Классификация

Задача: вычислить мошенника на страховых выплатах

Для выполнения лабораторной работы были выбраны метрики F1-score и ROC-AUC, так как исследуемый датасет является несбалансированным. Метрика Accuracy в данном случае неинформативна, так как модель, предсказывающая всем класс ‘0’ (не фрод), может иметь высокую Accuracy, но будет бесполезна. F1-score позволит контролировать баланс между ложными срабатываниями и пропуском мошенников.

Baseline

Скачаем датасет с kaggle

In [89]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                "buntyshah/auto-insurance-claims-data/versions/1",
                                "insurance_claims.csv")
df
```

Out[89]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 40 columns

Замапим колонки с Yes или No значениями в 1 и 0, пропуски заполним -1. Это нужно, потому что модели не умеют работать со строковыми данными.

In [90]:
```
df_clean = df.copy()

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
df_clean["property_damage"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
```

Закодируем каждое не числовое значение просто номером.

In [91]:
```
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean.head()
```

Out[91]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium
0	328	48	521585	940	2	1	1000	1406.91
1	228	42	342868	635	1	1	2000	1197.22
2	134	29	687698	413	2	0	2000	1413.14
3	256	41	227811	19	0	1	2000	1415.74
4	228	44	367455	922	0	2	1000	1583.91

5 rows × 40 columns

Заполним пропуски нулями. Модели не умеют работать с NaNами.

In [92]:
```
df_clean = df_clean.fillna(0)
```

Разделим на данные и таргет, предварительно удалив даты из датасета. Модели не умеют работать с датами. Далее разобьем на тестовую и обучающую выборку.

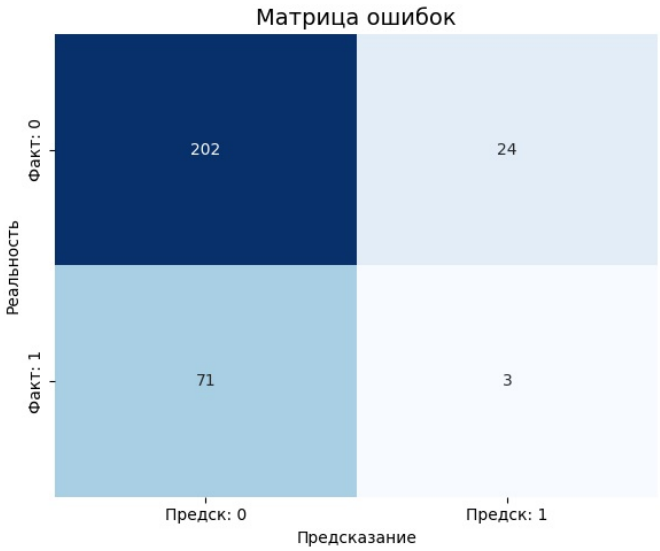
In [93]:
```
from sklearn.model_selection import train_test_split
drop_dates = ["policy_bind_date", "incident_date"]
df_clean = df_clean.drop(drop_dates, axis=1)
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

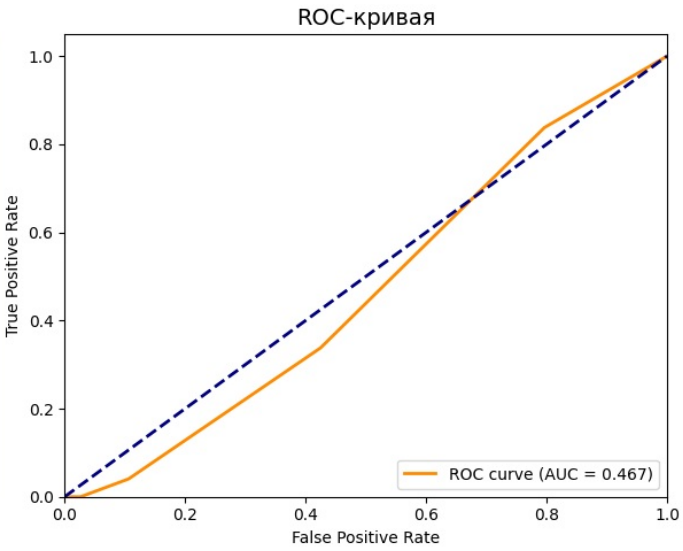
Обучим KNN с параметрами по умолчанию и выведем метрики.

In [94]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, f1_score, roc_auc_score

model = KNeighborsClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
F1-score (класс 1): 0.0594
ROC-AUC: 0.4674
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.89      0.81       226
           1       0.11      0.04      0.06        74

    accuracy                           0.68       300
   macro avg       0.43      0.47      0.43       300
weighted avg       0.58      0.68      0.62       300

==================================================
```

In [95]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc
def graphics(y_test, y_pred, y_prob):
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", cbar=False,
                xticklabels=['Предск: 0', 'Предск: 1'],
                yticklabels=['Факт: 0', 'Факт: 1'])
    plt.title("Матрица ошибок", fontsize=14)
    plt.ylabel("Реальность")
    plt.xlabel("Предсказание")

    plt.subplot(1, 2, 2)
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC-кривая', fontsize=14)
    plt.legend(loc="lower right")

    plt.tight_layout()
    plt.show()
graphics(y_test, y_pred, y_prob)
```



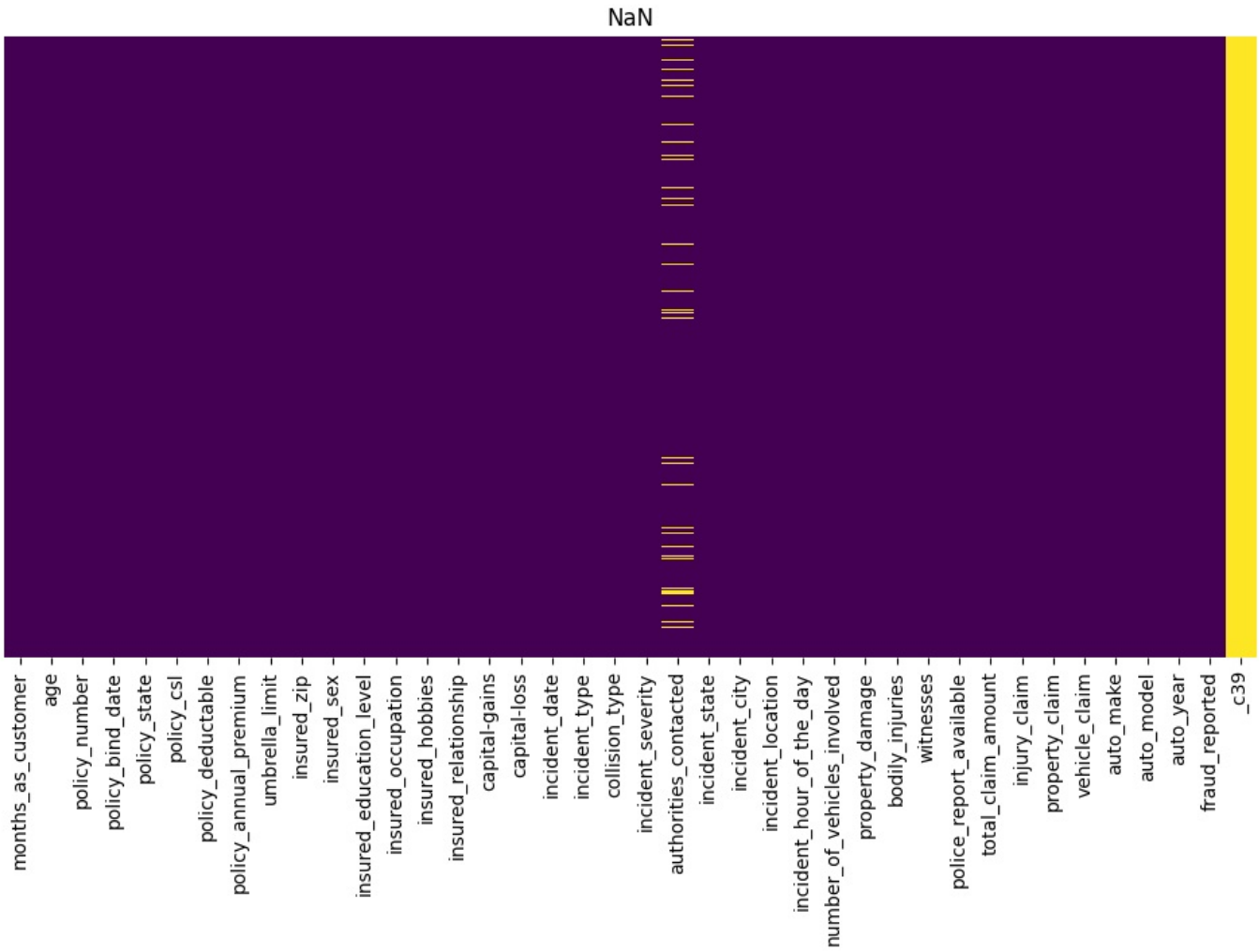


Модель очень плохо определяет мошенников. Accuracy, которая выводится в логах неинформативна, поскольку из-за дисбаланса классов многих просто записывает в честных, тем самым повышая accuracy. На это указывает и F1 метрика, которая для 1 класса очень высокая, а для второго крайне низкая. ROC-AUC менее 0.5, следовательно модель работает хуже случайного угадывания для ранжирования вероятностей класса 1. То есть предсказания вероятностей фактически не информативны.

Improved KNN

Загрузим датасет и сразу выведем пропуски по столбцам.

In [96]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df_i = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                  "buntyshah/auto-insurance-claims-data/versions/1",
                                  "insurance_claims.csv")

nulls = df_i.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df_i)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df_i.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



Видим, что столбец '_c39' полностью пустой, как следствие неинформативен, его можно сразу же удалить. Кроме того, в столбце с контактами таже присутствуют пропуски. Заполним их меткой 'Нет контактов'.

In [97]:
```
df_clean_i = df_i.copy()
df_clean_i = df_clean_i.drop(columns=["_c39"])
df_clean_i["authorities_contacted"] = df_clean_i["authorities_contacted"].fillna("No Contact")
```

Удостоверимся, что присутствует дисбаланс классов.

In [98]:
```
TARGET_NAME = "fraud_reported"
df_clean_i[TARGET_NAME].value_counts()
```

Out[98]:
```
fraud_reported
N    753
Y    247
Name: count, dtype: int64
```

Посмотрим на столбцы с бинарными метками 'police_report_available' и 'property_damage'

In [99]:
```
display(df_clean_i["police_report_available"].unique())
display(df_clean_i["property_damage"].unique())
```

```
array(['YES', '?', 'NO'], dtype=object)
array(['YES', '?', 'NO'], dtype=object)
```

Закодируем YES - 1, NO - 0 и пропуски -1. В целевой переменной так же поменяем значения на бинарные.

In [100…
```
import pandas as pd

df_clean_i["fraud_reported"] = df_clean_i["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean_i["police_report_available"] = df_clean_i["police_report_available"].map({'YES': 1, 'NO': 0, '?': -1})
df_clean_i["property_damage"] = df_clean_i["property_damage"].map({'YES': 1, 'NO': 0, '?': -1})
```

Переформатируем строковые столбцы с датой в то, что понимает pandas.

In [101…
```
dates_cols = ["policy_bind_date", "incident_date"]
for c in dates_cols:
    df_clean_i[c] = pd.to_datetime(df_clean_i[c])
df_clean_i
```

Out[101…

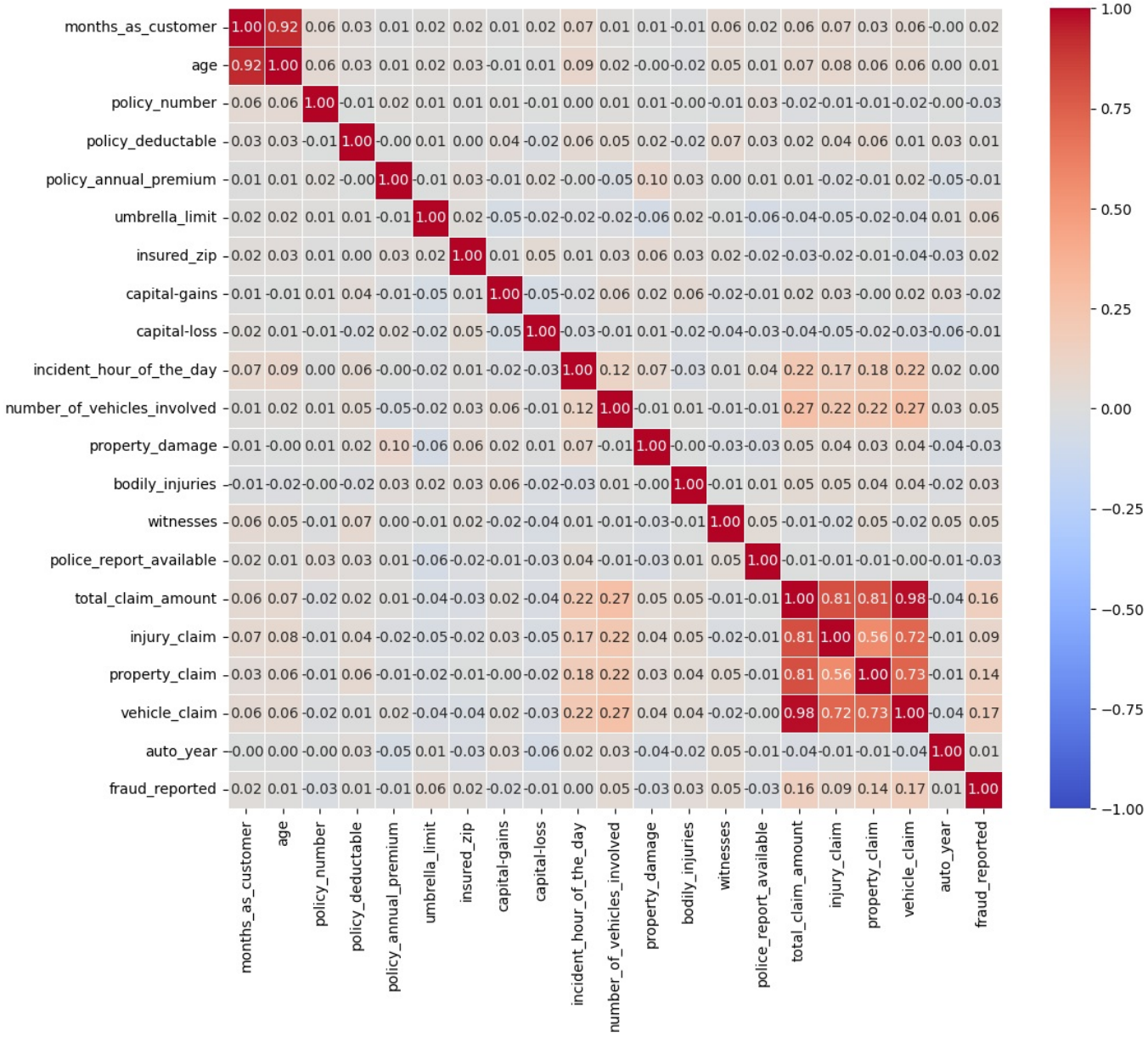
	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 39 columns

Проверим корелляции переменных друг с другом.

In [102…

```
num_cols = df_clean_i.select_dtypes(include=["int64", "float64"]).columns.tolist()

plt.figure(figsize=(12, 10))
correlation_matrix = df_clean_i[num_cols].corr()

sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            vmin=-1, vmax=1,
            linewidths=0.5)

plt.show()
```



Видим много кореллирующих столбцов. Может в будующем попробуем почистить данные от корелляций.

Проверим числовые столбцы на то, нет ли каких-то странностей в них.

In [103…

```
num_cols = df_clean_i.select_dtypes(include=["int64", "float64"]).columns.tolist()
df_clean_i[num_cols].describe().T
```

Out[103…

	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
property_damage	1000.0	-5.800000e-02	8.119700e-01	-1.00	-1.0000	0.0	1.000	1.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
police_report_available	1000.0	-2.900000e-02	8.104417e-01	-1.00	-1.0000	0.0	1.000	1.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00
fraud_reported	1000.0	2.470000e-01	4.314825e-01	0.00	0.0000	0.0	0.000	1.00

umbrella_limit -100000 min выглядит так, будто бы им заполняли отсутствие записей. Исправим это медианой.

In [104…

```
import numpy as np

df_pr_i = df_clean_i.copy()

median_value = df_pr_i.loc[df_pr_i['umbrella_limit'] != -100000, 'umbrella_limit'].median()
df_pr_i.loc[df_pr_i['umbrella_limit'] == -100000, 'umbrella_limit'] = median_value
```

Займемся feature инжинирингом. Сначала распихаем возраст по бакетам, потом разберём дату инцидента на месяц, год и день недели. Добавим некоторые фичи коэфициенты для стоимости средства передвижения, травм и имущества относительно размера выплаты. В конце уберем уже ненужные данные по датам.

In [105…

```
df_features_i = df_pr_i.copy()

df_features_i["age_bucket"] = pd.cut(
    df_features_i["age"],
    bins=[0, 25, 35, 45, 55, 65, 100],
    labels=["<25", "25-34", "35-44", "45-54", "55-64", "65+"],
)

df_features_i["incident_year"] = df_features_i["incident_date"].dt.year
df_features_i["incident_month"] = df_features_i["incident_date"].dt.month
df_features_i["incident_dow"] = df_features_i["incident_date"].dt.dayofweek

df_features_i["injury_ratio"] = df_features_i["injury_claim"] / (df_features_i["total_claim_amount"] + 1e-3)
df_features_i["property_ratio"] = df_features_i["property_claim"]  / (df_features_i["total_claim_amount"] + 1e-
df_features_i["vehicle_ratio"] = df_features_i["vehicle_claim"] / (df_features_i["total_claim_amount"] + 1e-3)

drop_dates = ["policy_bind_date", "incident_date"]
df_features_i = df_features_i.drop(drop_dates, axis=1)
```

In [106…

```
from sklearn.model_selection import train_test_split
X_i = df_features_i.drop(TARGET_NAME, axis=1)
y_i = df_features_i[TARGET_NAME]
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_i, y_i, test_size=0.3, random_state=42, stratify=
```

Тут пайплайн немного поменялся, я добавил нормализацию данных, поскольку KNN это жизненно необходимо, закодируем категориальные столбцы, применим SMOTE, чтобы выровнять дисбаланс классов, и при помощи GridSearch подберем гиперпараметры для числа соседей.

Кстати без SMOTE модель предсказывает только 1 класс (честных), а использование StandartScaler вместо Robust ухудшаешь

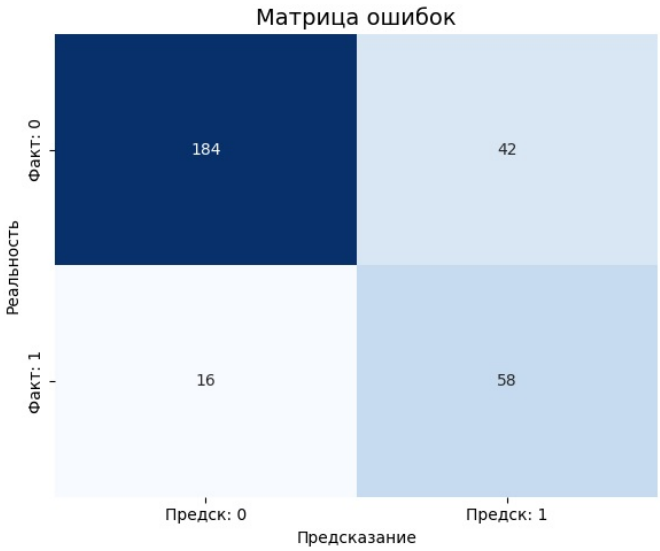
ROC-AUC на 0.3

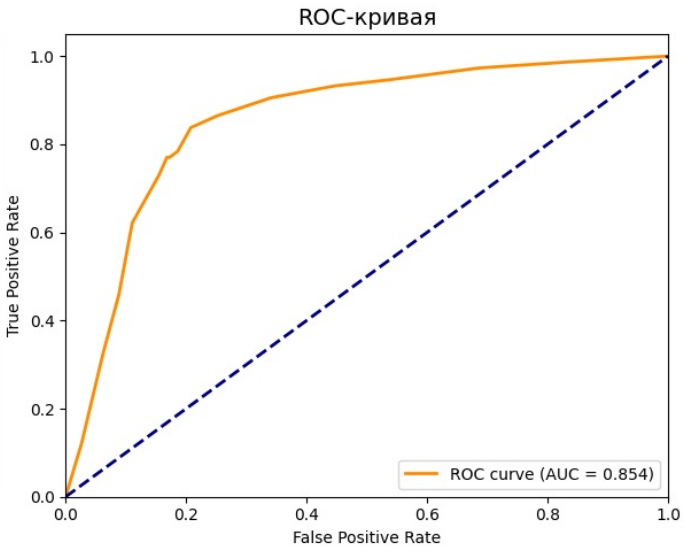
Очень сильно влияет на модель выбор энкодера для категориальных признаков. Использование Onehot нежелательно, поскольку сильно размывает расстояние между векторами, TargetEncoder также ломает модель, потому что не передаёт статистические различия между категориями. Я выбор остановил на WOEEncoder. Он идеален для датасетов с дисбалансом классов для бинарной классификации, поскольку учитывает вероятность приближения фичёй записи к той или иной категории.

In [107…
```
from category_encoders import WOEEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import RobustScaler

num_cols = X_i.select_dtypes(["int64", "float64"]).columns.tolist()
cat_cols = X_i.select_dtypes(["object", "category"]).columns.tolist()

numeric_pipe     = Pipeline([("scale", RobustScaler())])

categorical_pipe = Pipeline([("woe", WOEEncoder())])

ct  = ColumnTransformer([
    ("num", numeric_pipe, num_cols),
    ("cat", categorical_pipe, cat_cols)
])

model_pipe = ImbPipeline([
    ('ct', ct),
    ('smote', SMOTE(random_state=42)),
    ('model', KNeighborsClassifier(n_jobs=-1))
])

knn_params = {
    "model__n_neighbors": [3, 5, 7, 15, 31]
}

grid = GridSearchCV(model_pipe, knn_params, cv=3, scoring='roc_auc', verbose=1)
grid.fit(X_train_i, y_train_i)

print(f"Лучшее k: {grid.best_params_['model__n_neighbors']}")

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
Лучшее k: 15
F1-score (класс 1): 0.6667
ROC-AUC: 0.8545
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.81      0.86       226
           1       0.58      0.78      0.67        74

    accuracy                           0.81       300
   macro avg       0.75      0.80      0.77       300
weighted avg       0.84      0.81      0.82       300

==================================================
```

In [108…
```
graphics(y_test_i, y_pred, y_prob)
```





Модель угадывает большую часть мошенников, при этом почти не обвиняя честных людей в махинациях. ROC_AUC и F1 метрика сильно приросли посравнению с базовым пайплайном.

My implemenation

In [65]:
```python
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np
from collections import Counter
from scipy.spatial.distance import cdist

class MyKNN(ClassifierMixin, BaseEstimator):
    def __init__(self, n_neighbors=5):
        self.n_neighbors = n_neighbors

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
        self.classes_ = np.unique(self.y_train)
        return self

    def _compute_distances(self, X):
        X = np.array(X)
        return cdist(X, self.X_train, metric='euclidean')

    def predict(self, X):
        distances_matrix = self._compute_distances(X)
        predictions = []

        for distances in distances_matrix:
            k = min(self.n_neighbors, len(self.X_train))
            k_idx = np.argsort(distances)[:k]
            k_labels = self.y_train[k_idx]

            most_common = Counter(k_labels).most_common(1)[0][0]
            predictions.append(most_common)

        return np.array(predictions)

    def predict_proba(self, X):
        distances_matrix = self._compute_distances(X)
        probabilities = []

        for distances in distances_matrix:
            k = min(self.n_neighbors, len(self.X_train))
            k_idx = np.argsort(distances)[:k]
            k_labels = self.y_train[k_idx]
            prob_one = np.mean(k_labels == 1)
            probabilities.append([1 - prob_one, prob_one])

        return np.array(probabilities)
```

In [66]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, f1_score, roc_auc_score

model = MyKNN()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
```

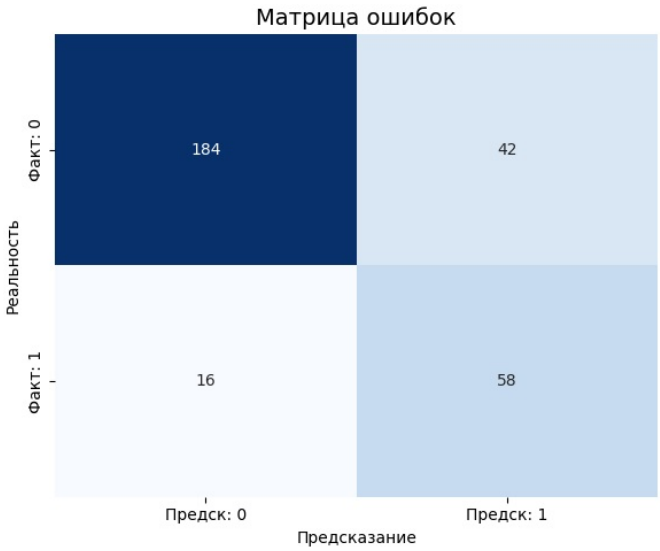
```
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

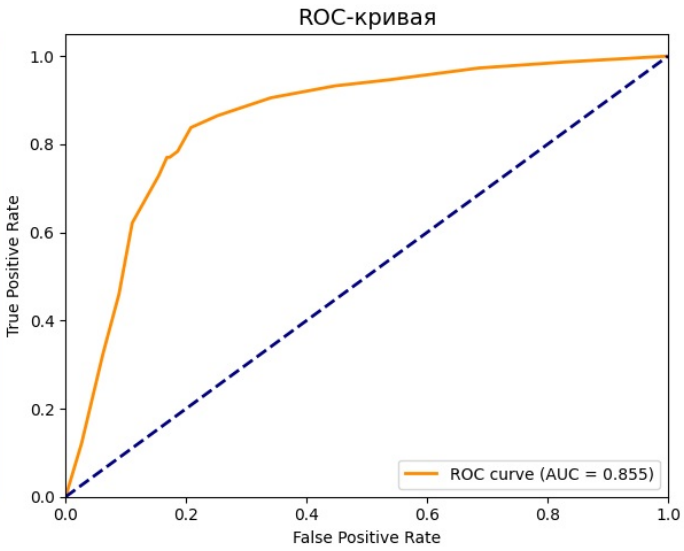
```
F1-score (класс 1): 0.0594
ROC-AUC: 0.4674
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.89      0.81       226
           1       0.11      0.04      0.06        74

    accuracy                           0.68       300
   macro avg       0.43      0.47      0.43       300
weighted avg       0.58      0.68      0.62       300

==================================================
```

In [67]:
```
model_pipe = ImbPipeline([
    ('ct', ct),
    ('smote', SMOTE(random_state=42)),
    ('model', MyKNN())
])

grid = GridSearchCV(model_pipe, knn_params, cv=3, scoring='roc_auc', verbose=1)
grid.fit(X_train_i, y_train_i)

print(f"Лучшее k: {grid.best_params_['model__n_neighbors']}")

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
Лучшее k: 15
F1-score (класс 1): 0.6667
ROC-AUC: 0.8546
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.81      0.86       226
           1       0.58      0.78      0.67        74

    accuracy                           0.81       300
   macro avg       0.75      0.80      0.77       300
weighted avg       0.84      0.81      0.82       300

==================================================
```

In [68]:
```
graphics(y_test, y_pred, y_prob)
```





Моя имплементация работает так же как и из scikit-learn. Там никаких хитростей нет, поэтому нормально работает.

	Base KNN	Improved KNN	My improved KNN
ROC_AUC	0.4674	0.8545	0.8545
F1 (1)	0.81	0.86	0.86
F1 (2)	0.06	0.67	0.67

Регрессия

Задача: предсказать зарплату игрока НХЛ по его статистике.

Будем использовать MAE, так как она очень наглядная, будет легко понять, насколько долларов ошибается модель. В качестве дополнительной метрики будем использовать R^2, чтобы смотреть насколько отличается предсказание от среднего.

