**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по индивидуальному домашнему заданию**
**по дисциплине “Машинное обучение”**
**Тема: Предсказание зарплаты игроков NBA на основе статистики матчей**

Студенты гр. 6307 _____ Новиков Б.М.

_____ Ходос А.А.

Преподователь _____ Жангиров Т.Р.

Санкт-Петербург
2020

## СОДЕРЖАНИЕ

## 1. Описание датасета и решаемой задачи

### 1.1. Описание задачи

Задача состоит в предсказании зарплаты игроков NBA на основе их информации (возраст, позиция) статистики матчей (количества сыгранных игр, показателей очков в среднем за игру и т.д.).

### 1.2. Данные статистики

Данные статистики были загружены с сайта баскетбольной статистики Basketball Reference [1]. Были собраны данные статистики за игру (т.к. их наиболее часто используют для сравнения в СМИ) за каждый сезон в период с 2015 по 2020.

Загруженный датасет статистики за сезон 2019/2020 представлен на рисунке 1. Датасеты за другие сезоны имеют те же атрибуты.

| Rk | Player | Pos | Age | Tm | G | GS | MP | FG | FGA | FG% | 3P | 3PA | 3P% | 2P | 2PA | 2P% | eFG% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Steven Adams… | C | 26 | OKC | 63 | 63 | 26.7 | 4.5 | 7.6 | 0.59… | 0.0 | 0.0 | 0.33… | 4.5 | 7.5 | 0.594 | 0.593 |
| 2 | Bam Adebayo\… | PF | 22 | MIA | 72 | 72 | 33.6 | 6.1 | 11.0 | 0.557 | 0.0 | 0.2 | 0.14… | 6.1 | 10.8 | 0.564 | 0.55… |
| 3 | LaMarcus Aldr… | C | 34 | SAS | 53 | 53 | 33.1 | 7.4 | 15.0 | 0.493 | 1.2 | 3.0 | 0.389 | 6.2 | 12.0 | 0.519 | 0.532 |
| 4 | Kyle Alexande… | C | 23 | MIA | 2 | 0 | 6.5 | 0.5 | 1.0 | 0.5 | 0.0 | 0.0 | nan | 0.5 | 1.0 | 0.5 | 0.5 |
| 5 | Nickeil Alexan… | SG | 21 | NOP | 47 | 1 | 12.6 | 2.1 | 5.7 | 0.368 | 1.0 | 2.8 | 0.34… | 1.1 | 2.8 | 0.391 | 0.455 |
| 6 | Grayson Allen\… | SG | 24 | MEM | 38 | 0 | 18.9 | 3.1 | 6.6 | 0.466 | 1.5 | 3.7 | 0.40… | 1.6 | 2.9 | 0.545 | 0.58 |
| 7 | Jarrett Allen\all… | C | 21 | BRK | 70 | 64 | 26.5 | 4.3 | 6.6 | 0.649 | 0.0 | 0.1 | 0.0 | 4.3 | 6.6 | 0.65… | 0.649 |
| 8 | Kadeem Allen\… | PG | 27 | NYK | 10 | 0 | 11.7 | 1.9 | 4.4 | 0.43… | 0.5 | 1.6 | 0.313 | 1.4 | 2.8 | 0.5 | 0.489 |
| 9 | Al-Farouq Ami… | PF | 29 | ORL | 18 | 2 | 21.1 | 1.4 | 4.8 | 0.29… | 0.5 | 2.0 | 0.25 | 0.9 | 2.8 | 0.32 | 0.34… |
| 10 | Justin Anderso… | SG | 26 | BRK | 10 | 1 | 10.7 | 1.0 | 3.8 | 0.263 | 0.6 | 2.9 | 0.207 | 0.4 | 0.9 | 0.444 | 0.342 |
| 11 | Kyle Anderson… | SF | 26 | MEM | 67 | 28 | 19.9 | 2.3 | 4.9 | 0.474 | 0.4 | 1.3 | 0.282 | 2.0 | 3.7 | 0.541 | 0.511 |

Рисунок 1 - Датасет статистики игроков NBA за сезон 20192020

Датасет состоит из следующих атрибутов:

- Rk индекс

- Player  имя
- Pos  позиция
- Age  возраст
- Tm  название команды
- G  количество игр
- GS  количество игр начинал
- MP  минут на площадке в среднем за игру
- FG  количество попаданий в среднем за игру
- FGA  количество бросков в среднем за игру
- FG%  процент попаданий в среднем за игру
- 3P  количество трехочковых попаданий в среднем за игру
- 3PA  количество трехочковых попыток в среднем за игру
- 3P%  процент трехочковых попаданий в среднем за игру
- 2P  количество двухочковых попаданий в среднем за игру
- 2PA  количество двухочковый попыток в среднем за игру
- 2P%  процент двухочковых попаданий в среднем за игру
- eFG%  процент эффективных бросков в среднем за игру
- FT  штрафные попадания в среднем за игру
- FTA штрафные попытки в среднем за игру
- FT%  процент штрафных попаданий в среднем за игру
- ORB  подборы в атаке в среднем за игру
- DRB  подборы в защите в среднем за игру
- TRB  всего подборов в среднем за игру
- AST  результативные передачи в среднем за игру
- STL  отборы в среднем за игру
- BLK  блокшоты в среднем за игру
- TOV  потери в среднем за игру
- PF  персональные фолы в среднем за игру
- PTS  набранные очки в среднем за игру

### 1.3. Данные контрактов

Данные актуальных контрактов игроков также были загружены с сайта Basketball Reference. Датасет представлен на рисунке 2.

| Rk | Player | Tm | 2020-21 | 2021-22 | 2022-23 | 2023-24 | 2024-25 | 2025-26 |
|---|---|---|---|---|---|---|---|---|
| 1 | Stephen Curry... | GSW | $43006362 | $45780966 | nan | nan | nan | nan |
| 2 | Russell Westb... | WAS | $41358814 | $44211146 | $47063478 | nan | nan | nan |
| 3 | Chris Paul\pau... | PHO | $41358814 | $44211146 | nan | nan | nan | nan |
| 4 | John Wall\wallj... | HOU | $41254920 | $44310840 | $47366760 | nan | nan | nan |
| 5 | James Harden... | HOU | $40824000 | $43848000 | $46872000 | nan | nan | nan |
| 6 | LeBron James... | LAL | $39219565 | nan | nan | nan | nan | nan |
| 7 | Kevin Durant\... | BRK | $39058950 | $40918900 | $42778850 | nan | nan | nan |
| 8 | Blake Griffin\g... | DET | $36595996 | $38957028 | nan | nan | nan | nan |
| 9 | Paul George\g... | LAC | $35450412 | $37895268 | nan | nan | nan | nan |
| 10 | Klay Thompso... | GSW | $35361360 | $37980720 | $40600080 | $43219440 | nan | nan |
| 11 | Mike Conley\c... | UTA | $34504132 | nan | nan | nan | nan | nan |

Рисунок 2 - Датасет актуальных контрактов игроков NBA

Датасет состоит из следующих атрибутов:

- Rk  индекс
- Player  имя
- Tm  название команды
- 2020-21  зарплата на сезон 2020-21
- 2021-22  зарплата на сезон 2021-22
- 2022-23  зарплата на сезон 2022-23
- 2023-24  зарплата на сезон 2023-24
- 2024-25  зарплата на сезон 2024-25
- 2025-26  зарплата на сезон 2025-26
- Signed Using  условие заключения контракта
- Guaranteed  гарантированная сумма выплат

### 1.4. Формирование итоговых датасетов

Используя датасеты описанные в пункта 1.2 и 1.3., можно сформировать итоговый датасет, состоящий из следующих признаков:

- позиция
- возраст
- данные статистики в среднем за игру
- зарплата на сезон 2020-21

Было подготовлено несколько датасетов, включающих статистику от одного до шести сезонов. Чтобы отличать признаки статистики, каждому признаку было добавлен суффикс, соответствующий сезона. Один из таких датасетов представлен на рисунке 3.

| | Salary | Pos | Age | G_19 | GS_19 | MP_19 | FG_19 | FGA... | FG%... | 3P_19 | 3PA_... | 3P%... | 2P_19 | 2PA_... | 2P%... | eFG... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Player | | | | | | | | | | | | | | | | |
| Stephen Curry\c... | 43006362 | PG | 31.0 | 5.0 | 5.0 | 27.8 | 6.6 | 16.4 | 0.402 | 2.4 | 9.8 | 0.245 | 4.2 | 6.6 | 0.636 | 0.47... |
| Russell Westbro... | 41358814 | PG | 31.0 | 57.0 | 57.0 | 35.9 | 10.6 | 22.5 | 0.47... | 1.0 | 3.7 | 0.258 | 9.6 | 18.7 | 0.514 | 0.493 |
| Chris Paul\paulc... | 41358814 | PG | 34.0 | 70.0 | 70.0 | 31.5 | 6.2 | 12.7 | 0.489 | 1.6 | 4.3 | 0.365 | 4.6 | 8.3 | 0.55... | 0.552 |
| James Harden\h... | 40824000 | SG | 30.0 | 68.0 | 68.0 | 36.5 | 9.9 | 22.3 | 0.444 | 4.4 | 12.4 | 0.355 | 5.5 | 9.9 | 0.556 | 0.54... |
| LeBron James\j... | 39219565 | PG | 35.0 | 67.0 | 67.0 | 34.6 | 9.6 | 19.4 | 0.493 | 2.2 | 6.3 | 0.348 | 7.4 | 13.1 | 0.564 | 0.55 |
| Blake Griffin\griff... | 36595996 | PF | 30.0 | 18.0 | 18.0 | 28.4 | 4.9 | 13.9 | 0.35... | 1.5 | 6.2 | 0.243 | 3.4 | 7.7 | 0.439 | 0.406 |
| Paul George\ge... | 35450412 | SG | 29.0 | 48.0 | 48.0 | 29.6 | 7.1 | 16.3 | 0.439 | 3.3 | 7.9 | 0.41... | 3.9 | 8.4 | 0.46... | 0.539 |
| Mike Conley\con... | 34504132 | PG | 32.0 | 47.0 | 41.0 | 29.0 | 4.9 | 12.1 | 0.409 | 2.0 | 5.4 | 0.375 | 2.9 | 6.6 | 0.43... | 0.494 |
| Jimmy Butler\bu... | 34379100 | SF | 30.0 | 58.0 | 58.0 | 33.8 | 5.9 | 13.1 | 0.455 | 0.5 | 2.1 | 0.244 | 5.4 | 11.0 | 0.495 | 0.474 |

Рисунок 3 - Итоговый датасет

Количество признаков и записей в зависимости от датасета представлены в таблице 1. Если игрок не играл в один из сезонов, то все признаки в датасете соответствующие этому сезону равны 0.

Таблица 1. Признаки и записи датасетов

| **Датасет** | **Кол-во признаков** | **Кол-во записей** |
|---|---|---|
| Один сезон | 28 | 406 |
| Два сезона | 53 | 419 |
| Три сезона | 78 | 424 |
| Четыре сезона | 103 | 428 |
| Пять сезонов | 128 | 428 |

## 2. Анализ и преобразование данных

### 2.1. Предварительная обработка признаков

В данном датасет надо обработать один признак - позицию. В баскетболе в зависимости от технического арсенала, физического состоянии и роста каждый игрок занимает четко определенную позицию (роль, амплуа) на площадке. Обычно разделяют 5 позиций, от низкорослых и быстрых до высоких и массивных: Point Guard (PG), Shooting Guard (SG), Small Forward (SF), Power Forward (PF), Center (C). Их можно привести в числовой признак (PG - 1, SG - 2, SF - 3, PF - 4, C - 5).

В нашем датасете есть игроки которые играют на нескольких позициях, поэтому их позиция записана как PG-SG, SG-SF и т.д. Таким образом правильным будет дать им промежуточное значение числового признака (PG-SG - 1.5, SG-SF - 2.5 и т.д.)

