prct1stWon	P1_Prct_ace	P1_PrctsvptWon	P1_1stRetpt	P1_1stRetptWon	P1_Prct1stRetptWon
1	1.0	6-1 7-5	Kristian Pless	0.656566	0.585714	0.731707	0.085714	0.085714	0.085714	0.085714	0.085714
2	0.0	6-2 6-3	Marc Rosset	0.616279	0.491228	0.607143	0.035088	0.035088	0.035088	0.035088	0.035088
3	0.0	6-0 6-4	Mikhail Youzhny	0.676923	0.645833	0.548387	0.020833	0.020833	0.020833	0.020833	0.020833
5	1.0	6-1 6-3	Andrei Stoliarov	0.600000	0.437500	0.750000	0.093750	0.093750	0.093750	0.093750	0.093750
6	1.0	6-3 6-4	Cyril Saulnier	0.705882	0.606557	0.837838	0.163934	0.163934	0.163934	0.163934	0.163934

In [102]:

```
bets_lst = ['Avg', 'Std', 'Max', 'B&W', 'B365', 'CB', 'EX', 'GB', 'IW', 'LB',  
'PS', 'SB', 'SJ', 'UB']
```

In [103]:

```
for col in bets_lst:
```

```

    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P1_'+col] =
matches_bets[col+'W']
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P2_'+col] =
matches_bets[col+'L']

    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P1_'+col] =
matches_bets[col+'L']
    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P2_'+col] =
matches_bets[col+'W']

matches_bets[['Player 1', 'Player 2', 'P1_Avg', 'P2_Avg', 'P1_B365', 'P2_B365', 'Winner']].tail()

```

Out [103]:

	Player 1	Player 2	P1_Avg	P2_Avg	P1_B365	P2_B365	Winner
29502	Wawrinka S.	Murray A.	3.652500	1.293333	3.75	1.28	Murray A.
29503	Cilic M.	Nishikori K.	2.446667	1.552500	2.37	1.57	Cilic M.
29504	Murray A.	Raonic M.	1.230000	4.252500	1.20	4.50	Murray A.
29505	Nishikori K.	Djokovic N.	4.240000	1.230000	4.33	1.22	Djokovic N.
29506	Djokovic N.	Murray A.	1.482500	2.633333	1.53	2.50	Murray A.

In [104]:

```

for col in ['Pts', 'Rank', 'sets']:
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P1_'+col] =
matches_bets['W'+col]
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P2_'+col] =
matches_bets['L'+col]

for col in ['1', '2', '3', '4', '5']:
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P1_SetNº'+col] =
matches_bets['W'+col]
    matches_bets.loc[matches_bets['P1 is a winner'] == 1, 'P2_SetNº'+col] =
matches_bets['L'+col]

    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P1_SetNº'+col] =
matches_bets['L'+col]
    matches_bets.loc[matches_bets['P1 is a winner'] == 0, 'P2_SetNº'+col] =
matches_bets['W'+col]

extra_lst = ['Pts', 'Rank', 'SetNº1', 'SetNº2', 'SetNº3', 'SetNº4', 'SetNº5']

matches_bets[['P1 is a winner', 'P1_Pts', 'P1_Rank', 'Player 1', 'P1_SetNº1',
, 'P2_SetNº1', 'P1_SetNº2', 'P2_SetNº2',
'P1_SetNº3', 'P2_SetNº3']].tail()

```

Out [104]:

	P1 is a winner	P1_Pts	P1_Rank	Player 1	P1_SetNº1	P2_SetNº1	P1_SetNº2	P2_SetNº2	P1
29502	0.0	NaN	NaN	Wawrinka S.	4.0	6.0	2	6	Na

29503	1.0 P1 is a winner	3450.0	7.0	Cilic M.	3.0	6.0	6	2	6
29504	winner	P1_Pts 11185.0	P1_Rank 1.0	Player 1 Murray A.	P1_SetNo1 5.0	P2_SetNo1 7.0	P1_SetNo2 7	P2_SetNo2 6	P1 7
29505	0.0	NaN	NaN	Nishikori K.	1.0	6.0	1	6	Na
29506	0.0	NaN	NaN	Djokovic N.	3.0	6.0	4	6	Na