Baseline

In [69]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter


path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")

df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)
```

Сделаем минимальные телодвижения, чтобы регрессор заработал: закодируем все строковые данные, уберем NaN

In [70]:
```
df_clean = df.fillna(0)
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean = df_clean.drop("Born", axis=1)
```

In [71]:
```
TARGET_NAME = "Salary"
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

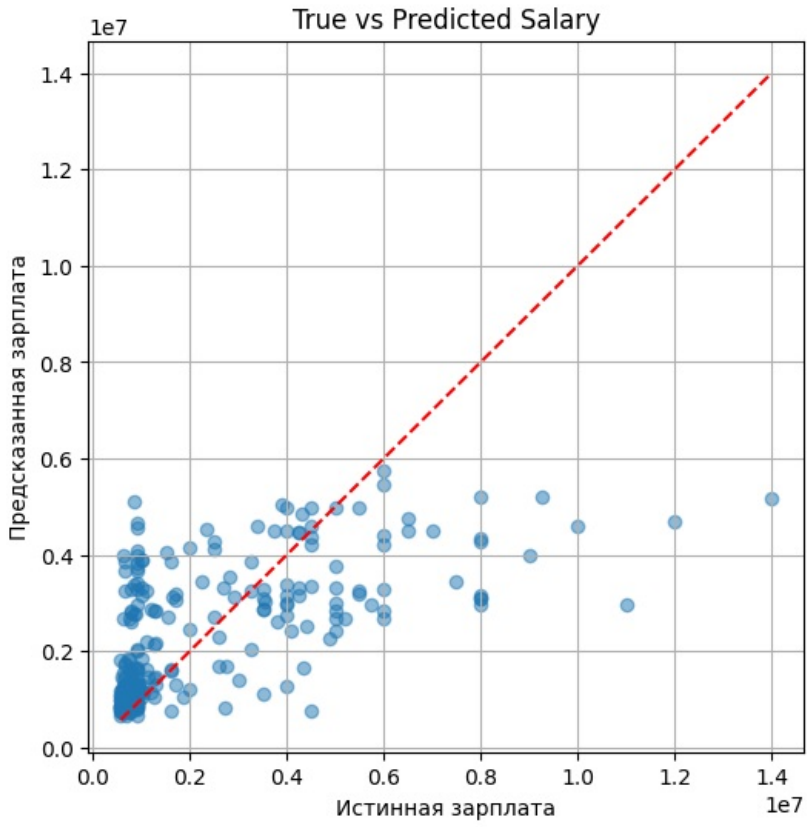
In [72]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
knn = KNeighborsRegressor(n_neighbors=25, weights='distance')
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

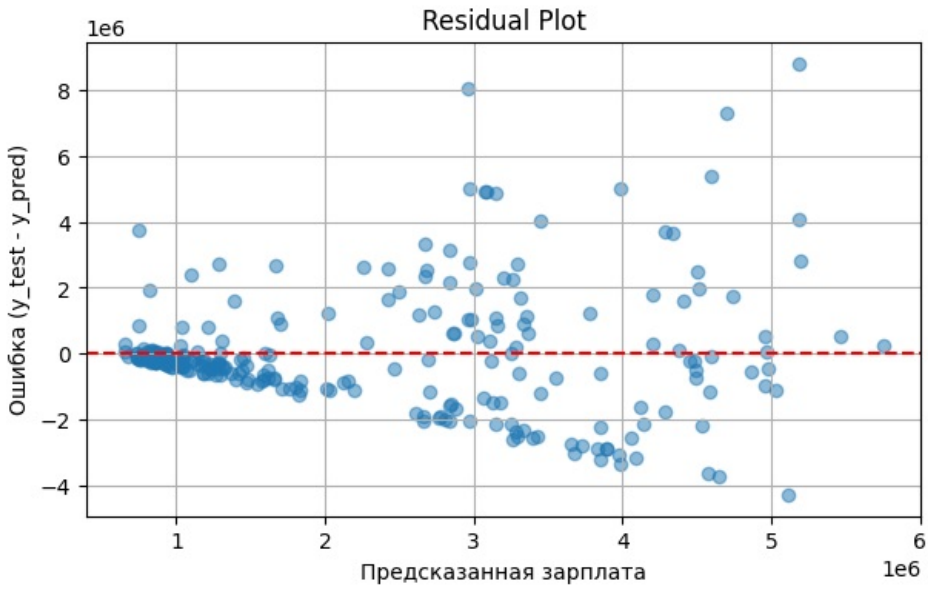
```
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, [y_train.mean()]*len(y_test))

print(f"--- Baseline KNN ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

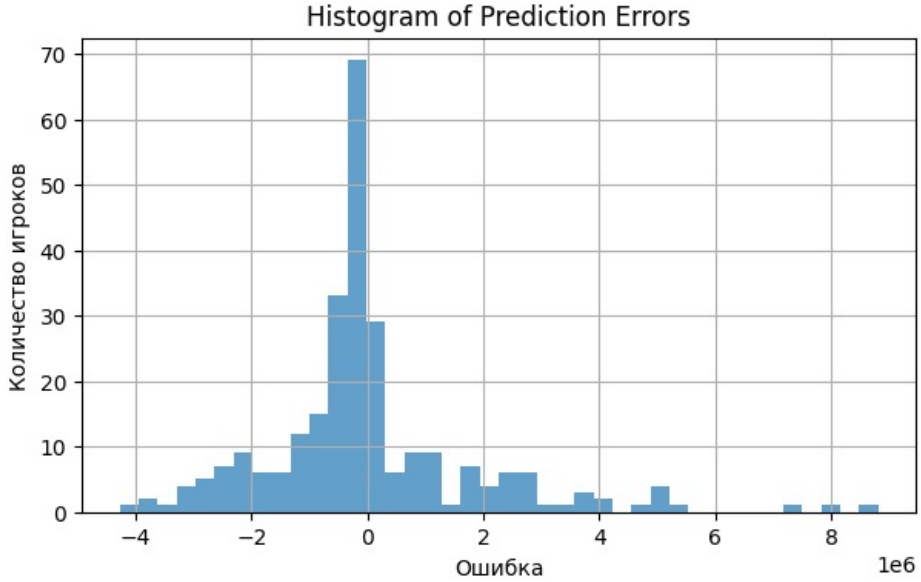
```
--- Baseline KNN ---
MAE (Ошибка в долларах): 1173197.566
R2 Score: 0.399
```

MAE получилось около миллиона. В целом неплохо, учитывая что данные используются только за один сезон, поэтому прямо кардинальных улучшений мы вряд-ли увидим. Ошибка всегда будет довольно-таки большая.

In [73]:
```
def true_vs_predicted(y_test, y_pred):
    plt.figure(figsize=(6,6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
    plt.xlabel("Истинная зарплата")
    plt.ylabel("Предсказанная зарплата")
    plt.title("True vs Predicted Salary")
    plt.grid(True)
    plt.show()

def resudial(y_test, y_pred):
    residuals = y_test - y_pred
    plt.figure(figsize=(7,4))
    plt.scatter(y_pred, residuals, alpha=0.5)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel("Предсказанная зарплата")
    plt.ylabel("Ошибка (y_test - y_pred)")
    plt.title("Residual Plot")
    plt.grid(True)
    plt.show()
def histogramm_error(y_test, y_pred):
    plt.figure(figsize=(7,4))
    residuals = y_test - y_pred
    plt.hist(residuals, bins=40, alpha=0.7)
    plt.title("Histogram of Prediction Errors")
    plt.xlabel("Ошибка")
    plt.ylabel("Количество игроков")
    plt.grid(True)
    plt.show()
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```







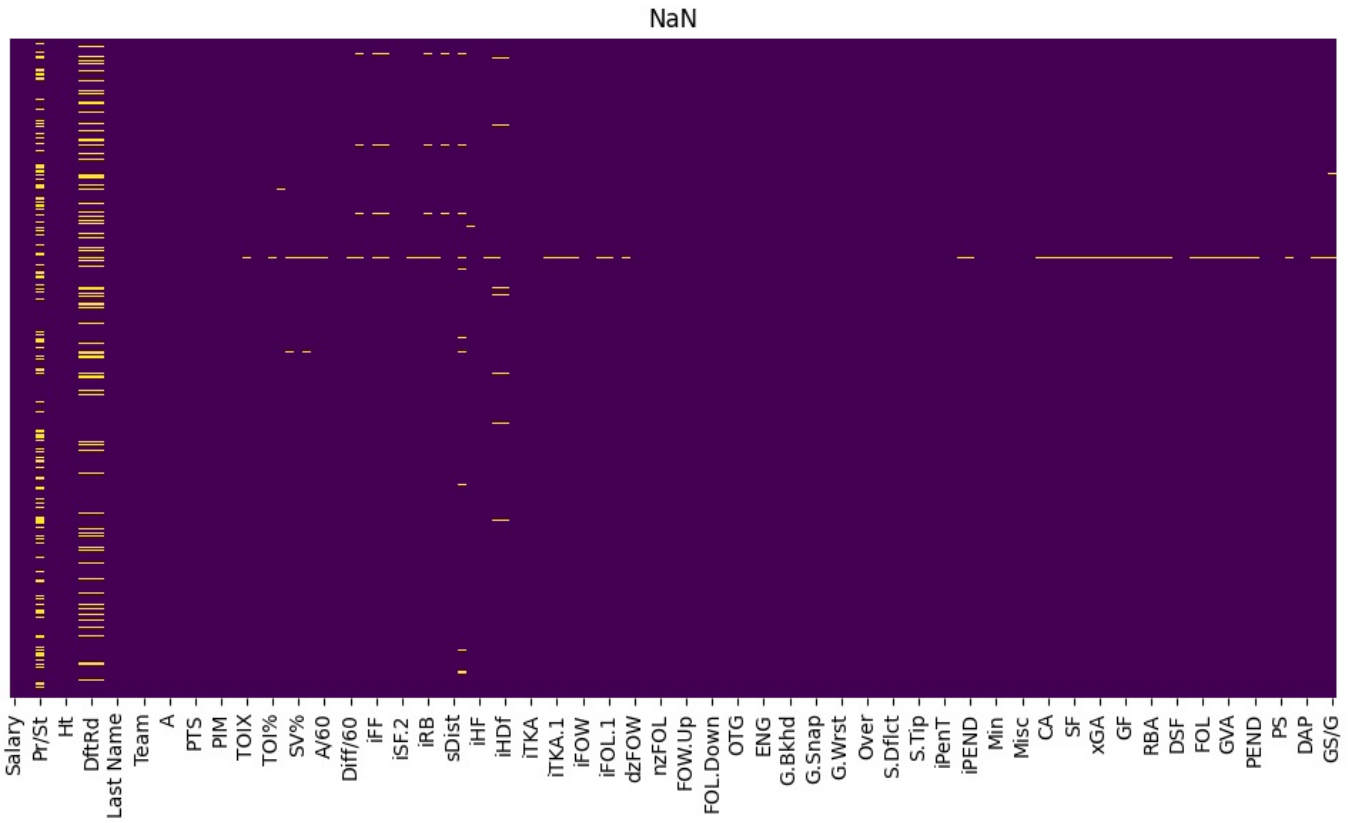
По этим графикам хорошо видно, что модель нормально справляется с игроками, с низкой зарплатой, проблемы возникают с игроками, чья зарплата выбивается из общей массы (у топовых хоккеистов). При это выбросы по предсказанной зарплате тоже вполне объясними, потому что это может быть, например, молодой игрок на контракте новичка (первом контракте), у которого хорошая статистика, на уровне топового хоккеиста, но при этом небольшая зарплата, или наоборот, поскольку в НХЛ заключают долгосрочные контракты, иногда могут быть дорогостоящие хоккеисты, которые состарились или деградировали, но имеют завышенную зарплату

Improved KNN

Загрузим датасет, посмотрим, где есть пропуски.

In [74]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter
from matplotlib import pyplot as plt


path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1_i = pd.read_csv(path + "/train.csv")
df2_i = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")
df2_i['Salary'] = salary['Salary'].values
df2_i = df2_i[df1_i.columns]
df = pd.concat([df1_i, df2_i], ignore_index=True)

nulls = df.isna().sum().sort_values(ascending=False)
```

```
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



Много игроков, которые не были задрафтованы, поэтому в этом столбце много пропусков. Pr/St тоже содержит достаточно пропусков, поскольку это только для Канады и США. В остальных столбцах пропуски постольку поскольку, заполнять их будем медианой.

In [75]:
```
num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
df[num_cols].describe().T
```

Out[75]:

	count	mean	std	min	25%	50%	75%	max
Salary	874.0	2.325289e+06	2.298253e+06	575000.00	742500.00	925000.00	3700000.00	14000000.00
Ht	874.0	7.308238e+01	2.105485e+00	66.00	72.00	73.00	75.00	81.00
Wt	874.0	2.008432e+02	1.506008e+01	157.00	190.00	200.00	210.00	265.00
DftYr	749.0	2.008708e+03	4.380158e+00	1990.00	2006.00	2010.00	2012.00	2016.00
DftRd	749.0	2.742323e+00	1.988358e+00	1.00	1.00	2.00	4.00	9.00
...	...	...	...	...	...	...	...	...
Grit	874.0	1.267815e+02	1.016121e+02	0.00	41.00	114.00	190.00	622.00
DAP	874.0	9.215675e+00	7.815029e+00	0.00	4.60	7.60	12.00	61.00
Pace	873.0	1.089439e+02	8.899877e+00	75.00	104.70	109.20	113.90	175.70
GS	873.0	2.187331e+01	2.198638e+01	-4.30	2.60	15.70	35.40	104.70
GS/G	872.0	3.401606e-01	2.925900e-01	-0.81	0.14	0.31	0.53	1.28

144 rows × 8 columns

Посмотрим теперь на топ кореллирующих столбцов.

In [76]:
```
import pandas as pd
import numpy as np

num_cols = df.select_dtypes(include=["int64", "float64"]).columns

corr_matrix = df[num_cols].corr()

corr_pairs = corr_matrix.unstack().reset_index()
corr_pairs.columns = ['feature_1', 'feature_2', 'correlation']
```

```
corr_pairs = corr_pairs[corr_pairs['feature_1'] < corr_pairs['feature_2']]

corr_pairs = corr_pairs.reindex(
    corr_pairs['correlation'].abs().sort_values(ascending=False).index
)

top40 = corr_pairs.head(40)

display(top40)
upper = np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
upper_corr = corr_matrix.where(upper)
```

	feature_1	feature_2	correlation
4931	iSF.1	iSF.2	0.999996
8267	iFOL	iFOL.1	0.999981
8122	iFOW	iFOW.1	0.999979
6381	iHF	iHF.1	0.999957
2321	TOI	TOIX	0.999952
7398	iBLK	iBLK.1	0.999948
4787	iSF	iSF.2	0.999937
4786	iSF	iSF.1	0.999931
7108	iGVA	iGVA.1	0.999907
7253	iTKA	iTKA.1	0.999843
2611	TOI/GP	TOI/GP.1	0.999289
15807	CF	FF	0.999216
4351	iCF	iCF.1	0.998819
16097	FF	SF	0.998722
10424	FOW.Close	iFOW	0.998613
10426	FOW.Close	iFOW.1	0.998603
16242	FA	SA	0.998366
10571	FOL.Close	iFOL.1	0.998221
10569	FOL.Close	iFOL	0.998215
15952	CA	FA	0.998114
15809	CF	SF	0.997739
15954	CA	SA	0.995524
4642	iFF	iSF.1	0.995443
4643	iFF	iSF.2	0.995427
4641	iFF	iSF	0.995415
17108	SCA	xGA	0.995015
16963	SCF	xGF	0.994495
1002	GP	OTOI	0.993385
13008	Wide	iMiss	0.993229
16244	FA	xGA	0.992949
15956	CA	xGA	0.992542
10136	FOW.Down	iFOW	0.992496
10138	FOW.Down	iFOW.1	0.992464
16099	FF	xGF	0.991109
2177	Shifts	TOIX	0.991098
16532	SA	xGA	0.991094
4496	iCF.1	iFF	0.990960
2899	TOI%	TOI/GP.1	0.990941
10281	FOL.Down	iFOL	0.990924
2898	TOI%	TOI/GP	0.990914

Их довольно-таки много. При том некоторые как-будто повторяются. Почистим данные от корреляций с некоторым барьером.

In [77]:
```
BARIER = 0.9998

to_drop = [col for col in upper_corr.columns if any(upper_corr[col] > BARIER)]
df_clean_i = df.drop(columns=to_drop)
df_clean_i.head()
```

Out[77]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	1(
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2(

5 rows × 145 columns

Займемся фича инжинирингом. Выделим возраст хоккеиста на начало сезона по тому, когда он родился. Добавим возраст в бакеты, также добавим в бакеты количество набранных очков. Добавим опыт, то есть сколько времени прошло с драфта. Дропнем бесполезные фичи по типу имени, фамилии, национальности (ее дублирует страна рождения), штата.

In [78]:
```
df_features_i = df_clean_i.copy()

df_features_i['Born'] = pd.to_datetime(df_features_i['Born'], format='%y-%m-%d')
reference_date = pd.Timestamp('2016-10-01')
df_features_i['Age'] = (reference_date - df_features_i['Born']).dt.days / 365.25
df_features_i['Experience'] = reference_date.year - df_features_i['DftYr']

df_features_i["Age_bucket"] = pd.cut(
    df_features_i["Age"],
    bins=[0, 20, 25, 30, 35, 40, 45],
    labels=["<20", "20-34", "25-29", "30-34", "35-39", "40+"],
)
df_features_i["PTS_bucket"] = pd.cut(
    df_features_i["PTS"],
    bins=[-1, 20, 40, 60, 80, 100, 120, 140, 160],
    labels=["<20", "20-39", "40-59", "60-79", "80-99", "100-119", "120-139", "140-159"],
)

features_to_drop = ['Born', 'Last Name', 'First Name', 'Nat', 'Pr/St']
df_features_i = df_features_i.drop(features_to_drop, axis=1)
df_features_i['Match'].value_counts()
```

Out[78]:
```
Match
0    870
1      4
Name: count, dtype: int64
```

In [79]:
```
has_nan = df_features_i.isnull().any()

columns_with_nan = has_nan[has_nan].index.tolist()

print("Столбцы, содержащие хотя бы один NaN:")
columns_with_nan
```

```
Столбцы, содержащие хотя бы один NaN:
```

Out[79]:
```
['DftYr',
 'DftRd',
 'Ovrl',
 'TOI%',
 'IPP%',
 'SH%',
 'SV%',
 'PDO',
 'F/60',
 'A/60',
 'Diff/60',
 'iCF',
 'iFF',
 'iSF',

 'ixG',
 'iSCF',
 'iRB',
 'iRS',
 'iDS',
 'sDist.1',
 'Pass',
 'iHA',
 'iHDf',
 'BLK%',
 '%FOT',
 'iPENT',
 'iPEND',
 'CF',
 'CA',
 'FF',
 'FA',
 'SF',
 'SA',
 'xGF',
 'xGA',
 'SCF',
 'SCA',
 'GF',
 'GA',
 'RBF',
 'RBA',
 'RSF',
 'RSA',
 'FOW',
 'FOL',
 'HF',
 'HA',
 'GVA',
 'TKA',
 'PENT',
 'PEND',
 'OTOI',
 'Pace',
 'GS',
 'GS/G',
 'Experience']
```

In [80]:
```
from scipy.stats.mstats import winsorize

TARGET_NAME = 'Salary'

X_i = df_features_i.drop(TARGET_NAME, axis=1)
y_i = df_features_i[TARGET_NAME]

X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(
    X_i, y_i, test_size=0.2, random_state=42
)
```

Я разделил столбцы на те, что по распределению напоминают нормальное распределение и все остальные. Для каждого из них в сетке будет выбран скалер. Пустые данные я заполнил KNNImputerом (почему бы и нет). One hot'ом закодировал категориальные колонки, но сделал это так, чтобы сильно не размножались данные.

В сетке поискал различные расстояния, скалеры, соседей, а также логарифмировал целевую переменную, потому что иначе из-за выбросов всё сломается.

In [81]:
```
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.compose import ColumnTransformer, TransformedTargetRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
import numpy as np
import pandas as pd

normal_candidates = ["Ht", "Wt", "+/-", "E+/-", "TOI%", "SH%", "SV%", "F/60",
                     "A/60", "Diff/60", "iHDf", "iPenDf", "NPD"]

cat_cols = X_i.select_dtypes(include=['object', 'category']).columns.tolist()

excluded_cols = cat_cols
normal_cols = [c for c in normal_candidates if c in X.columns and c not in excluded_cols]
power_cols = [c for c in X_i.columns if c not in normal_cols]

all_processed_cols = normal_cols + cat_cols
power_cols = [c for c in X_i.columns if c not in all_processed_cols]
power_cols = [c for c in power_cols if c in X_i.select_dtypes(include=np.number).columns]

cat_branch = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ("onehot", OneHotEncoder(
        handle_unknown='infrequent_if_exist',
        sparse_output=False,
        min_frequency=0.05
    ))
])

poly_branch = Pipeline([
    ('imputer', KNNImputer(n_neighbors=7)),
    ('scaler', StandardScaler())
])

normal_branch = Pipeline([
    ('imputer', KNNImputer(n_neighbors=7)),
    ('scaler', StandardScaler())
])

power_branch = Pipeline([
    ('imputer', KNNImputer(n_neighbors=7)),
    ('scaler', StandardScaler())
])

ct = ColumnTransformer(
    transformers=[
        ("cat_proc", cat_branch, cat_cols),
        ("normal_proc", normal_branch, normal_cols),
        ("power_proc", power_branch, power_cols),
    ], remainder='drop'
)
```

In [82]:

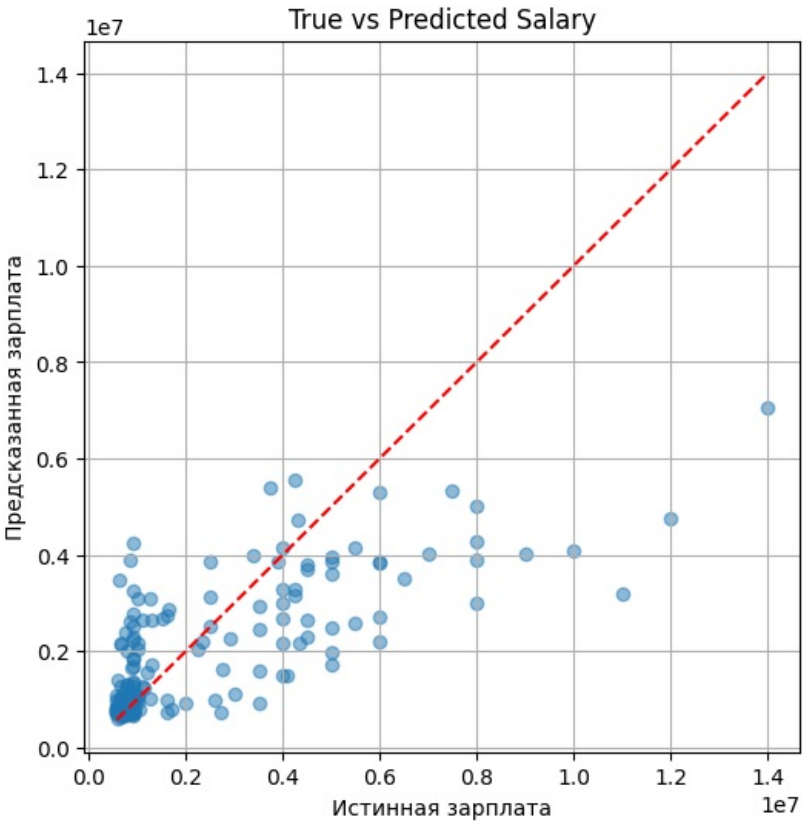
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import QuantileTransformer, StandardScaler, RobustScaler, PowerTransformer, MinMaxSc
from sklearn.neighbors import KNeighborsRegressor

from sklearn.pipeline import Pipeline

full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=KNeighborsRegressor(),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'model_wrapper__regressor__n_neighbors': [3, 5, 7, 13, 25, 41, 51],
    'model_wrapper__regressor__metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski'],
    'preprocessor__normal_proc__scaler': [
        StandardScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
    ],
    'preprocessor__power_proc__scaler': [
        RobustScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
    ]
}

grid = RandomizedSearchCV(
    full_pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
```

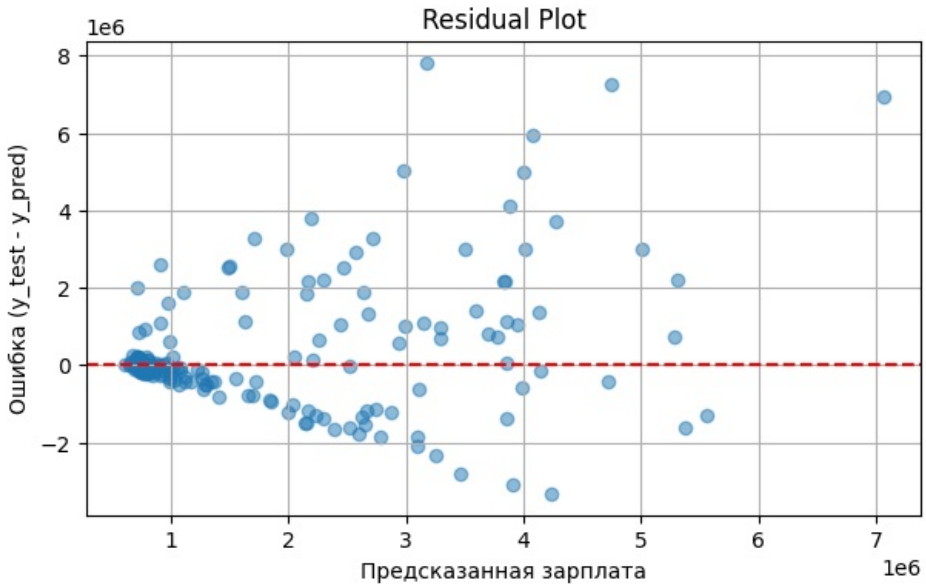
```
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)
print(grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
mae = mean_absolute_error(y_test_i, y_pred)
rmse = np.sqrt(mean_squared_error(y_test_i, y_pred))
r2 = r2_score(y_test_i, y_pred)
base_mean_error = mean_absolute_error(y_test_i, [y_train.mean()]*len(y_test_i))

print(f"--- Improved KNN ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

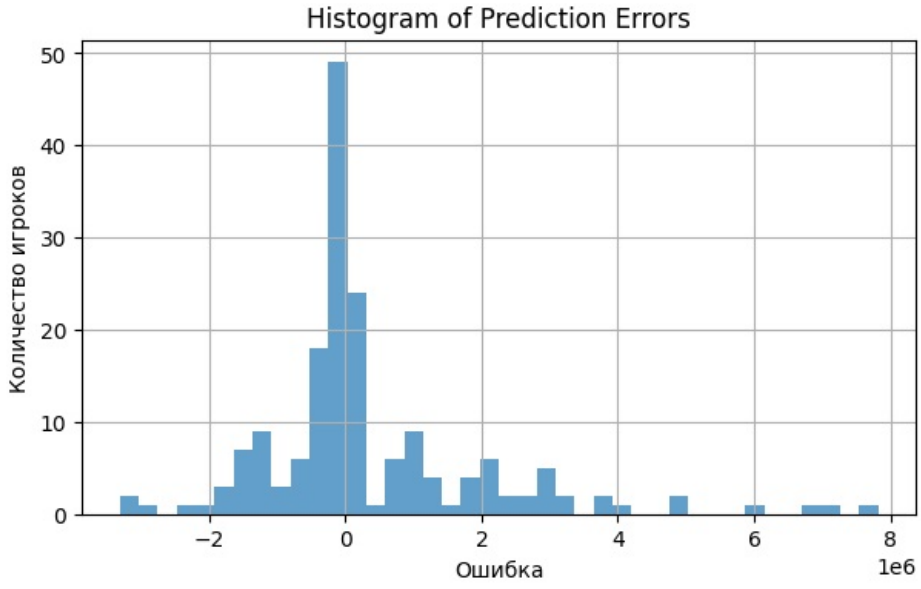
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'preprocessor__power_proc__scaler': PowerTransformer(), 'preprocessor__normal_proc__scaler': QuantileTransforme
r(n_quantiles=400), 'model_wrapper__regressor__n_neighbors': 13, 'model_wrapper__regressor__metric': 'manhattan'
}
--- Improved KNN ---
MAE (Ошибка в долларах): 1062416.379
R2 Score: 0.495
```

Ошибка упала, R2 вырос на 7 пунктов. Это хорошо.

In [83]:
```
true_vs_predicted(y_test_i, y_pred)
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```







Модель склонна занижать зарплату, мы видим левое смещение. Улучшенная версия стала лучше работать с игроками, чья зарплата на низком уровне.

Интересно посмотреть на то, каких игроков модель считает недооцененными и переоцененными и совместить с реальностью.

In [84]:
```
original_float_format = pd.get_option('display.float_format')

pd.set_option('display.float_format', '{:.2f}'.format)

best_model = grid.best_estimator_

y_full_pred = best_model.predict(X_i)

results_df = pd.DataFrame({
```

```
    'True_Salary': y_i,
    'Predicted_Salary': y_full_pred
})

results_df['Absolute_Error'] = np.abs(results_df['True_Salary'] - results_df['Predicted_Salary'])

test_indices = X_test_i.index
final_results = pd.merge(
    df[['First Name', 'Last Name']],
    results_df,
    left_index=True,
    right_index=True
)

final_results_sorted = final_results.sort_values(by='Absolute_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---")
print(final_results_sorted.head(10).to_string())

pd.set_option('display.float_format', original_float_format)
```

```
--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---
    First Name Last Name  True_Salary  Predicted_Salary  Absolute_Error
103    Patrick      Kane  13800000.00        5329770.03      8470229.97
817     Steven   Stamkos   9500000.00        1117478.18      8382521.82
208       P.K.    Subban  11000000.00        3177578.26      7822421.74
496   Jonathan     Toews  13800000.00        6027317.60      7772682.40
542       Shea     Weber  12000000.00        4753186.47      7246813.53
626       Anze   Kopitar  14000000.00        7061935.92      6938064.08
837       Ryan  O'Reilly  11000000.00        4655718.11      6344281.89
145     Sidney    Crosby  10900000.00        4788136.07      6111863.93
861      Corey     Perry  10000000.00        4025666.48      5974333.52
260       Alex  Ovechkin  10000000.00        4075099.36      5924900.64
```

У Джонатана Тейвза был не самый удачный сезон: небольшой +/-, не так много очков как раньше. Ну и в целом модель хуже справляется с хоккеистами, у которых высокая зарплата. Она склонна занижать.

In [85]:
```
import pandas as pd
final_results['Signed_Error'] = final_results['Predicted_Salary'] - final_results['True_Salary']

overestimated_players = final_results[final_results['Signed_Error'] > 0].copy()

top_overestimated = overestimated_players.sort_values(by='Signed_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---")

print(top_overestimated[['First Name', 'Last Name',
                         'True_Salary', 'Predicted_Salary', 'Signed_Error']].head(10).to_string())
```

```
--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---
    First Name     Last Name  True_Salary  Predicted_Salary  Signed_Error
525       Leon     Draisaitl     925000.0      4.947893e+06  4.022893e+06
382       Jack        Eichel     925000.0      4.240490e+06  3.315490e+06
775     Connor       McDavid     925000.0      4.132088e+06  3.207088e+06
830    Patrick         Eaves    1000000.0      4.187564e+06  3.187564e+06
472   Jonathan  Marchessault     750000.0      3.837899e+06  3.087899e+06
393         Bo        Horvat     832500.0      3.905027e+06  3.072527e+06
477     Connor         Brown     650000.0      3.648355e+06  2.998355e+06
443     Nikita       Zaitsev     925000.0      3.892236e+06  2.967236e+06
569    Nikolaj         Ehlers     925000.0      3.782880e+06  2.857880e+06
114     Auston      Matthews     925000.0      3.768449e+06  2.843449e+06
```

А тут мы видим будущих звёзд НХЛ на небольших контрактах новичка: Маршессо, Эйлерс. Ветеран Эрик Стаал на небольшом контракте, но с хорошей статистикой.

