Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Использование методов машинного обучения для прогнозирования очков в матчах NBA

Студент ИУ5-34М (Группа) _____ (Подпись, дата) Д.А. Богданов (И.О.Фамилия)

Руководитель _____ (Подпись, дата) Ю.Е. Гапанюк (И.О.Фамилия)

Консультант _____ (Подпись, дата) _____ (И.О.Фамилия)

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Использование методов машинного обучения для прогнозирования очков в матчах NBA

Студент группы ИУ5-34М

Богданов Дмитрий Александрович
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

исследовательская

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения НИР: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 17 нед.

Техническое задание _____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 25 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель НИР _____ Ю.Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

Студент _____ Д.А. Богданов
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

Введение

Задачей научно-исследовательской работы является построение модели машинного обучения для прогнозирования очков больше/меньше в матчах баскетбольной лиги США.

Больше/меньше — это предлагаемая линия ставок, где веб-сайт ставок предоставляет номер, который вы должны выбрать, если результат будет больше или меньше этого числа. Эта модель ставок будет относиться к наиболее распространенному значению больше/меньше, которое представляет собой общее количество очков (сумма обеих команд), набранных в конце игры. Например: если значение больше/меньше равно 225, и вы думаете, что результат будет ниже этого значения, вы возьмете меньше. Затем, после того как в игре команда А наберет 110 очков, а команда Б наберет 105 очков (110 + 105 = 115), вы выиграли пари, потому что 115 меньше 225.

Модель машинного обучения будет тренироваться на статистике матчей NBA, начиная с 2011 года. Данные взяты с сайта баскетбольной статистики stats.nba.com.

Сбор и очистка данных

Загрузка всех данных:

```
gamesdata = pd.read_csv("data/games.csv")

point_total = gamesdata.PTS_home + gamesdata.PTS_away
gamesdata["point_total"] = point_total
```

Применение ограничения:

```
gamesdata = gamesdata.loc[gamesdata.SEASON >= 2011]
gamesdata.sort_values(by=["GAME_DATE_EST"])
gamesdata = gamesdata.reset_index()
```

В некоторых столбцах были ошибки ввода, например, перед годами сезона стояло дополнительное число (например, 2016 в файле было как 22016).

Устраним это:

```
recorddata["SEASON_ID"] = recorddata["SEASON_ID"].apply(lambda x: x - 20000)
```

Парсинг данных

Чтобы базовое значение можно было сравнить с точностью полученной модели, необходимо очистить данные о предыдущих линиях ставок и сравнить их с результатом, чтобы рассчитать среднюю абсолютную ошибку линии ставок. Для этого использовалась библиотека scrapy.

```
class tableSpider(scrapy.Spider):
    name= 'table'

    start_url =[

"https://sportsdatabase.com/nba/query?output=default&sdql=date%2C+team%2C+sit
e%2C+o%3Ateam%2C+total%2C++points%2C+o%3Apoints+%40season%3E2010&submit=++S+D
+Q+L+%21++"
    ]

    def parse(self, response):
        for row in response.xpath('//*[@class="dataTables_wrapper no-
footer"]//tbody/tr'):
            if row.xpath('td[3]//text()').extract_first() == "home":
                yield {
                    'date' : row.xpath('td[1]//text()').extract_first(),
                    'team' : row.xpath('td[2]//text()').extract_first(),
                    'site' : row.xpath('td[3]//text()').extract_first(),
                    'o:team' : row.xpath('td[4]//text()').extract_first(),
                    'total' : row.xpath('td[5]//text()').extract_first(),
                    'points' : row.xpath('td[6]//text()').extract_first(),
                    'o:points' : row.xpath('td[7]//text()').extract_first(),
                }
```

Отсюда легко получить среднюю абсолютную ошибку линий ставок.

```
import pandas as pd

data = pd.read_csv("bets.csv")

# The rows have duplicates because of home/away, so remove every other row
data = data[data.site == "home"]

# Adding points for both teams together
data["real_total"] = data["points"] + data["o:points"]

# Calculating the difference between the line and outcome
data["error"] = data["real_total"] - data["total"]

# Making the error the absolute value
data["error"] = data.error.abs()

# Calculating mean absolute error between the lines and outcome
n = len(data)
total_error = data.error.sum()
mae_lines = total_error / n

print ("MAE over last 10 years",mae_lines)
```

```
MAE over last 10 years 13.42853888937936
```

Предварительная обработка и разведочный анализ данных

Подготовим данные для обучения:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pandas as pd
from sklearn import preprocessing

# Read the data
X_full = pd.read_csv("cleandata2.csv")
X_full["GAME_DATE_EST"] =
pd.to_datetime(X_full["GAME_DATE_EST"],infer_datetime_format=True)
X_full = X_full.sort_values(by=["GAME_DATE_EST"], ascending=False)
X_full = X_full.reset_index()
```

```
# Label / One-Hot encoding team IDs and season
le = preprocessing.OneHotEncoder()
le2 = preprocessing.LabelEncoder()
ohe_home = le.fit_transform(X_full[["HOME_TEAM_ID"]]).toarray()
ohe_away = le.transform(X_full[["VISITOR_TEAM_ID"]]).toarray()
X_full["SEASON"] = X_full["SEASON"] - 2010

# Drop old columns
X_full = X_full.drop(columns =[
    "HOME_TEAM_ID","VISITOR_TEAM_ID", "PTS_home",
"PTS_away","GAME_DATE_EST","index"], axis=1)
```

Вывод данных:

```
X_full.head(5)
```

```
   SEASON  point_total  point_average_last10  point_againts_average_last10  \
0       9        178.0                 98.55                        108.05
1       9        266.0                116.65                        109.30
2       9        251.0                112.35                        109.50
3       9        206.0                112.65                        112.55
4       9        236.0                120.80                        117.30

   away_point_average_last10  away_point_againts_average_last10  cgp  wins  \
0                     122.20                             107.50   59    21
1                     107.55                             107.90   59    40
2                     115.95                             107.40   59    40
3                     106.85                             110.05   59    25
4                     116.20                             109.40   59    26

   losses  winpercent  ...  losses_away  winpercen_away  hometeam-homewins  \
0      38       0.356  ...            8           0.864                  9
1      19       0.678  ...           23           0.617                 24
2      19       0.678  ...           17           0.712                 23
3      34       0.424  ...           41           0.328                 12
4      33       0.441  ...           13           0.776                 12

   hometeam-awaywins  awayteam-homewins  awayteam-awaywins  \
0                 12                 27                 24
1                 16                 28                  9
2                 17                 23                 19
3                 13                 11                  9
4                 14                 21                 24

   avgpointtotal_home  avgpointtotal_away  meanpointtotal   diff
```

```
0              205.64              222.40          214.02 -16.76
1              226.24              216.72          221.48   9.52
2              219.00              221.72          220.36  -2.72
3              217.48              221.00          219.24  -3.52
4              227.04              224.28          225.66   2.76

[5 rows x 22 columns]
```

Здесь мы можем видеть расположение наших данных на данный момент. Я объясню, что означают некоторые столбцы:

point_average_last10 - среднее количество очков, набранных хозяевами поля в последних 10 играх.

points_againts_average_last10 - среднее количество очков, набранных против хозяев поля в последних 10 играх.

Следующие два столбца такие же, но для команды посетителей.

Эти столбцы должны служить способом определения сильных оборонительных команд/сильных атакующих команд (которые, надеюсь, должны указывать на исход текущей игры).

cpg - текущие сыгранные игры (сколько игр они сыграли в этом сезоне) (_away то же самое, но для команды гостей)

```
X_full.describe()
```

```
             SEASON   point_total  point_average_last10  \
count  11915.000000  11915.000000          11915.000000
mean       4.934117    206.359295            100.741507
std        2.463316     21.943288              5.806101
min        1.000000    134.000000             83.300000
25%        3.000000    191.000000             96.550000
50%        5.000000    205.000000             99.800000
75%        7.000000    221.000000            104.750000
max        9.000000    329.000000            121.900000

       point_againts_average_last10  away_point_average_last10  \
count                  11915.000000               11915.000000
mean                     101.277327                 100.718275
std                        6.184010                   5.794636
```

```
min                        83.650000                 83.500000
25%                        96.900000                 96.550000
50%                       101.500000                 99.800000
75%                       105.850000                104.750000
max                       123.300000                122.200000

       away_point_againts_average_last10           cgp          wins  \
count                       11915.000000  11915.000000  11915.000000
mean                          101.247986     38.852119     20.076290
std                             6.141684     25.445032     15.677251
min                            83.850000      0.000000      0.000000
25%                            96.750000     16.000000      7.000000
50%                           101.500000     38.000000     17.000000
75%                           105.850000     60.000000     30.000000
max                           122.300000     82.000000     73.000000

              losses    winpercent  ...   losses_away  winpercen_away  \
count  11915.000000  11915.000000  ...  11915.000000    11915.000000
mean      18.776081      0.497770  ...     18.735124        0.504076
std       13.778356      0.208152  ...     13.736672        0.207207
min        0.000000      0.000000  ...      0.000000        0.000000
25%        7.000000      0.364000  ...      7.000000        0.371000
50%       17.000000      0.500000  ...     17.000000        0.509000
75%       28.000000      0.636000  ...     28.000000        0.639000
max       69.000000      1.000000  ...     71.000000        1.000000

       hometeam-homewins  hometeam-awaywins  awayteam-homewins  \
count       11915.000000       11915.000000       11915.000000
mean           11.566093           8.510197          11.823164
std             9.028124           6.986531           9.047625
min             0.000000           0.000000           0.000000
25%             4.000000           3.000000           4.000000
50%            10.000000           7.000000          10.000000
75%            18.000000          13.000000          18.000000
max            40.000000          34.000000          40.000000

       awayteam-awaywins  avgpointtotal_home  avgpointtotal_away  \
count       11915.000000        11915.000000        11915.000000
mean            8.307176          201.261316          201.245986
std             6.865404            9.153747            9.144523
min             0.000000          174.920000          174.920000
25%             3.000000          194.960000          194.960000
50%             7.000000          200.280000          200.280000
75%            13.000000          207.680000          207.680000
max            34.000000          230.120000          229.440000

