Project Report

Introduction:

With the advent of event tracking systems and wearable devices, teams in major sporting events like- NFL, NBA, EPL, IPL- to name a few, increasingly rely on analytical methods for assessment and decision making. Data Visualization is an integral part of the process, and we often see matches and player performances illustrated as visualizations during analysis and discussions.

Drawing inspiration from these recent advances and applications, I have attempted to apply the visualization techniques learnt in class for analyzing the data in one of my favorite sporting events, the Association of Tennis Professional (ATP) tours.

Like any other sport, there are common trends in Tennis matches, like- the dominance of players on a particular playing surface, the effect of having a strong serve in a match, the correlation of players performance across playing surfaces, consistency and many more. I have attempted to analyze a few of them in this project.

Data Sources:

For my analysis, I have used the following resources:

- Association of Tennis Professionals (ATP) is the governing body of men's professional tennis- All the grand slams, Masters 1000, 500, 250 and more professional tennis tournaments fall under the purview of ATP. The official website of ATP contains much of the data relevant to the matches and players involved in the tours. I have scraped this website using the selenium web-driver to extract data of current top 100 ATP players.
- For getting the historical data of matches played in the ATP tours, I have used the data from GitHub repo- https://github.com/JeffSackmann/tennis_atp. This repository is known for its comprehensive collection of data in professional tennis matches starting from year 1968. It is often cited in Kaggle kernels and literature.

Research Questions:

In this project, I will first analyze the consistency of players over time through efficient visualizations, then I will compare the serve and return abilities of players through efficient visualizations. Further, I will delve deeper into the factors affecting player's ability to serve. Finally, I will analyze the correlation of player's performance across surfaces.

Motivation:

Consistency: All the major events in Tennis like the Grand Slams, ATP Masters 1000, 500, 250 etc. are knockout tournaments. So, its pivotal for these top tennis players to maintain consistency not just through each individual game of a tournament, but throughout the season. So, it would be interesting to analyze the consistency of players through efficient visualizations.

Serve Analysis: Serve is an important aspect of a player's game. Holding or breaking the serve at crucial points determines a player's resolve and can lead to a mental advantage over an opponent. Further, a player's serve depends on several aspects. Two such important aspects are the playing surface and the height of the player. It is generally noted that the clay is slower surface and hence the chances of hitting an ace on clay is much low. Also, taller players have higher chances of hitting aces. John Isner is one of the tallest players in the tour and his serving capabilities is the most coveted attribute by other tennis players. So, it would be interesting to analyze the factors affecting serve through empirical analysis.

Correlation of Performance across surfaces: The playing conditions on a hard-court are a bargain between grass and clay, like the speed of the ball on hard courts is slightly less than grass courts but more than clay courts, among other things. So, generally players who are dominant on hard courts fare well on other two courts. But players who are fully-dominant on clay/grass courts do not necessarily dominate on other courts. So, it would be interesting to justify this notion through empirical analysis.

Methodology:

Consistency: Consistency is best reflected by a player's ranking throughout his career. The official website of ATP contains the career rankings of players since they turned Pro. I have scraped this data for the Top 100 ATP players using Selenium in python. The data contains the rankings of the player after each tournament. Using this data, I have created an interactive time series plot in Plotly to illustrate the rankings. The final plot was generated after several iterations and analysis and it illustrates the median rankings of a player over time in each calendar year, since he turned pro. I have created a similar plot to illustrate the evolution of player through career rankings over time.

Serve Analysis: The server and return capabilities of players are evaluated on the following metrics:

- Serve Stats - 1st Serve, 1st Serve Points Won, 2nd Serve Points Won, Break Points Faced, Break Points Saved, Service Games Played, Service Games Won, Total Service Points Won.
- Return Stats - 1st Serve Return Points Won, 2nd Serve Return Points Won, Break Points Opportunities, Break Points Converted, Return Games Played, Return Games Won, Return Points Won, Total Points Won.

To compare two players on these metrics, I have created two Radar plots. Radar plots are a popular alternative to parallel coordinate plots for multivariate visualization and are common in sports analytics.

To analyze the impact of playing surface and player's height on server, I have used the data of ATP matches from 1968-2020.The data is available in the GitHub repo-

https://github.com/JeffSackmann/tennis_atp.

The data is segregated into years. So, I combined the data of all the years from 1968-2020 into a single data frame. The data frame has more than 175k records.

To analyze the impact of surface on hitting an Ace, I have calculated the Total Number of Aces for each match and then selected only matches in Grand Slams tournaments. Then I grouped the data such that I would get the total number of aces on each surface. I then calculated the number of Aces per match on each surface.

I also verified the impact of height on hitting an Ace. For this I created a data frame of players involved in the matches. I included their heights and total number of Aces in all the matches they played. Here too, I considered the total number of aces per match to get a fair estimation of the relationship with height. I plotted the data and the linear trendline as an interactive scatter plot.

Correlation of Performance across surfaces: As mentioned before, the performance of players on hard courts relates to performance of players on grass courts. For this analysis, I will be calculating the win-index of players across each surface.

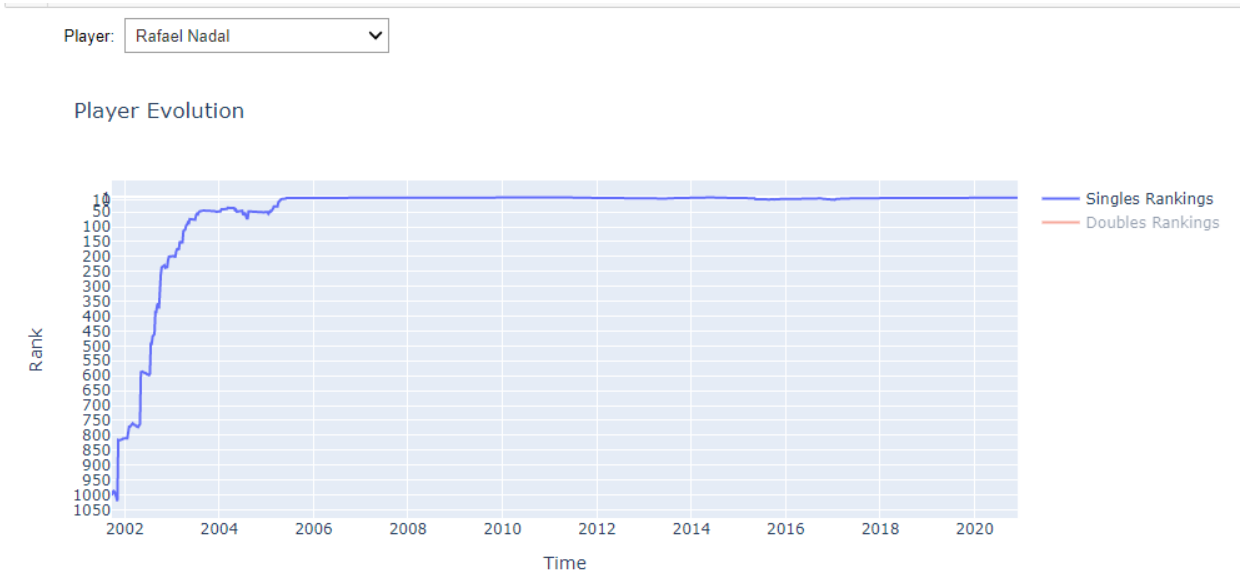
Win-index=(number of wins) / (number of wins + number of losses)

After calculating the win-index, I generated an annotated heatmap illustrating the correlation between win-index on each surface.