My implementation

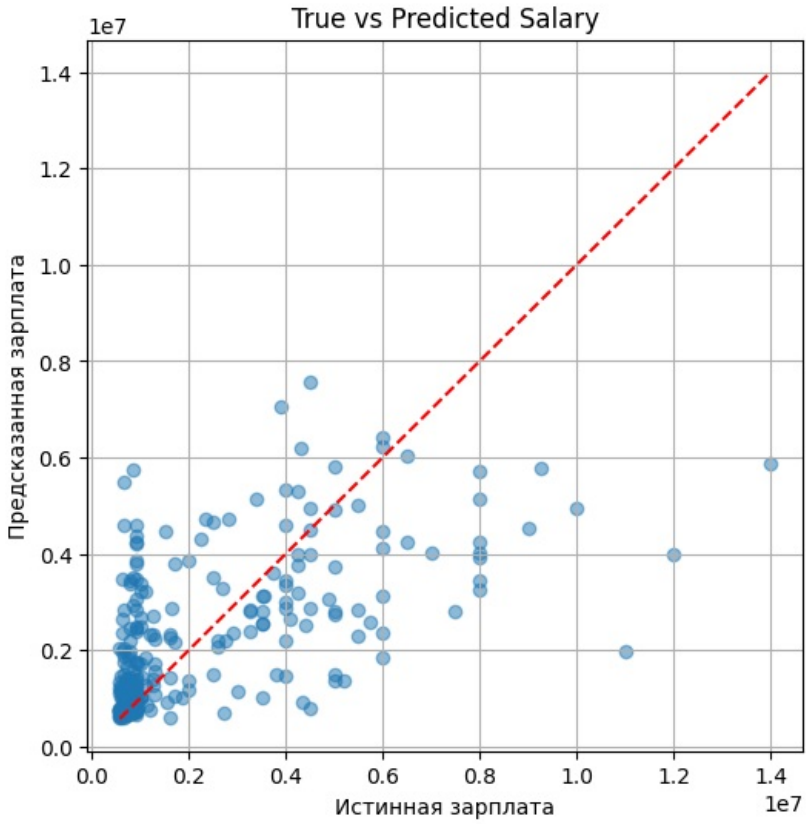
In [86]:
```
import numpy as np
from sklearn.base import RegressorMixin
from sklearn.utils.validation import check_X_y, check_is_fitted, check_array

class MyKNNRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, n_neighbors=5, metric="euclidean", weights="uniform"):
        self.n_neighbors = n_neighbors
        self.metric = metric
        self.weights = weights

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.X_train_ = X
        self.y_train_ = y
        self.n_features_in_ = X.shape[1]
```

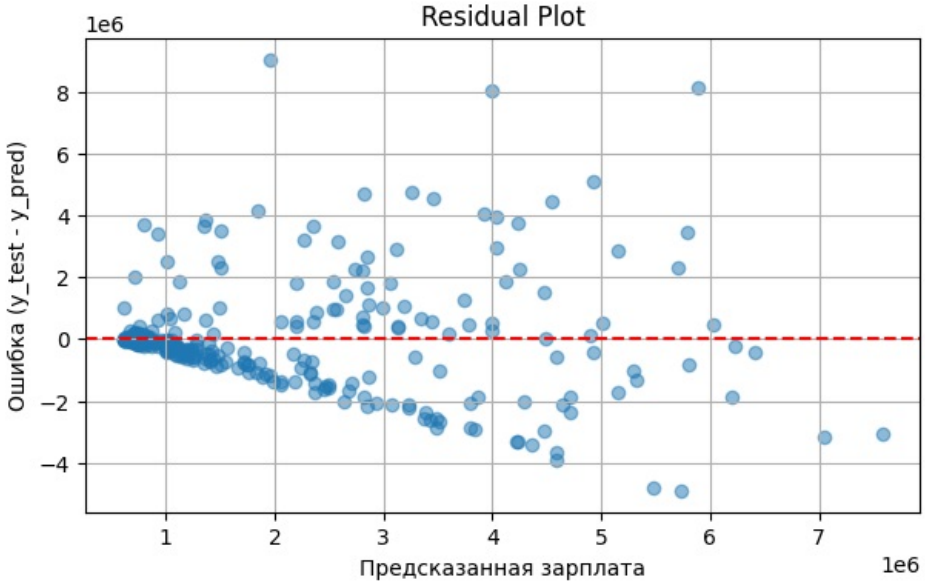
```
        return self

    def _distance(self, a, b):
        if self.metric == "euclidean":
            return np.sqrt(np.sum((a - b) ** 2))
        elif self.metric == "manhattan":
            return np.sum(np.abs(a - b))
        elif self.metric == "chebyshev":
            return np.max(np.abs(a - b))
        else:
            raise ValueError(f"Unknown metric: {self.metric}")

    def _get_neighbors(self, x):
        distances = np.array([self._distance(x, x_train) for x_train in self.X_train_])
        idx = distances.argsort()[:self.n_neighbors]
        return idx, distances[idx]

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)
        predictions = []
        for x in X:
            idx, dists = self._get_neighbors(x)
            neighbors_y = self.y_train_[idx]
            if self.weights == "uniform":
                pred = np.mean(neighbors_y)
            elif self.weights == "distance":
                dists = np.where(dists == 0, 1e-8, dists)
                w = 1 / dists
                pred = np.sum(w * neighbors_y) / np.sum(w)
            else:
                raise ValueError("Unknown weights mode")
            predictions.append(pred)
        return np.array(predictions)
```

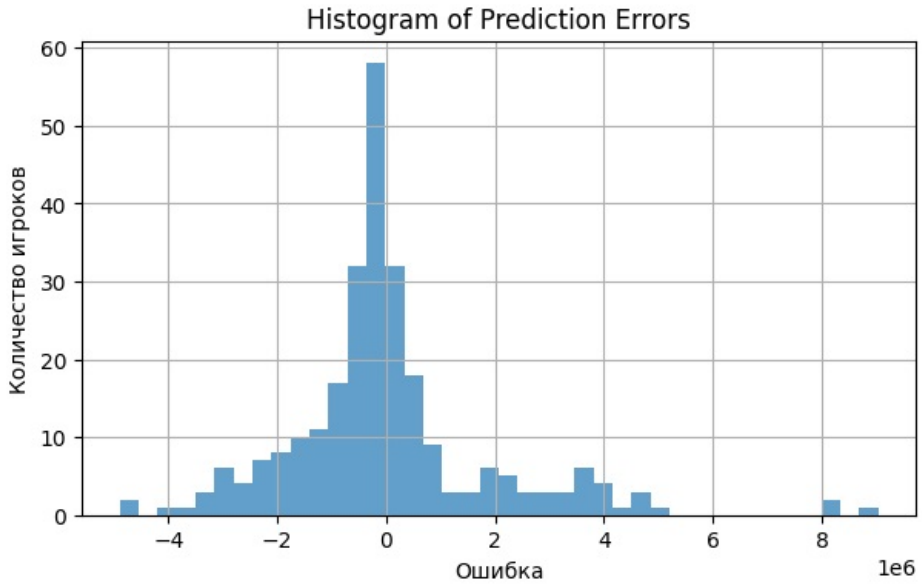
In [87]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
knn = MyKNNRegressor()
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, [y_train.mean()]*len(y_test))

print(f"--- Baseline My KNN ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```

```
--- Baseline My KNN ---
MAE (Ошибка в долларах): 1217521.047
R2 Score: 0.357
```


True vs Predicted Salary
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Предсказанная зарплата
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e7
Истинная зарплата

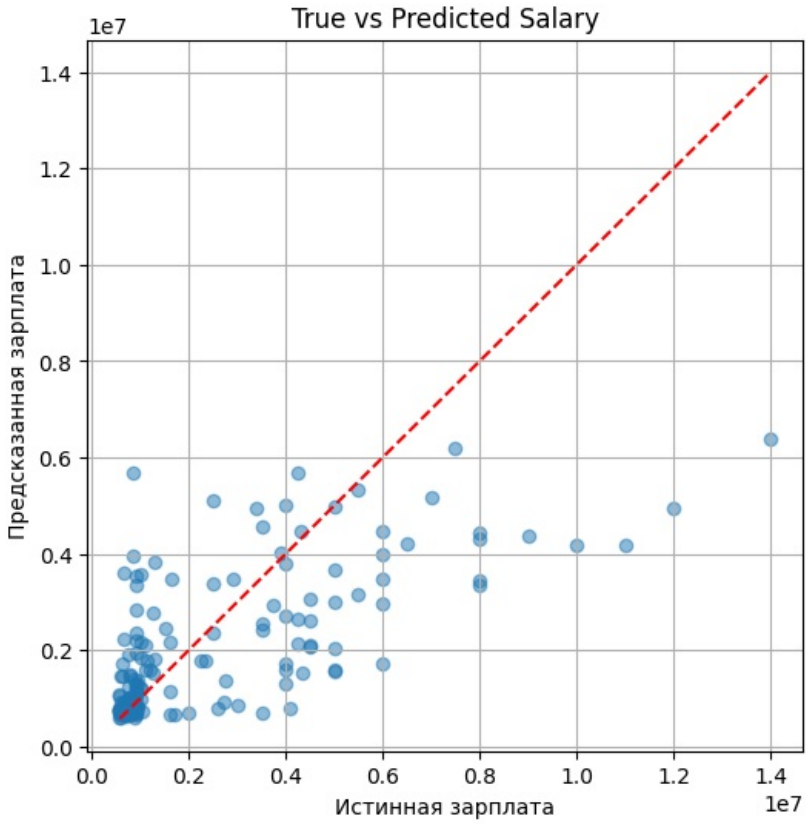

Residual Plot
1e6
8
6
4
2
0
−2
−4
Ошибка (y_test - y_pred)
1
2
3
4
5
6
7
1e6
Предсказанная зарплата

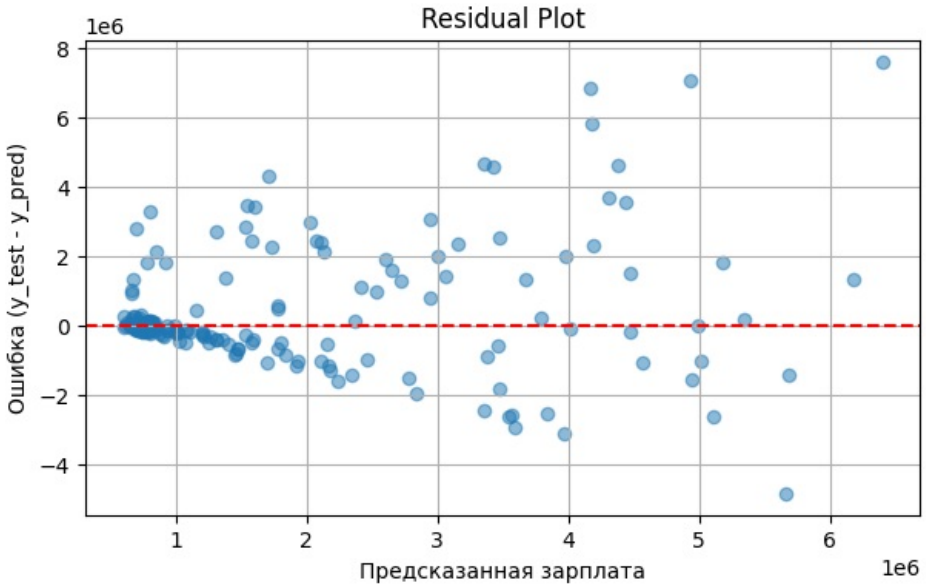


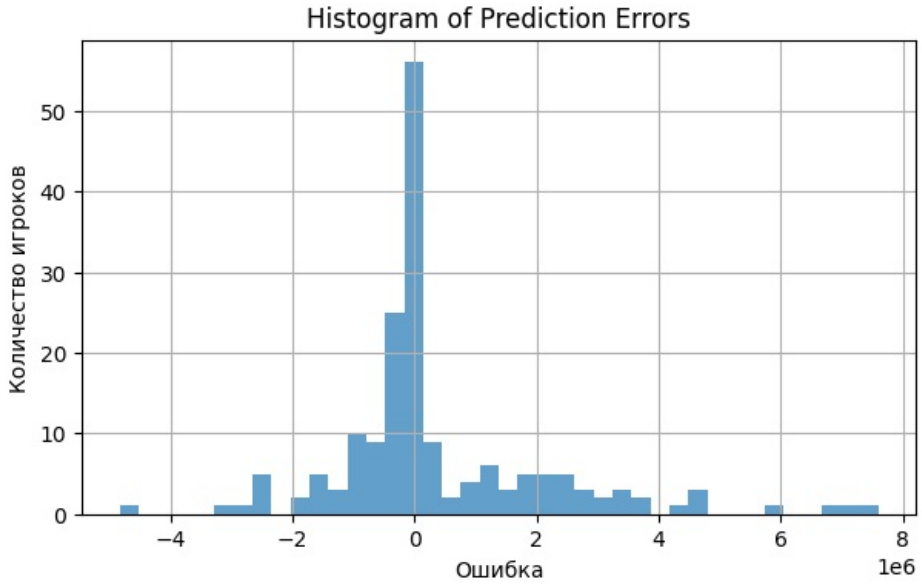
In [88]:

```python
from sklearn.preprocessing import QuantileTransformer, StandardScaler, RobustScaler, PowerTransformer, MinMaxSc
from sklearn.neighbors import KNeighborsRegressor

from sklearn.pipeline import Pipeline

full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=MyKNNRegressor(),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'model_wrapper__regressor__n_neighbors': [3, 5, 7, 13, 25, 41, 51],
    'model_wrapper__regressor__metric': ['euclidean', 'manhattan', 'chebyshev'],
    'preprocessor__normal_proc__scaler': [
        StandardScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
    ],
    'preprocessor__power_proc__scaler': [
        RobustScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
    ]
}

grid = RandomizedSearchCV(
    full_pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)
print(grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
mae = mean_absolute_error(y_test_i, y_pred)
rmse = np.sqrt(mean_squared_error(y_test_i, y_pred))
r2 = r2_score(y_test_i, y_pred)
base_mean_error = mean_absolute_error(y_test_i, [y_train.mean()]*len(y_test_i))

print(f"--- MyKNN ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
true_vs_predicted(y_test_i, y_pred)
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'preprocessor__power_proc__scaler': QuantileTransformer(n_quantiles=400), 'preprocessor__normal_proc__scaler':
QuantileTransformer(n_quantiles=400), 'model_wrapper__regressor__n_neighbors': 5, 'model_wrapper__regressor__met
ric': 'euclidean'}
--- MyKNN ---
MAE (Ошибка в долларах): 1105419.509
R2 Score: 0.458
```







Собственная имплементация показывает те же самые результаты, что и встроенная в sklearn, только обучение идёт несколько медленнее из-за отсутствия оптимизаций.

	Base KNN	Improved KNN	My KNN
R2	0.399	0.474	0.474
MAE	1173197.566	1027804.467	1027804.467

Вывод

KNN очень неплохо себя показала в задаче классфикации. Даже несмотря на дисбаланс классов этому методу удалось достичь гроссмейстерских 0.85 ROC_AUC. Учитывая, насколько простой это алгоритм, это очень хороший результат.

С задачей регрессии чуть сложнее, поскольку датасет имеет очень много признаков, а также сам по себе так составлен, что ошибка большая, KNN не удалось показать хороший результат. Но так или иначе это то, от чего мы будем отталкиваться.

Регрессия

Задача: предсказать зарплату игрока НХЛ по его статистике с использованием модели линейной регрессии.

Будем использовать MAE, так как она очень наглядная, будет легко понять, насколько долларов ошибается модель. В качестве дополнительной метрики будем использовать R^2, чтобы смотреть насколько отличается предсказание от среднего.

Baseline

Скачаем датасет с kaggle

In [1]:
```
import kagglehub
import pandas as pd
import numpy as np
from kagglehub import KaggleDatasetAdapter


path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")

df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)
df.head()
```

```
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: I
Progress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_i
nstall.html
  from .autonotebook import tqdm as notebook_tqdm
```

Out[1]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	1
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2

5 rows × 154 columns

Сделаем минимальные телодвижения, чтобы регрессор заработал: закодируем все строковые данные, уберем NaN

In [2]:
```
df['Born'] = pd.to_datetime(df['Born'], format='%y-%m-%d')
df_clean = df.fillna(0)
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean = df_clean.drop("Born", axis=1)
```

In [3]:
```
from sklearn.model_selection import train_test_split

TARGET_NAME = "Salary"
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

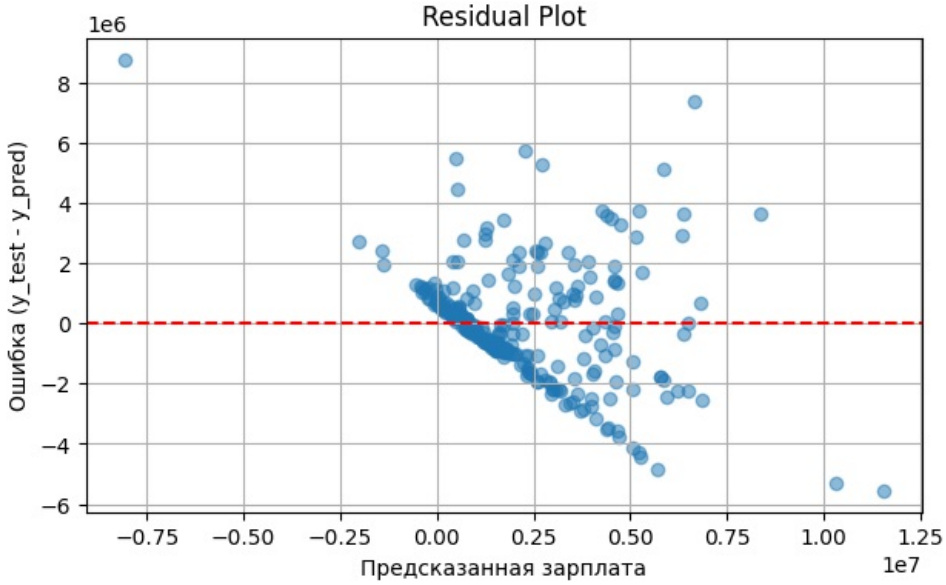
In [4]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import seaborn as sns
```

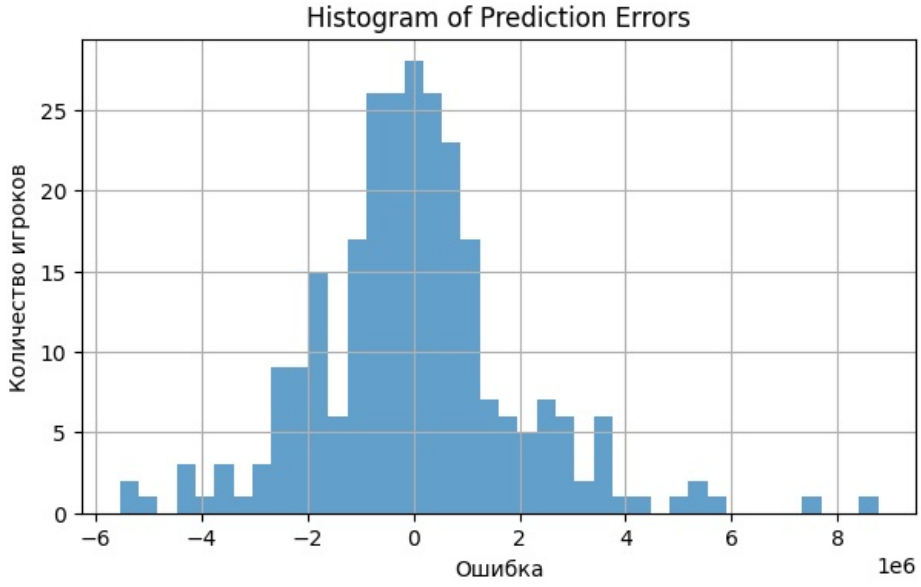
```
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, [y_train.mean()]*len(y_test))

print(f"--- Baseline LinearRegreesion ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
--- Baseline LinearRegreesion ---
MAE (Ошибка в долларах): 1381895.059
R2 Score: 0.324
```

Базовый линейный регрессор выдал результаты хуже, чем KNN. В целом может и логично, потому что похожие игроки в НХЛ по качеству должны зарабатывать примерно одинаково, что буквально есть KNN. Попробуем оптимизировать.

In [5]:
```
from matplotlib import pyplot as plt

def true_vs_predicted(y_test, y_pred):
    plt.figure(figsize=(6,6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
    plt.xlabel("Истинная зарплата")
    plt.ylabel("Предсказанная зарплата")
    plt.title("True vs Predicted Salary")
    plt.grid(True)
    plt.show()

def resudial(y_test, y_pred):
    residuals = y_test - y_pred
    plt.figure(figsize=(7,4))
    plt.scatter(y_pred, residuals, alpha=0.5)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel("Предсказанная зарплата")
    plt.ylabel("Ошибка (y_test - y_pred)")
    plt.title("Residual Plot")
    plt.grid(True)
    plt.show()
def histogramm_error(y_test, y_pred):
    plt.figure(figsize=(7,4))
    residuals = y_test - y_pred
    plt.hist(residuals, bins=40, alpha=0.7)
    plt.title("Histogram of Prediction Errors")
    plt.xlabel("Ошибка")
    plt.ylabel("Количество игроков")
    plt.grid(True)
    plt.show()
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```

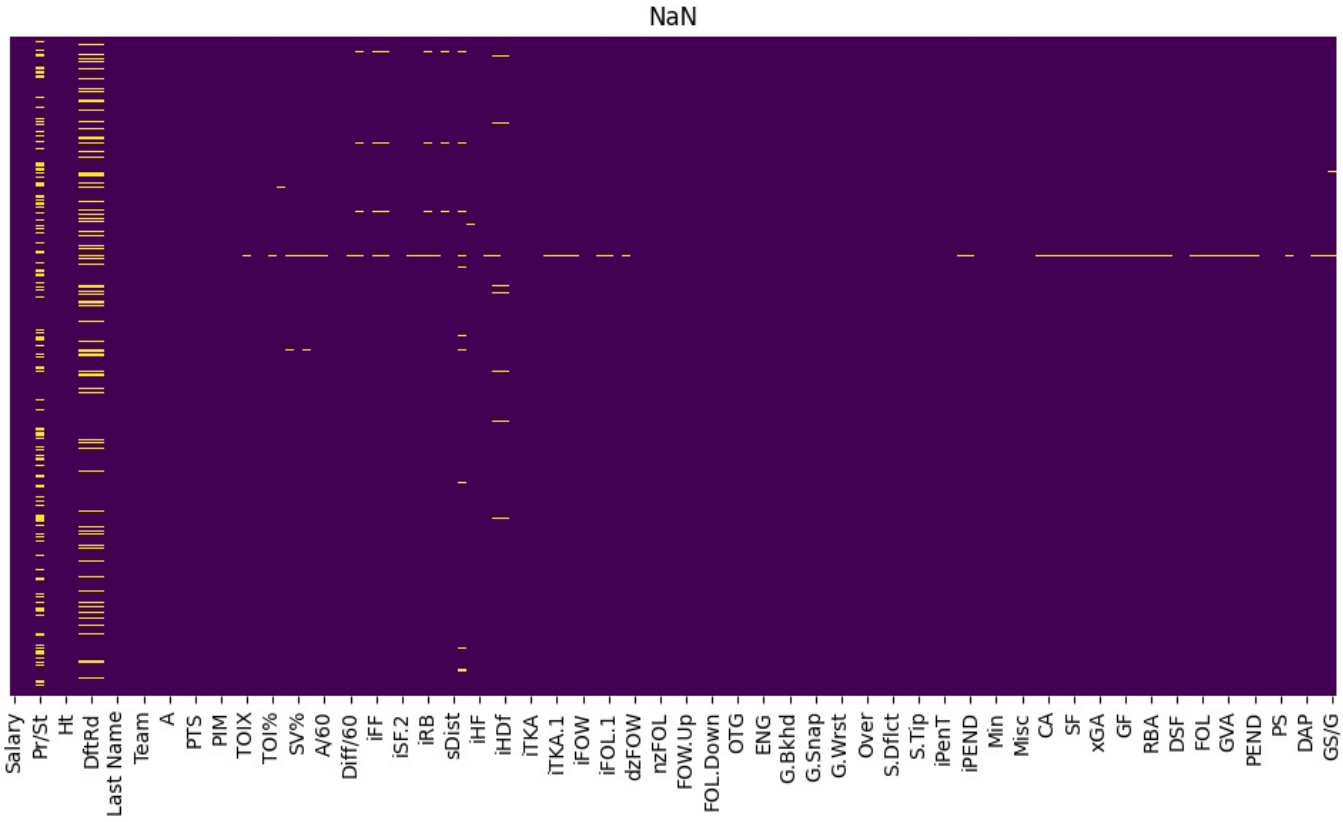




По графикам видно, что регрессору удалось даже предсказать отрицательную зарплату! Это круто, как же плохо надо играть?

Improved Linear Regression

In [6]:
```
import kagglehub
from matplotlib import pyplot as plt
import pandas as pd


path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")
df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [7]:
```
num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
df[num_cols].describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Salary	874.0	2.325289e+06	2.298253e+06	575000.00	742500.00	925000.00	3700000.00	14000000.00
Ht	874.0	7.308238e+01	2.105485e+00	66.00	72.00	73.00	75.00	81.00
Wt	874.0	2.008432e+02	1.506008e+01	157.00	190.00	200.00	210.00	265.00
DftYr	749.0	2.008708e+03	4.380158e+00	1990.00	2006.00	2010.00	2012.00	2016.00
DftRd	749.0	2.742323e+00	1.988358e+00	1.00	1.00	2.00	4.00	9.00
...	...	...	...	...	...	...	...	...
Grit	874.0	1.267815e+02	1.016121e+02	0.00	41.00	114.00	190.00	622.00
DAP	874.0	9.215675e+00	7.815029e+00	0.00	4.60	7.60	12.00	61.00
Pace	873.0	1.089439e+02	8.899877e+00	75.00	104.70	109.20	113.90	175.70
GS	873.0	2.187331e+01	2.198638e+01	-4.30	2.60	15.70	35.40	104.70
GS/G	872.0	3.401606e-01	2.925900e-01	-0.81	0.14	0.31	0.53	1.28

144 rows × 8 columns

In [8]:

```
import pandas as pd
import numpy as np

num_cols = df.select_dtypes(include=["int64", "float64"]).columns

corr_matrix = df[num_cols].corr()

corr_pairs = corr_matrix.unstack().reset_index()
corr_pairs.columns = ['feature_1', 'feature_2', 'correlation']

corr_pairs = corr_pairs[corr_pairs['feature_1'] < corr_pairs['feature_2']]

corr_pairs = corr_pairs.reindex(
    corr_pairs['correlation'].abs().sort_values(ascending=False).index
)

top40 = corr_pairs.head(40)

display(top40)
upper = np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
upper_corr = corr_matrix.where(upper)
```

	feature_1	feature_2	correlation
4931	iSF.1	iSF.2	0.999996
8267	iFOL	iFOL.1	0.999981
8122	iFOW	iFOW.1	0.999979
6381	iHF	iHF.1	0.999957
2321	TOI	TOIX	0.999952
7398	iBLK	iBLK.1	0.999948
4787	iSF	iSF.2	0.999937
4786	iSF	iSF.1	0.999931
7108	iGVA	iGVA.1	0.999907
7253	iTKA	iTKA.1	0.999843
2611	TOI/GP	TOI/GP.1	0.999289
15807	CF	FF	0.999216
4351	iCF	iCF.1	0.998819
16097	FF	SF	0.998722
10424	FOW.Close	iFOW	0.998613
10426	FOW.Close	iFOW.1	0.998603
16242	FA	SA	0.998366
10571	FOL.Close	iFOL.1	0.998221
10569	FOL.Close	iFOL	0.998215
15952	CA	FA	0.998114
15809	CF	SF	0.997739
15954	CA	SA	0.995524
4642	iFF	iSF.1	0.995443
4643	iFF	iSF.2	0.995427
4641	iFF	iSF	0.995415
17108	SCA	xGA	0.995015
16963	SCF	xGF	0.994495
1002	GP	OTOI	0.993385
13008	Wide	iMiss	0.993229
16244	FA	xGA	0.992949
15956	CA	xGA	0.992542
10136	FOW.Down	iFOW	0.992496
10138	FOW.Down	iFOW.1	0.992464
16099	FF	xGF	0.991109
2177	Shifts	TOIX	0.991098
16532	SA	xGA	0.991094
4496	iCF.1	iFF	0.990960
2899	TOI%	TOI/GP.1	0.990941
10281	FOL.Down	iFOL	0.990924
2898	TOI%	TOI/GP	0.990914

In [9]:
```
BARIER = 0.9998

to_drop = [col for col in upper_corr.columns if any(upper_corr[col] > BARIER)]
df_clean = df.drop(columns=to_drop)
df_clean.head()
```

Out[9]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	1
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2

5 rows × 145 columns

Тут я пытался добавить дополнительные признаки, но они плохо работали. Поэтому я их закомментировал.

In [10]:
```
df_features = df_clean.copy()

df_features['Born'] = pd.to_datetime(df_features['Born'], format='%y-%m-%d')
reference_date = pd.Timestamp('2016-10-01')
df_features['Age'] = (reference_date - df_features['Born']).dt.days / 365.25

df_features["Age_bucket"] = pd.cut(
    df_features["Age"],
    bins=[0, 20, 25, 30, 35, 40, 45],
    labels=["<20", "20-34", "25-29", "30-34", "35-39", "40+"],
)
df_features["PTS_bucket"] = pd.cut(
    df_features["PTS"],
    bins=[-1, 20, 40, 60, 80, 100, 120, 140, 160],
    labels=["<20", "20-39", "40-59", "60-79", "80-99", "100-119", "120-139", "140-159"],
)
# df_features['G_per_GP'] = np.where(
#     df_features['GP'] != 0,
#     df_features['G'] / df_features['GP'],
#     0
# )
# df_features['A_per_GP'] = np.where(
#     df_features['GP'] != 0,
#     df_features['A'] / df_features['GP'],
#     0
# )
# df_features['PTS_per_GP'] = np.where(
#     df_features['GP'] != 0,
#     df_features['PTS'] / df_features['GP'],
#     0
# )

df_features['Experience'] = reference_date.year - df_features['DftYr']
# df_features['Is_Drafted'] = df_features['DftYr'].notna().astype(int)
# df_features['Physical_Impact'] = df_features['Wt'] * df_features['Ht']

features_to_drop = ['Born', 'Last Name', 'First Name', 'Nat', 'Pr/St', 'City']
df_features = df_features.drop(features_to_drop, axis=1)
df_features['Match'].value_counts()
```

Out[10]:
```
Match
0    870
1      4
Name: count, dtype: int64
```

In [11]:
```
has_nan = df_features.isnull().any()

columns_with_nan = has_nan[has_nan].index.tolist()

print("Столбцы, содержащие хотя бы один NaN:")
columns_with_nan
```

```
Столбцы, содержащие хотя бы один NaN:
```

Out[11]:
```
['DftYr',
 'DftRd',
 'Ovrl',
 'TOI%',
 'IPP%',
 'SH%',
 'SV%',
 'PDO',
 'F/60',
 'A/60',
 'Diff/60',
 'iCF',
 'iFF',
 'iSF',
 'ixG',
 'iSCF',
 'iRB',
 'iRS',
 'iDS',
 'sDist.1',
 'Pass',
 'iHA',
 'iHDf',
 'BLK%',
 '%FOT',
 'iPENT',
 'iPEND',
 'CF',
 'CA',
 'FF',
 'FA',
 'SF',
 'SA',
 'xGF',
 'xGA',
 'SCF',
 'SCA',
 'GF',
 'GA',
 'RBF',
 'RBA',
 'RSF',
 'RSA',
 'FOW',
 'FOL',
 'HF',
 'HA',
 'GVA',
 'TKA',
 'PENT',
 'PEND',
 'OTOI',
 'Pace',
 'GS',
 'GS/G',
 'Experience']
```

Дополнительно обработаем выбросы по игрокам. Используем для этого winsorize, который заменит выбросы на 99 перцентиль.

In [12]:
```
from scipy.stats.mstats import winsorize


TARGET_NAME = 'Salary'

X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]

X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(
    X_i, y_i, test_size=0.3, random_state=42
)
y_train_i = winsorize(y_train_i, limits=[0.01, 0.01])
```

In [13]:
```
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.preprocessing import OneHotEncoder, QuantileTransformer, StandardScaler, RobustScaler, PowerTransf
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import HuberRegressor
from sklearn.compose import ColumnTransformer, TransformedTargetRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import pandas as pd
```