       meanpointtotal          diff
count    11915.000000  11915.000000
mean       201.253651      0.015331
```

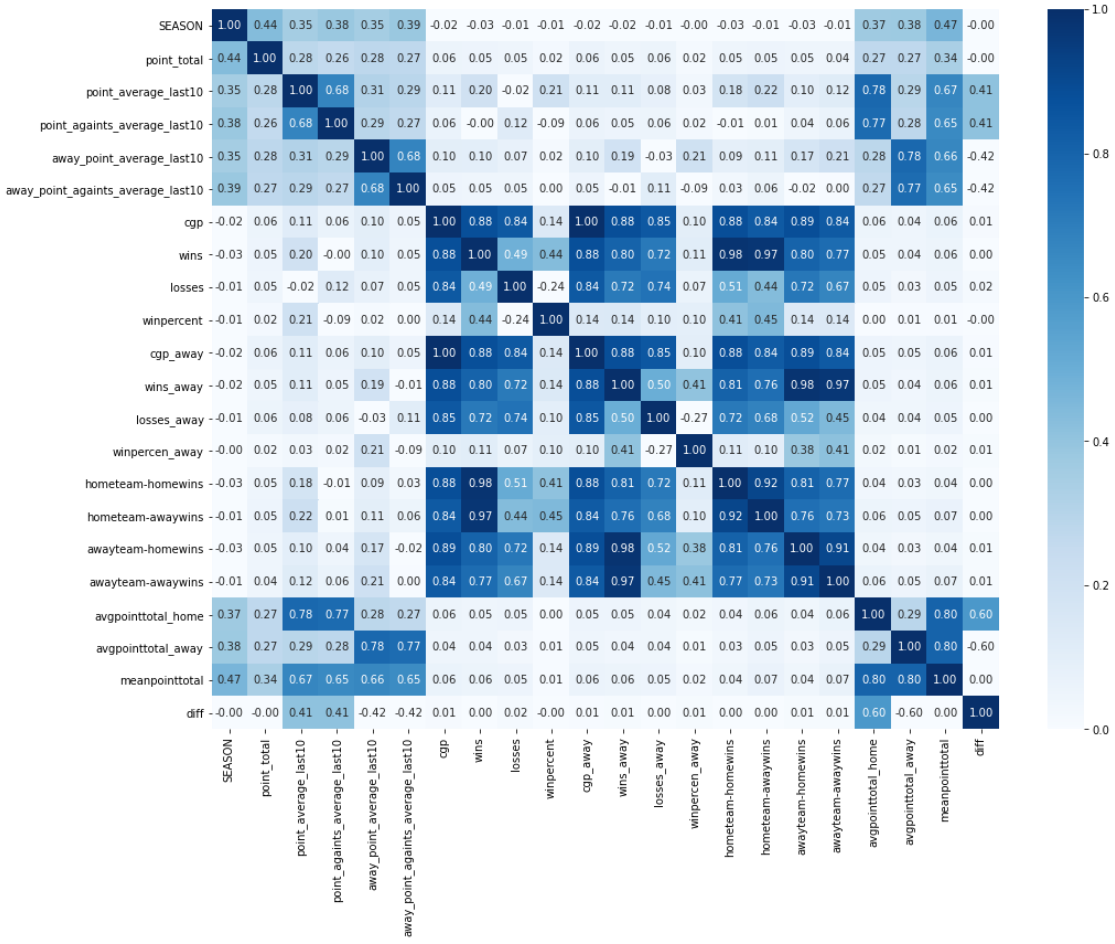
```
std          7.344656     10.911044
min        177.060000    -35.640000
25%        196.020000     -7.600000
50%        200.820000     -0.040000
75%        206.540000      7.635839
max        227.860000     36.000000

[8 rows x 22 columns]
```

```
X_full.point_total.describe()
```

```
count    11915.000000
mean       206.359295
std         21.943288
min        134.000000
25%        191.000000
50%        205.000000
75%        221.000000
max        329.000000
Name: point_total, dtype: float64
```

Проверка, чтобы увидеть корреляцию между всеми функциями. Как и ожидалось, существует много корреляций между функциями. Важно помнить об этом при выборе моделей/параметров.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(16, 12))
sns.heatmap(X_full.corr(),
            cmap="Blues",annot=True, fmt='.2f', vmin=0);
```



Есть некоторые сильно коррелированные функции, они будут рассмотрены позже. Характеристики с наибольшей корреляцией с целью: сезон, среднее общее количество баллов и point_average_last10/away_points_average_last10. Добавление столбцов с one-hot encoding:

```
ohe_home_df = pd.DataFrame(ohe_home)
ohe_away_df = pd.DataFrame(ohe_away,
columns=list("abcdefghijklmnopqrstuvwxyzABCD"))
X_full = pd.concat([X_full,ohe_home_df], axis=1)
X_full = pd.concat([X_full,ohe_away_df], axis=1)
print(X_full.shape)
```

```
(11915, 82)
```

