# Exploration of Soccer Dataset

### Research task: show most valued attributes for field players

Used European Soccer Dataset from www.kaggle.com (https://www.kaggle.com/paosheng/european-soccer-database)

In [1]:
```
#import required liblaries
import sqlite3
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

## Step 1. Acquire data

Dataset European Soccer is the sqlite db, so use sqlite3 library

In [2]:
```
conn = sqlite3.connect('database.sqlite')
df = pd.read_sql_query("SELECT * FROM Player_Attributes", conn)
df.head()
```

Out[2]:

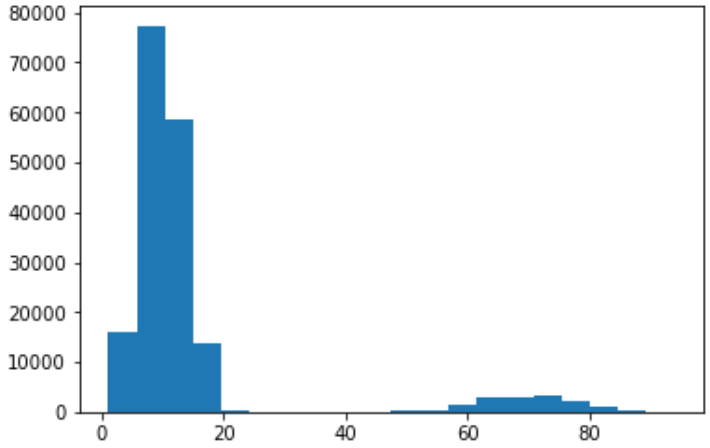
| | defensive_work_rate | crossing | ... | vision | penalties | marking | standing_tackle | sliding_tackle | gk_diving |
|---|---|---|---|---|---|---|---|---|---|
| 1 | medium | 49.0 | ... | 54.0 | 48.0 | 65.0 | 69.0 | 69.0 | 6.0 |
| 1 | medium | 49.0 | ... | 54.0 | 48.0 | 65.0 | 69.0 | 69.0 | 6.0 |
| 1 | medium | 49.0 | ... | 54.0 | 48.0 | 65.0 | 66.0 | 69.0 | 6.0 |
| 1 | medium | 48.0 | ... | 53.0 | 47.0 | 62.0 | 63.0 | 66.0 | 5.0 |
| 1 | medium | 48.0 | ... | 53.0 | 47.0 | 62.0 | 63.0 | 66.0 | 5.0 |

## Step 2. Data preparation

We see that table *Player_Attributes* contains attributes as for field players as for goalkeepers ('gk_diving' attribute for example). Because different skills required for both groups we must define how to select data only for field players.

In [3]:
```
#Remove Nans for initial data analysis
df.dropna(how='any', axis=0, inplace=True);
```

In [4]:
```
# Breefly look at distribution of attributes
plt.hist(df['gk_diving'], bins=20)
plt.show()
```




We see, that goalkeepers attributes has bimodal distribution. We may assume that high values of attributes relates to goalkeepers, low values for other players. So, we can divide data by *gk_diving* attribute.

```
Selects only data required data (attributes and row related to field players)
```

In [5]:
```
#Take required dependent variable with key
dep = df.loc[:, ['id','overall_rating']].columns
#Take independent variables
columns = dep.append(df.loc[:, 'crossing':'sliding_tackle'].columns)

