Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной математики и компьютерных наук

ОТЧЕТ ПО ПРОЕКТУ

по дисциплине «Введение в интеллектуальный анализ данных»

ПРЕДСКАЗАНИЕ ЦЕНЫ

Кострюков Никита Сергеевич

Романов Вячеслав Васильевич

Направление подготовки 02.04.02 Фундаментальная информатика и информационные технологии

Направленность (профиль) «Математика беспроводных сетей связи и интернета вещей»

Руководитель работы

д-р техн. наук, профессор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Замятин

*подпись*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г.

Авторы работы

студент группы № \_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.С. Кострюков

*подпись*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г.

студент группы № \_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Романов

*подпись*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г.

Томск – 2023

**СОДЕРЖАНИЕ**

1 Набор данных…………………………………………………………………………………..4

2 Предварительная обработка данных………………………………………………………….3

3 Корреляционный анализ………………………………………………………………………5

4 Прогнозирование цены………………………………………………………………………...7

4.1 Метод Random Forest……………………………………………………………………11

4.2 Метод XGBoost…………………………………………………………………………..16

4.3 Линейная регрессия……………………………………………………………………..18

4.4 Метод k-ближайших соседей…………………………………………………………...18

5 Вывод в Shiny…………………………………………………………………………………20

5.1 Метод Random Forest……………………………………………………………………22

5.2 Метод XGBoost…………………………………………………………………………..25

5.3 Линейная регрессия……………………………………………………………………..27

5.4 Метод k-ближайших соседей…………………………………………………………...29

5.5 Means of algorithms……………………………………………………….……………...32

ЗАКЛЮЧЕНИЕ…………………………………………………………………………………34

1. **Набор данных**

Выбранный для исследования цены набор данных содержит информацию о более чем 53 000 алмазах, включая их различные характеристики, такие как вес, цвет, прозрачность, огранка, глубина и другие. Набор данных взят из открытого источника и доступен на платформе Kaggle. Ссылка на набор данных: <https://www.kaggle.com/datasets/shivam2503/diamonds>. Исследований на этом наборе данных на сайте представлено достаточно, но подавляющее большинство исследовало на языке Python. Мы предпочли язык R, используемый в RStudio, так как знакомы с этим языком с курса эконометрики и статистического анализа данных.

Каждый алмаз в наборе данных представлен в виде отдельной строки, а каждый столбец соответствует различным характеристикам алмаза. Вот список всех столбцов и их описания:

* carat: вес алмаза в каратах;
* cut: качество огранки алмаза (Fair, Good, Very Good, Premium, Ideal);
* color: цвет алмаза (от J (худший) до D (лучший));
* clarity: прозрачность алмаза (от I1 (худший) до IF (лучший));
* depth: глубина алмаза в процентах;
* table: ширина верхней грани алмаза относительно самого широкого места в процентах;
* price: цена алмаза в долларах США;
* x: длина алмаза в мм;
* y: ширина алмаза в мм;
* z: высота алмаза в мм.

Набор данных может быть использован для различных задач, в том числе для построения моделей предсказания цен на алмазы на основе их характеристик, для анализа корреляции между различными параметрами алмазов и многого другого. Для исследования цены были выбраны четыре метода предсказания цены: метод случайного леса, бустинг, линейная регрессия и метод k-ближайших соседей.

Каждый метод был реализован в отдельном проекте.

1. **Предварительная обработка данных**

Перед тем как начать анализ данных, необходимо провести предварительную обработку данных. Это может включать в себя очистку данных, заполнение пропущенных значений, преобразование категориальных факторов в числовые, нормализацию данных и т.д.

Исходные данные загружаются из файла "diamonds.csv" с помощью функции read.csv.

Необходимо записать средние значения (или медианы) количественных столбцов, чтобы потом заменить выбросы или очень не логичные значения на средние. Например, для показателей размера "x", "y" и "z" значение 0 неадекватно.

Далее удаляем первый столбец - индексационный, а также удаляем дубликаты, то есть повторяющиеся строки. После удаления видим, что данных стало на 146 строк меньше.

Следующим этапом мы ищим выбросы для каждого количественного столбца с помощью коробок с усами. Все значения, выходящие за нижние пределы 1 и верхние пределы 3 квантилей заменяются на изначальные средние значения.

Также нам необходимо сохранить средние и среднеквадратичные отклонения количественных столбцов в отдельные таблицы для того, чтобы когда мы вводили новые данные, они сначала нормализовались, а потом вставлялись в загруженную модель, ведь наши модели обучены на нормализованных данных(количественных, кроме целевого фактора - цены).

Соответственно, нормализуем данные и приступаем к следующему этапу.

В итоге, предварительная обработка данных включает следующие шаги: выбор необходимых столбцов, удаление дубликатов, заполнение неадекватных значений средними, заполнение выбросов средними, нормализацию числовых данных.

В сравнении с прошлым проектом, предварительная обработка данных была гораздо проще: отсутствовала замена выбросов и замена неадекватных значений для параметров размера.

1. **Корреляционный анализ**

Корреляционный анализ данных позволяет исследовать зависимости между различными переменными.

Первая строка выбирает только те столбцы, которые будут использоваться для корреляционного анализа. В данной записи цена перенесена вперед для более удобного представления: "price", "carat", "cut\_numeric", "color\_numeric", "clarity\_numeric", "depth", "table", "x", "y" и "z".

Затем функция *cor()* используется для расчета корреляционной матрицы, которая показывает коэффициент корреляции Пирсона между каждой парой переменных (Рис.1). Коэффициент корреляции Пирсона измеряет линейную связь между двумя переменными, принимая значения от -1 до 1, где -1 указывает на полную обратную линейную связь, а 1 указывает на полную прямую линейную связь.

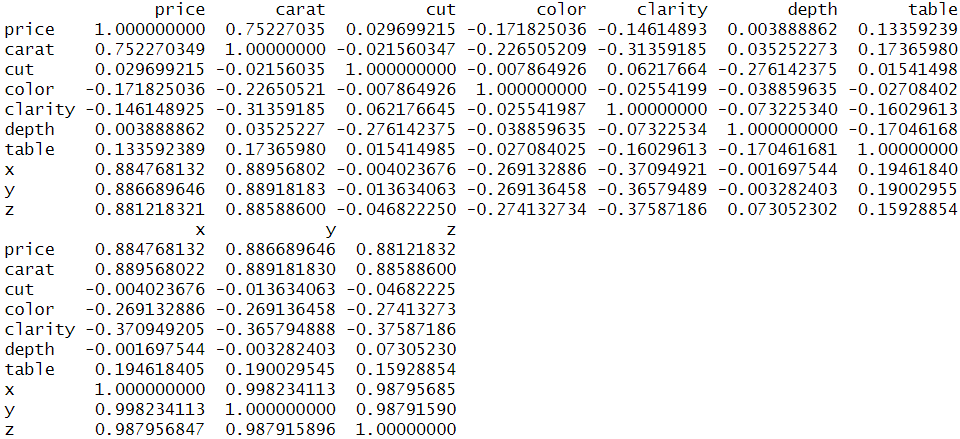


Рис.1 – Корреляционная матрица

Затем используется пакет "corrplot", чтобы построить график корреляций, где насыщенность цвета указывает на силу корреляции между переменными (Рис.2). Также можно видеть, что факторы "x", "y" и "z" сильно коррелируют с фактором "carat", поэтому их можно исключить из анализа, но этого не было сделано по одной причине, которая показалась мне важно. При дальнейшем использовании метода случайного леса было замечено, что при отсутствии параметров размера бриллианта, MSE принимало значение больше, чем при учтённых. Также далее увидим, что эти параметры важны, поэтому их оставляем в данных для обучения модели.

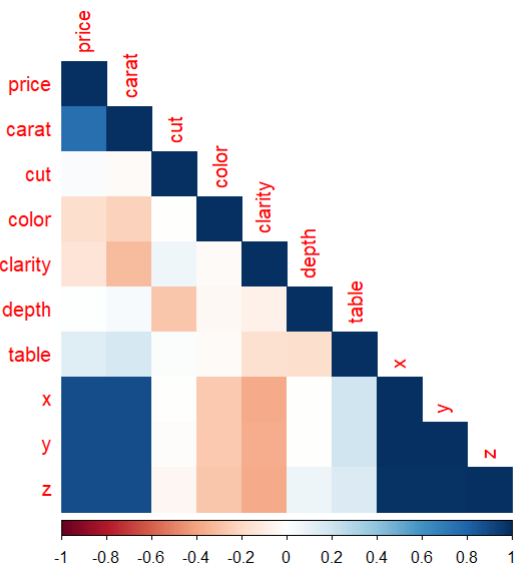


Рис.2 – График корреляций

В целом, анализ корреляции может помочь в выявлении зависимостей между переменными и может быть полезным инструментом для дальнейшего исследования данных. Конечно, его можно использовать в большем своём объёме и выявить с помощью него многое, но, как я посчитал на самом начале исследования, мне этих представлений будет достаточно, чтобы экспериментировать с прогнозом цены.

1. **Прогнозирование цены**

Сначала производим разделение исходных данных на обучающий и тестовый наборы в соотношении 7:3 случайным образом. Для повторяемости результатов устанавливается случайное зерно с помощью функции *set.seed()*.

Далее, с помощью пакета ggplot2 строятся исследовательские графики на обучающем наборе.