In [105]:

```
matches_bets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 28740 entries, 1 to 29506
Columns: 318 entries, tourney_id to P2_SetNo5
dtypes: datetime64[ns](2), float64(250), int32(9), int64(9), object(47), timedelta64[ns](1)
memory usage: 69.0+ MB
```

Выгружим датафрейм в формате csv:

In [106]:

```
matches_bets.to_csv('matches_bets.csv')
```

Создание средних показателей игрока на момент матча

In [107]:

```
player_stats = winner_loser_lst + w_l_lst + bets_lst + extra_lst # создаём
# список со всеми имеющимися показателями игрока
```

Далее сделаем MultiIndex, чтобы было удобнее присоединять статистические данных за прошлые матчи.

In [108]:

```
d1_lst = ['P1 is a winner']

for feature in player_stats:
    for player in ['P1_', 'P2_']:
        d1_lst.append(player + feature)
```

In [109]:

```
mux_lst = [('P1', ' is a winner')]

for stat in player_stats:
    for player in ['P1', 'P2',]:
        mux_lst.append((player, stat))

# не беру Comment, minutes, score
```

```
mx_lst[:2]
```

```
Out[109]:
```

```
[('P1', ' is a winner'), ('P1', 'name')]
```

```
In [110]:
```

```
mx = pd.MultiIndex.from_tuples(mx_lst)
```

```
In [111]:
```

```
d1 = pd.DataFrame(matches_bets[d1_lst].values, matches_bets[d1_lst].index,
mx).sort_index(1)
```

```
In [112]:
```

```
d1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 28740 entries, 1 to 29506
Columns: 123 entries, (P1, is a winner) to (P2, svptWon)
dtypes: object(123)
memory usage: 27.2+ MB
```

```
In [113]:
```

```
mx_lst = []
```

```
for gen in ['Tournament Year', 'Tournament Month', 'Tournament Day',
'draw_size', 'match_num',
    'ATP', 'Best of', 'Court', 'Year', 'Month', 'Day', 'Round', 'Series',
'Surface', 'Tournament', 'tourney_level']:
    mx_lst.append((General Info, gen))
```

```
# не беру Comment, minutes, score, surface, tourney_name, best_of,
tourney_id - для некоторых есть альтернатива,
# другие неинформативны
```

```
In [114]:
```

```
d3_lst = []
```

```
for gen in ['Tournament Year', 'Tournament Month', 'Tournament Day',
'draw_size', 'match_num',
    'ATP', 'Best of', 'Court', 'Year', 'Month', 'Day', 'Round', 'Series',
'Surface', 'Tournament', 'tourney_level']:
    d3_lst.append(gen)
```

```
#убрал Date, вставил Год, месяц, день
```

```
In [115]:
```

```
mx = pd.MultiIndex.from_tuples(mx_lst)
mx
```

```
Out[115]:
```

```
MultiIndex(levels=[['General Info'], ['ATP', 'Best of', 'Court', 'Day', 'Month',
'Round', 'Series', 'Surface', 'Tournament', 'Tournament Day', 'Tourna
```

```
ment Month', 'Tournament Year', 'Year', 'draw_size', 'match_num', 'tourney_level']],  
    labels=[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [11, 10  
, 9, 13, 14, 0, 1, 2, 12, 4, 3, 5, 6, 7, 8, 15]])
```

In [116]:

```
d3 = pd.DataFrame(matches_bets[d3_lst].values, matches_bets[d3_lst].index,  
mx).sort_index(1)  
#создание датафрейма с общей информацией о матче  
  
d3.head()
```

Out [116]:

General Info										
	ATP	Best of	Court	Day	Month	Round	Series	Surface	Tournament	Tournament Day
1	2	3	Outdoor	1	1	Quarterfinals	International	Hard	TATA Open	1
2	3	3	Outdoor	1	1	2nd Round	International	Hard	Qatar Open	1
3	2	3	Outdoor	1	1	2nd Round	International	Hard	TATA Open	1
5	2	3	Outdoor	1	1	2nd Round	International	Hard	TATA Open	1
6	2	3	Outdoor	1	1	2nd Round	International	Hard	TATA Open	1