#Select data related to field players
filtered = df[df['gk_diving']<25]
selecteddata = filtered[columns]
selecteddata.head()
```

Out[5]:

| | id | overall_rating | crossing | finishing | heading_accuracy | short_passing | volleys | dribbling | curve | fre |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 67.0 | 49.0 | 44.0 | 71.0 | 61.0 | 44.0 | 51.0 | 45.0 | |
| **1** | 2 | 67.0 | 49.0 | 44.0 | 71.0 | 61.0 | 44.0 | 51.0 | 45.0 | |
| **2** | 3 | 62.0 | 49.0 | 44.0 | 71.0 | 61.0 | 44.0 | 51.0 | 45.0 | |
| **3** | 4 | 61.0 | 48.0 | 43.0 | 70.0 | 60.0 | 43.0 | 50.0 | 44.0 | |
| **4** | 5 | 61.0 | 48.0 | 43.0 | 70.0 | 60.0 | 43.0 | 50.0 | 44.0 | |

5 rows × 30 columns

In [6]:
```
selecteddata.describe().T
```

Out[6]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **id** | 165615.0 | 91789.326740 | 53038.960805 | 1.0 | 45982.5 | 91665.0 | 137451.5 | 183978.0 |
| **overall_rating** | 165615.0 | 68.667512 | 6.976079 | 33.0 | 64.0 | 69.0 | 73.0 | 94.0 |
| **crossing** | 165615.0 | 58.340422 | 13.881403 | 8.0 | 49.0 | 60.0 | 68.0 | 95.0 |
| **finishing** | 165615.0 | 52.752776 | 17.116430 | 7.0 | 39.0 | 56.0 | 66.0 | 97.0 |
| **heading_accuracy** | 165615.0 | 60.612034 | 12.311963 | 2.0 | 52.0 | 61.0 | 69.0 | 98.0 |
| **short_passing** | 165615.0 | 65.513800 | 9.986661 | 15.0 | 60.0 | 66.0 | 72.0 | 97.0 |
| **volleys** | 165615.0 | 52.309712 | 16.085748 | 6.0 | 40.0 | 54.0 | 65.0 | 93.0 |
| **dribbling** | 165615.0 | 62.818700 | 13.488270 | 2.0 | 56.0 | 65.0 | 72.0 | 97.0 |
| **curve** | 165615.0 | 56.102823 | 15.425054 | 10.0 | 45.0 | 58.0 | 68.0 | 94.0 |
| **free_kick_accuracy** | 165615.0 | 52.112073 | 15.677617 | 6.0 | 40.0 | 52.0 | 65.0 | 97.0 |
| **long_passing** | 165615.0 | 59.161525 | 12.246135 | 11.0 | 52.0 | 60.0 | 68.0 | 97.0 |
| **ball_control** | 165615.0 | 66.907665 | 9.991540 | 15.0 | 61.0 | 68.0 | 74.0 | 97.0 |
| **acceleration** | 165615.0 | 69.642979 | 11.306552 | 21.0 | 64.0 | 70.0 | 77.0 | 97.0 |
| **sprint_speed** | 165615.0 | 70.038662 | 10.786578 | 20.0 | 65.0 | 71.0 | 77.0 | 97.0 |
| **agility** | 165615.0 | 67.548869 | 11.866140 | 19.0 | 60.0 | 69.0 | 76.0 | 96.0 |
| **reactions** | 165615.0 | 66.431549 | 8.868919 | 17.0 | 61.0 | 67.0 | 73.0 | 96.0 |
| **balance** | 165615.0 | 66.734535 | 11.966983 | 20.0 | 60.0 | 68.0 | 75.0 | 96.0 |
| **shot_power** | 165615.0 | 64.779966 | 12.997914 | 7.0 | 58.0 | 67.0 | 74.0 | 97.0 |
| **jumping** | 165615.0 | 67.145766 | 11.063809 | 19.0 | 61.0 | 68.0 | 74.0 | 96.0 |
| **stamina** | 165615.0 | 69.308861 | 10.661078 | 20.0 | 63.0 | 70.0 | 76.0 | 96.0 |
| **strength** | 165615.0 | 67.750053 | 12.086439 | 12.0 | 61.0 | 69.0 | 76.0 | 96.0 |
| **long_shots** | 165615.0 | 56.456354 | 15.695941 | 6.0 | 47.0 | 59.0 | 68.0 | 96.0 |
| **aggression** | 165615.0 | 63.120653 | 14.323907 | 6.0 | 54.0 | 65.0 | 74.0 | 97.0 |
| **interceptions** | 165615.0 | 54.129976 | 18.369112 | 10.0 | 39.0 | 59.0 | 69.0 | 96.0 |
| **positioning** | 165615.0 | 58.800260 | 15.523845 | 8.0 | 50.0 | 62.0 | 70.0 | 95.0 |
| **vision** | 165615.0 | 60.203756 | 12.944547 | 10.0 | 52.0 | 62.0 | 70.0 | 97.0 |
| **penalties** | 165615.0 | 57.181022 | 13.512969 | 8.0 | 48.0 | 58.0 | 67.0 | 96.0 |
| **marking** | 165615.0 | 49.258074 | 20.229920 | 4.0 | 28.0 | 54.0 | 67.0 | 94.0 |
| **standing_tackle** | 165615.0 | 53.126408 | 20.094533 | 4.0 | 34.0 | 59.0 | 70.0 | 95.0 |
| **sliding_tackle** | 165615.0 | 50.676014 | 20.400078 | 6.0 | 30.0 | 57.0 | 68.0 | 95.0 |