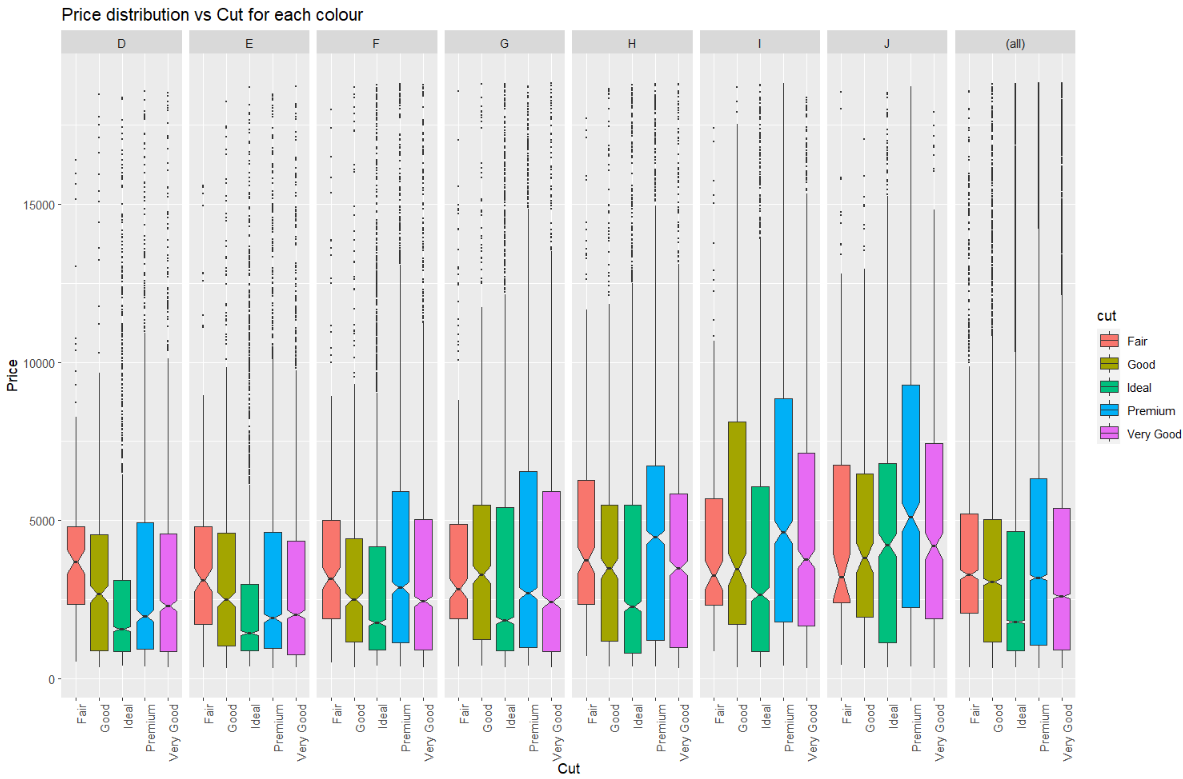


Рис.3 - Распределение цен в зависимости от качества огранки и цвета бриллиантов

По графику распределения цен в зависимости от качества огранки и цвета бриллиантов (Рис.3) видно, при ухудшении цвета бриллианта разброс цены увеличивается в среднем для всех состояний бриллианта. Также можно заметить, что бриллиантов идеального состояния и лучших цветов представлено в наборе данных меньше всего.

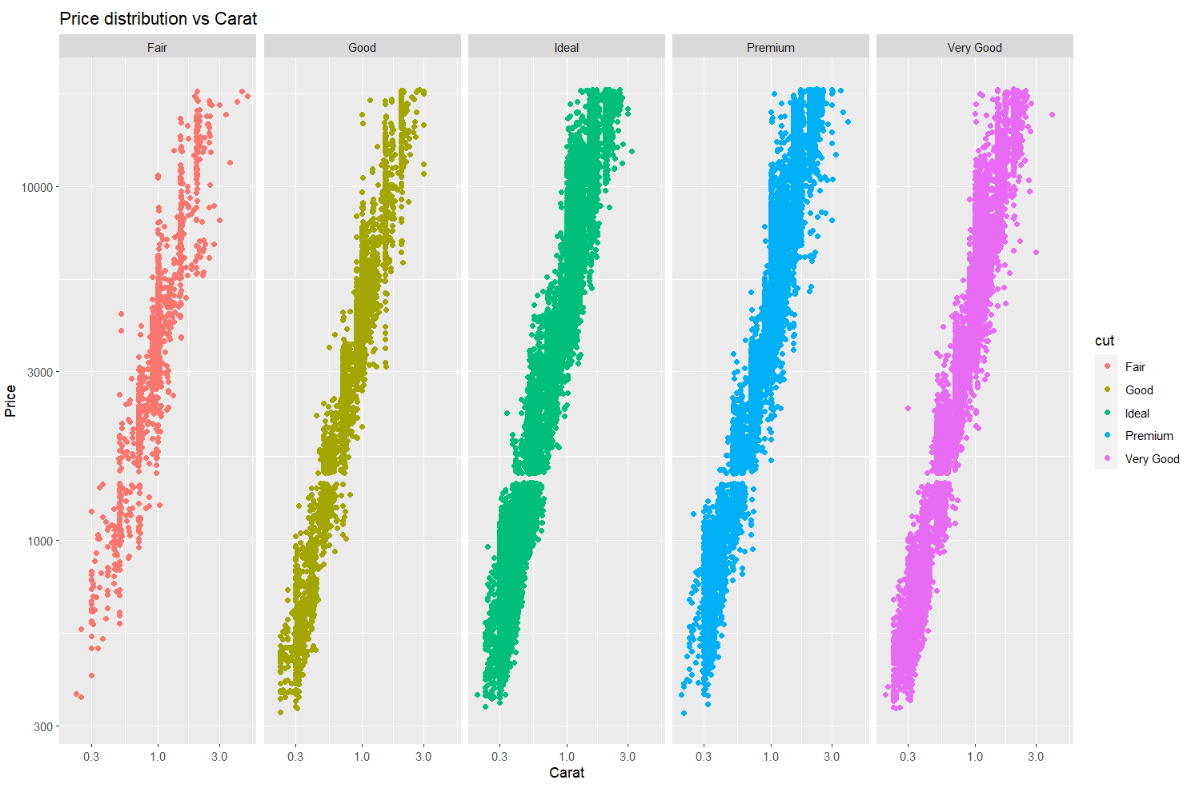


Рис.4 - Распределение цен в зависимости от веса бриллианта и качества огранки

На графике (Рис.4) можно увидеть, что в каждом распределении имеется “полоса” с отсутствующими значениями цены, то есть бриллианты, ценой примерно 1400$, отсутствуют в выбранном наборе данных. На всех подграфиках видно – при увеличении веса алмаза цена на него возрастает, что является логичным. Также заметно по насыщенности наблюдений, что для идеальной огранки зависимость цены и веса будет сильнее по сравнению с плохой огранкой.

 Рис.5 - Распределение цен в зависимости от веса бриллианта и качества огранки

На этом графике (Рис.5) отчетливо видно, как сильно может меняться цена при одном весе алмаза. Например, для веса равно единице цена может принимать значения от 2600$ до 17000$. Связано это с тем, что имеются и другие факторы, которые влияют на цену. Заметно, что при увеличении веса происходит рост цены всё быстрее. По общей тенденции бриллианты высшего качества находятся выше по цене, чем плохого качества.

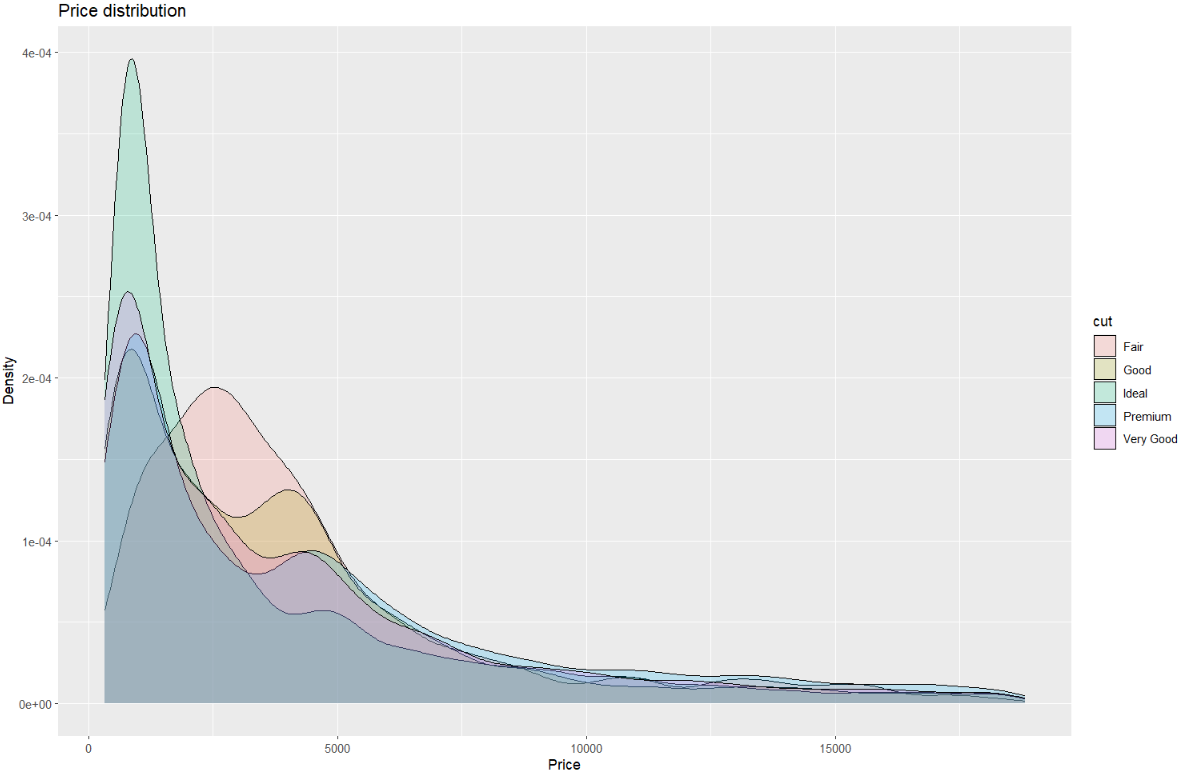


Рис.6 - Плотности распределения цен в зависимости от качества огранки

По графику (Рис.6) видно, что больше всего бриллиантов в диапащоне до 5000$, что уже было заметно на одном из предыдущих графиков. Также график позволяет увидеть, что чаще встречаются алмазы идеального качества в этом диапозане.