### 2.2. Анализ корреляции

Для каждого признака из статистики был проведен анализ на корреляцию с зарплатой. График наиболее коррелирующих признаков представлен на рисунке 4.

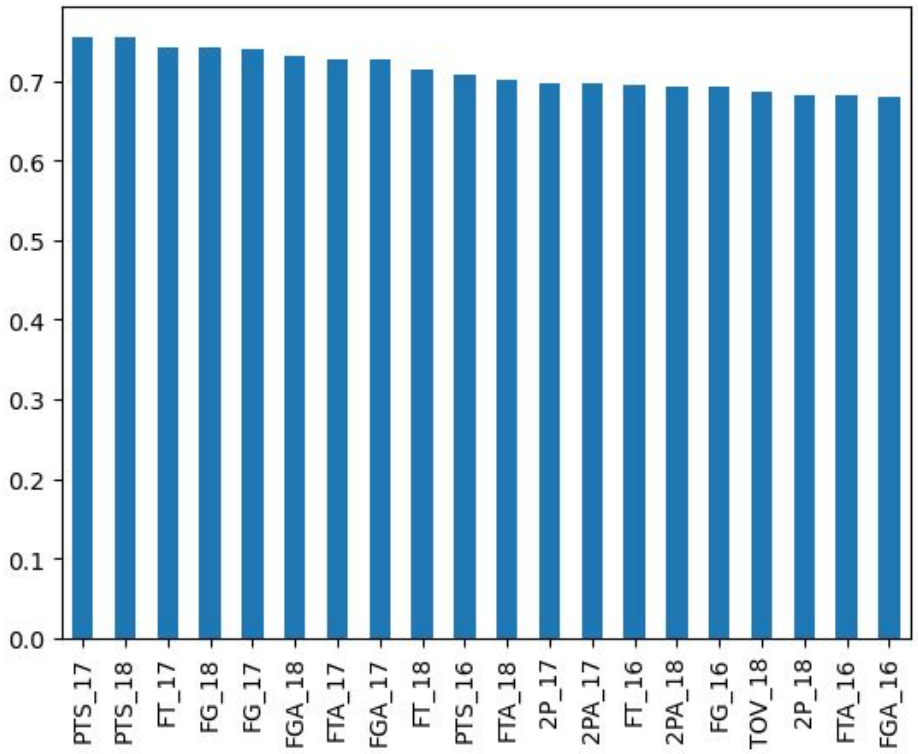


Рисунок 4 - график наиболее коррелирующих признаков статистики с зарплатой

Также каждый признак был визуализирован. Пример визуализации представлен на рисунке 5.

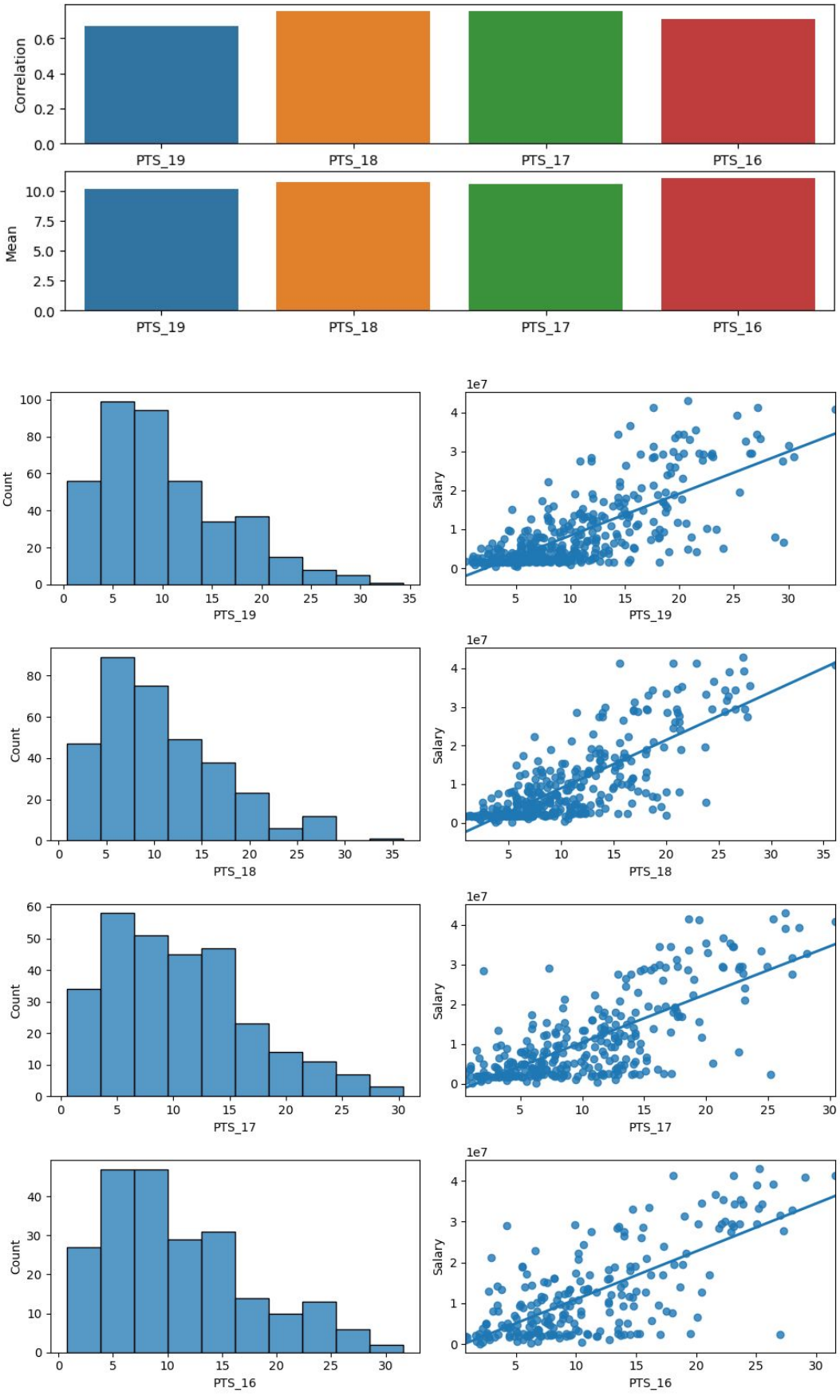


Рисунок 5 - визуализации признака статистики PTS

Карта корреляции для признаков, которые сильно коррелируют с зарплатой представлена на рисунке 6.

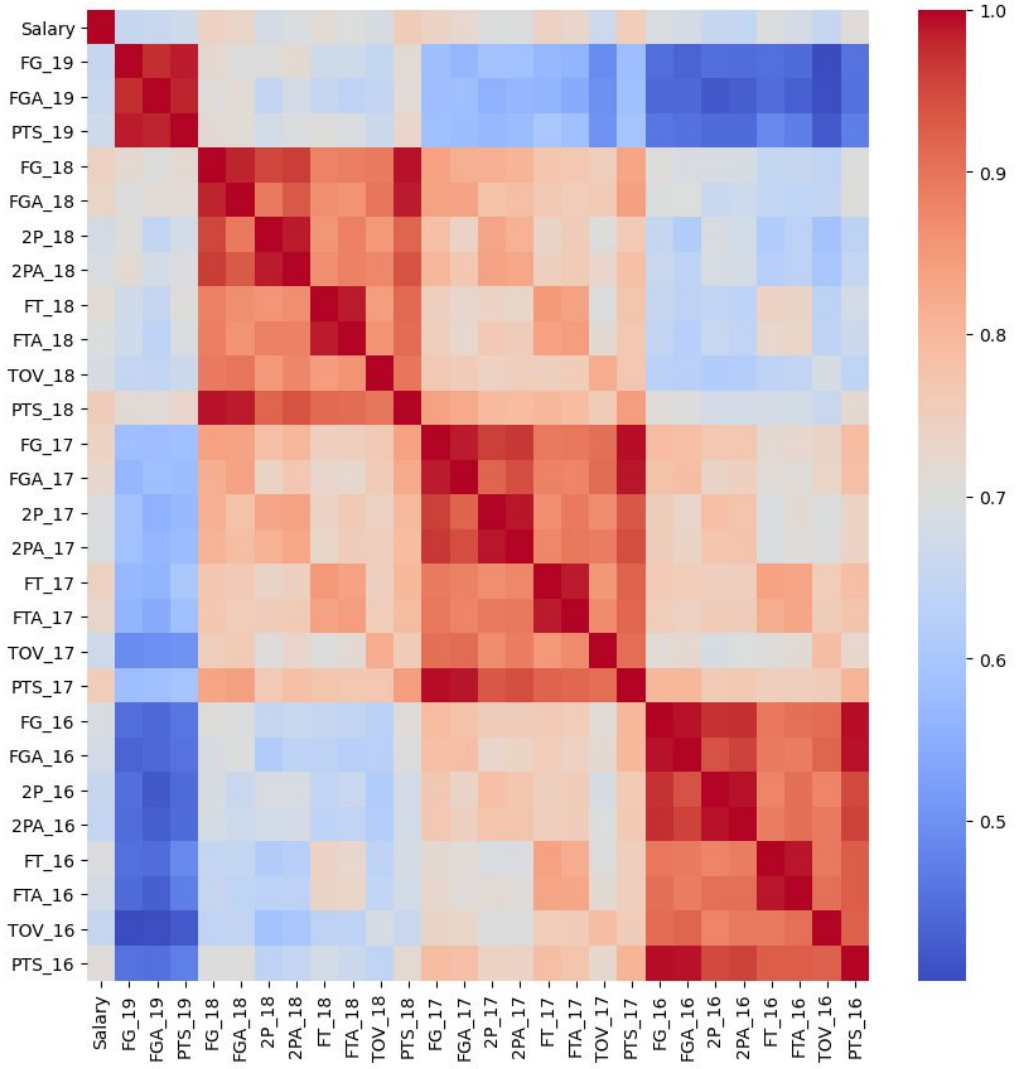


Рисунок 6 - карта корреляции признаков наиболее коррелирующих с зарплатой

Видно, что некоторые признаки сильно коррелируют между собой (особенно признаки статистики за определенный сезон). Признаки, которые сильно коррелируют друг с другом, называются коллинеарными. Удаление одной переменной в таких парах признаков часто помогает модели обобщать и быть более интерпретируемой. В связи с этим было

решено сохранить набор данных, состоящий только из признаков слабо коррелирующих между собой. Такой датасет состоит из 36 признаков. Карта корреляции нового набора представлена на рисунке 7.

