МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Кафедра ВТ

Расчетно-графическая работа

По дисциплине

«Представление знаний в системах искусственного интеллекта»

# Разработка web-интерфейса для моделей прогнозирования цены на алмазы

Факультет: АВТФ

Группа: АПИМ-25

Выполнил: Бадагов А.В. Клименко К.В.

Преподаватель: Яковина И. Н.

Отметка о защите:

Новосибирск 2025

**Оглавление**

## Цель

Знакомство элементами технологии создания баз знаний интеллектуальных систем.

## Задачи

- Освоение способа сохранения и загрузки обученных моделей машинного обучения.
- Разработка интерфейса доступа к сохраненным моделям.
- Организация произвольных запросов с использованием разработанного интерфейса.

## Используемые программные инструменты

- Язык программирования: python.
- Реализация алгоритмов машинного обучения: tensorflow, sklearn.
- Сериализация/десериализация: json, pickle, tensorflow.
- Web-интерфейс: streamlit.

## Ход работы

### 1 Подготовка датасета

Проведём анализ и проверку дата сета рисунок 1.1 на наличие пропусков см.рисунок 1.2.

```
Размер таблицы (47327, 10)
   carat  cut  color  clarity  depth  table  price     x     y     z
0   0.23    5      6        2   61.5   55.0  326.0  3.95  3.98  2.43
1   0.21    4      6        3   59.8   61.0  326.0  3.89  3.84  2.31
2   0.29    4      2        4   62.4   58.0  334.0  4.20  4.23  2.63
3   0.31    2      1        2   63.3   58.0  335.0  4.34  4.35  2.75
4   0.24    3      1        6   62.8   57.0  336.0  3.94  3.96  2.48
```

Рисунок 1.1 – Часть исходного датасета

```
Размер таблицы (47327, 10)
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

Рисунок 1.2 – Количество пустых значений в исходных данных

```
['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z']
['price']
Размер таблицы (47327, 9)
Размер таблицы (47327, 1)
```

Рисунок 1.3 – Объявление названий целевых (target) и независимой (features)

### 2 Нормализация данных и формирование обучающих и проверочные данные

Нормализация - приведения значений к диапазону от 0.0 до 1.0, для этого определяется минимальное значении и максимальное, где минимальному соответствует 0 максимальному соответственно 1.

В библиотеке sklearn, для выполнения процедуры нормализации, присутствует класс sklearn.preprocessing.MinMaxScaler, который содержит методы:

- *MinMaxScaler.fit( df )* - вычисление значений минимального, максимального значений и диапазона;
- *MinMaxScaler.transform( df )* - прямого преобразования из истинного значения к приведенному.

*Листинг 1 нормализация данных*

```
scalerNormX = MinMaxScaler()
scalerNormX.fit(dfX)
dfXNorm = pd.DataFrame (data = scalerNormX.transform(dfX), columns = dfX.columns,
index = dfX.index)
print("Размер таблицы", dfXNorm.shape)

scalerNormY = MinMaxScaler()
scalerNormY.fit(dfY)
dfYNorm = pd.DataFrame (data = scalerNormY.transform(dfY), columns = dfY.columns,
index = dfY.index)
print("Размер таблицы", dfYNorm.shape)
```

```
Размер таблицы (47327, 9)
Размер таблицы (47327, 1)
      carat   cut     color   clarity     depth     table         x         y  \
0  0.016667  1.00  0.833333  0.142857  0.457627  0.260870  0.048352  0.065359
1  0.005556  0.75  0.833333  0.285714  0.169492  0.782609  0.035165  0.034858
2  0.050000  0.75  0.166667  0.428571  0.610169  0.521739  0.103297  0.119826
3  0.061111  0.25  0.000000  0.142857  0.762712  0.521739  0.134066  0.145969
4  0.022222  0.50  0.000000  0.714286  0.677966  0.434783  0.046154  0.061002
```

Рисунок 2.1 – Представление данных в нормализованном виде

*Листинг 2 обучающую и проверочную части*

```
x_train,x_test, y_train, y_test = train_test_split(
    dfX,
    dfY,
    random_state=42,
    test_size=0.2
)

x_norm_train,x_norm_test, y_norm_train, y_norm_test = train_test_split(
    dfXNorm,
    dfYNorm,
    random_state=42,
    test_size=0.2
)
```

В листинге программы 2 *train_test_split()* осуществляет деление данных случайным образом.

## 3 Модель линейной регрессии

*Листинг 3 функции модели линейной регрессии*

```
def train_linear_model(x_tr, y_tr, x_t, y_t):
    model = linear_model.LinearRegression()
    model.fit(x_tr, y_tr)

    y_pred = model.predict(x_t)

    mse = mean_squared_error(y_t, y_pred)
    rmse = mse ** 0.5
    r2 = r2_score(y_t, y_pred)

