Выполнила Прус Ева, ИУ5-64, Вариант 16

In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```python
fifa = '/Users/eva/Documents/Учеба/jupnote/FIFA 2018 Statistics.csv'
df = pd.read_csv(fifa, sep=",")
```

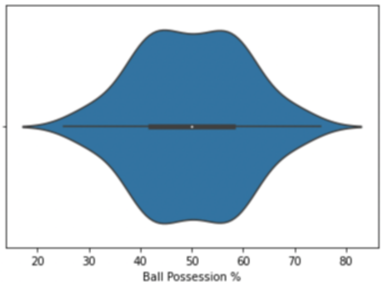
In [3]:
```python
df
```

Out[3]:

Team	Opponent	Goal Scored	Ball Possession %	Attempts	On-Target	Off-Target	Blocked	Corners	...	Yellow Card	Yellow & Red	Red	Man of the Match	1st Goal	Round	PSO	Goals in PSO	Own goals	Own goal Time
Russia	Saudi Arabia	5	40	13	7	3	3	6	...	0	0	0	Yes	12.0	Group Stage	No	0	NaN	NaN
Saudi Arabia	Russia	0	60	6	0	3	3	2	...	0	0	0	No	NaN	Group Stage	No	0	NaN	NaN
Egypt	Uruguay	0	43	8	3	3	2	0	...	2	0	0	No	NaN	Group Stage	No	0	NaN	NaN
Uruguay	Egypt	1	57	14	4	6	4	5	...	0	0	0	Yes	89.0	Group Stage	No	0	NaN	NaN
Morocco	Iran	0	64	13	3	6	4	5	...	1	0	0	No	NaN	Group Stage	No	0	1.0	90.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
England	Croatia	1	46	11	1	6	4	4	...	1	0	0	No	5.0	Semi-Finals	No	0	NaN	NaN

In [36]:
```python
sns.violinplot(x=df['Ball Possession %'])
```

Out[36]: <AxesSubplot:xlabel='Ball Possession %'>



In [4]:
```python
df.describe()
```

Out[4]:

	Goal Scored	Ball Possession %	Attempts	On-Target	Off-Target	Blocked	Corners	Offsides	Free Kicks	Saves	...	Passes	Distance Covered (Kms)
count	128.000000	128.000000	128.000000	128.000000	128.000000	128.000000	128.000000	128.000000	128.000000	128.000000	...	128.000000	128.000000
mean	1.320312	49.992188	12.593750	3.914062	5.273438	3.359375	4.718750	1.343750	14.890625	2.726562	...	462.648438	106.664062
std	1.156519	10.444074	5.245827	2.234403	2.409675	2.403195	2.446072	1.193404	4.724262	2.049447	...	151.186311	11.749537
min	0.000000	25.000000	3.000000	0.000000	1.000000	0.000000	0.000000	0.000000	5.000000	0.000000	...	189.000000	80.000000
25%	0.000000	42.000000	9.000000	2.000000	4.000000	1.750000	3.000000	0.000000	11.000000	1.000000	...	351.000000	101.000000
50%	1.000000	50.000000	12.000000	3.500000	5.000000	3.000000	5.000000	1.000000	15.000000	2.000000	...	462.000000	104.500000
75%	2.000000	58.000000	15.000000	5.000000	7.000000	4.000000	6.000000	2.000000	18.000000	4.000000	...	555.250000	109.000000
max	6.000000	75.000000	26.000000	12.000000	11.000000	10.000000	11.000000	5.000000	26.000000	9.000000	...	1137.000000	148.000000

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128 entries, 0 to 127
Data columns (total 27 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Date                    128 non-null    object
 1   Team                    128 non-null    object
 2   Opponent                128 non-null    object
 3   Goal Scored             128 non-null    int64
 4   Ball Possession %       128 non-null    int64
 5   Attempts                128 non-null    int64
 6   On-Target               128 non-null    int64
 7   Off-Target              128 non-null    int64
 8   Blocked                 128 non-null    int64
 9   Corners                 128 non-null    int64
 10  Offsides                128 non-null    int64
 11  Free Kicks              128 non-null    int64
 12  Saves                   128 non-null    int64
 13  Pass Accuracy %         128 non-null    int64
 14  Passes                  128 non-null    int64
 15  Distance Covered (Kms)  128 non-null    int64
 16  Fouls Committed         128 non-null    int64
 17  Yellow Card             128 non-null    int64
 18  Yellow & Red            128 non-null    int64
 19  Red                     128 non-null    int64
 20  Man of the Match        128 non-null    object
 21  1st Goal                94 non-null     float64
 22  Round                   128 non-null    object
 23  PSO                     128 non-null    object
 24  Goals in PSO            128 non-null    int64
 25  Own goals               12 non-null     float64
 26  Own goal Time           12 non-null     float64
dtypes: float64(3), int64(18), object(6)
memory usage: 27.1+ KB
```

Пропусков в категориальных данных обнаружено не было. Заполним пропуски у поля 1st Goal, так как в этом столбце их не так много.

