МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

ОТЧЁТ

Лабораторная работа №2
по курсу «**Методы машинного обучения**»

ИСПОЛНИТЕЛИ: Елизаров О.О.
ФИО
группа ИУ5-21М _____
подпись

"__"_____2022 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.У.
ФИО
_____
подпись

"__"_____2022 г.

Москва - 2022

Обработка признаков 2

Цель лабораторной работы: изучение продвинутых способов предварительной обработки данных для дальнейшего формирования моделей.

Задание:

1. Выбрать один или несколько наборов данных (датасетов) для решения следующих задач. Каждая задача может быть решена на отдельном датасете, или несколько задач могут быть решены на одном датасете. Просьба не использовать датасет, на котором данная задача решалась в лекции.
2. Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи:
    1) масштабирование признаков (не менее чем тремя способами);
    2) обработку выбросов для числовых признаков (по одному способу для удаления выбросов и для замены выбросов);
    3) обработку по крайней мере одного нестандартного признака (который не является числовым или категориальным);
    4) отбор признаков:
        - один метод из группы методов фильтрации (filter methods);
        - один метод из группы методов обертывания (wrapper methods);
        - один метод из группы методов вложений (embedded methods).

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MaxAbsScaler
```

In [137]:

```
data = pd.read_csv('archive/Pokemon.csv')
num_data=data.drop('#', axis=1) # redundant column
num_data=num_data.drop('Name', axis=1) # redundant column
num_data=num_data.drop('Type 1', axis=1) # redundant column
num_data=num_data.drop('Type 2', axis=1) # redundant column
num_data=num_data.drop('Legendary', axis=1) # redundant column
data.head()
```

Out[137]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1

In [138]:

```
num_data.describe()
```

Out[138]:

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen
count	800.00000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	80(
mean	435.10250	69.258750	79.001250	73.842500	72.820000	71.902500	68.277500	3
std	119.96304	25.534669	32.457366	31.183501	32.722294	27.828916	29.060474	1
min	180.00000	1.000000	5.000000	5.000000	10.000000	20.000000	5.000000	1
25%	330.00000	50.000000	55.000000	50.000000	49.750000	50.000000	45.000000	2
50%	450.00000	65.000000	75.000000	70.000000	65.000000	70.000000	65.000000	3
75%	515.00000	80.000000	100.000000	90.000000	95.000000	90.000000	90.000000	5
max	780.00000	255.000000	190.000000	230.000000	194.000000	230.000000	180.000000	6

In [139]:

```
x_all = num_data.drop('Generation', axis=1)
x_all
```

Out[139]:

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
0	318	45	49	49	65	65	45
1	405	60	62	63	80	80	60
2	525	80	82	83	100	100	80
3	625	80	100	123	122	120	80
4	309	39	52	43	60	50	65
...	...	...	...	...	...	...	...
795	600	50	100	150	100	150	50
796	700	50	160	110	160	110	110
797	600	80	110	60	150	130	70
798	680	80	160	60	170	130	80
799	600	80	110	120	130	90	70

800 rows × 7 columns

In [140]:

```
def arr_to_df(arr_scaled,df):
    res = pd.DataFrame(arr_scaled, columns=df.columns)
    return res
```

In [141]:

```
X_train, X_test, y_train, y_test = train_test_split(x_all, num_data['Generation'],
                                                    test_size=0.2,
                                                    random_state=1)
# Преобразуем массивы в DataFrame
X_train_df = arr_to_df(X_train,x_all)
X_test_df = arr_to_df(X_test,x_all)

X_train_df.shape, X_test_df.shape
```

Out[141]:

```
((640, 7), (160, 7))
```

In [ ]:

```

```

In [142]:

```
cs11 = StandardScaler()
data_cs11_scaled_temp = cs11.fit_transform(x_all)
# формируем DataFrame на основе массива
data_cs11_scaled =  arr_to_df(data_cs11_scaled_temp,x_all)
data_cs11_scaled
```

Out[142]:

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
0	-0.976765	-0.950626	-0.924906	-0.797154	-0.239130	-0.248189	-0.801503
1	-0.251088	-0.362822	-0.524130	-0.347917	0.219560	0.291156	-0.285015
2	0.749845	0.420917	0.092448	0.293849	0.831146	1.010283	0.403635
3	1.583957	0.420917	0.647369	1.577381	1.503891	1.729409	0.403635
4	-1.051836	-1.185748	-0.832419	-0.989683	-0.392027	-0.787533	-0.112853
...	...	...	...	...	...	...	...
795	1.375429	-0.754692	0.647369	2.443765	0.831146	2.808099	-0.629341
796	2.209541	-0.754692	2.497104	1.160233	2.665905	1.369846	1.436611
797	1.375429	0.420917	0.955658	-0.444182	2.360112	2.088973	0.059310
798	2.042718	0.420917	2.497104	-0.444182	2.971699	2.088973	0.403635
799	1.375429	0.420917	0.955658	1.481116	1.748526	0.650720	0.059310

800 rows × 7 columns

In [143]:

```
data_cs11_scaled.describe()
```

Out[143]:

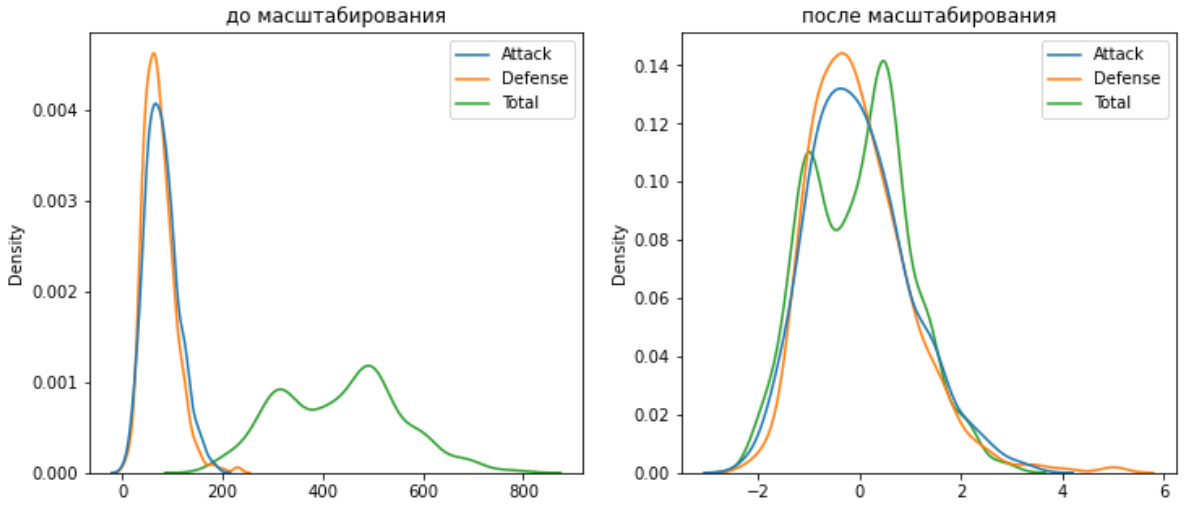
	Total	HP	Attack	Defense	Sp. Atk	Sp. De
count	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02	8.000000e+02
mean	-7.993606e-17	-2.660372e-16	-1.355860e-16	2.498002e-17	2.120526e-16	-1.171285e-16
std	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00	1.000626e+00
min	-2.127839e+00	-2.674852e+00	-2.281379e+00	-2.209039e+00	-1.920993e+00	-1.866223e+00
25%	-8.766721e-01	-7.546915e-01	-7.399327e-01	-7.650653e-01	-7.054650e-01	-7.875334e-01
50%	1.242618e-01	-1.668874e-01	-1.233542e-01	-1.232993e-01	-2.391303e-01	-6.840691e-02
75%	6.664343e-01	4.209167e-01	6.473688e-01	5.184667e-01	6.782494e-01	6.507196e-01
max	2.876830e+00	7.278632e+00	3.421972e+00	5.010829e+00	3.705602e+00	5.684605e+00

In [144]:

```
# Построение плотности распределения
def draw_kde(col_list, df1, df2, label1, label2):
    fig, (ax1, ax2) = plt.subplots(
        ncols=2, figsize=(12, 5))
    # первый график
    ax1.set_title(label1)
    sns.kdeplot(data=df1[col_list], ax=ax1)
    # второй график
    ax2.set_title(label2)
    sns.kdeplot(data=df2[col_list], ax=ax2)
    plt.show()
```

In [145]:

```
draw_kde(['Attack', 'Defense', 'Total'], num_data, data_cs11_scaled, 'до масштабирования',
```



Крайне сильно заметны изменения колонки total, теперь она более соответствует остальным, а не выглядит, как один большой выброс

In [ ]:

```
```

In [146]:

```
cs12 = StandardScaler()
cs12.fit(X_train)
data_cs12_scaled_train_temp = cs12.transform(X_train)
data_cs12_scaled_test_temp = cs12.transform(X_test)
# формируем DataFrame на основе массива
data_cs12_scaled_train = arr_to_df(data_cs12_scaled_train_temp,x_all)
data_cs12_scaled_test = arr_to_df(data_cs12_scaled_test_temp,x_all)
```

In [147]:

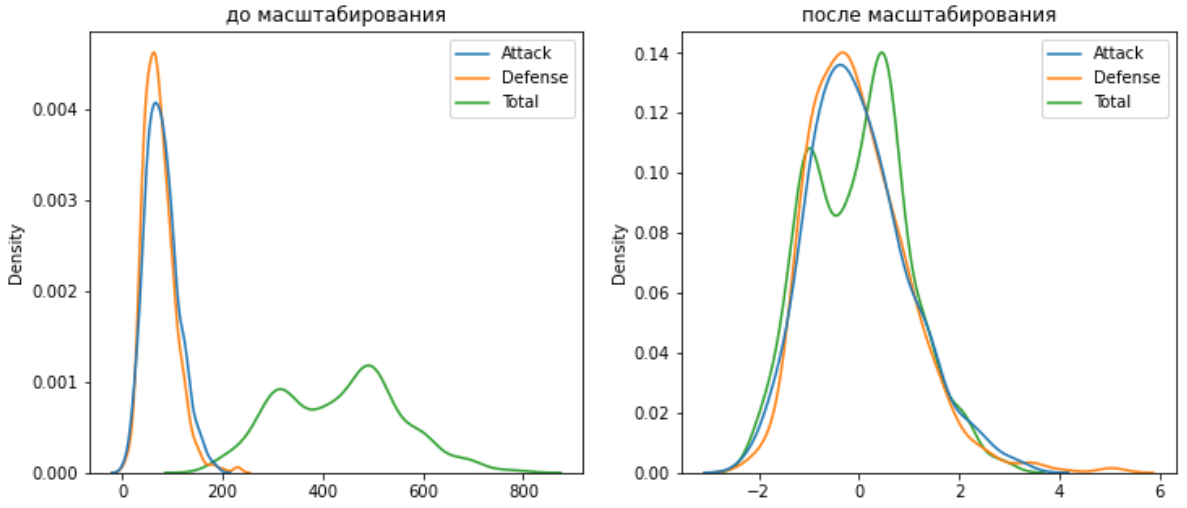
```
data_cs12_scaled_train.describe()
```

Out[147]:

	Total	HP	Attack	Defense	Sp. Atk	Sp. De
count	6.400000e+02	6.400000e+02	6.400000e+02	6.400000e+02	6.400000e+02	6.400000e+02
mean	-1.571659e-16	1.148387e-16	-1.712172e-16	2.005340e-16	1.384309e-16	2.168404e-16
std	1.000782e+00	1.000782e+00	1.000782e+00	1.000782e+00	1.000782e+00	1.000782e+00
min	-2.120630e+00	-2.579985e+00	-2.258464e+00	-2.230880e+00	-1.893436e+00	-1.839232e+00
25%	-8.762811e-01	-7.399885e-01	-7.432247e-01	-7.765841e-01	-7.479106e-01	-7.755478e-01
50%	1.191983e-01	-1.767242e-01	-1.371291e-01	-1.302302e-01	-2.244585e-01	-6.642487e-02
75%	6.584163e-01	4.334789e-01	6.204903e-01	5.161237e-01	6.858929e-01	6.426981e-01
max	2.856767e+00	6.957958e+00	3.347920e+00	5.040601e+00	3.690053e+00	5.606559e+00

In [148]:

```
draw_kde(['Attack', 'Defense', 'Total'], num_data, data_cs12_scaled_train, 'до масштабирова
```



In [ ]:

```

```

In [ ]:

```

```

In [149]:

```
class MeanNormalisation:

    def fit(self, param_df):
        self.means = X_train.mean(axis=0)
        maxs = X_train.max(axis=0)
        mins = X_train.min(axis=0)
        self.ranges = maxs - mins

    def transform(self, param_df):
        param_df_scaled = (param_df - self.means) / self.ranges
        return param_df_scaled

    def fit_transform(self, param_df):
        self.fit(param_df)
        return self.transform(param_df)
```

In [150]:

```
sc21 = MeanNormalisation()
data_cs21_scaled = sc21.fit_transform(x_all)
data_cs21_scaled.describe()
```

Out[150]:

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
count	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000	800.000000
mean	-0.000881	-0.001762	-0.002831	-0.000832	0.002300	0.000138	0.001014
std	0.199938	0.100530	0.175445	0.138593	0.177839	0.132519	0.166060
min	-0.426052	-0.270497	-0.402838	-0.306799	-0.339113	-0.247016	-0.360571
25%	-0.176052	-0.077584	-0.132568	-0.106799	-0.123081	-0.104159	-0.132000
50%	0.023948	-0.018529	-0.024459	-0.017910	-0.040200	-0.008921	-0.017714
75%	0.132281	0.040527	0.110676	0.070979	0.122843	0.086317	0.125143
max	0.573948	0.729503	0.597162	0.693201	0.660887	0.752984	0.639429

In [151]:

```
cs22 = MeanNormalisation()
cs22.fit(X_train)
data_cs22_scaled_train = cs22.transform(X_train)
data_cs22_scaled_test = cs22.transform(X_test)
```

In [152]:

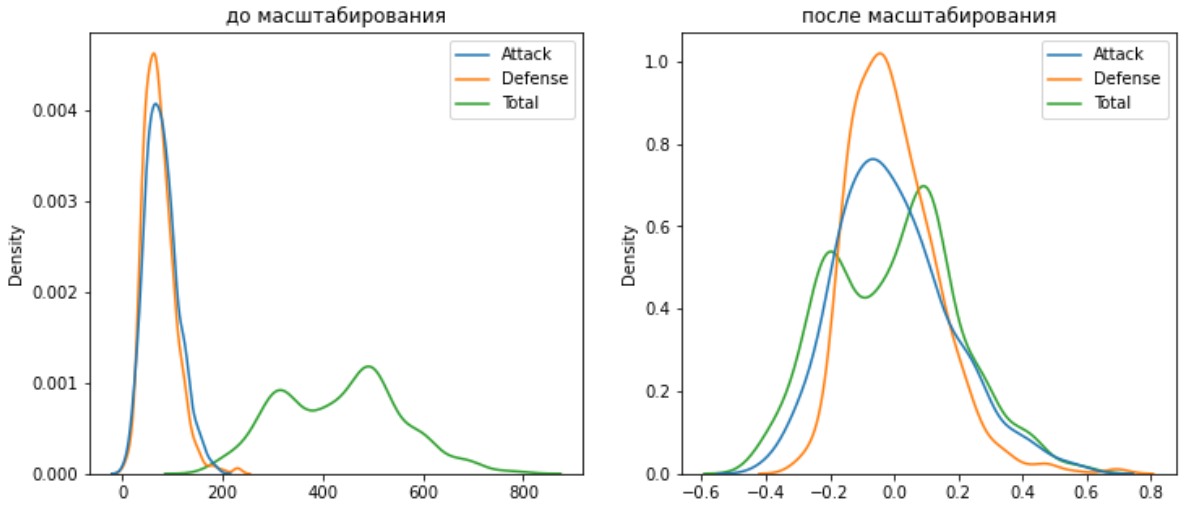
```
data_cs22_scaled_train.describe()
```

Out[152]:

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	
count	6.400000e+02	6.400000e+02	6.400000e+02	6.400000e+02	6.400000e+02	6.400000e+02	6
mean	-4.458239e-17	1.216475e-17	-3.439089e-17	2.467644e-17	3.625572e-17	2.484991e-17	3
std	2.010654e-01	1.049264e-01	1.785076e-01	1.376311e-01	1.792396e-01	1.344091e-01	1
min	-4.260521e-01	-2.704970e-01	-4.028378e-01	-3.067986e-01	-3.391135e-01	-2.470164e-01	
25%	-1.760521e-01	-7.758366e-02	-1.325676e-01	-1.067986e-01	-1.339504e-01	-1.041592e-01	
50%	2.394792e-02	-1.852854e-02	-2.445946e-02	-1.790972e-02	-4.020041e-02	-8.921131e-03	
75%	1.322812e-01	4.544783e-02	1.106757e-01	7.097917e-02	1.228431e-01	8.631696e-02	1
max	5.739479e-01	7.295030e-01	5.971622e-01	6.932014e-01	6.608865e-01	7.529836e-01	6

In [153]:

```
draw_kde(['Attack', 'Defense', 'Total'], num_data, data_cs22_scaled_train, 'до масштабирова
```



Сдвиг по оси X исправлен, а вот амплитуда немного вышла хуже, чем в прошлом способе

In [ ]:

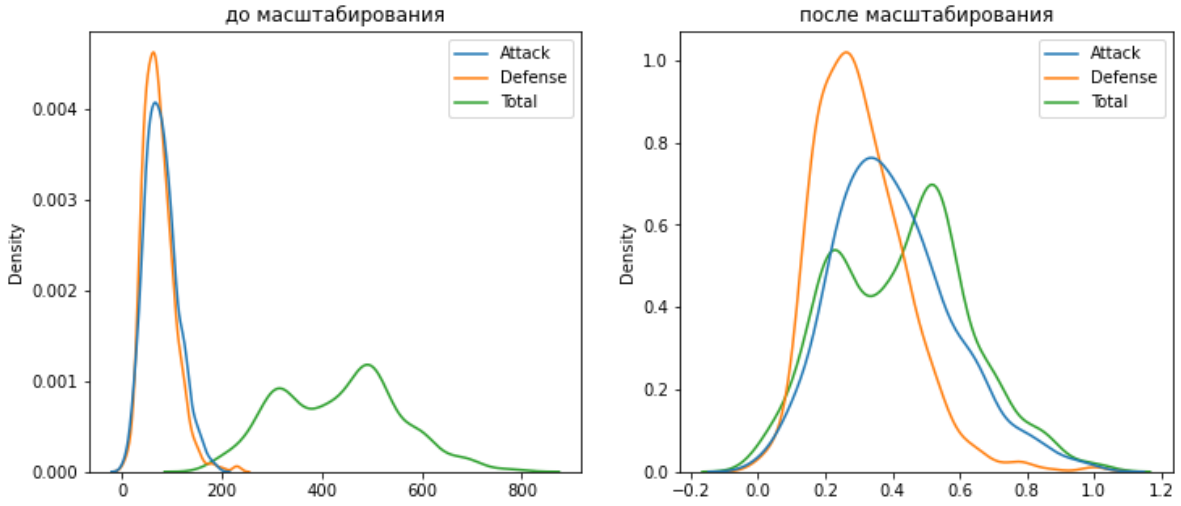
```

```

In [154]:

```
cs32 = MinMaxScaler()
cs32.fit(X_train)
data_cs32_scaled_train_temp = cs32.transform(X_train)
data_cs32_scaled_test_temp = cs32.transform(X_test)
# формируем DataFrame на основе массива
data_cs32_scaled_train = arr_to_df(data_cs32_scaled_train_temp,x_all)
data_cs32_scaled_test = arr_to_df(data_cs32_scaled_test_temp,x_all)
```

In [155]:

```
draw_kde(['Attack', 'Defense', 'Total'], num_data, data_cs32_scaled_train, 'до масштабирова
```



Результат во многом похож на MeanNormalisation, только теперь и по оси X сдвиг не полностью устранен

In [156]:

```
# Point 2
```

In [157]:

```
num_data.head()
```

Out[157]:

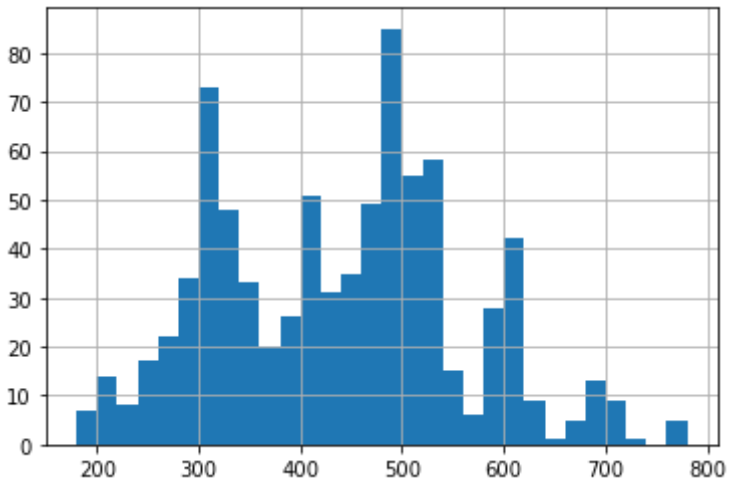
	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
0	318	45	49	49	65	65	45	1
1	405	60	62	63	80	80	60	1
2	525	80	82	83	100	100	80	1
3	625	80	100	123	122	120	80	1
4	309	39	52	43	60	50	65	1

In [158]:

```
def diagnostic_plots(df, variable):
    # гистограмма
    df[variable].hist(bins=30)
#     plt.show()
```

In [159]:

```
diagnostic_plots(num_data, 'Total')
```



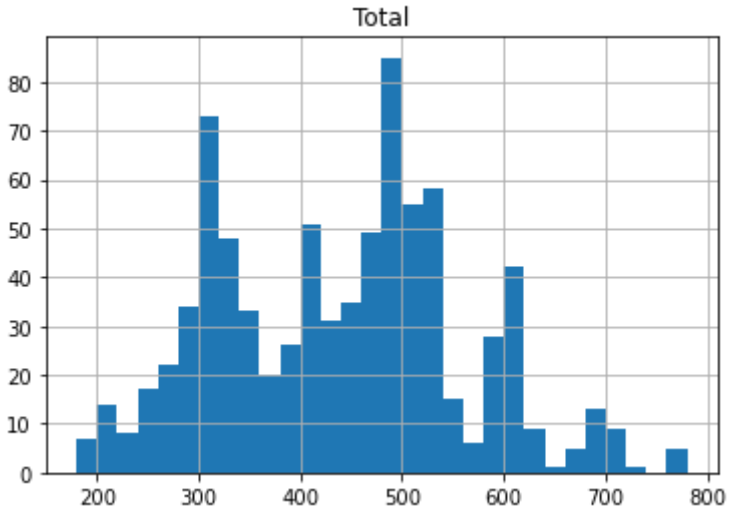
In [160]:

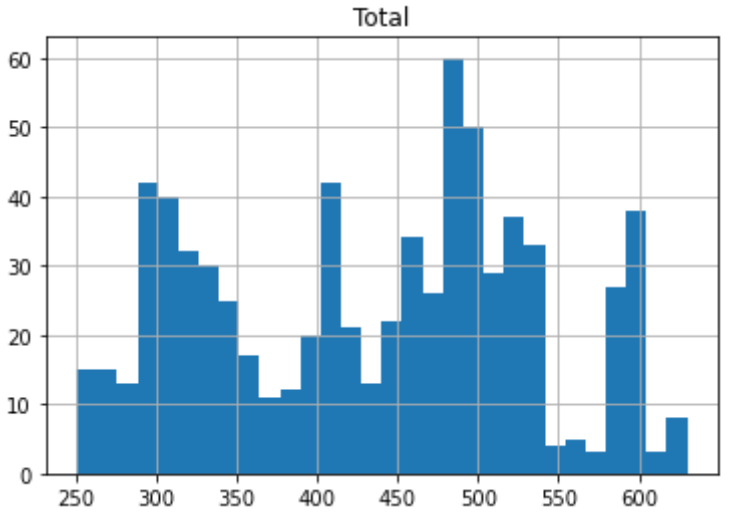
```
def get_outlier_boundaries(df, col, typ):
    if typ == 'SIGMA':
        K1 = 3
        lower_boundary = df[col].mean() - (K1 * df[col].std())
        upper_boundary = df[col].mean() + (K1 * df[col].std())

    elif typ== 'QUANTILE':
        lower_boundary = df[col].quantile(0.05)
        upper_boundary = df[col].quantile(0.95)

    elif typ == 'IRQ':
        K2 = 1.5
        IQR = df[col].quantile(0.75) - df[col].quantile(0.25)
        lower_boundary = df[col].quantile(0.25) - (K2 * IQR)
        upper_boundary = df[col].quantile(0.75) + (K2 * IQR)

    else:
        raise NameError('Unknown Outlier Boundary Type')

