Ярылгасимов Султан
Random Forest

Часть 1. Теория

Причина выбора темы

Random Forest — весьма популярная модель для решения задач регрессии, классификации и кластеризации. При обсуждении машинного обучение нередко всплывает именно это название, так как она широко используется в системах поддержки принятия решений и интеллектуального анализа данных. На втором курсе нам удалось лишь вскользь познакомиться с этим алгоритмом, когда мы решали задачу распознавания произнесенных цифр. Здесь, под «вскользь» имеется в виду, что мы абсолютно не пытались разбираться как оно работает и обращались к нему как к «черному ящику». Однако, написание этого реферата стало причиной того, что пришлось познакомиться с этой моделью и поближе и заглянуть в то, как оно работает.

Общее представление о «случайном лесе»

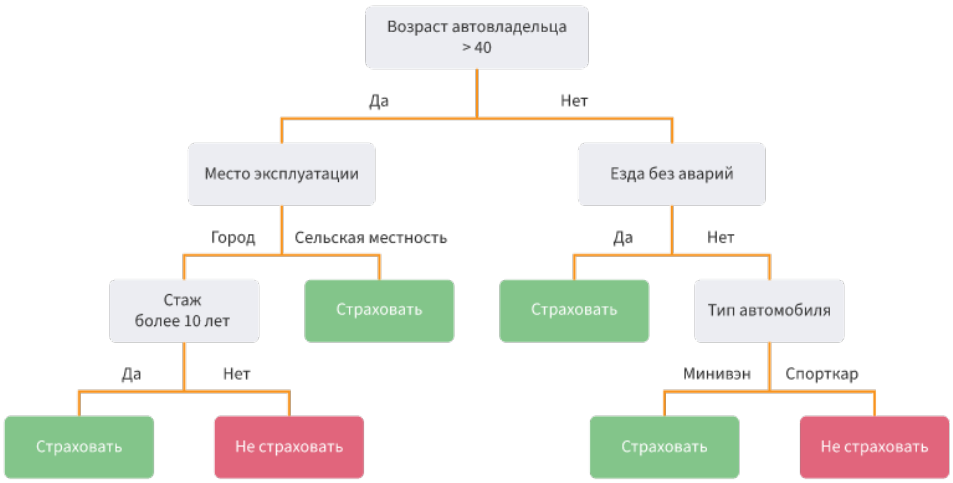
Random Forest — алгоритм машинного обучения, предложенный американскими математиками-статистиками Лео Брейманом и Адель Катлер. Основная идея заключается в использовании ансамбля решающих деревьев (понятие, в котором тоже придется разобраться), каждое из которых само по себе не дает высокое качество решения задачи, но вкупе показывают достаточно хороший результат.

Вкратце, алгоритм представляет собой голосования различных деревьев. В случае задачи классификации, объекту будет присвоен тот класс, который наберет больше всех голосов. В случае же регрессии, можно усреднить результаты с равными весами.

Алгоритм базируется на идее бэггинга (сокращение от английского bootstrap aggregating) Бреймана, но перед тем как переходить к нему, нужно выяснить, что из себя представляет «decision tree»

Структура решающего дерева (Decision Tree)

Решающие деревья представляют собой иерерхическую древовидную структуру, состоящую из решающих правил вида «Если ..., то ...». Правила автоматически генерируются в процессе обучения на обучающем множестве и, поскольку они формулируются практически на естественном языке, деревья решений как аналитические модели более вербализуемы и интерпретируемы, чем, скажем, нейронные сети. Снизу приведен пример того, как дерево может выглядеть для решения проблемы страхования автовладельца.



Разберемся в терминах, которые будем использовать далее.

Узел	Внутренний узел дерева, узел проверки
Корневой узел	Начальный узел дерева решений
Лист	Конечный узел дерева, узел решения, терминальный узел
Решающее правило	Условие в узле, проверка

Собственно, само дерево решений — это метод представления решающих правил в иерархической структуре, состоящей из элементов двух типов — узлов (node) и листьев (leaf). В узлах находятся решающие правила и производится проверка соответствия примеров этому правилу по какому-либо артибуту обучающего множества.

В простейшем случае, в результате проверки, множество примеров, попавших в узел, разбивается на два подмножества, в одно из которых попадают примеры, удовлетворяющие правилу, а в другое — не удовлетворяющие.

Затем к каждому подмножеству вновь применяется правило и процедура рекурсивно повторяется пока не будет достигнуто некоторое условие остановки алгоритма. В результате в последнем узле проверка и разбиение не производится и он объявляется листом. Лист определяет решение для каждого попавшего в него примера. Для дерева классификации — это класс, ассоциируемый с узлом, а для дерева регрессии — соответствующий листу модальный интервал целевой переменной.

Процесс построения решающего дерева

Процесс построения деревьев решений заключается в последовательном, рекурсивном разбиении обучающего множества на подмножества с применением решающих правил в узлах. Процесс разбиения продолжается до тех пор, пока все узлы в конце всех ветвей не будут объявлены листьями. Объявление узла листом может произойти естественным образом (когда он будет содержать единственный объект, или объекты только одного класса), или по достижении некоторого условия остановки, задаваемого пользователем (например, минимально допустимое число примеров в узле или максимальная глубина дерева).

Алгоритмы построения деревьев решений относят к категории так называемых жадных алгоритмов, то есть тех, которые допускают, что локально-оптимальные решения на каждом шаге (разбиения в узлах), приводят к оптимальному итоговому решению. В случае деревьев решений это означает, что если один раз был выбран атрибут, и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться назад и выбрать другой атрибут, который дал бы лучшее итоговое разбиение. Поэтому на этапе построения нельзя сказать обеспечит ли выбранный атрибут, в конечном итоге, оптимальное разбиение.