```
normal_candidates = ["Ht", "Wt", "+/-", "E+/-", "TOI%", "SH%", "SV%", "F/60",
                     "A/60", "Diff/60", "iHDf", "iPenDf", "NPD"]

cat_cols = X_i.select_dtypes(include=['object', 'category']).columns.tolist()

excluded_cols = cat_cols
normal_cols = [c for c in normal_candidates if c in X_i.columns and c not in excluded_cols]
power_cols = [c for c in X_i.columns if c not in normal_cols]

all_processed_cols = normal_cols + cat_cols
power_cols = [c for c in X_i.columns if c not in all_processed_cols]
power_cols = [c for c in power_cols if c in X_i.select_dtypes(include=np.number).columns]

cat_branch = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='infrequent_if_exist',
    sparse_output=False,
    min_frequency=0.02))
])

normal_branch = Pipeline([
    ('imputer', KNNImputer(n_neighbors=7)),
    ('scaler', StandardScaler())
])

power_branch = Pipeline([
    ('imputer', KNNImputer(n_neighbors=7)),
    ('scaler', PowerTransformer())
])

ct = ColumnTransformer(
    transformers=[
        ("cat_proc", cat_branch, cat_cols),
        ("normal_proc", normal_branch, normal_cols),
        ("power_proc", power_branch, power_cols),
    ], remainder='drop'
)
```

Тут почти всё тоже самое, что и в KNN, различие только в модели и параметрах для GridSearch. Лучше всего себя из моделей линейной регрессии показал HuberRegressor, он хорошо справляется с выбросами, имеет регуляризацию.

In [14]:

```
full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=HuberRegressor(max_iter=5000),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'preprocessor__normal_proc__scaler': [
        RobustScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
    ],
    'preprocessor__power_proc__scaler': [
        RobustScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
    ],
    'model_wrapper__regressor__alpha': [25.0, 50.0, 75.0,],
    'model_wrapper__regressor__epsilon': [1.0, 1.5]
}

grid = GridSearchCV(
    full_pipeline,
    param_grid,
    cv=3,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print(grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)

mae = mean_absolute_error(y_test_i, y_pred)
y_pred_train = grid.best_estimator_.predict(X_train_i)
train_mae = mean_absolute_error(y_train_i, y_pred_train)
```

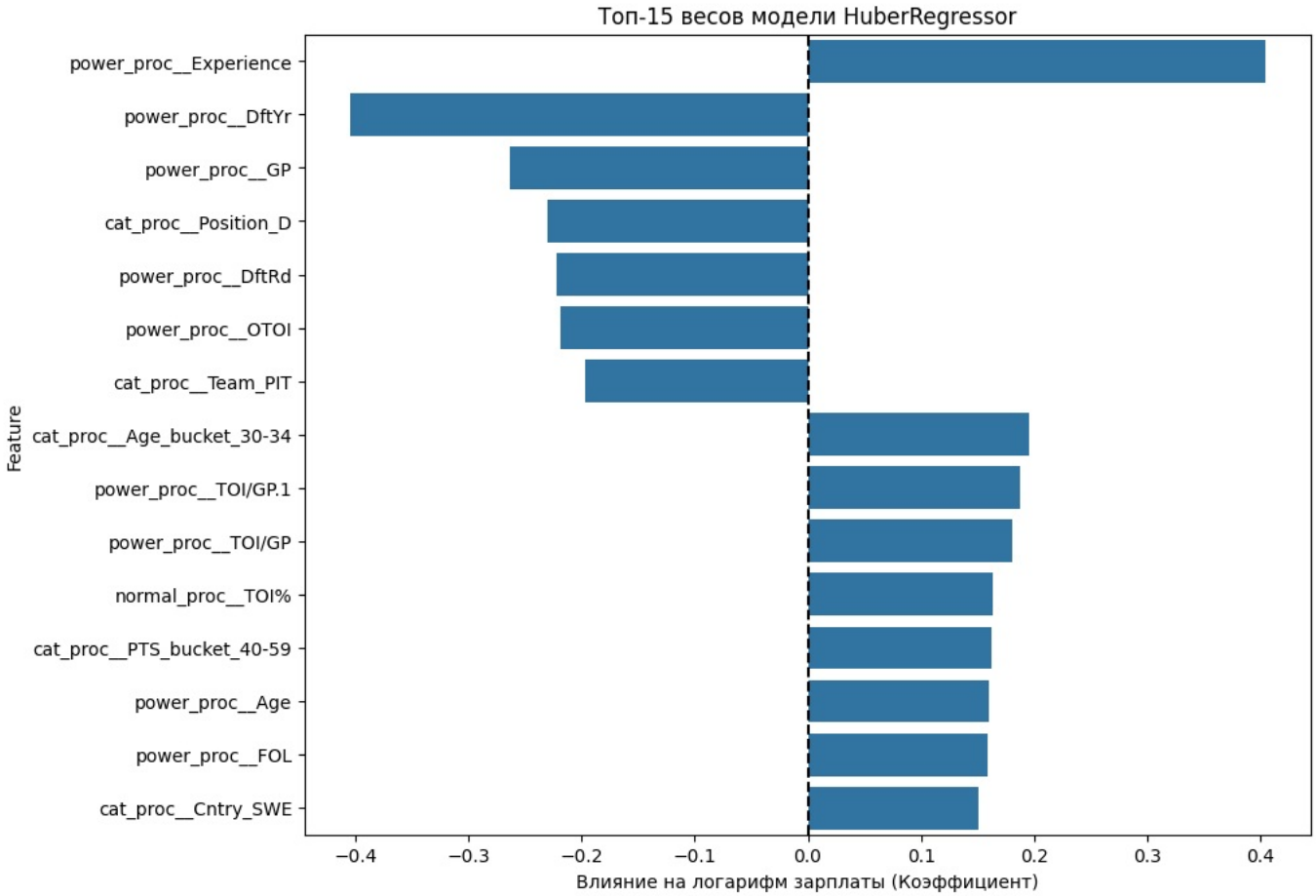
```
r2 = r2_score(y_test, y_pred)

print(f"\n--- HuberRegressor ---")
print(f"MAE: {mae:.3f}")
print(f"Train MAE: {train_mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
{'model_wrapper__regressor__alpha': 25.0, 'model_wrapper__regressor__epsilon': 1.5, 'preprocessor__normal_proc__
scaler': PowerTransformer(), 'preprocessor__power_proc__scaler': QuantileTransformer(n_quantiles=400)}

--- HuberRegressor ---
MAE: 874617.127
Train MAE: 741725.809
R2 Score: 0.645
```

Регрессор показал очень хороший результат.

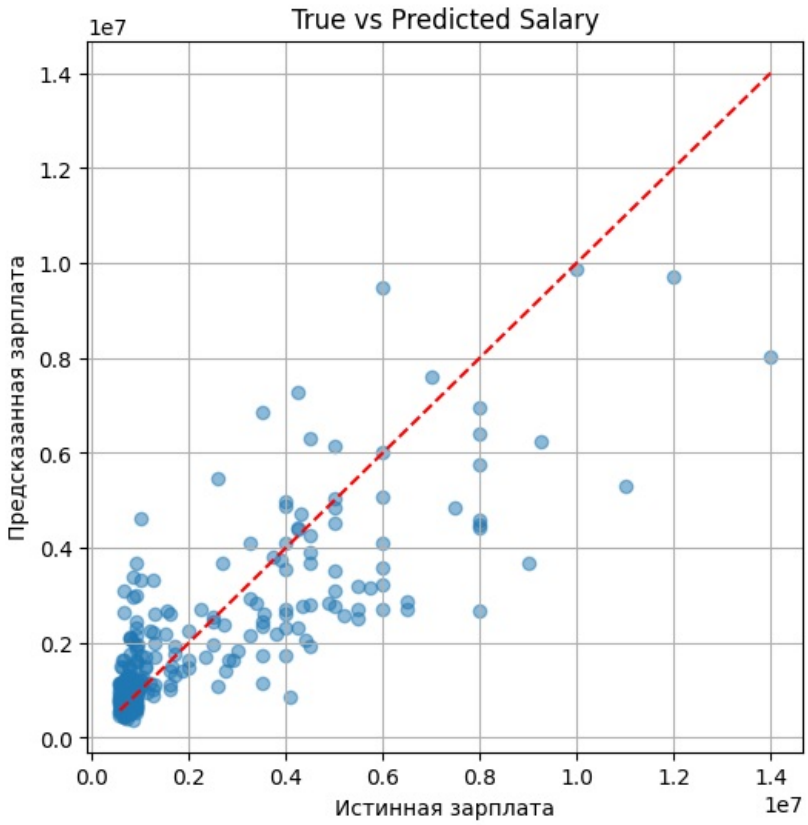
In [15]:

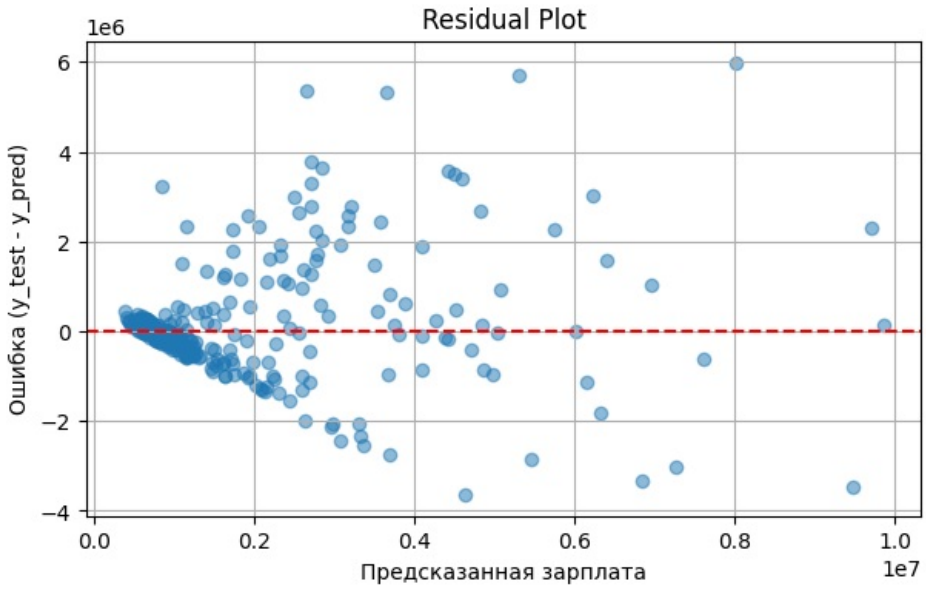
```
import pandas as pd
import numpy as np

def best_feature(s_grid):
    best_pipe = s_grid.best_estimator_

    preprocessor = best_pipe[0]
    feature_names = preprocessor.get_feature_names_out()

    model_wrapper = best_pipe[-1]
    regressor = model_wrapper.regressor_
    coefficients = regressor.coef_

    df_importance = pd.DataFrame({
        'Feature': feature_names,
        'Coefficient': coefficients,
        'Abs_Coefficient': np.abs(coefficients)
    })

    df_importance = df_importance.sort_values(by='Abs_Coefficient', ascending=False)

    import matplotlib.pyplot as plt
    import seaborn as sns

    plt.figure(figsize=(10, 8))
    sns.barplot(data=df_importance.head(15), x='Coefficient', y='Feature')
    plt.title("Топ-15 весов модели HuberRegressor")
    plt.xlabel("Влияние на логарифм зарплаты (Коэффициент)")
    plt.axvline(0, color='black', linestyle='--')
    plt.show()

best_feature(grid)
```

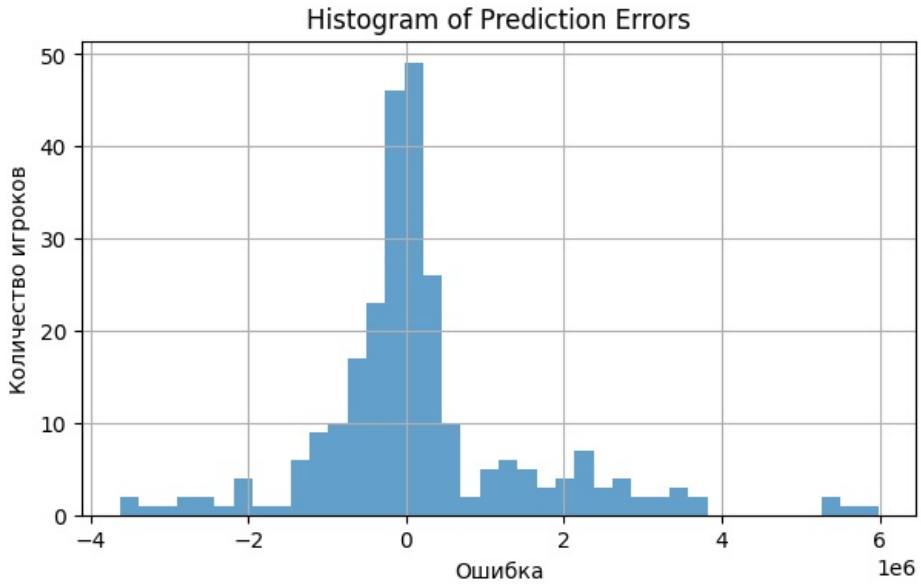


По коэфициентам модели видно, что модель делает вполне логичные выводы. На зарплату сильно влияют опыт игрока в лиге, год драфта (молодой игрок получает явно меньше, чем опытный), защитники и игроки из поздних раундов обычно получают меньше, а вот игроки в праймовом возрасте 30-35 больше

In [16]:
```
true_vs_predicted(y_test_i, y_pred)
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```







Данные стали сильно более кучнее относительно идеальной оси. Это хорошо. Плохо то, что модель всё еще плохо справляется с "выбросами" по зарплате. Если у игрока она очень большая относительно массы, модели сложно определить реальную стоимость и она в основном занижает ее.

In [17]:
```
original_float_format = pd.get_option('display.float_format')

pd.set_option('display.float_format', '{:.2f}'.format)

best_model = grid.best_estimator_

y_full_pred = best_model.predict(X_i)

results_df = pd.DataFrame({
    'True_Salary': y_i,
    'Predicted_Salary': y_full_pred
})

results_df['Absolute_Error'] = np.abs(results_df['True_Salary'] - results_df['Predicted_Salary'])

test_indices = X_test_i.index
final_results = pd.merge(
    df[['First Name', 'Last Name']],
    results_df,
    left_index=True,
    right_index=True
)

final_results_sorted = final_results.sort_values(by='Absolute_Error', ascending=False)
```

```
print("\n--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---")
print(final_results_sorted.head(10).to_string())

pd.set_option('display.float_format', original_float_format)
```

```
--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---
    First Name Last Name  True_Salary  Predicted_Salary  Absolute_Error
103    Patrick      Kane  13800000.00        5569645.95      8230354.05
817     Steven   Stamkos   9500000.00        2207866.03      7292133.97
496   Jonathan     Toews  13800000.00        6943347.88      6856652.12
626       Anze   Kopitar  14000000.00        8015766.60      5984233.40
830    Patrick     Eaves   1000000.00        6827481.26      5827481.26
208       P.K.    Subban  11000000.00        5302556.72      5697443.28
787      Derek    Stepan   8000000.00        2658989.11      5341010.89
168       Phil    Kessel   9000000.00        3665171.96      5334828.04
594     Evgeni    Malkin   9500000.00        4196835.57      5303164.43
669      Brent     Burns   5760000.00       10362818.20      4602818.20
```

Кэйн и Стэмкос подписывали контракт с рынка в статусе НСА, поэтому их зарплаты довольно высокие. Что мы тут и видим.

In [18]:
```
import pandas as pd
final_results['Signed_Error'] = final_results['Predicted_Salary'] - final_results['True_Salary']

overestimated_players = final_results[final_results['Signed_Error'] > 0].copy()

top_overestimated = overestimated_players.sort_values(by='Signed_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---")

print(top_overestimated[['First Name', 'Last Name',
                         'True_Salary', 'Predicted_Salary', 'Signed_Error']].head(10).to_string())
```

```
--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---
    First Name     Last Name  True_Salary  Predicted_Salary  Signed_Error
830    Patrick         Eaves    1000000.0      6.827481e+06  5.827481e+06
669      Brent         Burns    5760000.0      1.036282e+07  4.602818e+06
131    Jaromir          Jagr    4000000.0      7.736875e+06  3.736875e+06
532      Radim        Vrbata    1000000.0      4.631222e+06  3.631222e+06
299     Duncan         Keith    6000000.0      9.482628e+06  3.482628e+06
326     Martin        Hanzal    3500000.0      6.846601e+06  3.346601e+06
472   Jonathan  Marchessault     750000.0      3.950661e+06  3.200661e+06
286     Victor        Hedman    4250000.0      7.266826e+06  3.016826e+06
440     Thomas         Vanek    2600000.0      5.466796e+06  2.866796e+06
382       Jack        Eichel     925000.0      3.685323e+06  2.760323e+06
```

Тут мы видим всех тех же молодых будущих звёзд лиги, а также ветерана Ягра, который даже в свои очень большие годы всё еще отлично смотрелся. А также очень хорошего защитника Бёрнса.

My implementation

Напишем теперь свою линейную регрессию. Я решил реализовать Ridge.

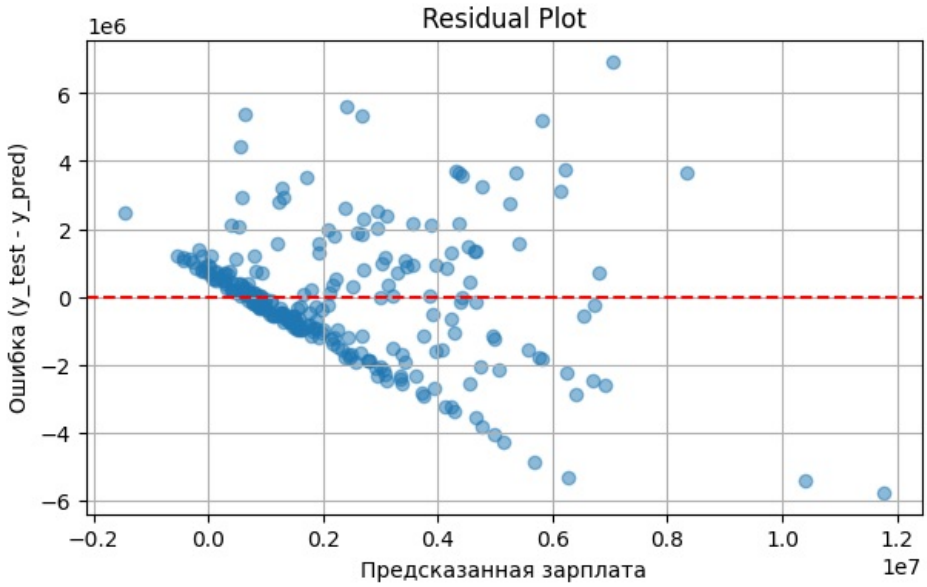
In [19]:
```
import numpy as np
from sklearn.base import BaseEstimator, RegressorMixin

class MyRidge(BaseEstimator, RegressorMixin):
    def __init__(self, alpha=1.0, fit_intercept=True):
        self.alpha = alpha
        self.fit_intercept = fit_intercept

    def fit(self, X, y):
        X = np.asarray(X)
        y = np.asarray(y).reshape(-1)

        if self.fit_intercept:
            X = np.column_stack([np.ones(X.shape[0]), X])

        XTX = X.T @ X

        I = np.eye(XTX.shape[0])
        if self.fit_intercept:
            I[0, 0] = 0

        XTy = X.T @ y

        self.coef_ = np.linalg.pinv(XTX + self.alpha * I) @ XTy

        return self

    def predict(self, X):
        X = np.asarray(X)
```

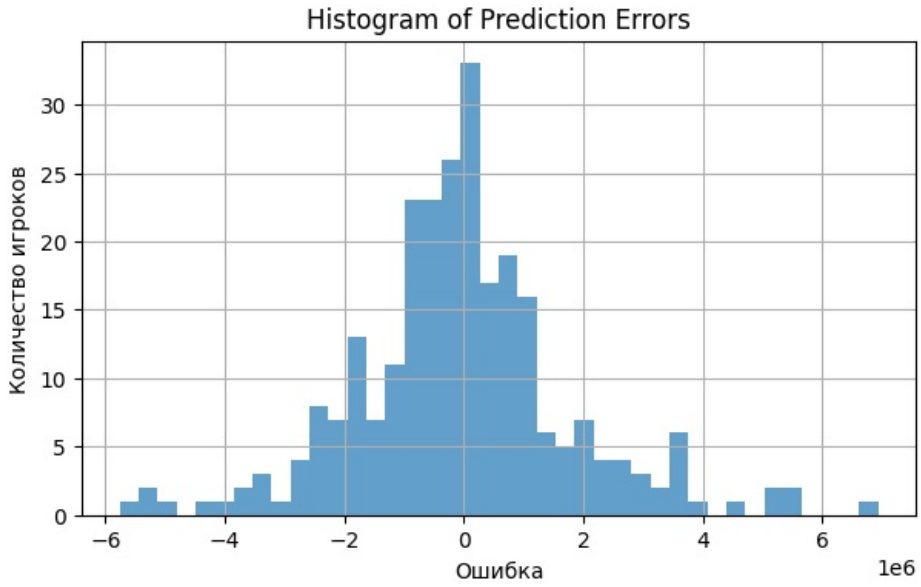
```
        if self.fit_intercept:
            X = np.column_stack([np.ones(X.shape[0]), X])

        return X @ self.coef_
```

In [20]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import seaborn as sns

model = MyRidge()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, [y_train.mean()]*len(y_test))

print(f"--- Baseline LinearRegreesion ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
--- Baseline LinearRegreesion ---
MAE (Ошибка в долларах): 1320465.400
R2 Score: 0.388
```

In [21]:
```
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```



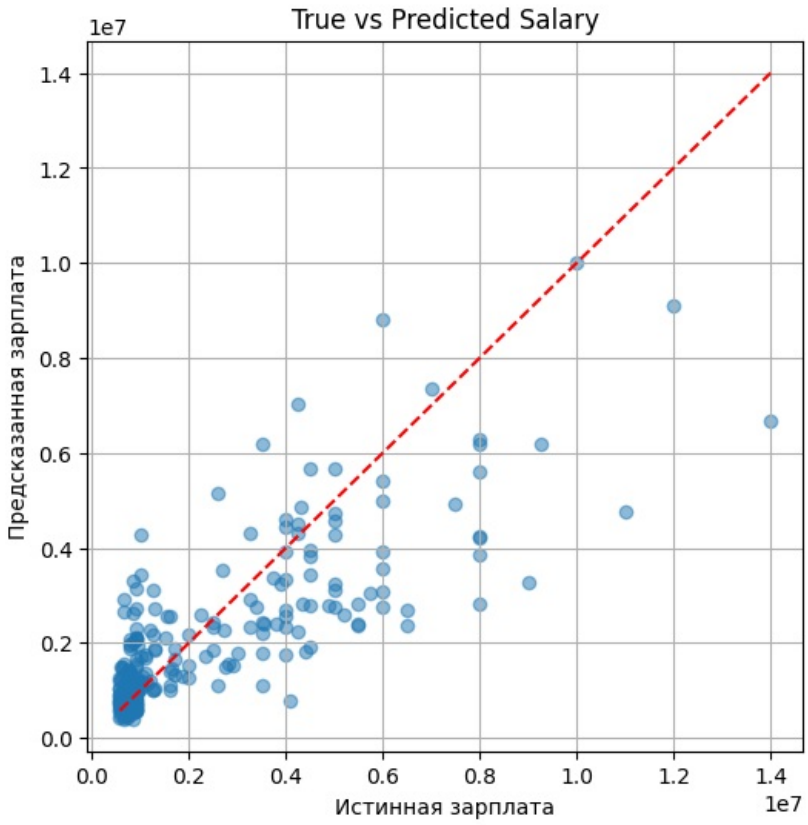


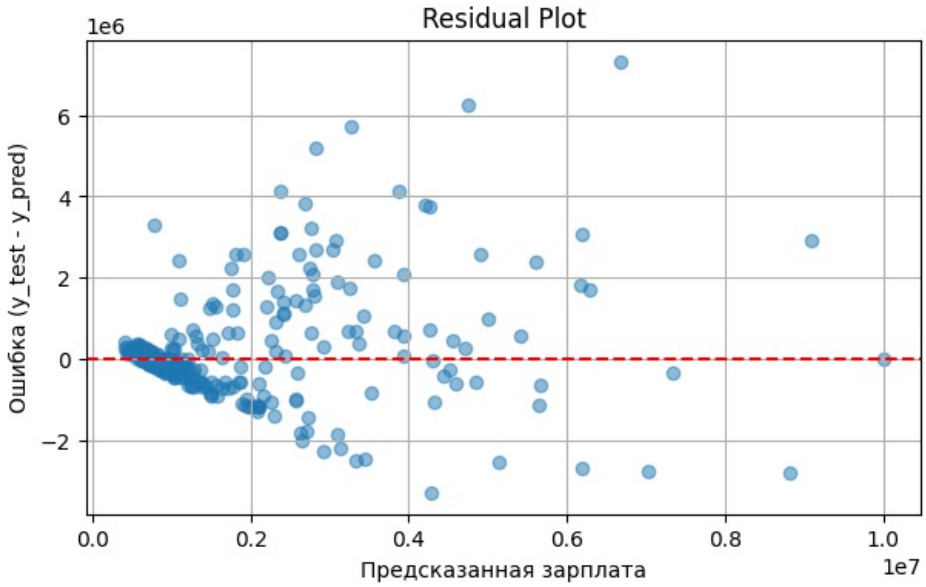
In [22]:
```
full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=MyRidge(),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'preprocessor__normal_proc__scaler': [
        RobustScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
```

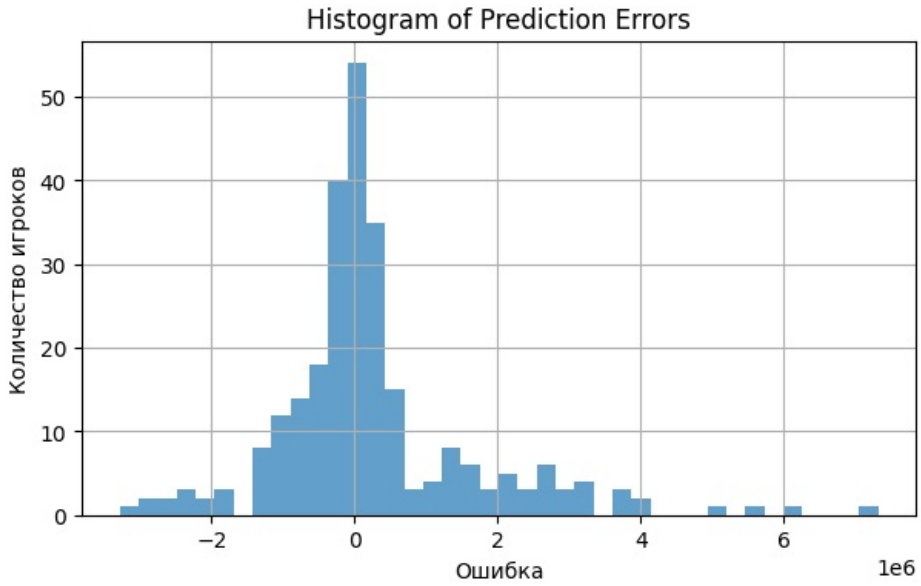
```
    ],
    'preprocessor__power_proc__scaler': [
        StandardScaler(),
        PowerTransformer(),
        QuantileTransformer(n_quantiles=400)
    ],
    'model_wrapper__regressor__alpha': [0.1, 1.0, 10.0, 50.0]
}

grid = GridSearchCV(
    full_pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print(grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)

mae = mean_absolute_error(y_test_i, y_pred)
r2 = r2_score(y_test_i, y_pred)

print(f"\n--- Ridge Regression ---")
print(f"MAE: {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
print(f"MAE для baseline (среднее): {base_mean_error:.3f}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
{'model_wrapper__regressor__alpha': 10.0, 'preprocessor__normal_proc__scaler': RobustScaler(), 'preprocessor__po
wer_proc__scaler': QuantileTransformer(n_quantiles=400)}

--- Ridge Regression ---
MAE: 892750.102
R2 Score: 0.622
MAE для baseline (среднее): 1877717.767
```

In [23]:
```
true_vs_predicted(y_test_i, y_pred)
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```







Собственная модель с регуляризацией работает неплохо, чуть хуже чем Huber, потому что хуже работает с выбросами.

	Base Linear Regression	Huber Regression	My Ridge
R2	0.324	0.645	0.622
MAE	1381895.059	874617.127	892750.102

Классификация

Задача: вычислить мошенника на страховых выплатах с использованием моделей линейной регрессии

Для выполнения лабораторной работы были выбраны метрики F1-score и ROC-AUC, так как исследуемый датасет является несбалансированным. Метрика Accuracy в данном случае неинформативна, так как модель, предсказывающая всем класс ‘0’ (не фрод), может иметь высокую Accuracy, но будет бесполезна. F1-score позволит контролировать баланс между ложными срабатываниями и пропуском мошенников.

Baseline

In [24]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                            "buntyshah/auto-insurance-claims-data/versions/1",
```

```
                                "insurance_claims.csv")
df
```

Out[24]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 40 columns

In [25]:
```
df_clean = df.copy()

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
df_clean["property_damage"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
```

In [26]:
```
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean.head()
```

Out[26]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium
0	328	48	521585	940	2	1	1000	1406.91
1	228	42	342868	635	1	1	2000	1197.22
2	134	29	687698	413	2	0	2000	1413.14
3	256	41	227811	19	0	1	2000	1415.74
4	228	44	367455	922	0	2	1000	1583.91