    return model, y_pred, r2, rmse
```

*linear_model.LinearRegression()* и *model.fit(x_tr, y_tr)* – данные команды отвечают за создание модели линейно регрессии и процесс обучения соответственно.

*model.predict(x_t)* – команда отвечает за прогнозирование.

Так же определим метки оценки качества модели:

- MSE - Mean Squared Error (Средняя квадратичная ошибка), показывает средний квадрат ошибок;
- RMSE - Root Mean Squared Error (Среднеквадратичная ошибка), показывает типичноре отклонение величины ошибки;
- $R^2$ -R-squared (Коэффициент детерминации), показывает на сколько сильно ошибка отличается от среднего значения.

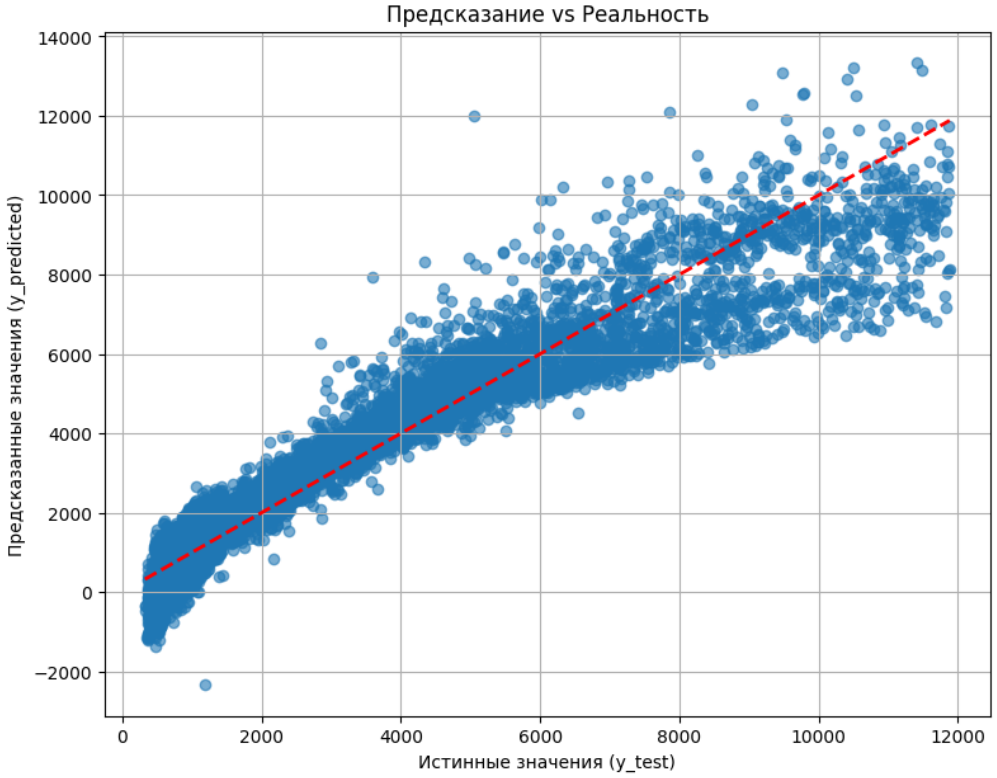



Рисунок 3.1 – Сравнение спрогнозированных значений с эталонными

В результате прогнозирования значений с помощью линейной регрессии коэффициент детерминации имеет значение $R^2 = 0.9152$, а среднеквадратичная ошибка RMSE = 798,569$.

## 4 Модель нейронной сети

*Листинг 4 функция модели нейроной сети*

```
def compile_model(add_layers:bool, feature):
  model = tf.keras.models.Sequential()

  # Входной слой
  model.add(tf.keras.layers.Input(shape=(len(feature),)))

  # Скрытые слои
  model.add(tf.keras.layers.Dense(64, activation='relu'))
  model.add(tf.keras.layers.Dense(32, activation='relu'))
  model.add(tf.keras.layers.Dense(1, activation='linear'))


  # Функции потерь и оптимизации
  fLoss=tf.keras.losses.MeanSquaredError()
  fOptimizer=tf.keras.optimizers.Adam(learning_rate=0.02)
  fMetricList = [tf.keras.losses.MeanSquaredError(),
tf.keras.losses.MeanAbsoluteError()]

  model.compile(
      loss=fLoss,
      optimizer=fOptimizer,
      metrics=fMetricList
  )
  return model

def fit_model_and_print_results(model, feature:list):
  totalHistoryLossTrain=[] # Вспомогательный список для хранения полной истории
обучения
  totalHistoryLossTest=[]  # Вспомогательный список для хранения полной истории
обучения
  history = model.fit(
    x_norm_train[feature],
    y_norm_train,
    validation_data=(
        x_norm_test[feature],
        y_norm_test
        ),
    epochs=20,
    batch_size=64,
    verbose=0
  )
  y_pred = model.predict(x_norm_test[feature])

  r2 = r2_score(y_norm_test, y_pred)
  mse = mean_squared_error(y_norm_test, y_pred)
  rmse = mse ** 0.5

  return y_pred, r2, rmse, history
```

На в ход нейронной сети подаётся 9 целевых параметра, которые потом являются входными параметрами для скрытого слоя состоящего из 64 нейронов с функцией активации *relu*, см. рисунок 4.1.

```
Вид ф-ии активации слоя: relu
Кол-во ВХодов  слоя: 9
Кол-во ВЫХодов слоя: 64
Кол-во нейронов слоя:    64
```

Рисунок 4.1 – Первый скрытый слой модели нейронной сети

Параметры второго скрытого слоя приведены на рисунке 4.2. он состоит из 32 нейронов с функцией активации *relu*.

```
Вид ф-ии активации слоя: relu
Кол-во ВХодов  слоя: 64
Кол-во ВЫХодов слоя: 32
Кол-во нейронов слоя:    32
```

Рисунок 4.2 – Второй скрытый слой модели нейронной сети

На выходной слой модеои состоит из 1 нейрона, значения цены и меет функцию активации *linea,* см. рисунок 4.3.

```
Вид ф-ии активации слоя: linear
Кол-во ВХодов  слоя: 32
Кол-во ВЫХодов слоя: 1
Кол-во нейронов слоя:    1
```

Рисунок 4.3 – Выходной слой модели

Функция оптимизации и оценки потерь используется *Adam* и *MSE* соответственно.

В результате прогнозирования значений с помощью модели нейронной сети коэффициент детерминации имеет значение $R^2 = 0.972$, а среднеквадратичная ошибка RMSE = 0.0394 в нормализованном виде.

## 5 Сохранение модели и формирование JSON файла

*Листинг 5 формирование JSON для модели линейной регрессии*

```
def build_model_info_linear(
    model,
    features,
    target_names,
    r2,
    rmse,
    requires_norm: bool = False
):