В основе большинства современных алгоритмов лежит следующий принцип: Пусть задано обучающее множество S, содержащее n примеров, для каждого из которых задана метка класса C_i (i=1..k), и m атрибутов A_j (j=1..m), которые, как предполагается, определяют принадлежность объекта к тому или иному классу. Тогда возможны три случая:

1. Все примеры множества S имеют одинаковую метку класса C_i (т.е. все обучающие примеры относятся только к одному классу). Очевидно, что обучение в этом случае не имеет смысла, поскольку все примеры, предъявляемые модели, будут одного класса, который и «научится» распознавать модель. Само дерево решений в этом случае будет представлять собой лист, ассоциированный с классом C_i Практическое использование такого дерева бессмысленно, поскольку любой новый объект оно будет относить только к этому классу.

2. Множество S вообще не содержит примеров, т.е. является пустым множеством. В этом случае для него тоже будет создан лист (применять правило, чтобы создать узел, к пустому множеству бессмысленно), класс которого будет выбран из другого множества (например, класс, который наиболее часто встречается в родительском множестве).

3. Множество S содержит обучающие примеры всех классов C_k. В этом случае требуется разбить множество S на подмножества, ассоциированные с классами. Для этого выбирается один из атрибутов A_j множества S который содержит два и более уникальных значения (a_1,a_2,...,a_p), где p — число уникальных значений признака. Затем множество S разбивается на p подмножеств (S_1,S_2,...,S_p), каждое из которых включает примеры, содержащие соответствующее значение атрибута. Затем выбирается следующий атрибут и разбиение повторяется. Это процедура будет рекурсивно повторяться до тех пор, пока все примеры в результирующих подмножествах не окажутся одного класса.

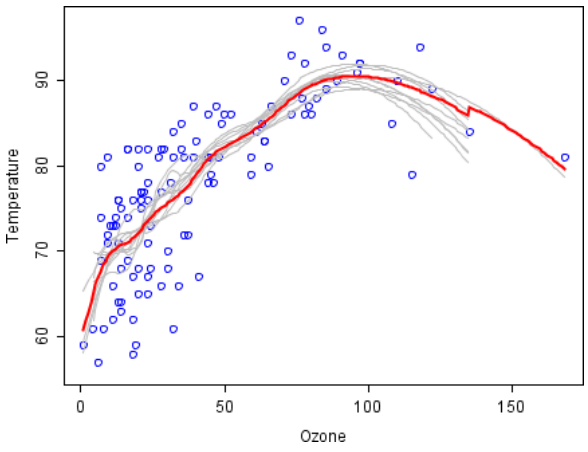
Bagging

После того как мы разобрались в том, что же такое решающее дерево, мы стали лучше представлять, как может выглядеть лес из таких деревьев, однако все еще стоит вопрос о том, как же эти леса строить. Очевидно, что если передать всем деревьям одинаковый набор данных, то они все будут построены одинаково и в построении леса не будет смысла. Так же, не факт, что использование фолдов на подобии того, как это делается в скользящем контроле (cross-validation) даст хороший результат.

Так вот, для усреднения моделей используется bagging, который был упомянут выше. Википедия дает следующее определение: 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, его можно использовать с любым видом метода. Бэггинг является частным видом усреднения модели.

Суть алгоритма состоит в том, что, допустим, нам дана выборка D, размера n. Бэггинг образует m новых наборов с размерами N, путем выборки из D равномерно и с возвратом (повторная выборка). Некоторые объекты попадут в выборку два или более раз, в то время как в среднем $N\left(1-1/N\right)^N$ (при больших N примерно N/e) образцов оказываются не вошедшими в набор (out-of-bag). Если N=n, то для достаточно больших выборок каждый набор будет содержать около $1-1/e$, то есть около 63.2% уникальных экземпляров из D, а остальные будут повторениями.

Справа приведен пример того, как решена проблема зависимости концентрации озона от температуры. Вместо построения единого сглаживателя, из всего набора данных были извлечено 100 выборок бутстрэпов данных. Наборы могли не совпадать между собой, но их среднее и дисперсия были одинаковы. Затем было сделано



предсказание на основе 100 выборок и взяв среднее от них, получили сборный показатель, который указан красной криво на рисунке.

Bagging в Random Forest

Бэггинг в случайном лесу проводит так же дополнительное действие между разбиением данных и составления деревьев по ним. Случайный лес (по крайней мере, тот, что реализован в sklearn) случайно отбирает признаки по тому же методу и только после этого, строит по ним деревья.

Преимущества и недостатки

Преимущества	**Недостатки**
Имеет высокую точность предсказания, на большинстве задач будет лучше линейных алгоритмов; точность сравнима с точностью бустинга	В отличие от одного дерева, результаты случайного леса сложнее интерпретировать
Практически не чувствителен к выбросам в данных из-за случайного сэмлирования	Нет формальных выводов (p-values), доступных для оценки важности переменных
Не чувствителен к масштабированию (и вообще к любым монотонным преобразованиям) значений признаков, связано с выбором случайных подпространств	Алгоритм работает хуже многих линейных методов, когда в выборке очень много разреженных признаков (тексты, Bag of words)
Не требует тщательной настройки параметров, хорошо работает «из коробки». С помощью «тюнинга» параметров можно достичь прироста от 0.5 до 3% точности в зависимости от задачи и данных	Алгоритм склонен к переобучению на некоторых задачах, особенно на зашумленных данных
Способен эффективно обрабатывать данные с большим числом признаков и классов	Для данных, включающих категориальные переменные с различным количеством уровней, случайные леса предвзяты в пользу признаков с большим количеством уровней: когда у признака много уровней, дерево будет сильнее подстраиваться именно под эти признаки, так как на них можно получить более высокое значение оптимизируемого функционала (типа прироста информации)
Высокая параллелизуемость и масштабируемость.	Больший размер получающихся моделей. Требуется O(NK) памяти для хранения модели, где K — число деревьев.

