МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Домашнее задание
по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬ: _Бабин В.Е.________
ФИО

группа ИУ5-22М ____________________
подпись

"__"_________2020 г.

ПРЕПОДАВАТЕЛЬ: ____________________
ФИО

____
подпись

"__"_________2020 г.

Москва - 2020

Домашнее задание по дисциплине направлено на решение комплексной задачи машинного обучения. Домашнее задание включает выполнение следующих шагов:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

dz

May 23, 2020

[243]:
```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

Will solve regression task for Overall target attribute

[216]:
```
target_attr = 'Overall'
```

[217]:
```
# dataset about FIFA 2019 statistics
data = pd.read_csv('data-fifa19.csv')
data.head()
```

[217]:
```
   Unnamed: 0      ID               Name  Age  \
0           0  158023           L. Messi   31
1           1   20801  Cristiano Ronaldo   33
2           2  190871          Neymar Jr   26
3           3  193080             De Gea   27
4           4  192985       K. De Bruyne   27

                                            Photo Nationality  \
0  https://cdn.sofifa.org/players/4/19/158023.png   Argentina
1   https://cdn.sofifa.org/players/4/19/20801.png    Portugal
2  https://cdn.sofifa.org/players/4/19/190871.png      Brazil
3  https://cdn.sofifa.org/players/4/19/193080.png       Spain
```

```
4  https://cdn.sofifa.org/players/4/19/192985.png     Belgium

                               Flag  Overall  Potential  \
0  https://cdn.sofifa.org/flags/52.png       94         94
1  https://cdn.sofifa.org/flags/38.png       94         94
2  https://cdn.sofifa.org/flags/54.png       92         93
3  https://cdn.sofifa.org/flags/45.png       91         93
4   https://cdn.sofifa.org/flags/7.png       91         92

                  Club  … Composure Marking StandingTackle  SlidingTackle  \
0          FC Barcelona  …       96.0    33.0           28.0           26.0
1              Juventus  …       95.0    28.0           31.0           23.0
2  Paris Saint-Germain  …       94.0    27.0           24.0           33.0
3    Manchester United  …       68.0    15.0           21.0           13.0
4      Manchester City  …       88.0    68.0           58.0           51.0

  GKDiving  GKHandling  GKKicking  GKPositioning GKReflexes Release Clause
0       6.0        11.0       15.0           14.0        8.0        €226.5M
1       7.0        11.0       15.0           14.0       11.0        €127.1M
2       9.0         9.0       15.0           15.0       11.0        €228.1M
3      90.0        85.0       87.0           88.0       94.0        €138.6M
4      15.0        13.0        5.0           10.0       13.0        €196.4M

[5 rows x 89 columns]
```

[218]:
```
# size of the dataset
data.shape
```

[218]:
```
(18207, 89)
```

Analysis and filling in data gaps

[219]:
```
# let`s take a look if our dataset has null values
for col in data.columns:
    print('{} - {}'.format(col, data[col].isnull().sum()))
```

```
Unnamed: 0 - 0
ID - 0
Name - 0
Age - 0
Photo - 0
Nationality - 0
Flag - 0
Overall - 0
Potential - 0
Club - 241
Club Logo - 0
Value - 0
```

Wage - 0
Special - 0
Preferred Foot - 48
International Reputation - 48
Weak Foot - 48
Skill Moves - 48
Work Rate - 48
Body Type - 48
Real Face - 48
Position - 60
Jersey Number - 60
Joined - 1553
Loaned From - 16943
Contract Valid Until - 289
Height - 48
Weight - 48
LS - 2085
ST - 2085
RS - 2085
LW - 2085
LF - 2085
CF - 2085
RF - 2085
RW - 2085
LAM - 2085
CAM - 2085
RAM - 2085
LM - 2085
LCM - 2085
CM - 2085
RCM - 2085
RM - 2085
LWB - 2085
LDM - 2085
CDM - 2085
RDM - 2085
RWB - 2085
LB - 2085
LCB - 2085
CB - 2085
RCB - 2085
RB - 2085
Crossing - 48
Finishing - 48
HeadingAccuracy - 48
ShortPassing - 48
Volleys - 48
Dribbling - 48

```
Curve - 48
FKAccuracy - 48
LongPassing - 48
BallControl - 48
Acceleration - 48
SprintSpeed - 48
Agility - 48
Reactions - 48
Balance - 48
ShotPower - 48
Jumping - 48
Stamina - 48
Strength - 48
LongShots - 48
Aggression - 48
Interceptions - 48
Positioning - 48
Vision - 48
Penalties - 48
Composure - 48
Marking - 48
StandingTackle - 48
SlidingTackle - 48
GKDiving - 48
GKHandling - 48
GKKicking - 48
GKPositioning - 48
GKReflexes - 48
Release Clause - 1564
```

[220]:
```
# to fill null values let`s take a look at column types
for col in data.columns:
    print('{} - {}'.format(col, data[col].dtypes))
```

```
Unnamed: 0 - int64
ID - int64
Name - object
Age - int64
Photo - object
Nationality - object
Flag - object
Overall - int64
Potential - int64
Club - object
Club Logo - object
Value - object
Wage - object
Special - int64
```

```
Preferred Foot - object
International Reputation - float64
Weak Foot - float64
Skill Moves - float64
Work Rate - object
Body Type - object
Real Face - object
Position - object
Jersey Number - float64
Joined - object
Loaned From - object
Contract Valid Until - object
Height - object
Weight - object
LS - object
ST - object
RS - object
LW - object
LF - object
CF - object
RF - object
RW - object
LAM - object
CAM - object
RAM - object
LM - object
LCM - object
CM - object
RCM - object
RM - object
LWB - object
LDM - object
CDM - object
RDM - object
RWB - object
LB - object
LCB - object
CB - object
RCB - object
RB - object
Crossing - float64
Finishing - float64
HeadingAccuracy - float64
ShortPassing - float64
Volleys - float64
Dribbling - float64
Curve - float64
FKAccuracy - float64
```

```
LongPassing - float64
BallControl - float64
Acceleration - float64
SprintSpeed - float64
Agility - float64
Reactions - float64
Balance - float64
ShotPower - float64
Jumping - float64
Stamina - float64
Strength - float64
LongShots - float64
Aggression - float64
Interceptions - float64
Positioning - float64
Vision - float64
Penalties - float64
Composure - float64
Marking - float64
StandingTackle - float64
SlidingTackle - float64
GKDiving - float64
GKHandling - float64
GKKicking - float64
GKPositioning - float64
GKReflexes - float64
Release Clause - object
```

[221]:

```python
# let`s delete object columns from dataset as it`s not necessary
obj_cols = []

for column in data.columns:
    dt = str(data[column].dtype)
    if dt == 'object':
        obj_cols.append(column)
print(obj_cols)
```

```
['Name', 'Photo', 'Nationality', 'Flag', 'Club', 'Club Logo', 'Value', 'Wage',
'Preferred Foot', 'Work Rate', 'Body Type', 'Real Face', 'Position', 'Joined',
'Loaned From', 'Contract Valid Until', 'Height', 'Weight', 'LS', 'ST', 'RS',
'LW', 'LF', 'CF', 'RF', 'RW', 'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM',
'RM', 'LWB', 'LDM', 'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB',
'Release Clause']
```

[222]:

```python
data.drop(obj_cols, axis='columns', inplace=True)
data.shape
```

[222]: (18207, 44)

[223]:
```
# we`ll impute null columns with most frequent values
cols_to_impute = list()

for col in data.columns:
    if data[col].isnull().sum() != 0:
        imputation = SimpleImputer(missing_values=np.nan,␣
 ↪strategy='most_frequent')
        col_imputed = imputation.fit_transform(data[[col]])
        data[col] = pd.DataFrame(col_imputed)
```

[224]:
```
# let`s check if we don`t have nulls now
data.isnull().sum()
```

[224]:
```
Unnamed: 0                  0
ID                          0
Age                         0
Overall                     0
Potential                   0
Special                     0
International Reputation    0
Weak Foot                   0
Skill Moves                 0
Jersey Number               0
Crossing                    0
Finishing                   0
HeadingAccuracy             0
ShortPassing                0
Volleys                     0
Dribbling                   0
Curve                       0
FKAccuracy                  0
LongPassing                 0
BallControl                 0
Acceleration                0
SprintSpeed                 0
Agility                     0
Reactions                   0
Balance                     0
ShotPower                   0
Jumping                     0
Stamina                     0
Strength                    0
LongShots                   0
Aggression                  0
Interceptions               0
```