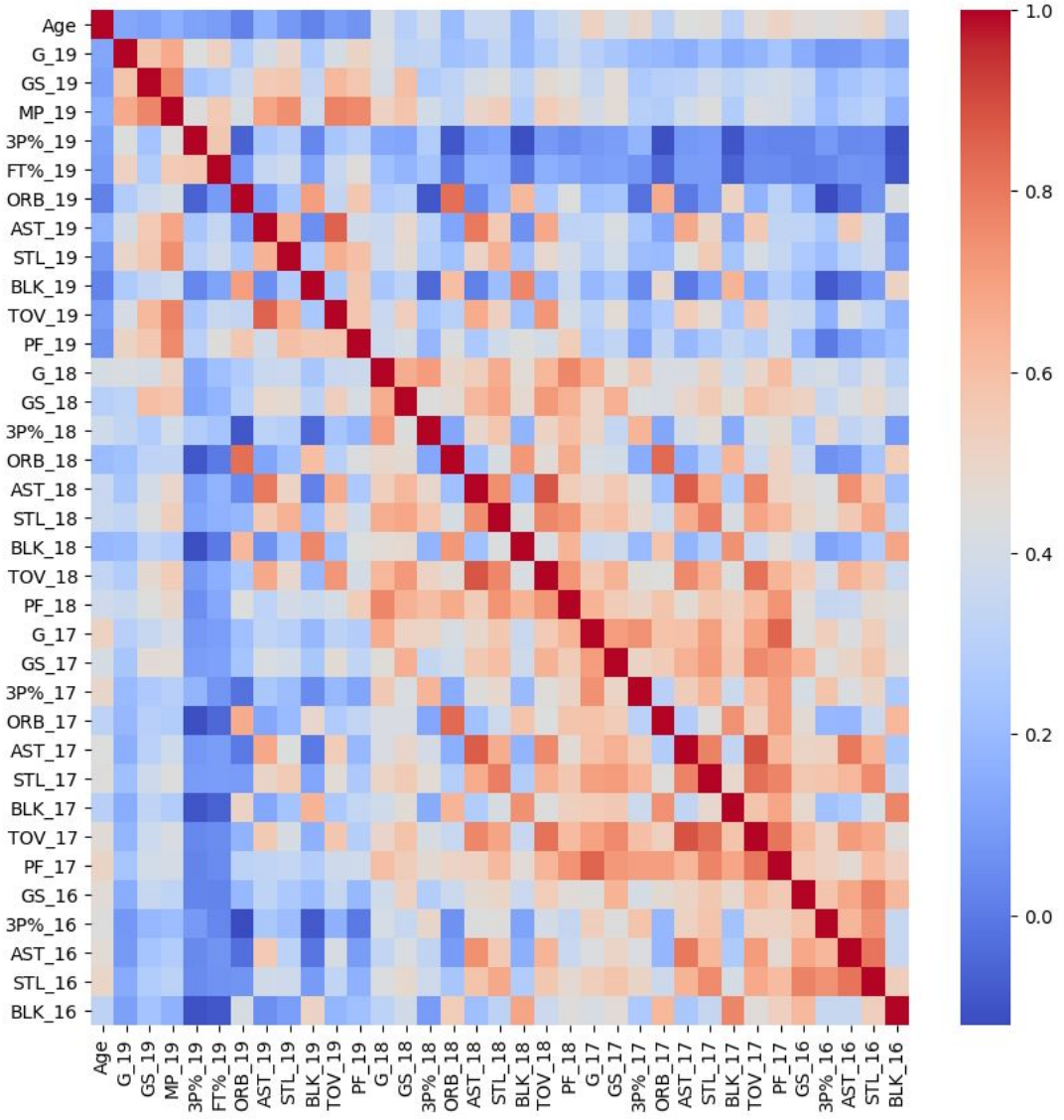


Рисунок 7 - карта корреляции нового набора данных

### 2.3. Факторный анализ

Также для повышения интерпретируемости, был проведен факторный анализ - данные были разделены на 39 факторов. Один из

факторов представлен на рисунке 8, его можно интерпретировать как “регулярно начинал игру 4 последних сезона”.

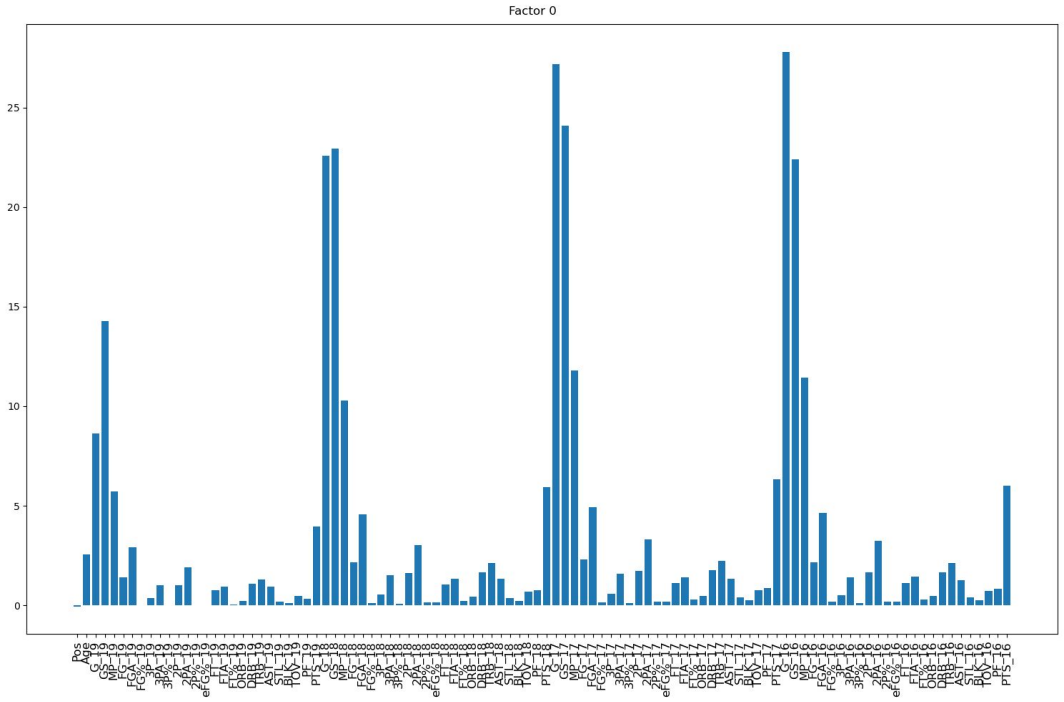


Рисунок 8 - фактор “регулярно начинал игру 4 последних сезона”

### 2.4. Анализ на нормировку

В случае множественной регрессии рекомендуют преобразовывать переменные. Все датасеты были преобразованы и разделены на следующие группы:

- без нормировки
- нормировка с помощью StandardScaler
- нормировка с помощью RobustScaler

### 2.5. Конструирование и выбор признаков

В работе различными способами для улучшения работы были предприняты попытки отобрать наиболее важные признаки и сконструировать новые. Данный процесс требует много времени и глубоких знаний в области.

## 3. Регрессионный анализ

### 3.1 Выбор моделей регрессии

Рассмотренные модели регрессий представлены в таблице 1.

Таблица 1 - Рассматриваемые модели регрессий

| | | |
|---|---|---|
| Простые регрессоры | Линейные регрессоры | LinearRegression |
| | | Ridge |
| | | RidgeCV |
| | | SGDRegressor |
| | Регрессоры на основе метода ближайших соседей | KneighborsRegressor |
| | Регрессоры на основе деревьев решений | DecisionTreeRegressor |

| | | |
|---|---|---|
| Регрессоры на основе ансамблирования | Бустинг | AdaBoostRegressor |
| | | GradientBoostingRegressor |
| | | HistGradientBoostingRegressor |
| | Бэггинг | BaggingRegressor |
| | | RandomForestRegressor |
| | Стэкинг | StackingRegressor |

Широкая выборка моделей регрессий обусловлена попыткой понять, какие именно типы и по какой причине показали себя эффективными. К примеру, выбор Ridge и RidgeCV обусловлен попыткой оценить значимость кросс-валидации при обучении, выбор таких методов ансамблирования как беггинг, бустинг и стекинг обусловлен попыткой понять, какой тип ансамблирования лучше подходит для данной задачи.

Следует также отметить, что каждая из выбранных моделей регрессий изначально оценивалась отдельно для подбора оптимальных параметров. К примеру, выяснилось, что для AdaBoostRegressor оптимальными параметрами являются learning_rate=2, n_estimators=100, что означает повышение количества оценщиков в два раза, а также уменьшение их вклада в 2 раза от изначальных значений.

### 3.1.1. Классические линейные регрессоры

**LinearRegression**

Регрессор на основе метода восстановления зависимости одной (объясняемой, зависимой) переменной y от другой или нескольких других переменных (факторов, регрессоров, независимых переменных) x с линейной функцией зависимости.

**Ridge**

Регрессор на основе метод регуляризации для некорректно поставленных задач. Используется на практике для решения проблемы мультиколлинеарности в линейной регрессии, которая часто возникает в моделях с большим количеством параметров.

Мультиколлинеарность — присутствие окололинейных отношений между независимыми переменными.

**RidgeCV**

Ridge со встроенной кросс-валидацией.

Кросс-валидация - процедура эмпирического оценивания обобщающей способности алгоритмов, обучаемых по прецедентам

**SGDRegressor**

Регрессор стохастического градиентного спуска (SGD) процедуру обучения, используя SGD, поддерживающую различные функции потерь и штрафы для соответствия моделям линейной регрессии.

SGD - это простой, но эффективный алгоритм оптимизации, используемый для поиска значений параметров / коэффициентов функций, которые минимизируют функцию стоимости.

### 3.1.2. Регрессоры на основе метода ближайших соседей

**KneighborsRegressor**

Регрессор на основе метода KNN. Целевой параметр предсказывается благодаря локальной интерполяции ассоциирующихся с целью соседей в тренировочном сете.

KNN — простой алгоритм, который хранит все наблюдения и предсказывает численный целевой показатель на основе меры схожести (к примеру, функции расстояния)

### 3.1.3. Регрессоры на основе деревьев решений

**DecisionTreeRegressor**

Регрессор на основе деревьев решений. Дерево решений - это инструмент для принятия решений, который использует древовидную структуру в виде блок-схемы или представляет собой модель решений и всех их возможных результатов, включая исходы, затраты на ввод и полезность.

Считается слабой моделью и широко используется в методах ансамблирования.

### 3.1.4 Регрессоры на основе ансамблирования

Идея данного типа регрессоров состоит в том, чтобы обучить несколько моделей, каждая из которых предназначена для прогнозирования или классификации набора результатов. Используя методы ансамбля, мы можем повысить стабильность окончательной модели и уменьшить ошибки. Комбинируя множество моделей, мы можем (в основном) уменьшить дисперсию, даже если они по отдельности невелики, поскольку мы не будем страдать от случайных ошибок из одного источника.

Главный принцип ансамблевого моделирования - объединить слабых учеников в одну группу с сильным учеником.

Выделяют три типа ансамблирования: бустинг, беггинг, стекинг:

| | **Беггинг** | **Бустинг** | **Стекинг** |
|---|---|---|---|
| Разбиение данных на подмножества | Случайное | Дает неправильно классифицированным результатами наибольшее предпочтение | Разнообразное |
| Цель достичь | Минимизация дисперсии | Увеличение качества предсказания | Обе |
| Методы, где это используется | Случайные подпространства | Градиентный спуск | Смешивание |
| Функции для комбинирования одиночных моделей | (Взвешенные) средние | Взвешенное голосование большинства | Логистическая регрессия |

Таблица - сводное описание видов ансамблирования

**Бустинг**

Бустинг (англ. boosting — улучшение) — это процедура последовательного построения композиции алгоритмов машинного обучения, когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов. Бустинг представляет собой жадный алгоритм построения композиции алгоритмов. Изначально понятие бустинга возникло в работах по вероятно почти корректному обучению в связи с вопросом: возможно ли, имея множество плохих (незначительно отличающихся от случайных) алгоритмов обучения, получить хороший.

***AdaBoostRegressor***

Алгоритм усиливает классификаторы, объединяя их в «комитет». AdaBoost является адаптивным в том смысле, что каждый следующий

комитет классификаторов строится по объектам, неверно классифицированным предыдущими комитетами. AdaBoost чувствителен к шуму в данных и выбросам. Однако он менее подвержен переобучению по сравнению с другими алгоритмами машинного обучения.

Регрессор AdaBoost - это метаоценщик, который начинает с подгонки регрессора к исходному набору данных, а затем подгоняет дополнительные копии регрессора к тому же набору данных, но где веса экземпляров корректируются в соответствии с ошибкой текущего прогноза. Таким образом, последующие регрессоры больше сосредотачиваются на сложных случаях.

***GradientBoostingRegressor***

Как Gradient Boost, так и AdaBoost работают с деревьями решений, однако деревья в Gradient Boost больше, чем деревья в AdaBoost. Как Gradient boost, так и AdaBoost масштабируют деревья решений, однако Gradient Boost масштабирует все деревья на одинаковую величину, в отличие от AdaBoost.