Часть 2. Практика

Неудача

Хотелось использовать датасет, который использовал на протяжении всего семестра — результаты футбольных матчей. Уже после того как я построил признаки, опираясь на усреднение результатов команды за последние 5 матчей, и считая, что команда «на выезде» и

«дома» играет по-разному, выяснилось, что никакой корреляции нет. Я построил модель, которая показывает результат равный 60%, однако оказалось, что она пытается угадать результат. В данной проблеме была замечательная интерпретируемость, так как можно было посчитать, что несколько товарищей, которые думают, что на результат команды влияют разные факторы, пытаются предугадать исход матча. Возможно, это еще получится реализовать нужно будет лишь найти коррелирующие данные, что лишь звучит просто.

В итоге, потратив уйму времени на изучение своего датасета, пришлось остановится на выборе другого, а именно на датасете об информации о выживших на Титанике, достаточно широко используемый для обучения датасете.

Информация о наборе данных.

Датасет представляет из себя таблицу с 10 столбцами и суммарно состоящую из 1309 строк, разделенную на обучающую и тестовую выборки. Столбцы содержат следующую информацию:

- **Passenger ID —** уникальный идентификационный номер пассажира (номер билета).
- **Survived —** строка содержащая проверочную информацию: выжил — 1, нет — 0
- **Pclass —** класс билета: 1 — Высший, 2 — Средний, 3 — Низший.
- **Age —** возраст пассажира.
- **Sibsp** — количество братьев-сестер и супруг на корабле (от англ. siblings/spouse).
- **Parch** — количество детей или родителей на корабле (от англ. parents/children).
- **Ticket** — номер билета.
- **Fare** — стоимость проезда пассажира.
- **Cabin** — номер каюты.
- **Embarked** — порт, откуда сел пассажир: S — Southampton, C — Cherbourgh, Q — Queenstown.

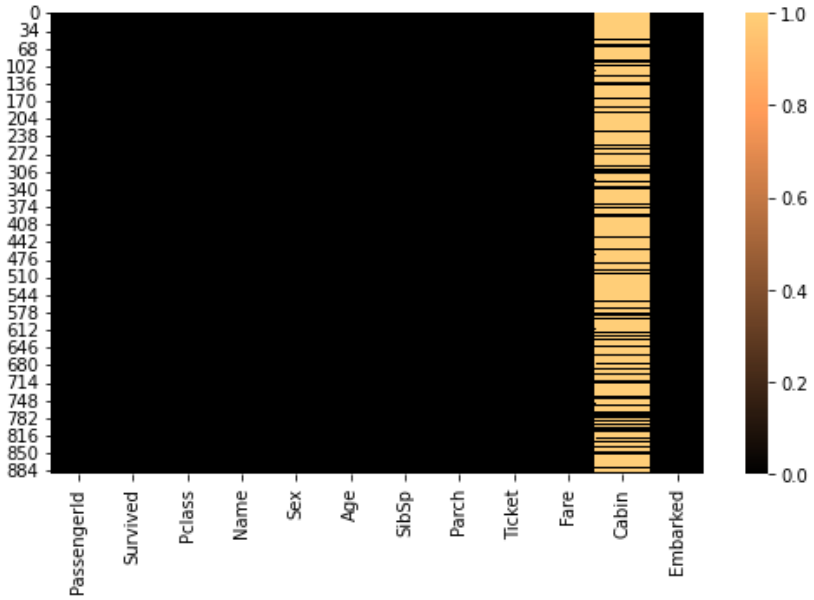
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Kimball, Mr. Edwin Nelson Jr	male	NaN	NaN	NaN	347082	NaN	C23 C25 C27	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

Как можно видеть, столбцы Age, Cabin и Embarked имеют пропуски, поэтому заполним их.

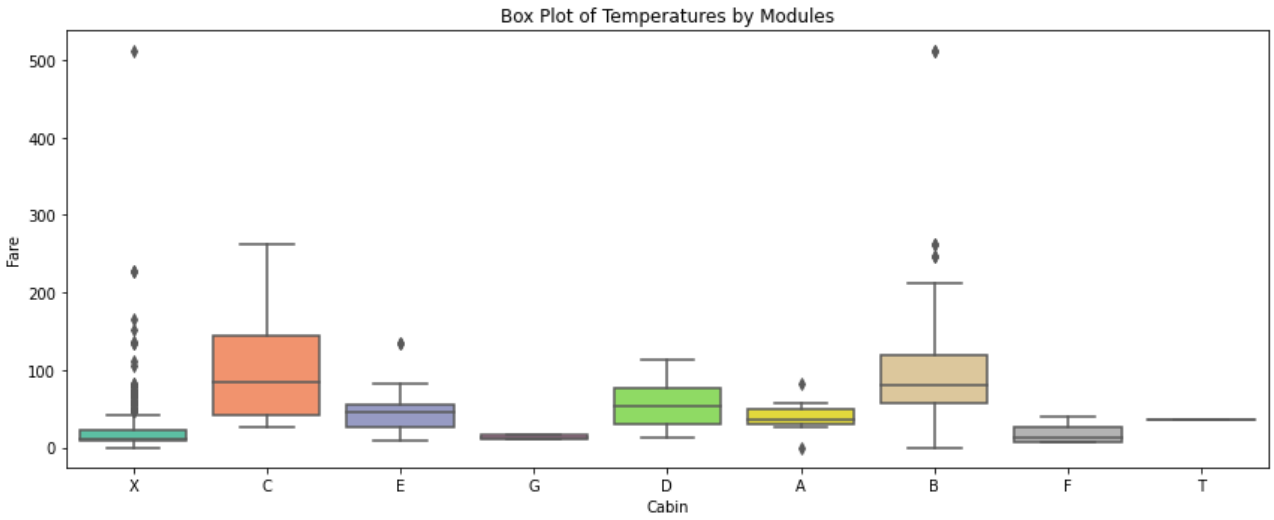
Заполнение пропущенных данных.