In [7]:
```
df_fg = df[['1st Goal']]
df_fg
```

Out[7]:

	1st Goal
0	12.0
1	NaN
2	NaN
3	89.0
4	NaN
...	...
123	5.0
124	4.0
125	NaN
126	18.0
127	28.0

128 rows × 1 columns

In [11]:
```
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```

In [12]:
```
indicator = MissingIndicator()
mask_missing_values_only = indicator.fit_transform(df_fg)
mask_missing_values_only
```

Out[12]:
```
array([[False],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
```

In [13]:
```
strategies=['mean', 'median', 'most_frequent']
```

In [14]:
```
def test_num_impute(strategy_param):
    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(df_fg)
    return data_num_imp[mask_missing_values_only]
```

```
При сравнении трех стратегий, была выбрана стратегия 'mean'
```

In [16]:
```
strategies[0], test_num_impute(strategies[0])
```

Out[16]:
```
('mean',
 array([39.45744681, 39.45744681, 39.45744681, 39.45744681, 39.45744681,
        39.45744681, 39.45744681, 39.45744681, 39.45744681, 39.45744681,
        39.45744681, 39.45744681, 39.45744681, 39.45744681, 39.45744681,
        39.45744681, 39.45744681, 39.45744681, 39.45744681, 39.45744681,
        39.45744681, 39.45744681, 39.45744681, 39.45744681, 39.45744681,
        39.45744681, 39.45744681, 39.45744681, 39.45744681, 39.45744681,
        39.45744681, 39.45744681, 39.45744681, 39.45744681]))
```

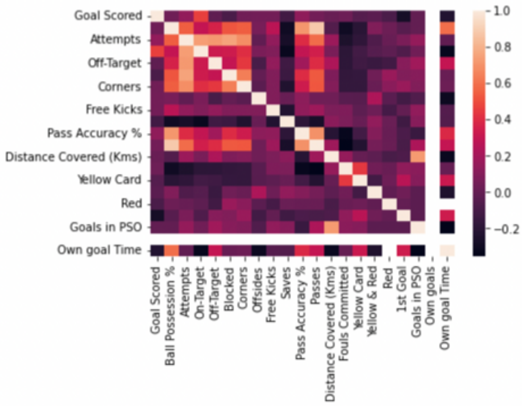
Таким образом, была использована импутация. В ходе реализации была использован метод SimpleImputor, использующий страгию "Среднее значение". При сравнении значений трех стратегий, именно "mean" подходил лучше всего.

In [23]:
```
df.columns
```

Out[23]:
```
Index(['Date', 'Team', 'Opponent', 'Goal Scored', 'Ball Possession %',
       'Attempts', 'On-Target', 'Off-Target', 'Blocked', 'Corners', 'Offsides',
       'Free Kicks', 'Saves', 'Pass Accuracy %', 'Passes',
       'Distance Covered (Kms)', 'Fouls Committed', 'Yellow Card',
       'Yellow & Red', 'Red', 'Man of the Match', '1st Goal', 'Round', 'PSO',
       'Goals in PSO', 'Own goals', 'Own goal Time'],
      dtype='object')
```

In [32]:
```
sns.heatmap(df.corr())
```

Out[32]: <AxesSubplot:>



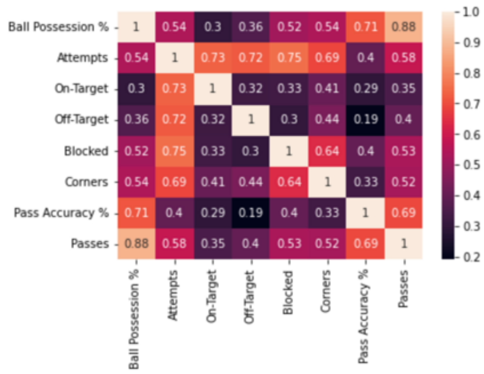
Уберем признаки, которые практически не влияют на другие признаки.

In [33]:
```
df.pop('Goal Scored')
df.pop('Offsides')
df.pop('Free Kicks')
df.pop('Saves')
df.pop('Distance Covered (Kms)')
df.pop('Fouls Committed')
df.pop('Yellow Card')
df.pop('Yellow & Red')
df.pop('Red')
df.pop('1st Goal')
df.pop('Own goals')
df.pop('Goals in PSO')
df.pop('Own goal Time')
```

Out[33]:
```
0       NaN
1       NaN
2       NaN
3       NaN
4      90.0
       ...
123     NaN
124     NaN
125     NaN
126    18.0
127     NaN
Name: Own goal Time, Length: 128, dtype: float64
```

In [35]:
```python
sns.heatmap(df.corr(), annot=True)
```

Out[35]: <AxesSubplot:>



Если брать целевым признак "attempts", то можно выбрать признаки, хорошо коррелирующие с ним, например "On-Target", "Off-Target","Blocked". Стоит отметить, что данные признаки низко коррелируют между собой, что также хорошо для их выбора.