```
Positioning            0
Vision                 0
Penalties              0
Composure              0
Marking                0
StandingTackle         0
SlidingTackle          0
GKDiving               0
GKHandling             0
GKKicking              0
GKPositioning          0
GKReflexes             0
dtype: int64
```

amount of columns still is too big

we'll delete some of them after correllation analysis

will delete those which correlates the less

Correlation analysis

[227]:
```
data.corr()
```

[227]:

	Unnamed: 0	ID	Age	Overall	Potential	\
Unnamed: 0	1.000000	0.415757	-0.454846	-0.972791	-0.633395	
ID	0.415757	1.000000	-0.739208	-0.417025	0.047074	
Age	-0.454846	-0.739208	1.000000	0.452350	-0.253312	
Overall	-0.972791	-0.417025	0.452350	1.000000	0.660939	
Potential	-0.633395	0.047074	-0.253312	0.660939	1.000000	
Special	-0.596508	-0.231352	0.236695	0.606960	0.383727	
International Reputation	-0.413535	-0.355900	0.253457	0.499654	0.372887	
Weak Foot	-0.203689	-0.075642	0.059790	0.211779	0.161922	
Skill Moves	-0.416201	-0.057126	0.027641	0.414906	0.354516	
Jersey Number	0.211294	0.181202	-0.240711	-0.216928	-0.008466	
Crossing	-0.388117	-0.131339	0.130268	0.393463	0.244481	
Finishing	-0.323755	-0.081781	0.068498	0.331139	0.241595	
HeadingAccuracy	-0.336754	-0.106465	0.146965	0.340027	0.199995	
ShortPassing	-0.491249	-0.135864	0.132689	0.501628	0.367819	
Volleys	-0.383189	-0.159481	0.142258	0.390525	0.253782	
Dribbling	-0.362752	-0.030010	0.010120	0.371400	0.313533	
Curve	-0.414615	-0.168861	0.143000	0.418138	0.278220	
FKAccuracy	-0.395207	-0.199338	0.193241	0.396773	0.230256	
LongPassing	-0.475610	-0.186069	0.180966	0.482453	0.319583	
BallControl	-0.448360	-0.099793	0.084823	0.459228	0.352971	
Acceleration	-0.184243	0.133243	-0.158481	0.196273	0.233627	
SprintSpeed	-0.198153	0.132407	-0.151508	0.210167	0.235934	
Agility	-0.255392	-0.019680	-0.019391	0.264294	0.221328	
Reactions	-0.830600	-0.407836	0.452489	0.848915	0.511816	

Balance	-0.096474	0.048578	-0.089780	0.102635	0.137232
ShotPower	-0.437835	-0.165362	0.156601	0.439414	0.286241
Jumping	-0.260438	-0.168866	0.176888	0.263570	0.108146
Stamina	-0.357484	-0.053636	0.097645	0.364944	0.201624
Strength	-0.342088	-0.259292	0.332369	0.348785	0.075226
LongShots	-0.416009	-0.160886	0.154792	0.419375	0.264977
Aggression	-0.395497	-0.227641	0.264743	0.394278	0.169817
Interceptions	-0.317126	-0.159830	0.197417	0.319739	0.153118
Positioning	-0.350592	-0.087946	0.082299	0.355569	0.244362
Vision	-0.489143	-0.214676	0.187152	0.498050	0.346891
Penalties	-0.338068	-0.140618	0.139376	0.341602	0.224329
Composure	-0.715321	-0.383926	0.390544	0.727088	0.439043
Marking	-0.279113	-0.109606	0.142526	0.285174	0.161215
StandingTackle	-0.246363	-0.085217	0.119432	0.250899	0.141567
SlidingTackle	-0.218080	-0.067794	0.102834	0.221286	0.127190
GKDiving	0.026709	-0.105737	0.101159	-0.025127	-0.052404
GKHandling	0.026209	-0.111229	0.106299	-0.024432	-0.053844
GKKicking	0.030090	-0.106674	0.104848	-0.028940	-0.058465
GKPositioning	0.019039	-0.118319	0.116268	-0.017055	-0.051770
GKReflexes	0.024804	-0.105864	0.103197	-0.022655	-0.052523

	Special	International Reputation	Weak Foot	\
Unnamed: 0	-0.596508	-0.413535	-0.203689	
ID	-0.231352	-0.355900	-0.075642	
Age	0.236695	0.253457	0.059790	
Overall	0.606960	0.499654	0.211779	
Potential	0.383727	0.372887	0.161922	
Special	1.000000	0.292186	0.341720	
International Reputation	0.292186	1.000000	0.128241	
Weak Foot	0.341720	0.128241	1.000000	
Skill Moves	0.763113	0.208429	0.340515	
Jersey Number	-0.133015	-0.076535	-0.035681	
Crossing	0.865412	0.191131	0.307881	
Finishing	0.723414	0.177775	0.357356	
HeadingAccuracy	0.644019	0.157195	0.183280	
ShortPassing	0.906170	0.242461	0.322151	
Volleys	0.773497	0.242763	0.357353	
Dribbling	0.873609	0.178626	0.352654	
Curve	0.851047	0.233103	0.345431	
FKAccuracy	0.806181	0.223577	0.330458	
LongPassing	0.845424	0.238925	0.277165	
BallControl	0.911501	0.217576	0.356389	
Acceleration	0.653960	0.044086	0.261465	
SprintSpeed	0.645645	0.043893	0.248851	
Agility	0.699268	0.100615	0.302086	
Reactions	0.596768	0.445227	0.201381	
Balance	0.586446	0.049849	0.254052	

ShotPower	0.834188	0.227038	0.332773
Jumping	0.321531	0.120575	0.069823
Stamina	0.792320	0.094531	0.232128
Strength	0.192845	0.131096	-0.008424
LongShots	0.839157	0.213362	0.355915
Aggression	0.665608	0.172847	0.131585
Interceptions	0.560845	0.128919	0.053214
Positioning	0.823728	0.182630	0.346903
Vision	0.761540	0.284283	0.337915
Penalties	0.734335	0.218753	0.330180
Composure	0.752046	0.392647	0.278149
Marking	0.561171	0.114649	0.065772
StandingTackle	0.537840	0.092109	0.042784
SlidingTackle	0.506155	0.078525	0.026246
GKDiving	-0.674051	0.004893	-0.231934
GKHandling	-0.673161	0.004227	-0.233131
GKKicking	-0.669902	0.000845	-0.229427
GKPositioning	-0.667814	0.007186	-0.231333
GKReflexes	-0.672778	0.003726	-0.232608

	Skill Moves	Jersey Number	…	Penalties	\
Unnamed: 0	-0.416201	0.211294	…	-0.338068	
ID	-0.057126	0.181202	…	-0.140618	
Age	0.027641	-0.240711	…	0.139376	
Overall	0.414906	-0.216928	…	0.341602	
Potential	0.354516	-0.008466	…	0.224329	
Special	0.763113	-0.133015	…	0.734335	
International Reputation	0.208429	-0.076535	…	0.218753	
Weak Foot	0.340515	-0.035681	…	0.330180	
Skill Moves	1.000000	-0.034060	…	0.690464	
Jersey Number	-0.034060	1.000000	…	-0.027658	
Crossing	0.739536	-0.077553	…	0.644985	
Finishing	0.742015	-0.007868	…	0.836942	
HeadingAccuracy	0.442396	-0.092092	…	0.551667	
ShortPassing	0.729563	-0.100341	…	0.675686	
Volleys	0.744304	-0.027223	…	0.828867	
Dribbling	0.838700	-0.028693	…	0.769059	
Curve	0.769727	-0.056247	…	0.751089	
FKAccuracy	0.700917	-0.068552	…	0.734417	
LongPassing	0.621091	-0.117916	…	0.541587	
BallControl	0.817134	-0.073628	…	0.769343	
Acceleration	0.651716	-0.005010	…	0.532637	
SprintSpeed	0.623588	-0.015514	…	0.520868	
Agility	0.681093	-0.034813	…	0.565886	
Reactions	0.376428	-0.192839	…	0.345854	
Balance	0.577863	0.007338	…	0.482540	
ShotPower	0.716605	-0.054921	…	0.794179	

Jumping	0.106973	-0.104745	…	0.133007
Stamina	0.569607	-0.128167	…	0.516150
Strength	-0.041730	-0.158181	…	0.054357
LongShots	0.751598	-0.047158	…	0.811616
Aggression	0.346902	-0.147373	…	0.335631
Interceptions	0.208459	-0.159479	…	0.110307
Positioning	0.780320	-0.026035	…	0.800790
Vision	0.673351	-0.078212	…	0.632603
Penalties	0.690464	-0.027658	…	1.000000
Composure	0.586499	-0.167379	…	0.551684
Marking	0.240421	-0.143256	…	0.151824
StandingTackle	0.209193	-0.134473	…	0.101314
SlidingTackle	0.177449	-0.125654	…	0.066180
GKDiving	-0.620681	0.005757	…	-0.619544
GKHandling	-0.618976	0.002358	…	-0.618584
GKKicking	-0.616428	0.001692	…	-0.613759
GKPositioning	-0.618080	-0.001985	…	-0.616694
GKReflexes	-0.621153	0.004009	…	-0.618721

	Composure	Marking	StandingTackle	SlidingTackle \
Unnamed: 0	-0.715321	-0.279113	-0.246363	-0.218080
ID	-0.383926	-0.109606	-0.085217	-0.067794
Age	0.390544	0.142526	0.119432	0.102834
Overall	0.727088	0.285174	0.250899	0.221286
Potential	0.439043	0.161215	0.141567	0.127190
Special	0.752046	0.561171	0.537840	0.506155
International Reputation	0.392647	0.114649	0.092109	0.078525
Weak Foot	0.278149	0.065772	0.042784	0.026246
Skill Moves	0.586499	0.240421	0.209193	0.177449
Jersey Number	-0.167379	-0.143256	-0.134473	-0.125654
Crossing	0.575305	0.443726	0.429793	0.410752
Finishing	0.533317	0.025265	-0.031557	-0.070428
HeadingAccuracy	0.507229	0.583279	0.561186	0.533817
ShortPassing	0.685120	0.559787	0.541334	0.508892
Volleys	0.595284	0.121414	0.073474	0.036120
Dribbling	0.597465	0.336567	0.301900	0.274598
Curve	0.616422	0.290232	0.262425	0.233772
FKAccuracy	0.585092	0.297731	0.278780	0.247610
LongPassing	0.645662	0.587526	0.587942	0.562747
BallControl	0.674852	0.453037	0.417971	0.385227
Acceleration	0.347473	0.195737	0.163482	0.158019
SprintSpeed	0.351647	0.212823	0.178532	0.172287
Agility	0.432545	0.167525	0.129745	0.117197
Reactions	0.685543	0.284000	0.255899	0.228850
Balance	0.310813	0.179061	0.154518	0.152912
ShotPower	0.634268	0.297801	0.257601	0.221383
Jumping	0.252420	0.279682	0.261265	0.260840

Stamina	0.523134	0.587902	0.570121	0.544822
Strength	0.280566	0.333495	0.332303	0.305027
LongShots	0.615973	0.216325	0.173456	0.134680
Aggression	0.515743	0.724193	0.744416	0.721624
Interceptions	0.397387	0.888476	0.941553	0.928388
Positioning	0.580486	0.203136	0.158793	0.124937
Vision	0.636281	0.177188	0.147026	0.113781
Penalties	0.551684	0.151824	0.101314	0.066180
Composure	1.000000	0.384066	0.351652	0.317479
Marking	0.384066	1.000000	0.906623	0.896023
StandingTackle	0.351652	0.906623	1.000000	0.974695
SlidingTackle	0.317479	0.896023	0.974695	1.000000
GKDiving	-0.378776	-0.551329	-0.531408	-0.509767
GKHandling	-0.375760	-0.552499	-0.532401	-0.510863
GKKicking	-0.374938	-0.549587	-0.531118	-0.509457
GKPositioning	-0.370276	-0.546906	-0.528033	-0.506064
GKReflexes	-0.377666	-0.551522	-0.531709	-0.509692

	GKDiving	GKHandling	GKKicking	GKPositioning	\
Unnamed: 0	0.026709	0.026209	0.030090	0.019039	
ID	-0.105737	-0.111229	-0.106674	-0.118319	
Age	0.101159	0.106299	0.104848	0.116268	
Overall	-0.025127	-0.024432	-0.028940	-0.017055	
Potential	-0.052404	-0.053844	-0.058465	-0.051770	
Special	-0.674051	-0.673161	-0.669902	-0.667814	
International Reputation	0.004893	0.004227	0.000845	0.007186	
Weak Foot	-0.231934	-0.233131	-0.229427	-0.231333	
Skill Moves	-0.620681	-0.618976	-0.616428	-0.618080	
Jersey Number	0.005757	0.002358	0.001692	-0.001985	
Crossing	-0.663313	-0.660346	-0.659773	-0.660309	
Finishing	-0.589071	-0.587355	-0.583336	-0.585061	
HeadingAccuracy	-0.750498	-0.749968	-0.746495	-0.744523	
ShortPassing	-0.729895	-0.728121	-0.724438	-0.723881	
Volleys	-0.590973	-0.588808	-0.585045	-0.586271	
Dribbling	-0.754768	-0.753287	-0.749854	-0.751454	
Curve	-0.606575	-0.603339	-0.600338	-0.603734	
FKAccuracy	-0.556366	-0.553488	-0.549828	-0.552488	
LongPassing	-0.597123	-0.595203	-0.591526	-0.591764	
BallControl	-0.788543	-0.786881	-0.783460	-0.783691	
Acceleration	-0.593132	-0.594979	-0.592207	-0.592257	
SprintSpeed	-0.597737	-0.599764	-0.597380	-0.596569	
Agility	-0.527917	-0.528621	-0.527260	-0.527122	
Reactions	-0.063388	-0.062268	-0.066148	-0.055359	
Balance	-0.504881	-0.506236	-0.504065	-0.503787	
ShotPower	-0.654390	-0.654244	-0.649386	-0.651553	
Jumping	-0.193129	-0.194026	-0.195501	-0.189411	
Stamina	-0.701550	-0.698641	-0.696789	-0.696159	

Strength	-0.111245	-0.109846	-0.110381	-0.104064
LongShots	-0.612675	-0.610932	-0.606012	-0.607393
Aggression	-0.576128	-0.576319	-0.573704	-0.571406
Interceptions	-0.486032	-0.486618	-0.485502	-0.481574
Positioning	-0.679645	-0.677830	-0.674462	-0.675700
Vision	-0.382143	-0.378007	-0.374872	-0.375974
Penalties	-0.619544	-0.618584	-0.613759	-0.616694
Composure	-0.378776	-0.375760	-0.374938	-0.370276
Marking	-0.551329	-0.552499	-0.549587	-0.546906
StandingTackle	-0.531408	-0.532401	-0.531118	-0.528033
SlidingTackle	-0.509767	-0.510863	-0.509457	-0.506064
GKDiving	1.000000	0.970279	0.965628	0.969863
GKHandling	0.970279	1.000000	0.965229	0.969419
GKKicking	0.965628	0.965229	1.000000	0.964328
GKPositioning	0.969863	0.969419	0.964328	1.000000
GKReflexes	0.973317	0.970275	0.966328	0.970141

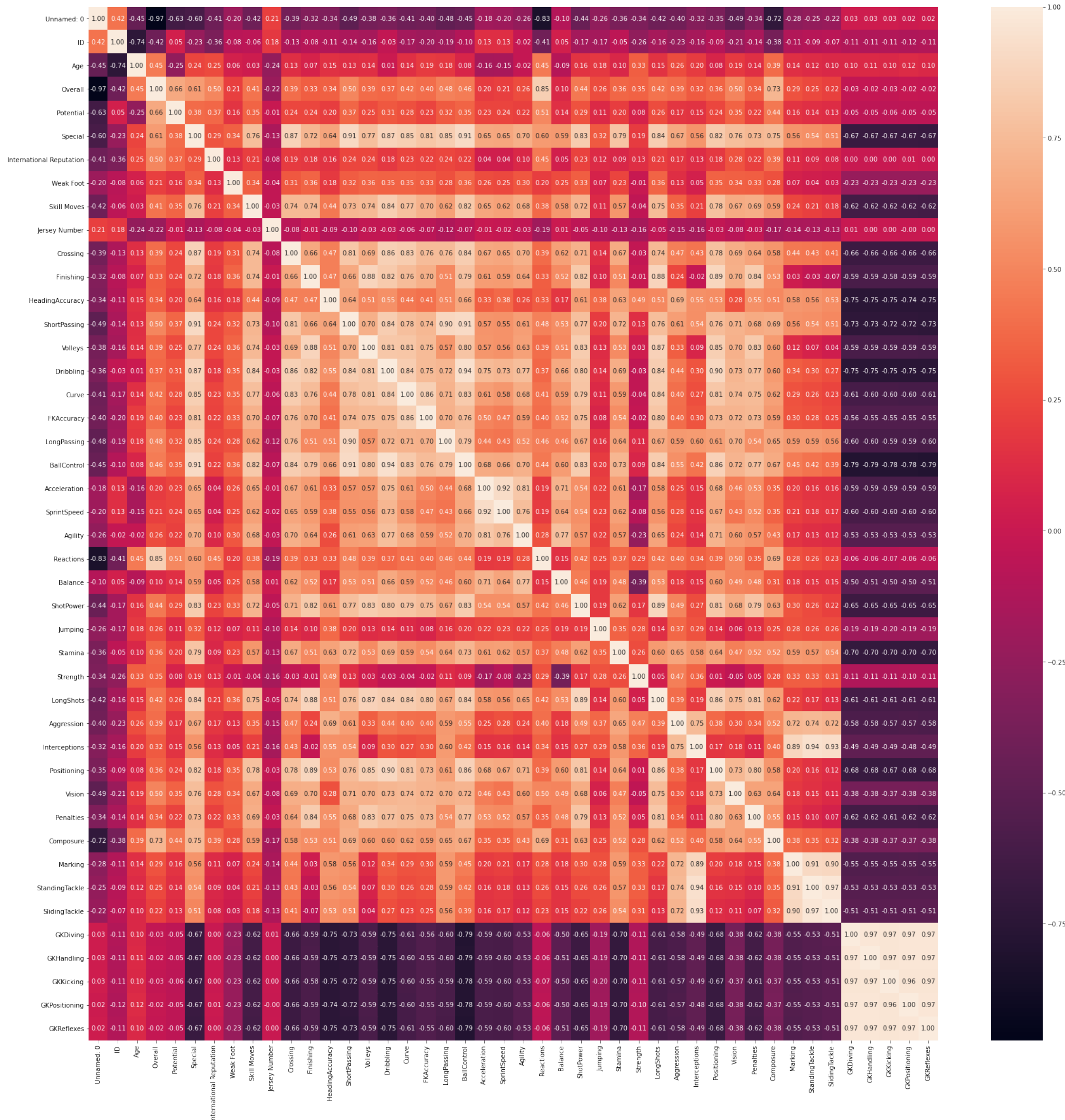
	GKReflexes
Unnamed: 0	0.024804
ID	-0.105864
Age	0.103197
Overall	-0.022655
Potential	-0.052523
Special	-0.672778
International Reputation	0.003726
Weak Foot	-0.232608
Skill Moves	-0.621153
Jersey Number	0.004009
Crossing	-0.662686
Finishing	-0.587118
HeadingAccuracy	-0.748975
ShortPassing	-0.728817
Volleys	-0.588809
Dribbling	-0.754445
Curve	-0.605152
FKAccuracy	-0.554767
LongPassing	-0.596086
BallControl	-0.788021
Acceleration	-0.593314
SprintSpeed	-0.597908
Agility	-0.529037
Reactions	-0.060286
Balance	-0.506107
ShotPower	-0.653616
Jumping	-0.192380
Stamina	-0.699754
Strength	-0.107682

```
LongShots                    -0.610276
Aggression                   -0.575345
Interceptions                -0.486324
Positioning                  -0.678712
Vision                       -0.381355
Penalties                    -0.618721
Composure                    -0.377666
Marking                      -0.551522
StandingTackle               -0.531709
SlidingTackle                -0.509692
GKDiving                      0.973317
GKHandling                    0.970275
GKKicking                     0.966328
GKPositioning                 0.970141
GKReflexes                    1.000000

[44 rows x 44 columns]
```

[228]:
```python
fig, ax = plt.subplots(figsize=(30,30))
sns.heatmap(data.corr(), annot=True, fmt=".2f")
```

[228]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x263e167e5f8>
```



Now we see columns that doesn‘t correlates with target attribute so we can delete some of them.

Correlation means influence. It mean that the column that correlates (has a large value of the correlation coefficient) with the target attribute will have a strong influence on it. And it matters when we solving the regression problem.

[232]:
```
columns_to_drop = ['GKDiving', 'GKHandling', 'GKKicking', 'GKPositioning',␣
↪'GKReflexes',
                   'Unnamed: 0', 'Acceleration', 'SprintSpeed', 'Agility',␣
↪'Balance',
                   'Marking', 'StandingTackle', 'SlidingTackle', 'Jersey␣
↪Number']
```

```
datac = data.copy()
datac.drop(columns_to_drop, axis='columns', inplace=True)
datac.head()
```

[232]:

```
       ID  Age  Overall  Potential  Special  International Reputation  \
0  158023   31       94         94     2202                       5.0
1   20801   33       94         94     2228                       5.0
2  190871   26       92         93     2143                       5.0
3  193080   27       91         93     1471                       4.0
4  192985   27       91         92     2281                       4.0

   Weak Foot  Skill Moves  Crossing  Finishing  …  Jumping  Stamina  \
0        4.0          4.0      84.0       95.0  …     68.0     72.0
1        4.0          5.0      84.0       94.0  …     95.0     88.0
2        5.0          5.0      79.0       87.0  …     61.0     81.0
3        3.0          1.0      17.0       13.0  …     67.0     43.0
4        5.0          4.0      93.0       82.0  …     63.0     90.0

   Strength  LongShots  Aggression  Interceptions  Positioning  Vision  \
0      59.0       94.0        48.0           22.0         94.0    94.0
1      79.0       93.0        63.0           29.0         95.0    82.0
2      49.0       82.0        56.0           36.0         89.0    87.0
3      64.0       12.0        38.0           30.0         12.0    68.0
4      75.0       91.0        76.0           61.0         87.0    94.0

   Penalties  Composure
0       75.0       96.0
1       85.0       95.0
2       81.0       94.0
3       40.0       68.0
4       79.0       88.0

[5 rows x 30 columns]
```

[234]:

```
# also now we see that target attribute correlates the most with next␣
↪attributes:
most_corr = ['Reactions', 'Composure', 'Potential', 'Special']
```

Conducting exploratory data analysis. Plotting the graphs needed to understand the data structure.

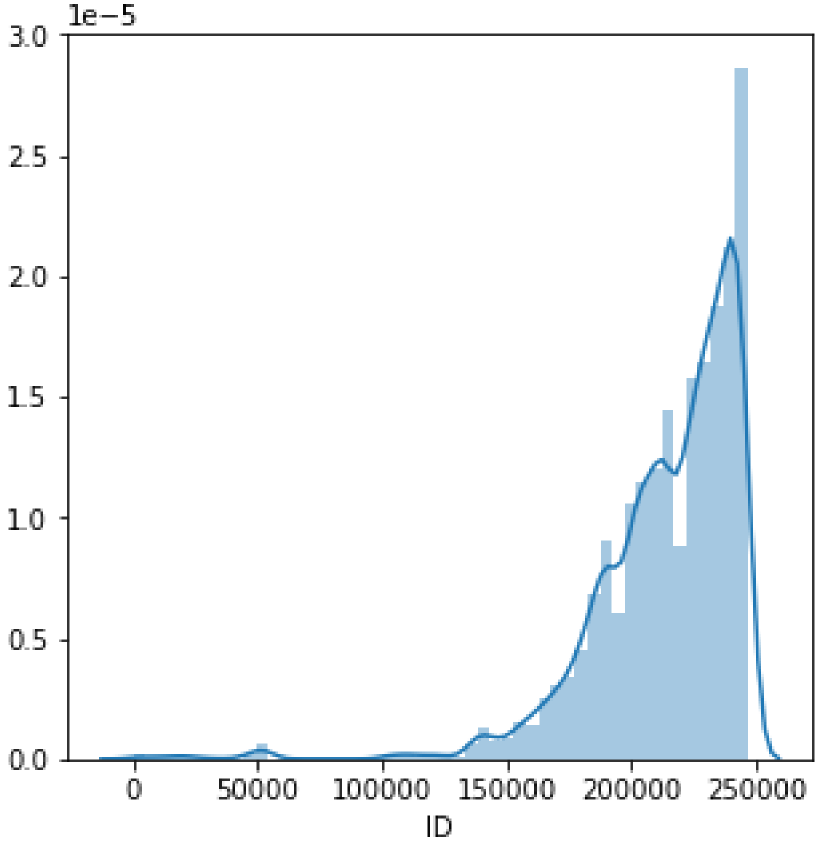
[233]:

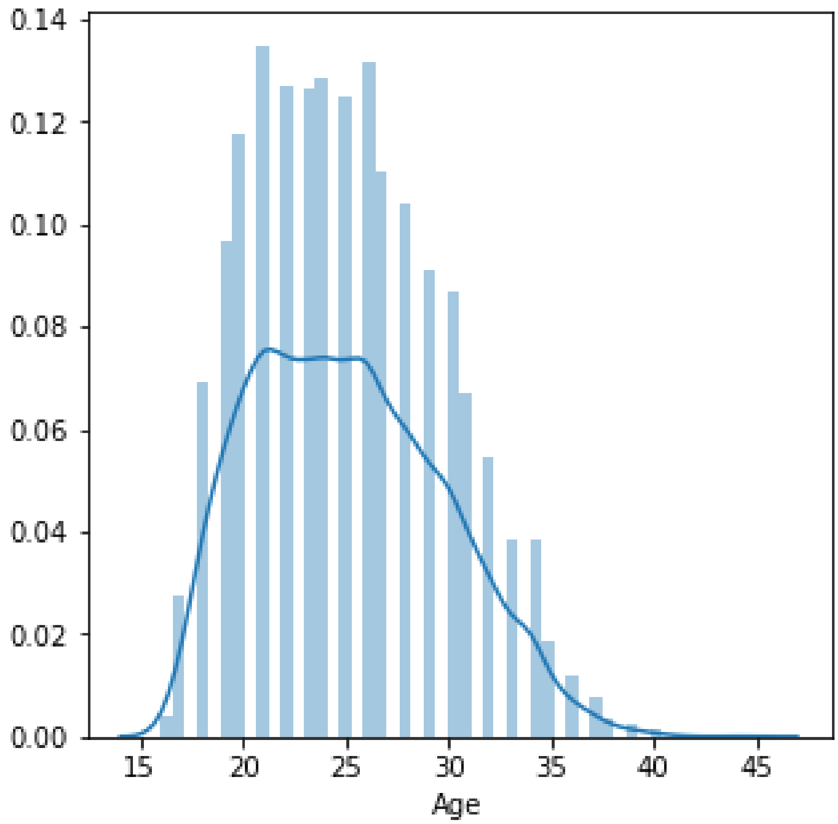
```
# distribution of attrs
for col in datac.columns:
    fig, ax = plt.subplots(figsize=(5,5))
    sns.distplot(datac[col])
```

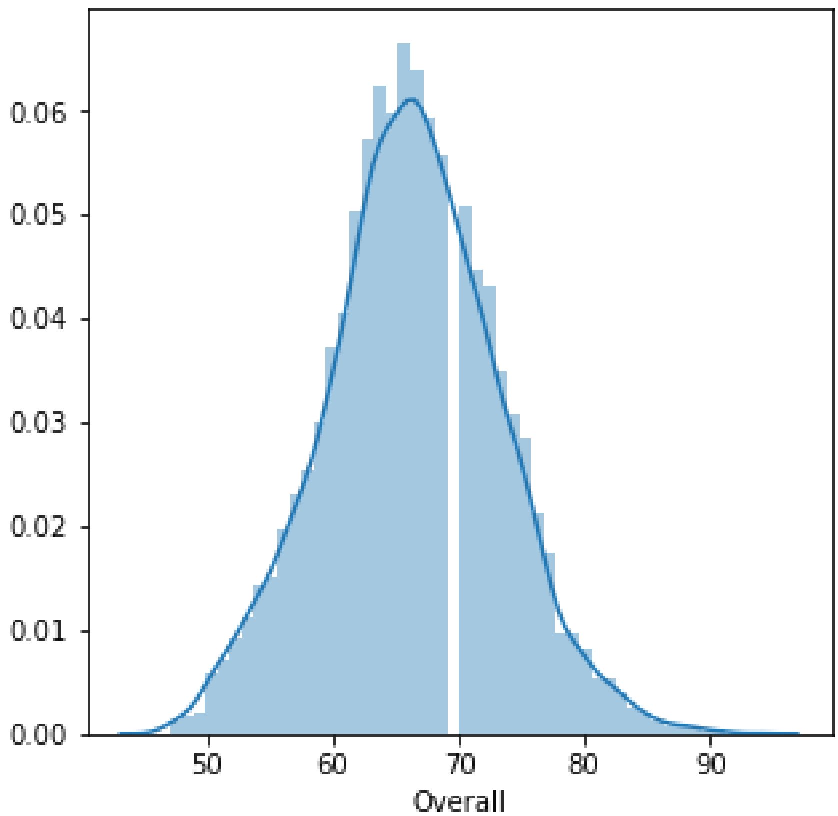
```
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
```

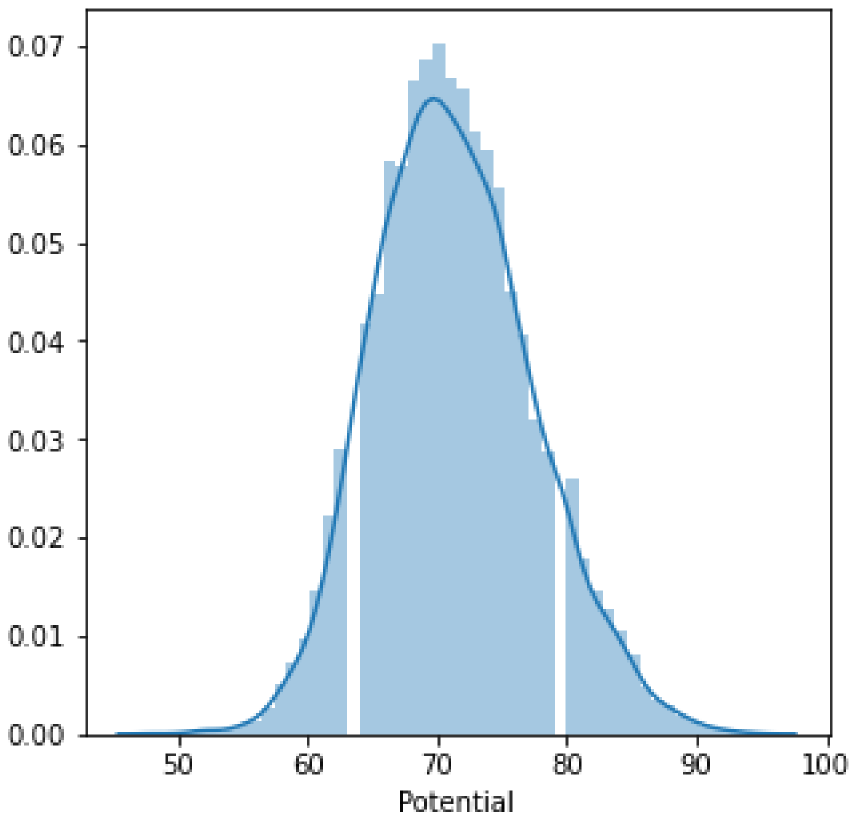
```
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
```

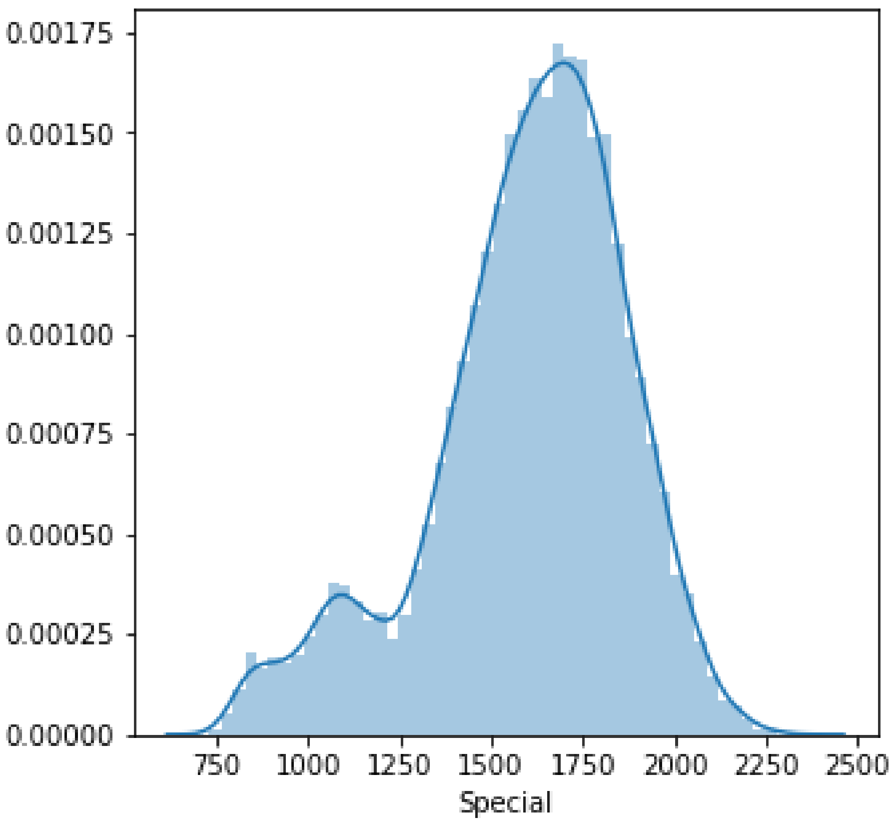
```
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
c:\users\viktorb.adft\virtualenv\tensorflow\lib\site-
packages\ipykernel_launcher.py:3: RuntimeWarning: More than 20 figures have been
opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  This is separate from the ipykernel package so we can avoid doing imports
until
```

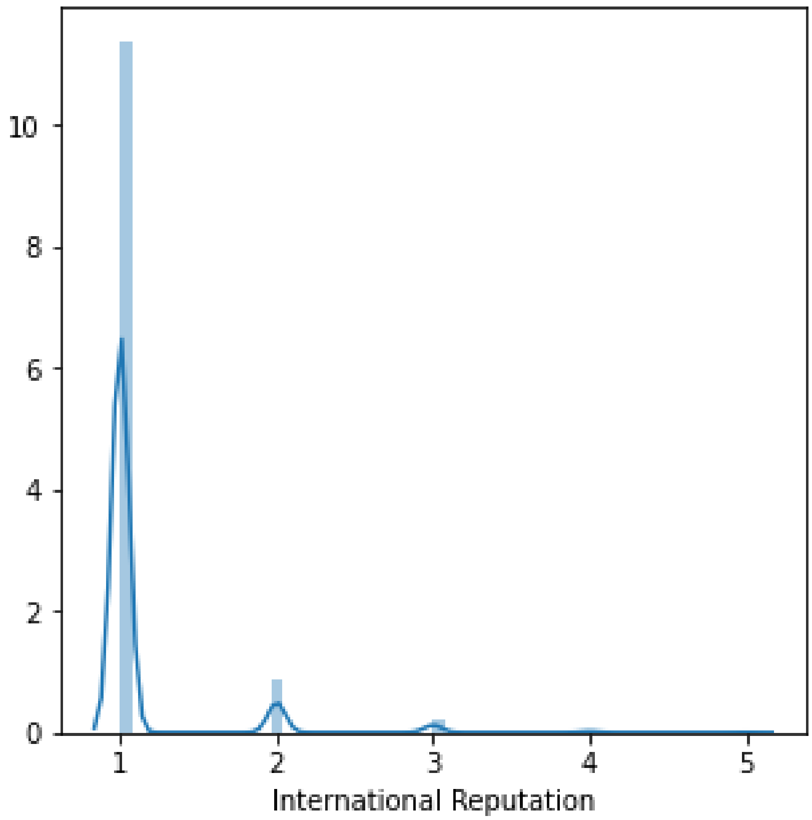

1e−5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50000
100000
150000
200000
250000
ID


0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
15
20
25
30
35
40
45
Age

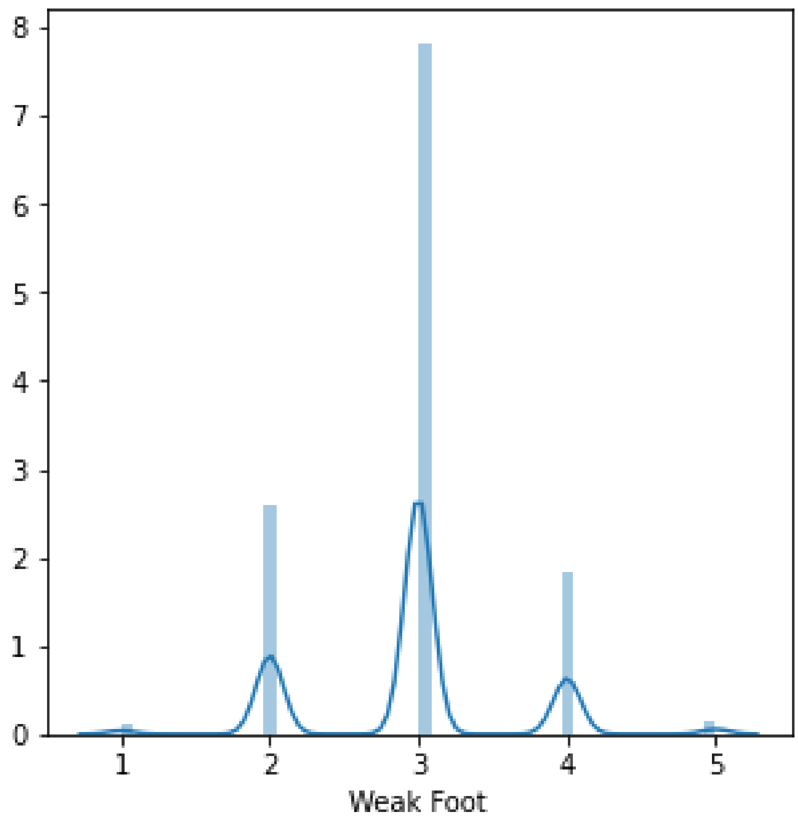

0.06
0.05
0.04
0.03
0.02
0.01
0.00
50
60
70
80
90
Overall


0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
50
60
70
80
90
100
Potential

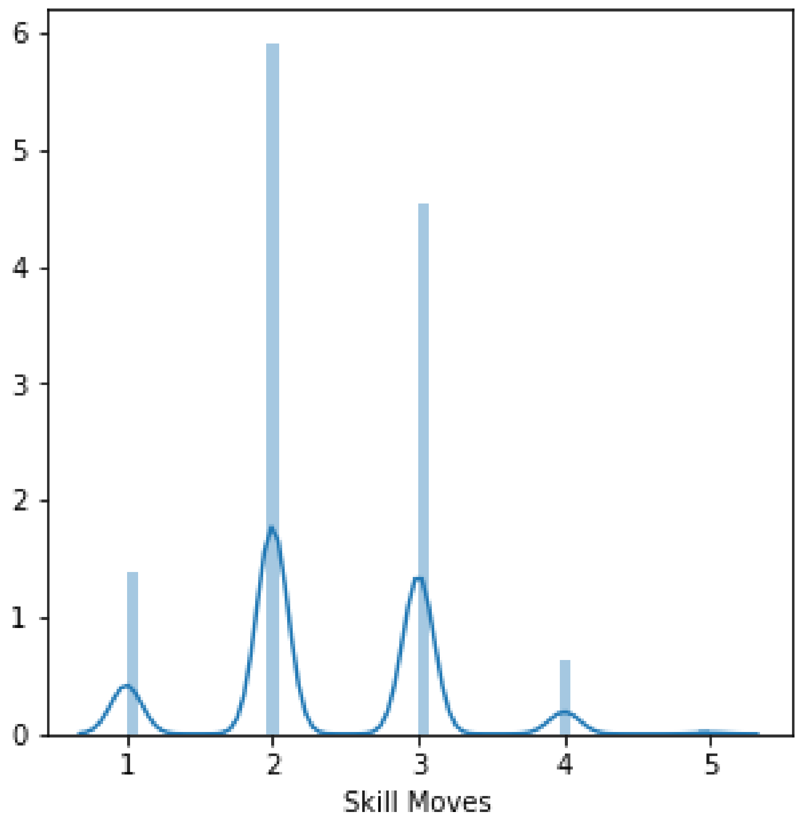

0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
750
1000
1250
1500
1750
2000
2250
2500
Special



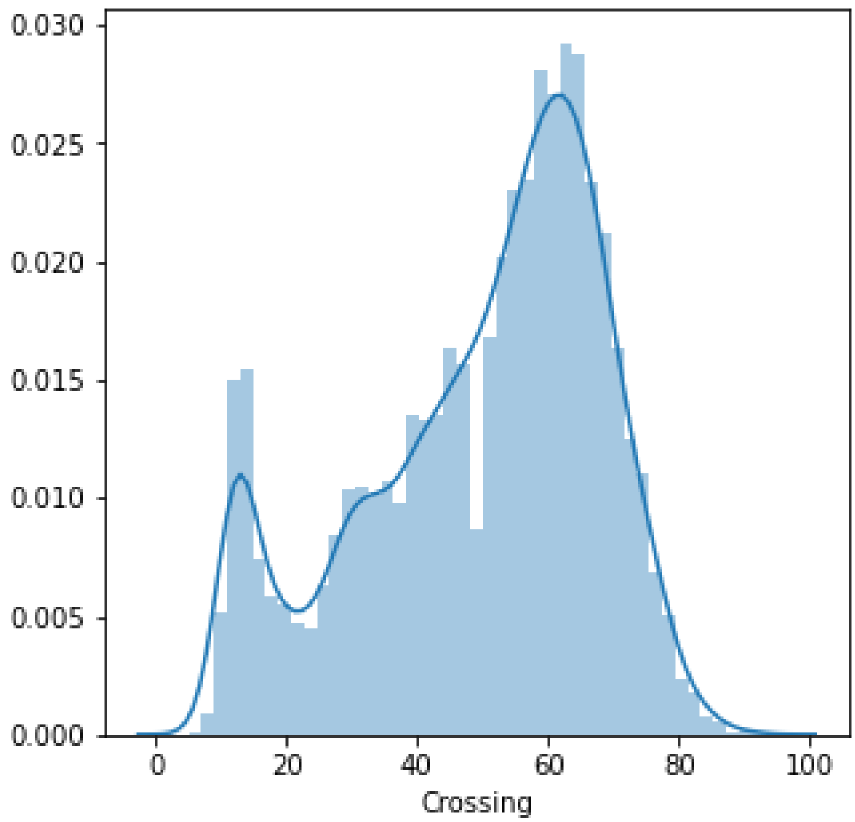
10
8
6
4
2
0
1
2
3
4
5
International Reputation