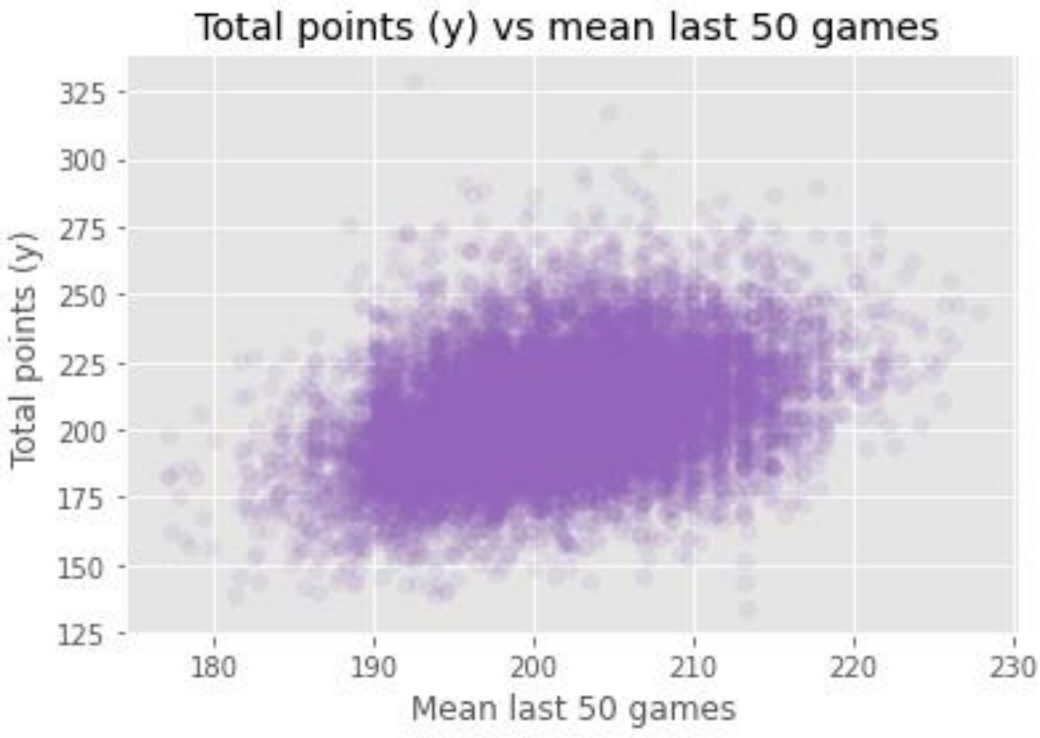
```
X_full.head(5)
```

```
   SEASON  point_total  point_average_last10  point_againts_average_last10  \
0       9        178.0                 98.55                        108.05
1       9        266.0                116.65                        109.30
2       9        251.0                112.35                        109.50
3       9        206.0                112.65                        112.55
4       9        236.0                120.80                        117.30

   away_point_average_last10  away_point_againts_average_last10  cgp  wins  \
0                     122.20                             107.50   59    21
1                     107.55                             107.90   59    40
2                     115.95                             107.40   59    40
3                     106.85                             110.05   59    25
4                     116.20                             109.40   59    26

   losses  winpercent  ...    u    v    w    x    y    z    A    B    C    D
0      38       0.356  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1      19       0.678  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2      19       0.678  ...  0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
3      34       0.424  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0
4      33       0.441  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 82 columns]
```

Разработка функций

Давайте посмотрим на некоторые отношения между функциями и целью:

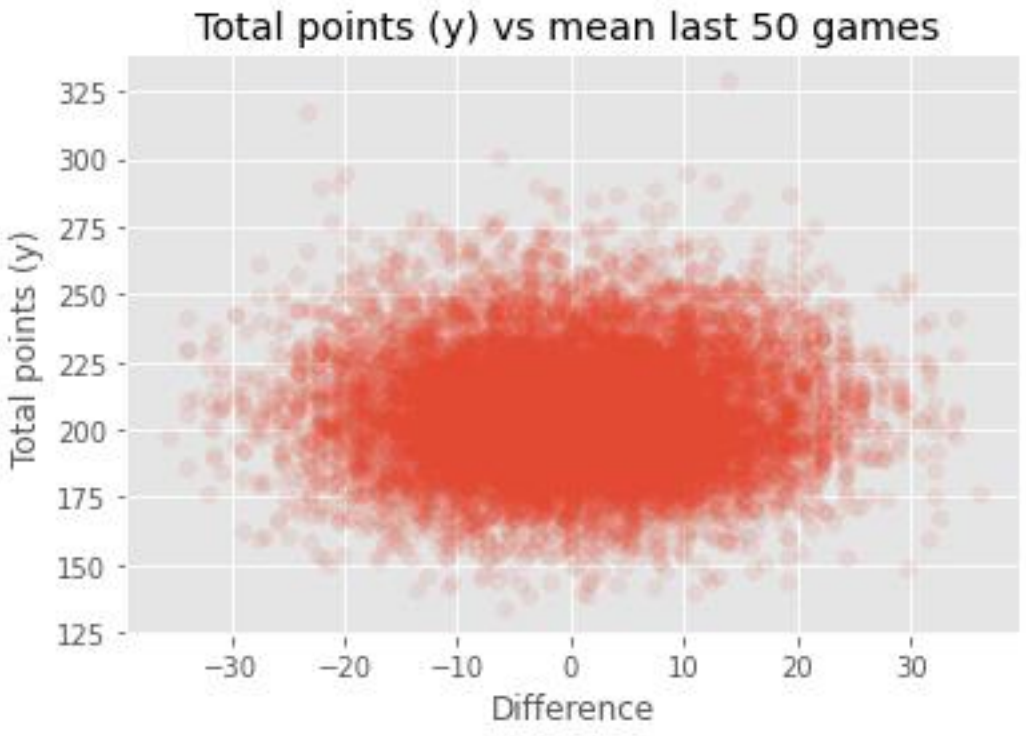
```
import matplotlib.pyplot as plt

with plt.style.context("ggplot"):
    plt.scatter(X_full.meanpointtotal, X_full.point_total, marker="o",
alpha=0.1, color='#9467bd')
    plt.xlabel("Mean last 50 games")
    plt.ylabel("Total points (y)")
    plt.title("Total points (y) vs mean last 50 games ")
fig1 = plt.figure();
```




```
<Figure size 432x288 with 0 Axes>
```

Шумов много и это не самая сильная корреляция. Тем не менее, существует четкая тенденция между общим количеством набранных очков и средним показателем обеих команд за последние 50 сыгранных игр.

```
with plt.style.context("ggplot"):
    plt.scatter(X_full["diff"], X_full.point_total, marker="o", alpha=0.1)
    plt.xlabel("Difference")
    plt.ylabel("Total points (y)")
    plt.title("Total points (y) vs mean last 50 games ")
fig2 = plt.figure();
```