 Рис.7 - Распределение цен в зависимости от глубины огранки и качества огранки

На графике (Рис.7) можно запечатлеть, что при увеличении цены бриллианта глубина алмаза сходится к 62. При глубине больше 64 начинает страдать качество.

Построение графиков позволило сформировать общее представление о наборе выбранных данных, зависимости параметров между друг другом и, самое главное, зависимости цены от некоторых параметров.

**4.1 Метод Random Forest**

Метод Random Forest (случайный лес) - это алгоритм машинного обучения, который является комбинацией алгоритма бэггинга и деревьев решений.

Принцип работы метода Random Forest заключается в создании ансамбля деревьев решений. Для создания каждого дерева выбирается случайная выборка данных и случайный набор признаков. Затем строится дерево решений по этой выборке, которое даёт ответы на вопросы с помощью последовательности решающих правил. Каждое дерево строится независимо от других деревьев.

Для прогнозирования метод Random Forest использует голосование большинства. То есть каждое дерево выдаёт свой прогноз, и окончательный ответ определяется путём голосования большинства.

Сначала модель случайного леса обучается на обучающей выборке *train* с помощью функции *randomForest()*. Аргумент *price~*. указывает, что целевой признак - это *price*, а все остальные признаки в выборке будут использоваться в качестве предикторов.

В коде выполняется анализ важности признаков (feature importance) для модели случайного леса (Random Forest), которая обучается на данных об огранке бриллиантов. С помощью функции *importance()* вычисляется важность каждого признака в модели. Результат сохраняется в объект *imp*. С помощью функции *dimnames()* из объекта *imp* извлекается список имен признаков, который сохраняется в переменной *vars*. Затем создаётся новый объект *imp*, который представляет собой *data.frame* с двумя столбцами: *vars* (список имен признаков) и *imp* (важность признака в модели, представленная численно). Далее объект *imp* сортируется по убыванию важности признаков. Затем, с помощью функции *varImpPlot()* строится график важности признаков. На графике (Рис.8) отображается важность каждого признака в модели, отсортированного по убыванию важности.

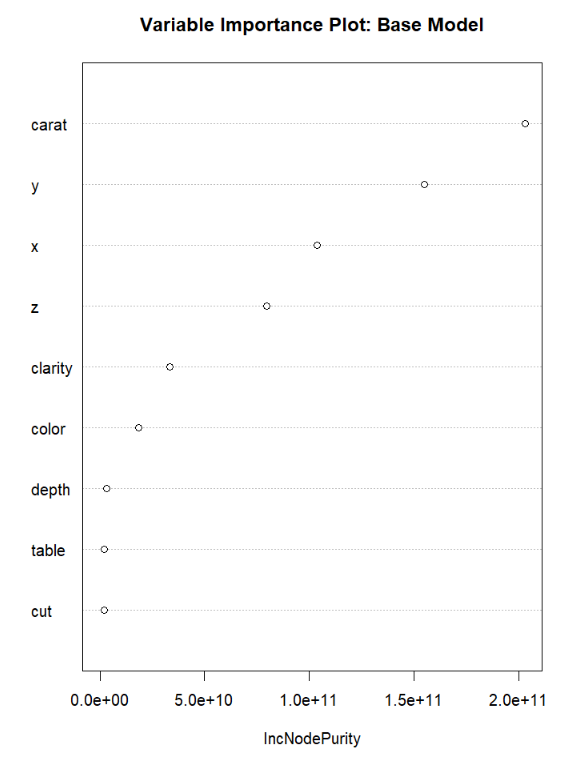


Рис.8 – Важность признаков

С помощью функции *plot()* строится график ошибки модели (Рис.9) по количеству деревьев (количество деревьев - это один из параметров, которые можно настроить при обучении модели случайного леса). График помогает оценить, как изменение количества деревьев влияет на ошибку модели.

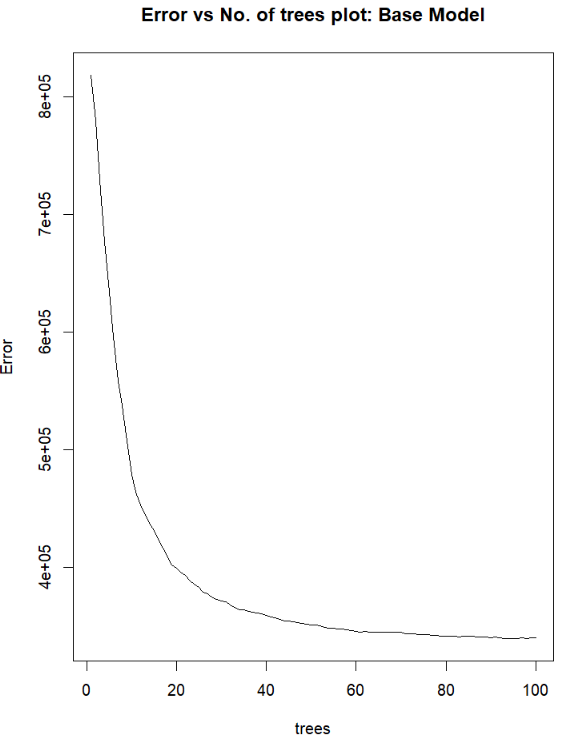


Рис.9 – Ошибки

На основании графика важности признаков можно сделать вывод, что признаки x, y, и z являются важными предикторами в модели, а последние три признака, а именно "cut", "depth" и "table", имеют низкую важность, поэтому их можно исключить из модели.

Далее мы создаем вектор *selected*, который содержит имена шести наиболее важных признаков, а также признак "price", который мы используем как целевую переменную для обучения модели. Затем мы обучаем новую модель, используя функцию *randomForest*. В отличие от предыдущей модели, мы указываем значения некоторых параметров. А именно, мы увеличиваем количество деревьев до 100, используя "ntree = 100", и устанавливаем "mtry = 3", что означает, что каждый раз при выборе случайных признаков для разделения дерева будет рассматриваться только три случайных признака. Также мы ограничиваем глубину каждого дерева до 10 уровней с помощью "maxdepth = 10" и минимальное количество наблюдений в листе до 5 с помощью "nodesize = 5". Эти параметры помогают контролировать переобучение и улучшить точность предсказаний. Также такие параметры заданы для того, что уменьшить RMSE.

Наконец, мы строим два графика (Рис.10): график важности признаков с помощью функции *varImpPlot* и график ошибки на обучающем наборе данных в зависимости от количества деревьев с помощью функции "plot". В обоих графиках используем красный цвет, чтобы отличить их от предыдущих графиков. График важности признаков показывает, что шесть выбранных признаков являются самыми важными для модели. График ошибки показывает, что ошибка уменьшается с увеличением количества деревьев и достигает плато приблизительно после 100 деревьев.

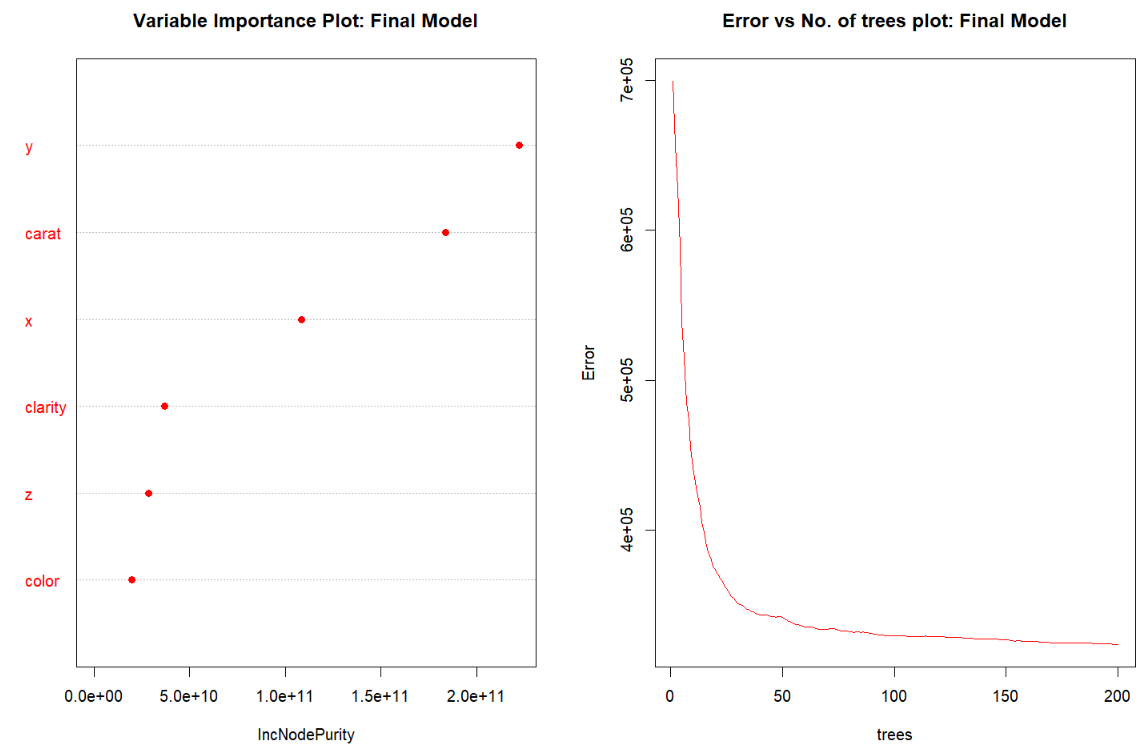


Рис.10 – Графики важности признаков и ошибок

Потом прогнозируются цены на тестовой выборке *test* с помощью функции *predict* с указанием модели *model1*. Затем создается датафрейм *result*, который содержит фактические цены (*actual*) и предсказанные цены (*predicted*). Далее, с помощью функции *paste*, выводятся некоторые статистические показатели, такие как средняя квадратическая ошибка (*mean(model1$mse)*) и корень из средней квадратической ошибки (*mean(sqrt(model1$mse))*). RMSE получился равным 540.370. Посмотрев результаты других на сайте Kaggle для моего набора данных, понял, что этот результат, скорее всего, является нормальным, так как у многих получались ошибки ещё больше.