    # тип модели
    model_type = type(model).__name__
    params = model.get_params()

    # коэффициенты и свободный член

    coef = np.asarray(model.coef_)
    intercept = np.asarray(model.intercept_)
    coef_list = coef.tolist()
    intercept_list = intercept.tolist()

    coef_by_feature = {}
    for i, tname in enumerate(target_names):
        coef_by_feature[tname] = {
            feat: float(w) for feat, w in zip(features, coef_list[i])
            }

    model_info = {
        "Тип модели": model_type,
        "Названия входных признаков": list(features),
        "Названия выходных признаков": list(target_names),
        "R2": float(r2),
        "RMSE": float(rmse),
        "Коэффициенты по признакам": coef_by_feature,
        "Свободный_член": intercept_list
    }
    return model_info
```

Функция *build_model_info_linear* формирует структурированное описание модели линейной регрессии. В цикле *for feat, w in zip(features, coef_list[i])* осуществляется сопоставление полученного веса с его признаком, *zip*.

*Листинг 6 сохранение модели линейной регрессии и JSON*

```
with open("m1_info.json", "w", encoding="utf-8") as f:
    js.dump(model_info_m1, f, ensure_ascii=False, indent=2)

with open("linear_model.pkl", "wb") as f:
    pickle.dump(model_m1, f)
```

*Листинг 7 формирование JSON для модели нейронной сети*

```
def build_model_info_from_keras(model, features, target_names, r2, rmse,
requires_norm: bool):
    # Описание структуры
    layer_desc = []
    for layer in model.layers:
        cfg = layer.get_config()
        layer_type = layer.__class__.__name__

        if isinstance(layer, tf.keras.layers.Dense):
            units = cfg.get("units")
            activation = cfg.get("activation")
            layer_desc.append(f"{layer_type}(units={units},
activation={activation})")
        else:
            layer_desc.append(layer_type)

    structure_str = " -> ".join(layer_desc)

    # Функция потерь
    if isinstance(model.loss, str):
        loss_name = model.loss
    else:
        loss_name = model.loss.__class__.__name__

    # Оптимизатор и его learning rate
    opt = model.optimizer
    opt_name = opt.__class__.__name__
    try:
        lr_value = float(tf.keras.backend.get_value(opt.learning_rate))
    except Exception:
        lr_value = None

    model_info = {
        "Тип модели": type(model).__name__,
        "Нормализованные данные": bool(requires_norm),
        "Названия входных признаков": list(features),
        "Названия выходных признаков": list(target_names),
        "R2": float(r2),
        "RMSE": float(rmse),
        "Структура": structure_str,
        "Функция потерь": loss_name,
        "Оптимизатор": f"{opt_name}(learning_rate={lr_value})"
    }
    return model_info
```

Функция *build_model_info_from_keras()* формирует структурированное описание модели нейронной сети. В цикл *for layer in model.layers:* извлекается тип слоя и его конфигурация. *if isinstance(layer, tf.keras.layers.Dense):*условие по которому определяектся тип слоя и выписываются его параметры. *float(tf.keras.backend.get_value(opt.learning_rate))* данная команда определяет заданные параметры в оптимизаторе, в случае их отсутствия выводит *None*, *except Exception: lr_value = None*

*Листинг 8 сохранение модели, шкалеров и JSON*

```
model_m2.save('keras_model.h5')
with open("scaler_x.pkl", "wb") as f:
    pickle.dump(scalerNormX, f)

with open("scaler_y.pkl", "wb") as f:
    pickle.dump(scalerNormY, f)

with open("m2_info.json", "w", encoding="utf-8") as f:
    js.dump(model_info_m2, f, ensure_ascii=False, indent=2)
```

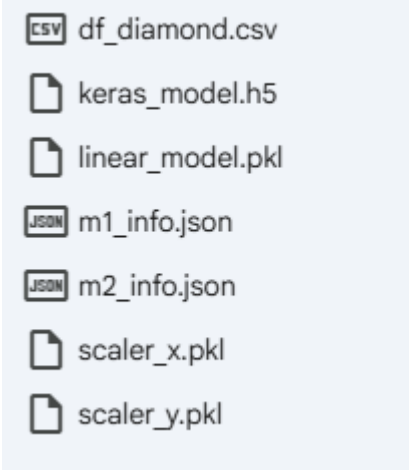



Рисунок 5.1 – сохранённые форматы моделей, шкалеров и *JSON*


```
{
  "Тип модели": "LinearRegression",
  "Названия входных признаков": [
    "carat",
    "cut",
    "color",
    "clarity",
    "depth",
    "table",
    "x",
    "y",
    "z",
  ],
  "Названия выходных признаков": [
    "price",
  ],
  R2: 0.9151602765339839,
  RMSE: 798.5693271182033,
  "Коэффициенты по признакам": {
    price: {
      carat: 10513.862029486445,
      cut: 59.70354835550495,
      color: 244.48529475706488,
      clarity: 376.3449362625993,
      depth: 7.856496977681996,
      table: -22.211148847180112,
      x: -1294.8241636591206,
      y: 972.5681219445239,
      z: -1176.118075554485,
    },
  },
  Свободный_член: [
    -624.2015349430158,
  ],
}
```


Рисунок 5.2 – *JSON* модели линейной регрессии

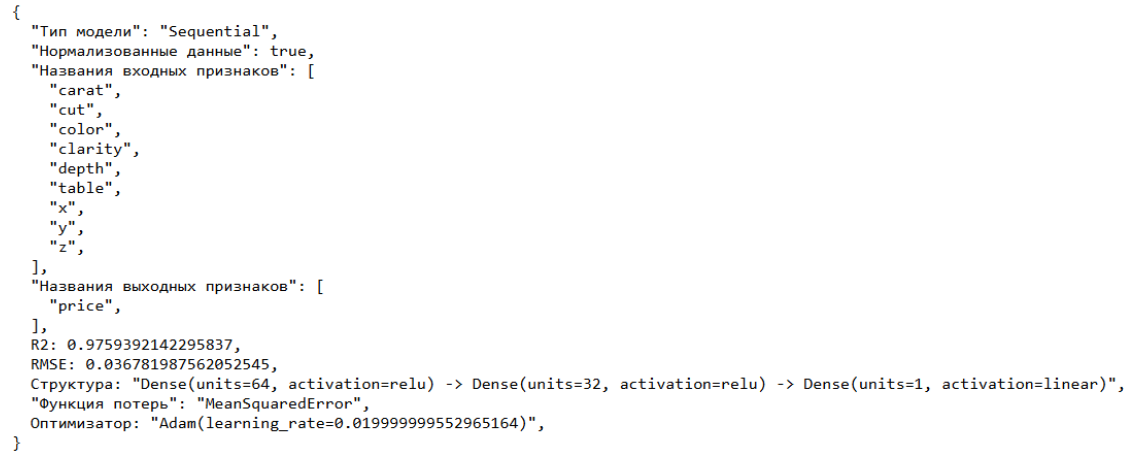



Рисунок 5.3 – *JSON* модели нейронной сети

## 6 Разработка web интерфейса

Разработанная программа по прогнозу цены за алмаз имеет модульную архитектуру, в работе было принято решение отделить внутреннюю логику от представления. Первым модулем является сервисный, он представлен в приложении А. Модуль графического интерфейса программы представлен в приложении Б.

*Листинг 9 функция проверки на наличие моделейи и шкалеров*