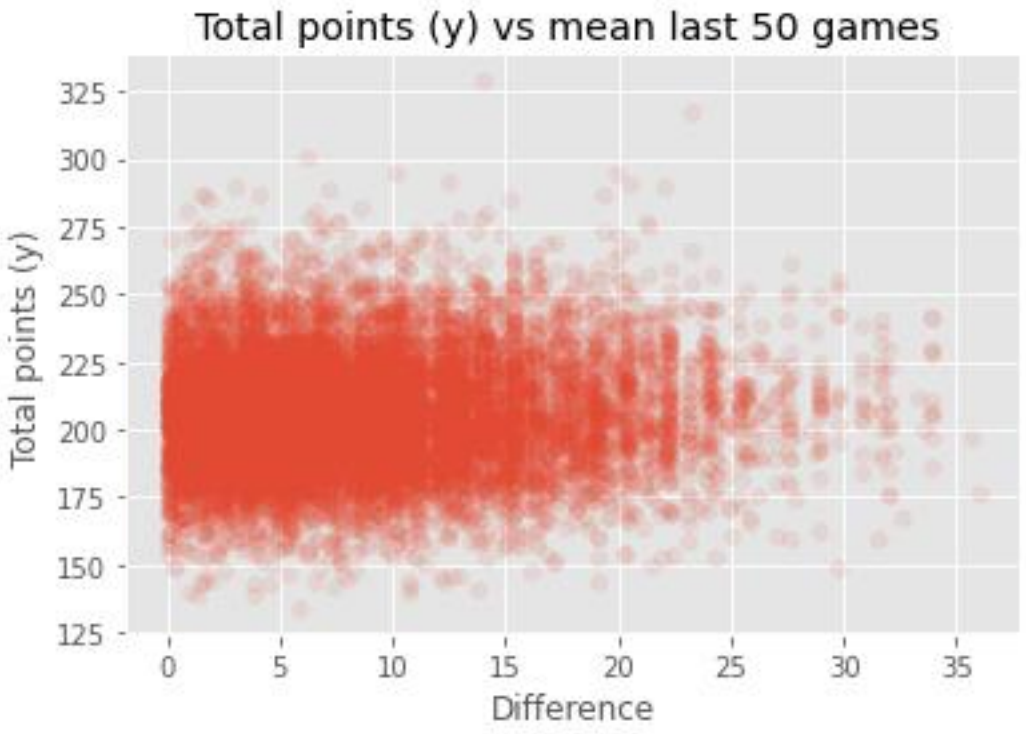



```
<Figure size 432x288 with 0 Axes>
```

Этот график показывает, что существует небольшая тенденция к средней результативности игры, когда между двумя командами существует большая разница. Но если команды ближе по среднему количеству последних 50 игр, то это может быть игра с высоким или низким результатом.

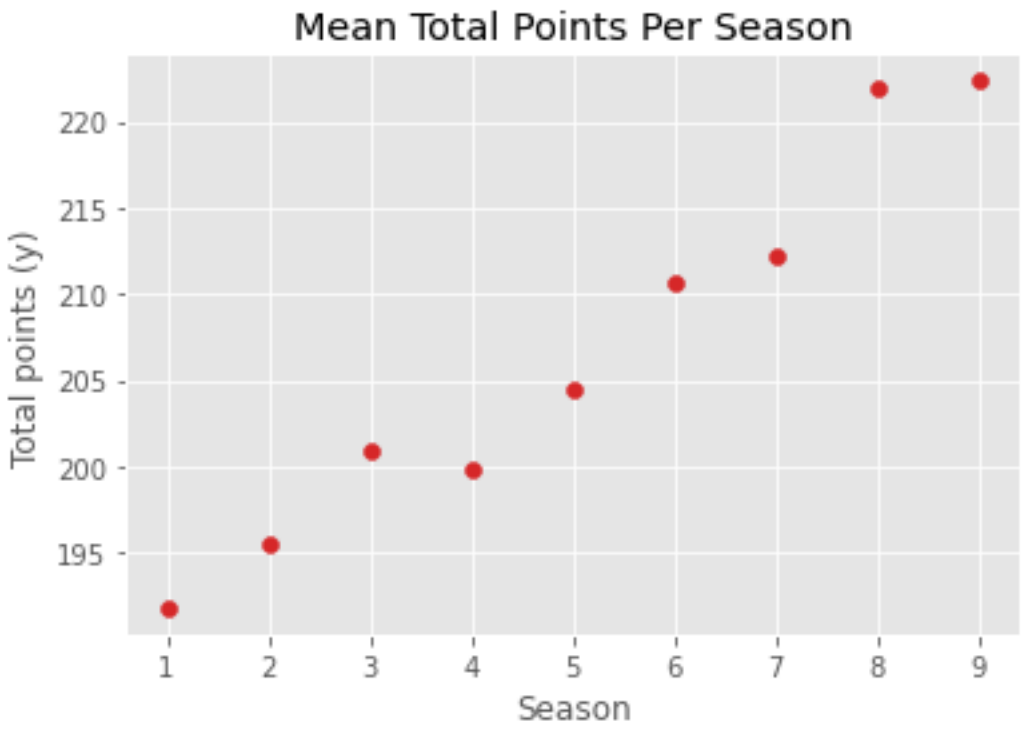
Эту тенденцию немного легче увидеть, если нанести на график абсолютное значение разницы.

```
with plt.style.context("ggplot"):
    plt.scatter(X_full["diff"].abs(), X_full.point_total, marker="o",
alpha=0.1)
    plt.xlabel("Difference")
    plt.ylabel("Total points (y)")
    plt.title("Total points (y) vs mean last 50 games ")
fig3 = plt.figure();
```




```
<Figure size 432x288 with 0 Axes>
```

```
#mean_df = X_full.copy()
#mean_df["target"] = y
season_avg = X_full.groupby(by=["SEASON"]).mean()

with plt.style.context("ggplot"):
    plt.scatter(season_avg.index, season_avg.point_total, color = "#d62728" )
    plt.xlabel("Season")
    plt.ylabel("Total points (y)")
    plt.title("Mean Total Points Per Season")
fig4 = plt.figure();
```




```
<Figure size 432x288 with 0 Axes>
```