5 rows × 40 columns

In [27]:
```
df_clean = df_clean.fillna(0)
```

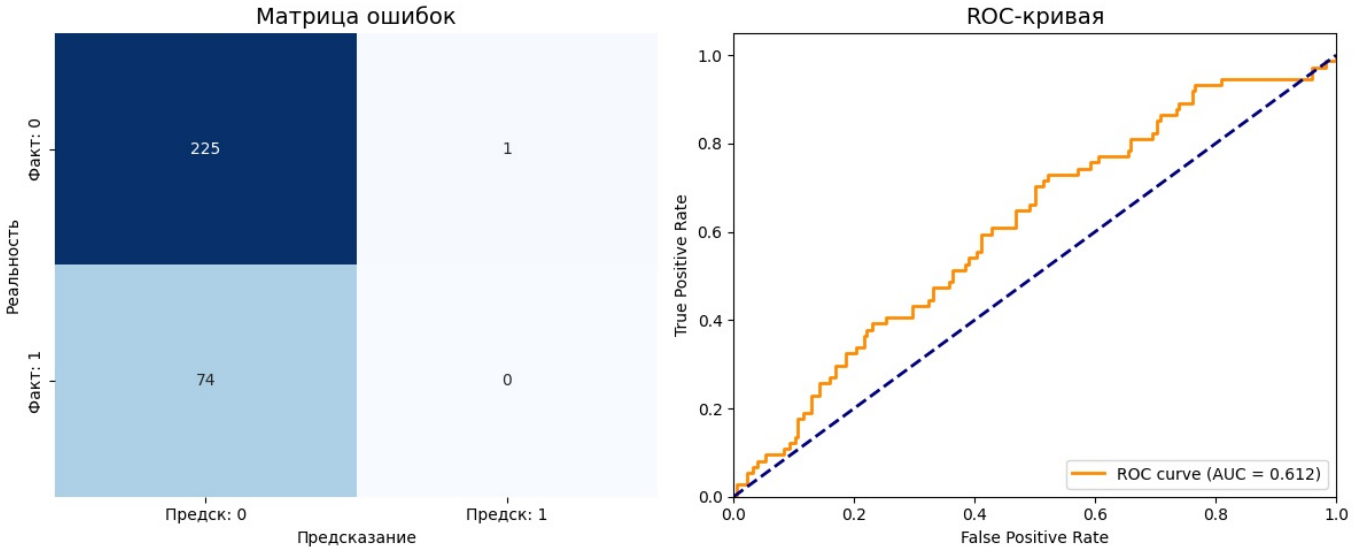
In [28]:
```
from sklearn.model_selection import train_test_split
drop_dates = ["policy_bind_date", "incident_date"]
df_clean = df_clean.drop(drop_dates, axis=1)
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

In [29]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, f1_score, roc_auc_score, confusion_matrix, roc_curve

model = LogisticRegression(max_iter=800)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
F1-score (класс 1): 0.0000
ROC-AUC: 0.6118
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.75      1.00      0.86       226
           1       0.00      0.00      0.00        74

    accuracy                           0.75       300
   macro avg       0.38      0.50      0.43       300
weighted avg       0.57      0.75      0.65       300

==================================================
```

```
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/linear_model/_logistic.
py:473: ConvergenceWarning: lbfgs failed to converge after 800 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=800).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

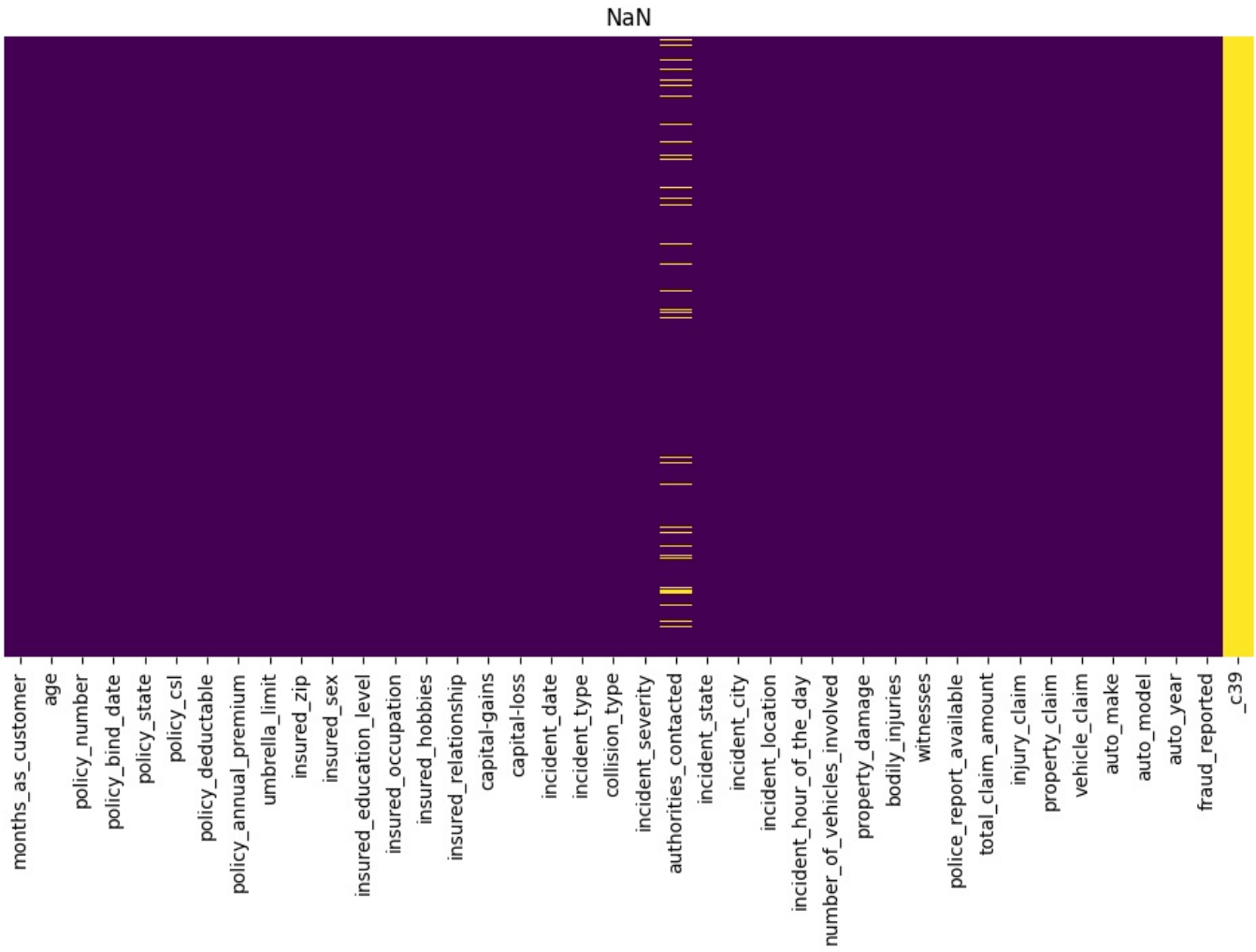
In [30]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc
def graphics(y_test, y_pred, y_prob):
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", cbar=False,
                xticklabels=['Предск: 0', 'Предск: 1'],
                yticklabels=['Факт: 0', 'Факт: 1'])
    plt.title("Матрица ошибок", fontsize=14)
    plt.ylabel("Реальность")
    plt.xlabel("Предсказание")

    plt.subplot(1, 2, 2)
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC-кривая', fontsize=14)
    plt.legend(loc="lower right")

    plt.tight_layout()
    plt.show()
graphics(y_test, y_pred, y_prob)
```



Базовая логистическая регрессия совсем не хочет искать мошенников. Ей проще записать всех в честных и в целом, она даже будет права xD

Improved Logistic Regressor

In [31]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                    "buntyshah/auto-insurance-claims-data/versions/1",
                                    "insurance_claims.csv")
df.head()

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [32]:
```
df_clean = df.copy()
df_clean = df_clean.drop(columns=["_c39"])
df_clean["authorities_contacted"] = df_clean["authorities_contacted"].fillna("No Contact")
```

In [33]:
```
TARGET_NAME = "fraud_reported"
df_clean[TARGET_NAME].value_counts()
```

Out[33]:
```
fraud_reported
N    753
Y    247
Name: count, dtype: int64
```

In [34]:
```
display(df_clean["police_report_available"].unique())
display(df_clean["property_damage"].unique())
```

```
array(['YES', '?', 'NO'], dtype=object)
array(['YES', '?', 'NO'], dtype=object)
```

In [35]:
```
import pandas as pd

df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0, '?': -1})
df_clean["property_damage"] = df_clean["property_damage"].map({'YES': 1, 'NO': 0, '?': -1})
```

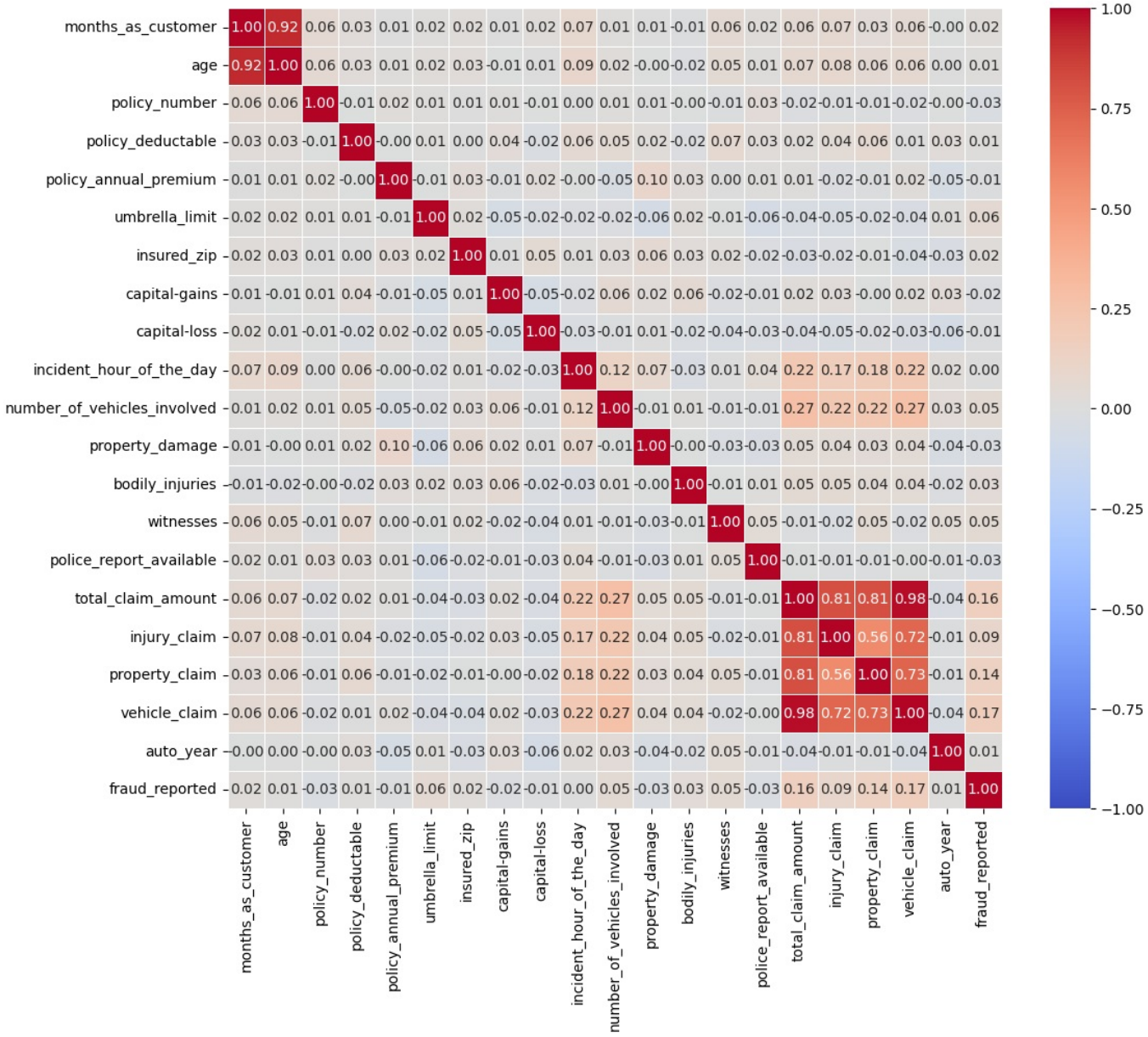
In [36]:
```
dates_cols = ["policy_bind_date", "incident_date"]
for c in dates_cols:
    df_clean[c] = pd.to_datetime(df_clean[c])
df_clean
```

Out[36]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 39 columns

In [37]:
```
num_cols = df_clean.select_dtypes(include=["int64", "float64"]).columns.tolist()

plt.figure(figsize=(12, 10))
correlation_matrix = df_clean[num_cols].corr()

sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            vmin=-1, vmax=1,
            linewidths=0.5)

plt.show()
```



In [38]:
```
num_cols = df_clean.select_dtypes(include=["int64", "float64"]).columns.tolist()
df_clean[num_cols].describe().T
```

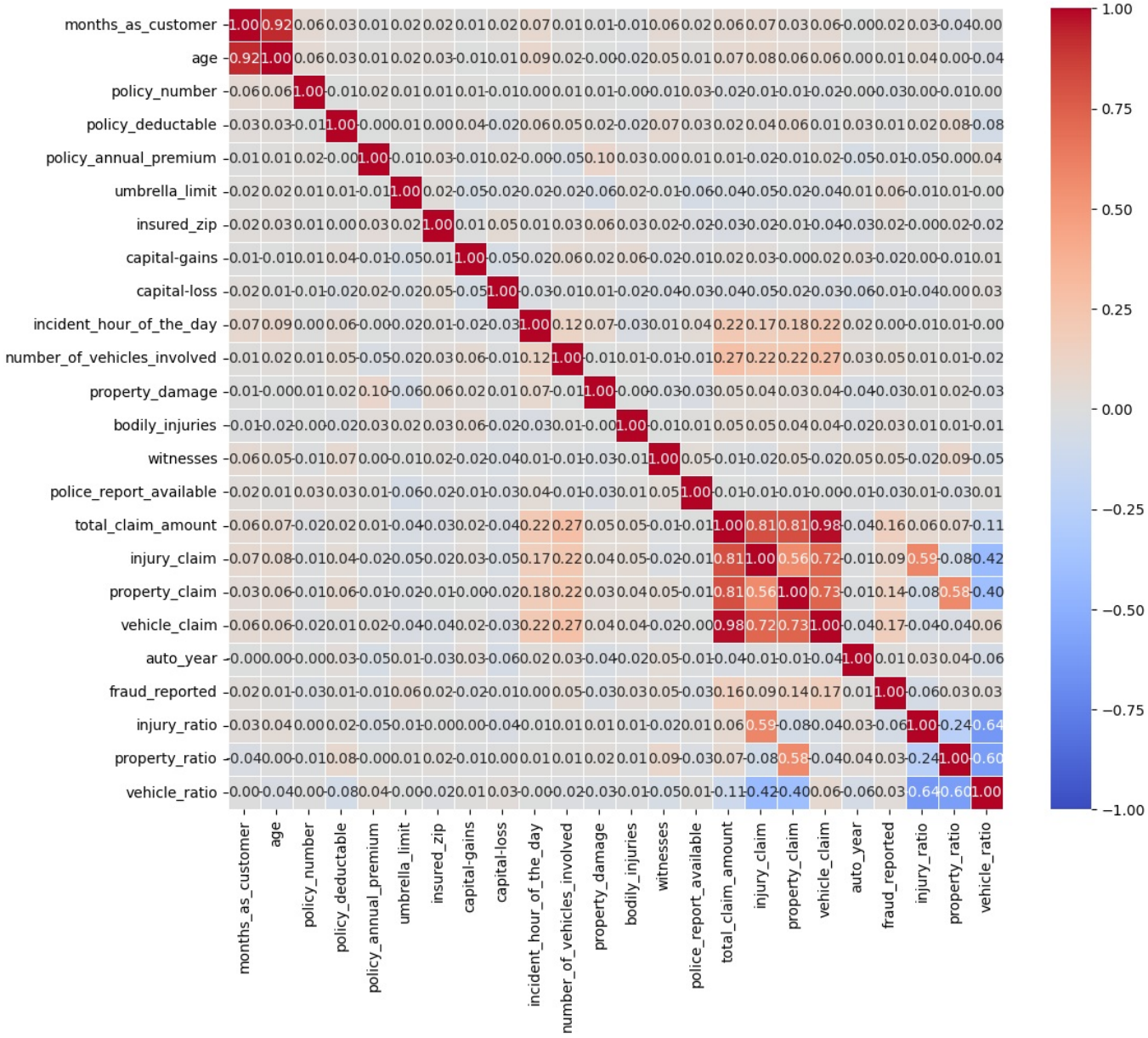
Out[38]:

	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
property_damage	1000.0	-5.800000e-02	8.119700e-01	-1.00	-1.0000	0.0	1.000	1.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
police_report_available	1000.0	-2.900000e-02	8.104417e-01	-1.00	-1.0000	0.0	1.000	1.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00
fraud_reported	1000.0	2.470000e-01	4.314825e-01	0.00	0.0000	0.0	0.000	1.00

In [39]:
```
import numpy as np

df_pr = df_clean.copy()

median_value = df_pr.loc[df_pr['umbrella_limit'] != -100000, 'umbrella_limit'].median()
df_pr.loc[df_pr['umbrella_limit'] == -100000, 'umbrella_limit'] = median_value
```

In [40]:
```
df_features = df_pr.copy()

df_features["age_bucket"] = pd.cut(
    df_features["age"],
    bins=[0, 25, 35, 45, 55, 65, 100],
    labels=["<25", "25-34", "35-44", "45-54", "55-64", "65+"],
)

df_features["incident_year"] = df_features["incident_date"].dt.year
df_features["incident_month"] = df_features["incident_date"].dt.month
df_features["incident_dow"] = df_features["incident_date"].dt.dayofweek

df_features["injury_ratio"] = df_features["injury_claim"] / (df_features["total_claim_amount"] + 1e-3)
df_features["property_ratio"] = df_features["property_claim"]  / (df_features["total_claim_amount"] + 1e-3)
df_features["vehicle_ratio"] = df_features["vehicle_claim"] / (df_features["total_claim_amount"] + 1e-3)

drop_dates = ["policy_bind_date", "incident_date"]
df_features = df_features.drop(drop_dates, axis=1)
```

In [41]:
```
num_cols = df_features.select_dtypes(include=["int64", "float64"]).columns.tolist()

plt.figure(figsize=(12, 10))
correlation_matrix = df_features[num_cols].corr()

sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            vmin=-1, vmax=1,
            linewidths=0.5)

plt.show()
```



In [42]:
```
from sklearn.model_selection import train_test_split
X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_i, y_i, test_size=0.25, random_state=42, stratify=
```

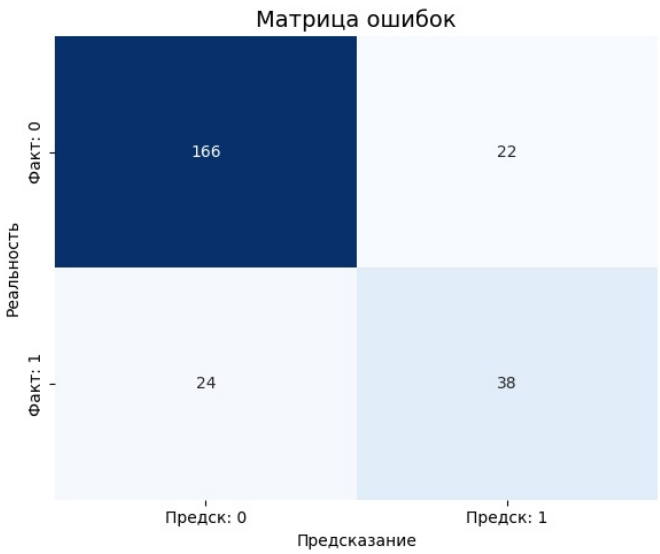
Тут тоже фактически изменения только в модели. Я использовал liblinear solver в сетку добавил l1, l2 регуляризацию. Перестал использовать SMOTE, поскольку логистический классификатор нативно работает с несбалансированными классами.

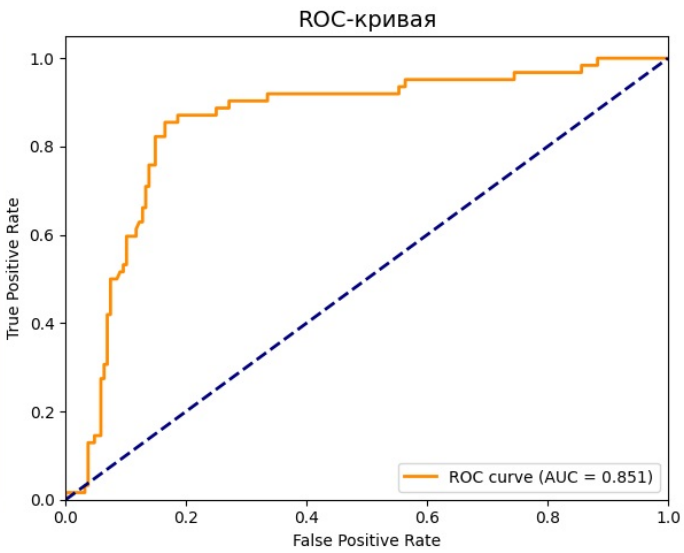
In [43]:
```
from category_encoders import WOEEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import QuantileTransformer
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import QuantileTransformer

num_cols = X_i.select_dtypes(["int64", "float64"]).columns.tolist()
cat_cols = X_i.select_dtypes(["object", "category"]).columns.tolist()

numeric_pipe     = Pipeline([("scale", QuantileTransformer(n_quantiles=400))])
categorical_pipe = Pipeline([("onehot", WOEEncoder())])


ct  = ColumnTransformer([
    ("num", numeric_pipe, num_cols),
    ("cat", categorical_pipe, cat_cols)
])

model_pipe = Pipeline([
    ('ct', ct),
    ('model', LogisticRegression(solver='liblinear', max_iter=5000, random_state=42))
])
```

```
param_grid = {
    'model__C': [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0],
    'model__penalty': ['l1', 'l2'],
    'model__class_weight': [None, 'balanced'],
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(
    model_pipe,
    param_grid,
    cv=cv,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print(grid.best_params_)

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
print("="*50)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
{'model__C': 0.1, 'model__class_weight': None, 'model__penalty': 'l1'}
F1-score (класс 1): 0.6230
ROC-AUC: 0.8513
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.88      0.88       188
           1       0.63      0.61      0.62        62

    accuracy                           0.82       250
   macro avg       0.75      0.75      0.75       250
weighted avg       0.81      0.82      0.81       250

==================================================
```

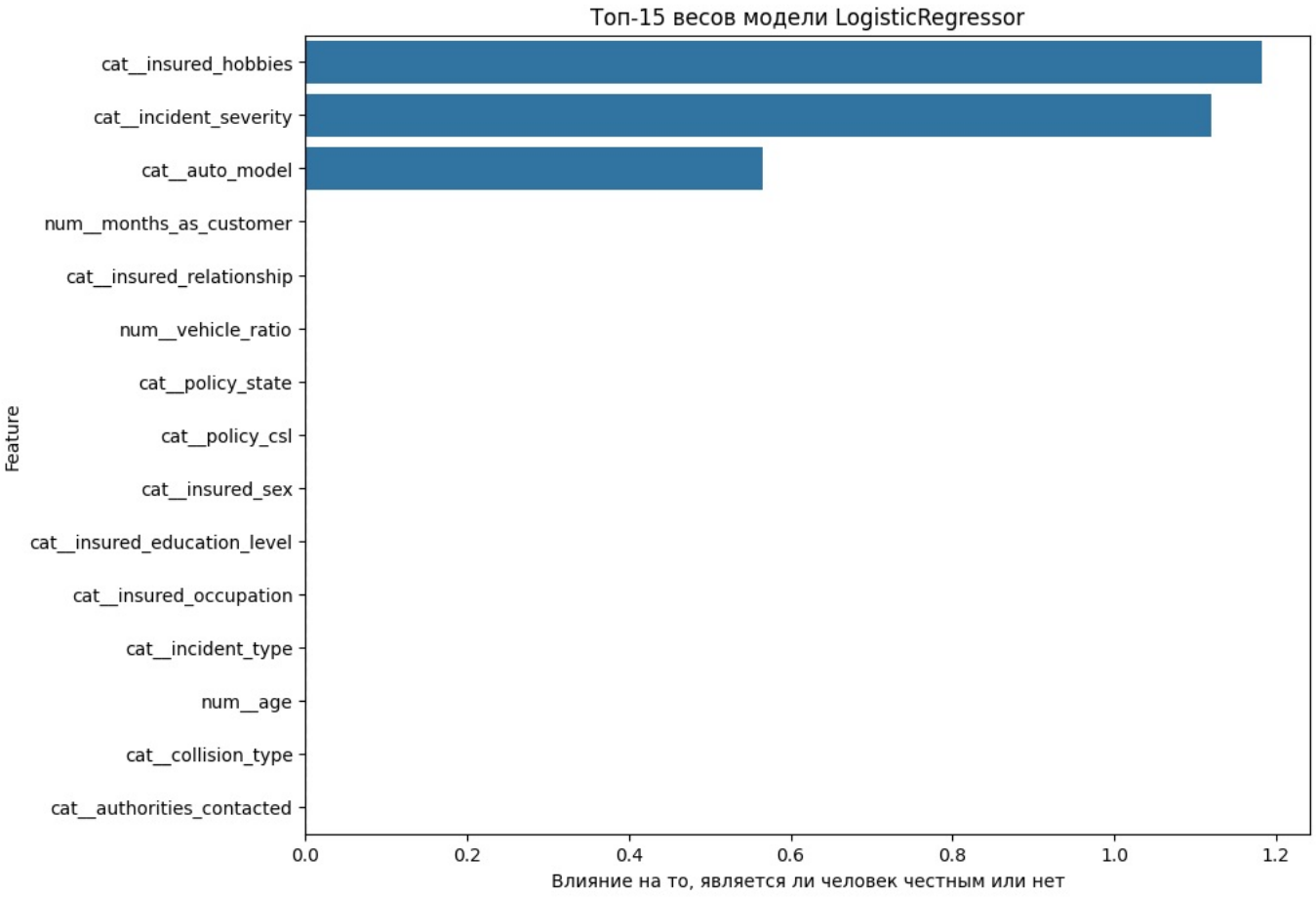
In [44]:
```
graphics(y_test_i, y_pred, y_prob)
```





В результате получилось чуть хуже, чем улучшенный KNN. Эта модель нашла меньше мошенников, но при этом и меньше неверно указала на честных людей. Лучше конечно в реальной жизни использовать KNN, потому что лучше лишний раз проверить честного, чем пропустить мошенника. Это дешевле

In [45]:
```
import pandas as pd
import numpy as np

best_pipe = grid.best_estimator_

feature_names = best_pipe.named_steps['ct'].get_feature_names_out()

model = best_pipe.named_steps['model']

coefficients = model.coef_.flatten()

df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients,
    'Abs_Coefficient': np.abs(coefficients)
})

df_importance = df_importance.sort_values(by='Abs_Coefficient', ascending=False)

plt.figure(figsize=(10, 8))
sns.barplot(data=df_importance.head(15), x='Coefficient', y='Feature')
plt.title("Топ-15 весов модели LogisticRegressor")
plt.xlabel("Влияние на то, является ли человек честным или нет")
plt.axvline(0, color='black', linestyle='--')
plt.show()
```



Модель посчитала, что самыми важными признаками является хобби, серъезность повреждений и модель автомобиля. В целом это сочитается с реальностью.

My implementation

In [46]:
```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.validation import check_X_y, check_is_fitted, check_array
from sklearn.utils.multiclass import unique_labels

class MyLogisticRegression(ClassifierMixin, BaseEstimator):
    def __init__(self, learning_rate=0.01, n_iters=1000, threshold=0.5,
                 reg_strength=0.0, class_weight=None):
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.threshold = threshold
        self.reg_strength = reg_strength
        self.class_weight = class_weight
        self.weights = None
```

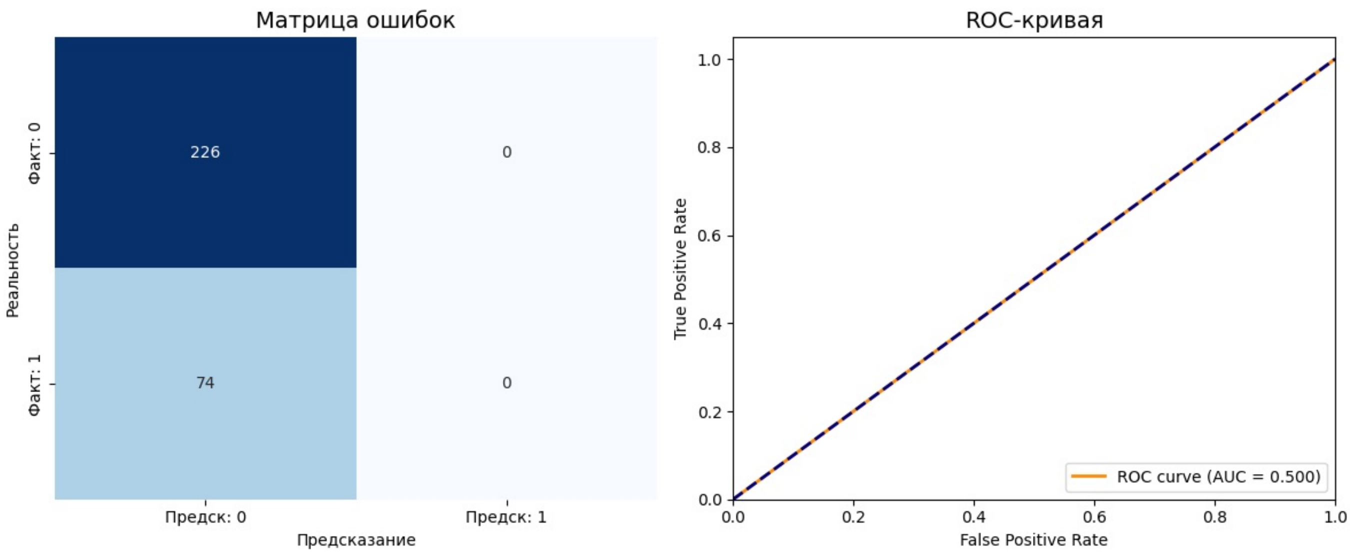
```
        self.bias = None

    def _sigmoid(self, z):
        z = np.clip(z, -250, 250)
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0
        sample_weights = np.ones(n_samples)
        if self.class_weight == 'balanced':
            n_pos = np.sum(y == 1)
            n_neg = np.sum(y == 0)
            weight_pos = n_samples / (2 * n_pos) if n_pos > 0 else 1.0
            weight_neg = n_samples / (2 * n_neg) if n_neg > 0 else 1.0
            sample_weights = np.where(y == 1, weight_pos, weight_neg)

        for _ in range(self.n_iters):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self._sigmoid(linear_model)
            error = y_predicted - y
            weighted_error = error * sample_weights
            dw = (1 / n_samples) * np.dot(X.T, weighted_error)
            db = (1 / n_samples) * np.sum(weighted_error)
            if self.reg_strength > 0:
                dw += (self.reg_strength / n_samples) * self.weights
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db
        return self

    def predict_proba(self, X):
        check_is_fitted(self)
        X = check_array(X)
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        return np.vstack(((1 - y_predicted), y_predicted)).T

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        return np.array([1 if i > self.threshold else 0 for i in y_predicted])
```