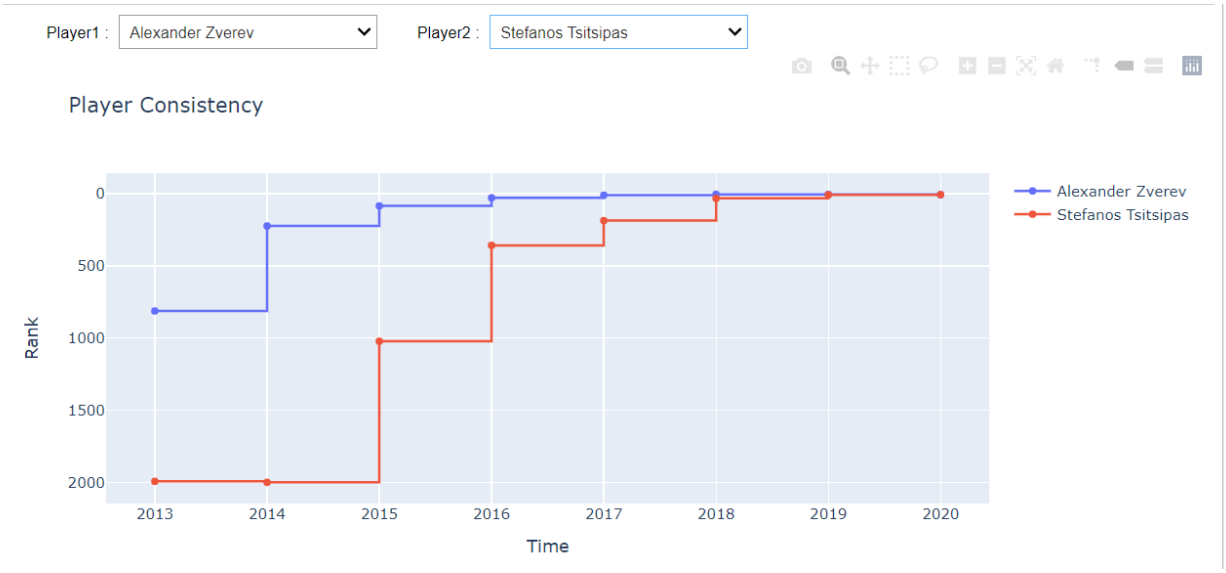
The correlation showed a positive and relatively stronger relationship between performance on grass courts and hard court. To see if I can visualize the positive correlation between performance of players on hard courts and grass courts, I clustered the players based on the win-index using the k-means clustering algorithm. I then plotted a 3-D plot to illustrate the relationship in performance.

Results and Discussion:

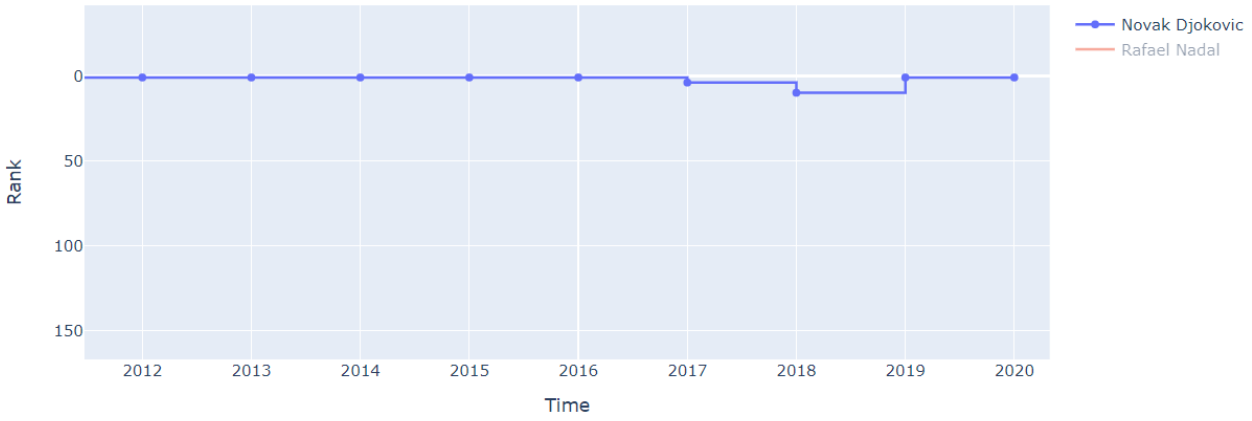
Consistency: In my initial attempt to visualize consistency through career rankings of a player, I used the ranking after each tournament. The plot looked like this-



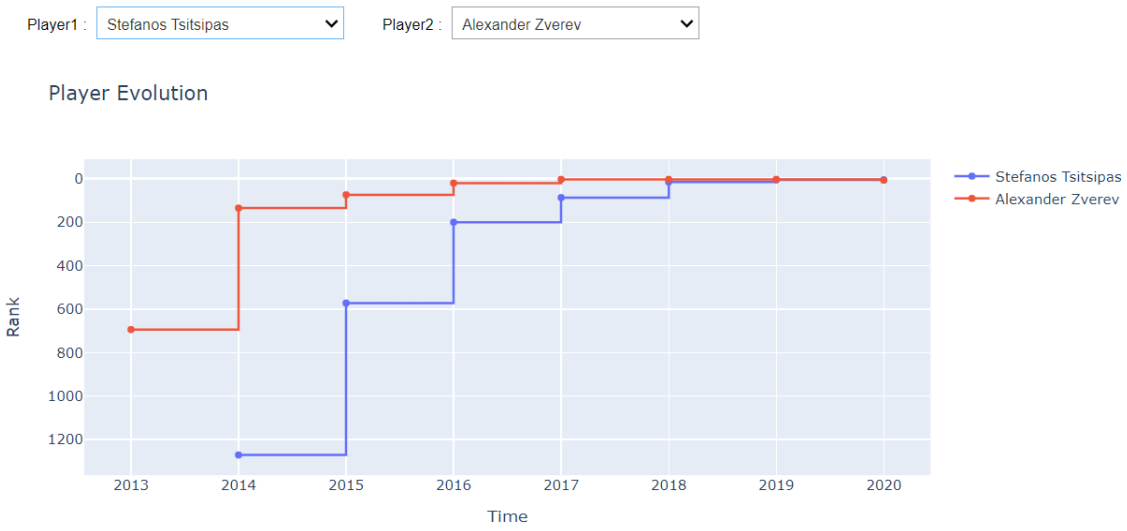
Although this plot provides a more granular view of a player's ranking, it is ugly and kind of misses the point. So, instead of using the ranking after each tournament, I decided to use the median rank of a player in a calendar year. I also added another drop down to the interactive plot. Now, the user can choose two players and their evolution or consistency. Now the plot looks like this-



The user can also choose to view only one player, by clicking the legend of the other player on the right. Further, a user can also a particular region of the plot by selecting it. On mouse hover, the plot prompts the ranking of the player at the time reflected on the x-axis. In my analysis, I have witnessed that once a player reaches the top 10s, he is usually consistent. Barring some exceptions, like when injuries hinder a player's performance. For example- Novak Djokovic, current world number 1, who usually dominated most of the matches and defended most of the titles he won, suffered injured and recovery from mid-2016. His ranking fell from median rank 1 in 2016 to 10 in 2018.