В среднем количество набранных очков со временем увеличивается. От 1 сезона 2011 году, вплоть до 2019 года

Модели

Метод проверки заключался в ручном разделении test train 90:10 без случайной перетасовки. То есть, когда мы используем все старые данные для обучения и новейшие 10% для проверки. Это было выбрано потому, что это был упрощенный способ получить хороший проверочный тест для этого типа модели. Перекрестная проверка не является хорошей идеей, потому что тогда мы будем тренироваться на данных, к которым у нас не будет доступа, если мы будем использовать эту модель на практике. Например, мы можем

потренироваться на данных за 2019 год и увидеть, что это сезон с высокими показателями результативности, а затем при прогнозировании игры в начале 2019 года мы получим лучшие результаты, чем должны. Теоретически лучшим методом был бы метод проверки, исключающий будущее, потому что именно так модель будет работать на практике. Для простоты мы использовали 90:10. Который должен дать достаточно приличные результаты для задачи.

```
y = X_full["point_total"]
X_full = X_full.drop(columns=['point_total'])

test = len(X_full) - int(len(X_full)*0.9)

X_train = X_full.iloc[test:]
X_valid = X_full.iloc[:test]
y_train = y.iloc[test:]
y_valid = y.iloc[:test]


from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_absolute_error

EN = ElasticNet().fit(X_train,y_train)

pred_EN = EN.predict(X_valid)
mae_EN = mean_absolute_error(pred_EN, y_valid)
print ("mae",mae_EN)

mae 15.979896935372482

from sklearn.linear_model import Lasso

ls = Lasso().fit(X_train, y_train)

pred_ls = ls.predict(X_valid)
mae_ls = mean_absolute_error(pred_ls, y_valid)
print("mae",mae_ls)

mae 15.939184128274167

from sklearn.linear_model import Ridge

rg = Ridge().fit(X_train, y_train)
```

```
pred_rg = rg.predict(X_valid)
mae_rg = mean_absolute_error(pred_rg, y_valid)
print("mae",mae_rg)
```

```
mae 16.059752446474615
```

```
from sklearn.linear_model import TheilSenRegressor

ts = TheilSenRegressor().fit(X_train, y_train)
pred_ts = ts.predict(X_valid)
mae_ts = mean_absolute_error(pred_ts,y_valid)
print("mae",mae_ts)
```

```
mae 16.05934497025308
```

```
#Lin reg
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

lin = LinearRegression(normalize=True, ).fit(
    X_train,y_train
)
pred_lin = lin.predict(X_valid)
mae_lin = mean_absolute_error(pred_lin, y_valid)
print("mae", mae_lin)
```

```
mae 16.06511775279205
```

```
from sklearn.neural_network import MLPRegressor

regr = MLPRegressor(random_state=1, max_iter=1000).fit(
    X_train, y_train
)

pred_mlp = regr.predict(X_valid)
mae_regr = mean_absolute_error(pred_mlp, y_valid)
print("mae",mae_regr)
```

```
mae 16.590031854345042
```

```
import xgboost as xgb

xg = xgb.XGBRegressor(
    booster="gblinear",
    objective="reg:squarederror",
    base_score="1",
    eval_metric="cox-nloglik"
    ).fit(X_train,y_train)
pred_xg = xg.predict(X_valid)
xg_regr = mean_absolute_error(pred_xg, y_valid)
```

```
print("mae",xg_regr)
print("done")
```

```
mae 16.092118333650117
done
```

Расчет MAE для угадывания среднего значения за последний сезон каждого года.

```
error = []
mean_last_season = season_avg.iloc[6,0]
for i in y_valid:
    error.append(abs(mean_last_season - i))
mae_mean = sum(error) / len(error)

print(mae_mean)
print(mean_last_season)
```

```
17.8650445808526
212.14544138929088
```

```
from hpsklearn import HyperoptEstimator, any_regressor
from hyperopt import tpe

estim = HyperoptEstimator(regressor=any_regressor("svr"))

estim.fit(X_train.values, y_train.values)

print(estim.score(X_valid,y_valid))
print(estim.best_model())
```

```
100%|██████████| 1/1 [00:02<00:00,  2.79s/trial, best loss:
1.4367301109787802]
100%|██████████| 2/2 [00:06<00:00,  3.16s/trial, best loss:
0.9270445930580935]
100%|██████████| 3/3 [00:11<00:00,  3.85s/trial, best loss:
0.8906897421481069]
100%|██████████| 4/4 [00:02<00:00,  1.38trial/s, best loss:
0.8906897421481069]
100%|██████████| 5/5 [00:15<00:00,  3.13s/trial, best loss:
0.8906897421481069]
100%|██████████| 6/6 [00:42<00:00,  7.10s/trial, best loss:
0.8842060154887895]
100%|██████████| 7/7 [00:04<00:00,  1.61trial/s, best loss:
0.8842060154887895]
100%|██████████| 8/8 [00:04<00:00,  1.67trial/s, best loss:
0.8842060154887895]
100%|██████████| 9/9 [00:14<00:00,  1.58s/trial, best loss:
0.8842060154887895]
100%|██████████| 10/10 [00:18<00:00,  1.85s/trial, best loss:
```

```
0.8842060154887895]
[22:25:38] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
[22:26:23] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
-0.47623640650593857
{'learner': XGBRegressor(base_score=0.5, booster='gbtree',
             colsample_bylevel=0.8886553695663921, colsample_bynode=1,
             colsample_bytree=0.6068338278675369,
gamma=0.0017889282555567038,
             gpu_id=-1, importance_type='gain', interaction_constraints='',
             learning_rate=0.0013963222902296055, max_delta_step=0,
max_depth=7,
             min_child_weight=19, missing=nan, monotone_constraints='()',
             n_estimators=5400, n_jobs=0, num_parallel_tree=1,
             objective='reg:linear', random_state=1,
             reg_alpha=0.22448614695919908, reg_lambda=2.9482948529431052,
             scale_pos_weight=1, seed=1, subsample=0.8571414026048771,
             tree_method='exact', validate_parameters=1, verbosity=None),
'preprocs': (MinMaxScaler(feature_range=(0.0, 1.0)),), 'ex_preprocs': ()}
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_testing = scaler.fit_transform(X_train)
X_testing_valid = scaler.transform(X_valid)
```

```
xg1 = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
             colsample_bylevel=0.8886553695663921, colsample_bynode=1,
             colsample_bytree=0.6068338278675369,
gamma=0.0017889282555567038,
             gpu_id=-1, importance_type='gain', interaction_constraints='',
             learning_rate=0.0013963222902296055, max_delta_step=0,
max_depth=7,
             min_child_weight=19, monotone_constraints='()',
             n_estimators=5400, n_jobs=0, num_parallel_tree=1,
             objective='reg:linear', random_state=1,
             reg_alpha=0.22448614695919908, reg_lambda=2.9482948529431052,
             scale_pos_weight=1, seed=1, subsample=0.8571414026048771,
             tree_method='exact', validate_parameters=1,
verbosity=None).fit(X_testing,y_train)

pred_xg1 = xg1.predict(X_testing_valid)
xg1_regr = mean_absolute_error(pred_xg1, y_valid)
print("mae",xg1_regr)
```

```
[23:55:01] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
```

```
[23:55:47] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
mae 15.716765845381973
```