    return lower_boundary, upper_boundary
```

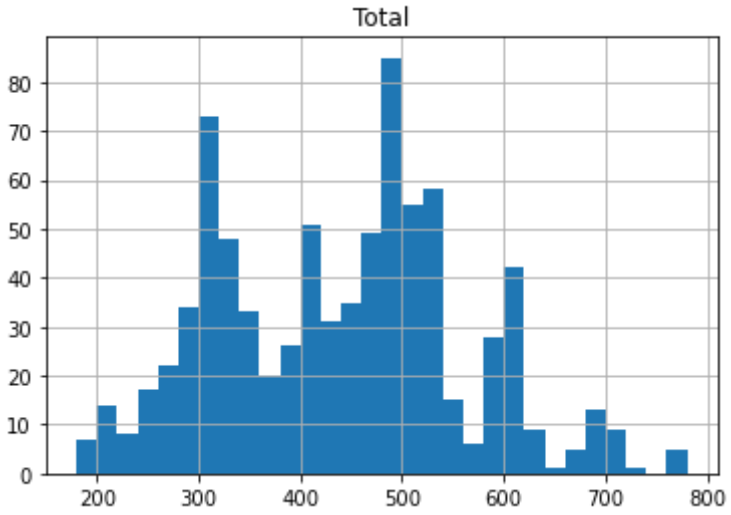
In [161]:

```
typs = ['SIGMA','QUANTILE','IRQ']
for t in typs:
    col = [ 'Total']
    lower_boundary, upper_boundary = get_outlier_boundaries(num_data, col,t)
     # Флаги для удаления выбросов
    outliers_temp = np.where(num_data[col] > upper_boundary, True, np.where(num_data[col] <
     # Удаление данных на основе флага
    data_trimmed = num_data.loc[~(outliers_temp), ]
#     title = 'Поле-{}, строк-{}'.format(col, data_trimmed.shape[0])
    print(t)
    diagnostic_plots(data_trimmed, col)
```

```
SIGMA
QUANTILE
IRQ
```







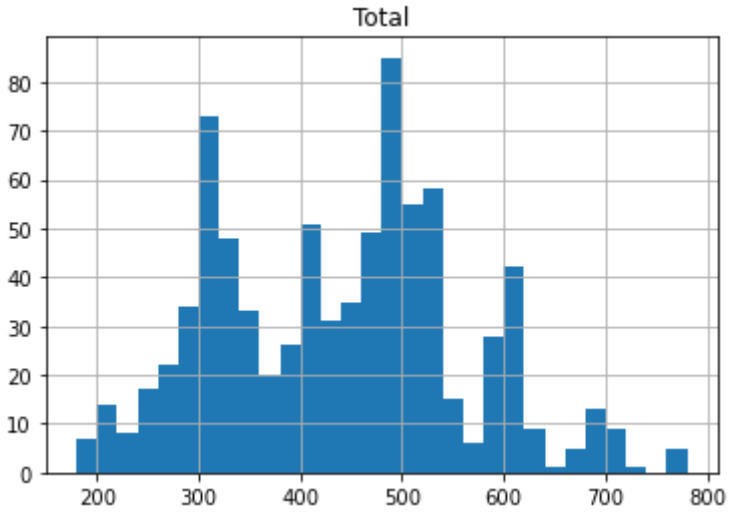
Видно, что подход с квантилями сразу дает весьма достойный результат

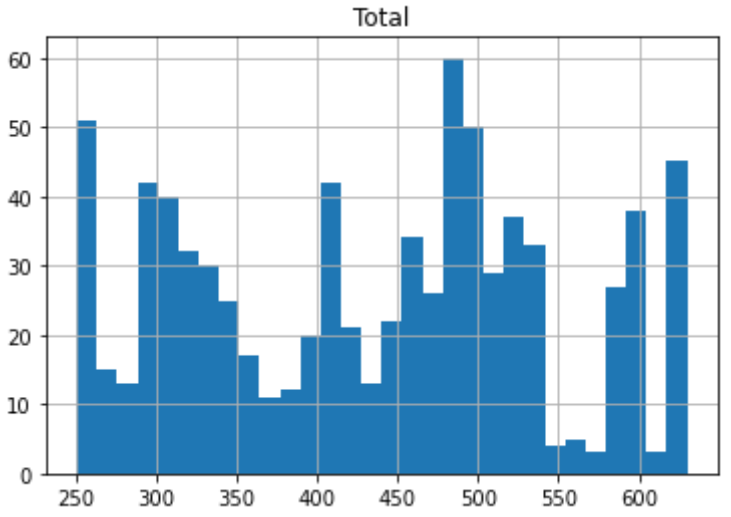
In [ ]:

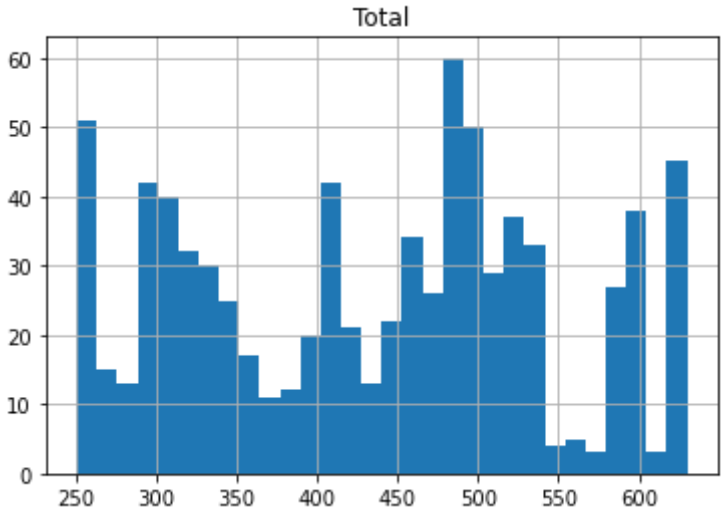
In [162]:

```
typs = ['SIGMA','QUANTILE','IRQ']
for t in typs:
    col = [ 'Total']
    lower_boundary, upper_boundary = get_outlier_boundaries(num_data, col,t)
     # Флаги для удаления выбросов
    outliers_temp = np.where(num_data[col] > upper_boundary, True, np.where(num_data[col] <
     # Удаление данных на основе флага
    num_data[col] = np.where(num_data[col] > upper_boundary, upper_boundary,
                                    np.where(num_data[col] < lower_boundary, lower_boundary, n
    print(t)
    diagnostic_plots(num_data, col)
```