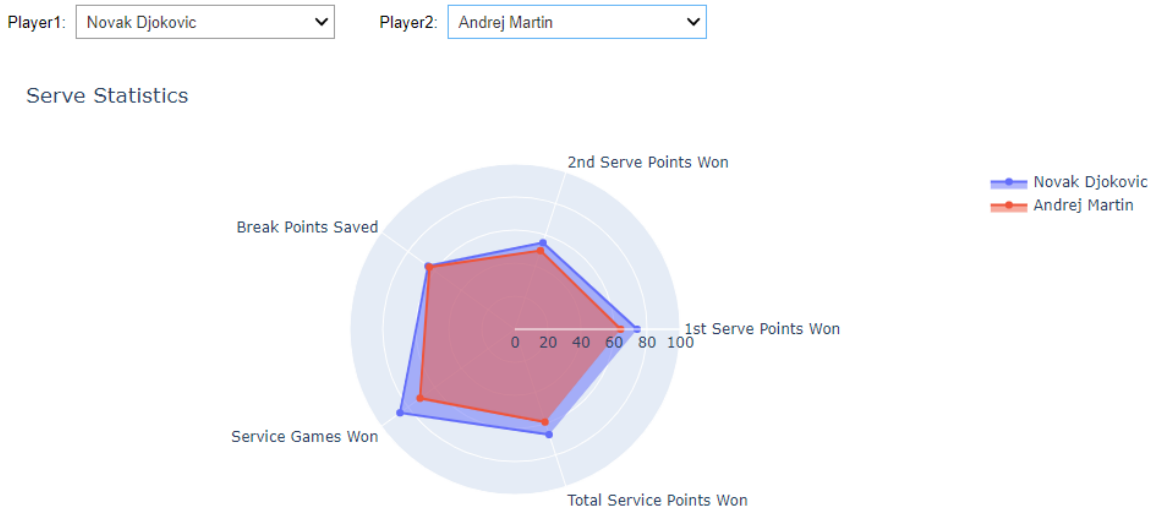
The plot illustrating the evolution of players looks like this-



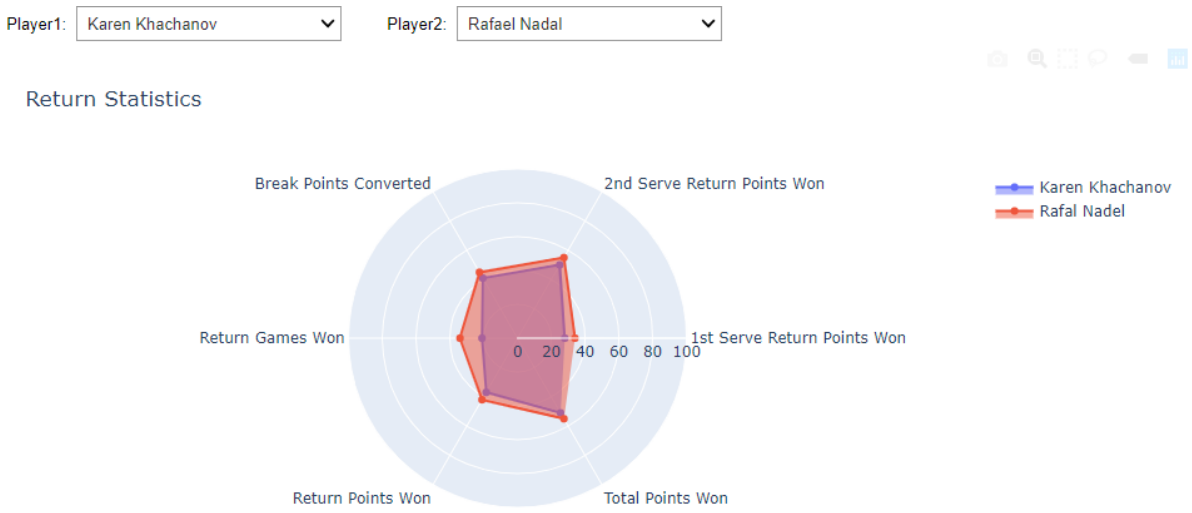
The above plot has the same properties as the previous plot for consistency.

Serve Analysis: I have created the following two Radar plots to compare the serve and return statistics of players:

Serve Statistics:

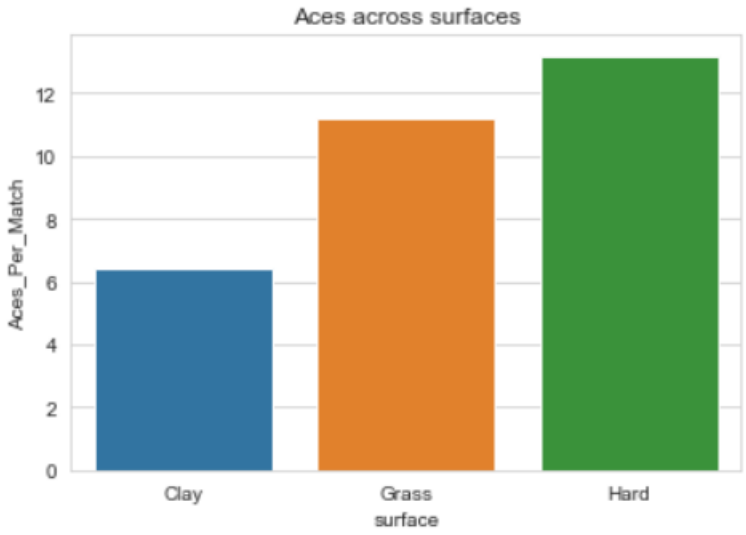


Return Statistics:



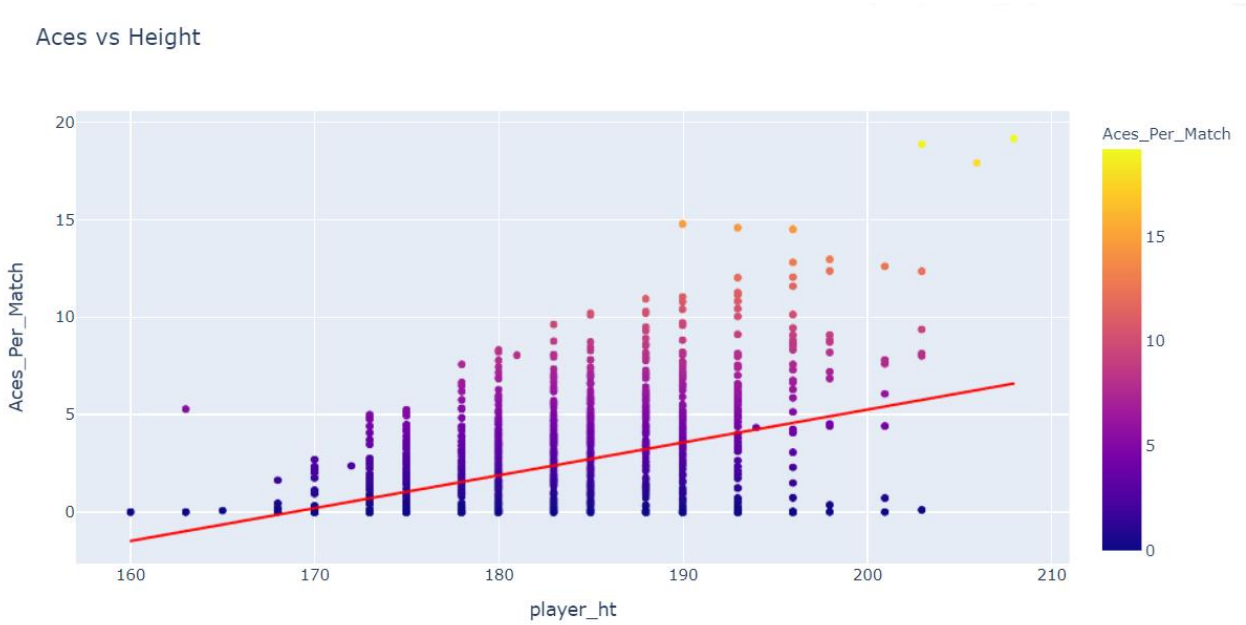
Both the plots are interactive. Each have two dropdowns where a user can select two players to compare their stats.

I have plotted the impact of surface on hitting an ace as a bar graph:



In the plot we can see that historically, the number of Aces per match on grass courts and hard courts are more than that of clay courts. This justifies our intuition that since grass and hard courts are faster surfaces, the chances of hitting an Ace is high.