Результаты

```
results = {"Model": ["ElasticNet","LinearRegression",
                     "MLPRegression","XGBoost",
                     "Lasso","Ridge","TheilSen",
                     "Betting lines","Guessing the mean",
                     "HyperoptEstimator"],
           "MAE Score": [mae_EN,mae_lin,
                         mae_regr,xg_regr,mae_ls,
                         mae_rg,mae_ts,mae_lines,
                         mae_mean,xg1_regr]}
result_df = pd.DataFrame(data=results)
result_df = result_df.sort_values(by=["MAE Score"])
print(result_df.to_string(index=False))
```

```
              Model  MAE Score
     Betting lines  13.428539
 HyperoptEstimator  15.716766
             Lasso  15.939184
        ElasticNet  15.979897
          TheilSen  16.059345
             Ridge  16.059752
  LinearRegression  16.065118
           XGBoost  16.092118
     MLPRegression  16.590032
 Guessing the mean  17.865045
```

Лучше всего работает гипероптимизированный XGBoostRegressor.

Интерпретация точности моделей:

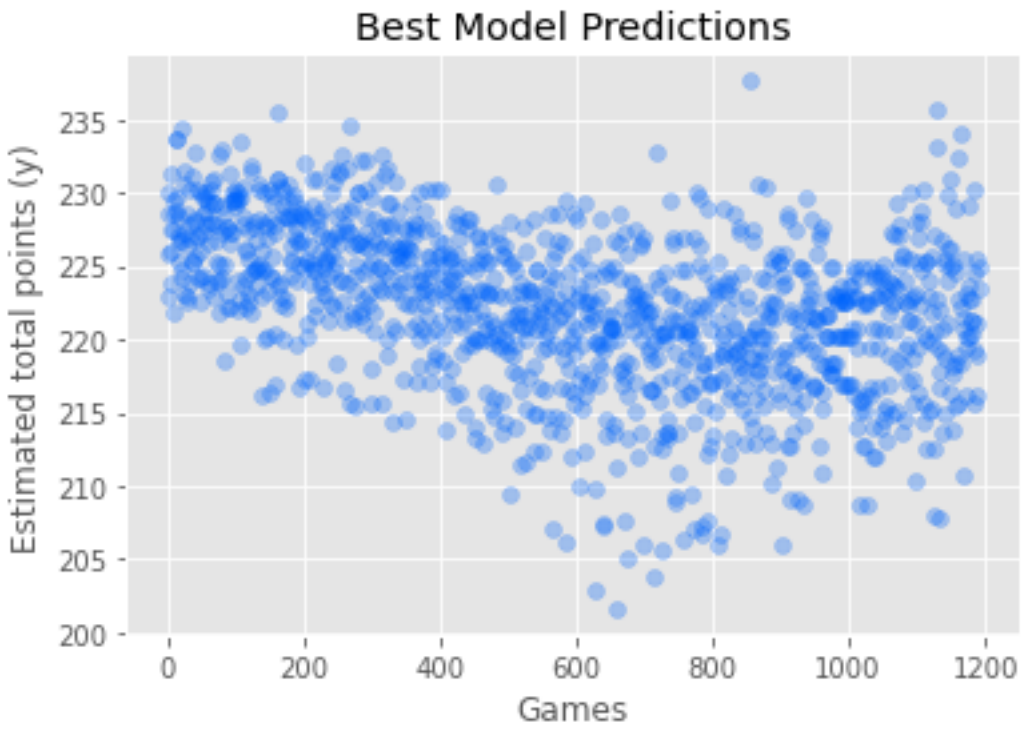
```
bttm = (1 - (xg1_regr / mae_mean))*100
wttl = (1 - (mae_lines / xg1_regr))*100
print("The model is {:.0f}% better than guessing the mean".format(bttm))
print("The model is {:.0f}% worse than the betting lines".format(wttl))
```

```
The model is 12% better than guessing the mean
The model is 15% worse than the betting lines
```

В целевой переменной так много шума, что предсказать общее количество очков, набранных в NBA, непросто. Использование прошлых результатов в качестве оценки общего количества баллов не очень хорошо работает. «Они набрали 130 очков в прошлой игре, и это снова будет много» — эти данные показывают, что это явно ошибочное утверждение. Даже лучшие модели букмекерской конторы имеют среднюю абсолютную ошибку 13,40 балла.