```
Now we have data ready for analysis
```

# Step 3. Data analysis

We assume that mostly valued attributed good correlated with *overall rating*. So, we calculate pair correlations between attributes and *overall rating*, select good correllated attributes thas has R≥0.40 and visualize.

Instead of use **corr** function for full and excess matrix of pair correlations, we transform data for calculate only necessary correlations via grouping

In [30]:
```
#Transform independent variables data in parameter/value form via melt
melteddata = pd.melt(selecteddata, id_vars=['id'], value_vars=attributeColumns, var_na
#Join with dependent variable on id of player
attributes = df[dep].merge(melteddata, on='id' )
attributes.head(12)
```

Out[30]:

| | id | overall_rating | attribute | value_of_attribute |
|---|---|---|---|---|
| **0** | 1 | 67.0 | crossing | 49.0 |
| **1** | 1 | 67.0 | finishing | 44.0 |
| **2** | 1 | 67.0 | heading_accuracy | 71.0 |
| **3** | 1 | 67.0 | short_passing | 61.0 |
| **4** | 1 | 67.0 | volleys | 44.0 |
| **5** | 1 | 67.0 | dribbling | 51.0 |
| **6** | 1 | 67.0 | curve | 45.0 |
| **7** | 1 | 67.0 | free_kick_accuracy | 39.0 |
| **8** | 1 | 67.0 | long_passing | 64.0 |
| **9** | 1 | 67.0 | ball_control | 49.0 |
| **10** | 1 | 67.0 | acceleration | 60.0 |
| **11** | 1 | 67.0 | sprint_speed | 64.0 |

In [32]:
```
#calculate pair correlations with group by and aggregate function corr
groupedresult = attributes.groupby('attribute')['overall_rating', 'value_of_attribute
#Get even rows for clean up result
corrs = pd.DataFrame(groupedresult['value_of_attribute'].iloc[::2])
corrs.head(10)
```

Out[32]:

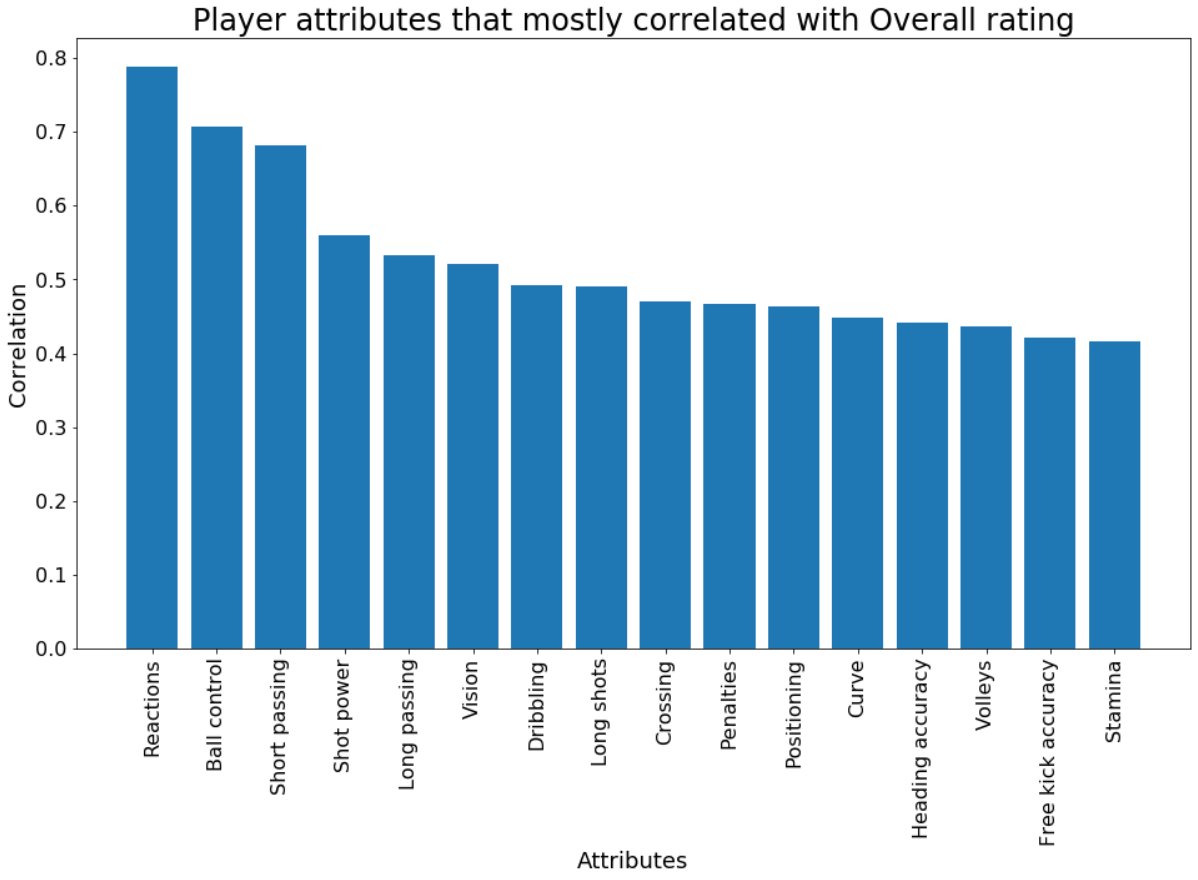
| | | value_of_attribute |
|---|---|---|
| **attribute** | | |
| **acceleration** | **overall_rating** | 0.275184 |
| **aggression** | **overall_rating** | 0.366963 |
| **agility** | **overall_rating** | 0.260139 |
| **balance** | **overall_rating** | 0.175277 |
| **ball_control** | **overall_rating** | 0.706608 |
| **crossing** | **overall_rating** | 0.470939 |
| **curve** | **overall_rating** | 0.449042 |
| **dribbling** | **overall_rating** | 0.492527 |
| **finishing** | **overall_rating** | 0.390949 |
| **free_kick_accuracy** | **overall_rating** | 0.422069 |

In [34]:
```
#Order correlations for visualization
orderedcorrs = corrs.sort_values(by=['value_of_attribute'], ascending=False)
#Drop poor correlated attributes
goodcorrs = orderedcorrs[orderedcorrs['value_of_attribute']>=0.4]