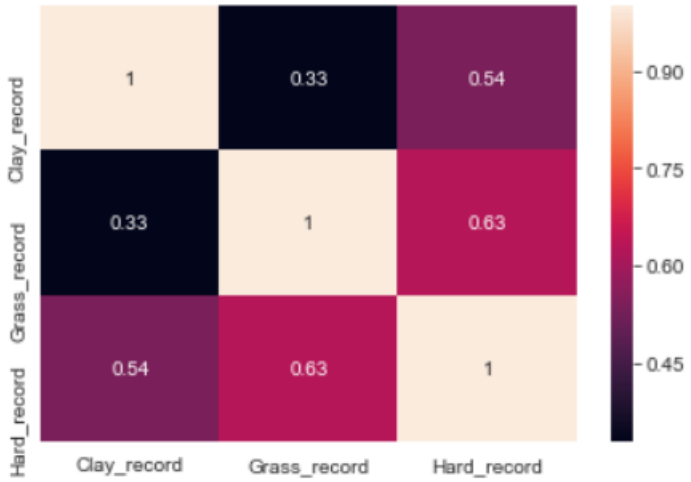
The interactive illustrating the relationship between height and number of Aces per match is as follow:



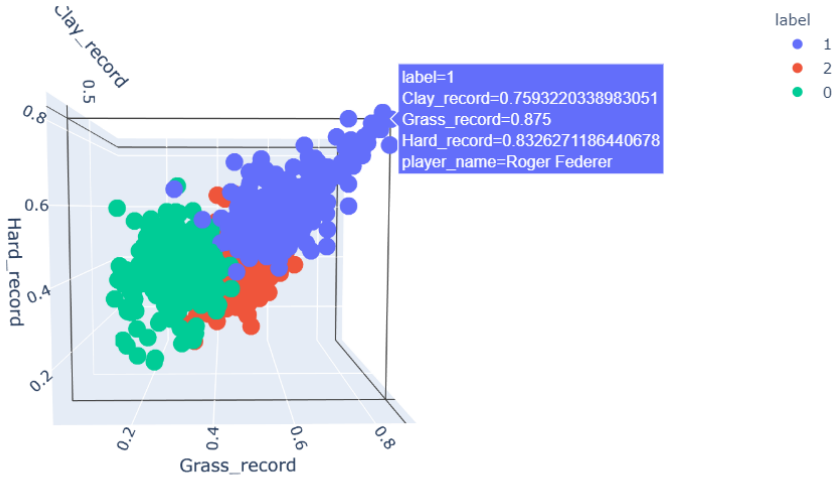
In the above plot we can clearly see that height has a positive impact on the ability to hit aces. Players like John Isner, Kevin Anderson, Ivo Karlovic, Albano Olivetti are few of the taller players in the Tour and they have the highest chances for hitting aces among all the players. The plot is also interactive- data relevant to the points and trend line are displayed when a user hover over them.

Correlation of Performance across surfaces:

The correlation matrix illustrating the relationship between performance on playing surfaces is as follows:



Above we can see that the there is a strong correlation between performance on hard courts and grass courts. I have plotted a 3-D plot of players performance across three surfaces after labelling using k-mean clustering algorithm. Following is the plot:



In the above plot we can see the positive correlation, i.e., good hardcourt players are also good grass court players. The plot is a 3-D interactive plot. It can be rotated, zoomed-in, and zoomed-out. On mouse hover, a prompt describing the data relevant to the points is displayed.

Future work and enhancements: The techniques used in this project are good starting points, and hopefully would pique the interest of others. Consistency can be further analyzed via change in performance attributes over time. Age is also an important factor that affects consistency. So, it would be interesting to see the impact of age on performance attribute which in turn would impact consistency. Serving and returning well in crucial moments of a match determines a player's resolve. Since tennis tournaments are played in a knock-out fashion, crucial moments are quite common. It would be interesting to see the impact of such moments on a player's ability to serve well.

Conclusion: In this project, we have seen how the data analysis and visualization techniques can be used to explore the hidden trends in tennis data. We evaluated the consistency and serving abilities of a player, analyzed the factors affecting the serving abilities of a players, and finally we observed how the performance of players relates across surfaces. The techniques and approaches used here can be improved further for a more comprehensive analysis. Similar techniques can be used to uncover trends in other sports data.

References:

1) **https://tennisvisuals.com/#**
2) **https://kaggle.com/**
3) **https://fivethirtyeight.com/**
4) **https://github.com/JeffSackmann/tennis_atp**

Appendix

December 7, 2020

1 Final Project Code

This section contains the code for generating the visualizations described in the report.

Import the required libraries.

[1]:
```
import pandas as pd
import plotly
from plotly.offline import init_notebook_mode
import warnings
import plotly.graph_objects as go
from ipywidgets import widgets

warnings.filterwarnings('ignore')


init_notebook_mode(connected=True)
```

Research Question 1: First lets explore the evolution of the current top 100 ATP players since they turned pro.

Import the dataset containing the stats of Top 100 ATP players. I have personally curated this dataset by scraping data from https://www.atptour.com/ using the selenium webdriver for chrome.

[2]:
```
atp_top_100=pd.read_csv('data/ATP_TOP_100.csv')
atp_top_100.drop('Unnamed: 0',inplace=True,axis=1) # Remove the extra index␣
↪column.
```

Let's see how the data looks like-

[3]:
```
atp_top_100.head()
```

[3]:
```
   Ranking           Player  Age  Points  Tourn Played  \
0        1   Novak Djokovic   33  12,030            18
1        2     Rafael Nadal   34   9,850            18
2        3    Dominic Thiem   27   9,125            21
3        4  Daniil Medvedev   24   8,470            24
4        5    Roger Federer   39   6,630            16

                                         profile_url    Aces Double Faults  \
0  https://www.atptour.com/en/players/novak-djoko...   5,870         2,478
1  https://www.atptour.com/en/players/rafael-nada...   3,604         1,878
```

```
2  https://www.atptour.com/en/players/dominic-thi...   2,615         1,244
3  https://www.atptour.com/en/players/daniil-medv...   1,881           815
4  https://www.atptour.com/en/players/roger-feder...  11,344         2,742

  1st Serve 1st Serve Points Won  ... Service Games Won  \
0       65%                  74%  ...                86%
1       68%                  72%  ...                86%
2       60%                  74%  ...                83%
3       59%                  74%  ...                82%
4       62%                  77%  ...                89%

  Total Service Points Won 1st Serve Return Points Won  \
0                      67%                         34%
1                      67%                         34%
2                      66%                         30%
3                      65%                         30%
4                      70%                         33%

  2nd Serve Return Points Won Break Points Opportunities  \
0                         55%                     9,497
1                         55%                    10,299
2                         50%                     3,376
3                         53%                     1,812
4                         51%                    11,822

  Break Points Converted Return Games Played Return Games Won  \
0                    44%              13,141               32%
1                    45%              13,824               34%
2                    39%               5,605               24%
3                    40%               2,921               25%
4                    41%              18,229               27%

  Return Points Won Total Points Won
0               42%               54%
1               42%               55%
2               38%               51%
3               39%               52%
4               40%               54%

[5 rows x 24 columns]
```

Now, let's extract the ranking history of the current world number 1, Novak Djokovic (my favorite).

[4]:
```
player_hist_df=pd.read_csv(f'data/Ranking_hist_1.csv')
player_hist_df.head()
```

[4]:
```
         Date Singles Doubles
0  2020.11.30       1     155
```

```
1  2020.11.23       1     155
2  2020.11.16       1     155
3  2020.11.09       1     154
4  2020.11.02       1     151
```

Let's only consider his records in Singles matches.

[5]:
```
player_hist_df=player_hist_df[['Date','Singles']]
```

Now, lets preprocess the data-

[6]:
```
def num_preprocess(x):

    noise={'%',',','T'}

    result=""

    if not isinstance(x,int):
        for l in x:
            if l not in noise:
                result+=l

        return int(result)
    return x


def preprocess(df_temp):
    """
    1. Convert date to appropriate format in pandas.
    2. Remove noise from Singles rankings column.
    3. Choose the best rank of a player in a year.
    4. Remove rows when ranking=0

    """

    df_temp['Date']=pd.to_datetime(df_temp.Date,format='%Y.%m.%d')

    df_temp['Singles']=df_temp['Singles'].apply(num_preprocess)
    df_temp['Singles']=pd.to_numeric(df_temp['Singles'])

    df_temp['Year']=pd.DatetimeIndex(df_temp.Date).year




preprocess(player_hist_df)
plot_df=player_hist_df.groupby(['Year'],as_index=False)['Singles'].min()
plot_df=plot_df[plot_df.Singles!=0]
```

Now, let's an interactive plot where a user can choose to compare the evolution of two players.