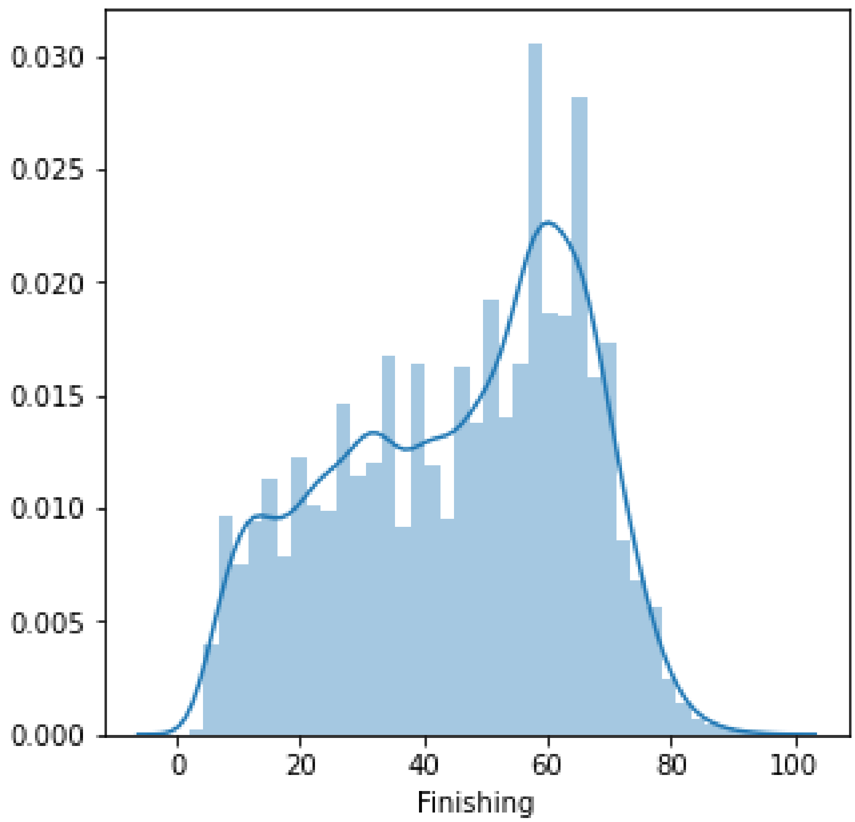


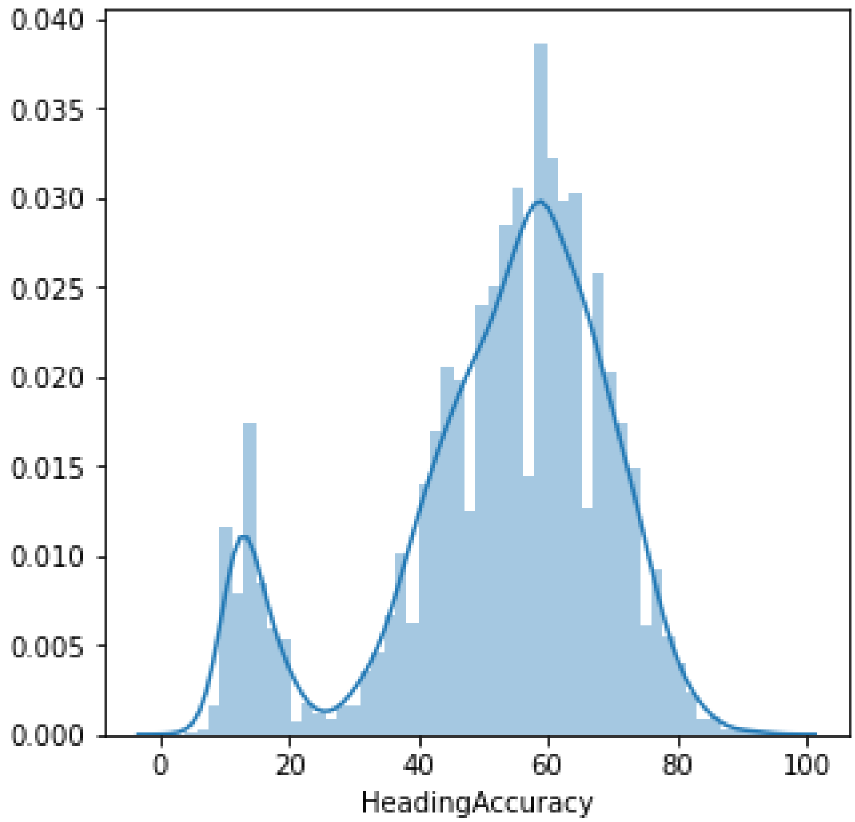
8
7
6
5
4
3
2
1
0
1
2
3
4
5
Weak Foot

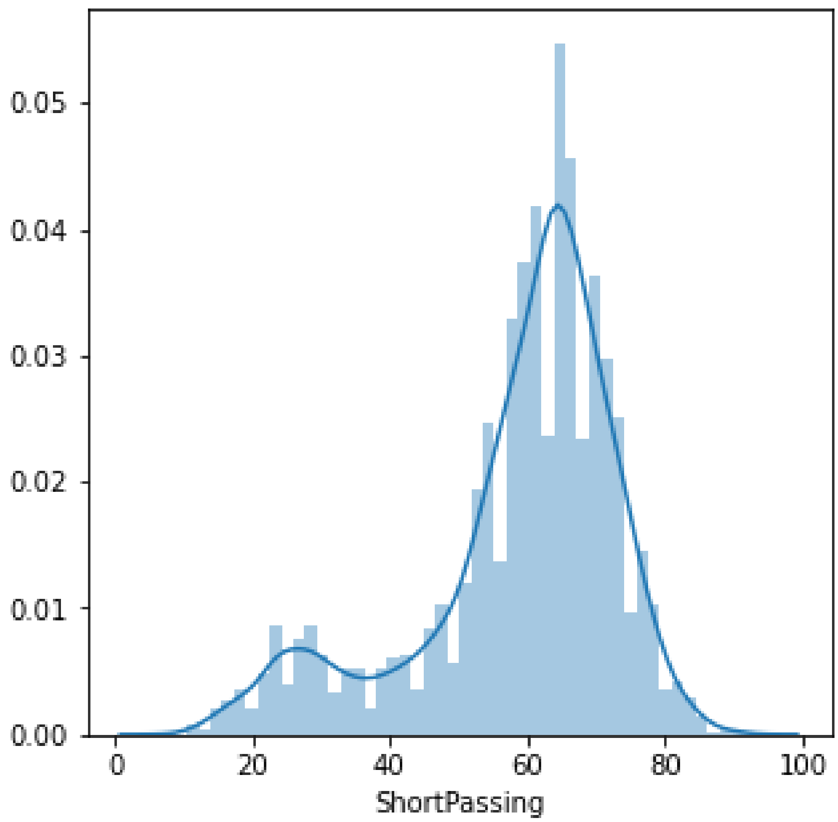


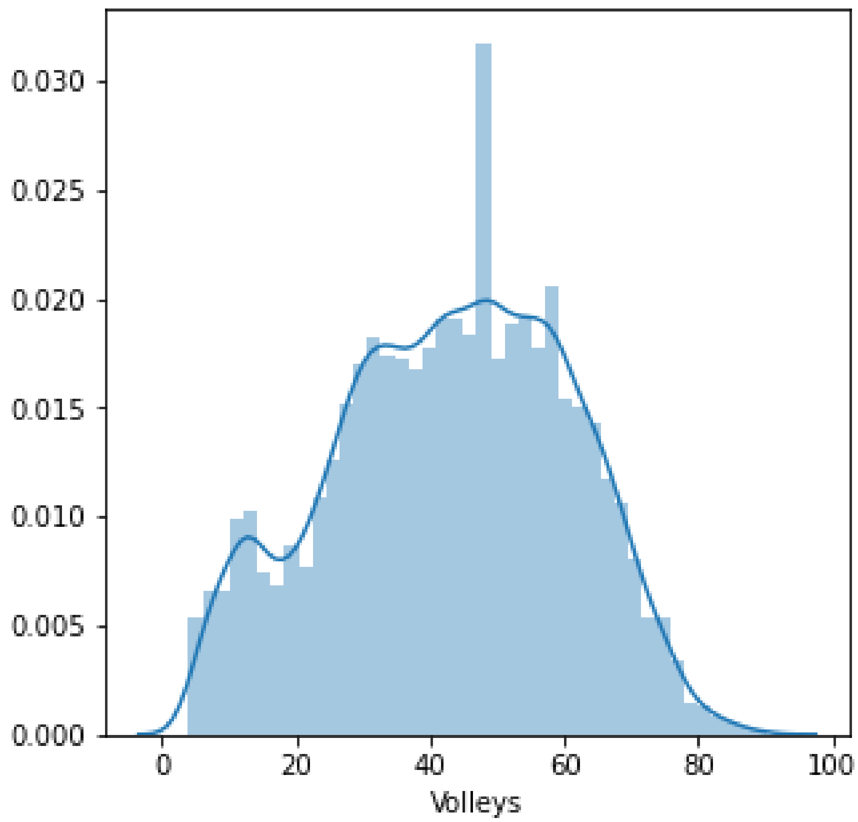
6
5
4
3
2
1
0
1
2
3
4
5
Skill Moves

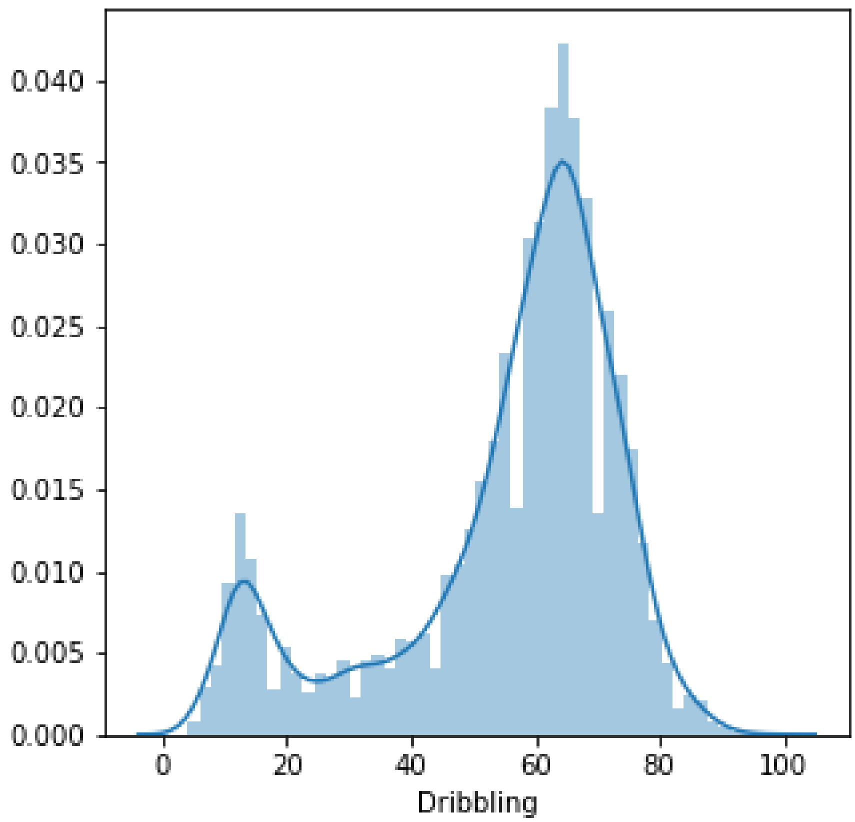

0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Crossing

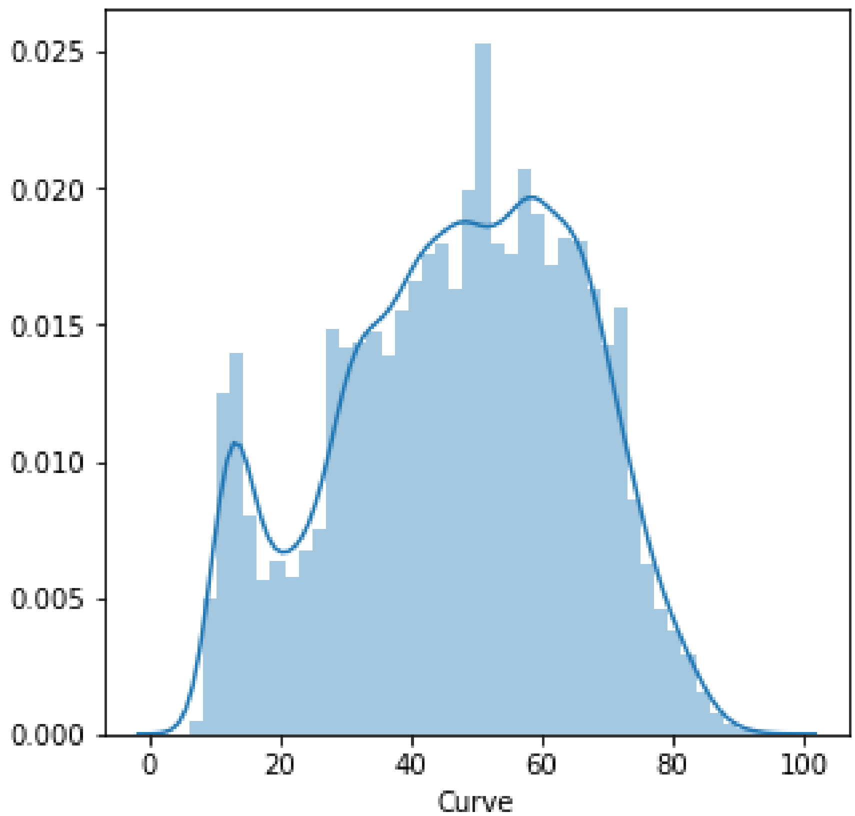

0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Finishing

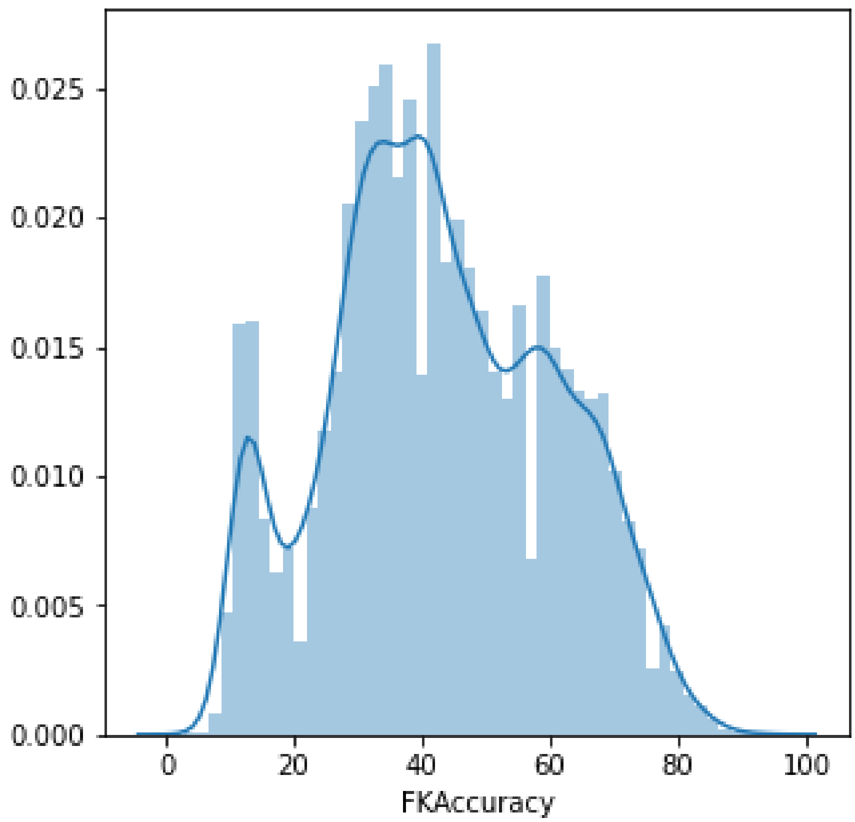

0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
HeadingAccuracy

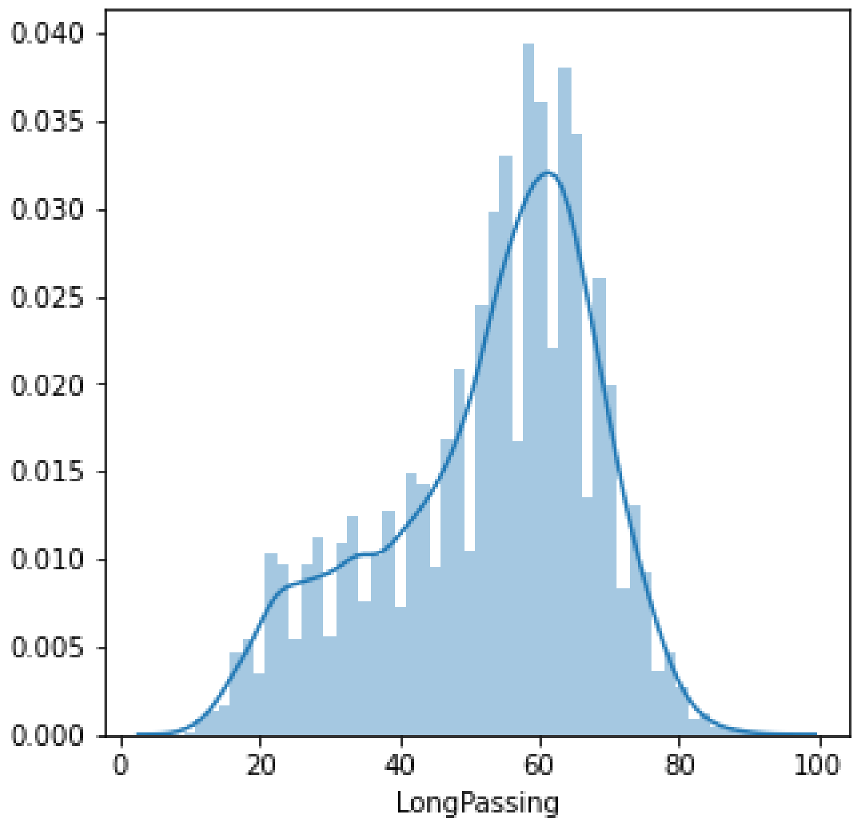

0.05
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
100
ShortPassing

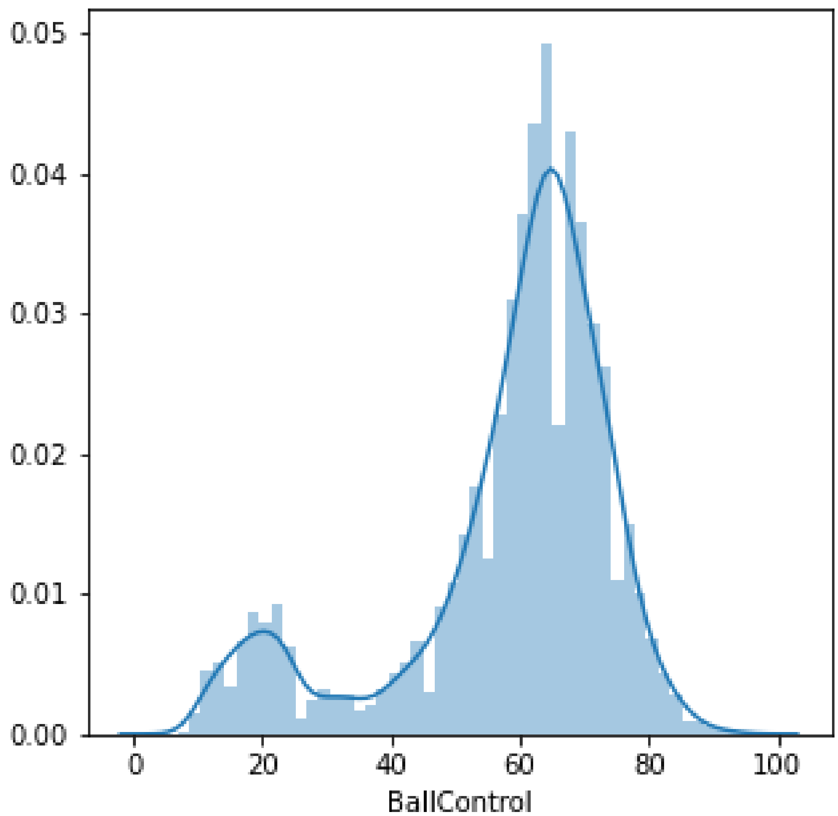

0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Volleys

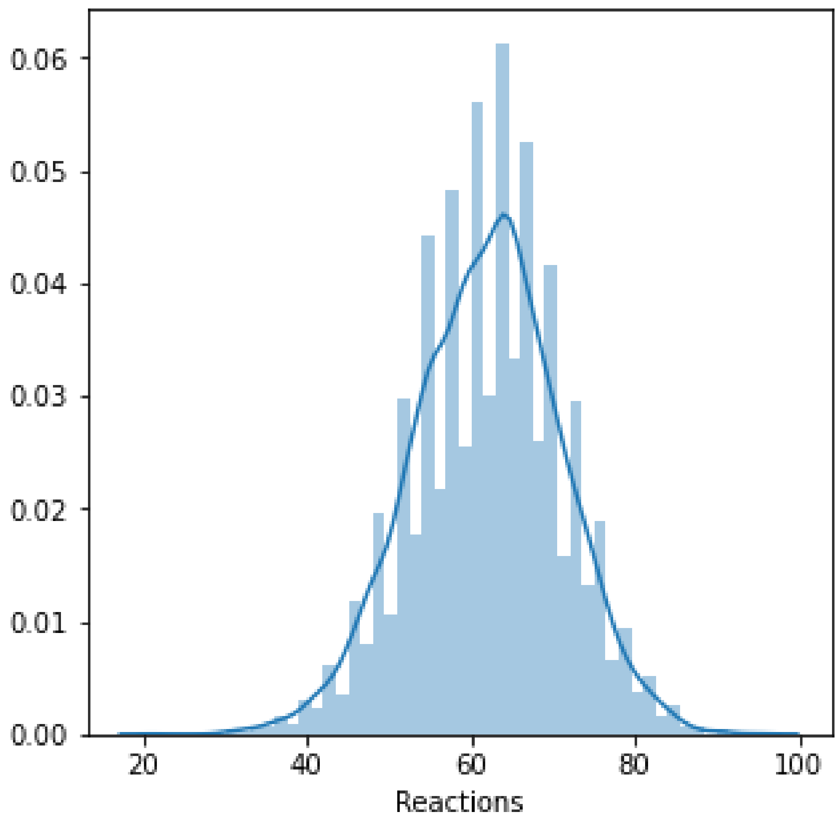

0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Dribbling

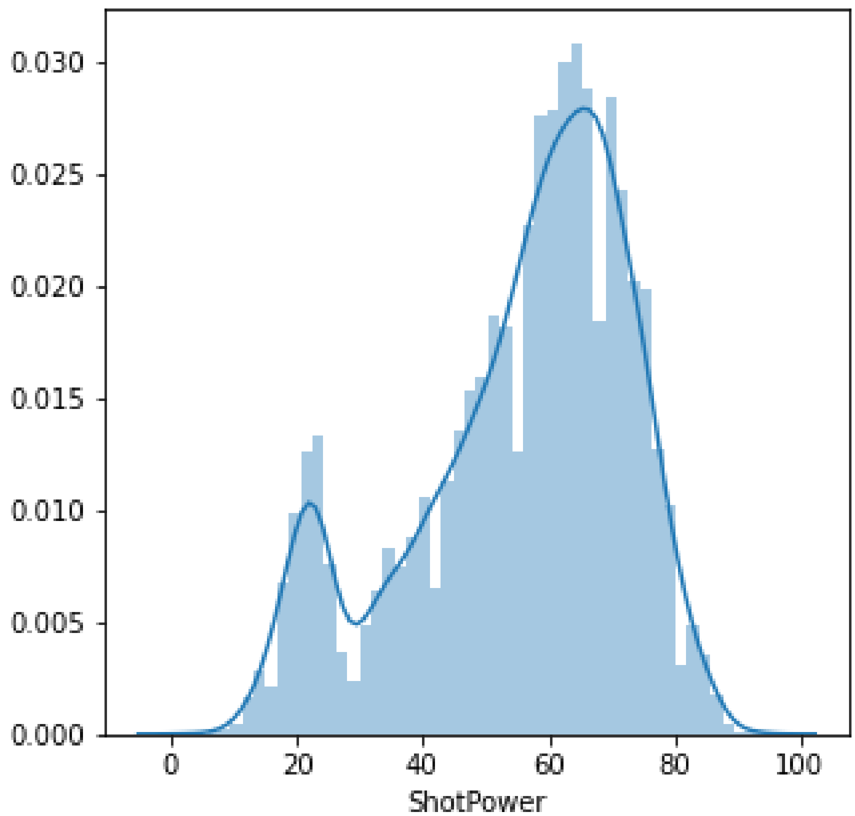

0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Curve

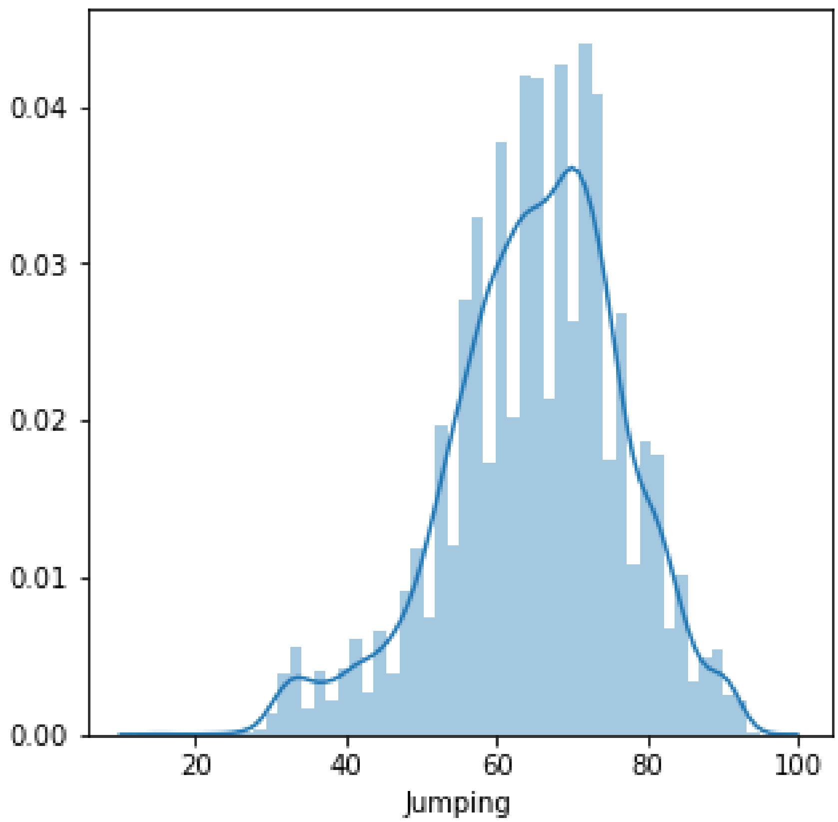

0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
FKAccuracy

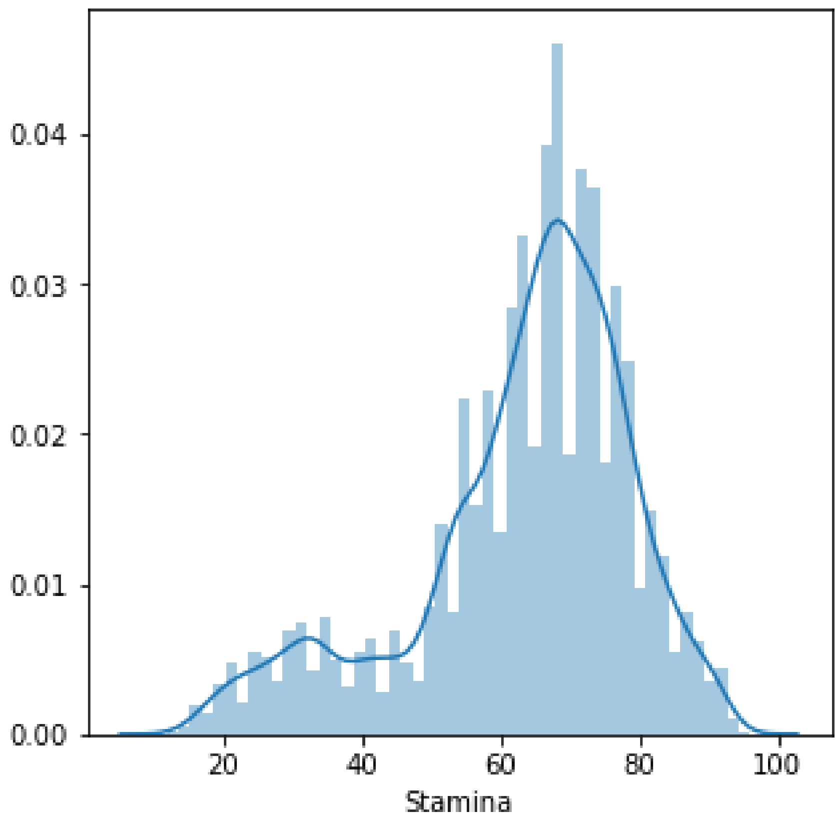

0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
LongPassing

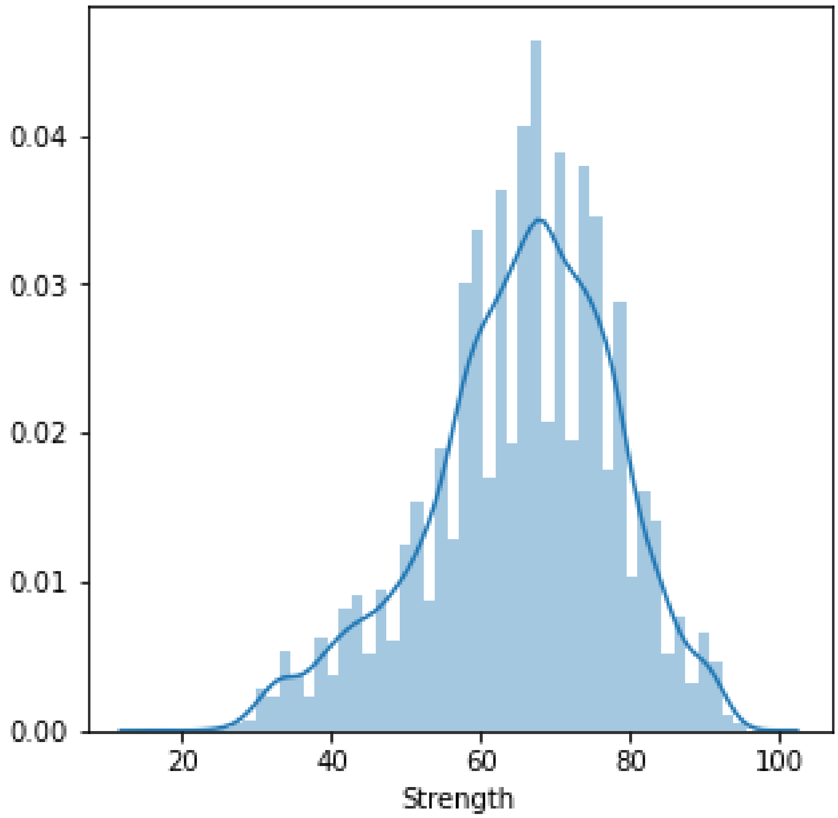

0.05
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
100
BallControl