2 строки не содержат значение столбца **Embarked**. Заполним их самым полулярным портом (S — Southampton). Если бы отсутствовало больше значений, то имел бы смысл заполнять их чем-то более сложным, например, обратиться к вероятностям.

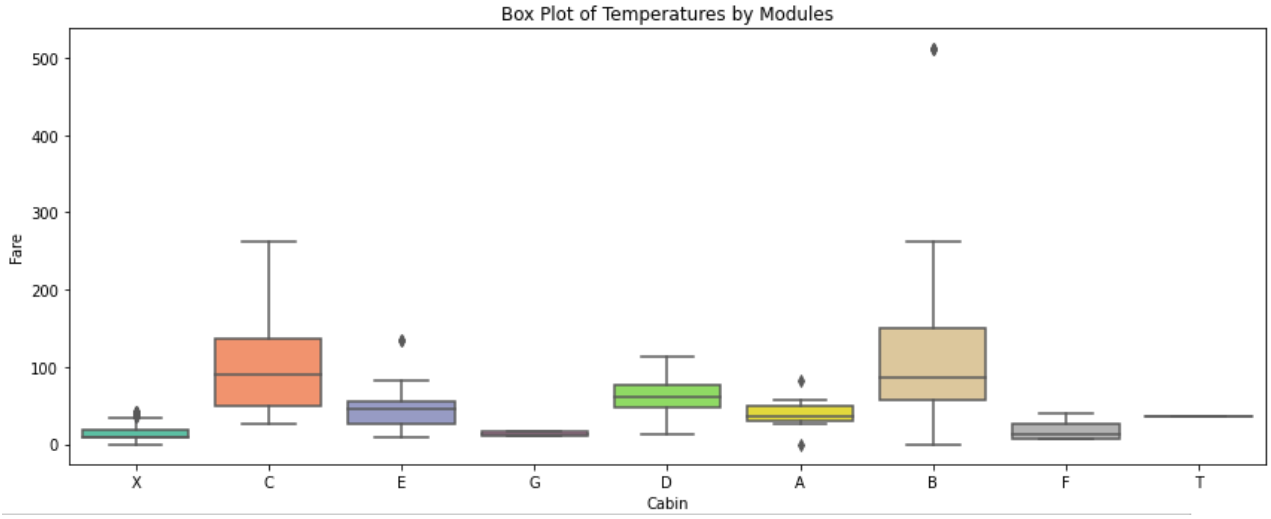
Заполение Age происходит более интересным образом. Для этого, обратим внимание, что все имена в наборе содержат приставки «Mr.», «Ms.», «Mrs», «Dr.», «Miss», «Master». Заполнять пропуски усредняя их по данным званиям будет правильнее и точнее. Объединяя похожие, получим, что средний возраст «Miss» - 22, «Mrs.» - 36, «Mr.» - 33, «Master» - 5.



Как видно, почти все данные о каютах потеряны, однако можно попытаться их заполнить, рассматривая только первую букву, которая отвечает за положение каюты в корабле. Пометим все неизвестный за «X», а остальные по первым буквам.



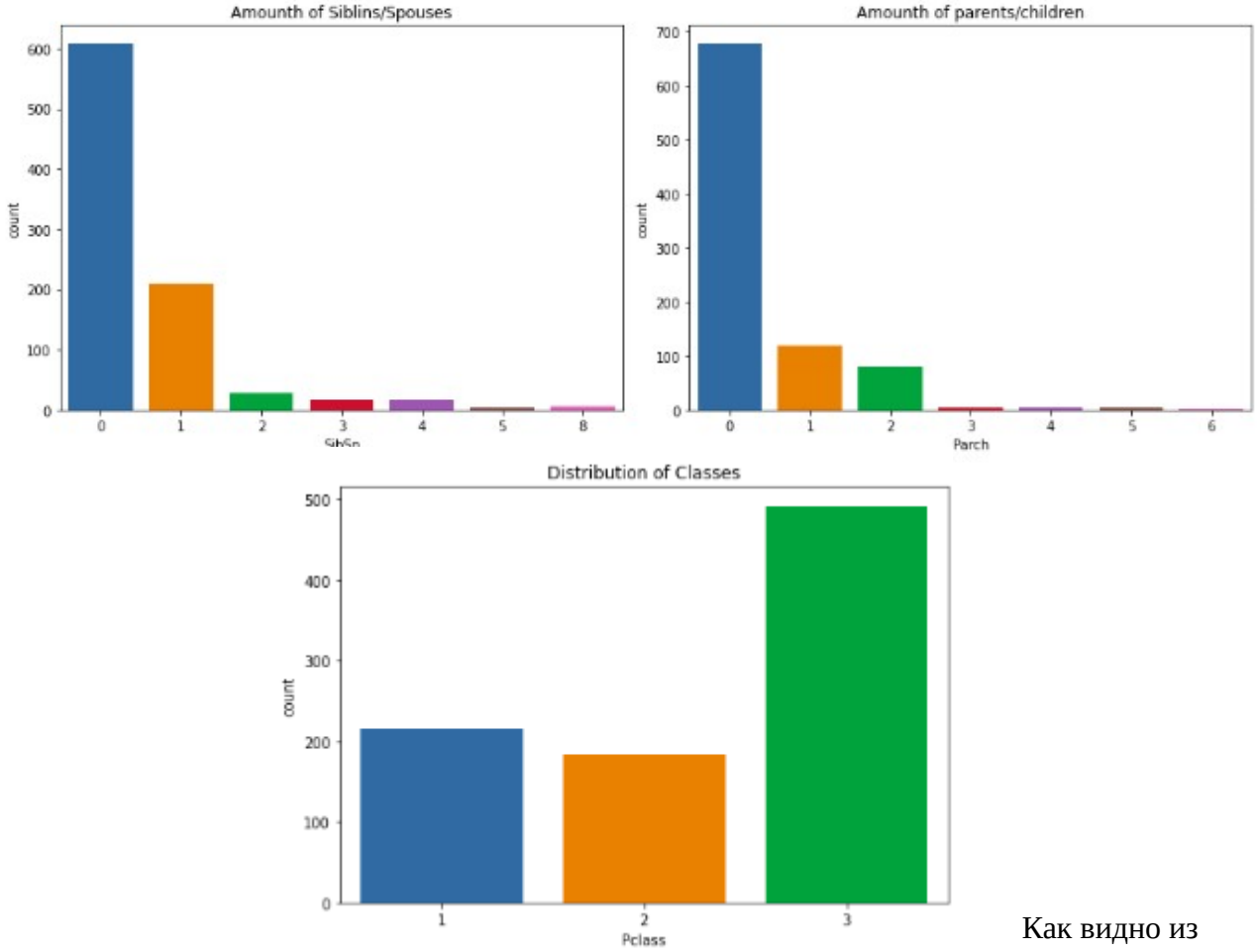
Очевидна зависимость платы пассажиров и выбора их кают, поэтому видно, что большинство из выбравших «X» заплатили относительно меньше. Однако, имеются и выбросы, которые можно распределить по классам C и B, которые имеют более высокую цену проезда. Поступим так.