***HistGradientBoostingRegressor***

Этот оценщик намного быстрее, чем GradientBoostingRegressor для больших наборов данных (n_samples> = 10 000).

**Беггинг**

Бутстрэп-агрегирование или бэггинг, это метаалгоритм композиционного обучения машин, предназначенный для улучшения стабильности и точности алгоритмов машинного обучения, используемых в статистической классификации и регрессии. Алгоритм также уменьшает дисперсию и помогает избежать переобучения. Хотя он обычно

применяется к методам обучения машин на основе деревьев решений, его можно использовать с любым видом метода. Бэггинг является частным видом усреднения модели.

***BaggingRegressor***

Данный регрессор это метаоценщик, которая подбирает базовые регрессоры для каждого из случайных подмножеств исходного набора данных, а затем агрегирует их индивидуальные прогнозы (либо путем голосования, либо путем усреднения) для формирования окончательного прогноза. Такая метаоценка обычно может использоваться как способ уменьшить дисперсию оценщика черного ящика (например, дерева решений) путем введения рандомизации в его процедуру построения и последующего создания ансамбля из него.

***RandomForestRegressor***

Основная идея, лежащая в основе этого, состоит в том, чтобы объединить несколько деревьев решений для определения окончательного результата, а не полагаться на отдельные деревья решений.

Таким образом, случайный лес — бэггинг над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков.

Из-за случайного выбора функций деревья более независимы друг от друга по сравнению с беггингом, что часто приводит к лучшему качеству прогнозирования (из-за лучшего компромисса смещения дисперсии), и я бы сказал, что это также быстрее, чем беггинг , потому что каждое дерево учится только на подмножестве функций.

**Стекинг**

Стекинг - это метод смешивания оценок. В этой стратегии некоторые оценщики индивидуально подбираются к некоторым обучающим данным, в то время как окончательная оценщик обучается с использованием суммированных прогнозов этих базовых оценщиков.

***StackingRegressor***

Регрессия на основе метода стекинга. Стекинг в этом случае предоставляет альтернативу ручному выбору лучшей модели для работы с данными путем объединения результатов нескольких учащихся без необходимости специально выбирать модель. Производительность суммирования обычно близка к лучшей модели, а иногда может превосходить производительность прогнозирования каждой отдельной модели.

### 3.2 Выбор данных для использования в задаче регрессии

В качестве данных для обучения модели были рассмотрены данные о статистике игроков за разное количество сезонов. К примеру, за один сезон (2019 год) , за два (2018-2019) или за четыре сезона (2016-2019) и т.д. Данный подход позволил неявным образом моделям найти связи между целевым параметром и результативностью игрока на протяжении некоторого времени.

Оценка эффективности работы лучших моделей регрессий моделей по показателю R2 для одного сезона стабильно показывала результат 60-62%, для двух — 63-65%, для четырех — 71-73% и далее без увеличения. Из полученных результатов следует, что информация о статистике игрока за 4 последних года является основанием для его зарплаты. По этой причине дальнейший анализ основывается на статистике игроков за 4 сезона.

### 3.3 Результаты регрессионного тестирования

Исследование эффективности модели регрессии проводилось при оценке трех параметров:

1. Коэффициент детерминации (R2 Score) - это доля дисперсии зависимой переменной, объясняемая рассматриваемой моделью зависимости, то есть объясняющими переменными.
2. Максимальная разница между результатом предсказания и предсказываемым результатом (MaxError)
3. Модуль средней ошибки предсказания (MeanAbsError )

В результате комплексного исследования эффективности моделей регрессий были получены следующие результаты, сведенные в таблицу:

Таблица 2 - Результат исследования моделей регрессий

| | **Name** | **Regressor** | **R2** | **MeanAbsError** | **MaxError** |
|---|---|---|---|---|---|
| **0** | LinearRegression | LinearRegression() | 0.625102 | 0.430778 | 1.591711 |
| **1** | Ridge | Ridge() | 0.667382 | 0.399255 | 1.550742 |
| **2** | RidgeCV | RidgeCV(alphas=array([ 0.1, 1. , 10. ])) | 0.685149 | 0.390121 | 1.531716 |
| **3** | DecisionTreeRegressor | DecisionTreeRegressor(criterion='mae', max_dep... | 0.592706 | 0.388818 | 1.864711 |
| **4** | KNeighborsRegressor | KNeighborsRegressor(n_neighbors=10, weights='d... | 0.713452 | 0.352634 | 1.704897 |
| **5** | SGDRegressor | SGDRegressor() | 0.669060 | 0.412403 | 1.540591 |
| **6** | AdaBoostRegressor + LinearRegression | (LinearRegression(), LinearRegression(), Linea... | 0.634323 | 0.437458 | 1.630954 |
| **7** | AdaBoostRegressor + Ridge | (Ridge(random_state=209652396), Ridge(random_s... | 0.626733 | 0.442976 | 1.608372 |
| **8** | AdaBoostRegressor + RidgeCV | (RidgeCV(alphas=array([ 0.1, 1. , 10. ])), Ri... | 0.559466 | 0.483079 | 1.690171 |
| **9** | AdaBoostRegressor + DecisionTreeRegressor | (DecisionTreeRegressor(criterion='mae', max_de... | 0.736578 | 0.341355 | 1.526695 |
| **10** | AdaBoostRegressor + KNeighborsRegressor | (KNeighborsRegressor(n_neighbors=10, weights='... | 0.689469 | 0.412175 | 1.646894 |

| | | | | | |
|---|---|---|---|---|---|
| **11** | AdaBoostRegressor + SGDRegressor | (SGDRegressor(random_state=209652396), SGDRegr... | 0.560441 | 0.482508 | 1.748977 |
| **12** | BaggingRegressor + LinearRegression | (LinearRegression(), LinearRegression(), Linea... | 0.640379 | 0.421301 | 1.577332 |
| **13** | BaggingRegressor + Ridge | (Ridge(random_state=2087557356), Ridge(random_... | 0.682426 | 0.388782 | 1.527266 |
| **14** | BaggingRegressor + RidgeCV | (RidgeCV(alphas=array([ 0.1, 1. , 10. ])), Ri... | 0.692531 | 0.385374 | 1.510708 |
| **15** | BaggingRegressor + DecisionTreeRegressor | (DecisionTreeRegressor(criterion='mae', max_de... | 0.708075 | 0.345247 | 1.573110 |
| **16** | BaggingRegressor + KNeighborsRegressor | (KNeighborsRegressor(n_neighbors=10, weights='... | 0.718262 | 0.348250 | 1.716198 |
| **17** | BaggingRegressor + SGDRegressor | (SGDRegressor(random_state=2087557356), SGDReg... | 0.682783 | 0.392401 | 1.546143 |
| **18** | GradientBoostingRegressor | ([DecisionTreeRegressor(criterion='friedman_ms... | 0.696705 | 0.355754 | 1.607489 |
| **19** | HistGradientBoostingRegressor | HistGradientBoostingRegressor(max_iter=400, ra... | 0.678432 | 0.361552 | 1.737918 |
| **20** | RandomForestRegressor | (DecisionTreeRegressor(max_features='auto', ra... | 0.726785 | 0.337664 | 1.521397 |
| **21** | Stacking + best | StackingRegressor(estimators=[('GradientBoosti... | 0.658076 | 0.371119 | 1.863818 |

По полученным данным были построены графики оценочных параметров для каждого регрессора.

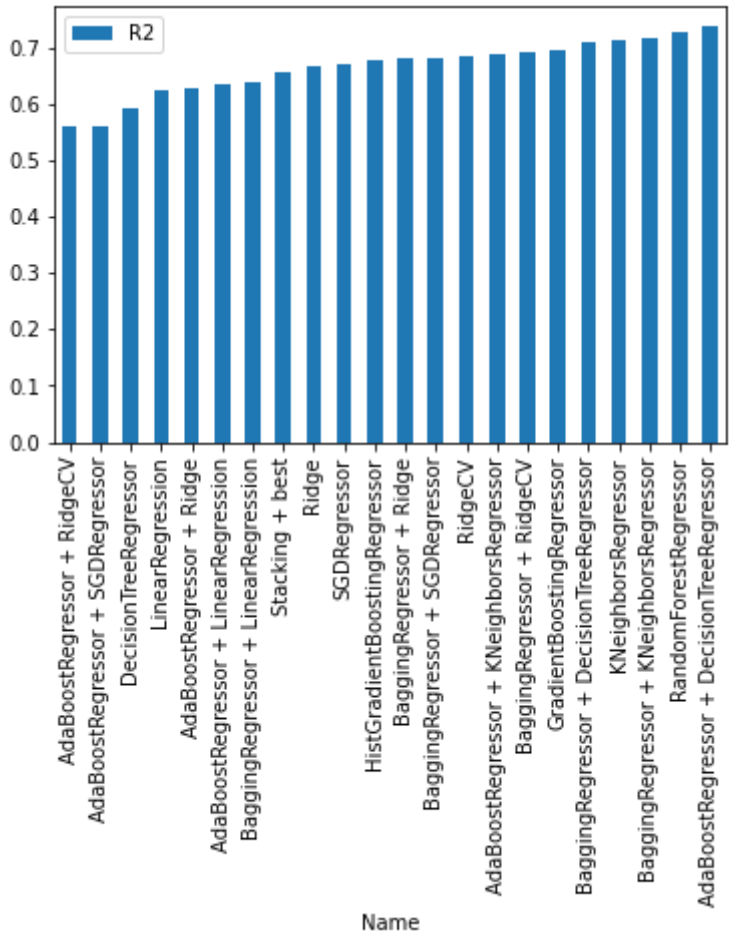


Рисунок 9 - График R2 Score

Таблица 3 - Список лучших и худших регрессоров по параметру R2

| **Худшие** | **Лучшие** |
|---|---|
| 1. AdaBoostRegressor + RidgeCV<br>2. AdaBoostRegressor + SGDRegressor<br>3. DecisionTreeRegressor | 1. AdaBoostRegressor + DecisionTreeRegressor<br>2. RandomForestRegressor<br>3. BaggingRegressor + KNeighborsRegressor |

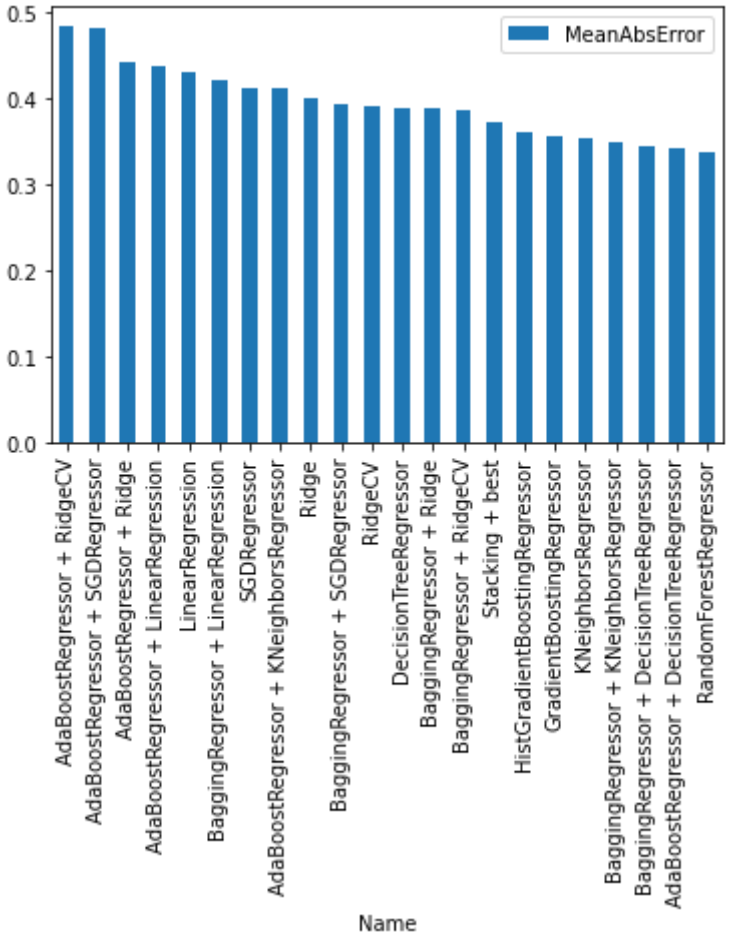


Рисунок 10 - График MeanAbsError

Таблица 4 - Список лучших и худших регрессоров по параметру MeanAbsError

| **Худшие** | **Лучшие** |
|---|---|
| 1. AdaBoostRegressor + RidgeCV<br>2. AdaBoostRegressor + SGDRegressor<br>3. AdaBoostRegressor + Ridge | 1. RandomForestRegressor<br>2. AdaBoostRegressor + DecisionTreeRegressor<br>3. BaggingRegressor + DecisionTreeRegressor |