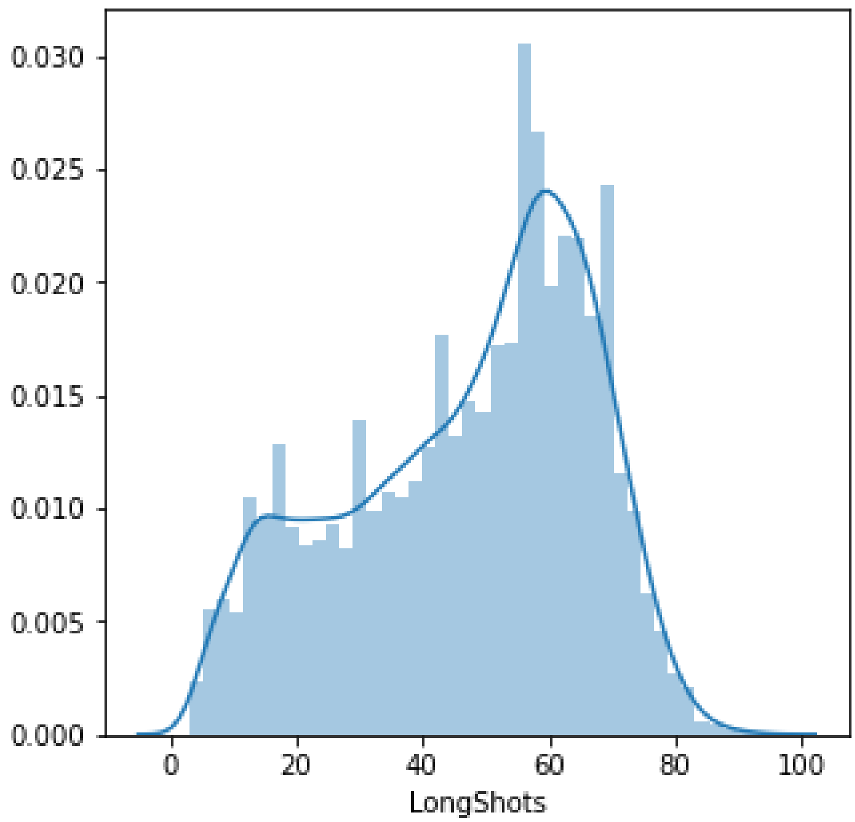

0.06
0.05
0.04
0.03
0.02
0.01
0.00
20
40
60
80
100
Reactions

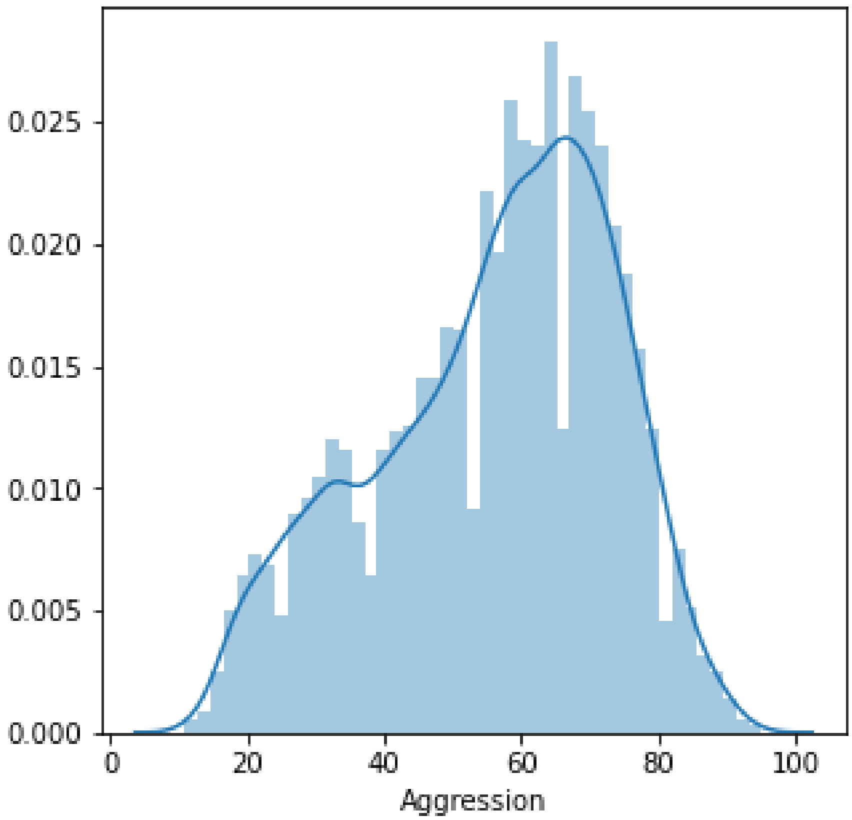

0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
ShotPower

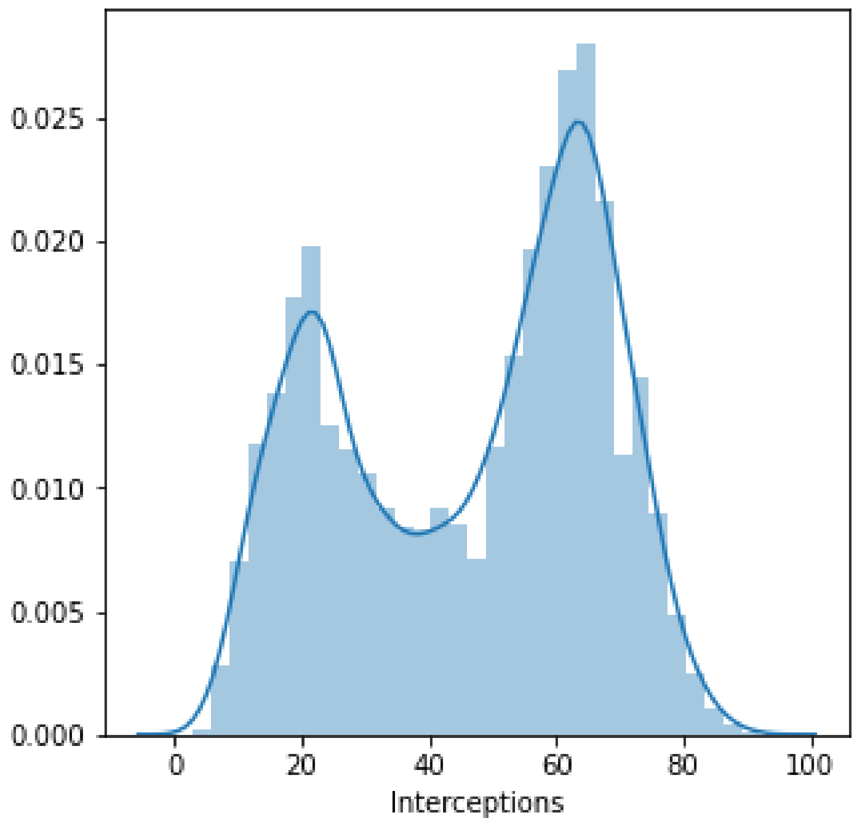

0.04
0.03
0.02
0.01
0.00
20
40
60
80
100
Jumping

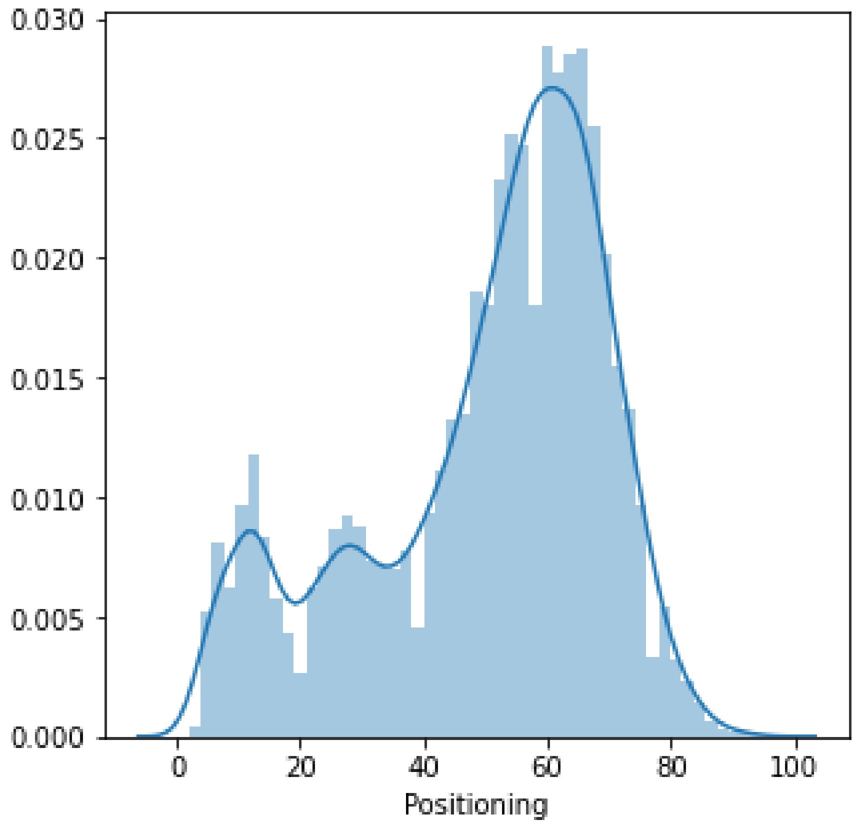

0.04
0.03
0.02
0.01
0.00
20
40
60
80
100
Stamina

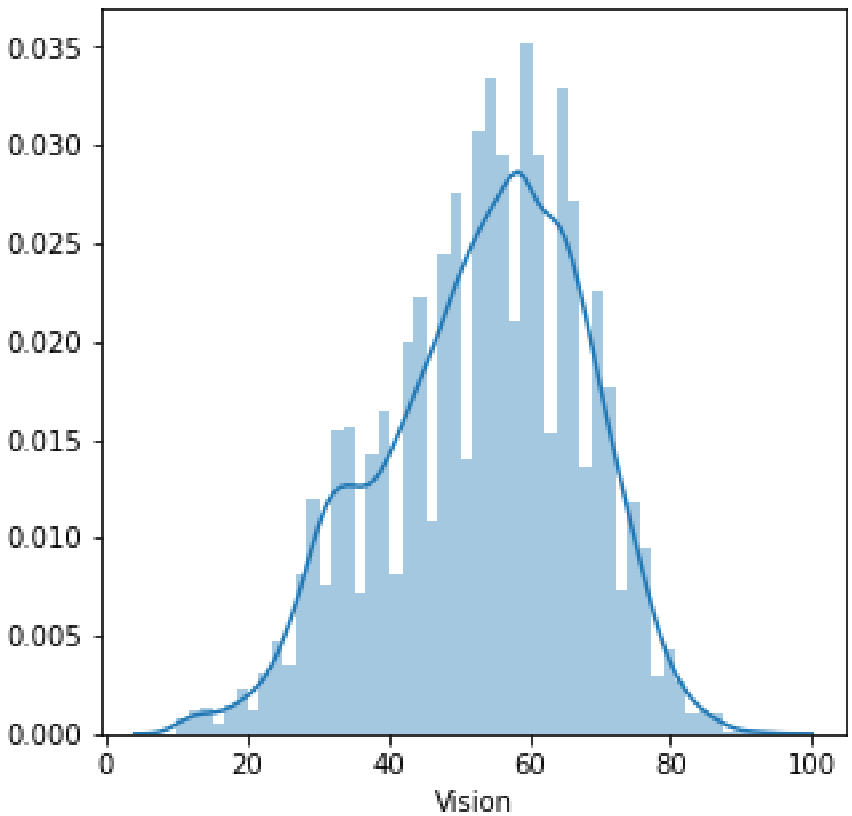

0.04
0.03
0.02
0.01
0.00
20
40
60
80
100
Strength

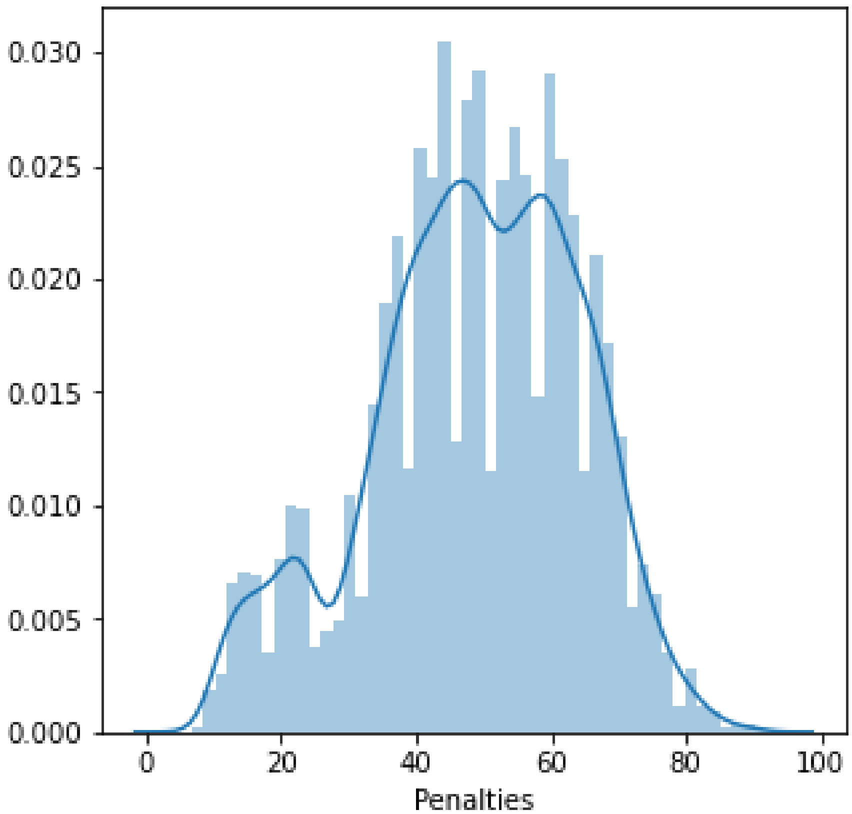

0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
LongShots

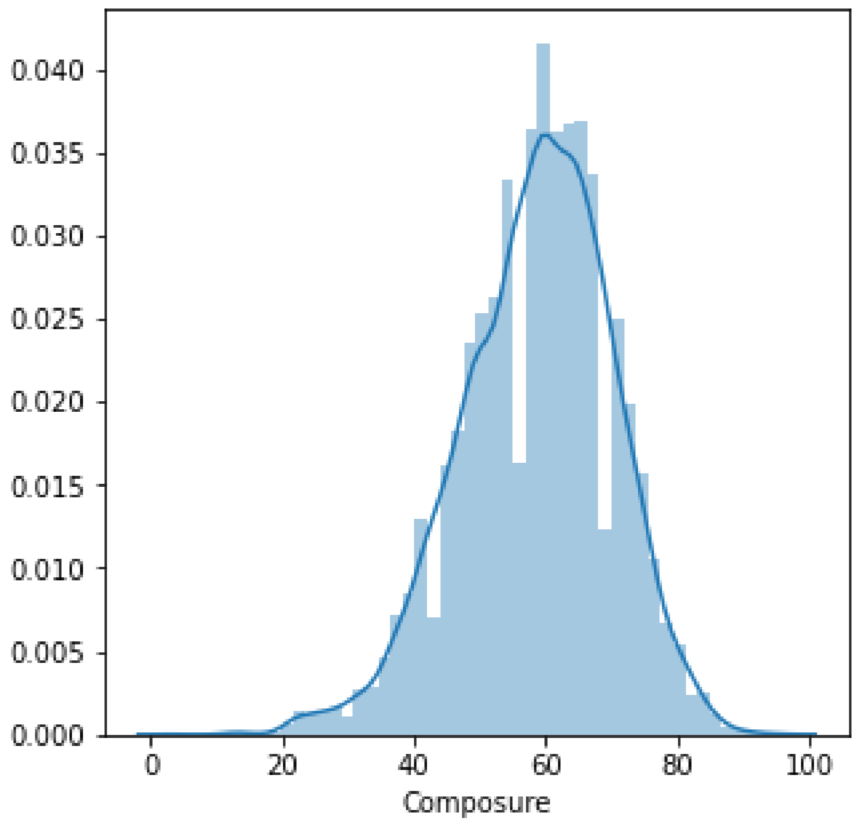

0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Aggression


0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Interceptions

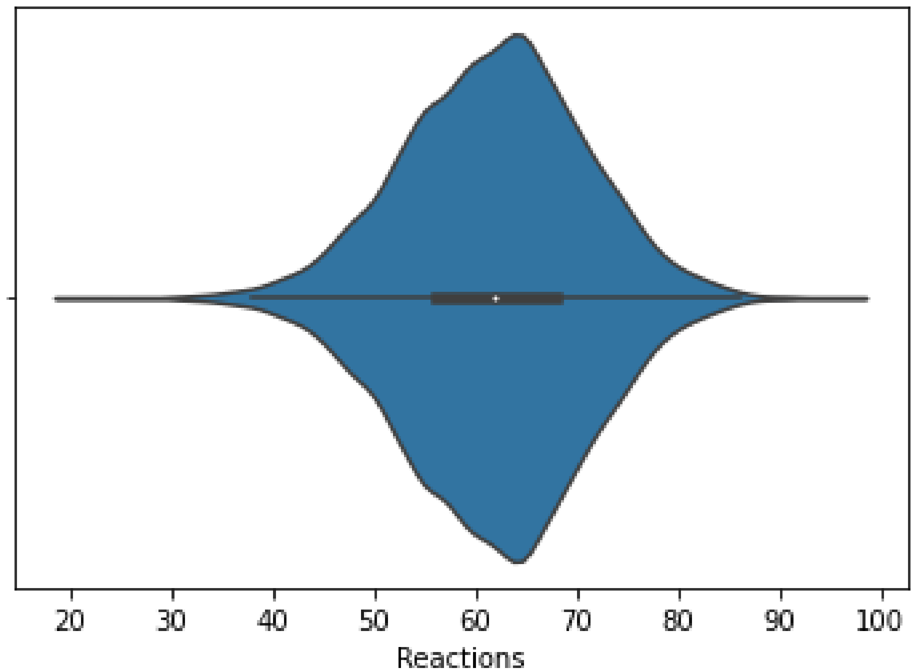

0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Positioning

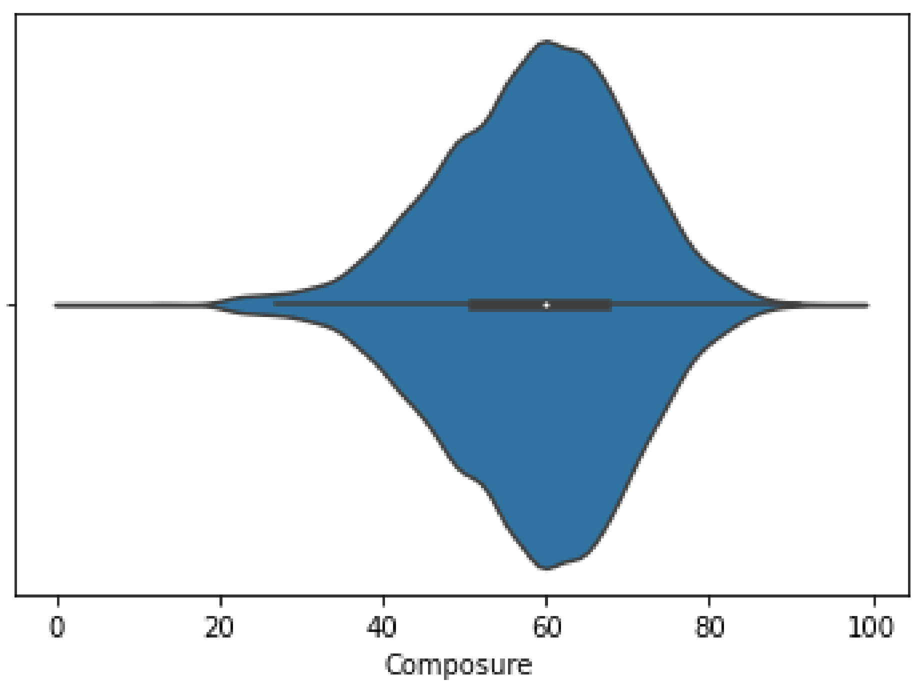

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Vision

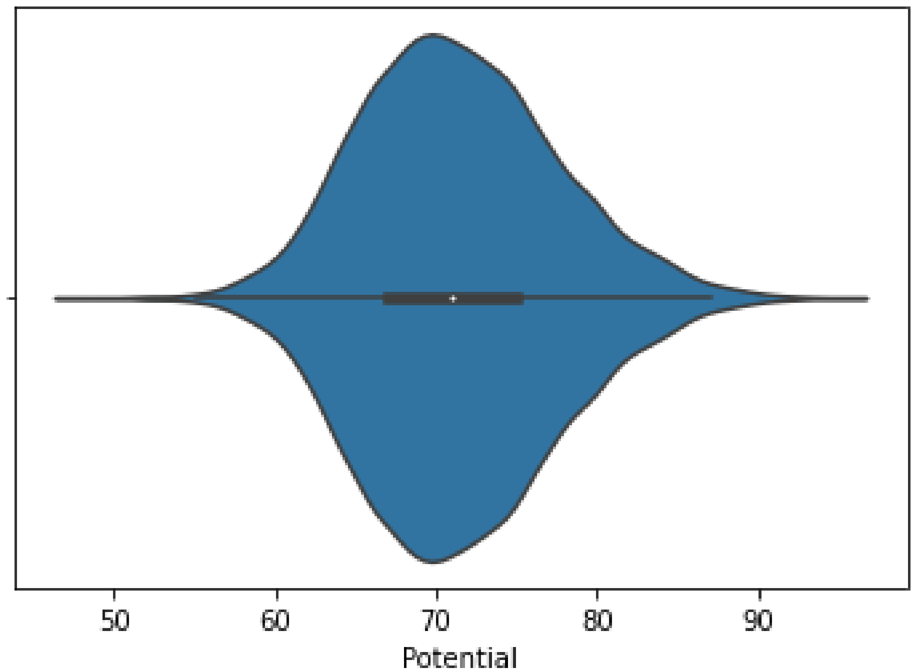

0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Penalties

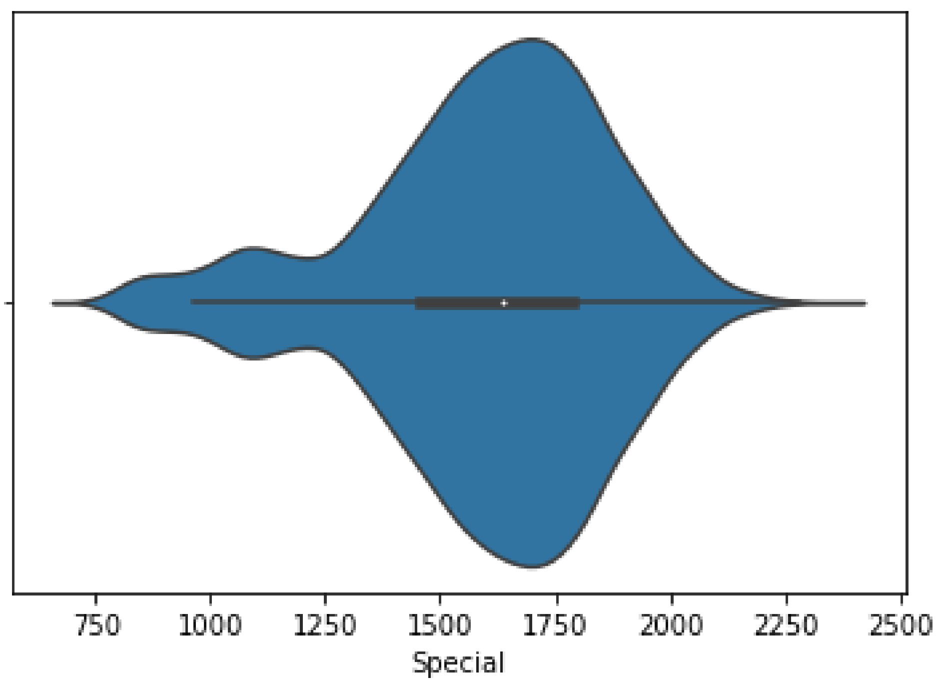


[236]:
```
# violin plot for most corr params
for col in most_corr:
    sns.violinplot(x=datac[col])
    plt.show()
```

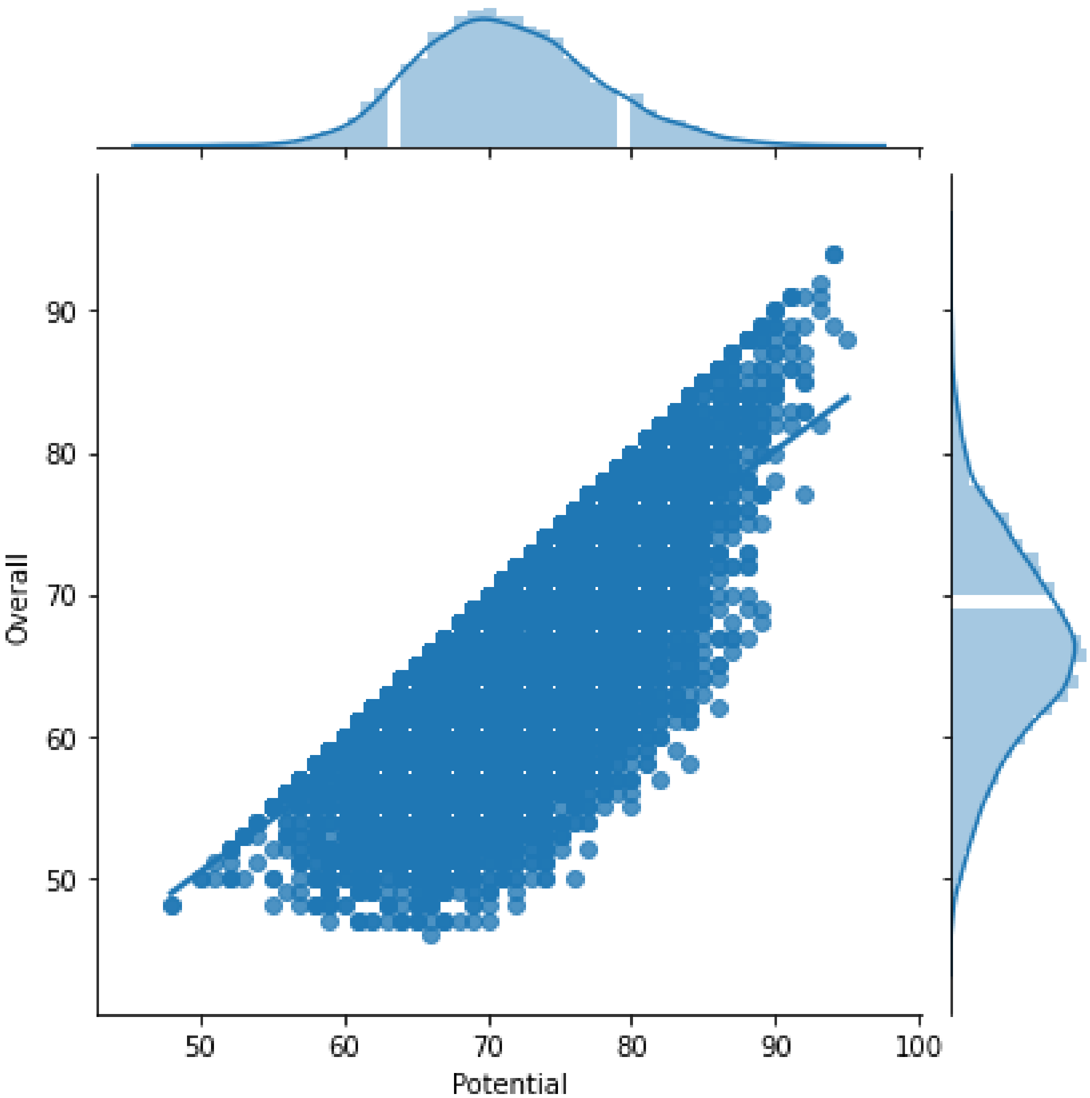

20
30
40
50
60
70
80
90
100
Reactions