[7]:
```
trace1 = go.Scatter(x=plot_df['Year'], y=plot_df['Singles'],
                    name=f"{atp_top_100.iloc[0]['Player']}",
                                            line= {"shape": 'hv'})


trace2= go.Scatter(x=plot_df['Year'], y=plot_df['Singles'],
                    name=f'{atp_top_100.iloc[0]["Player"]}', line= {"shape":␣
 ↪'hv'})
```

[8]:
```
g1 = go.FigureWidget(data=[trace1,trace2],
                     layout=go.Layout(
                         title=dict(
                             text='Player Evolution'
                         ),
                         xaxis=dict(
                             title='Time'
                         ),
                         yaxis=dict(
                             title='Rank',
                             autorange="reversed"

                         )
                      ))
```

[9]:
```
player1_name = widgets.Dropdown(
    description='Player1 :   ',
    value='Novak Djokovic',
    options=atp_top_100.Player.unique().tolist())


player2_name = widgets.Dropdown(
    description='Player2 :   ',
    value='Novak Djokovic',
    options=atp_top_100.Player.unique().tolist())



container = widgets.HBox(children=[player1_name,player2_name])
```

[10]:
```
prev_player1="Novak Djokovic"
prev_player2="Novak Djokovic"

def response(change):
    global prev_player1, prev_player2
    with g1.batch_update():
        if prev_player1!=player1_name.value:
            new_player=player1_name.value
            index=0
```

```
            prev_player1=new_player
        else:
            new_player=player2_name.value
            index=1
            prev_player2=new_player

        new_player_rank=atp_top_100.loc[atp_top_100['Player']==new_player].
↪iloc[0]['Ranking']
        new_player_history=pd.read_csv(f'data/Ranking_hist_{new_player_rank}.
↪csv')[['Date','Singles']]
        #new_plot_df=preprocess(new_player_history)

        preprocess(new_player_history)
        new_plot_df=new_player_history.
↪groupby(['Year'],as_index=False)['Singles'].min()
        new_plot_df=new_plot_df[new_plot_df.Singles!=0]



        g1.data[index].x=new_plot_df['Year']
        g1.data[index].y=new_plot_df['Singles']
        g1.data[index].name=new_player



player1_name.observe(response, names="value")
player2_name.observe(response,names="value")
widgets.VBox([container,g1])
```

```
VBox(children=(HBox(children=(Dropdown(description='Player1 :   ', options=('Novak Djokovic', 'R
```

[11]:
```
plot_df_con=player_hist_df.groupby(['Year'],as_index=False)['Singles'].median()
plot_df_con=plot_df_con[plot_df_con.Singles!=0]
```

[12]:
```
trace1_con = go.Scatter(x=plot_df_con['Year'], y=plot_df_con['Singles'],
                  name=f"{atp_top_100.iloc[0]['Player']}",
                                          line= {"shape": 'hv'})


trace2_con= go.Scatter(x=plot_df_con['Year'], y=plot_df_con['Singles'],
                  name=f'{atp_top_100.iloc[0]["Player"]}', line= {"shape":␣
↪'hv'})
```

[13]:
```
g1_con = go.FigureWidget(data=[trace1_con,trace2_con],
                   layout=go.Layout(
                       title=dict(
                           text='Player Consistency'
```

```
                        ),
                        xaxis=dict(
                            title='Time'
                        ),
                        yaxis=dict(
                            title='Rank',
                            autorange="reversed"

                        )
                    ))
```

[14]:
```
player1_name_con = widgets.Dropdown(
    description='Player1 :   ',
    value='Novak Djokovic',
    options=atp_top_100.Player.unique().tolist())


player2_name_con = widgets.Dropdown(
    description='Player2 :   ',
    value='Novak Djokovic',
    options=atp_top_100.Player.unique().tolist())



container_con = widgets.HBox(children=[player1_name_con,player2_name_con])
```

[15]:
```
prev_player1="Novak Djokovic"
prev_player2="Novak Djokovic"

def response(change):
    global prev_player1, prev_player2

    with g1_con.batch_update():
        if prev_player1!=player1_name_con.value:
            new_player=player1_name_con.value
            index=0
            prev_player1=new_player

        else:
            new_player=player2_name_con.value
            index=1
            prev_player2=new_player

        new_player_rank=atp_top_100.loc[atp_top_100['Player']==new_player].
 ↪iloc[0]['Ranking']
        new_player_history=pd.read_csv(f'data/Ranking_hist_{new_player_rank}.
 ↪csv')[['Date','Singles']]
```

```
        preprocess(new_player_history)
        new_plot_df=new_player_history.
↪groupby(['Year'],as_index=False)['Singles'].median()
        new_plot_df=new_plot_df[new_plot_df.Singles!=0]


        g1_con.data[index].x=new_plot_df['Year']
        g1_con.data[index].y=new_plot_df['Singles']
        g1_con.data[index].name=new_player



player1_name_con.observe(response, names="value")
player2_name_con.observe(response,names="value")
widgets.VBox([container_con,g1_con])
```

```
VBox(children=(HBox(children=(Dropdown(description='Player1 :   ', options=('Novak Djokovic', 'R
```

The above plots can be used for a lot of reasons-

1. We can track the evolutions and consistency of a player in his career so far.
2. Whether the age of a player impacts his rankings.
3. Compare the career records of two players.
4. Analyze player rankings in a period of time.

Research Question 2: Now, let's see the impact of serve in a player's performance. Serve is an important aspect of a player's game- holding or breaking the serve at crucial points can turn a match around. There are two kinds of metrics which help us evaluate a player on his ability to serve or break an opponents serve-

1. Serve Stats - 1st Serve, 1st Serve Points Won, 2nd Serve Points Won, Break Points Faced, Break Points Saved, Service Games Played, Service Games Won, Total Service Points Won
2. Return Stats - 1st Serve Return Points Won, 2nd Serve Return Points Won, Break Points Opportunities, Break Points Converted, Return Games Played, Return Games Won, Return Points Won, Total Points Won

Let's first explore the serve statistics of players. Let's build a similar interactive plot as above.

[16]:
```
player1_name_s = widgets.Dropdown(
    description='Player1:   ',
    value='Novak Djokovic',
    options=atp_top_100.Player.unique().tolist())

player2_name_s = widgets.Dropdown(
    description='Player2:   ',
    value='Rafael Nadal',
```

```
    options=atp_top_100.Player.unique().tolist())


container_s = widgets.HBox(children=[player1_name_s,player2_name_s])
```

[17]:
```
serve_stats=pd.read_csv('data/Serve_Stats.csv',index_col=False)
return_stats=pd.read_csv('data/Return_Stats.csv',index_col=False)
```

[18]:
```
serve_stats.drop('Unnamed: 0',inplace=True,axis=1)
return_stats.drop('Unnamed: 0',inplace=True,axis=1)
```

[19]:
```
serve_stats.dtypes
```

[19]:
```
Aces                        object
Double Faults               object
1st Serve                   object
1st Serve Points Won        object
2nd Serve Points Won        object
Break Points Faced          object
Break Points Saved          object
Service Games Played        object
Service Games Won           object
Total Service Points Won    object
dtype: object
```

[20]:
```
return_stats.dtypes
```

[20]:
```
1st Serve Return Points Won    object
2nd Serve Return Points Won    object
Break Points Opportunities     object
Break Points Converted         object
Return Games Played            object
Return Games Won               object
Return Points Won              object
Total Points Won               object
dtype: object
```