In [47]:
```
model = MyLogisticRegression()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
F1-score (класс 1): 0.0000
ROC-AUC: 0.5000
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.75      1.00      0.86       226
           1       0.00      0.00      0.00        74

    accuracy                           0.75       300
   macro avg       0.38      0.50      0.43       300
weighted avg       0.57      0.75      0.65       300

==================================================
```

```
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```

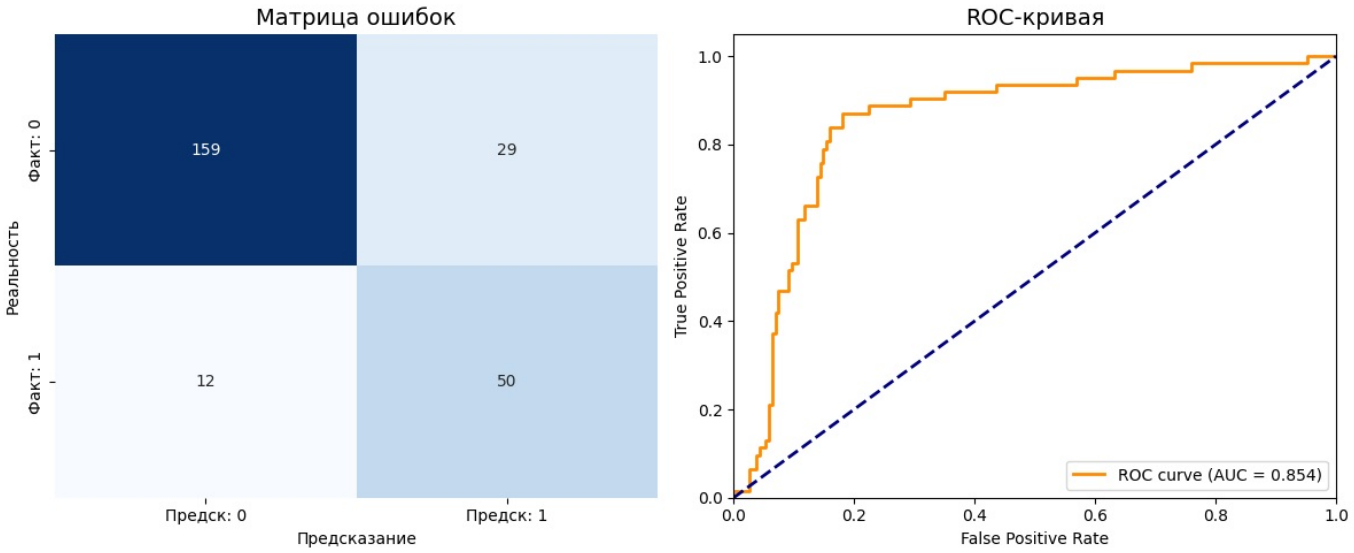
In [48]:
```
graphics(y_test, y_pred, y_prob)
```



In [49]:
```
model_pipe = Pipeline([
    ('ct', ct),
    ('model', MyLogisticRegression(class_weight='balanced'))
])

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

param_grid = {
    'model__learning_rate': [0.01, 0.1, 0.5],
    'model__n_iters': [2000],
    'model__reg_strength': [0.0, 0.1, 1.0, 10.0]
}

grid = GridSearchCV(
    model_pipe,
    param_grid,
    cv=3,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)
print(grid.best_params_)

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
print("="*50)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
{'model__learning_rate': 0.01, 'model__n_iters': 2000, 'model__reg_strength': 0.1}
F1-score (класс 1): 0.7092
ROC-AUC: 0.8542
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.85      0.89       188
           1       0.63      0.81      0.71        62

    accuracy                           0.84       250
   macro avg       0.78      0.83      0.80       250
weighted avg       0.86      0.84      0.84       250

==================================================
```

In [50]:
```
graphics(y_test_i, y_pred, y_prob)
```



Моя модель использует обычный градиентный спуск, в отличие от решателя от scikit-learn, и она показала лучшие результаты! Моя модель чаще находит мошенников, но при этом чуть чаще ошибается на честных.

	Base Linear Classifier	Logistic Classifier	My Logistic Classifier
ROC_AUC	0.6118	0.8513	0.8542
F1 (1 класс)	0.86	0.88	0.89
F1 (2 класс)	0.00	0.62	0.71

Вывод

Линейные алгоритмы дали хороший результат в обоих задачах. Пишутся они легко, интерпретируются также, поэтому я смог вывести фичи, которые модель считает основными для этих задач. Улучшение в задаче регрессии очень значительное, линейной модели легче найти закономерность среди большого количества фичей, чем размывающееся от большого количества расстояние в KNN.

Регрессия

Задача: предсказать зарплату игрока НХЛ по его статистике с использованием модели решающего дерева.

Будем использовать MAE, так как она очень наглядная, будет легко понять, насколько долларов ошибается модель. В качестве дополнительной метрики будем использовать R^2, чтобы смотреть насколько отличается предсказание от среднего.

Baseline

Скачаем датасет с kaggle

In [1]:
```
import kagglehub
import pandas as pd

path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")

df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)
df.head()
```

```
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: I
Progress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_i
nstall.html
  from .autonotebook import tqdm as notebook_tqdm
```

Out[1]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	14
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2'

5 rows × 154 columns

In [2]:
```
df['Born'] = pd.to_datetime(df['Born'], format='%y-%m-%d')
df_clean = df.fillna(0)
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean = df_clean.drop("Born", axis=1)
```

In [3]:
```
from sklearn.model_selection import train_test_split

TARGET_NAME = "Salary"
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

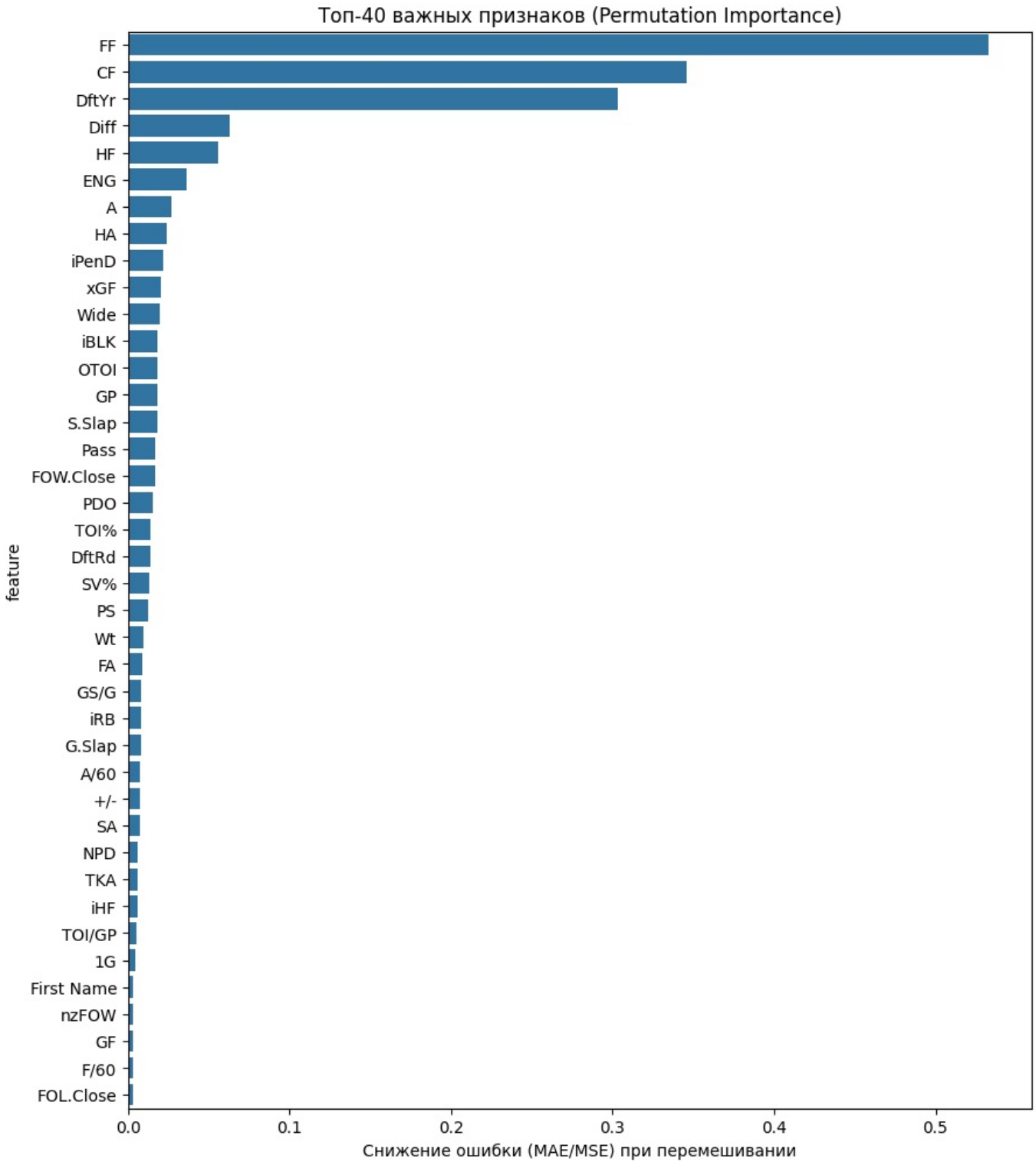
In [4]:
```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

dt = DecisionTreeRegressor(random_state=42, max_depth=None, min_samples_leaf=1)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)
```

```
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, np.full(len(y_test), y_train.mean()))

print("--- Decision Tree Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
--- Decision Tree Regressor ---
MAE: 1220967.395
R2 Score: 0.322
```

In [5]:

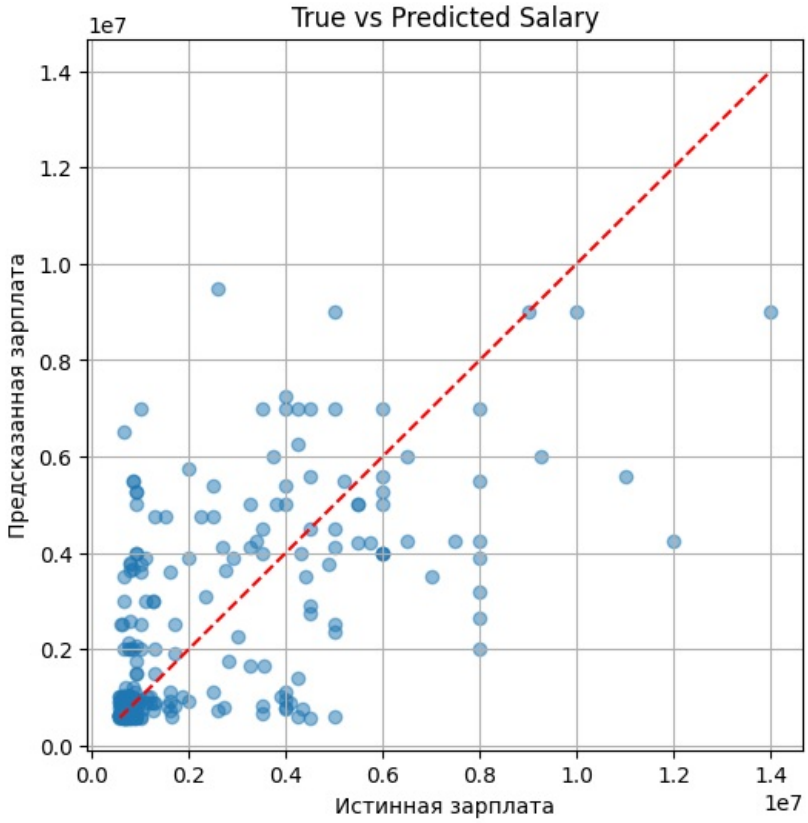
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.inspection import permutation_importance

def important_features(model, X_test, y_test):
    perm_result = permutation_importance(
        model,
        X_test,
        y_test,
        n_repeats=10,
        random_state=42,
        n_jobs=-1
    )

    perm_df = pd.DataFrame({
        'feature': X_test.columns,
        'importance': perm_result.importances_mean
    })

    top_40_perm = perm_df.sort_values(by='importance', ascending=False).head(40)

    plt.figure(figsize=(10, 12))
    sns.barplot(data=top_40_perm, x='importance', y='feature')
    plt.title("Топ-40 важных признаков (Permutation Importance)")
    plt.xlabel("Снижение ошибки (MAE/MSE) при перемешивании")
    plt.show()

important_features(dt, X_test, y_test)
```

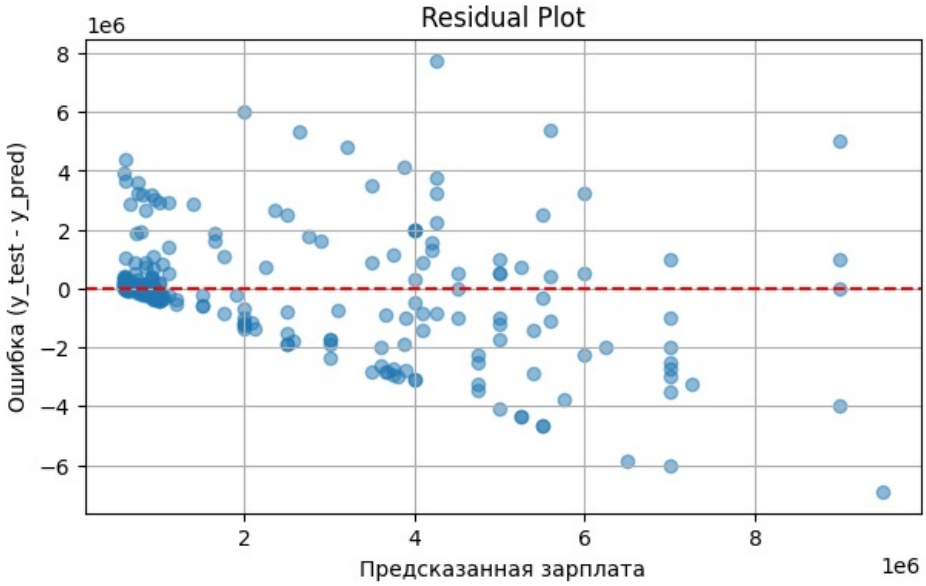


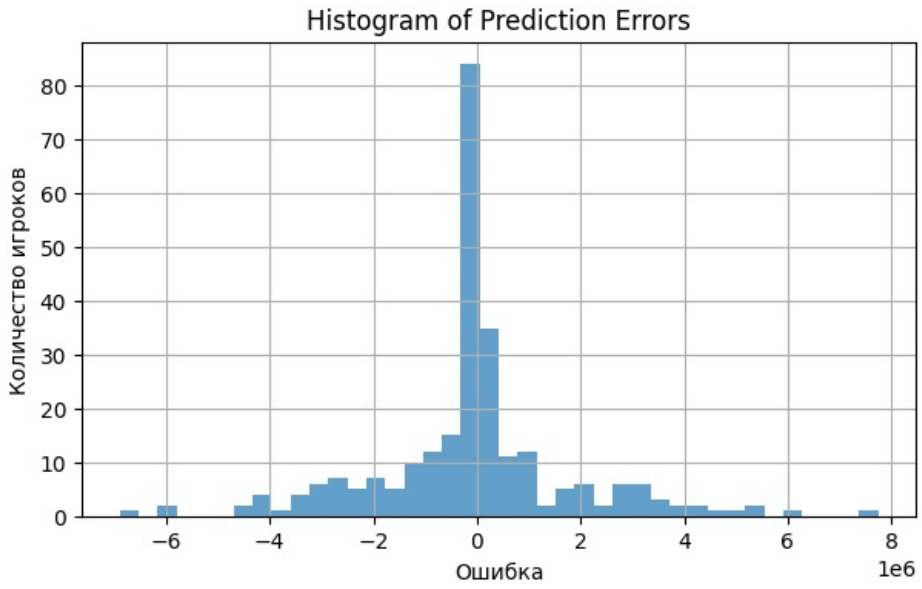
In [6]:

```python
from matplotlib import pyplot as plt

def true_vs_predicted(y_test, y_pred):
    plt.figure(figsize=(6,6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
    plt.xlabel("Истинная зарплата")
    plt.ylabel("Предсказанная зарплата")
    plt.title("True vs Predicted Salary")
    plt.grid(True)
    plt.show()

def resudial(y_test, y_pred):
    residuals = y_test - y_pred
    plt.figure(figsize=(7,4))
    plt.scatter(y_pred, residuals, alpha=0.5)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel("Предсказанная зарплата")
    plt.ylabel("Ошибка (y_test - y_pred)")
    plt.title("Residual Plot")
    plt.grid(True)
    plt.show()
def histogramm_error(y_test, y_pred):
    plt.figure(figsize=(7,4))
```

```
    residuals = y_test - y_pred
    plt.hist(residuals, bins=40, alpha=0.7)
    plt.title("Histogram of Prediction Errors")
    plt.xlabel("Ошибка")
    plt.ylabel("Количество игроков")
    plt.grid(True)
    plt.show()
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```



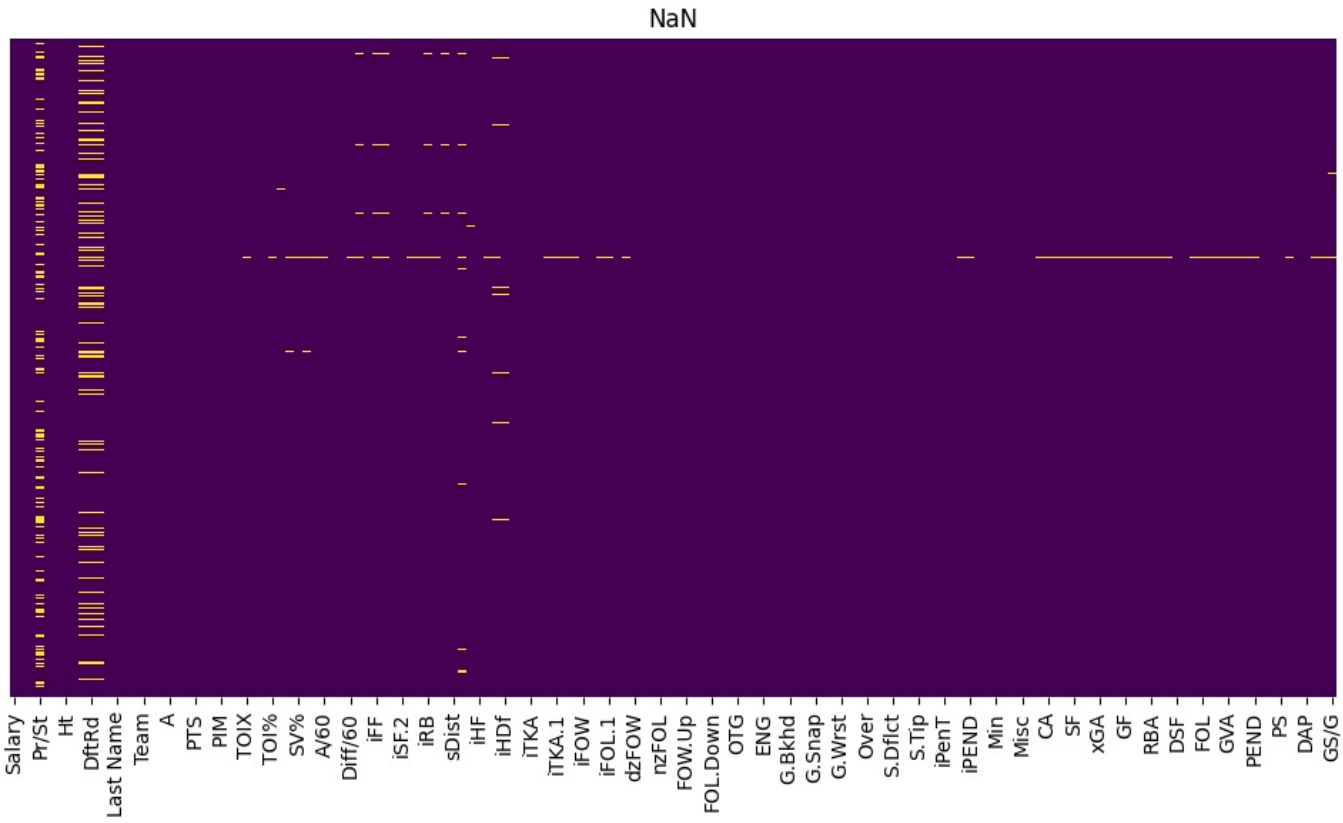




Решающее дерево на базовом пайплайне даёт так себе результаты, ошибки сильно разбросаны вокруг диагонали.

Improved Desicion Tree

In [7]:
```
import kagglehub
from matplotlib import pyplot as plt
import seaborn as sns

path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")
df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [8]:
```
num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
df[num_cols].describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
Salary	874.0	2.325289e+06	2.298253e+06	575000.00	742500.00	925000.00	3700000.00	14000000.00
Ht	874.0	7.308238e+01	2.105485e+00	66.00	72.00	73.00	75.00	81.00
Wt	874.0	2.008432e+02	1.506008e+01	157.00	190.00	200.00	210.00	265.00
DftYr	749.0	2.008708e+03	4.380158e+00	1990.00	2006.00	2010.00	2012.00	2016.00
DftRd	749.0	2.742323e+00	1.988358e+00	1.00	1.00	2.00	4.00	9.00
...	...	...	...	...	...	...	...	...
Grit	874.0	1.267815e+02	1.016121e+02	0.00	41.00	114.00	190.00	622.00
DAP	874.0	9.215675e+00	7.815029e+00	0.00	4.60	7.60	12.00	61.00
Pace	873.0	1.089439e+02	8.899877e+00	75.00	104.70	109.20	113.90	175.70
GS	873.0	2.187331e+01	2.198638e+01	-4.30	2.60	15.70	35.40	104.70
GS/G	872.0	3.401606e-01	2.925900e-01	-0.81	0.14	0.31	0.53	1.28

144 rows × 8 columns

In [9]:
```
import pandas as pd
import numpy as np

num_cols = df.select_dtypes(include=["int64", "float64"]).columns

corr_matrix = df[num_cols].corr()

corr_pairs = corr_matrix.unstack().reset_index()
corr_pairs.columns = ['feature_1', 'feature_2', 'correlation']

corr_pairs = corr_pairs[corr_pairs['feature_1'] < corr_pairs['feature_2']]

corr_pairs = corr_pairs.reindex(
    corr_pairs['correlation'].abs().sort_values(ascending=False).index
)

top40 = corr_pairs.head(40)

display(top40)
upper = np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
upper_corr = corr_matrix.where(upper)
```

	feature_1	feature_2	correlation
4931	iSF.1	iSF.2	0.999996
8267	iFOL	iFOL.1	0.999981
8122	iFOW	iFOW.1	0.999979
6381	iHF	iHF.1	0.999957
2321	TOI	TOIX	0.999952
7398	iBLK	iBLK.1	0.999948
4787	iSF	iSF.2	0.999937
4786	iSF	iSF.1	0.999931
7108	iGVA	iGVA.1	0.999907
7253	iTKA	iTKA.1	0.999843
2611	TOI/GP	TOI/GP.1	0.999289
15807	CF	FF	0.999216
4351	iCF	iCF.1	0.998819
16097	FF	SF	0.998722
10424	FOW.Close	iFOW	0.998613
10426	FOW.Close	iFOW.1	0.998603
16242	FA	SA	0.998366
10571	FOL.Close	iFOL.1	0.998221
10569	FOL.Close	iFOL	0.998215
15952	CA	FA	0.998114
15809	CF	SF	0.997739
15954	CA	SA	0.995524
4642	iFF	iSF.1	0.995443
4643	iFF	iSF.2	0.995427
4641	iFF	iSF	0.995415
17108	SCA	xGA	0.995015
16963	SCF	xGF	0.994495
1002	GP	OTOI	0.993385
13008	Wide	iMiss	0.993229
16244	FA	xGA	0.992949
15956	CA	xGA	0.992542
10136	FOW.Down	iFOW	0.992496
10138	FOW.Down	iFOW.1	0.992464
16099	FF	xGF	0.991109
2177	Shifts	TOIX	0.991098
16532	SA	xGA	0.991094
4496	iCF.1	iFF	0.990960
2899	TOI%	TOI/GP.1	0.990941
10281	FOL.Down	iFOL	0.990924
2898	TOI%	TOI/GP	0.990914

In [10]:
```
BARIER = 0.9998

to_drop = [col for col in upper_corr.columns if any(upper_corr[col] > BARIER)]
df_clean = df.drop(columns=to_drop)
df_clean.head()
```

Out[10]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	1
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2

5 rows × 145 columns

Тут я добавил дополнительные фичи, потому что с ними дерево работает лучше. Удалил бакеты, поскольку модель сама может определить такого рода фичи.

In [11]:
```
df_features = df_clean.copy()

df_features['Born'] = pd.to_datetime(df_features['Born'], format='%y-%m-%d')
reference_date = pd.Timestamp('2016-10-01')
df_features['Age'] = (reference_date - df_features['Born']).dt.days / 365.25

df_features['Experience'] = reference_date.year - df_features['DftYr']
df_features['Age_squared'] = df_features['Age'] ** 2
df_features['G_per_GP'] = df_features['G'] / df_features['GP'].replace(0, 1)
df_features['A_per_GP'] = df_features['A'] / df_features['GP'].replace(0, 1)
df_features['PTS_per_GP'] = df_features['PTS'] / df_features['GP'].replace(0, 1)
df_features['Is_Drafted'] = df_features['DftYr'].notna().astype(int)
df_features['Physical_Impact'] = df_features['Wt'] * df_features['Ht']

features_to_drop = ['Born', 'Last Name', 'First Name', 'Nat', 'Pr/St', 'City']
df_features = df_features.drop(features_to_drop, axis=1)
df_features['Match'].value_counts()
```

Out[11]:
```
Match
0    870
1      4
Name: count, dtype: int64
```

In [12]:
```
has_nan = df_features.isnull().any()

columns_with_nan = has_nan[has_nan].index.tolist()

print("Столбцы, содержащие хотя бы один NaN:")
columns_with_nan
```

```
Столбцы, содержащие хотя бы один NaN:
```

Out[12]:
```
['DftYr',
 'DftRd',
 'Ovrl',
 'TOI%',
 'IPP%',
 'SH%',
 'SV%',
 'PDO',
 'F/60',
 'A/60',
 'Diff/60',
 'iCF',
 'iFF',
 'iSF',
 'ixG',
 'iSCF',
 'iRB',
 'iRS',
 'iDS',
 'sDist.1',
 'Pass',
 'iHA',
 'iHDf',
 'BLK%',
 '%FOT',

 'iPENT',
 'iPEND',
 'CF',
 'CA',
 'FF',
 'FA',
 'SF',
 'SA',
 'xGF',
 'xGA',
 'SCF',
 'SCA',
 'GF',
 'GA',
 'RBF',
 'RBA',
 'RSF',
 'RSA',
 'FOW',
 'FOL',
 'HF',
 'HA',
 'GVA',
 'TKA',
 'PENT',
 'PEND',
 'OTOI',
 'Pace',
 'GS',
 'GS/G',
 'Experience']
```

Тут обработку выбросов я добавлять не стал. Она ухудшает результаты модели.

In [13]:
```
from scipy.stats.mstats import winsorize
TARGET_NAME = 'Salary'

X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]

X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(
    X_i, y_i, test_size=0.2, random_state=42
)
```

Из предобработки я убрал нормализацию, потому что деревьям это не нужно

In [14]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
import pandas as pd

cat_cols = X_i.select_dtypes(include=['object', 'category']).columns.tolist()

cat_branch = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
```

```
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

num_cols = X_i.select_dtypes(include=["int64", "float64"]).columns

num_branch = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
])

ct = ColumnTransformer(
    transformers=[
        ("cat_proc", cat_branch, cat_cols),
        ("num_proc", num_branch, num_cols),
    ], remainder='drop'
)
```

Здесь я использую модель решающего дерева, также в гриде перебираю параметры дерева. Использование большой сетки только ухудшало результат, поэтому я от нёё отказался.

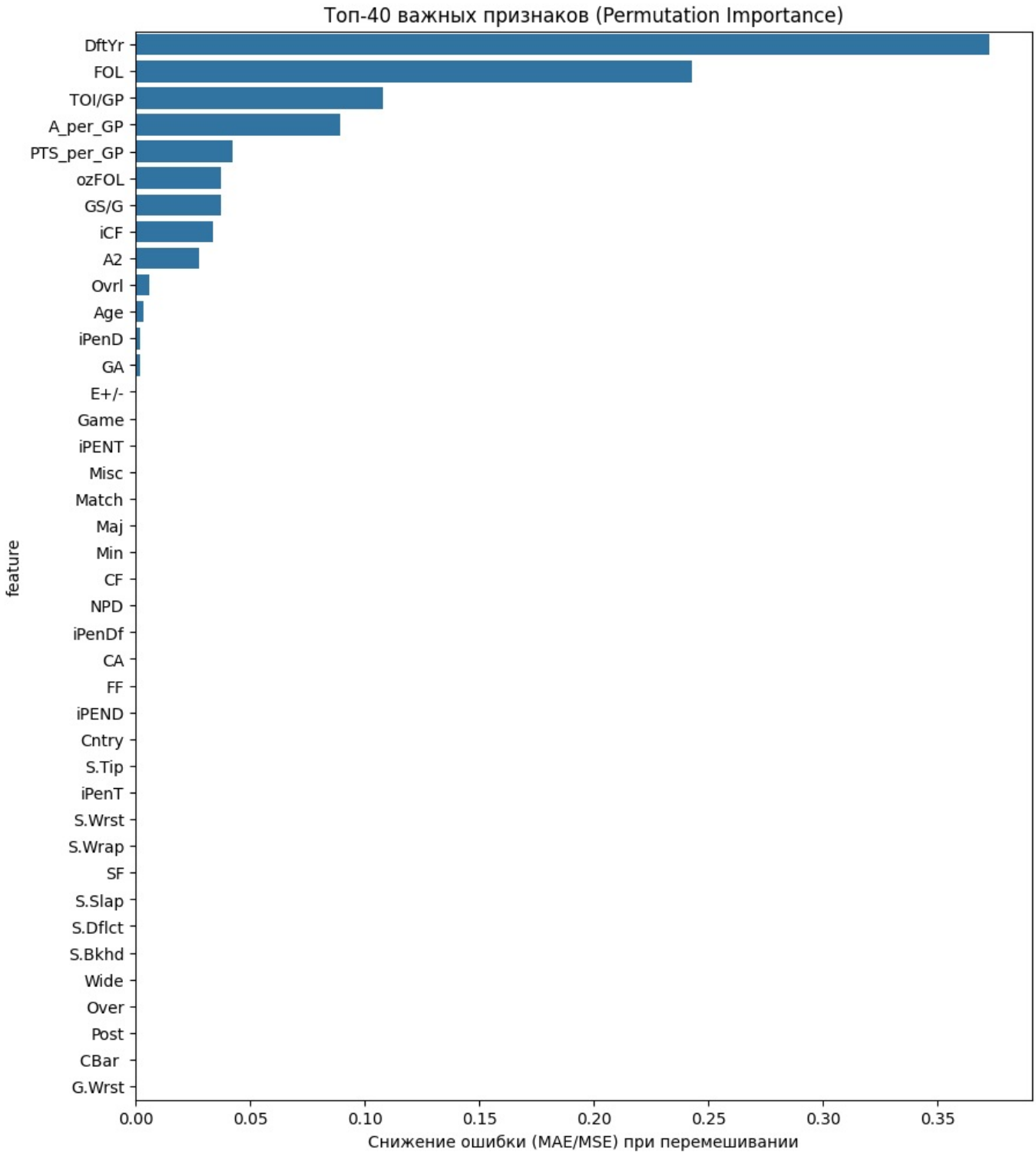
In [15]:
```
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.compose import TransformedTargetRegressor
import numpy as np

full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=DecisionTreeRegressor(random_state=42),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