```
SIGMA
QUANTILE
IRQ
```







Замена данных сработала не очень красиво в наших случаях, не просто заменив выбросы, а увеличив их почти до масимальных значений.

In [ ]:

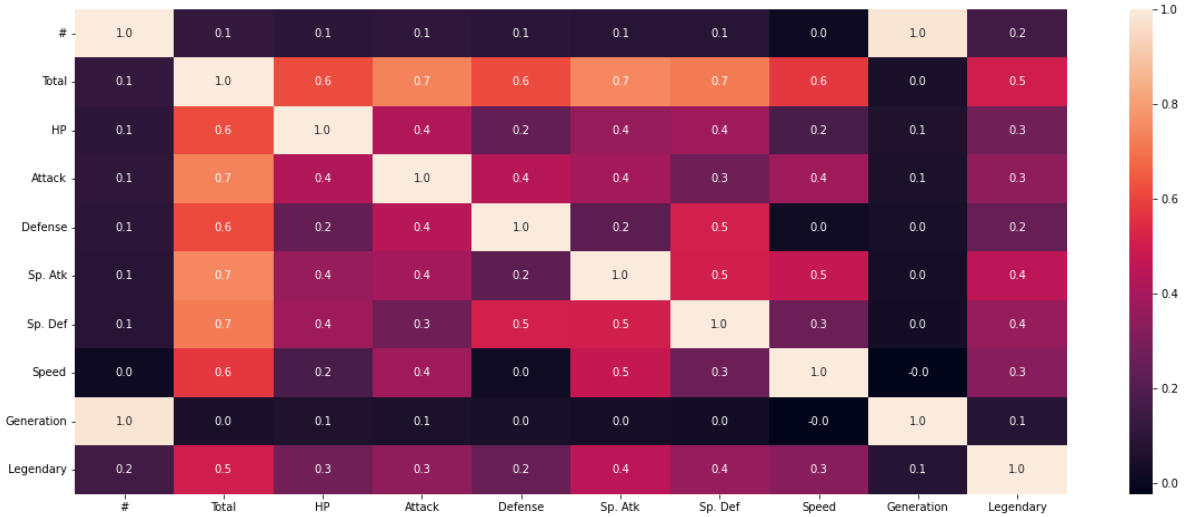
```
```

In [163]:

```
fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(20,8))
sns.heatmap(data.corr(), ax=ax, annot=True, fmt='.1f')
```

Out[163]:

```
<AxesSubplot:>
```



In [ ]:

```
```

In [164]:

```
dd = pd.read_csv('archive/full_grouped.csv')
dd
```

Out[164]:

	Date	Country/Region	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered
0	2020-01-22	Afghanistan	0	0	0	0	0	0	0
1	2020-01-22	Albania	0	0	0	0	0	0	0
2	2020-01-22	Algeria	0	0	0	0	0	0	0
3	2020-01-22	Andorra	0	0	0	0	0	0	0
4	2020-01-22	Angola	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
35151	2020-07-27	West Bank and Gaza	10621	78	3752	6791	152	2	0
35152	2020-07-27	Western Sahara	10	1	8	1	0	0	0
35153	2020-07-27	Yemen	1691	483	833	375	10	4	36
35154	2020-07-27	Zambia	4552	140	2815	1597	71	1	465
35155	2020-07-27	Zimbabwe	2704	36	542	2126	192	2	24

35156 rows × 10 columns

In [165]:

```
dd['time']=pd.to_datetime(dd['Date'],format="%Y-%m-%d")
dd.time
```

Out[165]:

```
0       2020-01-22
1       2020-01-22
2       2020-01-22
3       2020-01-22
4       2020-01-22
           ...
35151   2020-07-27
35152   2020-07-27
35153   2020-07-27
35154   2020-07-27
35155   2020-07-27
Name: time, Length: 35156, dtype: datetime64[ns]
```

In [166]:

```
dd['dtime'] = dd['Date'].astype('datetime64[s]')
```

In [167]:

```
dd['Day'] = dd.dtime.dt.day
dd.Day
```

Out[167]:

```
0        22
1        22
2        22
3        22
4        22
         ..
35151    27
35152    27
35153    27
35154    27
35155    27
Name: Day, Length: 35156, dtype: int64
```

Тут мы из датасета с данными о ковиде взяли поле даты и преобразовали его к типу datime, у которого уже легко взять такие поля, как день-месяц-год

In [ ]:

```

```

In [168]:

```
import scipy.stats as stats
from sklearn.svm import SVR
from sklearn.svm import LinearSVC
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import Lasso
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import mutual_info_classif, mutual_info_regression
from sklearn.feature_selection import SelectKBest, SelectPercentile
from IPython.display import Image
```

In [ ]:

```

```

Методы фильтрации

In [169]:

```
selector_1211 = VarianceThreshold(threshold=0.15)
selector_1211.fit(num_data)
# Значения дисперсий для каждого признака
selector_1211.variances_
```

Out[169]:

```
array([1.23564606e+04, 6.51204298e+02, 1.05216375e+03, 9.71195194e+02,
       1.06941010e+03, 7.73480494e+02, 8.43455494e+02, 2.75643594e+00])
```

In [170]:

```
new_data = num_data.drop('Total', axis=1)
```

In [171]:

```
#  Формирование DataFrame с сильными корреляциями
def make_corr_df(df):
    cr = df.corr()
    cr = cr.abs().unstack()
    cr = cr.sort_values(ascending=False)
    cr = cr[cr >= 0.4]
    cr = cr[cr < 1]
    cr = pd.DataFrame(cr).reset_index()
    cr.columns = ['f1', 'f2', 'corr']
    return cr
```

In [172]:

```
make_corr_df(new_data)
```

Out[172]:

	f1	f2	corr
0	Defense	Sp. Def	0.510747
1	Sp. Def	Defense	0.510747
2	Sp. Def	Sp. Atk	0.506121
3	Sp. Atk	Sp. Def	0.506121
4	Speed	Sp. Atk	0.473018
5	Sp. Atk	Speed	0.473018
6	Attack	Defense	0.438687
7	Defense	Attack	0.438687
8	HP	Attack	0.422386
9	Attack	HP	0.422386

In [173]:

```
# Обнаружение групп коррелирующих признаков
def corr_groups(cr):
    grouped_feature_list = []
    correlated_groups = []

    for feature in cr['f1'].unique():
        if feature not in grouped_feature_list:
            # находим коррелирующие признаки
            correlated_block = cr[cr['f1'] == feature]
            cur_dups = list(correlated_block['f2'].unique()) + [feature]
            grouped_feature_list = grouped_feature_list + cur_dups
            correlated_groups.append(cur_dups)
    return correlated_groups
```

In [174]:

```
corr_groups(make_corr_df(new_data))
```

Out[174]:

```
[['Sp. Def', 'Attack', 'Defense'],
 ['Sp. Def', 'Speed', 'Sp. Atk'],
 ['Attack', 'HP']]
```

Корреляционный подход нам весьма подошел, хотя и само пороговое значение корреляции пришлось немного уменьшить

Методы обертывания

In [175]:

```
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS

knn = KNeighborsClassifier(n_neighbors=3)
```

In [176]:

```
efs1 = EFS(knn,
           min_features=2,
           max_features=4,
           scoring='accuracy',
           print_progress=True,
           cv=5)


efs1 = efs1.fit(X_test, y_test, custom_feature_names=x_all.columns)

print('Best accuracy score: %.2f' % efs1.best_score_)
print('Best subset (indices):', efs1.best_idx_)
print('Best subset (corresponding names):', efs1.best_feature_names_)
```

```
Features: 91/91

Best accuracy score: 0.28
Best subset (indices): (3, 5)
Best subset (corresponding names): ('Defense', 'Sp. Def')
```

In [177]:

```
efs2 = EFS(knn,
           min_features=1,
           max_features=2,
           scoring='accuracy',
           print_progress=True,
           cv=5)

efs2 = efs2.fit(X_test, y_test, custom_feature_names=x_all.columns)

print('Best accuracy score: %.2f' % efs2.best_score_)
print('Best subset (indices):', efs2.best_idx_)
print('Best subset (corresponding names):', efs2.best_feature_names_)
```

```
Features: 28/28

Best accuracy score: 0.28
Best subset (indices): (3, 5)
Best subset (corresponding names): ('Defense', 'Sp. Def')
```

Обе попытки выдали нам одинаковые результаты, хоть и с маленькой точностью. Однако, повтор результата может побудить нас довериться ему.

In [ ]:

```

```

In [ ]:

```

```

In [178]:

```
# Используем L1-регуляризацию
e_lr1 = LogisticRegression(C=1000, solver='liblinear', penalty='l1', max_iter=500, random_s
e_lr1.fit(X_test, y_test)
# Коэффициенты регрессии
e_lr1.coef_
```

Out[178]:

```
array([[ 0.00044174, -0.01451076,  0.00602051, -0.00774738,  0.00272651,
        -0.00650975,  0.00971658],
       [ 0.00041432,  0.00285558, -0.0072538 ,  0.00787721,  0.00056511,
        -0.00575581, -0.00999194],
       [ 0.00064436, -0.01483897,  0.00062053,  0.00071967, -0.00599423,
         0.00923666, -0.00396246],
       [ 0.00062633, -0.00633129, -0.00334808,  0.00885312,  0.00112828,
         0.00437094,  0.01081077],
       [ 0.00062096,  0.02211308,  0.0061139 , -0.01463032,  0.00481425,
         0.0018714 , -0.01150729],
       [-0.00085975,  0.00740241, -0.00629096,  0.00303599, -0.00270999,
        -0.01398867,  0.00072755]])
```

In [179]:

```
# Все 7 признаков являются "хорошими"
sel_e_lr1 = SelectFromModel(e_lr1)
sel_e_lr1.fit(X_test, y_test)
sel_e_lr1.get_support()
```

Out[179]:

```
array([ True,  True,  True,  True,  True,  True,  True])
```

In [180]:

```
e_lr2 = LinearSVC(C=0.01, penalty="l1", max_iter=2000, dual=False)
e_lr2.fit(X_test, y_test)
# Коэффициенты регрессии
e_lr2.coef_
```

```
D:\ml\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Libline
ar failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[180]:

```
array([[ 0.00000000e+00, -5.95422503e-03,  8.53685972e-04,
        -2.71060113e-03,  0.00000000e+00, -2.10478464e-03,
         1.68921523e-03],
       [-1.39320238e-03, -9.98841784e-04,  0.00000000e+00,
         1.42757728e-03,  5.88361735e-05,  0.00000000e+00,
        -2.22605070e-03],
       [-3.91798458e-04, -5.47329758e-03,  0.00000000e+00,
         0.00000000e+00, -9.46323425e-04,  1.33583037e-03,
        -1.54188333e-03],
       [-1.92735430e-04, -7.82489820e-03,  0.00000000e+00,
         1.79092422e-04,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [ 0.00000000e+00,  1.18270259e-03,  1.42905760e-03,
        -5.21998113e-03,  7.53305957e-04, -4.57991881e-04,
        -5.29487869e-03],
       [-1.44426988e-03,  0.00000000e+00,  0.00000000e+00,
         1.32774643e-04,  0.00000000e+00, -1.53767673e-03,
         0.00000000e+00]])
```

In [181]:

```
# Все 7 признаков являются "хорошими"
sel_e_lr2 = SelectFromModel(e_lr2)
sel_e_lr2.fit(X_test, y_test)
sel_e_lr2.get_support()
```

```
D:\ml\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Libline
ar failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[181]:

```
array([ True,  True,  True,  True,  True,  True,  True])
```

In [ ]:

In [182]:

```
# Используем L1-регуляризацию
e_ls1 = Lasso(random_state=1)
e_ls1.fit(X_test, y_test)
# Коэффициенты регрессии
list(zip(X_test.columns, e_ls1.coef_))
```

Out[182]:

```
[('Total', -0.00021237631034044932),
 ('HP', 0.007888506427171038),
 ('Attack', -0.0),
 ('Defense', -0.0),
 ('Sp. Atk', 0.0),
 ('Sp. Def', -0.0),
 ('Speed', -0.003058263057428995)]
```

In [183]:

```
sel_e_ls1 = SelectFromModel(e_ls1)
sel_e_ls1.fit(X_test, y_test)
list(zip(X_test.columns, sel_e_ls1.get_support()))
```

Out[183]:

```
[('Total', True),
 ('HP', True),
 ('Attack', False),
 ('Defense', False),
 ('Sp. Atk', False),
 ('Sp. Def', False),
 ('Speed', True)]
```

In [ ]:

```

```

In [ ]:

```

```

In [184]:

```
dtc1 = DecisionTreeClassifier()
rfc1 = RandomForestClassifier()
gbc1 = GradientBoostingClassifier()
dtc1.fit(X_test, y_test)
rfc1.fit(X_test, y_test)
gbc1.fit(X_test, y_test)