Далее строится график (Рис.11) с помощью библиотеки ggplot2, который отображает фактические цены (*x=actual*) и предсказанные цены (*y=predicted*) с точками, цвет которых показывает ошибку в предсказаниях (*color=predicted-actual*). Через этот график можно сделать вывод, что модель хорошо предсказывает цены для простых бриллиантов с ценой до 3000 долларов. Однако, для бриллиантов с более высокой ценой, ошибка в предсказаниях возрастает.

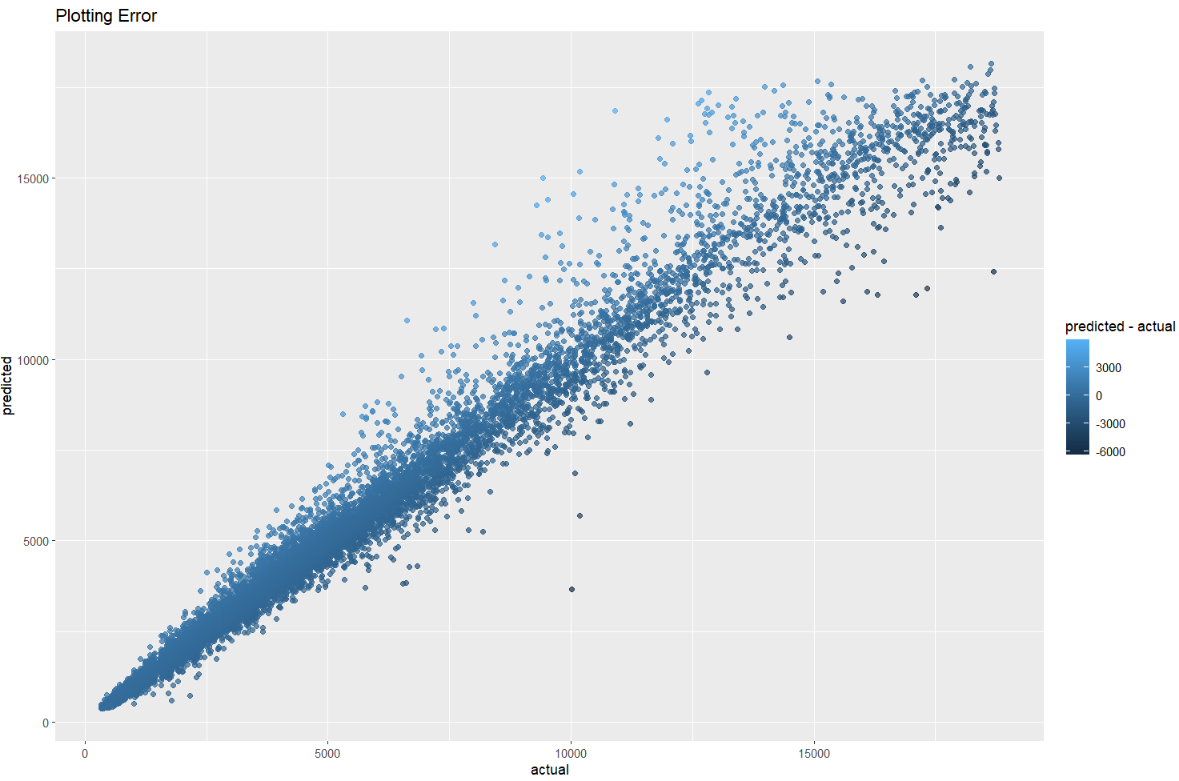


Рис.11 – График с ошибками в предсказании

Также представлен еще один график, который отображает распределение цен бриллиантов (Рис.12). Он показывает, что количество бриллиантов с ценой до 3000 долларов намного больше, что может объяснить наблюдаемую ошибку в предсказаниях модели для бриллиантов с более высокой ценой.

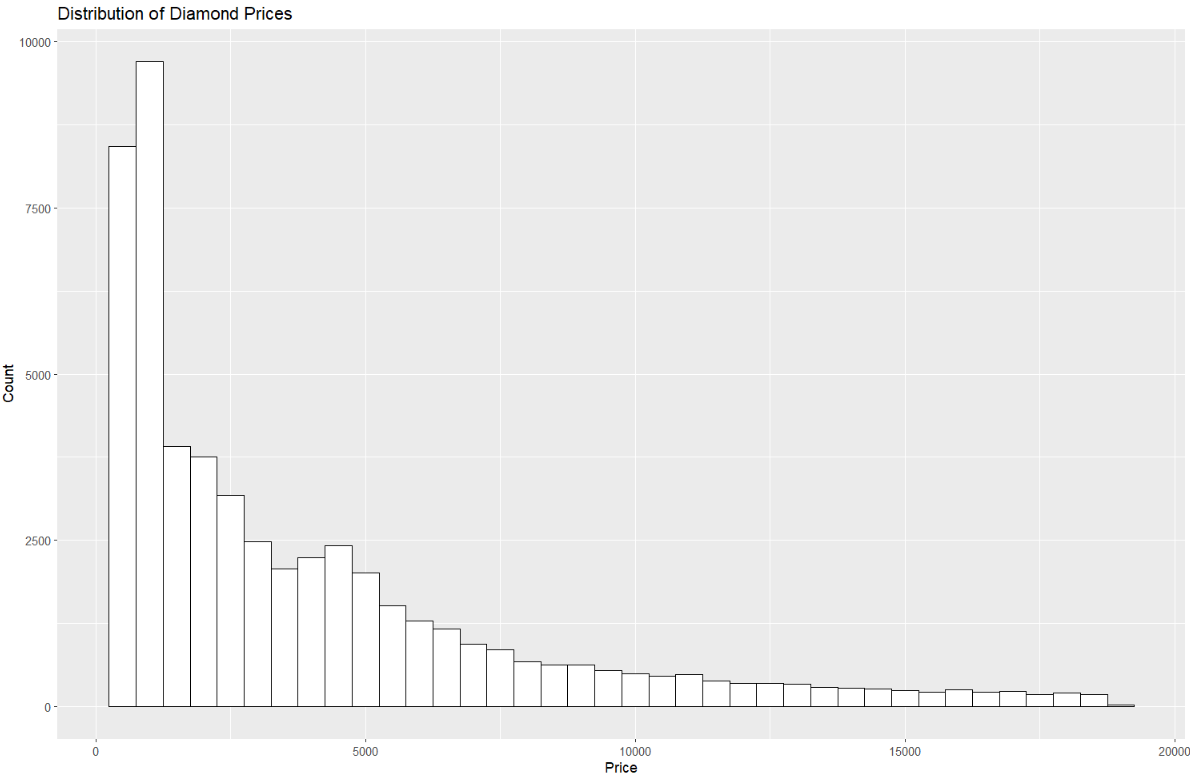


Рис.12 – Распределение цен

Наконец, создаётся новый датафрейм с параметрами нового бриллианта, у которого известны значения по параметрам. Затем, чтобы модель могла использовать параметры цвета и чистоты, они переводятся в факторы с помощью функции *factor()* и устанавливаются уровни факторов, соответствующие уровням в обучающем наборе *train*. Также, как упоминалось ранее, мы нормализуем данные для вставки в модель. Далее, для предсказания цены на бриллиант используется обученная модель *model* с помощью функции *predict(),* в которую передаются значения параметров нового бриллианта. Результатом является предсказанная цена на бриллиант.

* 1. **Метод XGBoost**

XGBoost (Extreme Gradient Boosting) - это библиотека машинного обучения, которая использует градиентный бустинг для построения моделей.

Основная идея XGBoost заключается в том, что он последовательно добавляет слабые модели в ансамбль, учитывая ошибки предыдущих моделей. В отличие от классического градиентного бустинга, XGBoost использует двойную регуляризацию, что позволяет сократить переобучение и повысить точность предсказания. Он штрафует признаки с малой важностью, поэтому может работать с набором данных, включающим большое количество признаков, даже если некоторые из них не являются значимыми.

Для начала я вновь разбил данные на обучающую и тестовую выборки в соотношении 7:3 с помощью функции sample. Далее происходит создание матриц *DMatrix* для обучающей и тестовой выборок. Для этого столбцы данных приводятся к числовому типу с помощью функции *lapply*, а затем создаются матрицы *dtrain* и *dtest* с помощью функции *xgb.DMatrix*. В параметрах функции указывается, что первый столбец данных является целевой переменной (*label*), которую нужно предсказать, поэтому его убирают из матрицы данных с помощью оператора -1. Далее создается список *params*, содержащий параметры модели *XGBoost*. В этом списке указывается целевая переменная *objective*, которая задается как "reg:squarederror" для регрессии, скорость обучения *eta*, максимальная глубина дерева *max\_depth*, минимальное количество обучающих примеров в листе дерева *min\_child\_weight*, доля обучающих примеров, используемых для обучения каждого дерева *subsample*, а также доля признаков, используемых для обучения каждого дерева *colsample\_bytree*. Для отслеживания процесса обучения создается список *watchlist*, который содержит матрицы *dtrain* и *dtest*, соответствующие обучающей и тестовой выборкам.

Далее, с помощью функции *xgb.train* производится обучение модели. В качестве параметров передаются *params*, содержащий список параметров модели, *data*, содержащий матрицу обучающей выборки *dtrain*, *nrounds*, максимальное количество итераций, *early\_stopping\_rounds*, количество итераций без улучшения модели, при которых обучение останавливается, *watchlist*, содержащий список данных для отслеживания обучения и *verbose*, уровень вывода информации об обучении модели.