```
def init_models_and_scalers():
    if 'models_loaded' not in st.session_state:
        try:
            linear_model, keras_model, scaler_x, scaler_y =
load_models_and_scalers()
            st.session_state.update({
                'linear_model': linear_model,
                'keras_model': keras_model,
                'scaler_x': scaler_x,
                'scaler_y': scaler_y,
                'models_loaded': True
            })
            st.toast("✅ Модели и шкалёры успешно загружены.")
        except Exception as e:
            st.toast(f"❌ Ошибка загрузки: {e}")
            st.stop()
```

Функция *init_models_and_scalers()* однократно проводит проверку на уже загруженные модели и объекты шкалеры. Условие *if 'models_loaded' not in st. session_state: как раз и*используется для проверки на ранее уже загруженные файлы, в случае неудачи выдает ошибку.

*Листинг 10 функция загрузки файлов модели и шкалеров*

```
def load_models_and_scalers():
    with open(FOLDER + 'linear_model.pkl', 'rb') as f:
        lin_model = pickle.load(f)
```

```
    ker_model = load_model(FOLDER + 'keras_model.h5', compile=False)

    with open(FOLDER + 'scaler_x.pkl', 'rb') as f:
        scal_x = pickle.load(f)

    with open(FOLDER + 'scaler_y.pkl', 'rb') as f:
        scal_y = pickle.load(f)

    return lin_model, ker_model, scal_x, scal_y
```

Команда *with open()* в своей конструкции указывает путь к файлу который необходимо открыть, команда *pickle.load(f)* обеспечивает считывание файла.

*Листинг 11 настройка интерфейса*

```
notification_shown = False

st.set_page_config(page_title="Прогнозирование цены бриллианта", layout="wide")
st.title("💎 Прогнозирование цены бриллианта")

service.init_models_and_scalers()

linear_model = st.session_state.linear_model
keras_model = st.session_state.keras_model
scaler_x:MinMaxScaler = st.session_state.scaler_x
scaler_y:MinMaxScaler = st.session_state.scaler_y

input_column, margin, output_column = st.columns([2, 0.25, 3])
```

*st.set_page_config* –устанавливать название страницы.

*st.title* – заголовок страницы.

*input_column, margin, output_column = st.columns([2, 0.25, 3])* – разделение экрана на рабочие зоны.

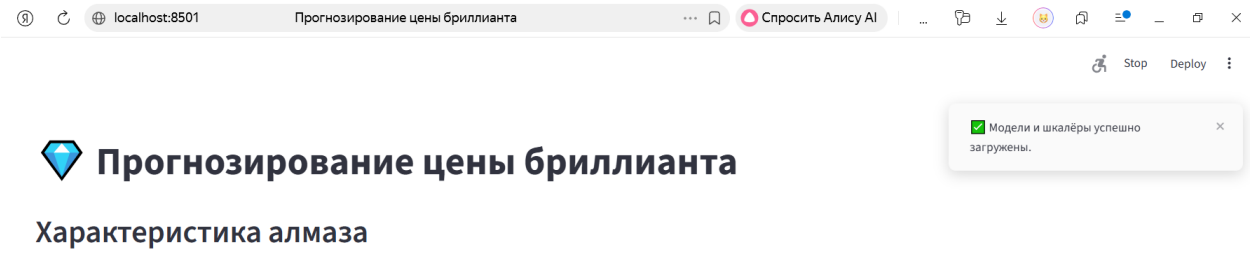



Рисунок 6.1 – Результат части программы листинг 11

*Листинг 12 ввод характеристик, левая рабочая зона*

```
st.header("Характеристика алмаза")

carat = st.slider("Караты", min_value=0.2, max_value=2.0, value=1.1, step=0.01)
cut = st.selectbox("Качество огранки", ["Fair", "Good", "Very Good", "Premium",
"Ideal"], index=2)
color = st.radio("Цвет", ['J', 'I', 'H', 'G', 'F', 'E', 'D'], index=3,
horizontal=True)
clarity = st.selectbox("Чистота", ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2',
'VVS1', 'IF'], index=4)
```