In [117]:

```
d1 = pd.concat([d1, d3], axis=1) # объединение информации об игроках и о матче  
  
d1.head()
```

Out [117]:

	P1									
	is a winner	1stIn	1stRetpt	1stRetptWon	1stWon	2ndIn	2ndRetpt	2ndRetptWon	2ndWon	Avg
1	1	41	28	14	30	24	22	12	13	1.9
2	0	28	27	4	17	25	11	4	10	1.4
3	0	31	30	7	17	13	19	9	5	1.6
5	1	28	28	11	21	32	18	10	22	2.4
6	1	37	26	10	31	23	18	8	9	2.3

5 rows × 139 columns

In [118]:

```
d_ans = d1['P1'][['is a winner']] # записываем ответы в отдельный вектор  
d1 = d1.drop(['P1', 'is a winner'], 1) # удаляем ответы из датафрейма с да
```

```
d1.head()
```

Out [118]:

	P1										
	1stIn	1stRetpt	1stRetptWon	1stWon	2ndIn	2ndRetpt	2ndRetptWon	2ndWon	Avg	B&T	
1	41	28	14	30	24	22	12	13	1.925	Nan	
2	28	27	4	17	25	11	4	10	1.42	Nan	
3	31	30	7	17	13	19	9	5	1.6325	Nan	
5	28	28	11	21	32	18	10	22	2.45	Nan	
6	37	26	10	31	23	18	8	9	2.35	Nan	

5 rows × 138 columns



In [119]:

```
d_ans.tail()
```

Out [119]:

```
29502      0
29503      1
29504      1
29505      0
29506      0
Name: is a winner, dtype: object
```

In [120]:

```
d1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 28740 entries, 1 to 29506
Columns: 138 entries, (P1, 1stIn) to (General Info, tourney_level)
dtypes: object(138)
memory usage: 30.5+ MB
```

В датафрейме много признаков, являющихся non-null objects, хотя информацию они несут числовую, что может помешать нам при машинном обучении. Создадим список из таких признаков и преобразуем их тип числовой:

In [121]:

```
must_be_numeric_obj = [('P1', 'rank'), ('P1', 'id'), ('P1', 'seed'), ('P1', 'ht'),
('P1', 'age'), ('P1', 'rank_points'),
('P2', 'rank'), ('P2', 'id'), ('P2', 'seed'), ('P2', 'ht'),
('P2', 'age'), ('P2', 'rank_points'),
('General Info', 'Tournament Year'), ('General Info',
'Tournament Month'),
('General Info', 'Tournament Day'), ('General Info',
'draw_size'), ('General Info', 'match_num'),
('General Info', 'ATP'), ('General Info', 'Best of'), ('General
Info', 'Tourney_Level'))]
```

```
    ('Info', 'Year'), ('General Info', 'Month'),
    ('General Info', 'Day'))]
```

```
d1[must_be_numeric_obj] = d1[must_be_numeric_obj].apply(pd.to_numeric)
```

```
In [122]:
```

```
d1.to_csv('df.csv', encoding='utf-8')
```

```
In [123]:
```

```
mix = pd.MultiIndex.from_tuples(mix_lst) # создадим 2 списка с номером игрока и его статистикой
```

Подготовка данных к обучению

```
In [124]:
```

```
prev_stats = w_1_lst #создадим такой же датафрейм с новым названием, чтобы далее было понятно,  
# что мы работаем именно с данными, прошлые значения которых несут в себе важную информацию
```

```
In [125]:
```

```
f = lambda x: x.expanding().mean().shift()  
  
g = d1.stack(0).groupby('name')[prev_stats]  
d2 = g.apply(f).unstack().swaplevel(0, 1, 1)  
d1 = d1.join(d2, rsuffix=' Mean Previous Stats')  
d1.head()
```

```
Out[125]:
```