Затем, модель используется для предсказания цены бриллианта на тестовой выборке с помощью функции *predict*. Вычисляется RMSE (корень из средней квадратичной ошибки) между ценами на бриллианты из тестовой выборки и соответствующими предсказанными ценами. Она равна 534.669.

Создаём новый набор данных new\_data, который содержит информацию о бриллианте. Затем в строках категориальные переменные cut, color и clarity преобразуются из символьного типа в числовой, используя функцию *as.factor* для преобразования в фактор, а затем *as.numeric* для преобразования фактора в числовой тип. Нормализуем данные для вставки в модель. Потом переименовываются столбцы в новом наборе данных, чтобы их названия совпадали с названиями столбцов в обучающем наборе данных. Затем создается матрица DMatrix для нового набора данных с помощью функции *xgb.DMatrix*, и модель используется для предсказания цены с помощью функции *predict*.

* 1. **Линейная регрессия**

Линейная регрессия - это метод машинного обучения, используемый для моделирования отношений между зависимой переменной (целевой) и одной или несколькими независимыми переменными (признаками). Основная идея линейной регрессии заключается в поиске линейной зависимости, которая наилучшим образом описывает данные.

Для этой модели берутся уже очищенные и подготовленные данные для обучения. Сначала мы задаем категориальным переменным факторный тип данных. Далее создается модель, которая зависит от всех факторов, так как они все являются для этой модели значимыми.

Задаются входящие данные для этой модели и выводится результат.

* 1. **Метод k-ближайших соседей**

Метод k-ближайших соседей (KNN) представляет собой простой и эффективный алгоритм машинного обучения, который используется как для классификации, так и для регрессии. Основная идея заключается в том, что объекты с похожими признаками обычно находятся близко друг к другу в пространстве признаков и имеют схожие значения целевой переменной.

В случае классификации, для нового объекта определяется принадлежность к определенному классу на основе меток его ближайших соседей. Принимается метка класса, которая является наиболее распространенной среди K ближайших соседей.

В случае регрессии, для нового объекта вычисляется среднее значение целевой переменной среди его ближайших соседей.

Определение числа соседей (K) и выбор метрики расстояния между объектами - ключевые параметры, которые могут влиять на результаты алгоритма. Кроме того, KNN является ленивым алгоритмом (lazy learning), что означает, что он откладывает процесс обучения до момента предсказания. Это было замечено нами в момент сохранения модели. Это делает алгоритм гибким, но также требует вычислительных ресурсов при предсказании, что занимает память.

Одним из недостатков KNN является его чувствительность к выбросам в данных и неэффективность в случае большого объема данных. Тем не менее, KNN остается привлекательным методом в ситуациях, где данных немного или когда структура данных сложно описать явными моделями.

В этом методе используются все тоже: очищенные данные, нормализования данных, кодирование категориальным переменных факторными.

Для предсказания цены используется функция knn.

1. **Вывод в Shiny**

Стоит отметить, что сайт работает как на компьютере, так и на телефоне. Для реализации на телефоне потребовалось дополнительное время.