```
depth = st.slider("Глубина", min_value=58.8, max_value=64.7, value=61.8, step=0.1)
table = st.slider("Плоская грань", min_value=52.0, max_value=63.5, value=57.8,
step=0.1)
x = st.slider("Длина", min_value=3.73, max_value=8.28, value=6.0, step=0.01)
y = st.slider("Высота", min_value=3.68, max_value=8.27, value=5.98, step=0.01)
z = st.slider("Ширина", min_value=1.41, max_value=5.3, value=3.36, step=0.01)
```

*st.header* – загаловок рабочей зоны.

*st.slider* – слайдер, ползунок изменяет значения параметра, указывается наименование, максимальное и минимальное значение, значение полмолчания и шаг изменения.

*st.selectbox* – выпадающий список, index присваивается значение по умолчанию.

*st.radio* – выбор одного значения из набора.

*Листинг 13 от рисовка кнопки прогнозирования*

```
### Кнопка предсказания
st.markdown("""
<style>
    button[kind="primary"] {
        background: linear-gradient(135deg, #8b5cf6, #7c3aed);
        color: white;
        border-radius: 12px;
        padding: 16px 28px;
        font-size: 20px;
        font-weight: 600;
        box-shadow: 0 4px 14px rgba(139, 92, 246, 0.3);
    }
    button[kind="primary"]:hover {
        transform: translateY(-2px);
        box-shadow: 0 6px 20px rgba(139, 92, 246, 0.4);
    }
</style>
""", unsafe_allow_html=True)

clicked = st.button("Предсказать", type="primary", use_container_width=True)
```

Внутри *st.markdown* задаются параметры кнопки градиентный фон, белый текст, скругление, размер/жирность шрифта и тень.

*st.button* – кнопка без фиксации, в момент нажатия выдает значение *True*.

*Листинг 14 создание таблицы исходных значений*

```
### Таблица исходных X
st.markdown(
    "<h2 style='text-align: center; color: #2d3748; font-size:
30px';>исходные X</h2>",
    unsafe_allow_html=True
)
df_raw_x = pd.DataFrame(input_data)
st.dataframe(
    df_raw_x,
    hide_index=True,
    use_container_width=True,
    column_config={
```

```
        "carat": st.column_config.NumberColumn("carat", format="%.2f"),
        "depth": st.column_config.NumberColumn("depth", format="%.1f"),
        "table": st.column_config.NumberColumn("table", format="%.1f"),
        "x": st.column_config.NumberColumn("x", format="%.2f"),
        "y": st.column_config.NumberColumn("y", format="%.2f"),
        "z": st.column_config.NumberColumn("z", format="%.2f"),
    }
)
```

Внутри *st.markdown* задаются параметры таблицы, выравнивание, размер шрифт.

*pd.DataFrame(input_data)* – создание датафрейма.

*st.dataframe* – отображение введённых значений из левой рабочей зоны в таблицу.

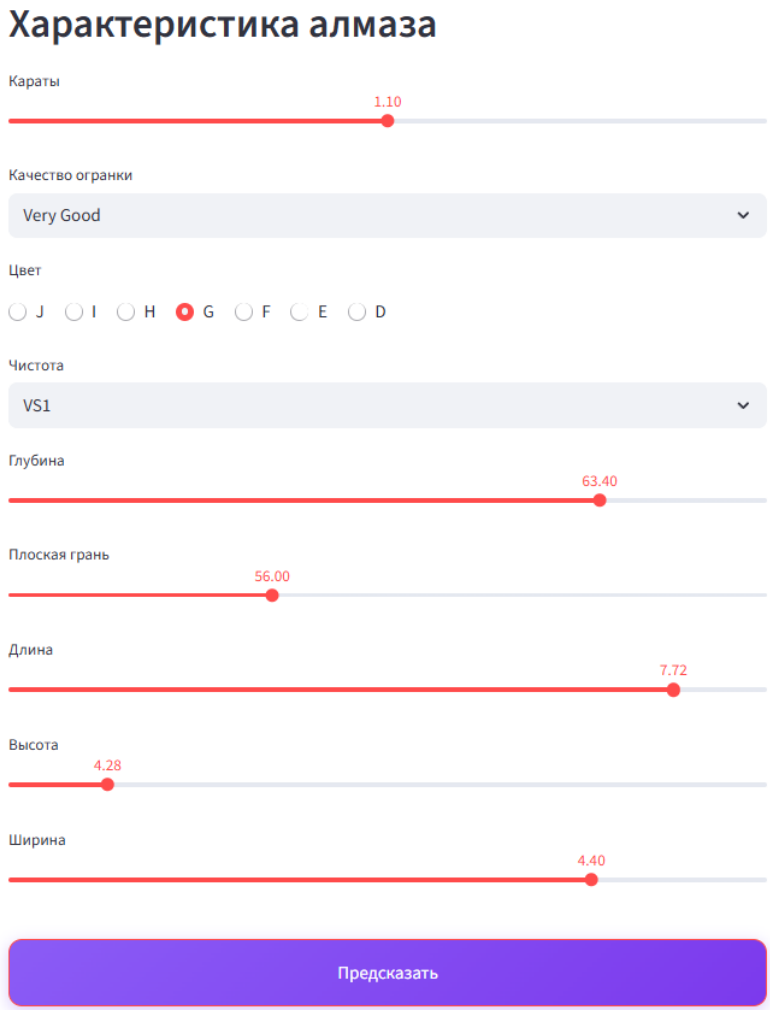



Рисунок 6.2 – Результат части программы листинг 12 и 13, левая рабочая зона

исходные X

| carat | cut | color | clarity | depth | table | x | y | z |
|---|---|---|---|---|---|---|---|---|
| 1.10 | Very Good | G | VS1 | 63.4 | 56.0 | 7.72 | 4.28 | 4.40 |

нормализованные X

| carat | cut | color | clarity | depth | table | x | y | z |
|---|---|---|---|---|---|---|---|---|
| 0.5 | 0.5 | 0.5 | 0.5714 | 0.7797 | 0.3478 | 0.8769 | 0.1307 | 0.7686 |

Рисунок 6.3 – Результат части программы листинг 14 и 15

*Листинг 15 создание таблицы нормализхованных значений*