[21]:
```
for column in serve_stats.columns:
    serve_stats[column]=serve_stats[column].apply(num_preprocess)
    serve_stats[column]=pd.to_numeric(serve_stats[column])

for column in return_stats.columns:
    return_stats[column]=return_stats[column].apply(num_preprocess)
    return_stats[column]=pd.to_numeric(return_stats[column])
```

[22]:
```
serve_stats.dtypes
```

[22]:
```
Aces                        int64
Double Faults               int64
1st Serve                   int64
1st Serve Points Won        int64
```

```
2nd Serve Points Won        int64
Break Points Faced          int64
Break Points Saved          int64
Service Games Played        int64
Service Games Won           int64
Total Service Points Won    int64
dtype: object
```

[23]:
```
return_stats.dtypes
```

[23]:
```
1st Serve Return Points Won    int64
2nd Serve Return Points Won    int64
Break Points Opportunities     int64
Break Points Converted         int64
Return Games Played            int64
Return Games Won               int64
Return Points Won              int64
Total Points Won               int64
dtype: object
```

[24]:
```
serve_stats_agg=serve_stats[['1st Serve Points Won',
       '2nd Serve Points Won',
        'Break Points Saved',
         'Service Games Won',
       'Total Service Points Won']]
return_stats_agg=return_stats[[
    '1st Serve Return Points Won', '2nd Serve Return Points Won',
    'Break Points Converted', 'Return Games Won', 'Return Points Won',
       'Total Points Won']]
```

Unlike the previous plot, the data here is multivariate. A better way to visualize the mulitvariate data is through a radar plot or parallel coordinate plots. I chose to use radar plots as they are commonly used in the sports domain, also they look cool.

[25]:
```
trace3=go.Scatterpolar(r=serve_stats_agg.iloc[0].values.
 ↪tolist(),theta=serve_stats_agg.iloc[0].keys().tolist(),
                       fill='toself',name='Novak Djokovic')

trace4=go.Scatterpolar(r=serve_stats_agg.iloc[1].values.
 ↪tolist(),theta=serve_stats_agg.iloc[1].keys().tolist(),
                       fill='toself',name='Rafal Nadel')
```

[26]:
```
g2 = go.FigureWidget(data=[trace3, trace4],
                     layout=go.Layout(
                         title=dict(
                             text='Serve Statistics'
                         ),
                         polar=dict(
                             radialaxis=dict(
                               visible=True,
```

```
                                range=[0, 100]
                            )),
                        showlegend=True
                    ))
```

[27]:
```
prev1='Novak Djokovic'
prev2='Rafal Nadel'


def serv_response(change):
    global prev1,prev2

    new_player=change['new']

    new_player_rank=atp_top_100.loc[atp_top_100['Player']==new_player].
 ↪iloc[0]['Ranking']
    new_player_stats=serve_stats_agg.iloc[new_player_rank-1]

    with g2.batch_update():
        if prev1!=player1_name_s.value:
            g2.data[0].r = new_player_stats.values.tolist()
            g2.data[0].name=new_player
            prev1=new_player
        else:
            g2.data[1].r = new_player_stats.values.tolist()
            g2.data[1].name=new_player
            prev2=new_player


player1_name_s.observe(serv_response, names="value")
player2_name_s.observe(serv_response, names="value")
widgets.VBox([container_s,g2])
```

```
VBox(children=(HBox(children=(Dropdown(description='Player1:    ', options=('Novak Djokovic', 'Ra
```

Now, let's generate a similar plot to illustrate the return statistics of players.

[28]:
```
trace5=go.Scatterpolar(r=return_stats_agg.iloc[0].values.
 ↪tolist(),theta=return_stats_agg.iloc[0].keys().tolist(),
                       fill='toself',name='Novak Djokovic')

trace6=go.Scatterpolar(r=return_stats_agg.iloc[1].values.
 ↪tolist(),theta=return_stats_agg.iloc[1].keys().tolist(),
                       fill='toself',name='Rafal Nadel')
```

[29]:
```
g3 = go.FigureWidget(data=[trace5, trace6],
                     layout=go.Layout(
                         title=dict(
                             text='Return Statistics'
                         ),
                         polar=dict(
                             radialaxis=dict(
                               visible=True,
                               range=[0, 100]
                             )),
                         showlegend=True
                     ))
```

[30]:
```
prev1='Novak Djokovic'
prev2='Rafal Nadel'
def ret_response(change):
    global prev1,prev2
    new_player=change['new']

    new_player_rank=atp_top_100.loc[atp_top_100['Player']==new_player].
 ↪iloc[0]['Ranking']
    new_player_stats=return_stats_agg.iloc[new_player_rank-1]

    with g3.batch_update():
        if prev1!=player1_name_r.value:
            g3.data[0].r = new_player_stats.values.tolist()
            g3.data[0].name=new_player
            prev1=new_player

        else:
            g3.data[1].r = new_player_stats.values.tolist()
            g3.data[1].name=new_player
            prev2=new_player


player1_name_r = widgets.Dropdown(
    description='Player1:   ',
    value='Novak Djokovic',
    options=atp_top_100.Player.unique().tolist())

player2_name_r = widgets.Dropdown(
    description='Player2:   ',
    value='Rafael Nadal',
    options=atp_top_100.Player.unique().tolist())


container_r = widgets.HBox(children=[player1_name_r,player2_name_r])
```

```
player1_name_r.observe(ret_response, names="value")
player2_name_r.observe(ret_response, names="value")
widgets.VBox([container_r,g3])
```

```
VBox(children=(HBox(children=(Dropdown(description='Player1:    ', options=('Novak Djokovic', 'Ra
```

The plots above can be used for the following analysis-

1) How a player serves and returns opponents serve.

2) How two players compare in serving and returning opponents serve.

There's more to a player's serve than mere stats. Two important criterias that determine a player's serve are-

1. Court surface - Professional tennis matches are typically played on four types of surfaces - Grass, Clay, Hard, and Carpet. The speed of the ball is impacted by the surface type and so is the serve. To that end, Grass and Hard courts are considered faster playing surfaces, while clay is a relatively slower surface and requires much physical strength to hit the ball.

2. Player's height - Taller players often have an advantage in hitting aces. John Isner, Ivo Karlovic are two of the taller players in professional tennis, and they excel at hitting aces.

So, let's compare the ability to hit aces based on playing surface. To explore this I have used ATP matches data from https://github.com/JeffSackmann/tennis_atp.

This repository is known for its comprehensive collection of data in professional tennis matches starting from year 1968 and is often cited in kaggle kernels and literature. So, I think it's a reliable resource.

Let's import the matches data -

[31]:
```
year_range=range(1968,2021,1)
```

[33]:
```
match_dfs=[ pd.read_csv(f'data/atp_matches_{year}.csv') for year in year_range ]
```

[34]:
```
master_match_df=pd.concat(match_dfs)
```

[35]:
```
master_match_df.shape
```

[35]:
```
(177642, 49)
```

The master_match_df has data relevant to all the ATP matches from 1968-2020. Now, let's clean the data, and extract the aces played during grandslams over the years. Grand slams are the grand stage of tennis. So, evaluating the aces played during grand slams would give us a fair idea.