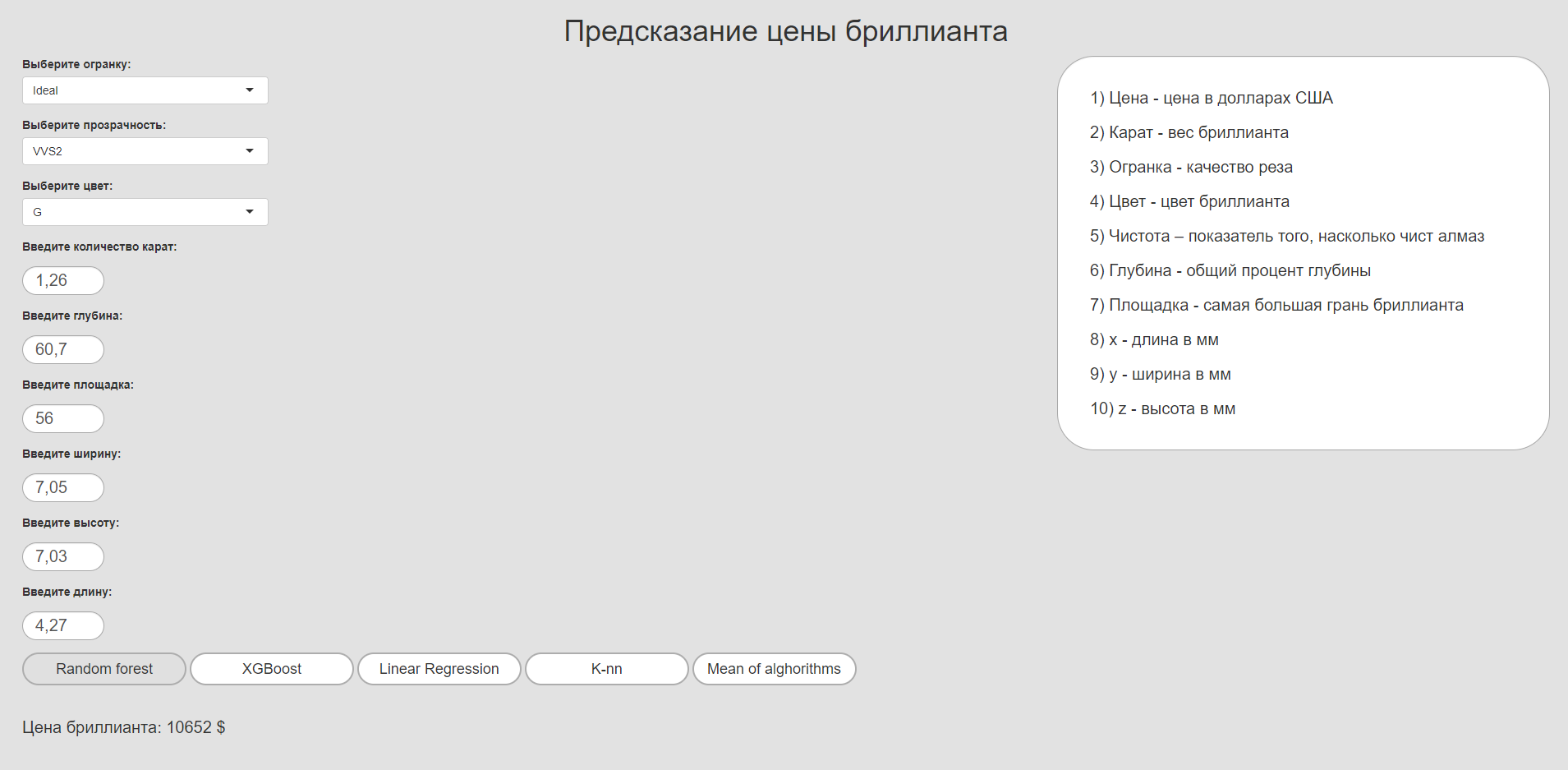
****

Рисунок 12 - Интерфейс сайта №1

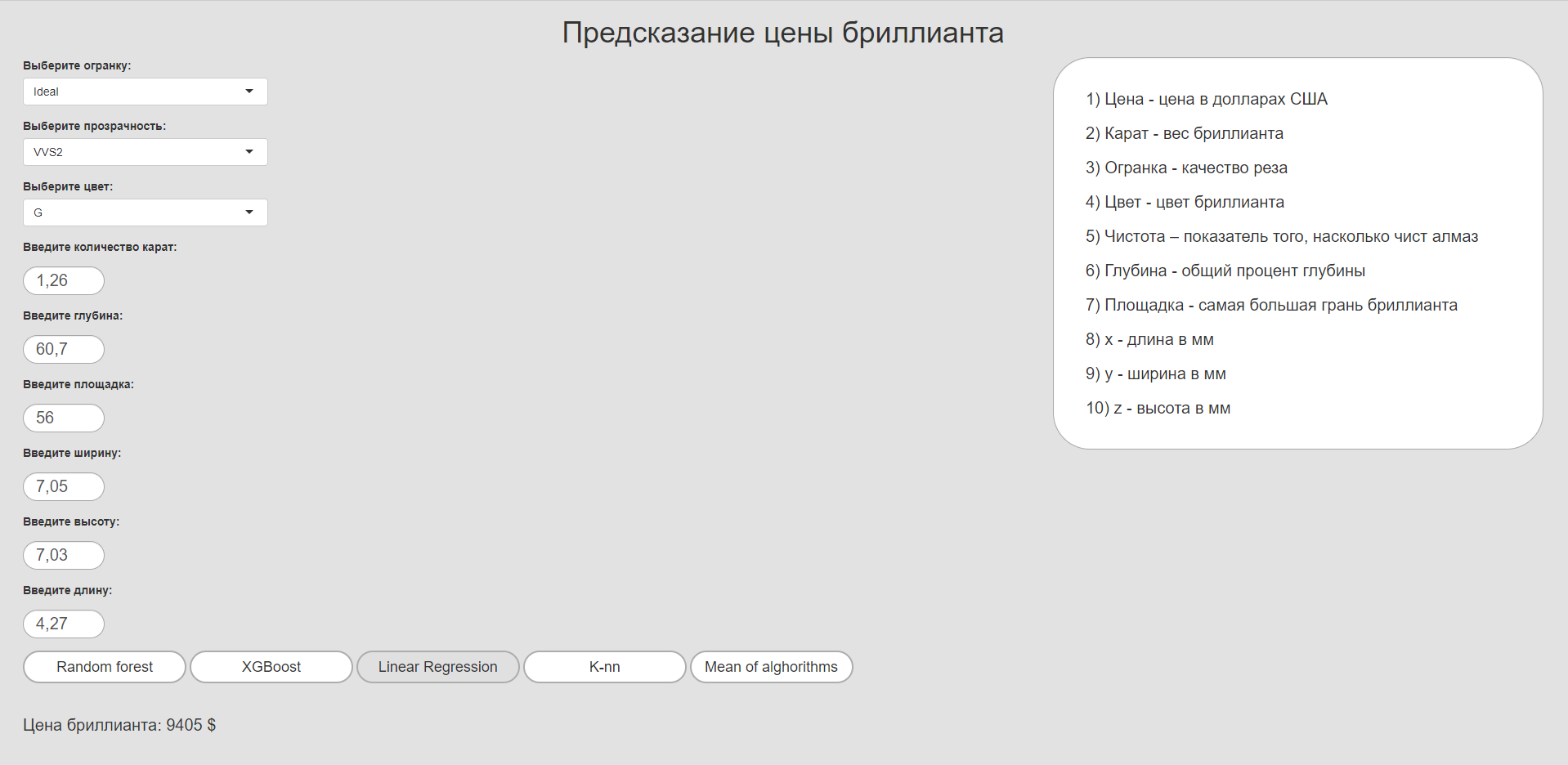


Рисунок 13 - Интерфейс сайта №2

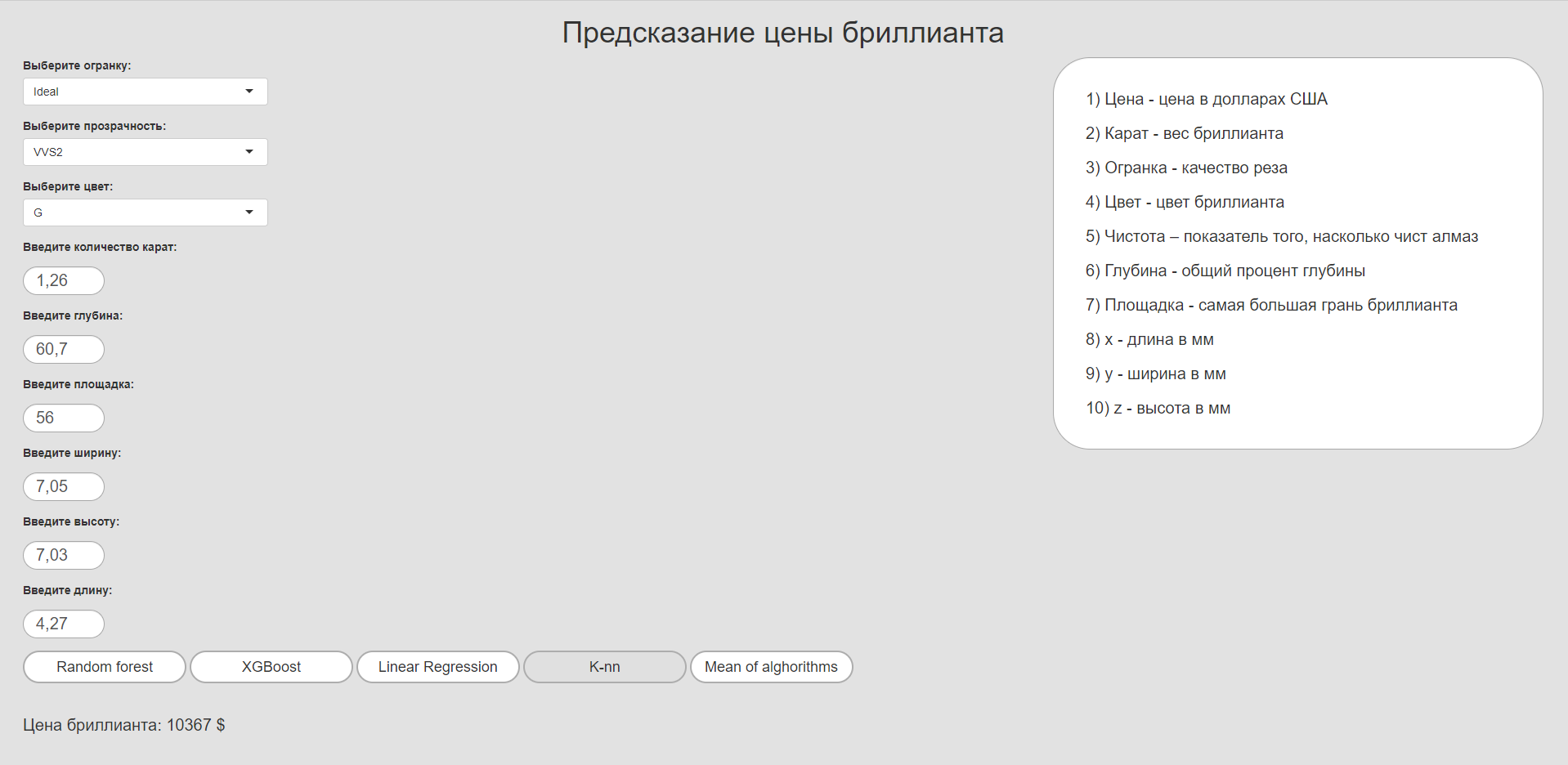


Рисунок 14 - Интерфейс сайта №3

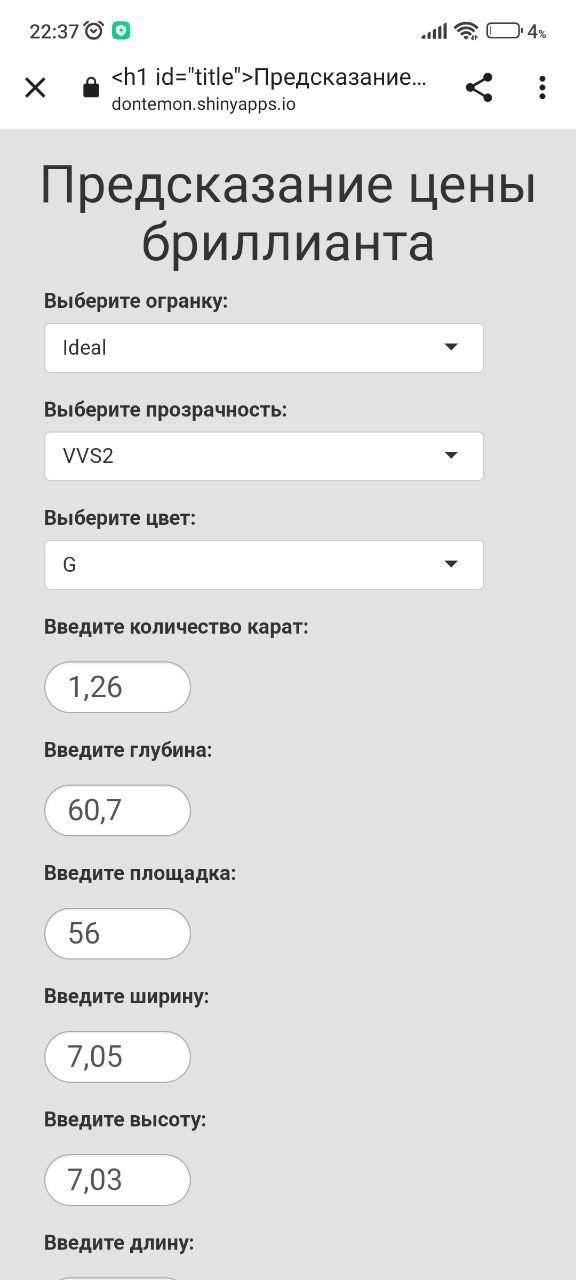
****

Рисунок 15 - Интерфейс сайта №4

С помощью кнопки Mean of algorithms можно узнать среднюю цену бриллианта по четырем моделям.

**5.1 Random Forest**

Первым делом нам нужно сказать и подключить используемые библиотеки:  
 1) caret предоставляет удобный интерфейс для обучения множества моделей машинного обучения. Он упрощает процесс выбора моделей, подбора гиперпараметров и оценки их производительности.

2) shiny — это фреймворк для создания интерактивных веб-приложений на R. Он позволяет легко создавать веб-приложения с использованием R-кода и встроенных элементов управления.

3) corrplot предоставляет инструменты для визуализации матриц корреляции. Он часто используется для анализа взаимосвязей между переменными.

4) ggplot2 — это библиотека для создания графиков с использованием грамматики графики. Она позволяет создавать сложные и красочные графики с учетом принципов "грамматики графики" (grammar of graphics).

5) randomForest предоставляет функционал для построения случайных лесов, мощного метода ансамблевого машинного обучения.

6) xgboost (Extreme Gradient Boosting) — это библиотека для реализации градиентного бустинга. Она часто используется для построения моделей, предсказывающих сложные зависимости в данных.

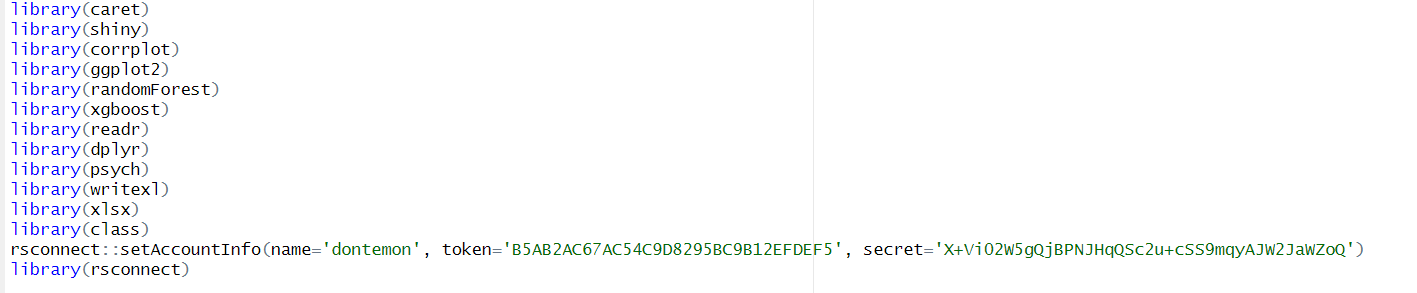
7) readr обеспечивает быстрые функции чтения данных в формате "текстовой таблицы" (CSV, TSV и другие). Он является частью пакета tidyverse.