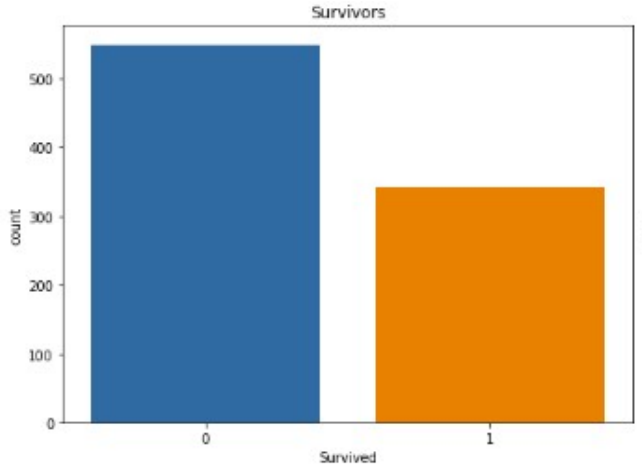
Таким образом, заполненные данные выглядят более реалистично, и тем самым мы заполнили все пропуски.

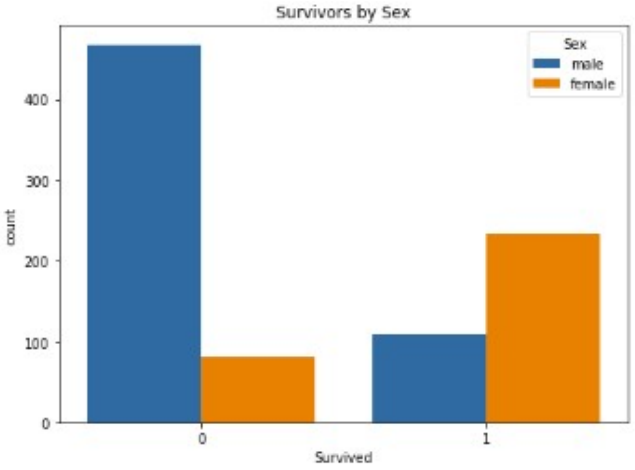
Создание признаков.

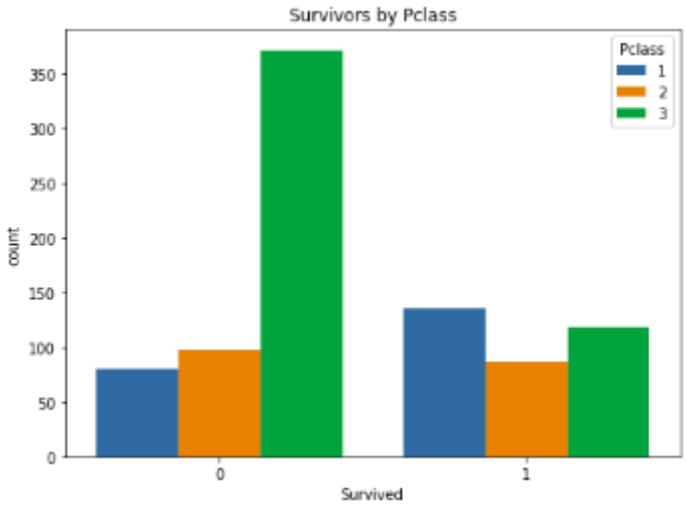
Говорят, что создание признаков занимает около 80% времени затраченного на решение проблемы машинного обучения, однако именно эта часть по большей части влияет на качество будущей модели. Суть этого процесса заключается в том, чтобы получить как можно больше информации из существующих признаков, и создание новых на основе существующих. Начнем с визуализации.