#prepare attribute names for use in charts
attributenames = pd.Series([str(name[0]) for name in goodcorrs.index[:]])
attributeColumns = selecteddata.loc[:, 'crossing':'sliding_tackle'].columns
readable_attributenames = attributenames[:].str.replace('_',' ').str.capitalize()
headers = dict(zip(attributenames, readable_attributenames))

readable_attributenames
```

Out[34]:
```
0              Reactions
1           Ball control
2          Short passing
3             Shot power
4           Long passing
5                 Vision
6              Dribbling
7             Long shots
8               Crossing
9              Penalties
10           Positioning
11                 Curve
12      Heading accuracy
13               Volleys
14    Free kick accuracy
15               Stamina
dtype: object
```

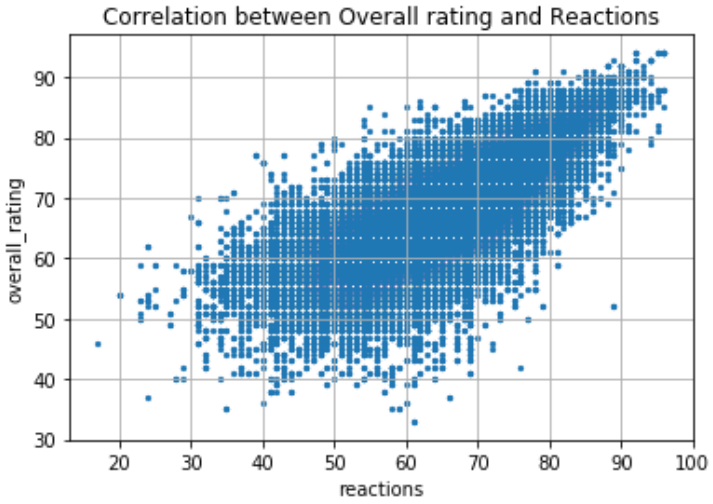