0
20
40
60
80
100
Composure

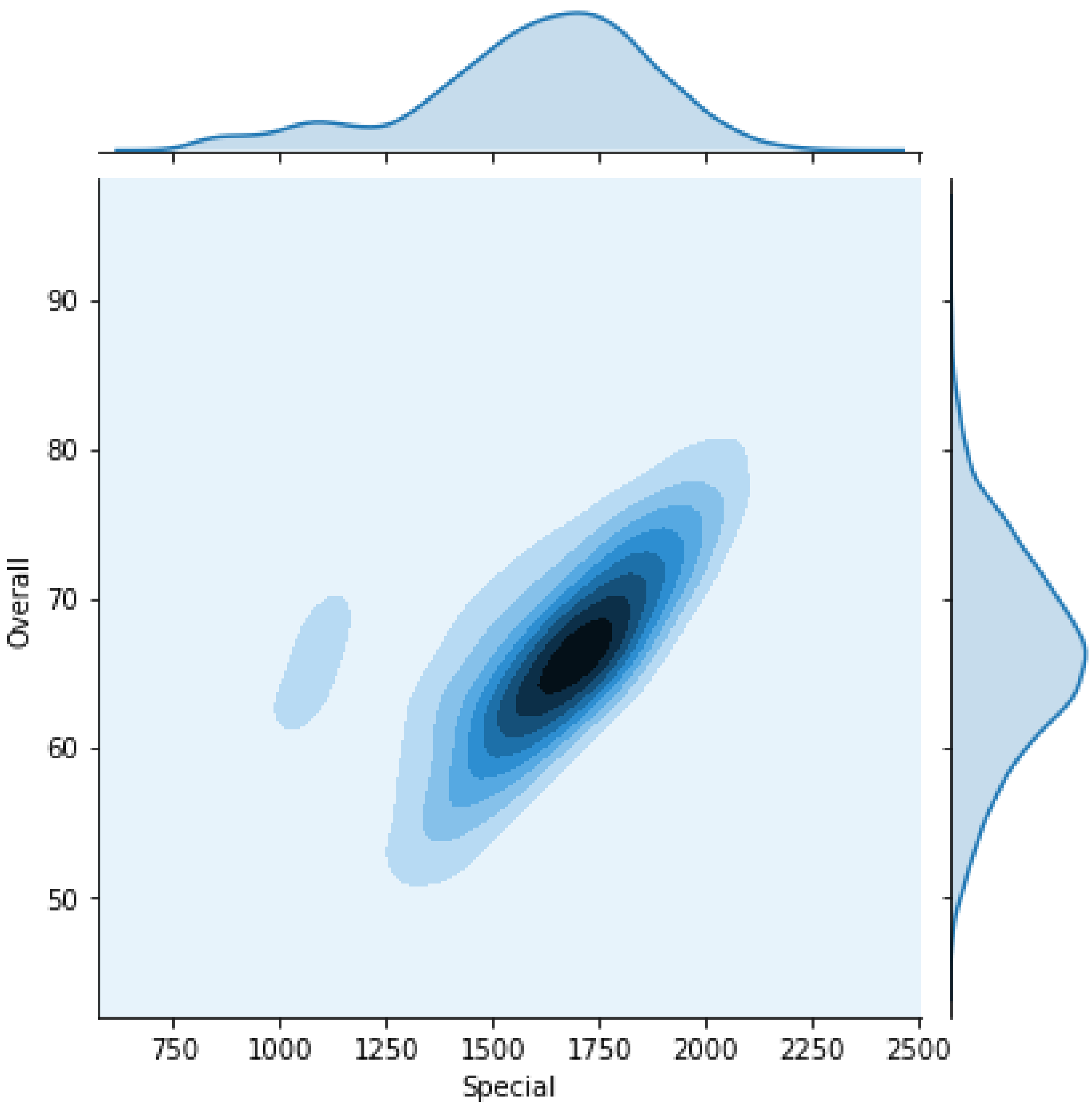




[162]:
```python
# dependence target attribute Overall of Potential
sns.jointplot(x="Potential", y=target_attr, data=data, kind="reg");
```



[238]:
```
# dependence target attribute Overall of Special
sns.jointplot(x="Special", y=target_attr, data=data, kind="kde");
```



Data preparation for model training

[239]:
```
# Split on target_attr as y and most_corr columns as X
X = datac[most_corr]
y = datac[target_attr]

X.head()
```

[239]:

	Reactions	Composure	Potential	Special
0	95.0	96.0	94	2202
1	96.0	95.0	94	2228
2	94.0	94.0	93	2143
3	90.0	68.0	93	1471

```
4       91.0       88.0          92     2281
```

[240]:
```
y.head()
```

[240]:
```
0    94
1    94
2    92
3    91
4    91
Name: Overall, dtype: int64
```

[242]:
```
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

[242]:

	Reactions	Composure	Potential	Special
count	1.820700e+04	1.820700e+04	1.820700e+04	1.820700e+04
mean	-4.995302e-17	1.498591e-16	6.993423e-16	-1.748356e-16
std	1.000027e+00	1.000027e+00	1.000027e+00	1.000027e+00
min	-4.538432e+00	-4.872788e+00	-3.798249e+00	-3.180037e+00
25%	-6.494538e-01	-6.699829e-01	-7.019345e-01	-5.165847e-01
50%	1.722820e-02	1.180430e-01	-5.007872e-02	1.364381e-01
75%	6.839102e-01	7.309521e-01	6.017770e-01	6.940755e-01
max	3.795093e+00	3.270147e+00	3.861056e+00	2.744861e+00

Metric Selection

As metrics we will use the following:

1) mean_absolute_error shows how wrong we are on average
2) median_absolute_error shows how wrong we are in half the dataset
3) r2_score shows the quality of the machine learning model in regression tasks

[172]:
```
# func for metrics calculation
def test_model(model):
    print("mean_absolute_error:",
          mean_absolute_error(y_test, model.predict(X_test)))
    print("median_absolute_error:",
          median_absolute_error(y_test, model.predict(X_test)))
    print("r2_score:",
          r2_score(y_test, model.predict(X_test)))
```

Forming training and test samples

[245]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                            test_size=0.2, random_state=1)
print(X_train.shape)
```

```
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(14565, 4)
(3642, 4)
(14565,)
(3642,)
```

Building a basic solution

As machine learning models we will use the following:

1) K nearest neighbors method
2) Decision tree
3) Random forest

K nearest neighbors method

[246]:
```
# with hyperparameter k=3
knn_3 = KNeighborsRegressor(n_neighbors=3)
knn_3.fit(X_train, y_train)
```

[246]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                    weights='uniform')
```

[247]:
```
test_model(knn_3)
```