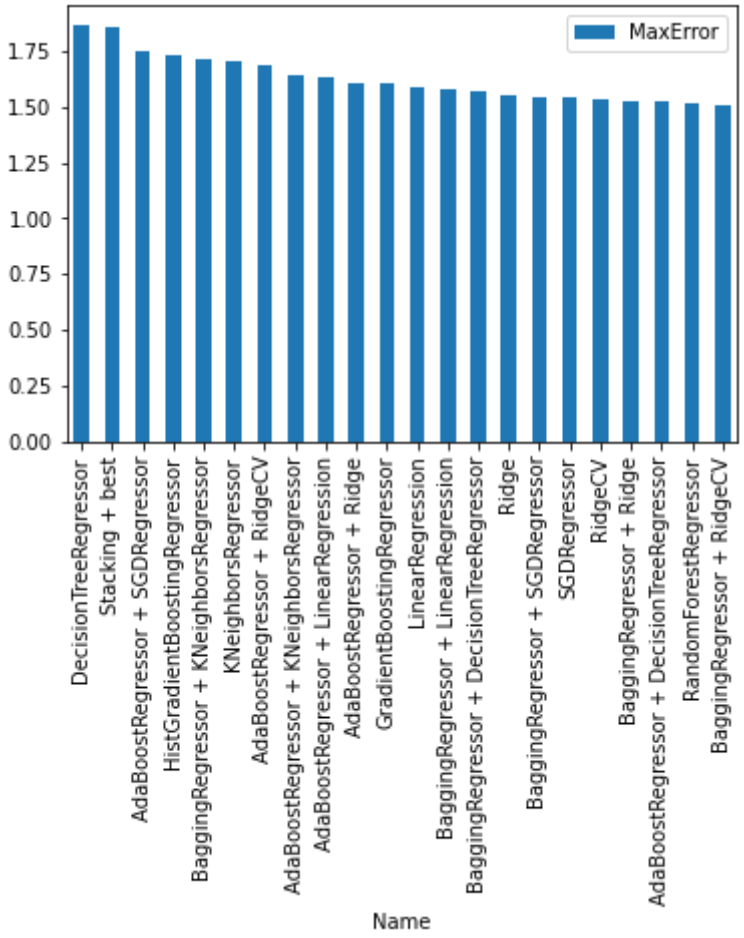


Рисунок 11 - График MaxError

Таблица 5 - Список лучших и худших регрессоров по параметру MaxError

| **Худшие** | **Лучшие** |
|---|---|
| 1. DecisionTreeRegressor<br>2. Stacking + best<br>3. AdaBoostRegressor + SGDRegressor | 1. BaggingRegressor + RidgeCV<br>2. RandomForestRegressor<br>3. AdaBoostRegressor + DecisionTreeRegressor |

По данным графикам видно, что лидирующие позиции занимают AdaBoostRegressor + DecisionTreeRegressor и RandomForestRegressor, которые представляют собой методы бустинга и беггинга, соответственно. По показателю R2 лидирует бустинг, а по модулю средней ошибки — беггинг, что сходится с целями использования данных методов: бустинг — увеличение эффективности предсказания, беггинг — минимизация дисперсии.

Также стоит отметить, что худшие показатели имеют модели бустинга с использованием в виде оценщика не слабых регрессий, что также сходится с идеей методов ансамблирования, а именно применения в виде оценщиков слабых моделей. Данная зависимость объясняется тем, что при использовании сильных учеников в методах бустинга, множество рассматриваемых гипотез сильно уменьшается из-за отсутствия достаточного количества ошибок, что приводит к невозможности для бустера изменить основную гипотезу и в результате модель, скорее всего, окажется близка по результату к использованию этого ученика без бустера.

### 3.4 Итоговая модель

В ходе исследования результатов работы моделей на наших данных были получены ответы на следующие вопросы:

Таблица 6 - Выводы по разделу

| Вопрос | Ответ |
|---|---|
| Лучшая модель регрессии | AdaBoostRegressor + DecisionTreeRegressor |
| Лучшая линейная регрессия | RidgeCV |
| Лучший метод ансамблирования | бустинг |

| | |
|---|---|
| Лучший беггинг | RandomForestRegressor |
| Лучший бустинг | AdaBoostRegressor |
| Лучший простой регрессор | RidgeCV |
| Лучшая связка с AdaBoostRegressor | DecisionTreeRegressor |
| Лучшая связка с BaggingRegressor | KNeighborsRegressor |
| Без или с кросс-валидацией (на примере Ridge) | с использованием кросс-валидации |
| LinearRegression против SGDRegressor | SGDRegressor |
| Деревья в AdaBoostRegressor или в GradientBoostingRegressor | AdaBoostRegressor |
| Бустинг или стекинг | бустинг |
| Ансамблирование со слабым или с сильным регрессором | со слабыми |

Таким образом, конечное предпочтение отдается методу AdaBoostRegressor + DecisionTreeRegressor. Однако, немаловажным будет также отметить результаты RidgeCV, который несмотря на относительную простоту в сравнении с методами ансамблирования, показывает очень хорошие результаты в скорости и в качестве предсказания.

# 4. Результаты

## 4.1. Проблемы

### 4.1.1. Несколько команд за сезон

По правилам NBA разрешено проводить обмены игроков в течении сезона, таким образом некоторые игроки имеют несколько строчек статистики за каждую из команд, в которой он играл. Примером такого игрока является Тревор Ариза.

| Rk | Player | Pos | Age | Tm | G | GS | MP | FG | FGA | FG% | 3P | 3PA | 3P% | 2P | 2PA | 2P% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | Trevor Ariza\arizatr01 | SF | 34 | TOT | 53 | 21 | 28.2 | 2.7 | 6.1 | 0.43... | 1.5 | 3.9 | 0.37... | 1.2 | 2.2 | 0.556 |
| 19 | Trevor Ariza\arizatr01 | SF | 34 | SAC | 32 | 0 | 24.7 | 2.0 | 5.2 | 0.38... | 1.3 | 3.8 | 0.35... | 0.7 | 1.3 | 0.488 |
| 19 | Trevor Ariza\arizatr01 | SF | 34 | POR | 21 | 21 | 33.4 | 3.7 | 7.6 | 0.491 | 1.6 | 4.0 | 0.4 | 2.1 | 3.5 | 0.595 |

Рисунок 12 - пример игрока, сыгравшего за несколько команд в течении сезона

Проблема состоит в том, что вместо одного игрока, который провел целый сезон, в анализе будет участвовать несколько игроков, которые провели некоторую часть сезона. Решением этой проблемы стало использование общих данных за весь сезон без статистики игр за определенные команды.

На примере игрока Тревора Аризы видно, что ему отводилась разная роль в командах, за которые он играл. Таким образом, теряется часть информации за сезон, которая могла бы повлиять на предсказание зарплаты игроков.

### 4.1.2. Не сыгравшие ни одной игры

Контракт на сезон 2020-21 могут иметь игроки, не игравшие еще игр в NBA или травмированные игроки, которые пропустили несколько сезонов подряд. Примером травмированного игрока, который не сыграл ни одной игры в сезоне 2019-20 и при этом получающий одну из самых больших зарплат в лиге, является Джон Уолл.

| | Salary | Pos | Age | G_19 | GS_19 | MP_19 | FG_19 | FGA_19 | FG%_19 |
|---|---|---|---|---|---|---|---|---|---|
| Player | | | | | | | | | |
| John Wall\walljo01 | 41254920 | PG | 29.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

Рисунок 13 - пример игрока пропустившего сезон

Игроки, не сыгравшие ни одной игры, были удалены из датасета, таким образом они не влияют на предсказание зарплаты.

### 4.1.3. Имеющие несколько контрактов

Как оказалось, ряд игроков имеют несколько заключенных контрактов с разными команда в связи с финансовыми правилами NBA, обязывающими команду выплатить всю зарплату игроку при отказе от него. Количество мест в команде и потолок зарплат ограничен, поэтому иногда от игроков просто отказываются вместо совершение обмена с его участием.

Если с игроком был заключен контракт на несколько сезонов, новый контракт по выплате будет заключен на сумму равной сумме зарплат за все сезоны. Примером такого игрока является Николя Батюм, который имел контракт на 3 года с зарплатой 9 млн, а после отказа от него подписан новый контракт на 27 млн. При этом реальная его зарплата на сезон 2020-21 равна 2.5 млн.

| Rk | Player | Tm | 2020-21 | 2021-22 | 2022-23 | 2023-24 | 2024-25 | 2025-26 |
|---|---|---|---|---|---|---|---|---|
| 41 | Nicolas Batum… | CHO | $27130434 | nan | nan | nan | nan | nan |
| 301 | Nicolas Batum… | LAC | $2564753 | nan | nan | nan | nan | nan |

Рисунок 14 - пример игрока, имеющего несколько контрактов

Таким образом в итоговый датасет вместо одного игрока попадало два с разными зарплатами, одна из которых может совсем не соответствовать действительности, что негативно сказывалось на результатах регрессии.

Для каждого из пяти таких игроков были определены, какие из контрактов отражают его действительную зарплату, а какие долг предыдущей команды. Контракты с долгом команды удалялись вручную из датасета контрактов.

### 4.1.4. Отсутствие данных о ментальности, лояльности игрока и др.

При изучении области был сделан вывод о неполноте информации, которая могла бы описывать уровень зарплаты игрока. На зарплату игрока в NBA могут влиять многие факторы, которые не описываются статистикой. Это может быть ментальность, лояльность, опытность игрока.

Примером игрока для которого сильно не хватает информации является Демаркус Казинс. Набирая в среднем очень высокую статистику (25.2 очка, 12.9 подборов и 5.4 передачи за 36.2 минут) в сезоне 2017-18 игрок получает предложение лишь на 5 млн на сезон 2018-19, 3.5 млн на сезон 2019-20 и 2.3 млн на сезон 2020-21. В это же время Энтони Дэвис, показывающий похожую статистику, получает 32 млн за сезон.

Казинс является конфликтным, недисциплинированным игроком, в которого не каждый захочет вкладывать деньги.

### 4.2. Визуализация результатов

#### 4.2.1. Зависимость ошибок от размера обучающих данных

Графики зависимости оценочных параметров от размера обучающих данных представлены на рисунках 15-17. Лучшие результаты модель дает при соотношение тестовые данные к обучающим 1:4.