# param_grid = {
#     'model_wrapper__regressor__max_depth': [3, 5],
#     'model_wrapper__regressor__min_samples_leaf': [10, 15],
#     'model_wrapper__regressor__min_samples_split': [2, 5, 10, 20],

#     'model_wrapper__regressor__max_features': [
#         None, 'sqrt', 'log2', 0.5
#     ],

#     'model_wrapper__regressor__criterion': ['squared_error', 'friedman_mse']
# }

param_grid = {
    'model_wrapper__regressor__max_depth': [3, 5],
    'model_wrapper__regressor__min_samples_leaf': [10, 15],
}

grid = RandomizedSearchCV(
    full_pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1,
    n_iter=50
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
y_pred_train = grid.best_estimator_.predict(X_train_i)

mae = mean_absolute_error(y_test_i, y_pred)
train_mae = mean_absolute_error(y_train_i, y_pred_train)
r2 = r2_score(y_test_i, y_pred)

print(f"\n--- Decision Tree Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"Train MAE: {train_mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

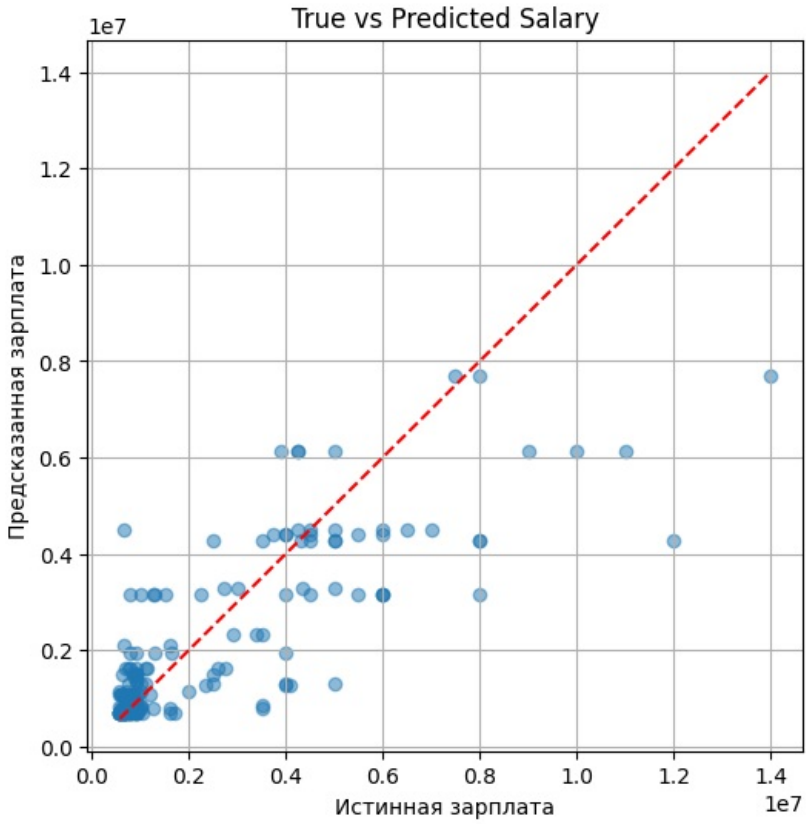
```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/model_selection/_search
.py:317: UserWarning: The total space of parameters 4 is smaller than n_iter=50. Running 4 iterations. For exhau
stive searches, use GridSearchCV.
  warnings.warn(
```

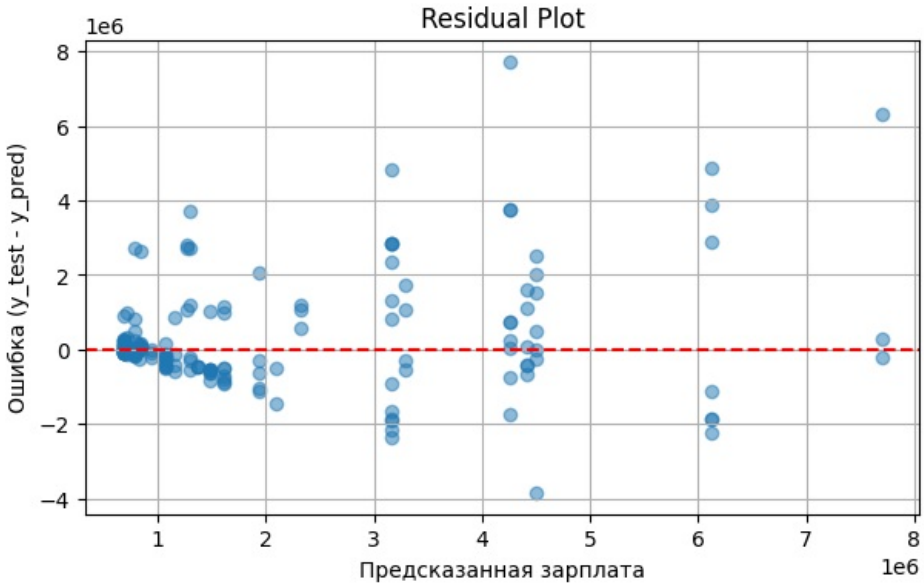
```
Лучшие параметры: {'model_wrapper__regressor__min_samples_leaf': 15, 'model_wrapper__regressor__max_depth': 5}

--- Decision Tree Regressor ---
MAE: 865388.517
Train MAE: 743009.212
R2 Score: 0.635
```

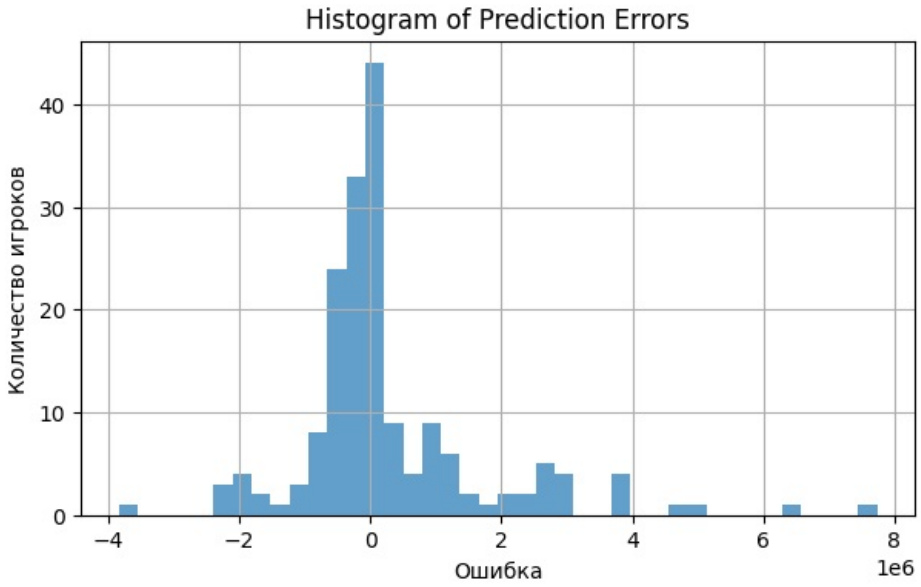
Получились хорошие результаты относительно предыдущих моделей. MAE выросла, а вот R2 упало. Модель стала хуже объяснять вариацию зарплат, но при этом в абсолютных значениях ошибка упала.

In [16]:
```
important_features(grid.best_estimator_, X_test_i, y_test_i)
true_vs_predicted(y_test_i, y_pred)
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```




True vs Predicted Salary
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Предсказанная зарплата
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e7
Истинная зарплата


Residual Plot
1e6
8
6
4
2
0
−2
−4
Ошибка (y_test - y_pred)
1
2
3
4
5
6
7
8
1e6
Предсказанная зарплата



Решающему дереву тяжело найти сложную зависимость, поэтому оно записывает тут большая дискретизация получается. Многим игрокам модель ставит одну и ту же зарплату. Но в целом даже неплохо, получилось лучше, чем в KNN.

In [17]:
```
original_float_format = pd.get_option('display.float_format')

pd.set_option('display.float_format', '{:.2f}'.format)

best_model = grid.best_estimator_

y_full_pred = best_model.predict(X_i)

results_df = pd.DataFrame({
    'True_Salary': y_i,
    'Predicted_Salary': y_full_pred
})

results_df['Absolute_Error'] = np.abs(results_df['True_Salary'] - results_df['Predicted_Salary'])

test_indices = X_test.index
final_results = pd.merge(
    df[['First Name', 'Last Name']],
    results_df,
    left_index=True,
    right_index=True
)

final_results_sorted = final_results.sort_values(by='Absolute_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---")
print(final_results_sorted.head(10).to_string())

pd.set_option('display.float_format', original_float_format)
```

```
--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---
    First Name Last Name  True_Salary  Predicted_Salary  Absolute_Error
542       Shea     Weber  12000000.00        4262683.20      7737316.80
103    Patrick      Kane  13800000.00        6123924.62      7676075.38
626       Anze   Kopitar  14000000.00        7698442.19      6301557.81
817     Steven   Stamkos   9500000.00        3289625.10      6210374.90
496   Jonathan     Toews  13800000.00        7698442.19      6101557.81
861      Corey     Perry  10000000.00        4493458.47      5506541.53
721       Sean   Monahan   6500000.00        1481300.80      5018699.20
564       Ryan  Callahan   6500000.00        1609227.50      4890772.50
208       P.K.    Subban  11000000.00        6123924.62      4876075.38
868       Loui  Eriksson   8000000.00        3164706.55      4835293.45
```

Ши Веббер конечно не играл на ту зарплату, которую получал. К 2016 году он был уже на закате карьеры, но всё еще получал много по старому контракту.

In [18]:
```python
final_results['Signed_Error'] = final_results['Predicted_Salary'] - final_results['True_Salary']

overestimated_players = final_results[final_results['Signed_Error'] > 0].copy()

top_overestimated = overestimated_players.sort_values(by='Signed_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---")

print(top_overestimated[['First Name', 'Last Name',
                         'True_Salary', 'Predicted_Salary', 'Signed_Error']].head(10).to_string())
```

```
--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---
    First Name Last Name  True_Salary  Predicted_Salary  Signed_Error
665     Justin   Schultz    1400000.0      6.123925e+06  4.723925e+06
849        Sam    Gagner     650000.0      4.493458e+06  3.843458e+06
223       Kyle    Turris    4000000.0      7.698442e+06  3.698442e+06
532      Radim    Vrbata    1000000.0      4.493458e+06  3.493458e+06
659     Artemi   Panarin     925000.0      4.410530e+06  3.485530e+06
830    Patrick     Eaves    1000000.0      4.262683e+06  3.262683e+06
313      Brian    Strait     600000.0      3.289625e+06  2.689625e+06
302        Tom     Pyatt     800000.0      3.164707e+06  2.364707e+06
412       Mark  Barberio     800000.0      3.164707e+06  2.364707e+06
863      Brian  Dumoulin     800000.0      3.164707e+06  2.364707e+06
```

А тут интересно два человека: Дмитрий Орлов -- игрок Вашингтон Кэпиталз, надёжный защитник, который уже чз пару лет возьмет кубок Стэнли вместе с Александром Овечкиным. А второй -- это Марк Барберио, который сейчас играет в Северстали. На мой взгляд, он получал именно те деньги, которые заслужил в этом сезоне: 26 игр, 4 очка, 15 минут среднее игровое время. Слабо работает решающее дерево.

My implementation

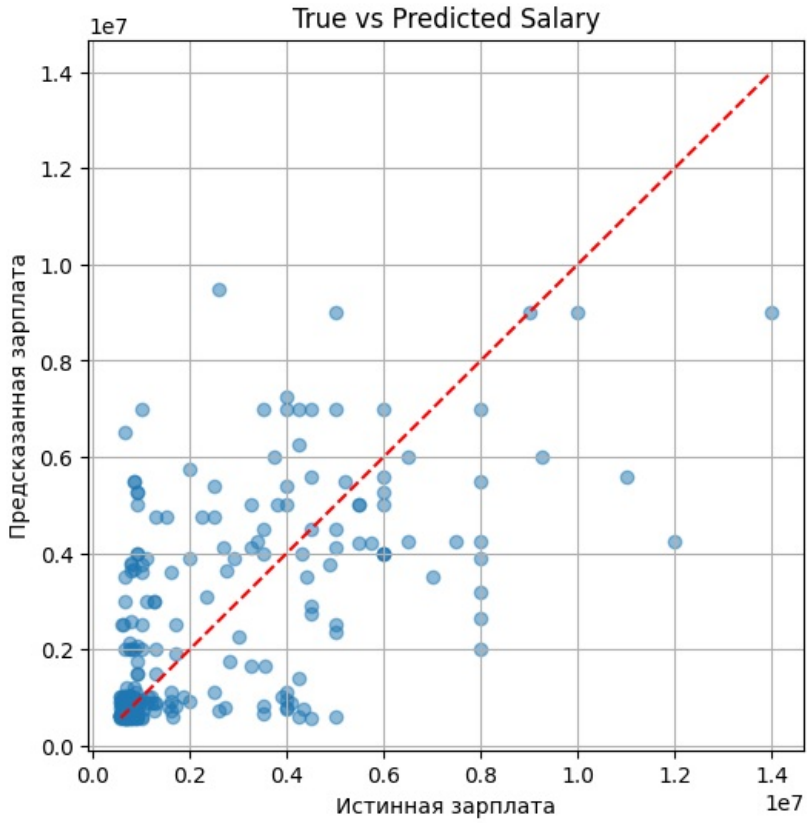
In [19]:
```python
import numpy as np
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted

class Node:
    def __init__(self, feature_index=None, threshold=None, left=None, right=None, value=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

class MyDecisionTreeRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, max_depth=None, min_samples_leaf=1):
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
        self.root = None

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.n_features_ = X.shape[1]
        self.root = self._build_tree(X, y)
        return self

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)
        return np.array([self._make_prediction(x, self.root) for x in X])

    def _build_tree(self, X, y, depth=0):
        num_samples, num_features = X.shape
        if (self.max_depth is not None and depth >= self.max_depth) or \
           (num_samples < self.min_samples_leaf * 2) or \
           (np.var(y) == 0):
```

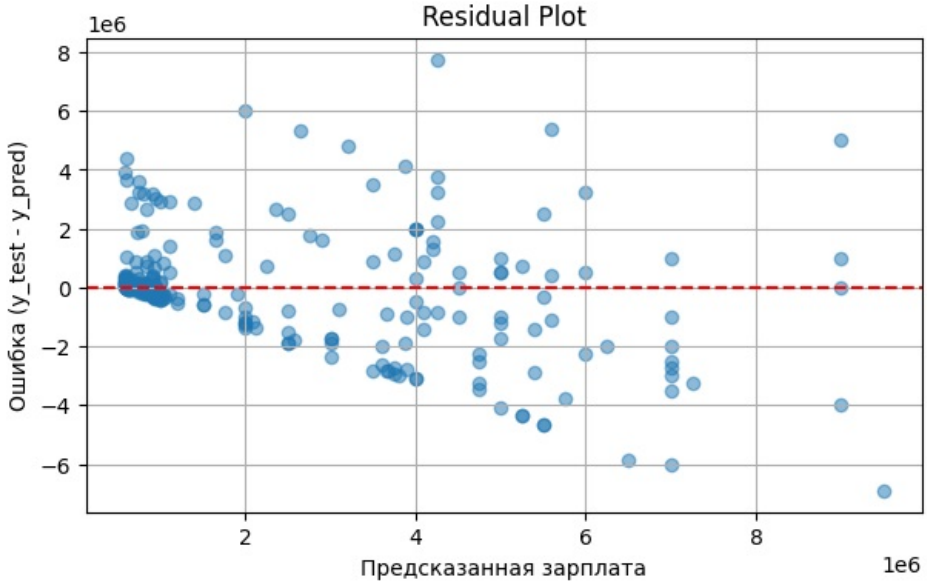
```
            return Node(value=np.mean(y))

        best_split = self._get_best_split(X, y, num_features)

        if best_split["gain"] > 0:
            left_subtree = self._build_tree(best_split["X_left"], best_split["y_left"], depth + 1)
            right_subtree = self._build_tree(best_split["X_right"], best_split["y_right"], depth + 1)
            return Node(
                feature_index=best_split["feature_index"],
                threshold=best_split["threshold"],
                left=left_subtree,
                right=right_subtree
            )
        return Node(value=np.mean(y))

    def _get_best_split(self, X, y, num_features):
        best_split = {"gain": -1, "feature_index": None, "threshold": None}
        max_gain = -float("inf")
        current_uncertainty = np.var(y) * len(y)

        for feature_index in range(num_features):
            feature_values = X[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            for threshold in possible_thresholds:
                left_mask = feature_values <= threshold
                right_mask = ~left_mask
                if np.sum(left_mask) < self.min_samples_leaf or np.sum(right_mask) < self.min_samples_leaf:
                    continue

                y_left, y_right = y[left_mask], y[right_mask]

                current_split_uncertainty = (len(y_left) * np.var(y_left)) + (len(y_right) * np.var(y_right))
                gain = current_uncertainty - current_split_uncertainty

                if gain > max_gain:
                    max_gain = gain
                    best_split = {
                        "feature_index": feature_index,
                        "threshold": threshold,
                        "X_left": X[left_mask],
                        "y_left": y_left,
                        "X_right": X[right_mask],
                        "y_right": y_right,
                        "gain": gain
                    }
        return best_split

    def _make_prediction(self, x, node):
        if node.value is not None:
            return node.value
        if x[node.feature_index] <= node.threshold:
            return self._make_prediction(x, node.left)
        else:
            return self._make_prediction(x, node.right)
```

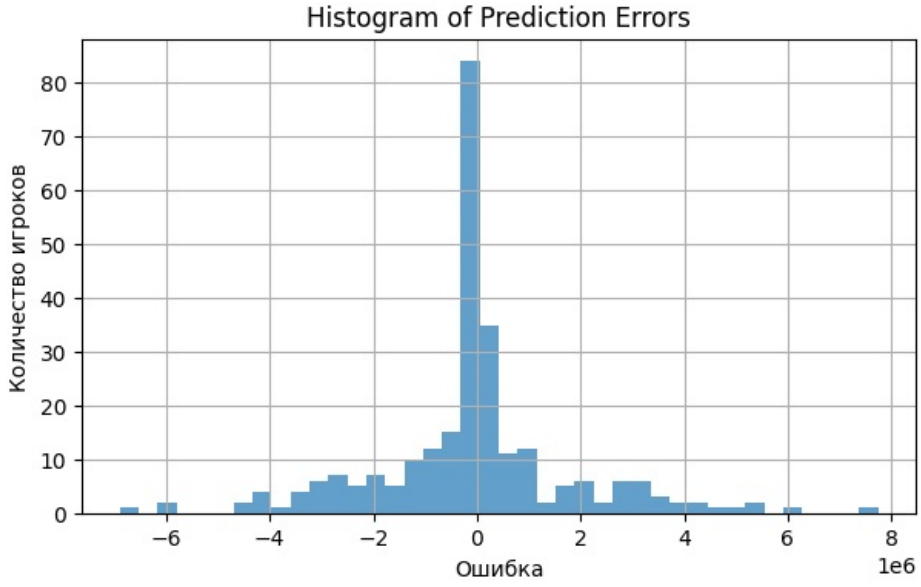
In [20]:
```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

dt = DecisionTreeRegressor(random_state=42, max_depth=None, min_samples_leaf=1)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, np.full(len(y_test), y_train.mean()))

print("--- Decision Tree Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
--- Decision Tree Regressor ---
MAE: 1220967.395
R2 Score: 0.322
```

In [21]:
```
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```



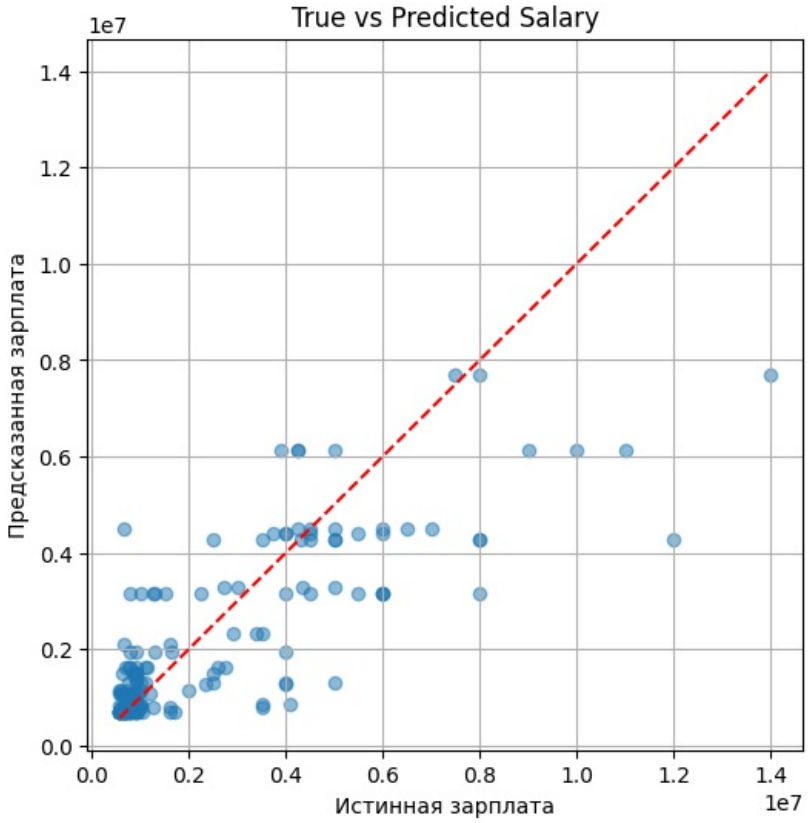


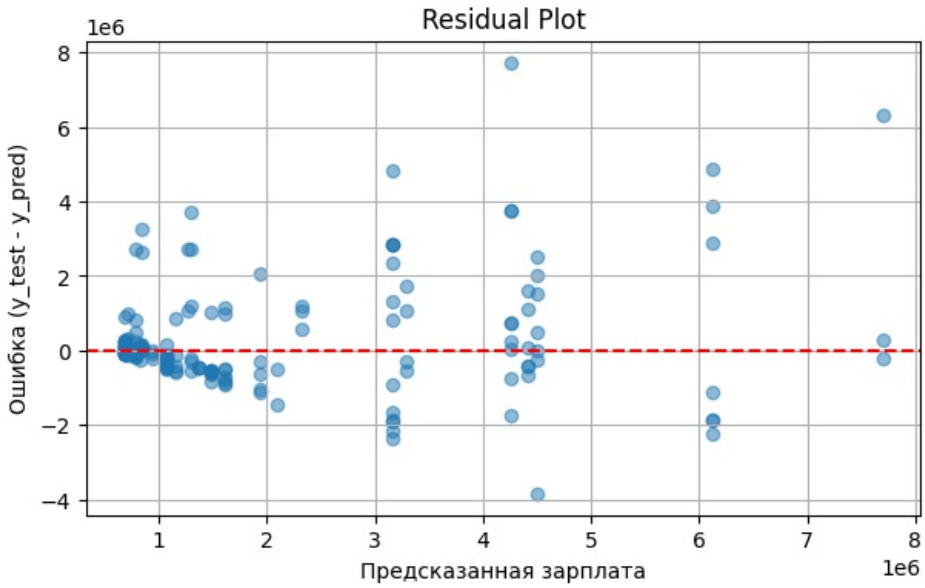


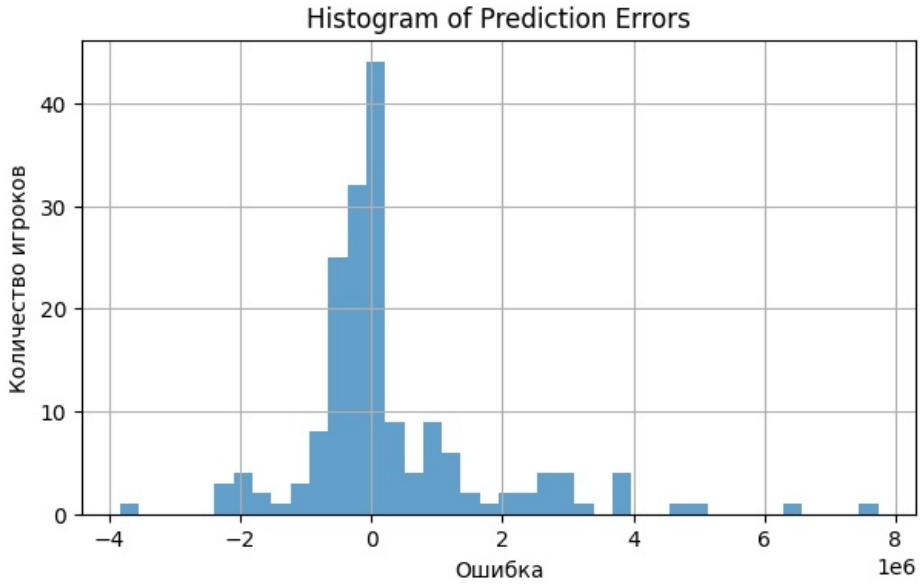
In [22]:
```
full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
```

```
        regressor=MyDecisionTreeRegressor(),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'model_wrapper__regressor__max_depth': [3, 5],
    'model_wrapper__regressor__min_samples_leaf': [10, 15],
}

grid = GridSearchCV(
    full_pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
y_pred_train = grid.best_estimator_.predict(X_train_i)

mae = mean_absolute_error(y_test_i, y_pred)
train_mae = mean_absolute_error(y_train_i, y_pred_train)
r2 = r2_score(y_test_i, y_pred)

print(f"\n--- Decision Tree Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"Train MAE: {train_mae:.3f}")
print(f"R2 Score: {r2:.3f}")
true_vs_predicted(y_test_i, y_pred)
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Лучшие параметры: {'model_wrapper__regressor__max_depth': 5, 'model_wrapper__regressor__min_samples_leaf': 15}

--- Decision Tree Regressor ---
MAE: 870330.527
Train MAE: 743009.212
R2 Score: 0.632
```







Моя модель не сильно хуже, чем реализация scikit-learn, но при этом сильно медленнее. В моей реализации отсутсвует pruning, плюс алгоритм поиска значений медленный. Но тут совсем незначительные изменения получаются.

	Base Decision Tree	Decision Tree	My Decision Tree
MAE	1220967.395	865388.517	870330.527
R2	0.322	0.635	0.632

Классификация

Задача: вычислить мошенника на страховых выплатах с использованием модели решающего дерева

Для выполнения лабораторной работы были выбраны метрики F1-score и ROC-AUC, так как исследуемый датасет является несбалансированным. Метрика Accuracy в данном случае неинформативна, так как модель, предсказывающая всем класс ‘0’ (не фрод), может иметь высокую Accuracy, но будет бесполезна. F1-score позволит контролировать баланс между ложными срабатываниями и пропуском мошенников.