```
mean_absolute_error: 2.035237049240344
median_absolute_error: 1.3333333333333357
r2_score: 0.8313539424192752
```

Decision tree

[248]:
```
# with unlimited depth
dt_none = DecisionTreeRegressor(max_depth=None)
dt_none.fit(X_train, y_train)
```

[248]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

[249]:
```
test_model(dt_none)
```

```
mean_absolute_error: 2.5054914881933006
median_absolute_error: 2.0
r2_score: 0.7255805478951515
```

Random forest

```
[250]: # with hyperparameter n=70:
       ran_70 = RandomForestRegressor(n_estimators=70)
       ran_70.fit(X_train, y_train)
```

```
[250]: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                             max_depth=None, max_features='auto', max_leaf_nodes=None,
                             max_samples=None, min_impurity_decrease=0.0,
                             min_impurity_split=None, min_samples_leaf=1,
                             min_samples_split=2, min_weight_fraction_leaf=0.0,
                             n_estimators=70, n_jobs=None, oob_score=False,
                             random_state=None, verbose=0, warm_start=False)
```

```
[251]: test_model(ran_70)
```

```
mean_absolute_error: 1.8826661175178476
median_absolute_error: 1.3142857142857167
r2_score: 0.8540068796853917
```

Selection of hyperparameters

K nearest neighbors model

Will try to find best K hyperparameter for this model

```
[252]: # list of customizable parameters
       param_range = np.arange(1, 25, 1)
       tuned_parameters = [{'n_neighbors': param_range}]
       tuned_parameters
```

```
[252]: [{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
       15, 16, 17,
                 18, 19, 20, 21, 22, 23, 24])}]
```

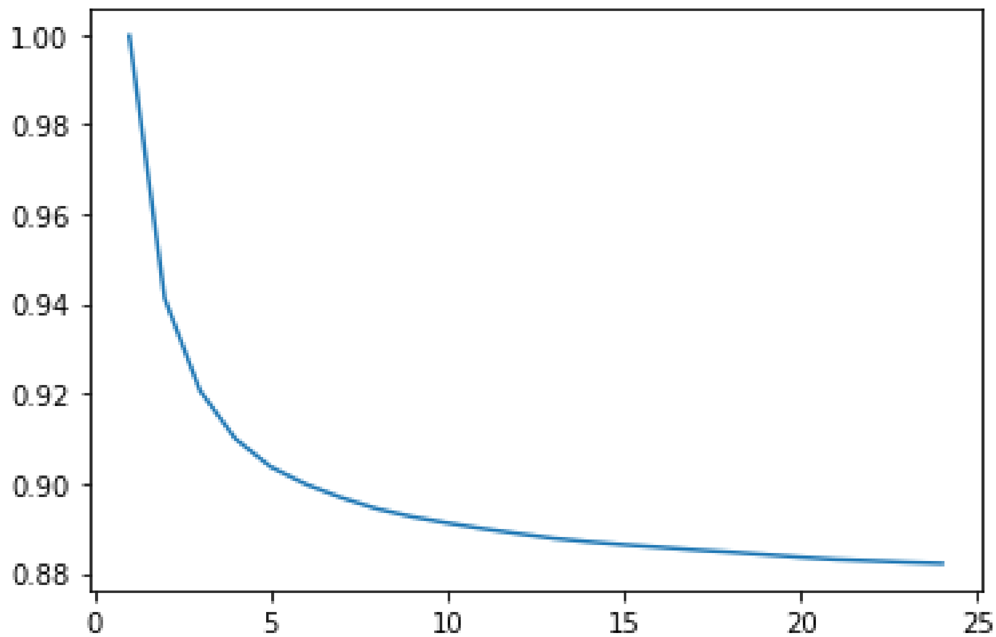
```
[253]: gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters,
                         cv=ShuffleSplit(n_splits=10), scoring="r2",
                         return_train_score=True, n_jobs=-1)
       gs.fit(X, y)
       gs.best_estimator_
```

```
[253]: KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=None, n_neighbors=24, p=2,
                           weights='uniform')
```

We‘ll check now graphics for train and test selections

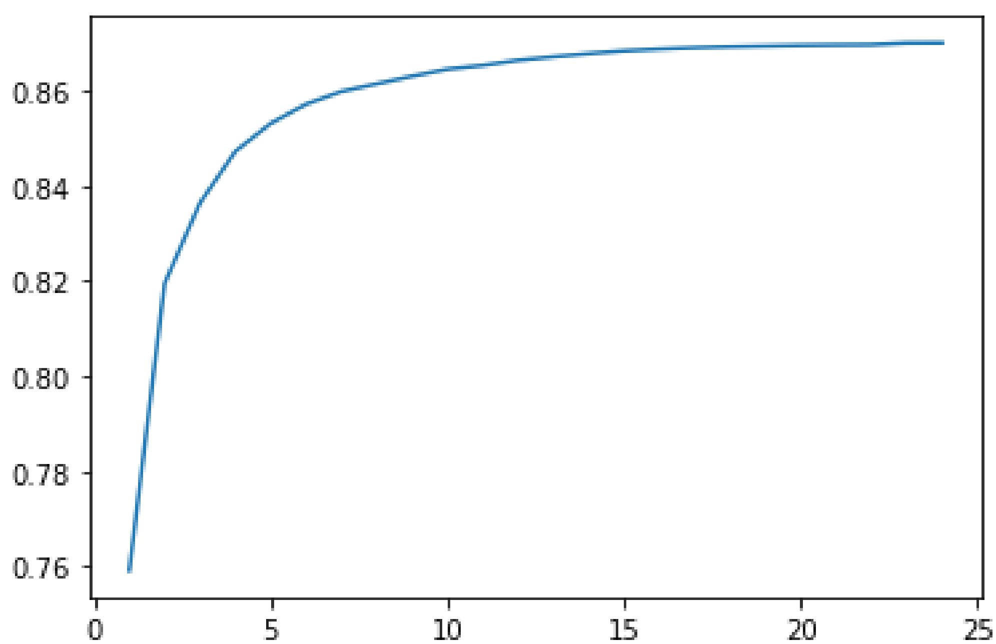
[254]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"])
```

[254]: [<matplotlib.lines.Line2D at 0x263d00d0be0>]



[255]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"])
```

[255]: [<matplotlib.lines.Line2D at 0x263d00cfac8>]



We have best result with K = 24

[256]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 1.845769265971078
median_absolute_error: 1.3333333333333286
r2_score: 0.8650133006653038
```

[257]:
```
test_model(knn_3)
```

```
mean_absolute_error: 2.035237049240344
median_absolute_error: 1.3333333333333357
r2_score: 0.8313539424192752
```

We see now that model with optimal hyperparameter better than our first baseline model for K nearest neighbors model

Decision tree

Will try to found best hyperparameter "depth of the decision tree"

[258]:
```
# list of customizable parameters
param_range = np.arange(1, 25, 1)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

[258]:
```
[{'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
15, 16, 17,
         18, 19, 20, 21, 22, 23, 24])}]
```

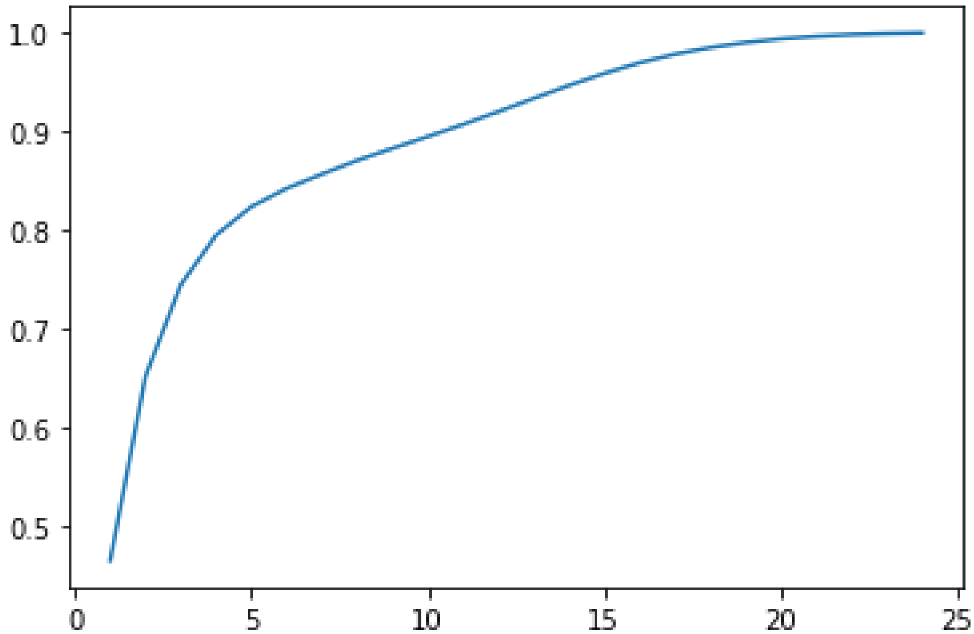
[259]:
```
gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[259]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=9,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

We'll check now graphics for train and test selections

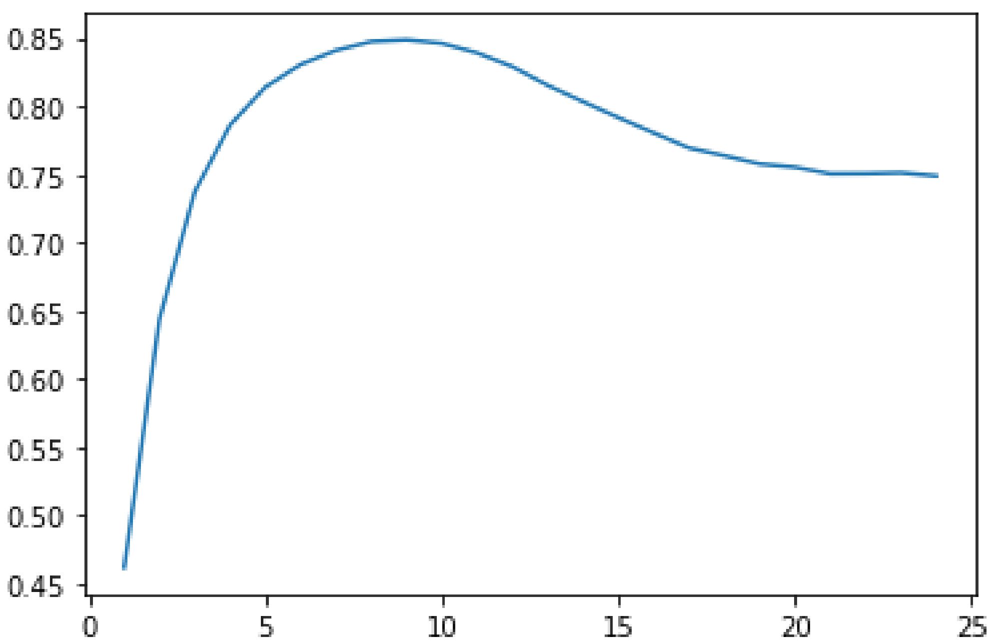
[260]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"])
```

[260]: [<matplotlib.lines.Line2D at 0x263cd91b0f0>]



[261]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"])
```

[261]: [<matplotlib.lines.Line2D at 0x263c1917eb8>]



We have best result with “depth of the decision tree” = 9

[262]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 1.966545426104523
median_absolute_error: 1.4226202207331475
r2_score: 0.8449414092575913
```

[263]:
```
test_model(dt_none)
```

```
mean_absolute_error: 2.5054914881933006
median_absolute_error: 2.0
r2_score: 0.7255805478951515
```

We see now that model with optimal hyperparameter better than our first baseline model for Decision tree model

Random forest

Will try to found best hyperparameter n

[264]:
```
# list of customizable parameters
param_range = np.arange(30, 300, 30)
tuned_parameters = [{'n_estimators': param_range}]
tuned_parameters
```

[264]:
```
[{'n_estimators': array([ 30,  60,  90, 120, 150, 180, 210, 240, 270])}]
```

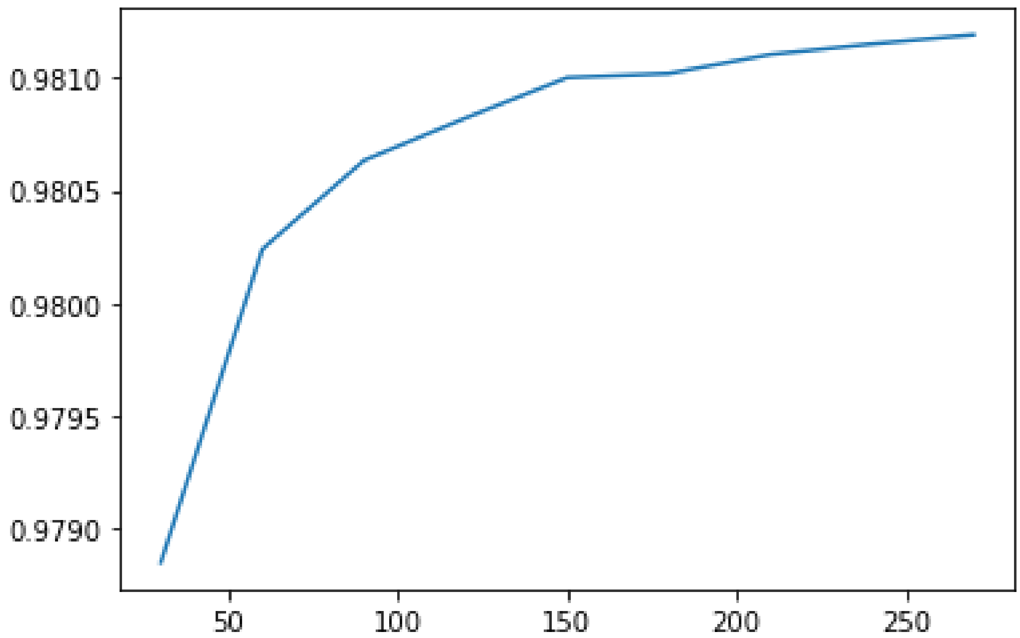
[265]:
```
gs = GridSearchCV(RandomForestRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[265]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=240, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

We‘ll check now graphics for train and test selections

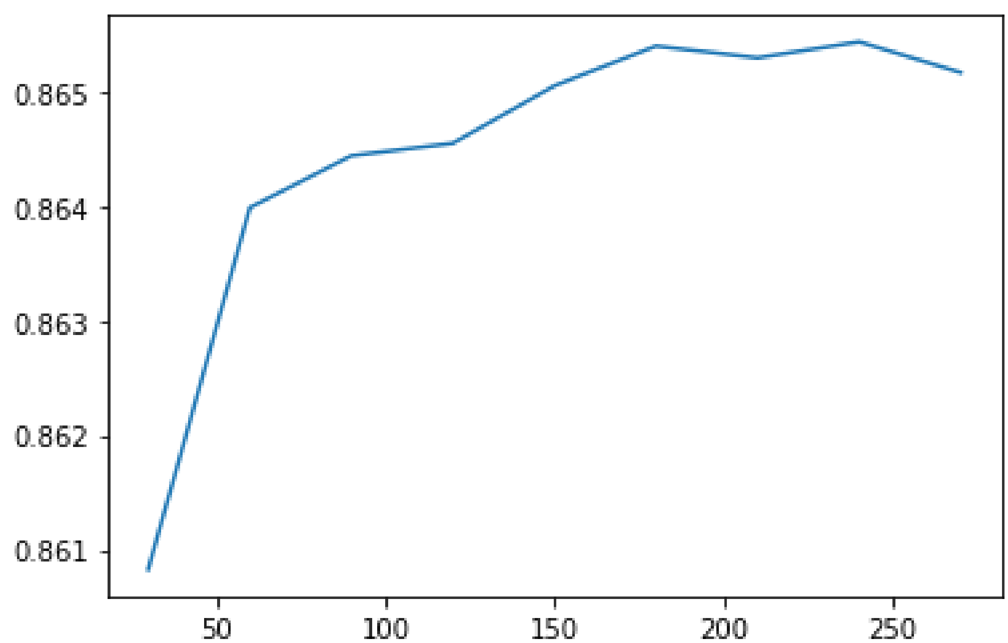
[266]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"])
```

[266]: [<matplotlib.lines.Line2D at 0x263c720a828>]



[267]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"])
```

[267]: [<matplotlib.lines.Line2D at 0x263c17b9668>]



We have best result with n = 240

[268]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 1.8716266545287343
median_absolute_error: 1.3083333333333336
r2_score: 0.8560122565060393
```

[269]:
```
test_model(ran_70)
```

```
mean_absolute_error: 1.8826661175178476
median_absolute_error: 1.3142857142857167
r2_score: 0.8540068796853917
```

We see now that model with optimal hyperparameter better than our first baseline model for Random forest model

CONCLUSION: we built 3 models with next optimal hyperparameters:

1) K nearest neighbors 0.865
2) Decision tree 0.8449
3) Random forest 0.856

And as we can see, the results are not very different

But still the best model for regression task with current dataset is **K nearest neighbors**

[ ]:
```
```