```
### Таблица нормализованных X
df_x = service.get_prepared_df_x(df_raw_x)
st.markdown(
    "<h2 style='text-align: center; color: #2d3748; font-size:
30px';>нормализованные X</h2>",
    unsafe_allow_html=True
)
columns = ['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y',
'z']
df_norm_x = pd.DataFrame (data = scaler_x.transform(df_x), columns =
df_x.columns, index = df_x.index)
st.dataframe(df_norm_x, hide_index=True, use_container_width=True)
```

*service.get_prepared_df_x(df_raw_x)* – запуск функции преобразования признаков в численный вид.

*pd.DataFrame (data = scaler_x.transform(df_x), columns = df_x.columns, index = df_x.index)* – формирование датафрейма с нормализованными данными.

*st.dataframe* – отображение нормализованных значений.

Листинг 16 результаты расчёта

```
if clicked:
    with col_linear:
        st.header("Линейная регрессия")
        st.write("**R²=0.9163**")
        st.write("**RMSE=797.49**")

        st.subheader("предсказанный Y")
        pred_price_linear = max(0, linear_model.predict(df_x).item())

        st.metric(label="price", value=f"${pred_price_linear:,.2f}")

    with col_neural:
        st.header("Нейронная сеть")
        st.write("**R²=0.974**")
        st.write("**RMSE=0.0353**")

        st.subheader("нормализованный Y")
        norm_y_neural = max(0, keras_model.predict(df_norm_x).item())
```

```
        st.metric(label="price", value=f"{norm_y_neural:.4f}")