# Step 4. Present data

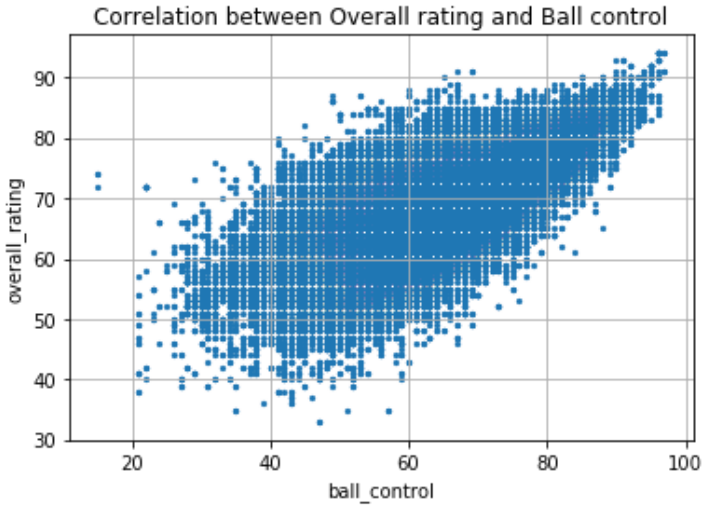
In [20]:
```
#Visualize data via bar chart of correlation coefficients. We must get styled chart be

plt.figure(num=None, figsize=(16, 9), dpi=72, facecolor='w', edgecolor='k')

plt.bar(attributenames.index, goodcorrs['value_of_attribute'].values)
plt.xticks(attributenames.index, readable_attributenames, rotation=90, fontsize=16)
plt.yticks(fontsize=16)
plt.ylabel('Correlation', fontsize=18)
plt.xlabel('Attributes', fontsize=18)
plt.title('Player attributes that mostly correlated with Overall rating', fontsize=24)
plt.show()
```

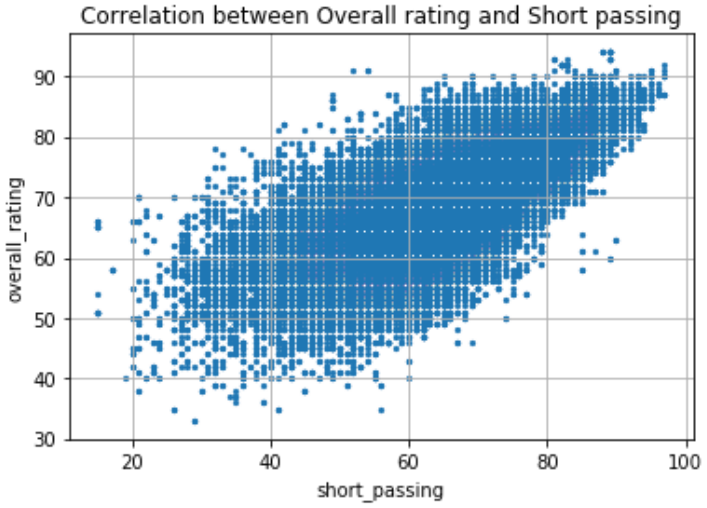



# Additional tests

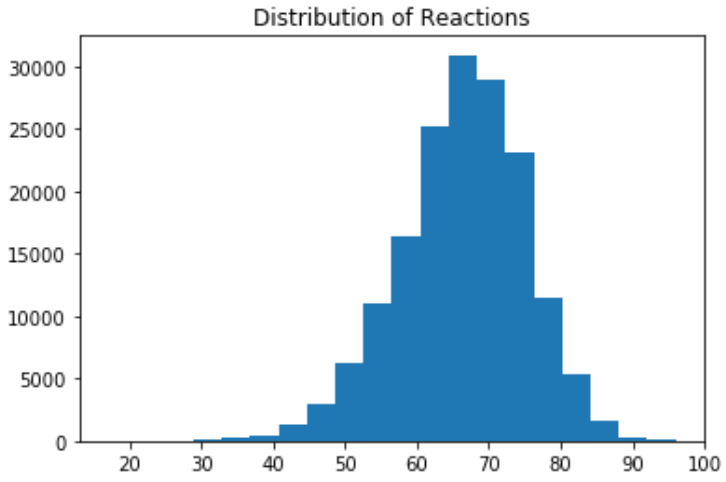
In [37]:
```
# We must make sure that correlations not distorted by outliers, multimodality etc.
# Lets see scattertplots and histograms for top 3 attributes