# Важность признаков
dtc1.feature_importances_, sum(dtc1.feature_importances_)
```

Out[184]:

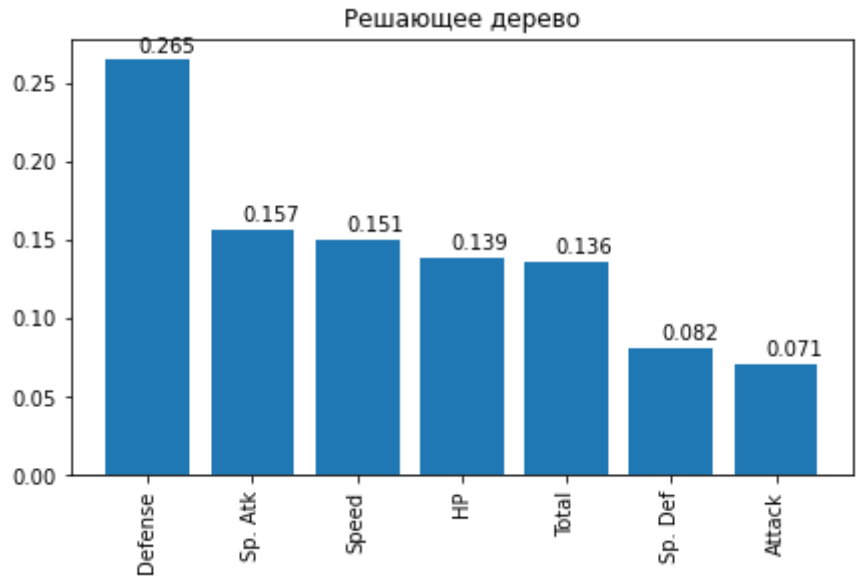
```
(array([0.13609838, 0.13872571, 0.07080153, 0.26492692, 0.15705504,
        0.08154458, 0.15084784]),
 1.0)
```

In [185]:

```
from operator import itemgetter

def draw_feature_importances(tree_model, X_dataset, title, figsize=(7,4)):
    """
    Вывод важности признаков в виде графика
    """
    # Сортировка значений важности признаков по убыванию
    list_to_sort = list(zip(X_dataset.columns.values, tree_model.feature_importances_))
    sorted_list = sorted(list_to_sort, key=itemgetter(1), reverse = True)
    # Названия признаков
    labels = [x for x,_ in sorted_list]
    # Важности признаков
    data = [x for _,x in sorted_list]
    # Вывод графика
    fig, ax = plt.subplots(figsize=figsize)
    ax.set_title(title)
    ind = np.arange(len(labels))
    plt.bar(ind, data)
    plt.xticks(ind, labels, rotation='vertical')
    # Вывод значений
    for a,b in zip(ind, data):
        plt.text(a-0.1, b+0.005, str(round(b,3)))
    plt.show()
    return labels, data
```

In [186]:

```
_,_=draw_feature_importances(dtc1, X_test_df, 'Решающее дерево')
```



На поколение больше всего влияла защита

In [133]:

```
xall = num_data.drop('Legendary', axis=1)
xall
```

Out[133]:

	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation
0	45	49	49	65	65	45	1
1	60	62	63	80	80	60	1
2	80	82	83	100	100	80	1
3	80	100	123	122	120	80	1
4	39	52	43	60	50	65	1
...	...	...	...	...	...	...	...
795	50	100	150	100	150	50	6
796	50	160	110	160	110	110	6
797	80	110	60	150	130	70	6
798	80	160	60	170	130	80	6
799	80	110	120	130	90	70	6

800 rows × 7 columns

In [134]:

```
X_train, X_test, y_train, y_test = train_test_split(xall, num_data['Legendary'],
                                                    test_size=0.2,
                                                    random_state=1)
# Преобразуем массивы в DataFrame
X_train_df = arr_to_df(X_train,xall)
X_test_df = arr_to_df(X_test,xall)

X_train_df.shape, X_test_df.shape
```

Out[134]:

```
((640, 7), (160, 7))
```

In [135]:

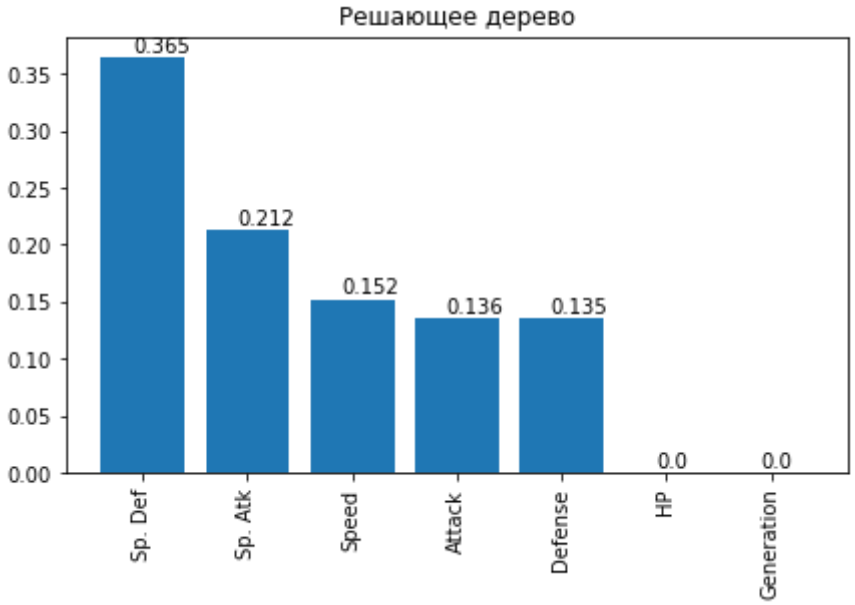
```
dtc1 = DecisionTreeClassifier()
rfc1 = RandomForestClassifier()
gbc1 = GradientBoostingClassifier()
dtc1.fit(X_test, y_test)
rfc1.fit(X_test, y_test)
gbc1.fit(X_test, y_test)

# Важность признаков
dtc1.feature_importances_, sum(dtc1.feature_importances_)
```

Out[135]:

```
(array([0.        , 0.13576153, 0.13513514, 0.21235521, 0.3647354 ,
        0.15201273, 0.        ]),
 1.0)
```

In [136]:

```
_,_=draw_feature_importances(dtc1, X_test_df, 'Решающее дерево')
```



А вот на легендарность покемона больше всего влияют особые значения Атаки и защиты