8) dplyr — это часть tidyverse и предоставляет удобные функции для манипулирования и агрегирования данных. Он используется для эффективного фильтрования, сортировки и обобщения данных.

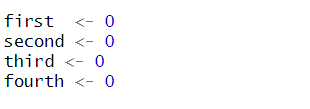
9) psych содержит функции для статистического анализа психологических данных. Он предоставляет методы для описательной статистики, факторного анализа и других процедур.

10) writexl предоставляет функции для записи данных в формате Excel. Он удобен для создания Excel-файлов из R.

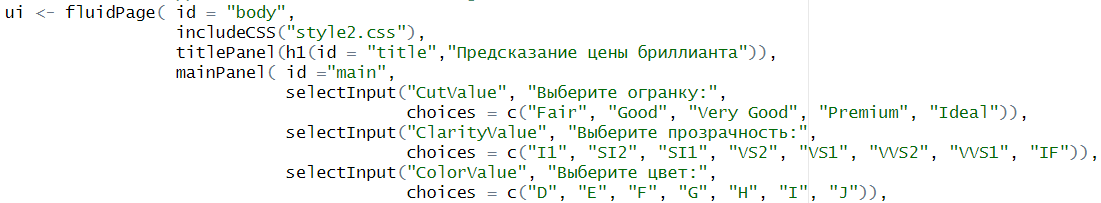
11) xlsx — это библиотека для чтения и записи файлов Excel. Она предоставляет функции для работы с Excel-файлами.

12) class — это базовый пакет R, который предоставляет функцию knn для реализации метода k-ближайших соседей. Этот метод часто используется в задачах классификации.

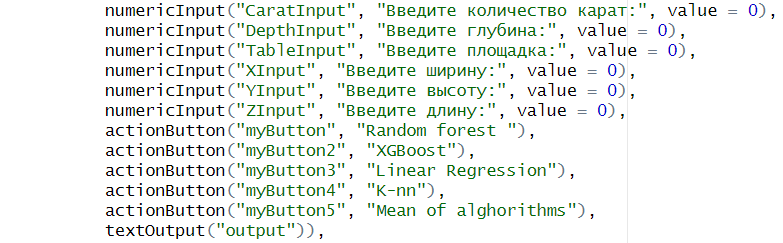
Объявляем глобальные переменные.



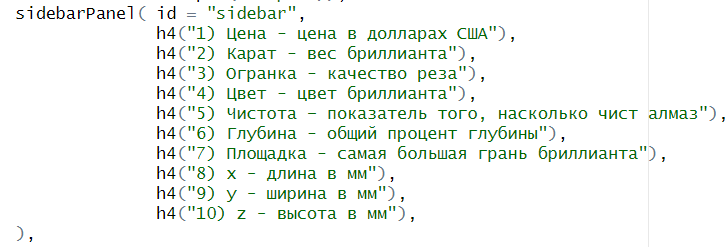
Далее создаем пользовательский интерфейс. На скриншоте ниже представлена его первая часть, на которой реализованы подключения к таблице стилей style2.css, задан заголовок странице и прописаны три выпадающих списка.



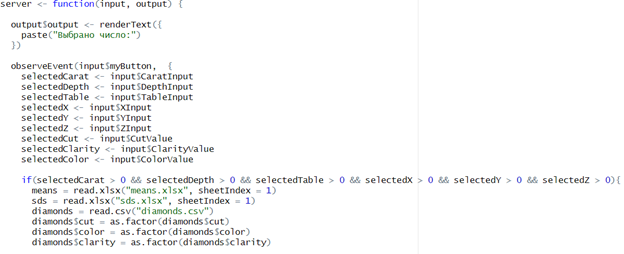
На скриншоте ниже, представлена вторая часть, на которой прописаны оставшиеся поля для ввода параметров, а также пять кнопок, нажимая которые, будет выполняться соответствующий алгоритм.

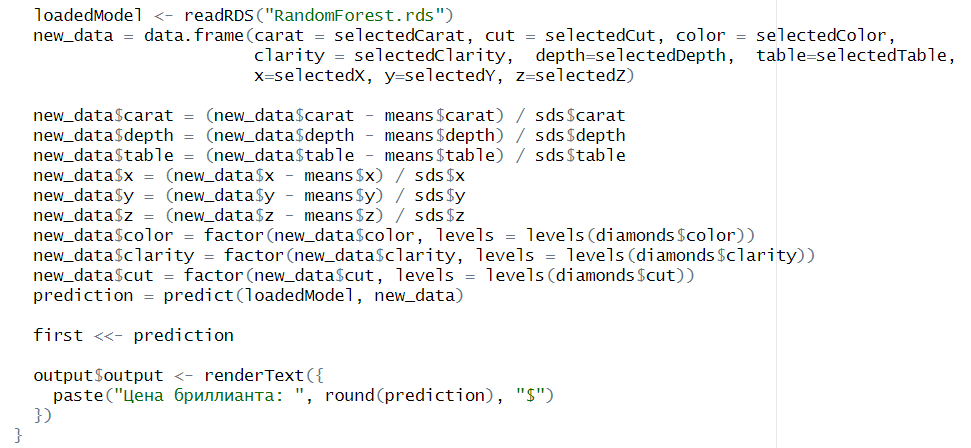


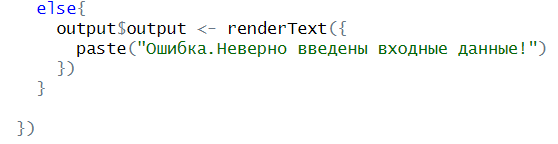
Третья часть представленная ниже является глоссарием для пользователя.



Далее мы прописываем события, которые происходят при нажатии на каждую из кнопок, для начала мы запоминаем значения, которые вписал пользователь, далее проверяем правильность ввода, и если все значения больше нуля, то переписываем из EXCEL файлов среднее и среднеквадратичное отклонение по каждому из факторов и считываем дата-сет, после чего изменяем формат столбцов Огранки, Цвета и Чистоты на факторный.

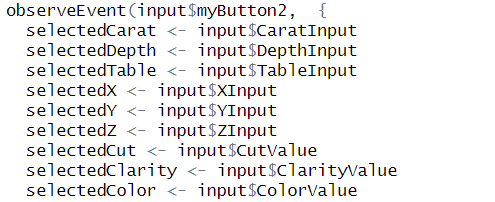


Далее в первом случае мы загружаем модель randomForest и создаем новый элемент со значениями, которые ввёл пользователь. После чего нормализуем новые данные с использованием средних и среднеквадратичных отклонений из таблицы diamonds и предсказываем цену бриллианта. После сохраняем предсказанную цену в глобальную переменную first и выводим её на экран.  


Если же среди введенных значений присутствует хотя бы одно меньше либо равно нулю, то выводится сообщение о некорректности введенных данных.  


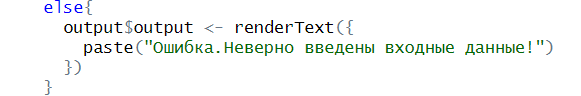
**5.2 XGBoost**

Получение данных из входных полей (input): Значения введенных пользователем данных из различных полей формы (CaratInput, DepthInput, TableInput, XInput, YInput, ZInput, CutValue, ClarityValue, ColorValue) сохраняются в соответствующих переменных (selectedCarat, selectedDepth, и так далее).

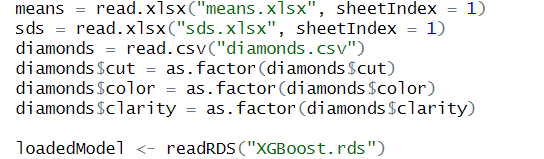


Проверка условия if: Перед выполнением дальнейших действий проверяется условие, что все введенные данные положительны. Если хотя бы одно из значений меньше или равно 0, выводится сообщение об ошибке.

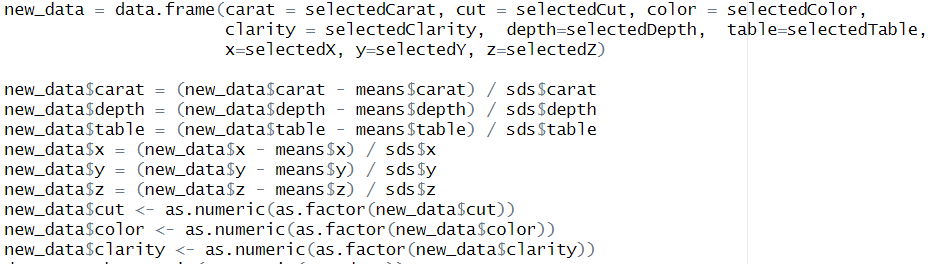




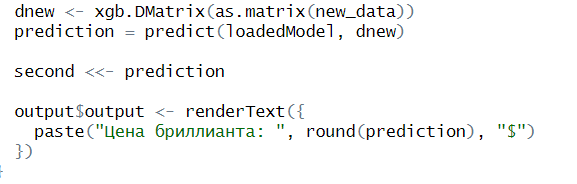
Чтение данных и модели: Если введенные данные корректны, читаются данные из файлов diamonds.csv, means.xlsx и sds.xlsx. Также читается модель XGBoost из файла XGBoost.rds.



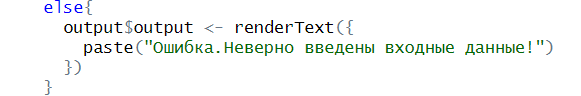
Подготовка новых данных для предсказания: Создается новый dataframe new\_data, в который записываются введенные пользователем данные. Затем выполняется стандартизация признаков на основе средних значений и среднеквадратических отклонений.



Предсказание с использованием загруженной модели: После подготовки новых данных они преобразуются в объект xgb.DMatrix, и используется ранее загруженная модель XGBoost для предсказания цены бриллианта. Сохранение предсказания в глобальную переменную second: Результат предсказания сохраняется в глобальную переменную second с использованием оператора <<-.