        st.subheader("предсказанный Y")
        pred_price_neural = scaler_y.inverse_transform([[norm_y_neural]]).item()
        st.metric(label="price", value=f"${pred_price_neural:,.2f}")
```

*max()* – оперделяет наибольше значение из двух, избавляется от отрицательных значений.

*st.metric* – отвечает за вывод результата прогнозируемого значения цены.


Рисунок 6.4 - Результат части программы листинг 16

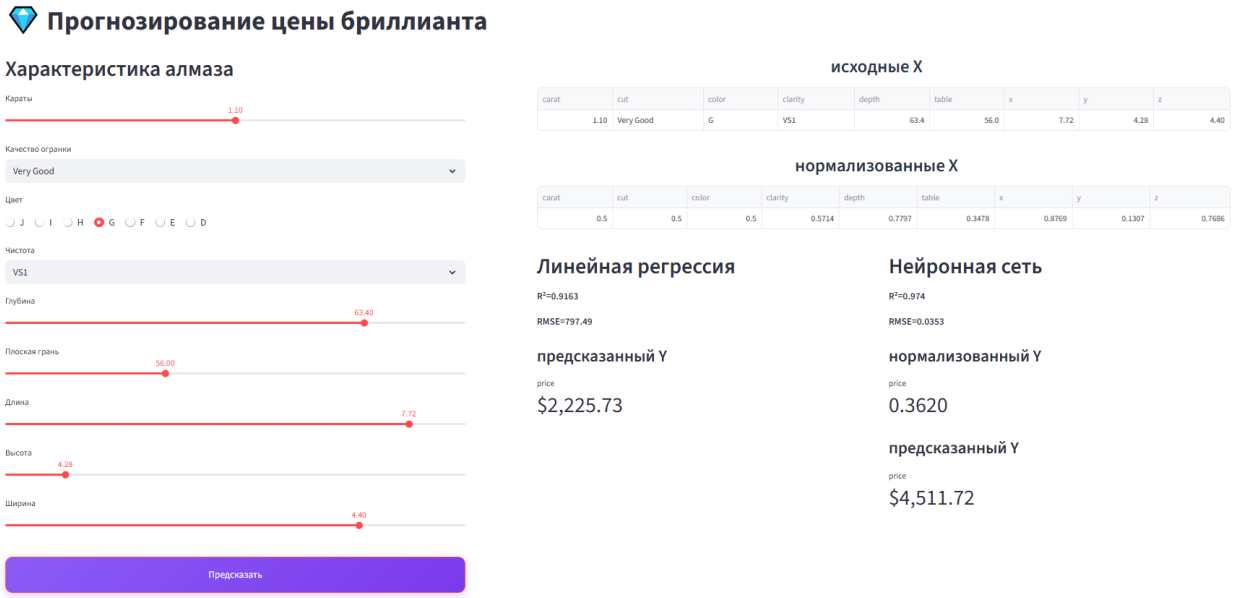



Рисунок 6.5 – Web интерфейс

## 7 Проверка полученного результата

С помощью данных приведённых в таблицах 7.1 и 7.2 графически визуализируем полученные значения на рисунке 7.1.

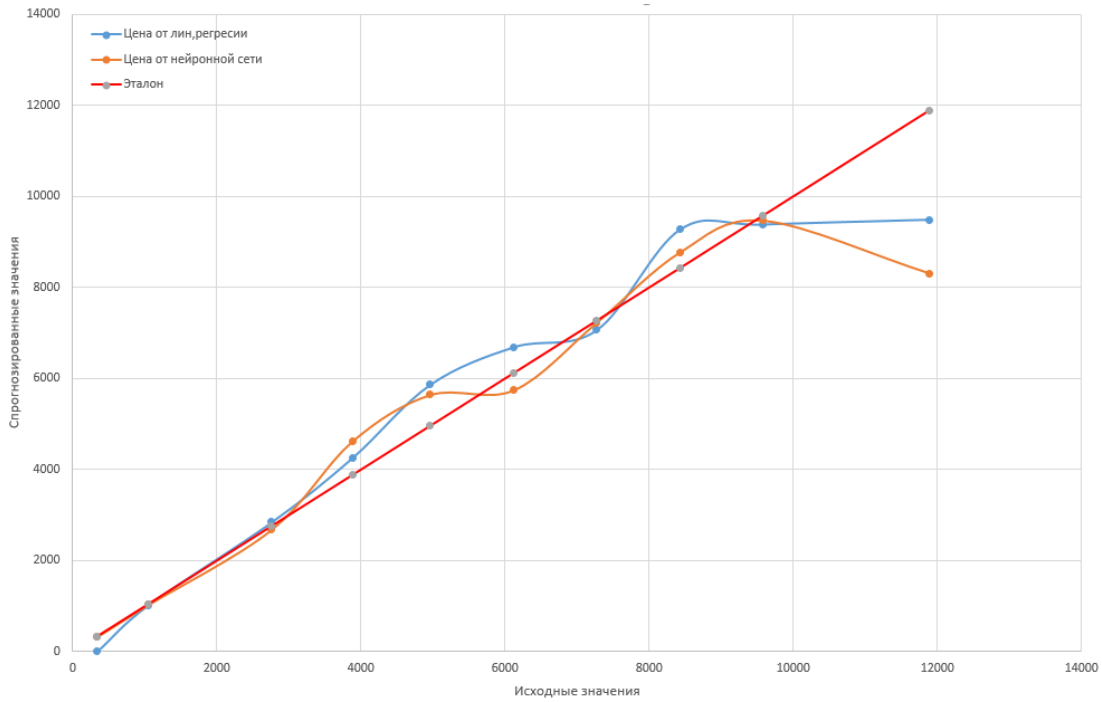
Таблица 7.1 – Вводимые параметры

| № | carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.24 | Very good | J | VV2 | 62.8 | 57 | 336 | 3.94 | 3.96 | 2.48 |
| 2 | 1.68 | Ideal | E | SI2 | 60.4 | 55 | 11888 | 7.79 | 7.7 | 4.68 |
| 3 | 0.44 | Ideal | E | SI1 | 61.1 | 56 | 1040 | 4.92 | 4.9 | 3.0 |
| 4 | 0.7 | Ideal | E | SI1 | 62.5 | 57 | 2757 | 5.7 | 5.72 | .57 |
| 5 | 0.96 | Premium | I | VS2 | 60.6 | 60 | 3876,0 | 6.4 | 6.47 | 3.9 |
| 6 | 1.16 | Premium | I | VS2 | 62.2 | 58.0 | 4958 | 6.74 | 6.67 | 4.17 |
| 7 | 1.22 | Premium | E | SI2 | 61.5 | 58 | 6121 | 6.86 | 6.84 | 4.21 |
| 8 | 1.32 | Very Good | H | SI1 | 62.3 | 57 | 7270 | 7.04 | 6.99 | 4.37 |
| 9 | 1.5 | Very Good | J | VVS2 | 63.3 | 57 | 8427 | 7.28 | 7.25 | 4.6 |
| 10 | 1.51 | Good | I | VS1 | 63.6 | 57 | 9581 | 7.14 | 7.21 | 4.56 |

Таблица 7.2 – Сравнение результатов

| № | Исходное значение цены | Цена от лин.регресии | Цена от нейронной сети |
|---|---|---|---|
| 1 | 336 | 0 | 326 |
| 2 | 1040 | 1012.21 | 1025.69 |
| 3 | 2757 | 2841.03 | 2681.95 |
| 4 | 3878 | 4251.54 | 4622.68 |
| 5 | 4958 | 5860.97 | 5640.77 |
| 6 | 6121 | 6687.42 | 5746.64 |
| 7 | 7270 | 7062.18 | 7231.42 |
| 8 | 8427 | 9274.19 | 8774.98 |
| 9 | 9581 | 9379.54 | 9465.55 |
| 10 | 11888 | 9485.56 | 8309.66 |

В результате полученных значений, по рисунку 7.1, спрогнозированные значения нейроной сети являются наиболее близкими к эталонным, при значениях цены более 9500 и на промежутке от 2700 до 5000 линейная регрессия показывает наиболее точное значени. Для более эфективноого прогнозирования необходимо комбинироватиь работу моделей.




Рисуно 7.1 – Графическое отображение результатов

## Заключение

В результате проделанной работы были выделены независимые признаки и целевая переменная, после чего выполнена нормализация методом MinMaxScaler и сформированы обучающая и тестовая выборки (train/test). Также реализованы две модели прогнозирования цены: линейная регрессия и нейронная сеть.

Для линейной регрессии получены показатели качества порядка $R^2 \approx 0.915$ и RMSE ≈ 798$, что подтверждает работоспособность линейного приближения. Для нейронной сети построена архитектура с двумя скрытыми слоями (64 и 32 нейрона, ReLU) и выходом из одного нейрона (linear), обучение выполнено на нормализованных данных с функцией потерь MSE и оптимизатором Adam; достигнуты метрики $R^2 \approx 0.972$ и RMSE ≈ 0.039 в нормализованном виде, что демонстрирует более высокую точность по сравнению с линейной моделью.

Проверка работоспособности на наборе тестовых примеров показала, что в целом прогнозы нейронной сети чаще оказываются ближе к эталонным**,** однако в отдельных диапазонах значений линейная регрессия может давать сопоставимый или более точный результат**,** что обосновывает идею комбинирования моделей для повышения устойчивости прогноза.

## Приложение А

*Листинг программы сервисного модуля*

```
import streamlit as st
import pickle
import numpy as np
import pandas as pd
from pandas import DataFrame
from tensorflow.keras.models import load_model

FOLDER = 'models/'

def init_models_and_scalers():
    if 'models_loaded' not in st.session_state:
        try:
            linear_model, keras_model, scaler_x, scaler_y =
load_models_and_scalers()
            st.session_state.update({
                'linear_model': linear_model,
                'keras_model': keras_model,
                'scaler_x': scaler_x,
                'scaler_y': scaler_y,
                'models_loaded': True
            })
            st.toast("✅ Модели и шкалёры успешно загружены.")
        except Exception as e:
            st.toast(f"❌ Ошибка загрузки: {e}")
            st.stop()