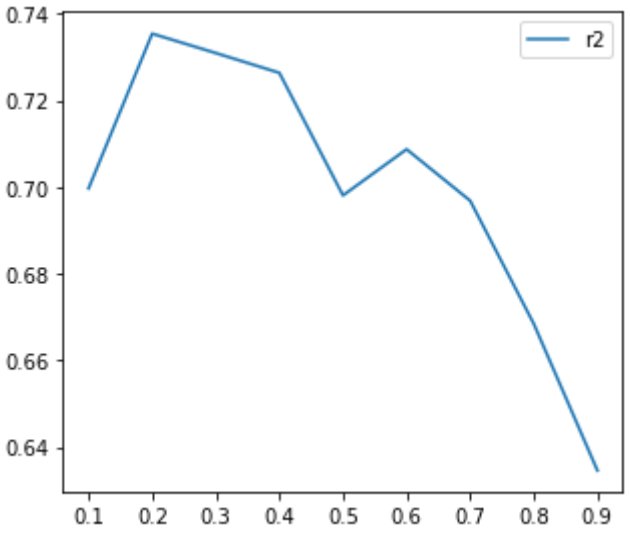


Рисунок 15 - Зависимость R2 от размера тестовых данных

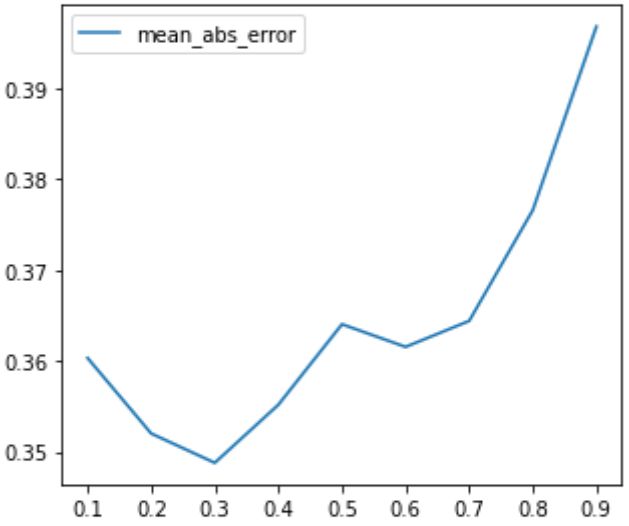


Рисунок 16 - Зависимость MeanAbsError от размера тестовых данных

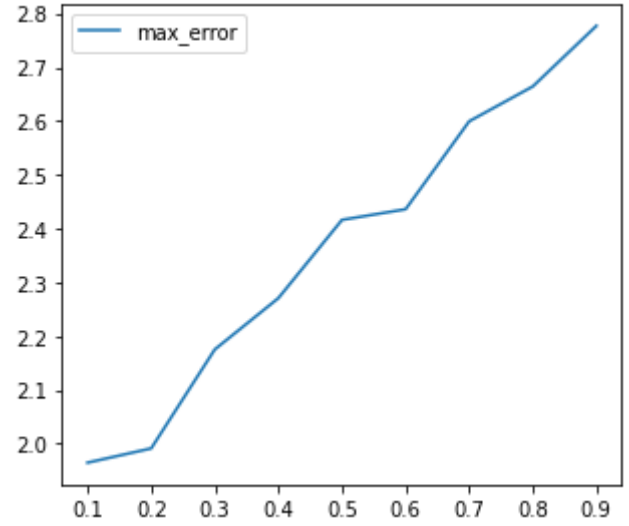


Рисунок 17 - Зависимость MaxError от размера тестовых данных

### 4.2.2. Сравнение реальных и предсказанных зарплат

Гистограммы сравнения реальных и предсказанных зарплат представлены на рисунке 18. Как видно, модель для большинства игроков дает хороший результат предсказания зарплаты.

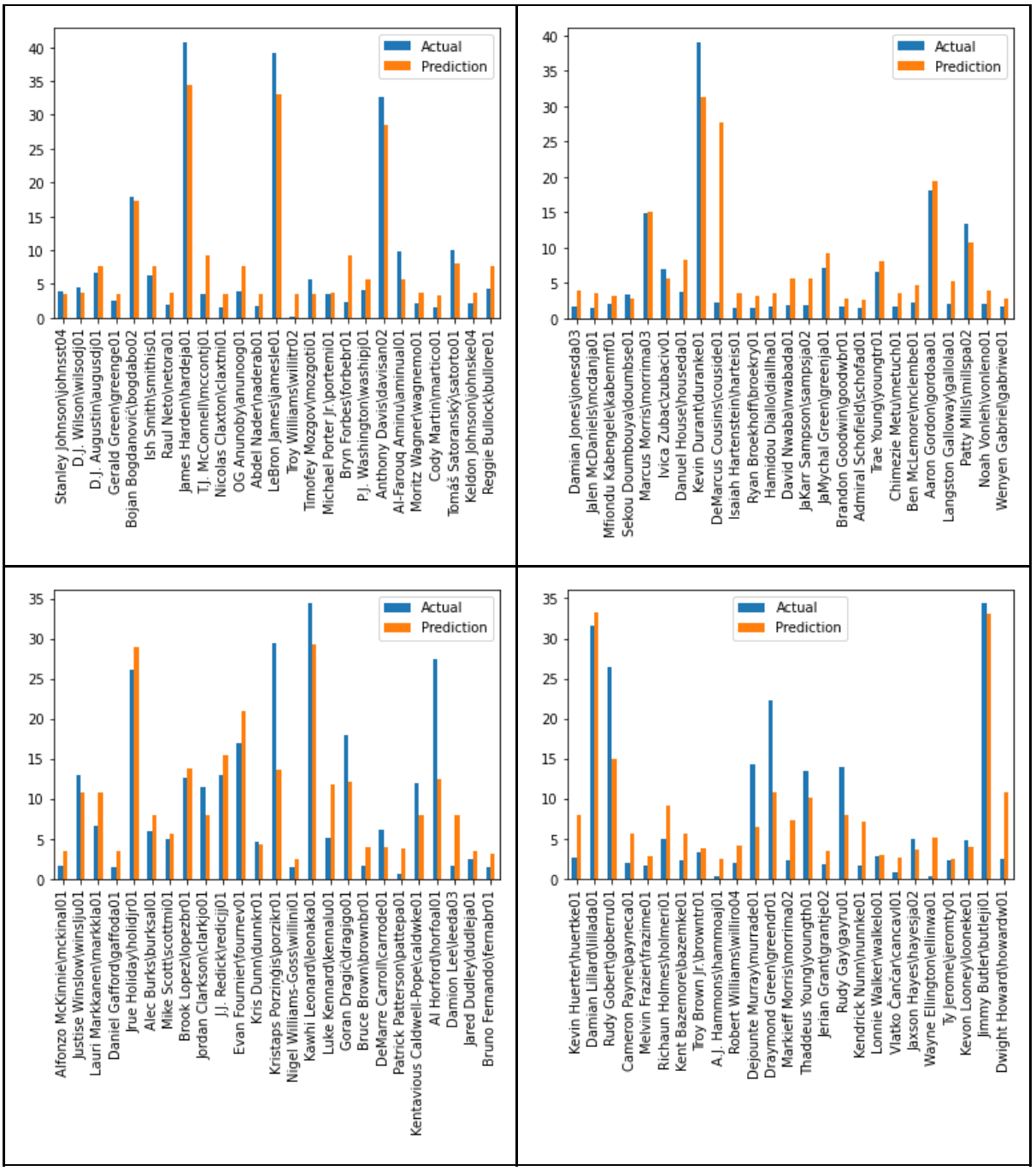


Рисунок 18 - Гистограммы сравнения реальных и предсказанных зарплат игроков

## 5. Вывод

В данной работе была решена задачи предсказания зарплаты игроков NBA с приемлемыми результатами. В ходе работы были

сформированы итоговые датасеты для анализа, которые включают статистику от одного до последних шести сезонов, обработаны признаки.

Произведен анализ корреляции, по итогам которого был построен датасет с данными, которые слабо коррелируют друг с другом, а также факторный анализ и построен соответствующий датасет. Однако результаты регрессии на таким данных несколько хуже, чем на полных данных.

Были исследованы простые регрессоры (линейные, на основе ближайших соседей, на основе деревьев) и регрессоры на основе ансамблирования (бустинг, бэггинт, стэкинг). Наилучший результат показали регрессоры AdaBoostRegressor + DecisionTreeRegressor и RandomForestRegressor. Однако в некоторых случаях, можно использовать регрессор RidgeCV, который показал средние результаты, однако скорость его работы в разы быстрее.

Были описаны проблемы, мешающие регрессии, и способы их решения, которые были применены в ходе работы.

Представлена визуализация результата, на которой видно, что регрессия дала хорошие результаты и большинство предсказаний не сильно отличаются от актуальных значений зарплат.