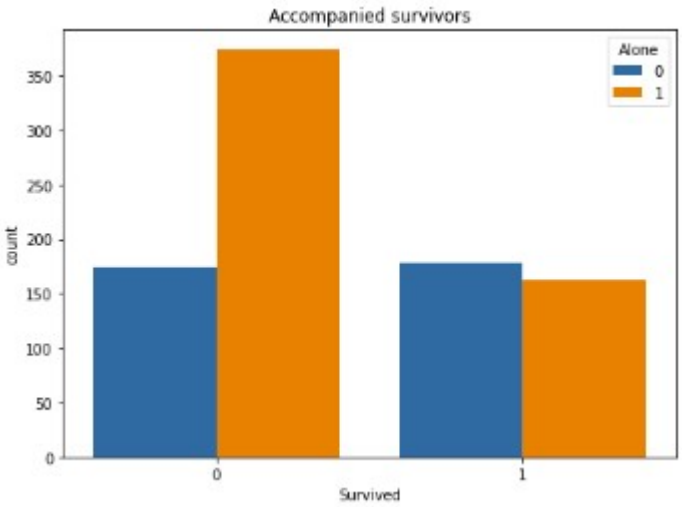


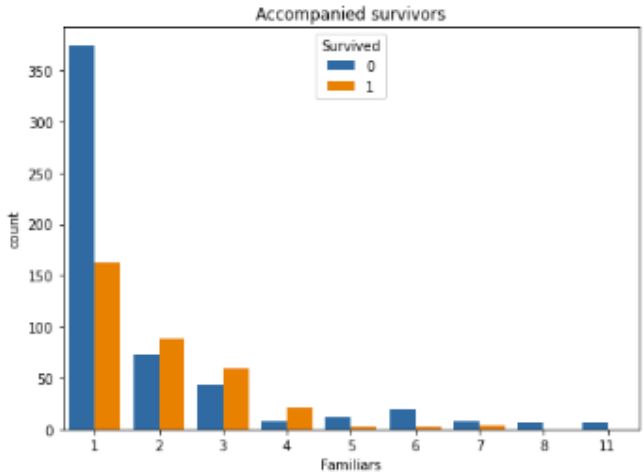
Как видно из графиков, большинство пассажиров путешествовали одни, так же как большинство были из 3-класса, что соответствует тому, что мы видели в сравнении кают и оплаты. Большинство из тех, кто не был приписан к какой-либо каюте, имели низкую оплату, и принадлежат к 3-классу. Основываясь, на этом, можно добавить бинарный признак, сообщающий о том ехал ли пассажир один и признак, содержащий количество родственников, которые ехали с пассажиром.