for a in attributenames[:3]:
    plt.scatter(selecteddata[a], selecteddata['overall_rating'], s=5)
    plt.xlabel(a)
    plt.ylabel('overall_rating')
    plt.grid(True)
    plt.title('Correlation between Overall rating and {}'.format(headers[a]))
    plt.show()
```

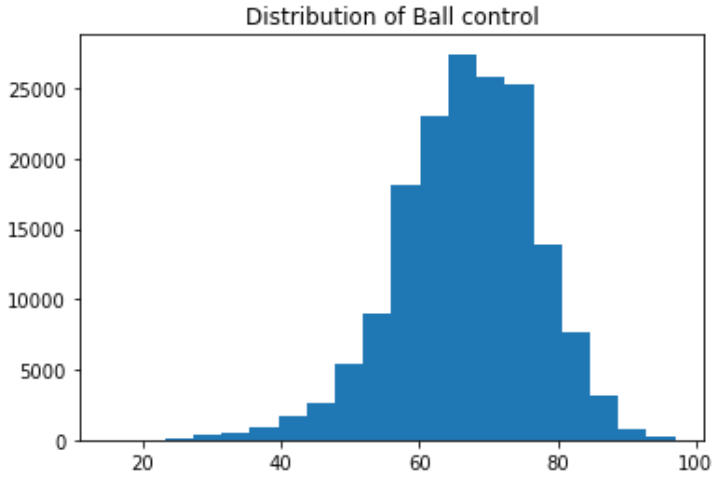


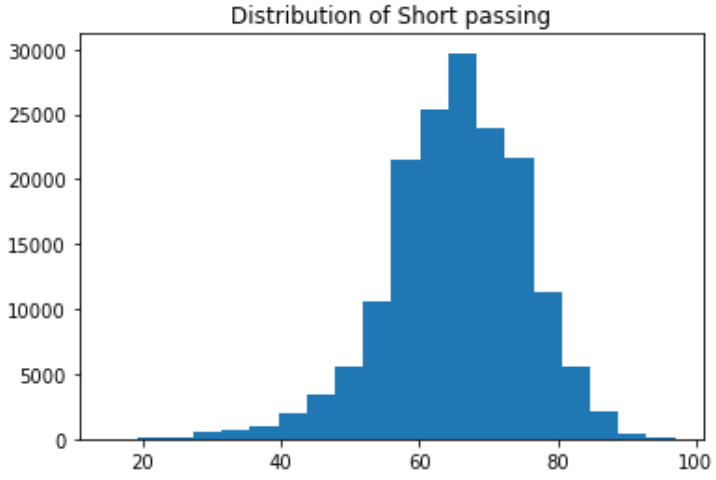



In [19]:
```
for a in attributenames[:3]:
    plt.hist(selecteddata[a], bins=20)
    plt.title('Distribution of {}'.format(headers[a]))
    plt.show()
```




## Resume: Most valued attributes for field players is:

- **Reactions**
- **Ball control**
- **Short passing**