[36]:
```
# Covert date to appropriate format
master_match_df['tourney_date']=pd.to_datetime(master_match_df.
 ↪tourney_date,format='%Y%m%d')
```

[37]:
```
# Count the total number of aces in each match.
master_match_df['Total_Aces']=master_match_df['w_ace']+master_match_df['l_ace']
```

[38]:
```
# Create a Match Count variable which can utilized to count the number of␣
 ↪matches during aggregation.
master_match_df['Match_Count']=[1]*len(master_match_df)
```

[39]:
```
# Aggregate the number of matches, aces played on each surface ny tournament␣
 ↪level

master_surface_mactes=master_match_df.
 ↪groupby(['surface','tourney_level','tourney_name'],

                                                 ␣
 ↪as_index=False)['Total_Aces','Match_Count'].sum()
```

[40]:
```
# Extract the data for grand slams.

plot_df=master_surface_mactes[master_surface_mactes['tourney_level']=='G']
```

[41]:
```
plot_df
```

[41]:

	surface	tourney_level	tourney_name	Total_Aces	Match_Count
2112	Clay	G	Roland Garros	42776.0	6683
2113	Clay	G	US Open	0.0	381
2305	Grass	G	Australian Chps.	0.0	61
2306	Grass	G	Australian Open	0.0	1224
2307	Grass	G	Australian Open-2	0.0	63
2308	Grass	G	US Open	0.0	837
2309	Grass	G	Wimbledon	74009.0	6604
3684	Hard	G	Australian Open	62414.0	4191
3685	Hard	G	US Open	62556.0	5335
3686	Hard	G	Us Open	2351.0	127

[42]:
```
plot_df=plot_df[plot_df.Total_Aces!=0]
```

[43]:
```
plot_df['tourney_name']=[name.upper() for name in plot_df['tourney_name']]
```

[44]:
```
plot_df
```

[44]:

	surface	tourney_level	tourney_name	Total_Aces	Match_Count
2112	Clay	G	ROLAND GARROS	42776.0	6683
2309	Grass	G	WIMBLEDON	74009.0	6604
3684	Hard	G	AUSTRALIAN OPEN	62414.0	4191
3685	Hard	G	US OPEN	62556.0	5335
3686	Hard	G	US OPEN	2351.0	127

Two of the four grand slams are played on hard courts, so that would make the plot biased if just consider the total aces on each surface- as there would more hard court tournaments than

clay or grass. One way to fix this would be consider the mean number of matches played on each types of courts in grad slams and then calculate the total number of aces per match. So, lets do that below-

[45]:
```
plot_df=plot_df.groupby(['surface'],as_index=False)['Total_Aces','Match_Count'].
↪mean()
```

[46]:
```
plot_df['Aces_Per_Match']=plot_df.Total_Aces/plot_df.Match_Count
```

[47]:
```
plot_df
```

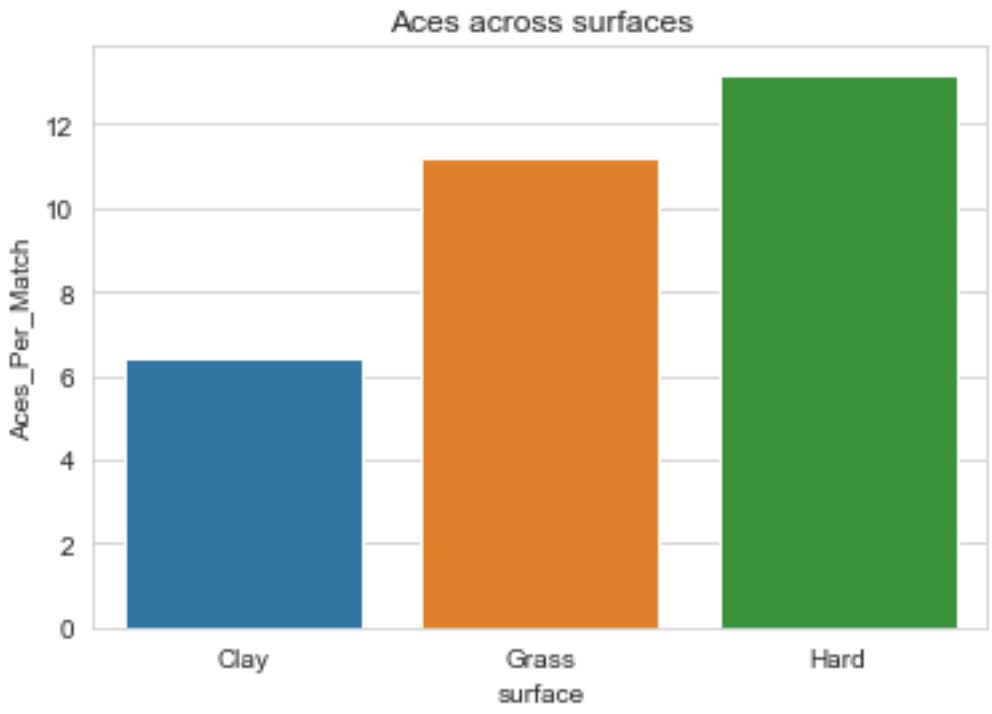
[47]:

	surface	Total_Aces	Match_Count	Aces_Per_Match
0	Clay	42776.000000	6683.000000	6.400718
1	Grass	74009.000000	6604.000000	11.206693
2	Hard	42440.333333	3217.666667	13.189786

Let' plot the results -

[49]:
```
import seaborn as sns

sns.set_style("whitegrid")
ax = sns.barplot(x="surface", y="Aces_Per_Match", data=plot_df)
ax.set_title('Aces across surfaces');
```



Above we can see that historically, the number of aces per match on grass courts and hard courts are more than that of clay courts. This is justified by our intution that since grass and hard courts are faster surfaces, the chances of hitting an Ace is high.

Now, lets verify the impact of height on hitting an ace. For this I will create a data frame of all players in the master_match_df and include their heights and total number of Aces across all the matches played by the player. Here too, if we consider the total number of aces, then we won't get a fair estimation of the relationship. So, I calculated the number of aces per match for each player.

[50]:
```
winners_df=master_match_df[['winner_name','winner_hand','winner_ht','winner_age','surface','w_a
losers_df=master_match_df[['loser_name','loser_hand','loser_ht','loser_age','surface','l_ace','
```

[51]:
```
win_aces_by_height=winners_df.
 ↪groupby(['winner_name','winner_ht'],as_index=False)['w_ace','Match_Count'].
 ↪sum()
loser_aces_by_height=losers_df.
 ↪groupby(['loser_name','loser_ht'],as_index=False)['l_ace','Match_Count'].sum()
```

[52]:
```
win_aces_by_height = win_aces_by_height.rename(columns={'winner_name':␣
 ↪'player_name',
                                                        'winner_ht':
 ↪'player_ht',
                                                        'w_ace':'aces',
                                                          'Match_Count':
 ↪'Matches_Played'

                                                       })



loser_aces_by_height=loser_aces_by_height.rename(
    columns={

            'loser_name': 'player_name',
            'loser_ht':'player_ht',
            'l_ace':'aces',
        'Match_Count':'Matches_Played'

    })
```

[53]:
```
players_dfs=[win_aces_by_height,loser_aces_by_height]
players_df=pd.concat(players_dfs)
```

[54]:
```
players_df=players_df.
 ↪groupby(['player_name','player_ht'],as_index=False)['aces','Matches_Played'].
 ↪sum()
```

[55]:
```
players_df['Aces_Per_Match']=players_df.aces/players_df.Matches_Played
```

[56]:
```
players_df=players_df[players_df.Matches_Played>10]
```

Let's visualize the relationship using a trend line-

[57]:
```
import plotly.express as px
```

```
fig = px.
 ↪scatter(players_df,y='Aces_Per_Match',x='player_ht',hover_data=['aces','player_name','player_
                trendline_color_override='red',title='Aces vs␣
 ↪Height',color='Aces_Per_Match')
fig.show()
```

In the above plot we can clearly see that height has a positive impact on the abillity to hit aces. Players like John Isner, Kevin Anderson, Ivo Karlovic, Albano Olivetti are few of the taller players in the Tour and they have the highest chances for hitting aces among all the players. The plot is also interactive- data relevant to the points and trend line are displayed when a user hovers over them.

Research Question 3: Now, let's analyze how a player's performance is related across surfaces. The playing conditions on a hard-court are kind of a bargain between grass and clay- the speed of the ball is slightly less than grass courts but more than clay courts. So, generally players who are dominant on hard courts fare well on other two courts. But players who are fully-dominant on clay/grass courts do not necessarily dominate on other courts.