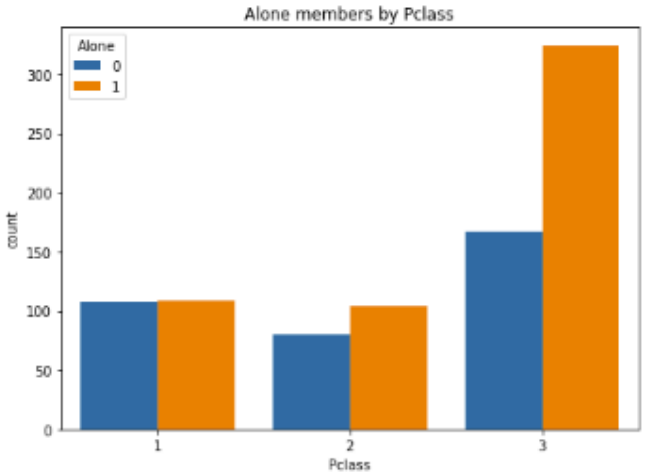




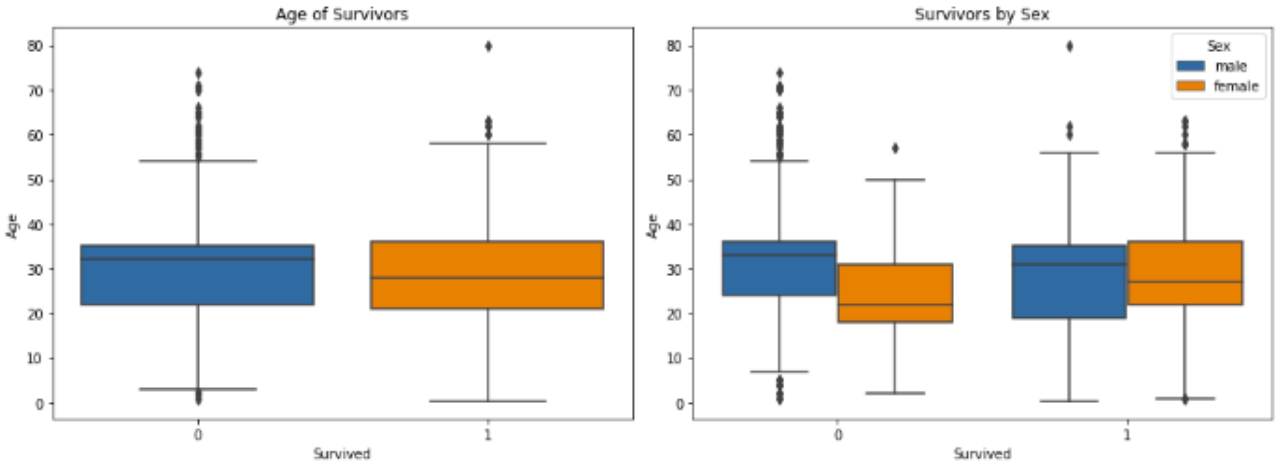


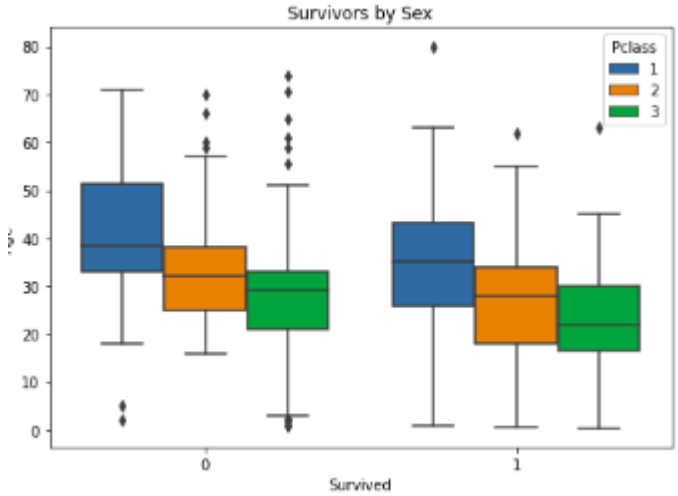


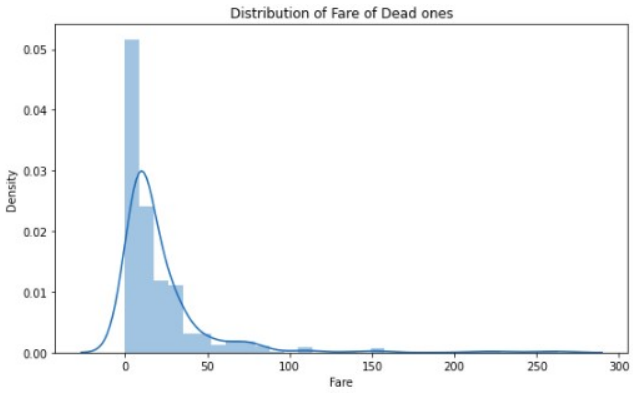


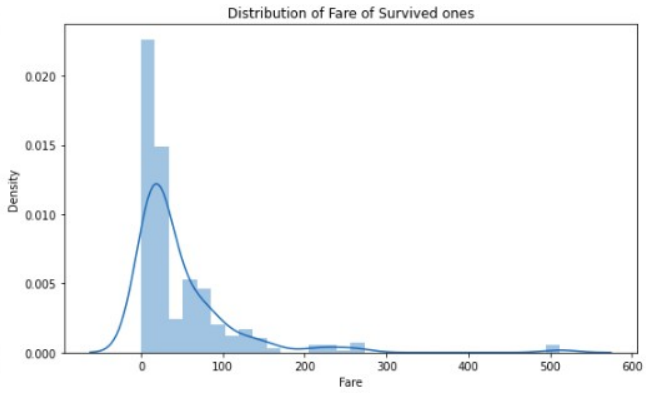


Как видно из визуализации, большинство пассажиров погибло, а среди выживших больше женщин чем мужчин, что очевидно, так как рабочие и персонал на судне в основном были мужчины. Так же, можно заметить, что большинство погибших — представители класса 3, что могло произойти из-за того, что их каюты находились в труднодоступных местах, и их спасали последними. Так же, большинство погибших были одиноки, что имеет смысл, так как представители третьего класса в основном путешествовали одни. Давайте теперь посмотрим на возраст и плату выживших.

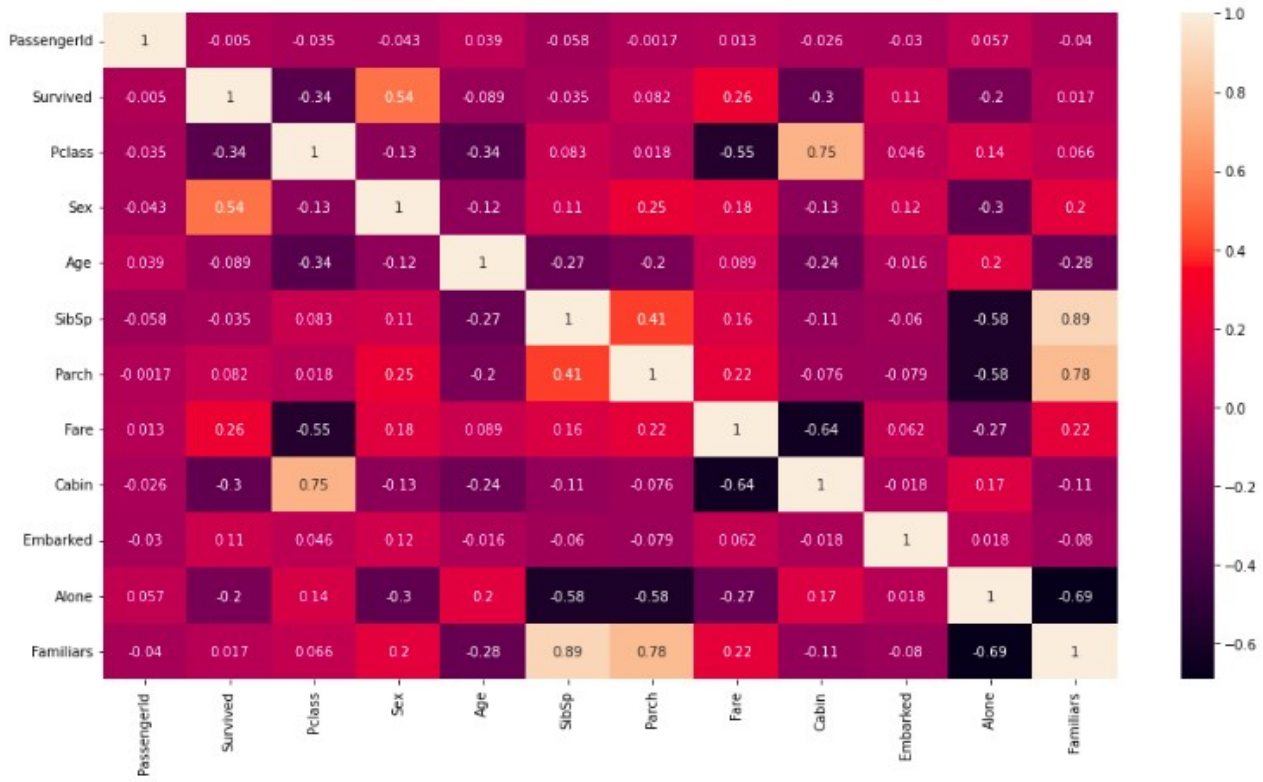








Наглядно можем убедиться, что оплата у тех, кто выжил была выше чем у тех, кто умер. Те кто выжил, были несколько моложе тех, кто умер. Так же, пассажиры 1-класса были старше остальных. Далее, заметна некоторая зависимость между признаками Fare и Survived. Рассмотрим корреляцию.