# ПРИЛОЖЕНИЕ А

## Исходный код программы

```
#%%

import pandas as pd

stats14 = pd.read_csv('import/stats14-15.csv')
stats15 = pd.read_csv('import/stats15-16.csv')
stats16 = pd.read_csv('import/stats16-17.csv')
stats17 = pd.read_csv('import/stats17-18.csv')
stats18 = pd.read_csv('import/stats18-19.csv')
stats19 = pd.read_csv('import/stats19-20.csv')

#%%

stats19

#%%

# def add_columns(stats, prev_stats=None):
#     stats['TeamChanged'] = 0

#     for p in stats.Player:
#     if prev_stats is not None and not any(item in stats[['Tm']] for item
in prev_stats[['Tm']]):
#           stats.loc[stats['Player'] == p, 'TeamChanged'] += 1

#%%

def remove_columns(stats):

      # Удалить строки со статистикой игрока за команды, если их больше 1
(оставить общую статистику за сезон)
      for p in stats.Player:
      if len(stats[stats.Player == p]) > 1:
            stats.drop(stats[(stats.Player    ==    p)    &    (stats.Tm    !=
'TOT')].index, inplace=True)
            #stats.loc[stats['Player'] == p, 'TeamChanged'] += 1

      # Сделать столбец с именем индексом
```

```
        stats.set_index('Player', inplace=True)

        # Убрать данные ранга и команды
        stats.drop(['Rk', 'Tm'], axis=True, inplace=True)

#%%

# add_columns(stats14)
# add_columns(stats15, stats14)
# add_columns(stats16, stats15)
# add_columns(stats17, stats16)
# add_columns(stats18, stats17)
# add_columns(stats19, stats18)

remove_columns(stats14)
remove_columns(stats15)
remove_columns(stats16)
remove_columns(stats17)
remove_columns(stats18)
remove_columns(stats19)

#%%

# def get_pos_and_age(stats, years_ago)

#%%

info = stats19[['Pos', 'Age']]

stats14['Age'] = stats17['Age'] + 5
stats15['Age'] = stats17['Age'] + 4
stats16['Age'] = stats17['Age'] + 3
stats17['Age'] = stats17['Age'] + 2
stats18['Age'] = stats18['Age'] + 1

idxs = [idx for idx in stats18.index if idx not in info.index]
info = info.append(stats18.loc[idxs, ['Pos', 'Age']])

idxs = [idx for idx in stats17.index if idx not in info.index]
info = info.append(stats17.loc[idxs, ['Pos', 'Age']])
```

```
idxs = [idx for idx in stats16.index if idx not in info.index]
info = info.append(stats16.loc[idxs, ['Pos', 'Age']])

idxs = [idx for idx in stats15.index if idx not in info.index]
info = info.append(stats15.loc[idxs, ['Pos', 'Age']])

idxs = [idx for idx in stats14.index if idx not in info.index]
info = info.append(stats14.loc[idxs, ['Pos', 'Age']])

#%%

stats14.drop(info, axis=1, inplace=True)
stats15.drop(info, axis=1, inplace=True)
stats16.drop(info, axis=1, inplace=True)
stats17.drop(info, axis=1, inplace=True)
stats18.drop(info, axis=1, inplace=True)
stats19.drop(info, axis=1, inplace=True)

stats14.rename(columns=lambda x: x + '_14', inplace=True)
stats15.rename(columns=lambda x: x + '_15', inplace=True)
stats16.rename(columns=lambda x: x + '_16', inplace=True)
stats17.rename(columns=lambda x: x + '_17', inplace=True)
stats18.rename(columns=lambda x: x + '_18', inplace=True)
stats19.rename(columns=lambda x: x + '_19', inplace=True)

#%%



#%%

salary = pd.read_csv('import/salary.csv')

#%%

salary[salary['Player'] == 'Nicolas Batum\\batumni01']

#%%

# Выбрать столбцы с именем и зарплатой на сезон 2020-21
salary = salary[['Player', '2020-21']]
```

```
# Удаление устаревшего контракта у игроков с двумя контрактами с двумя контрактами с двумя контрактами (с двумя контрактами)
salary.drop_duplicates(subset='Player', keep='last',inplace=True)

# Сделать стобец с именем индексом
salary.set_index('Player', inplace=True)

# Переименовать столбец зарплаты и сделать значения столбцов int с двумя контрактами
salary.rename(columns={'2020-21': 'Salary'}, inplace=True)
salary['Salary'] = salary['Salary'].str.replace('$', '').astype(int)

#%%

salary

#%%

michael = pd.read_csv('import/michael.csv')
michael

#%%

michael['Player'] = 'Michael Jordan'
michael = michael.drop(['Season', 'Age', 'Tm', 'Lg'], axis=1)

lbls = '_16', '_17', '_18', '_19'
for (index, row), lbl in zip(michael.iterrows(), lbls):
      print(row)
michael

#%%

one_season = salary.join(info).join(stats19)
one_season.fillna(0, inplace=True)

one_season

#%%
```

```
# убрать новичков
one_season = one_season[one_season.Pos != 0]

# убрать тех, кто не играл вообще
one_season = one_season[one_season.G_19 != 0]

one_season

#%%

one_season.shape

#%%

export_file_path = 'export/one_season.csv'
one_season.to_csv(export_file_path, header=True)

#%%

two_seasons = salary.join(info).join(stats19).join(stats18)
two_seasons.fillna(0, inplace=True)

# убрать новичков
two_seasons = two_seasons[two_seasons.Pos != 0]

# убрать тех, кто не играл вообще
two_seasons = two_seasons[(two_seasons.G_19 != 0) | (two_seasons.G_18 != 0)]

two_seasons

#%%

two_seasons.shape

#%%

export_file_path = 'export/two_seasons.csv'
two_seasons.to_csv(export_file_path, header=True)

#%%
```

```
three_seasons = salary.join(info).join(stats19).join(stats18).join(stats17)
three_seasons.fillna(0, inplace=True)

# убрать новичков
three_seasons = three_seasons[three_seasons.Pos != 0]

# убрать тех, кто не играл вообще
three_seasons = three_seasons[(three_seasons.G_19 != 0) | (three_seasons.G_18
!= 0) | (three_seasons.G_17 != 0)]

three_seasons

#%%

three_seasons.shape

#%%

export_file_path = 'export/three_seasons.csv'
three_seasons.to_csv(export_file_path, header=True)

#%%

four_seasons                                                                  =
salary.join(info).join(stats19).join(stats18).join(stats17).join(stats16)
four_seasons.fillna(0, inplace=True)

# убрать новичков
four_seasons = four_seasons[four_seasons.Pos != 0]

# убрать тех, кто не играл вообще
four_seasons = four_seasons[(four_seasons.G_19 != 0) | (four_seasons.G_18 !=
0) |
                        (four_seasons.G_17 != 0) | (four_seasons.G_16 != 0)]

four_seasons.loc['DeMarcus Cousins\\couside01']

#%%

four_seasons.shape
```

```
#%%

export_file_path = 'export/four_seasons.csv'
four_seasons.to_csv(export_file_path, header=True)

#%%

five_seasons                                                                    =
salary.join(info).join(stats19).join(stats18).join(stats17).join(stats16).joi
n(stats15)
five_seasons.fillna(0, inplace=True)

# убрать новичков
five_seasons = five_seasons[five_seasons.Pos != 0]

# убрать тех, кто не играл вообще
five_seasons = five_seasons[(five_seasons.G_19 != 0) | (five_seasons.G_18 !=
0) |
                            (five_seasons.G_17 != 0) | (five_seasons.G_16 != 0)
|
                            (five_seasons.G_15 != 0)]

five_seasons

#%%

five_seasons.shape

#%%

export_file_path = 'export/five_seasons.csv'
five_seasons.to_csv(export_file_path, header=True)
```

```
#%%

import pandas as pd

#%%

data = pd.read_csv('four_seasons.csv')
data.set_index('Player', inplace=True)

data

#%%

import seaborn as sns
import matplotlib.pyplot as plt

#%%



#%%

strong_corr_labels = data.corr()[data.corr()['Salary'] > 0.65].index

#%%

plt.figure(figsize=(10, 10))
sns.heatmap(data[strong_corr_labels].corr(), cmap='coolwarm')
plt.show()

#%%

reduced_data = data.copy()

#%%

low_corr_cols = reduced_data.drop('Salary', axis=True).columns
low_corr = abs(reduced_data.corr()) < 0.91
for i in low_corr.columns:
```

```
      for j in low_corr.columns:
      if i != j and low_corr[i][j] == False and j in low_corr_cols:
            low_corr_cols = low_corr_cols.drop(j)

len(low_corr_cols)

#%%

#export_file_path = 'export/four_seasons_low_corr.csv'
#reduced_data[cols].to_csv(export_file_path, header=True)

#%%

plt.figure(figsize=(10, 10))
sns.heatmap(reduced_data[low_corr_cols].corr(), cmap='coolwarm')
plt.show()

#%%

from IPython.display import display, Markdown

#%%

def default_analysis_for_numeric_feature(features, target):
      N = len(features)
      name = features[0].name.split('_')[0]

      display(Markdown(f'# *{name}*'))

      fig, axes = plt.subplots(2, 1, figsize=(10, 4))
      sns.barplot(ax=axes[0],  x=[feature.name  for  feature  in  features],
y=[target.corr(feature) for feature in features])
      sns.barplot(ax=axes[1],  x=[feature.name  for  feature  in  features],
y=[feature[feature!=0].mean() for feature in features])
      axes[0].set_ylabel('Correlation')
      axes[1].set_ylabel('Mean')
      plt.show()

      fig, axes = plt.subplots(N, 2, figsize=(10, N*3))
      for i in range(N):
```

```
      sns.histplot(ax=axes[i,      0],      data=features[i][features[i]!=0],
kde=False, bins=10)
      sns.regplot(ax=axes[i,        1],        x=features[i][features[i]!=0],
y=target[features[i]!=0], ci=False)
      plt.tight_layout()
      plt.show()



#%%



#%% md

## Most correlated features

#%%

most_corr_features                                                    =
data.corr()['Salary'].drop('Salary').sort_values(ascending=False).head(20)
most_corr_features.plot.bar()

plt.show()

#%% md

## Salary

#%%

#sns.displot(data['Salary'])
#plt.show()

#%%

## Numerical features analysis

#%%

cols_postfixs = ['_19', '_18', '_17', '_16']
```

```
cols_prefixs = set([col.split('_')[0] for col in reduced_data.columns if
len(col.split('_')) > 1])

for col_prefix in cols_prefixs:
      cols = [col_prefix + cols_postfix for cols_postfix in cols_postfixs]
      features = [data[col] for col in cols]

      default_analysis_for_numeric_feature(features, data['Salary'])



#%% md



#%%



#%%



#%%



#%%



#%%

from sklearn.decomposition import FactorAnalysis

#%%

def pos_to_num(pos):
      pos_dict = {'PG': 1, 'SG': 2, 'SF': 3, 'PF': 4, 'C': 5,
            'PG-SG': 1.5, 'SG-SF': 2.5, 'SF-PF': 3.5, 'PF-C': 4.5,
            'SG-PG': 1.5, 'SF-SG': 2.5, 'PF-SF': 3.5, 'C-PF': 4.5,
```

```
            'SF-C': 4}
      return pos_dict[pos]

reduced_data['Pos'] = reduced_data['Pos'].apply(pos_to_num)

#%%

transformer = FactorAnalysis()
data_transformed = transformer.fit_transform(reduced_data[low_corr_cols])

#%%

fa_components = pd.DataFrame([[el for el in lst if el] for lst in
transformer.components_],
                             columns=low_corr_cols)
fa_components.dropna(inplace=True)

fa_components

#%%

i = 0
fig, axs = plt.subplots(6, 3, figsize=(15, 30))
for i in range(len(fa_components)):
      axs[i // 3, i % 3].bar(fa_components.columns, fa_components.iloc[i])
      axs[i // 3, i % 3].set_title(f'Factor {i}')
      axs[i  //  3,  i  %  3].set_xticklabels(fa_components.columns,
rotation='vertical')
      i += 1

plt.tight_layout()
plt.show()

#%%



#%%

ransformer = FactorAnalysis()
```

```
data_transformed = transformer.fit_transform(reduced_data.drop(['Salary'], axis=True))

#%%

fa_components = pd.DataFrame([[el for el in lst if el] for lst in transformer.components_],
                             columns=reduced_data.drop(['Salary'], axis=True).columns)
fa_components.dropna(inplace=True)

fa_components

#%%

fa_data = pd.DataFrame([[el for el in lst if el] for lst in data_transformed])
fa_data.set_index(data.index, inplace=True)

fa_data

#%%

for i in range(len(fa_components)):
      plt.figure(figsize=(15, 10))
      plt.bar(fa_components.columns, fa_components.iloc[i])
      plt.suptitle(f'Factor {i}')
      plt.rc('xtick', labelsize=12)
      plt.xticks(rotation='vertical')
      plt.tight_layout()
      plt.show()


#%%

import pandas as pd

#%%

def pos_to_num(pos):
      pos_dict = {'PG': 1, 'SG': 2, 'SF': 3, 'PF': 4, 'C': 5,
```

```
            'PG-SG': 1.5, 'SG-SF': 2.5, 'SF-PF': 3.5, 'PF-C': 4.5,
            'SG-PG': 1.5, 'SF-SG': 2.5, 'PF-SF': 3.5, 'C-PF': 4.5,
            'SF-C': 4}
      return pos_dict[pos]

#%%

one_season = pd.read_csv('import/one_season.csv')
one_season.set_index('Player', inplace=True)
one_season.Pos = one_season.Pos.apply(pos_to_num)
one_season

#%%

two_seasons = pd.read_csv('import/two_seasons.csv')
two_seasons.set_index('Player', inplace=True)
two_seasons.Pos = two_seasons.Pos.apply(pos_to_num)
two_seasons

#%%

three_seasons = pd.read_csv('import/three_seasons.csv')
three_seasons.set_index('Player', inplace=True)
three_seasons.Pos = three_seasons.Pos.apply(pos_to_num)
three_seasons

#%%

four_seasons = pd.read_csv('import/four_seasons.csv')
four_seasons.set_index('Player', inplace=True)
four_seasons.Pos = four_seasons.Pos.apply(pos_to_num)
four_seasons

#%%

five_seasons = pd.read_csv('import/five_seasons.csv')
five_seasons.set_index('Player', inplace=True)
five_seasons.Pos = five_seasons.Pos.apply(pos_to_num)
five_seasons

#%%
```

```