	P1										
	1stIn	1stRetpt	1stRetptWon	1stWon	2ndIn	2ndRetpt	2ndRetptWon	2ndWon	Avg	B&'	
1	41	28	14	30	24	22	12	13	1.925	NaN	
2	28	27	4	17	25	11	4	10	1.42	NaN	
3	31	30	7	17	13	19	9	5	1.6325	NaN	
5	28	28	11	21	32	18	10	22	2.45	NaN	
6	37	26	10	31	23	18	8	9	2.35	NaN	

5 rows × 228 columns

Сразу найдём вероятность 1 игрока выиграть матч исходя их ставок:

```
In [126]:
```

```
avg_P1_wprb = 1 - ((d1['P1']['Avg']) / (d1['P1']['Avg'] + d1['P2']['Avg'])).  
#или  
#d1['P1'].apply(lambda x: 1 - (x / (x + d1['P2'][x])))
```

```
apply(pd.to_numeric, errors='coerce')
avg_P1_wprb[:3]
```

Out [126]:

```
1    0.466389
2    0.629485
3    0.547471
Name: Avg, dtype: float64
```

Далее создадим датафрейм с колонкой из псевдослучайных чисел, чтобы потом из него сделать датафрейм для обучения:

In [127]:

```
learn = pd.DataFrame(np.random.randn(len(d1.index), 1))
learn.tail()
```

Out [127]:

	0
28735	-0.506323
28736	0.570500
28737	-0.702044
28738	0.533999
28739	0.000590

Теперь добавим в датафрейм "learn" признаки, которые будут использоваться при обучении:

In [128]:

```
for player in ['P1', 'P2']:
    for feature in ['rank', 'id', 'seed', 'entry', 'hand', 'ht', 'ioc', 'age',
', 'rank_points']:
        learn[player + ' ' + feature] = d1[(player, feature)]
# не взял 'name', так как этот признак не считаю важным: многие люди как раз и делают ставки, исходя только из имени человека

for player_mean in ['P1 Mean Previous Stats', 'P2 Mean Previous Stats']:
    for feature in w_l_lst:
        learn[player_mean + ' ' + feature] = d1[(player_mean, feature)]

for gen_info in ['Tournament Year', 'Tournament Month', 'Tournament Day',
'draw_size', 'match_num', 'ATP', 'Best of', 'Court',
'Year', 'Month', 'Day', 'Round', 'Series', 'Surface', 'Tournament']:
    learn[gen_info] = d1[('General Info', gen_info)]
```

Удалим ненужную колонку с псевдослучайными числами:

In [129]:

```
del learn[0]
```

А теперь воспользуемся one-hot кодированием и сделаем из текстовых признаков бинарные. Опять же, это делается для корректного машинного обучения.

In [130]:

```
learn = pd.get_dummies(learn, columns=['P1_entry', 'P1_hand', 'P1_ioc',
                                         'P2_entry', 'P2_hand', 'P2_ioc',
                                         'Court', 'Tournament', 'Round', 'Series',
                                         'Surface'])
```

In [131]:

```
learn.tail()
```

Out [131]:

	P1_rank	P1_id	P1_seed	P1_ht	P1_age	P1_rank_points	P2_rank	P2_id	P2_
28735	67.0	105357.0	NaN	183.0	27.036277	770.0	2.0	104918.0	2.0
28736	123.0	105041.0	NaN	185.0	28.648871	501.0	13.0	105227.0	9.0
28737	29.0	105449.0	NaN	188.0	26.507871	1385.0	37.0	105777.0	NaN
28738	9.0	104607.0	10.0	196.0	30.776181	2950.0	28.0	100644.0	24.0
28739	21.0	103852.0	22.0	188.0	34.767967	1630.0	18.0	106401.0	15.0

5 rows × 448 columns

In [132]:

```
learn['P2_Mean_Previous_Stats_ServeAccuracy'].tail()
```

Out [132]:

```
28735    0.686631
28736    0.672177
28737    0.697773
28738    0.681666
28739    0.711765
Name: P2_Mean_Previous_Stats_ServeAccuracy, dtype: float64
```

Теперь компьютер точно не будет ругаться на наши данные!