@st.cache_resource
def load_models_and_scalers():
    with open(FOLDER + 'linear_model.pkl', 'rb') as f:
        lin_model = pickle.load(f)

    ker_model = load_model(FOLDER + 'keras_model.h5', compile=False)

    with open(FOLDER + 'scaler_x.pkl', 'rb') as f:
        scal_x = pickle.load(f)

    with open(FOLDER + 'scaler_y.pkl', 'rb') as f:
        scal_y = pickle.load(f)

    return lin_model, ker_model, scal_x, scal_y


def get_prepared_df_x(df) -> DataFrame:
    df = df.copy()
    color_order = ['J', 'I', 'H', 'G', 'F', 'E', 'D']
    df['color'] = df['color'].map({v: i + 1 for i, v in enumerate(color_order)})

    cut_order = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']
    df['cut'] = df['cut'].map({v: i + 1 for i, v in enumerate(cut_order)})

    clarity_order = ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF']
    df['clarity'] = df['clarity'].map({v: i + 1 for i, v in
enumerate(clarity_order)})
    return df
```

## Приложение Б

*Листинг программы модуля графического интерфейса*

```
import streamlit as st
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

import service

notification_shown = False

st.set_page_config(page_title="Прогнозирование цены бриллианта", layout="wide")
st.title("💎 Прогнозирование цены бриллианта")

service.init_models_and_scalers()

linear_model = st.session_state.linear_model
keras_model = st.session_state.keras_model
scaler_x:MinMaxScaler = st.session_state.scaler_x
scaler_y:MinMaxScaler = st.session_state.scaler_y

input_column, margin, output_column = st.columns([2, 0.25, 3])

with input_column:
    ### Входные данные
    st.header("Характеристика алмаза")

    carat = st.slider("Караты", min_value=0.2, max_value=2.0, value=1.1, step=0.01)
    cut = st.selectbox("Качество огранки", ["Fair", "Good", "Very Good", "Premium",
"Ideal"], index=2)
    color = st.radio("Цвет", ['J', 'I', 'H', 'G', 'F', 'E', 'D'], index=3,
horizontal=True)
    clarity = st.selectbox("Чистота", ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2',
'VVS1', 'IF'], index=4)
    depth = st.slider("Глубина", min_value=58.8, max_value=64.7, value=61.8,
step=0.1)
    table = st.slider("Плоская грань", min_value=52.0, max_value=63.5, value=57.8,
step=0.1)
    x = st.slider("Длина", min_value=3.73, max_value=8.28, value=6.0, step=0.01)
    y = st.slider("Высота", min_value=3.68, max_value=8.27, value=5.98, step=0.01)
    z = st.slider("Ширина", min_value=1.41, max_value=5.3, value=3.36, step=0.01)

    ### Кнопка предсказания
    st.markdown("""
    <style>
        button[kind="primary"] {
            background: linear-gradient(135deg, #8b5cf6, #7c3aed);
            color: white;
            border-radius: 12px;
            padding: 16px 28px;
            font-size: 20px;
            font-weight: 600;
            box-shadow: 0 4px 14px rgba(139, 92, 246, 0.3);
        }
        button[kind="primary"]:hover {
            transform: translateY(-2px);
            box-shadow: 0 6px 20px rgba(139, 92, 246, 0.4);
        }
    </style>
```

```
    """, unsafe_allow_html=True)

    clicked = st.button("Предсказать", type="primary", use_container_width=True)

with output_column:
    input_data = {
        'carat': [carat],
        'cut': [cut],
        'color': [color],
        'clarity': [clarity],
        'depth': [depth],
        'table': [table],
        'x': [x],
        'y': [y],
        'z': [z]
    }

    ### Таблица исходных X
    st.markdown(
        "<h2 style='text-align: center; color: #2d3748; font-size: 30px';>исходные
X</h2>",
        unsafe_allow_html=True
    )
    df_raw_x = pd.DataFrame(input_data)
    st.dataframe(
        df_raw_x,
        hide_index=True,
        use_container_width=True,
        column_config={
            "carat": st.column_config.NumberColumn("carat", format="%.2f"),
            "depth": st.column_config.NumberColumn("depth", format="%.1f"),
            "table": st.column_config.NumberColumn("table", format="%.1f"),
            "x": st.column_config.NumberColumn("x", format="%.2f"),
            "y": st.column_config.NumberColumn("y", format="%.2f"),
            "z": st.column_config.NumberColumn("z", format="%.2f"),
        }
    )

    ### Таблица нормализованных X
    df_x = service.get_prepared_df_x(df_raw_x)
    st.markdown(
        "<h2 style='text-align: center; color: #2d3748; font-size:
30px';>нормализованные X</h2>",
        unsafe_allow_html=True
    )
    columns = ['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z']
    df_norm_x = pd.DataFrame (data = scaler_x.transform(df_x), columns =
df_x.columns, index = df_x.index)
    st.dataframe(df_norm_x, hide_index=True, use_container_width=True)

    col_linear, col_neural = st.columns(2)

    if clicked:
        with col_linear:
            st.header("Линейная регрессия")
            st.write("**R²=0.9163**")
            st.write("**RMSE=797.49**")

            st.subheader("предсказанный Y")
            pred_price_linear = max(0, linear_model.predict(df_x).item())
```

```
            st.metric(label="price", value=f"${pred_price_linear:,.2f}")

        with col_neural:
            st.header("Нейронная сеть")
            st.write("**R²=0.974**")
            st.write("**RMSE=0.0353**")

            st.subheader("нормализованный Y")
            norm_y_neural = max(0, keras_model.predict(df_norm_x).item())
            st.metric(label="price", value=f"{norm_y_neural:.4f}")

            st.subheader("предсказанный Y")
            pred_price_neural =
scaler_y.inverse_transform([[norm_y_neural]]).item()
            st.metric(label="price", value=f"${pred_price_neural:,.2f}")

st.markdown("---")
```