four_seasons_reduced = pd.read_csv('import/four_seasons_reduced.csv')
four_seasons_reduced.set_index('Player', inplace=True)
four_seasons_reduced.Pos = four_seasons_reduced.Pos.apply(pos_to_num)
four_seasons_reduced

#%%

four_seasons_low_corr = pd.read_csv('import/four_seasons_low_corr.csv')
four_seasons_low_corr.set_index('Player', inplace=True)
four_seasons_low_corr = four_seasons_low_corr.join(four_seasons['Salary'])
four_seasons_low_corr.Pos = four_seasons_low_corr.Pos.apply(pos_to_num)
four_seasons_low_corr

#%%

from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import *
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression, RidgeCV, SGDRegressor,
LassoCV, Ridge
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import Normalizer, StandardScaler, MinMaxScaler,
RobustScaler
from sklearn.feature_selection import RFE
import seaborn as sns
import numpy as np

#%%

my_data = four_seasons.copy() #four_seasons.copy() #drop('Salary',
axis=True)

scaler = StandardScaler().fit(my_data)
#scaler = RobustScaler().fit(my_data)
my_data = pd.DataFrame(scaler.transform(my_data), columns=my_data.columns,
index=my_data.index)

#%%
```

```
#%%

X_train, X_test, y_train, y_test = train_test_split(my_data.drop('Salary',
axis=True), my_data['Salary'],
                                                    test_size=0.1,
random_state=9)

#%%

def estimate_regression(name, regressor):

      regressor.fit(X_train, y_train)
      y_pred = regressor.predict(X_test)

      print(regressor)
      print('Mean   Absolute   Error:',   metrics.mean_absolute_error(y_test,
y_pred))
      print('Max Error:', metrics.max_error(y_test, y_pred))
      print('R2 Score:', metrics.r2_score(y_test, y_pred))
      print()

      return estimate_df.append({'Name': name, 'Regressor': regressor, 'R2':
metrics.r2_score(y_test, y_pred),
                                 'MeanAbsError':
metrics.mean_absolute_error(y_test, y_pred),
                                 'MaxError':          metrics.max_error(y_test,
y_pred)}, ignore_index=True)

#%%

estimate_df     =     pd.DataFrame(columns=['Name',     'Regressor',     'R2',
'MeanAbsError', 'MaxError'])

#%% md

## Простые регрессоры

#%%
```

```
simple_regression_names    =    ['LinearRegression',    'Ridge',    'RidgeCV',
'DecisionTreeRegressor', 'KNeighborsRegressor',
                         'SGDRegressor']
simple_regressors = [LinearRegression(copy_X=True),
                  Ridge(),
                  RidgeCV(),
                  DecisionTreeRegressor(random_state=0,     criterion='mae',
max_depth=4),
                  KNeighborsRegressor(n_neighbors=10, weights='distance'),
                  SGDRegressor()
                  ]

#%%

for name, regressor in zip(simple_regression_names, simple_regressors):
      estimate_df = estimate_regression(name, regressor)


#%% md

## Ансамблирование

#%% md

## AdaBoost

#%%

for name, regressor in zip(simple_regression_names, simple_regressors):
      estimate_df = estimate_regression(f'AdaBoostRegressor + {name}',
                                        AdaBoostRegressor(regressor,
random_state=0, learning_rate=2, n_estimators=100))

#%% md

##  BaggingRegressor

#%%

for name, regressor in zip(simple_regression_names, simple_regressors):
```

```
      estimate_df = estimate_regression(f'BaggingRegressor + {name}',
                                        BaggingRegressor(regressor,
random_state=0, n_estimators=100))

#%% md

## Остальные

#%%

complex_regression_names           =           ['GradientBoostingRegressor',
'HistGradientBoostingRegressor', 'RandomForestRegressor']
complex_regressors        =        [GradientBoostingRegressor(random_state=0,
n_estimators=150),
                  HistGradientBoostingRegressor(random_state=0,
max_iter=400),
                  RandomForestRegressor(random_state=0)]

#%%

for name, regressor in zip(complex_regression_names, complex_regressors):
      estimate_df = estimate_regression(name, regressor)

#%% md

## Стекинг

#%% md

### worst

#%%

top_regressors_estims           =           estimate_df.sort_values(by='R2',
ascending=False).tail(5)['Regressor'].values
top_regressors_names            =           estimate_df.sort_values(by='R2',
ascending=False).tail(5)['Name'].values

#%%
```

```
top_regressors = [(name, reg) for reg, name in zip(top_regressors_estims,
top_regressors_names)]

#%%

#estimate_df      =      estimate_regression(f'Stacking      +      worst',
StackingRegressor(estimators=top_regressors,
#                  final_estimator=RandomForestRegressor(random_state=0)))

#%% md

### best

#%%

top_regressors_estims           =           estimate_df.sort_values(by='R2',
ascending=False).head(5)['Regressor'].values
top_regressors_names            =           estimate_df.sort_values(by='R2',
ascending=False).head(5)['Name'].values

#%%

top_regressors = [(name, reg) for reg, name in zip(top_regressors_estims,
top_regressors_names)]

#%%


estimate_df       =       estimate_regression(f'Stacking       +       best',
StackingRegressor(estimators=top_regressors,
                  final_estimator=RandomForestRegressor(random_state=0)))

#%% md

## Сравнение результатов

#%%

estimate_df

#%%
```

```
estimate_df[['Name', 'R2']].sort_values(by='R2')

#%%

import matplotlib.pyplot as plt

estimate_df.sort_values('R2').plot.bar('Name', 'R2')
estimate_df.sort_values('MeanAbsError',     ascending=False).plot.bar('Name',
'MeanAbsError')
estimate_df.sort_values('MaxError',         ascending=False).plot.bar('Name',
'MaxError')

#%%

top5 = estimate_df.sort_values(by='R2', ascending=False).head(5)
top5

#%%

def estimate_regression_n_times(regressor, N):
      r2 = 0
      mean_abs_error = 0
      max_error = 0

      for i in range(N):
      X_train,          X_test,          y_train,          y_test          =
train_test_split(my_data.drop('Salary', axis=True), my_data['Salary'],
                                                                  test_size=0.1,
random_state=10+i)

      regressor.fit(X_train, y_train)
      y_pred = regressor.predict(X_test)

      r2 += metrics.r2_score(y_test, y_pred)
      mean_abs_error += metrics.mean_absolute_error(y_test, y_pred)
      max_error += metrics.max_error(y_test, y_pred)

      top5.loc[top5['Regressor'] == regressor, 'R2'] = r2 / N
      top5.loc[top5['Regressor']    ==    regressor,    'MeanAbsError']    =
mean_abs_error / N
```

```
      top5.loc[top5['Regressor'] == regressor, 'MaxError'] = max_error / N


#%%

for regressor in top5['Regressor']:
      estimate_regression_n_times(regressor, 10)

top5

#%%

top5.sort_values('R2').plot.bar('Name', 'R2')
top5.sort_values('MeanAbsError',            ascending=False).plot.bar('Name',
'MeanAbsError')
top5.sort_values('MaxError', ascending=False).plot.bar('Name', 'MaxError')

#%%

for index, row in top5.sort_values('R2').iterrows():
      print(top5.sort_values('R2').loc[index]['Name'],                    ":",
top5.sort_values('R2').loc[index]['R2'])

print()
for index, row in top5.sort_values('MeanAbsError').iterrows():
      print(top5.sort_values('MeanAbsError',
ascending=False).loc[index]['Name'], ":",
      top5.sort_values('MeanAbsError',
ascending=False).loc[index]['MeanAbsError'])

print()
for index, row in top5.sort_values('MaxError', ascending=False).iterrows():
      print(top5.sort_values('MaxError',
ascending=False).loc[index]['Name'], ":",
      top5.sort_values('MaxError').loc[index]['MaxError'])

#%%

r2 = []
mean_abs_error = []
max_error = []
```

```
test_size_range = np.arange(0.1, 1, 0.1)

for test_size in test_size_range:

      r2_sum = 0
      mean_abs_error_sum = 0
      max_error_sum = 0

      for i in range(10):
      X_train,          X_test,          y_train,          y_test          =
train_test_split(my_data.drop('Salary', axis=True), my_data['Salary'],

test_size=test_size, random_state=i)
      regressor                                                             =
BaggingRegressor(base_estimator=DecisionTreeRegressor(criterion='mae',

max_depth=4,

random_state=0),
                                    n_estimators=100, random_state=0)

      regressor.fit(X_train, y_train)
      y_pred = regressor.predict(X_test)

      r2_sum += metrics.r2_score(y_test, y_pred)
      mean_abs_error_sum += metrics.mean_absolute_error(y_test, y_pred)
      max_error_sum += metrics.max_error(y_test, y_pred)

      r2.append(r2_sum / 10)
      mean_abs_error.append(mean_abs_error_sum / 10)
      max_error.append(max_error_sum / 10)

plt.plot(test_size_range, r2)
plt.show()
plt.plot(test_size_range, mean_abs_error)
plt.show()
plt.plot(test_size_range, max_error)
plt.show()

#%%
```

```
# X_train, X_test, y_train, y_test = train_test_split(my_data.drop('Salary',
axis=True), my_data['Salary'],
#                                                     test_size=test_size,
random_state=i)
#                                 regressor                                 =
BaggingRegressor(base_estimator=DecisionTreeRegressor(criterion='mae',
#
max_depth=4,
#
random_state=0),
#                               n_estimators=100, random_state=0)

# regressor.fit(X_train, y_train)
# y_pred = regressor.predict(X_test)

#%%

# from sklearn.inspection import permutation_importance

# r = permutation_importance(regressor,
#                               X_test, y_test,
#                               n_repeats=10,
#                               random_state=0)

#%%

# for i in r.importances_mean.argsort()[:-20:-1]:
#     plt.bar(my_data.columns[i], r.importances_mean[i])
# plt.xticks(rotation='vertical')
# plt.show()

#%%

X_train,  X_test,  y_train,  y_test  =  train_test_split(my_data.drop('Salary',
axis=True), my_data['Salary'],
                                                    test_size=0.3,
random_state=551)

regressor                                                                 =
BaggingRegressor(base_estimator=DecisionTreeRegressor(criterion='mae',
```

```
max_depth=4,

random_state=0),
                              n_estimators=100, random_state=0)

regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Max Error:', metrics.max_error(y_test, y_pred))
print('R2 Score:', metrics.r2_score(y_test, y_pred))
print()

#%%

compare_salary = pd.DataFrame(columns=['Actual', 'Prediction'])

#%%

inversed_data = pd.DataFrame(X_test, columns=my_data.columns)
inversed_data['Salary'] = y_pred
inversed_data     =     pd.DataFrame(scaler.inverse_transform(inversed_data),
columns=inversed_data.columns,
                              index=inversed_data.index)

compare_salary['Prediction'] = inversed_data['Salary']

#%%

inversed_data = pd.DataFrame(X_test, columns=my_data.columns)
inversed_data['Salary'] = y_test
inversed_data     =     pd.DataFrame(scaler.inverse_transform(inversed_data),
columns=inversed_data.columns,
                              index=inversed_data.index)

compare_salary['Actual'] = inversed_data['Salary']

#%%

compare_salary.sort_values('Prediction', ascending=False)
```

```
#%%

export_file_path = 'export/compare_salary.csv'
compare_salary.to_csv(export_file_path, header=True)

#%%

import matplotlib.pyplot as plt
for i in range(0, len(compare_salary), 25):
      compare_salary[i:i + 25].plot.bar()
      plt.show()
```

```
#%%

import pandas as pd

compare_salary = pd.read_csv('compare_salary.csv')
compare_salary.set_index('Player', inplace=True)
compare_salary

#%%

import matplotlib.pyplot as plt

for i in range(0, len(compare_salary), 25):
      compare_salary[i:i + 25].plot.bar()
      plt.show()

#%%

import numpy as np

# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
      df = df.select_dtypes(include =[np.number]) # keep only numerical columns
      # Remove rows and columns that would lead to df being singular
      df = df.dropna('columns')
      df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
      columnNames = list(df)
      if len(columnNames) > 100: # reduce the number of columns for matrix inversion of kernel density plots
      columnNames = columnNames[:10]
      df = df[columnNames]
      ax = pd.plotting.hist_series(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
      corrs = df.corr().values
      for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
      ax[i, j].annotate('%.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center', size=textSize)
      plt.suptitle('Scatter and Density Plot')
      plt.show()
```

#%%