Сохраним наши данные в формате csv:

In [133]:

```
learn.to_csv('df.csv', encoding='utf-8')
```

In [134]:

```
d_ans.to_csv('df_ans.csv')
```

Заполним все пропуски медианой, чтобы Random Forest Classifier не ругался:

```
In [126]:
```

```
learn = learn.fillna(learn.median())
```

Обучение

```
In [127]:
```

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score, roc_auc_score, make_scorer, mean_absolute_error
from sklearn.model_selection import train_test_split

skf = StratifiedKFold(n_splits = 4, shuffle=True, random_state=2016)
```

```
In [128]:
```

```
X_train, X_test, y_train, y_test = train_test_split(learn, d_ans, test_size=0.2)
```

```
In [129]:
```

```
y_train_S6 = np.asarray(y_train, dtype="|S6") # иначе ругается обучающее дерево
y_test_S6 = np.asarray(y_test, dtype="|S6")
```

RandomForestClassifier

```
In [130]:
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=310, n_jobs=-1, oob_score=True, random_state=2016)
```

Воспользуемся GridSearchCV, чтобы подобрать оптимальные гиперпараметры, и посмотрим на качество на обучающей выборке.

```
In [131]:
```

```
rfc_param = {'max_depth': np.arange(1, 20), 'criterion': ['gini', 'entropy']}
gs_rfc = GridSearchCV(rfc, rfc_param, scoring='accuracy')
gs_rfc.fit(X_train, y_train_S6)
```

```
Out[131]:
```

```
GridSearchCV(cv=None, error_score='raise',
            estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
            criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=310, n_jobs=-1, oob_score=True, random_state=2016,
            verbose=0, warm_start=False),
            fit_params={}, iid=True, n_jobs=1,
```

```
param_grid={'criterion': ['gini', 'entropy'], 'max_depth': array([ 1
,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19])},
pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
scoring='accuracy', verbose=0)
```

In [132]:

```
print('GridSearchCV RF accuracy = ', accuracy_score(y_test_S6, gs_rfc.predict(X_test)))
```

GridSearchCV RF accuracy = 0.491127348643

In [133]:

```
gs_rfc.best_params_
```

Out [133]:

```
{'criterion': 'entropy', 'max_depth': 13}
```

Присвоим лучшие гиперпараметры переменным:

In [146]:

```
best_criterion = gs_rfc.best_params_.get('criterion')
best_max_depth = gs_rfc.best_params_.get('max_depth')
```

Далее посмотрим, как менялось качество при использовании обоих критериев с увеличением максимальной глубины дерева.

In [134]:

```
crit = np.array(gs_rfc.cv_results_['param_criterion'])
depth = np.array(gs_rfc.cv_results_['param_max_depth'])
score = np.array(gs_rfc.cv_results_['mean_test_score'])
```

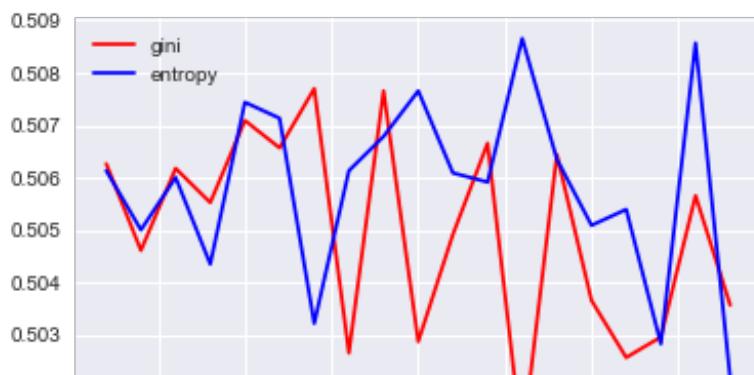
In [135]:

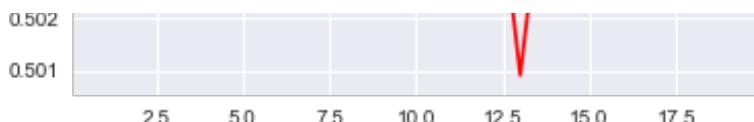
```
idx = crit == 'gini'
plt.plot(depth[idx], score[idx], c='r')
idx = crit == 'entropy'
plt.plot(depth[idx], score[idx], c='b')

plt.legend(['gini', 'entropy'])
```

Out [135]:

```
<matplotlib.legend.Legend at 0x1f15789dda0>
```





А теперь обучим нашу модель, воспользовавшись оптимальными гиперпараметрами.

In [136]:

```
best_rfc = RandomForestClassifier(n_estimators=310,
criterion=best_criterion, max_depth=best_max_depth, n_jobs=-1,
random_state=2016)
best_rfc.fit(X_train, y_train_S6)
y_pred = best_rfc.predict(X_test)

print('RandomForestClassifier best accuracy = ', accuracy_score(y_test_S6,
y_pred))
```

RandomForestClassifier best accuracy = 0.504523312457

In [152]:

```
y_pred_proba = best_rfc.predict_proba(X_test)
print ('RandomForestClassifier best MAE (predicting probabilities) = ', mean_absolute_error(y_test, y_pred_proba[:, 1]))
```

RandomForestClassifier best MAE (predicting probabilities) = 0.500202406609

К сожалению, качество вышло плохим, что вызывает сомнения в возможности хорошего предсказания с помощью имеющихся данных. Но далее мы еще попробуем XGBoost Classifier: может, он покажет себя лучше.

А пока посмотрим, какие признаки при обучении Random Forest Classifier посчитал наиболее важными.

In [138]:

```
features_rfc = pd.Series(best_rfc.feature_importances_, index=X_train.columns)
features_rfc.sort(ascending=False)
best_features_rfc = features_rfc[:10]
```

In [139]:

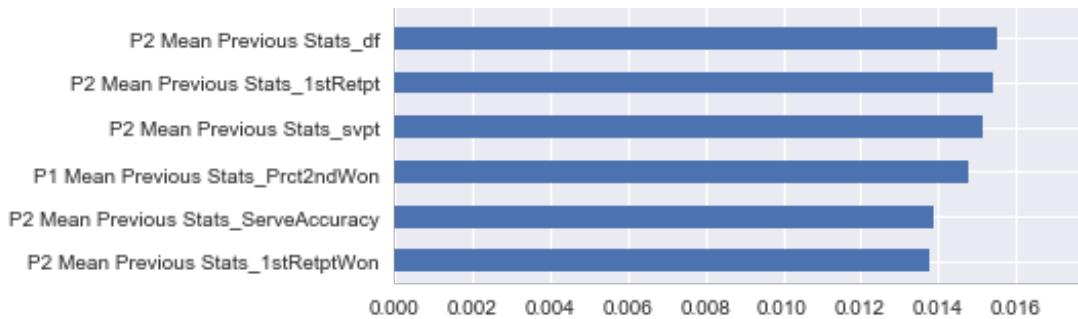
```
best_features_rfc = pd.DataFrame(best_features_rfc[::-1])
best_features_rfc.plot.barh()

plt.legend('')
plt.title('Важность признаков для предсказания (RandomForestClassifier)')
```

Out [139]:

<matplotlib.text.Text at 0x1f100239208>





Пятерка самых важных признаков для Random Forest Classifier выглядит следующим образом:

1. Возраст игрока 1
2. Среднее число сыгранных очков на приёме 2-ой подачи за матч игрока 2
3. Средний процент двойных ошибок за матч игрока 2
4. Средний процент выигранных очков на приёме 1-ой подачи за матч игрока 1
5. Среднее число двойных ошибок за матч игрока 2

Попробуем обучить модель на только на 10 самых важных признаках: вдруг это улучшит качество?

In [155]:

```
bf = ['P1_age', 'P1_Mean Previous Stats_2ndRetpt', 'P2_Mean Previous Stats_Prct_df',
      'P1_Mean Previous Stats_1stRetptWon', 'P2_Mean Previous Stats_df',
      'P2_Mean Previous Stats_1stRetpt',
      'P2_Mean Previous Stats_svpt', 'P1_Mean Previous Stats_Prct2ndWon',
      'P2_Mean Previous Stats_ServeAccuracy',
      'P2_Mean Previous Stats_1stRetptWon']
```

In [156]:

```
rfc.fit(X_train[bf], y_train_S6)
y_pred_new = rfc.predict(X_test[bf])
```

In [159]:

```
print ('RandomForestClassifier accuracy (usage of only the most important features) = ', accuracy_score(y_test_S6, y_pred_new))
```

```
RandomForestClassifier accuracy (usage of only the most important features)
= 0.494432846207
```

К сожалению, качество осталось примерно таким же.

Примечание: предсказания, полученные с помощью XGBoost Classifier, дали аналогичное качества. Из этого можно сделать вывод, что дело не в пропущенных данных, так как данный классификатор умеет работать с NaN.

Точность прогнозов исходя из ставок

А теперь узнаем качество предсказаний, если бы мы всегда говорили, что матч выиграет игрок, ставка на которого меньше.

```
In [163]:
```

```
y_test_df = pd.DataFrame(y_test)
y_test_df['indexes'] = y_test_df.index
indexes = list(y_test_df['indexes'])
```

```
In [164]:
```

```
avg_P1_wprb = avg_P1_wprb[indexes]
```

```
In [173]:
```

```
print('MAE предсказаний вероятностей букмекерского рынка = ', mean_absolute_error(y_test, avg_P1_wprb))
```

```
MAE предсказаний вероятностей букмекерского рынка = 0.311238691719
```

```
In [167]:
```

```
avg_P1_wprb = [np.round(item) for item in avg_P1_wprb] # округляем для грубого предсказания победителя
```

```
In [168]:
```

```
avg_P1_wprb = [int(item) for item in avg_P1_wprb]
avg_P1_wprb[:3]
```

```
Out[168]:
```

```
[0, 0, 1]
```

```
In [169]:
```

```
y_test = list(y_test)
```

```
In [175]:
```

```
print('Доля правильных предсказаний букмекерским рынком исходов матчей = ',
accuracy_score(y_test, avg_P1_wprb))
```

```
Доля правильных предсказаний букмекерским рынком исходов матчей = 0.688761308281
```

По качеству предсказаний рынок обогнал компьютер.

Итоги работы

К сожалению, наши модели (XGBoost см. далее) не обогнали по точности наивное предсказание о том, что фаворит согласно ставкам в букмекерских конторах выиграет матч. Возможно, им просто не хватило данных, а может, есть такие данные типа психологического состояния и личных взаимоотношений игроков, которые сложно оценить, и поэтому тяжело использовать при машинном обучении, хотя они могут играть значительную роль в определении исхода матча. При этом люди, делающие ставки, могут как-то учесть эти факторы и, следовательно, букмекерские ставки показывают себя лучшим "предсказателем", чем алгоритмы машинного обучения.

Но не стоит забывать о том, что нам удалось:

1. Собрать большое количество данных о теннисных матчах и обработать их
2. Визуализировать эти данные
3. Найти интересные взаимосвязи, которые могут пригодиться для построения стратегий при игре на ставках

Вполне возможно, что результаты этой работы могут не только помочь выиграть на игре на букмекерском рынке, но и обратить внимание теннисистов и их тренеров на факторы, наиболее сильно отражающие разницу между победителем и проигравшим.

Дополнительно: обучение с помощью XGBoost

Попробуем обучить модель с помощью XGBoost, воспользовавшись ранее сформированными датафреймами. Здесь не возникнет проблемы с заполнением пропущенных данных: XGBoost сам с ними справится.

In [35]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [16]:

```
df_ans = pd.read_csv('df_ans.csv')
```

In [17]:

```
df = pd.read_csv('Desktop/df.csv')
```

In [18]:

```
df_ans = np.append(1, df_ans['1.0'])
```

Уберём лишний столбец из датафрейма с признаками:

In [19]:

```
del df['Unnamed: 0']
```

In [20]:

```
df_ans
```

Out[20]:

```
array([ 1.,  0.,  0., ...,  1.,  0.,  0.])
```

In [21]:

```
import xgboost as xgb
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.cross_validation import train_test_split
```

In [22]:

```
seed=2016
```

In [23]:

```
X_train, X_test, y_train, y_test = train_test_split(df, df_ans, test_size=0.2)
```

Обучим модель:

In [24] :

```
model = xgb.XGBClassifier()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
predictions_proba = model.predict_proba(X_test)
```

In [25] :

```
score_auc = roc_auc_score(y_test, predictions_proba[:,1])
score_acc = accuracy_score(y_test, predictions)
```

Посмотрим на качество предсказания вероятностей (площадь под ROC-AUC кривой):

In [26] :

```
score_auc
```

Out [26] :

0.49845554048818286

К сожалению, показатели качества опять находятся на низком уровне: значит, дело точно не в пропущенных данных. Что ж, это тоже результат!

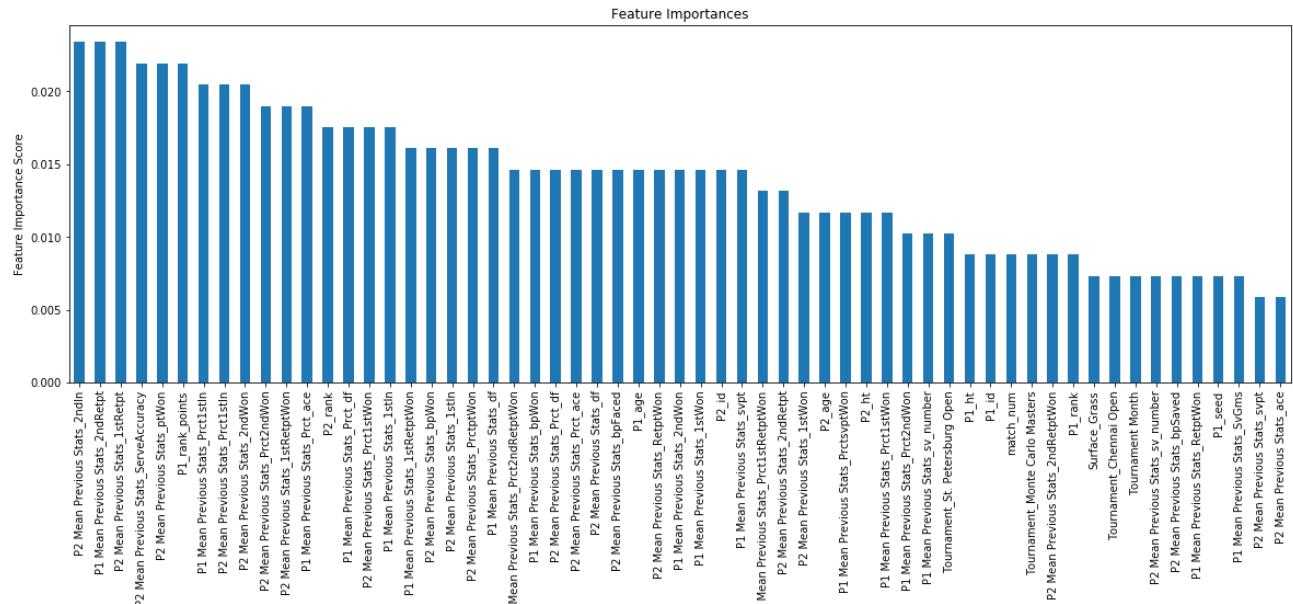
Напоследок узнаем, какие признаки XGBoost посчитал наиболее важными:

In [37] :

```
plt.figure(figsize=(20,6))
feat_imp = pd.Series(model.feature_importances_,
X_train.columns).sort_values(ascending=False) [1:60]
feat_imp.plot(kind='bar', title='Feature Importances')
plt.ylabel('Feature Importance Score')
```

Out [37] :

<matplotlib.text.Text at 0x1149f2b00>



Пятерка самых важных признаков для XGBoost Classifier выглядит следующим образом:

1. Точность 2-ой подачи игрока 2
2. Среднее число сыгранных очков на приёме 2-ой подачи за матч игрока 1
3. Среднее число сыгранных очков на приёме 1-ой подачи за матч игрока 2
4. Точность подачи игрока 2
5. Среднее число выигранных очков за матч игрока 2