For this analysis, I will be calculating the win-index of players across each surface.

win_index=(number_of_wins) / (number_of_wins+number_of_losses)

[58]:
```
surfaces=master_match_df.surface.unique().tolist()
surfaces
```

[58]:
```
['Grass', 'Clay', 'Carpet', 'Hard', nan]
```

[59]:
```
import numpy as np
player_names=master_match_df.winner_name.unique().tolist()+master_match_df.
 ↪loser_name.unique().tolist()
player_names=np.unique(player_names)
```

[60]:
```
player_records={
    'player_name':player_names
}


player_records=pd.DataFrame.from_dict(player_records)
```

[61]:
```
surface_wins_df=master_match_df.
 ↪groupby(['winner_name','surface'],as_index=False)['Match_Count'].sum()
surface_losses_df=master_match_df.
 ↪groupby(['loser_name','surface'],as_index=False)['Match_Count'].sum()
overall_wins_df=master_match_df.
 ↪groupby(['winner_name'],as_index=False)['Match_Count'].sum()
overall_losses_df=master_match_df.
 ↪groupby(['loser_name'],as_index=False)['Match_Count'].sum()
```

[62]:
```
grass_win_records=surface_wins_df[surface_wins_df.surface=='Grass']
clay_win_records=surface_wins_df[surface_wins_df.surface=='Clay']
hard_win_records=surface_wins_df[surface_wins_df.surface=='Hard']
```

```
grass_loss_records=surface_losses_df[surface_losses_df.surface=='Grass']
clay_loss_records=surface_losses_df[surface_losses_df.surface=='Clay']
hard_loss_records=surface_losses_df[surface_losses_df.surface=='Hard']
```

[63]:
```
overall_wins_df=overall_wins_df.rename(
columns={

    'winner_name':'player_name',
    'Match_Count':'overall_wins'
}

)

overall_losses_df=overall_losses_df.rename(
columns={

    'loser_name':'player_name',
    'Match_Count':'overall_losses'
}

)


grass_win_records=grass_win_records.rename(

columns={

    'winner_name':'player_name',
    'Match_Count':'grass_wins'

}

)


grass_loss_records=grass_loss_records.rename(

columns={

    'loser_name':'player_name',
    'Match_Count':'grass_losses'

}

)
```

```
clay_win_records=clay_win_records.rename(

columns={

    'winner_name':'player_name',
    'Match_Count':'clay_wins'

}

)


clay_loss_records=clay_loss_records.rename(

columns={

    'loser_name':'player_name',
    'Match_Count':'clay_losses'

}

)



hard_win_records=hard_win_records.rename(

columns={

    'winner_name':'player_name',
    'Match_Count':'hard_wins'

}

)



hard_loss_records=hard_loss_records.rename(

columns={

    'loser_name':'player_name',
    'Match_Count':'hard_losses'

}
```

```
)
```

[64]:
```
grass_win_records.drop(columns=['surface'],inplace=True)
grass_loss_records.drop(columns=['surface'],inplace=True)
hard_win_records.drop(columns=['surface'],inplace=True)
hard_loss_records.drop(columns=['surface'],inplace=True)
clay_win_records.drop(columns=['surface'],inplace=True)
clay_loss_records.drop(columns=['surface'],inplace=True)
```

[65]:
```
player_records=player_records.merge(overall_wins_df,on='player_name',how='left')
player_records=player_records.
 ↪merge(overall_losses_df,on='player_name',how='left')
player_records=player_records.
 ↪merge(grass_win_records,on='player_name',how='left')
player_records=player_records.
 ↪merge(grass_loss_records,on='player_name',how='left')
player_records=player_records.merge(clay_win_records,on='player_name',how='left')
player_records=player_records.
 ↪merge(clay_loss_records,on='player_name',how='left')
player_records=player_records.merge(hard_win_records,on='player_name',how='left')
player_records=player_records.
 ↪merge(hard_loss_records,on='player_name',how='left')
```

[66]:
```
player_records['clay_win_index']=player_records['clay_wins']/
 ↪(player_records['clay_wins']+player_records['clay_losses'])
player_records['grass_win_index']=player_records['grass_wins']/
 ↪(player_records['grass_wins']+player_records['grass_losses'])
player_records['hard_win_index']=player_records['hard_wins']/
 ↪(player_records['hard_wins']+player_records['hard_losses'])
```

[67]:
```
player_records=player_records.dropna()
```

[68]:
```
player_records=player_records[(player_records.overall_wins+player_records.
 ↪overall_losses)>100]
```

Now that we have the win-index of all the players, across each surface, lets use this data to cluster players into groups. Let's use a simple yet poweful clustering algorithm called k-means for this analysis.

[69]:
```
from sklearn.cluster import KMeans


data= {'Clay_record': player_records.clay_win_index,
       'Grass_record' :player_records.grass_win_index,
       'Hard_record':player_records.hard_win_index}
```

[70]:
```
kmeans_df =  pd.DataFrame(data=data)

kmeans = KMeans(n_clusters = 3, random_state = 0).fit(kmeans_df)
kmeans_df['label'] = kmeans.labels_
```

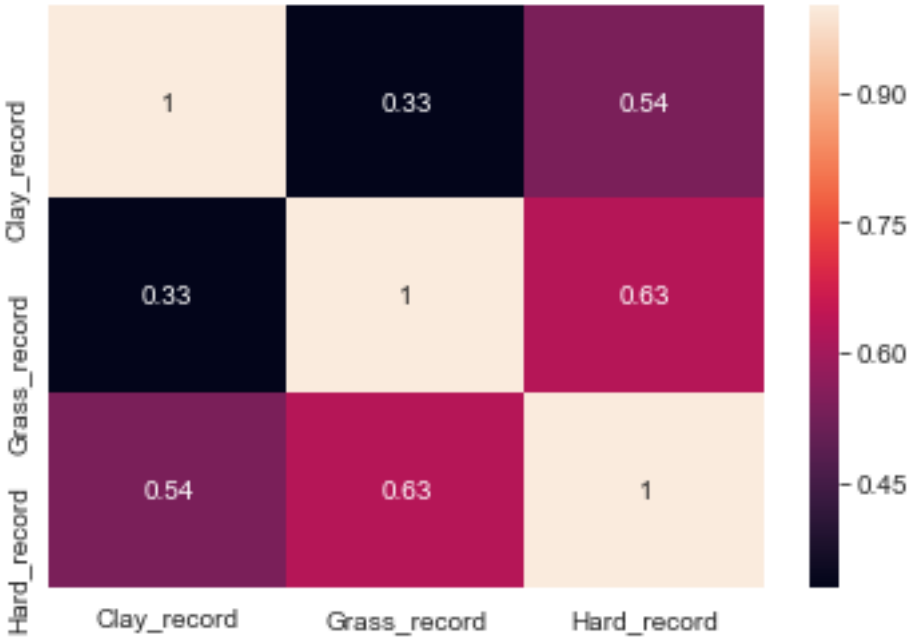
```
kmeans_df['player_name']=player_records.player_name
```

[71]:
```
kmeans_df['label']=kmeans_df['label'].apply(str)
```

[72]:
```
fig = px.scatter_3d(kmeans_df, x='Clay_record', y='Grass_record',␣
↪z='Hard_record',
                color='label',hover_data=kmeans_df.columns.values)
fig.show()
```

In the above plot, we can clearly see the positive correlation between hard court records and grass or clay court records, the correlation with grass court records is most dominant. The correlation of grass court records and clay court records is not significant. For quatitative analysis of this relationship, lets plot a correlation heatmap-

[73]:
```
sns.heatmap(kmeans_df[['Clay_record','Grass_record','Hard_record']].
↪corr(),annot=True);
```



In above heat-map, we can clearly see that the relationship of hart court records and grass court records is significant.