Visualization of model predictions:

```
import numpy as np
with plt.style.context("ggplot"):
    plt.scatter(range(len(pred_xg1)), pred_xg1, marker="o", alpha=0.3,color =
"#0066ff")
    plt.xlabel("Games")
    plt.ylabel("Estimated total points (y)")
    plt.title("Best Model Predictions")
fig6 = plt.figure();
```




```
<Figure size 432x288 with 0 Axes>
```

Объяснение того, почему прогнозы выглядят так, заключается в том, что первые ~ 400 игр относятся к сезону 2018 года, а остальные — к сезону 2019 года.

Visualization of actual outcomes:

```
with plt.style.context("ggplot"):
    plt.scatter(range(len(y_valid)), y_valid, marker="o", alpha=0.3,
color="#0066ff")
    plt.xlabel("Games")
    plt.ylabel("Total points (y)")
    plt.title("Real Outcomes")
fig7 = plt.figure();
```

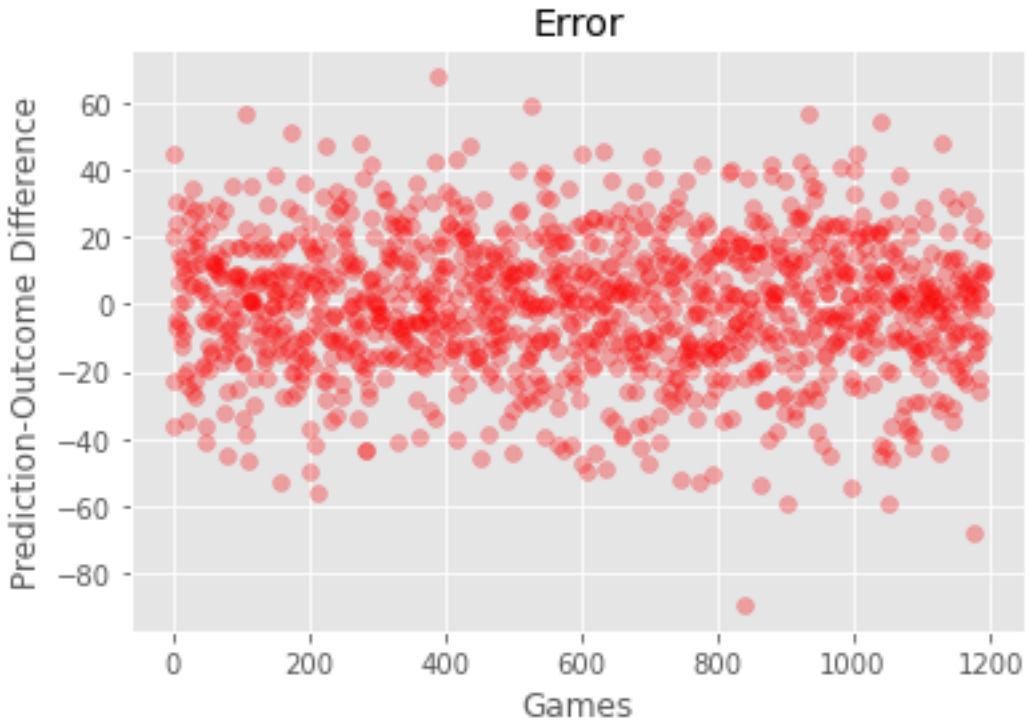

```
<Figure size 432x288 with 0 Axes>
```

Они выглядят одинаково, но масштаб по оси абсцисс намного больше для графика реальных результатов.

Чтобы увидеть, предсказывает ли наша лучшая модель более высокий или более низкий результат, чем реальный результат, мы можем вычислить среднюю ошибку, не принимая абсолютное значение:

```
ls = []
for i in range(len(pred_ts)):
    ls.append(pred_xg1[i] - y_valid.iloc[i])
print ("Average Error of Lasso:",sum(ls))
with plt.style.context("ggplot"):
    plt.scatter(range(len(ls)), ls, marker="o", alpha=0.3, color="#ff0000")
    plt.xlabel("Games")
    plt.ylabel("Prediction-Outcome Difference")
    plt.title("Error")
fig7 = plt.figure();
```

```
Average Error of Lasso: -31.4903564453125
```




```
<Figure size 432x288 with 0 Axes>
```

Положительное значение означает, что модель в среднем прогнозирует немного более высокий результат, чем реальный результат, однако точность прогнозирования все равно достаточно хорошая.

Список литературы

1. К. Элбон. Машинное обучение с использованием Python. Сборник рецептов— Санкт-Петербург, Вильямс, 2019 – 200 с.
2. Ю.Е. Гапанюк. Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных. — [Электронный ресурс] — Режим доступа. — URL: https://nbviewer.org/github/ugapanyuk/ml_course_2020/blob/master/common/notebooks/missing/handling_missing_norm.ipynb (Дата обращения: 15.12.2022).
3. А. Мюллер, С. Гвидо. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными. — Санкт-Петербург, Вильямс, 2020 – 480 с.