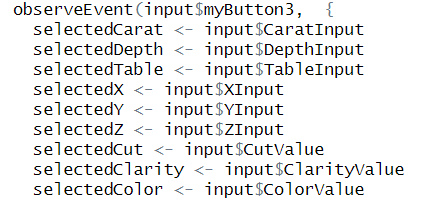


Вывод результата в интерфейсе Shiny: В зависимости от успешности предсказания, приложение выводит результат в текстовом формате. Если данные введены неверно, выводится сообщение об ошибке.



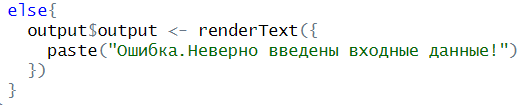
**5.3 Linear Regression**

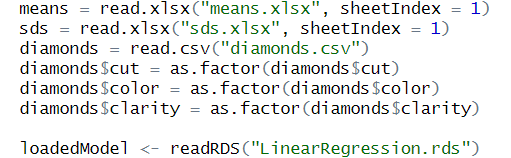
Получение данных из входных полей (input): Значения введенных пользователем данных из различных полей формы (CaratInput, DepthInput, TableInput, XInput, YInput, ZInput, CutValue, ClarityValue, ColorValue) сохраняются в соответствующих переменных (selectedCarat, selectedDepth, и так далее).



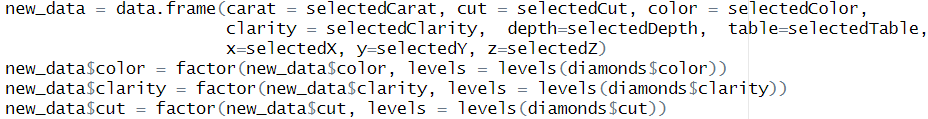
Проверка условия if: Перед выполнением дальнейших действий проверяется условие, что все введенные данные положительны. Если хотя бы одно из значений меньше или равно 0, выводится сообщение об ошибке.



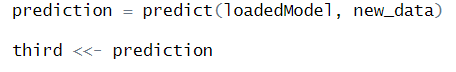
  
 Чтение данных и модели: Если введенные данные корректны, читаются данные из файлов diamonds.csv, means.xlsx и sds.xlsx. Также читается модель линейной регрессии из файла LinearRegression.rds.



Подготовка новых данных для предсказания: Создается новый dataframe new\_data, в который записываются введенные пользователем данные. Факторы (color, clarity, cut) приводятся к факторному формату с использованием уровней из diamonds.

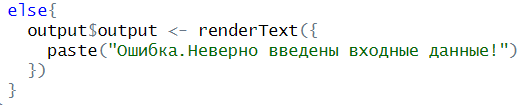


Предсказание с использованием загруженной модели: После подготовки новых данных используется ранее загруженная модель линейной регрессии для предсказания цены бриллианта. Сохранение предсказания в глобальную переменную third: Результат предсказания сохраняется в глобальную переменную third с использованием оператора <<-.

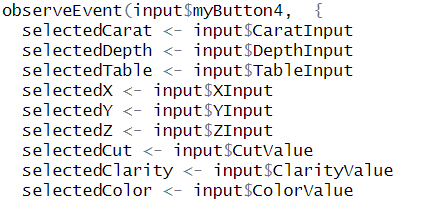


Вывод результата в интерфейсе Shiny: В зависимости от успешности предсказания, приложение выводит результат в текстовом формате. Если данные введены неверно, выводится сообщение об ошибке.

****

****

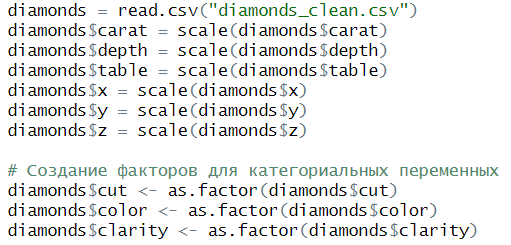
**5.4 KNN** Получение данных из входных полей (input): Значения введенных пользователем данных из различных полей формы (CaratInput, DepthInput, TableInput, XInput, YInput, ZInput, CutValue, ClarityValue, ColorValue) сохраняются в соответствующих переменных (selectedCarat, selectedDepth, и так далее).



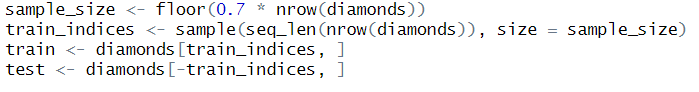
Проверка условия if: Перед выполнением дальнейших действий проверяется условие, что все введенные данные положительны. Если хотя бы одно из значений меньше или равно 0, выводится сообщение об ошибке.



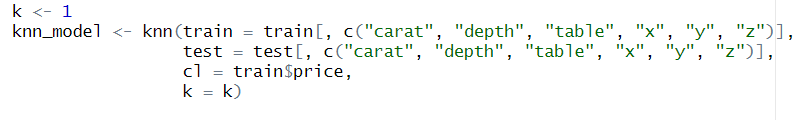
Чтение данных и подготовка для модели k ближайших соседей (knn): Если введенные данные корректны, читаются данные из файла diamonds\_clean.csv. Затем проводится стандартизация (z-преобразование) признаков. Факторы (cut, color, clarity) приводятся к факторному формату.



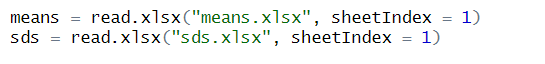
Определение обучающего и тестового наборов: Данные разделяются на обучающий и тестовый наборы. В данном случае, 70% данных используются для обучения, и 30% для тестирования.



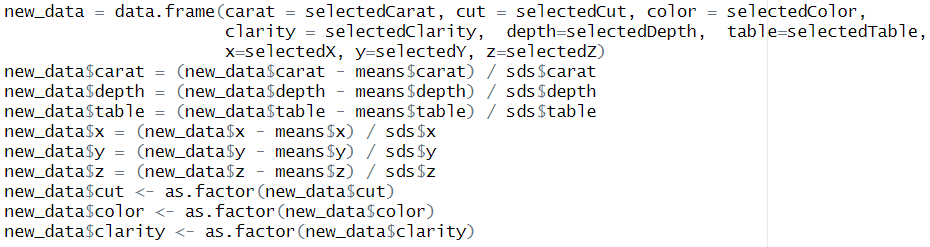
Построение модели k ближайших соседей (knn): С использованием функции knn строится модель k ближайших соседей на обучающем наборе.



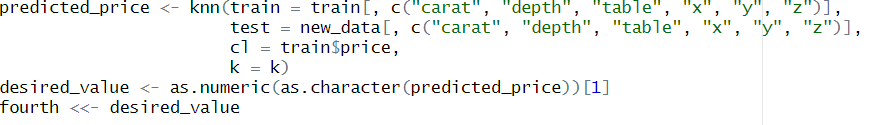
Чтение данных для нормализации: Читаются данные из файлов means.xlsx и sds.xlsx, которые будут использоваться для нормализации введенных пользователем данных.



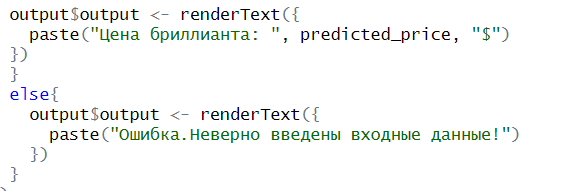
Нормализация новых данных для предсказания: Новые введенные данные нормализуются с использованием средних значений и среднеквадратических отклонений из файлов means.xlsx и sds.xlsx.



Предсказание с использованием модели k ближайших соседей (knn): После нормализации новых данных, с использованием ранее построенной модели k ближайших соседей, выполняется предсказание цены бриллианта. Сохранение предсказания в глобальную переменную fourth: Результат предсказания сохраняется в глобальную переменную fourth с использованием оператора <<-

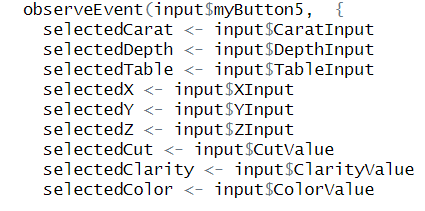
.

Вывод результата в интерфейсе Shiny: В зависимости от успешности предсказания, приложение выводит результат в текстовом формате. Если данные введены неверно, выводится сообщение об ошибке.



**5.5 Mean of algorithms**

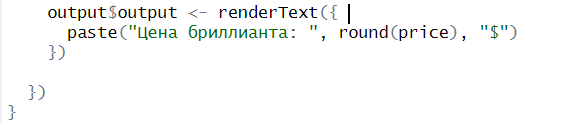
Получение данных из входных полей (input): Значения введенных пользователем данных из различных полей формы (CaratInput, DepthInput, TableInput, XInput, YInput, ZInput, CutValue, ClarityValue, ColorValue) сохраняются в соответствующих переменных (selectedCarat, selectedDepth, и так далее).



Вычисление средней цены (price): Суммируются значения из глобальных переменных first, second, third и fourth, и затем делится на 4 для вычисления средней цены бриллианта.



Вывод результата в интерфейсе Shiny: Результат (средняя цена) выводится в текстовом формате с использованием функции renderText в блоке output$output <- renderText({...}).



Таким образом, при нажатии на кнопку myButton5, код выполняет вычисление средней цены на основе ранее предсказанных значений (first, second, third, fourth) и выводит эту среднюю цену в интерфейсе Shiny.



**ЗАКЛЮЧЕНИЕ**

В данной работе были использованы методы Random Forest, XGBoost, Линейная регрессия и метод k-ближайших соседей для предсказания цены на алмазы на основе других факторов. Настроен пользовательский интерфейс с помощью Shiny. По итогу выполнения, можно сделать вывод, что для этого набора данных предсказанная цена будет точнее тогда, когда бриллиант не будет таким дорогим в плане качества, размеров и т.д.

Самая точная модель из выбранных - Random Forest, так как его RMSE наименьший.