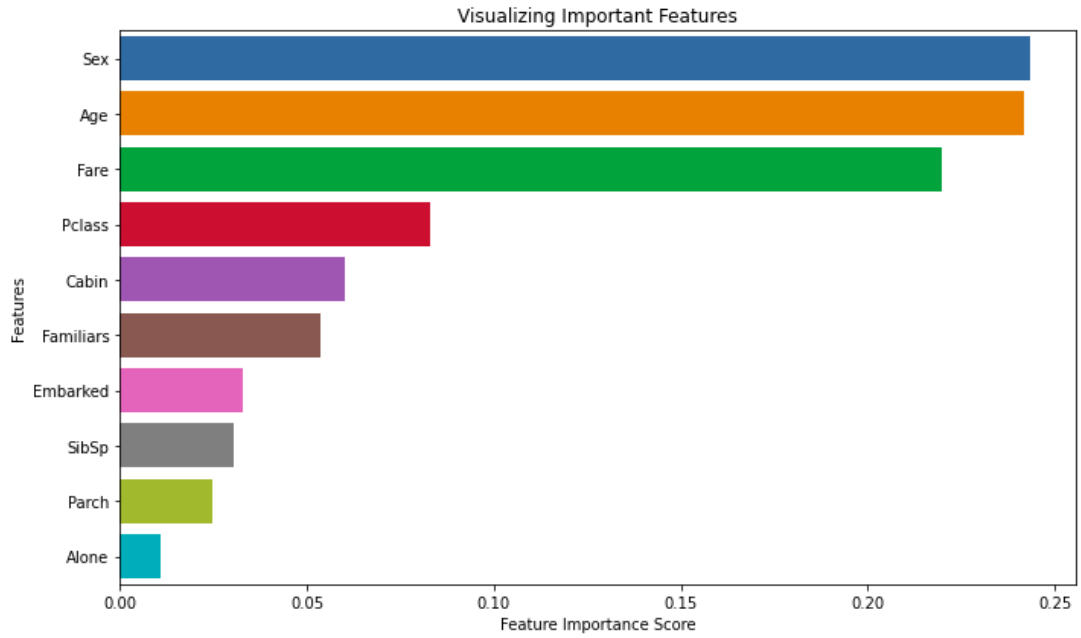


Теперь можем отбросить ненужные признаки как Name, Ticket, PassengerID, и перейти к построению модели.

Построение модели.

Первым делом, мы масштабируем наши данные, используя знакомый нам MinMaxScaler(). Модель строится с использованием RandomForestClassifier, который дает возможность взглянуть на значимость признаков.

```
Accuracy: 0.7597765363128491
```



На первом замере, мы получаем точность в 76%. Однако, если отбросить наименее важные признаки (SibSp, Parch, Alone), можно улучшить модель.

```
Accuracy: 0.7821229050279329
              precision    recall  f1-score   support

           0       0.73      0.92      0.81        92
           1       0.89      0.63      0.74        87

    accuracy                           0.78       179
   macro avg       0.81      0.78      0.78       179
weighted avg       0.80      0.78      0.78       179
```

Так же, еще одним способом улучшить уже готовую модель, является подбор параметров для модели. В коде, есть дополнительная информация о том, как каждый из параметром влияет на точность, но чтобы посмотреть на них вкупе, была использована функция GridSearchCV, которая проходится по всей сетке и выдает наилучший результат. По итогам ее выполнения, точность вырастает до 84% при параметрах: max_depth=10, max_features=4, n_estimators=20.

Сравнение с похожими алгоритмами.

Проведем сравнение с алгоритмами из того же «семейства»: решающее дерево, бэггинг над решающими деревьями, сверхслучайные деревья. При одинаковом числе деревьев (n_estimators=10), случайный лес показывает наилучший результат. Почему случайный лес превосходит первые две модели, однако не совсем очевидно, почему он точнее чем сверхслучайные деревья. Причина заключается в том, что последняя модель позволяет уменьшать дисперсию за счет случайно генерируемых порогов, однако это отражается на точности модели. Данный метод стоит использовать при сильном переобучении на случайном лесе или при градиентном бустинге.

```
[85] from sklearn.ensemble import BaggingClassifier
     from sklearn.tree import DecisionTreeClassifier
     dtree = DecisionTreeClassifier().fit(X_train, y_train)
     d_predict = dtree.predict(X_test)
     print(accuracy_score(d_predict, y_test))

     0.7877094972067039
```

```
[92] bc = BaggingClassifier(DecisionTreeClassifier())
     bc.fit(X_train, y_train)
     y_pred = bc.predict(X_test)
     print(accuracy_score(y_pred, y_test))

     0.7988826815642458
```

```
     from sklearn.ensemble import ExtraTreesClassifier
     etc = ExtraTreesClassifier(n_estimators=10)
     etc.fit(X_train, y_train)
     y_pred = etc.predict(X_test)
     print(accuracy_score(y_pred, y_test))

     0.8044692737430168
```

```
[90] rfc = RandomForestClassifier(n_estimators=10)
     rfc.fit(X_train, y_train)
     y_pred = rfc.predict(X_test)
     print(accuracy_score(y_pred, y_test))

     0.8268156424581006
```

Заключение.

В ходе работы было изучено, почему ансамбли показывают лучший результат, и по какой причине RandomForest является столь популярной моделью. Дополнительно, мне удалось использовать знания для построения модели на реальных данных и даже несколько раз ошибиться, что однозначно добавляет знаний в ходе проделанной работы.

Использованные ресурсы.

1. Decison Tree vs. Random Forest — Which Algorithm Shoud You Use. www.analytcisvidhya.com
2. Random Forest Algorithm with Python and Scikit-learn. stackabuse.com
3. Открытый курс машинного обучения. Тема 5. Композиции: бэггинг, случайный лес. habr.com
4. Деревья решений: общие принципы. loginom.ru
5. Random Forest on Titanic Dataset. medium.com
6. Titanic — Machine Learning from Disaster. www.kaggle.com