Baseline Desicion Tree

In [23]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                    "buntyshah/auto-insurance-claims-data/versions/1",
                                    "insurance_claims.csv")
```

```
df
```

Out[23]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 40 columns

In [24]:
```
df_clean = df.copy()

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
df_clean["property_damage"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
df_clean = df_clean.fillna(0)
```

In [25]:
```
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean.head()
```

Out[25]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium
0	328	48	521585	940	2	1	1000	1406.91
1	228	42	342868	635	1	1	2000	1197.22
2	134	29	687698	413	2	0	2000	1413.14
3	256	41	227811	19	0	1	2000	1415.74
4	228	44	367455	922	0	2	1000	1583.91

5 rows × 40 columns

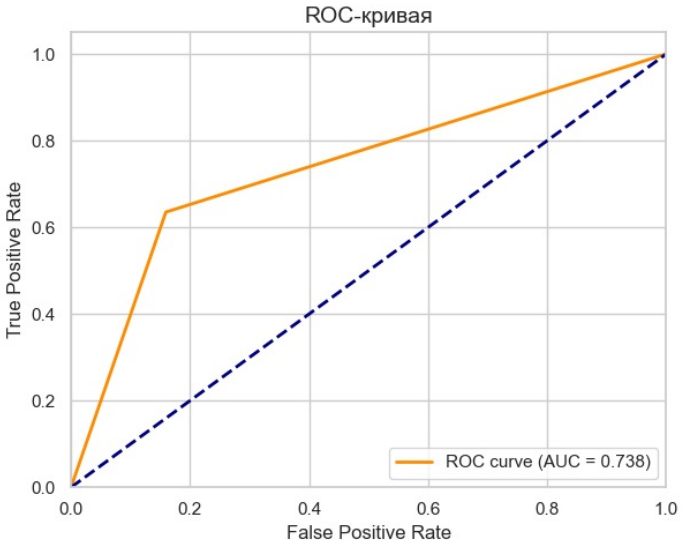
In [26]:
```
from sklearn.model_selection import train_test_split
drop_dates = ["policy_bind_date", "incident_date"]
df_clean = df_clean.drop(drop_dates, axis=1)
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

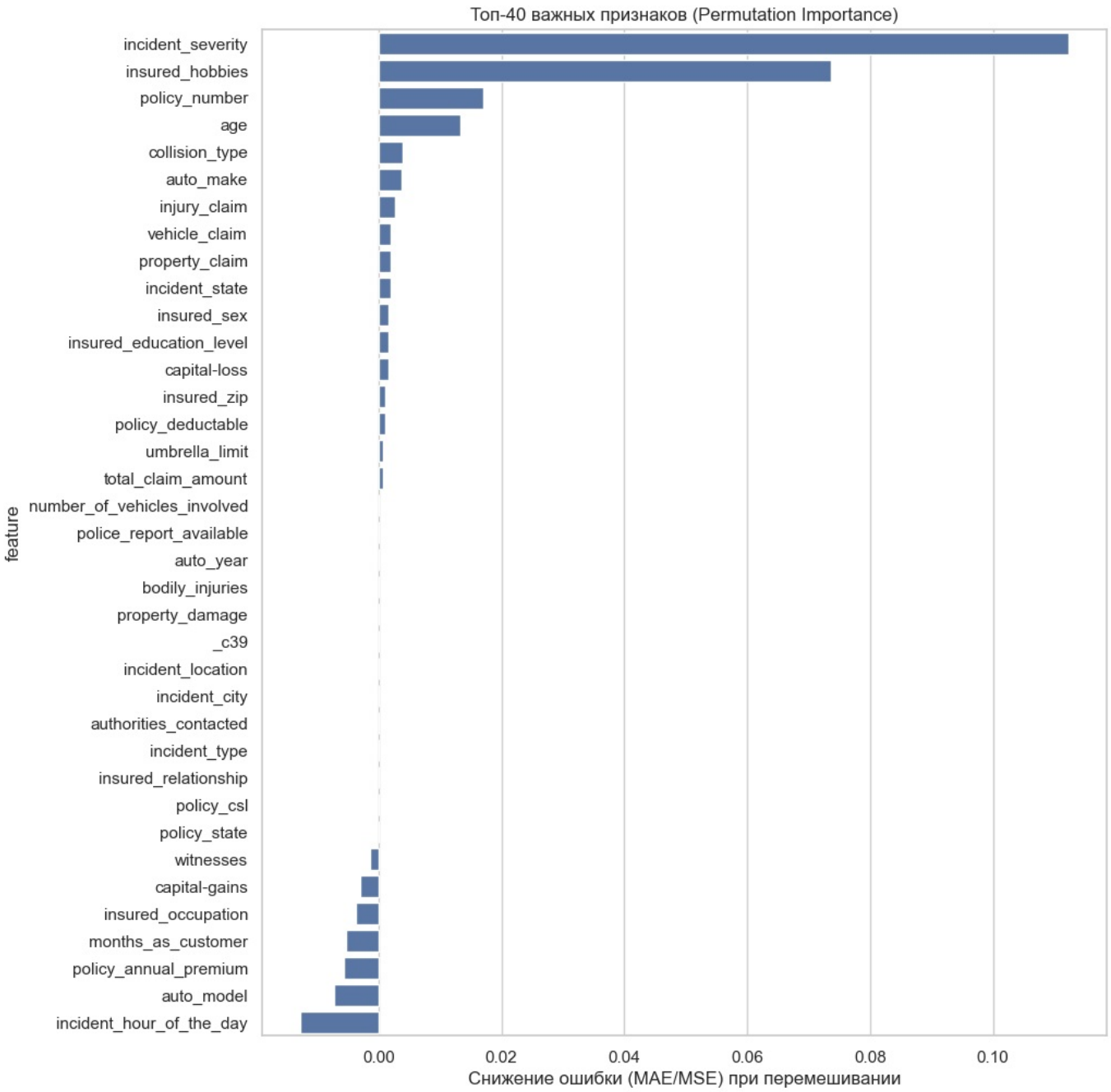
In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, f1_score, roc_auc_score, confusion_matrix, roc_curve

model = DecisionTreeClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
F1-score (класс 1): 0.5987
ROC-AUC: 0.7379
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.84      0.86       226
           1       0.57      0.64      0.60        74

    accuracy                           0.79       300
   macro avg       0.72      0.74      0.73       300
weighted avg       0.80      0.79      0.79       300

==================================================
```

In [28]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc
def graphics(y_test, y_pred, y_prob):
    sns.set(style="whitegrid")
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", cbar=False,
                xticklabels=['Предск: 0', 'Предск: 1'],
                yticklabels=['Факт: 0', 'Факт: 1'])
    plt.title("Матрица ошибок", fontsize=14)
    plt.ylabel("Реальность")
    plt.xlabel("Предсказание")

    plt.subplot(1, 2, 2)
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC-кривая', fontsize=14)
    plt.legend(loc="lower right")

    plt.tight_layout()
    plt.show()
def feature_important(model, X_test, y_test):

    perm_result = permutation_importance(
        model,
        X_test,
        y_test,
        n_repeats=10,
        random_state=42,
        n_jobs=-1
    )

    perm_df = pd.DataFrame({
        'feature': X_test.columns,
        'importance': perm_result.importances_mean
    })

    top_40_perm = perm_df.sort_values(by='importance', ascending=False).head(40)

    plt.figure(figsize=(10, 12))
    sns.barplot(data=top_40_perm, x='importance', y='feature')
    plt.title("Топ-40 важных признаков (Permutation Importance)")
    plt.xlabel("Снижение ошибки (MAE/MSE) при перемешивании")
    plt.show()
graphics(y_test, y_pred, y_prob)
feature_important(model, X_test, y_test)
```



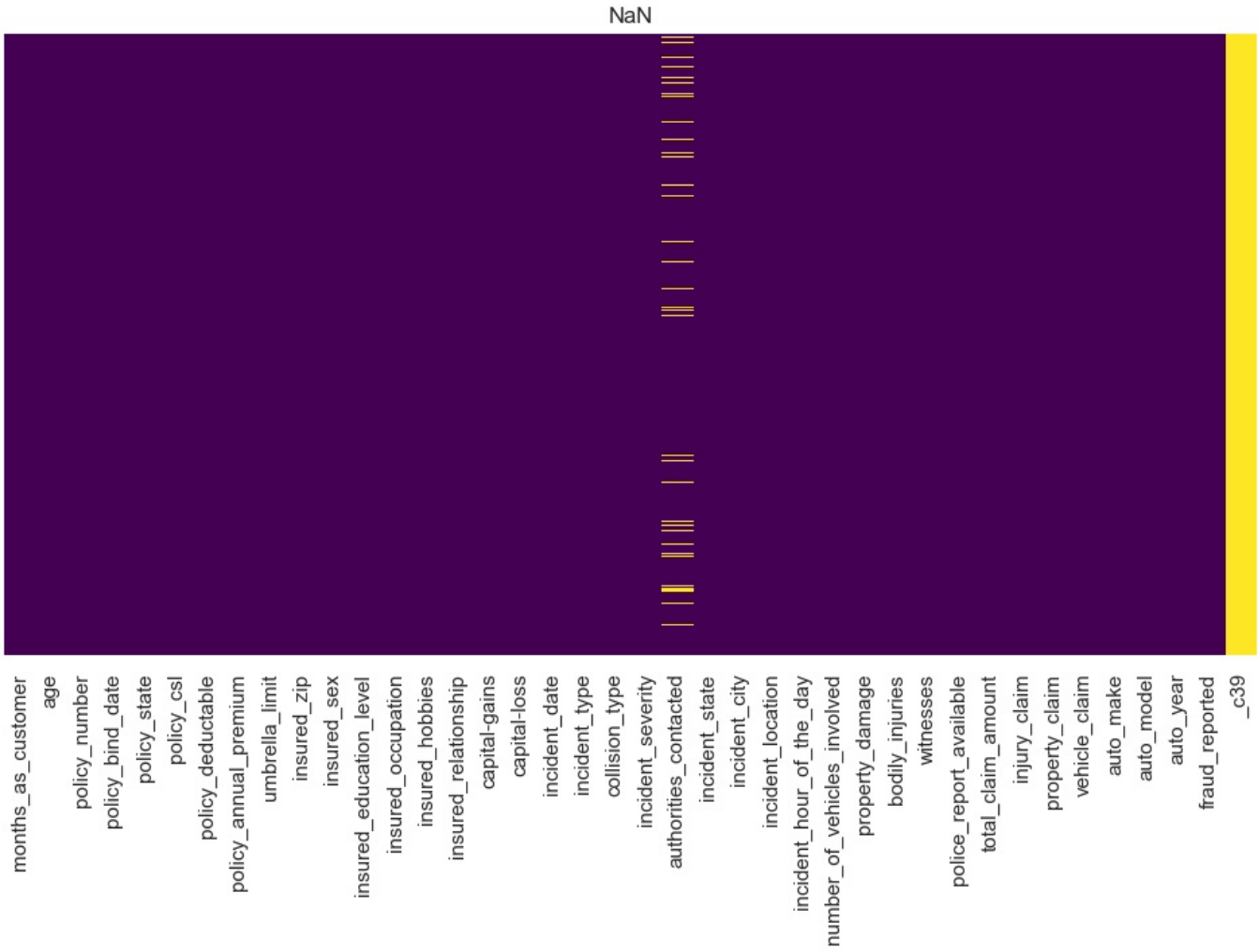


Решающее дерево даже без преобработки показывает очень хорошие результаты. Вероятно это связано с тем, что модель не зависима от нормализации данных как KNN и Logistic Resgressor.

Improved Decision Tree

In [29]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter
```

```
df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                    "buntyshah/auto-insurance-claims-data/versions/1",
                                    "insurance_claims.csv")
df.head()

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [30]:
```
df_clean = df.copy()
df_clean["fraud_reported"].value_counts()
```

Out[30]:
```
fraud_reported
N    753
Y    247
Name: count, dtype: int64
```

In [31]:
```
display(df_clean["police_report_available"].unique())
display(df_clean["property_damage"].unique())
```

```
array(['YES', '?', 'NO'], dtype=object)
array(['YES', '?', 'NO'], dtype=object)
```

In [32]:
```
df_clean = df_clean.drop(columns=["_c39"])

df_clean["authorities_contacted"] = df_clean["authorities_contacted"].fillna("No Contact")

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0, '?': -1})
df_clean["property_damage"] = df_clean["property_damage"].map({'YES': 1, 'NO': 0, '?': -1})

dates_cols = ["policy_bind_date", "incident_date"]
for c in dates_cols:
    df_clean[c] = pd.to_datetime(df_clean[c])

df_clean
```

Out[32]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 39 columns

In [33]:
```
num_cols = df_clean.select_dtypes(include=["int64", "float64"]).columns.tolist()
df_clean[num_cols].describe().T
```

Out[33]:

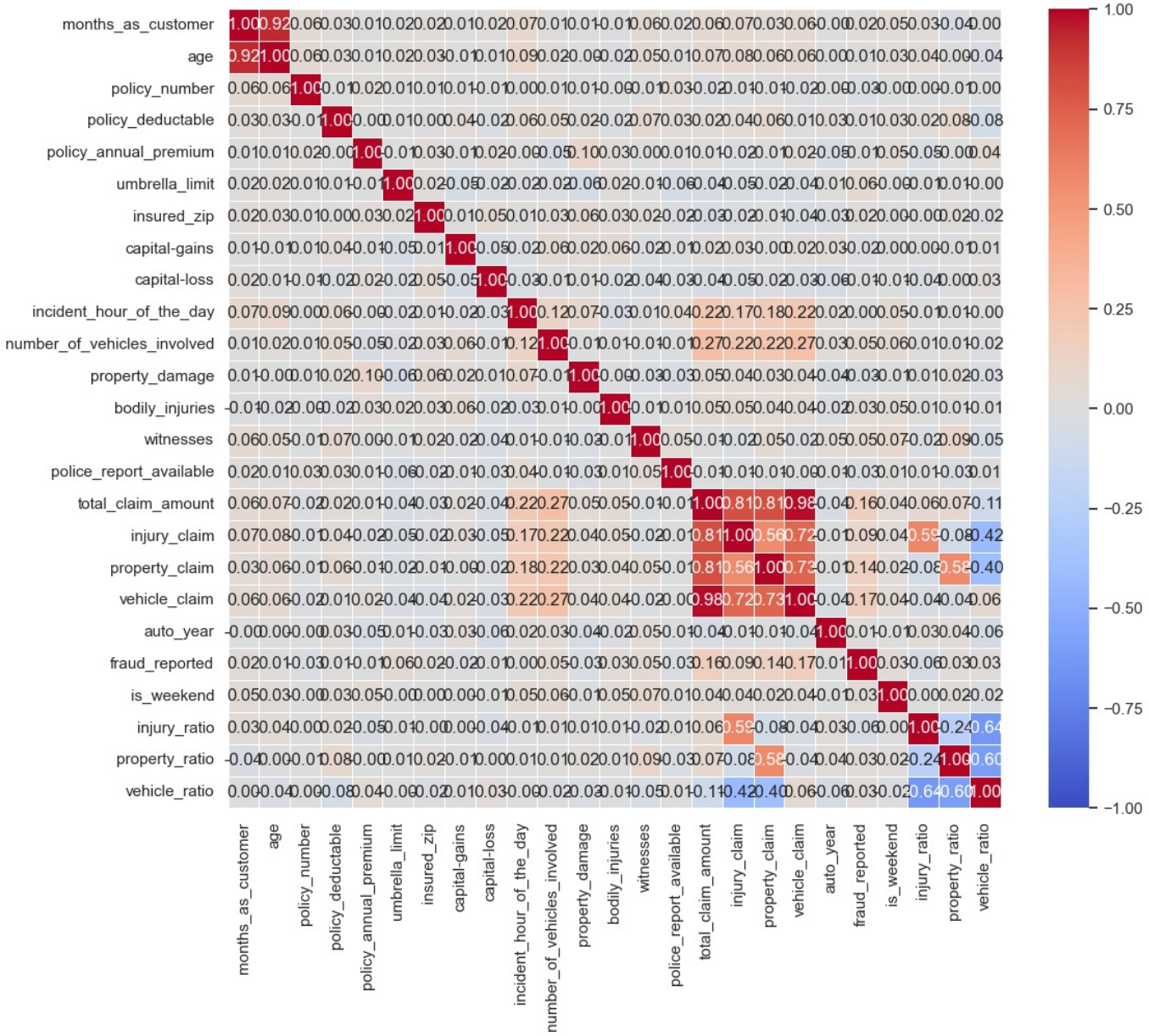
	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
property_damage	1000.0	-5.800000e-02	8.119700e-01	-1.00	-1.0000	0.0	1.000	1.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
police_report_available	1000.0	-2.900000e-02	8.104417e-01	-1.00	-1.0000	0.0	1.000	1.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00
fraud_reported	1000.0	2.470000e-01	4.314825e-01	0.00	0.0000	0.0	0.000	1.00

In [34]:
```
import numpy as np

df_pr = df_clean.copy()

median_value = df_pr.loc[df_pr['umbrella_limit'] != -100000, 'umbrella_limit'].median()
df_pr.loc[df_pr['umbrella_limit'] == -100000, 'umbrella_limit'] = median_value
```

In [35]:
```
df_features = df_pr.copy()

df_clean["policy_tenure_months"] = ((df_clean["incident_date"] - df_clean["policy_bind_date"]).dt.days / 30).as

df_features["incident_year"] = df_features["incident_date"].dt.year
df_features["incident_month"] = df_features["incident_date"].dt.month
df_features["incident_dow"] = df_features["incident_date"].dt.dayofweek
df_features["is_weekend"] = df_features["incident_dow"].isin([5, 6]).astype(int)

df_features["injury_ratio"] = df_features["injury_claim"] / (df_features["total_claim_amount"] + 1e-3)
df_features["property_ratio"] = df_features["property_claim"]  / (df_features["total_claim_amount"] + 1e-3)
```

```
df_features["vehicle_ratio"] = df_features["vehicle_claim"] / (df_features["total_claim_amount"] + 1e-3)

drop_dates = ["policy_bind_date", "incident_date"]
df_features = df_features.drop(drop_dates, axis=1)
```

In [36]:
```
num_cols = df_features.select_dtypes(include=["int64", "float64"]).columns.tolist()

plt.figure(figsize=(12, 10))
correlation_matrix = df_features[num_cols].corr()

sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            vmin=-1, vmax=1,
            linewidths=0.5)

plt.show()
```



In [37]:
```
from sklearn.model_selection import train_test_split
X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_i, y_i, test_size=0.25, random_state=42, stratify
```

Я удалил нормализацию из пайплайна, поскольку она не нужна для решающего дерева. Добавил параметры в сетку для решающего дерева. Также я вернул сюда OneHot Encoder для лучшей интерпретации результатов, хоть ROC_AUC и незначительно падает.

Поскольку я убрал нормализацию, я убрал и KNN импутер, который нормально работает только с нормализованными данными. Теперь используется просто медиана.

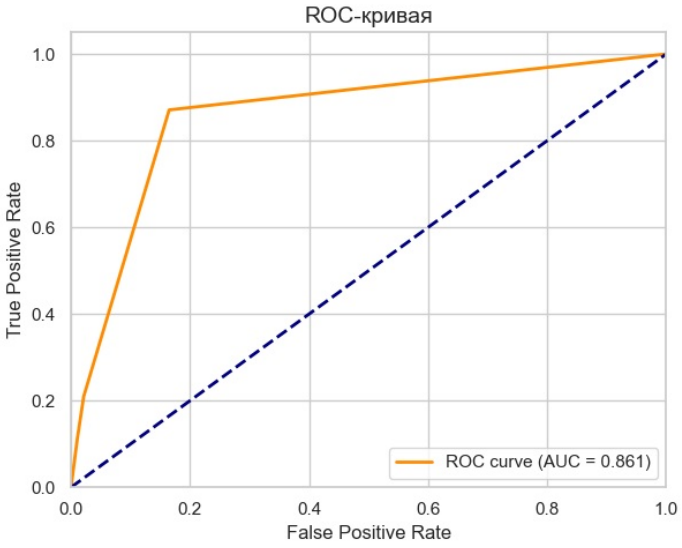
In [38]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

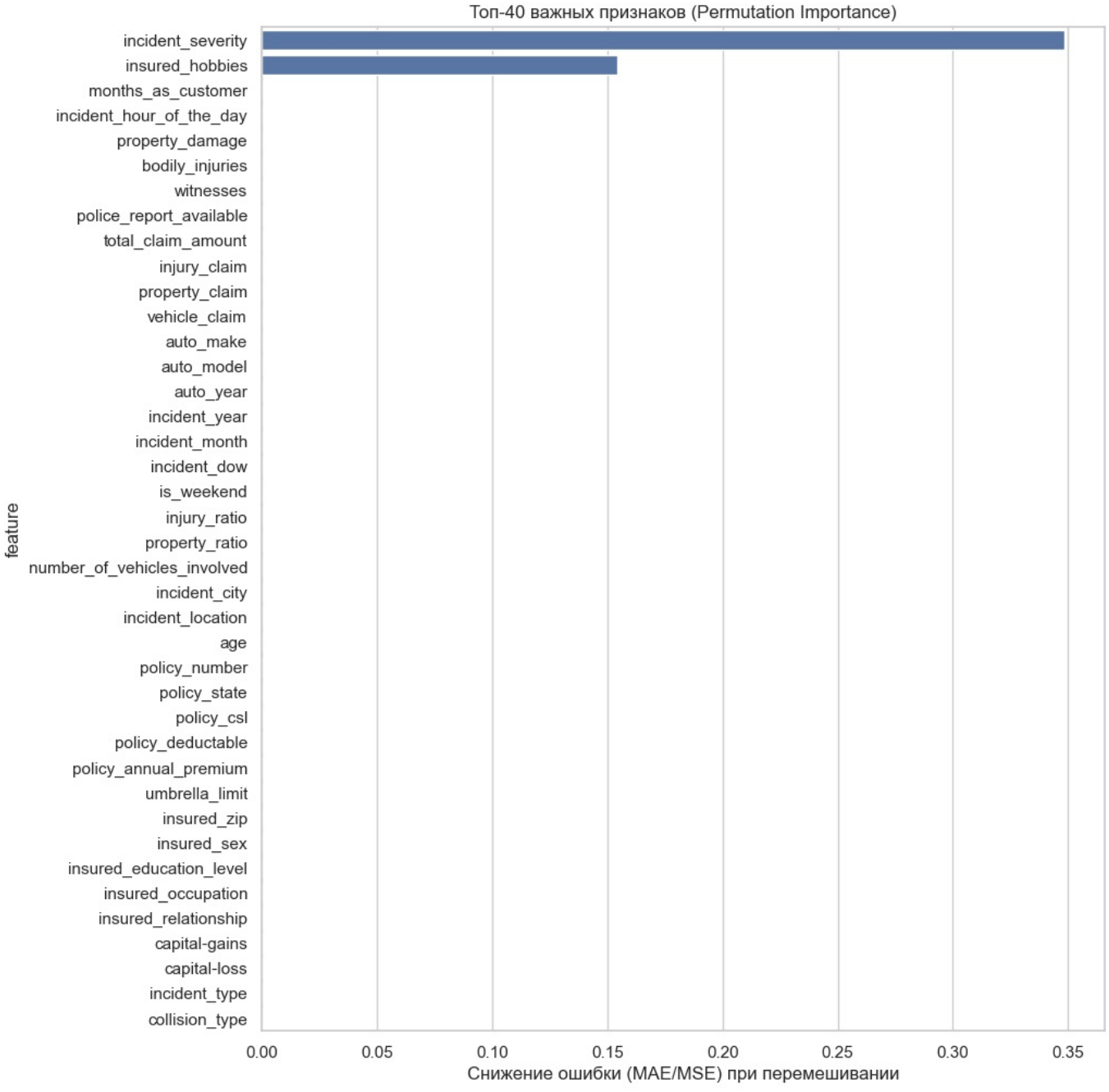
```python
from sklearn.preprocessing import OneHotEncoder, QuantileTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import f1_score, roc_auc_score, classification_report

num_cols = X_train_i.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = X_train_i.select_dtypes(include=["object", "category"]).columns.tolist()

categorical_pipe = Pipeline([
    ("onehot", OneHotEncoder(
        handle_unknown='infrequent_if_exist',
        sparse_output=False,
        min_frequency=2
    ))
])

ct = ColumnTransformer([
    ("cat", categorical_pipe, cat_cols)
])

model_pipe = Pipeline([
    ('ct', ct),
    ('model', DecisionTreeClassifier(random_state=42))
])

param_grid = {
    'model__max_depth': [5, 8, 10, 12, 15, None],
    'model__min_samples_leaf': [2, 4, 8, 10],
    'model__ccp_alpha': [0.0, 0.001, 0.005, 0.01, 0.02],
    'model__criterion': ['gini', 'entropy']
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(
    model_pipe,
    param_grid,
    cv=cv,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print(f"Лучшие параметры: {grid.best_params_}")

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print("-" * 30)
print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
print("="*50)
```

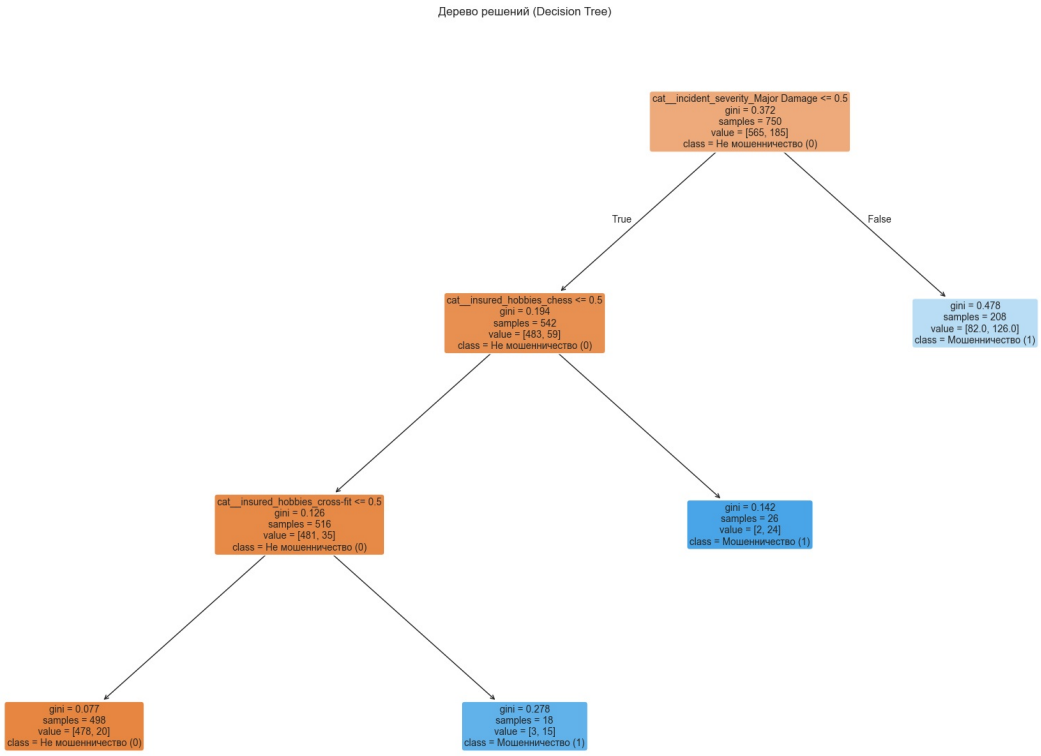
```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
Лучшие параметры: {'model__ccp_alpha': 0.005, 'model__criterion': 'gini', 'model__max_depth': 5, 'model__min_sam
ples_leaf': 10}
------------------------------
F1-score (класс 1): 0.7347
ROC-AUC: 0.8611
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.84      0.89       188
           1       0.64      0.87      0.73        62

    accuracy                           0.84       250
   macro avg       0.79      0.85      0.81       250
weighted avg       0.87      0.84      0.85       250

==================================================
```

In [39]:

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.inspection import permutation_importance

def tree_plot(fitted_grid):
    best_dt_model = fitted_grid.best_estimator_['model']
    ct_fitted = fitted_grid.best_estimator_['ct']

    feature_names = ct_fitted.get_feature_names_out()

    import matplotlib.pyplot as plt
    from sklearn.tree import plot_tree

    plt.figure(figsize=(25, 15))

    plot_tree(
        best_dt_model,
        feature_names=feature_names,
        class_names=['Не мошенничество (0)', 'Мошенничество (1)'],
        filled=True,
        rounded=True,
        fontsize=10
    )
    plt.title("Дерево решений (Decision Tree)")
    plt.show()

graphics(y_test_i, y_pred, y_prob)
feature_important(grid, X_test_i, y_test_i)
tree_plot(grid)
```




Топ-40 важных признаков (Permutation Importance)
incident_severity
insured_hobbies
months_as_customer
incident_hour_of_the_day
property_damage
bodily_injuries
witnesses
police_report_available
total_claim_amount
injury_claim
property_claim
vehicle_claim
auto_make
auto_model
auto_year
incident_year
incident_month
incident_dow
is_weekend
injury_ratio
property_ratio
number_of_vehicles_involved
incident_city
incident_location
age
policy_number
policy_state
policy_csl
policy_deductable
policy_annual_premium
umbrella_limit
insured_zip
insured_sex
insured_education_level
insured_occupation
insured_relationship
capital-gains
capital-loss
incident_type
collision_type
feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Снижение ошибки (MAE/MSE) при перемешивании



Решающее дерево почти не теряет мошенников, это очень хорошо. Также получилась интересная форма ROC_AUC кривой. Тут мы видим острые углы, поскольку решающее дерево имеет низкую разрешающую способность, что мы и видим по построению дерева.

Довольно забавно, что хобби так сильно влияют на то, является ли человек мошенник xD

My implementation

In [40]:
```python
import numpy as np
from collections import Counter
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
from sklearn.utils.multiclass import unique_labels

class Node:
    def __init__(self, feature_index=None, threshold=None, left=None, right=None,
                 value=None, proba=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
        self.proba = proba

class MyDecisionTreeClassifier(ClassifierMixin, BaseEstimator):
    def __init__(self, max_depth=None, min_samples_leaf=1, criterion='gini'):
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
        self.criterion = criterion
        self.root = None

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)
        self.n_classes_ = len(self.classes_)
        self.n_features_ = X.shape[1]

        self.root = self._build_tree(X, y)
        return self

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)
        return np.array([self._make_prediction(x, self.root) for x in X])

    def predict_proba(self, X):
        check_is_fitted(self)
        X = check_array(X)
```

```
        probas = []
        for x in X:
            node_probs = self._get_node_proba(x, self.root)
            row_proba = [node_probs.get(c, 0.0) for c in self.classes_]
            probas.append(row_proba)

        return np.array(probas)

    def _build_tree(self, X, y, depth=0):
        num_samples, num_features = X.shape
        unique_classes = len(np.unique(y))

        if (self.max_depth is not None and depth >= self.max_depth) or \
           (num_samples < self.min_samples_leaf * 2) or \
           (unique_classes == 1):
            return self._create_leaf_node(y)

        best_split = self._get_best_split(X, y, num_features)

        if best_split["gain"] > 0:
            left_subtree = self._build_tree(best_split["X_left"], best_split["y_left"], depth + 1)
            right_subtree = self._build_tree(best_split["X_right"], best_split["y_right"], depth + 1)
            return Node(
                feature_index=best_split["feature_index"],
                threshold=best_split["threshold"],
                left=left_subtree,
                right=right_subtree
            )

        return self._create_leaf_node(y)

    def _create_leaf_node(self, y):
        counts = Counter(y)
        most_common = counts.most_common(1)[0][0]
        total = len(y)
        probs = {cls: count / total for cls, count in counts.items()}
        return Node(value=most_common, proba=probs)

    def _get_best_split(self, X, y, num_features):
        best_split = {"gain": -1, "feature_index": None, "threshold": None}
        max_info_gain = -float("inf")
        parent_impurity = self._calculate_impurity(y)

        for feature_index in range(num_features):
            feature_values = X[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            for threshold in possible_thresholds:
                left_mask = feature_values <= threshold
                right_mask = ~left_mask
                if np.sum(left_mask) < self.min_samples_leaf or np.sum(right_mask) < self.min_samples_leaf:
                    continue

                y_left, y_right = y[left_mask], y[right_mask]
                n = len(y)
                n_l, n_r = len(y_left), len(y_right)
                child_impurity = (n_l / n) * self._calculate_impurity(y_left) + \
                                 (n_r / n) * self._calculate_impurity(y_right)
                info_gain = parent_impurity - child_impurity

                if info_gain > max_info_gain:
                    max_info_gain = info_gain
                    best_split = {
                        "feature_index": feature_index,
                        "threshold": threshold,
                        "X_left": X[left_mask],
                        "y_left": y_left,
                        "X_right": X[right_mask],
                        "y_right": y_right,
                        "gain": info_gain
                    }
        return best_split

    def _calculate_impurity(self, y):
        if len(y) == 0: return 0
        counts = np.unique(y, return_counts=True)[1]
        probabilities = counts / len(y)
        if self.criterion == 'gini':
            return 1 - np.sum(probabilities ** 2)
        elif self.criterion == 'entropy':
            return -np.sum(probabilities * np.log2(probabilities + 1e-9))
        else:
            raise ValueError("Unknown criterion. Use 'gini' or 'entropy'")
```

```
    def _make_prediction(self, x, node):
        if node.value is not None: return node.value
        if x[node.feature_index] <= node.threshold:
            return self._make_prediction(x, node.left)
        return self._make_prediction(x, node.right)

    def _get_node_proba(self, x, node):
        if node.proba is not None: return node.proba
        if x[node.feature_index] <= node.threshold:
            return self._get_node_proba(x, node.left)
        return self._get_node_proba(x, node.right)
```

Без предобработки

In [41]:
```
model = MyDecisionTreeClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
F1-score (класс 1): 0.6065
ROC-AUC: 0.7423
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.85      0.86       226
           1       0.58      0.64      0.61        74

    accuracy                           0.80       300
   macro avg       0.73      0.74      0.73       300
weighted avg       0.80      0.80      0.80       300

==================================================
```

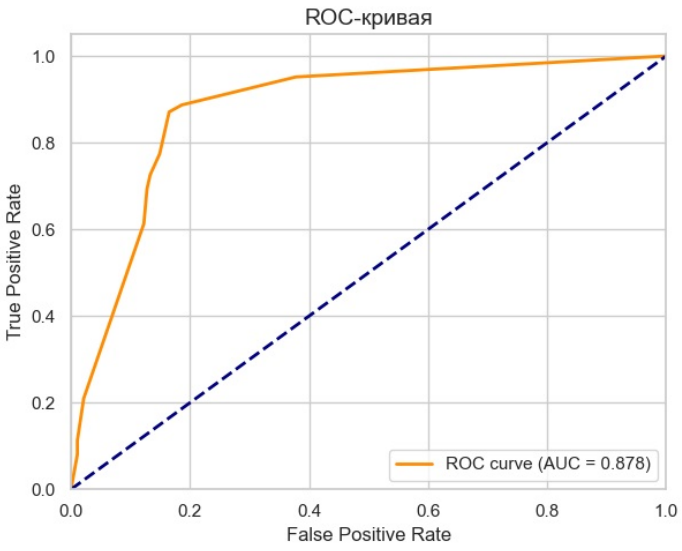
С предобработкой

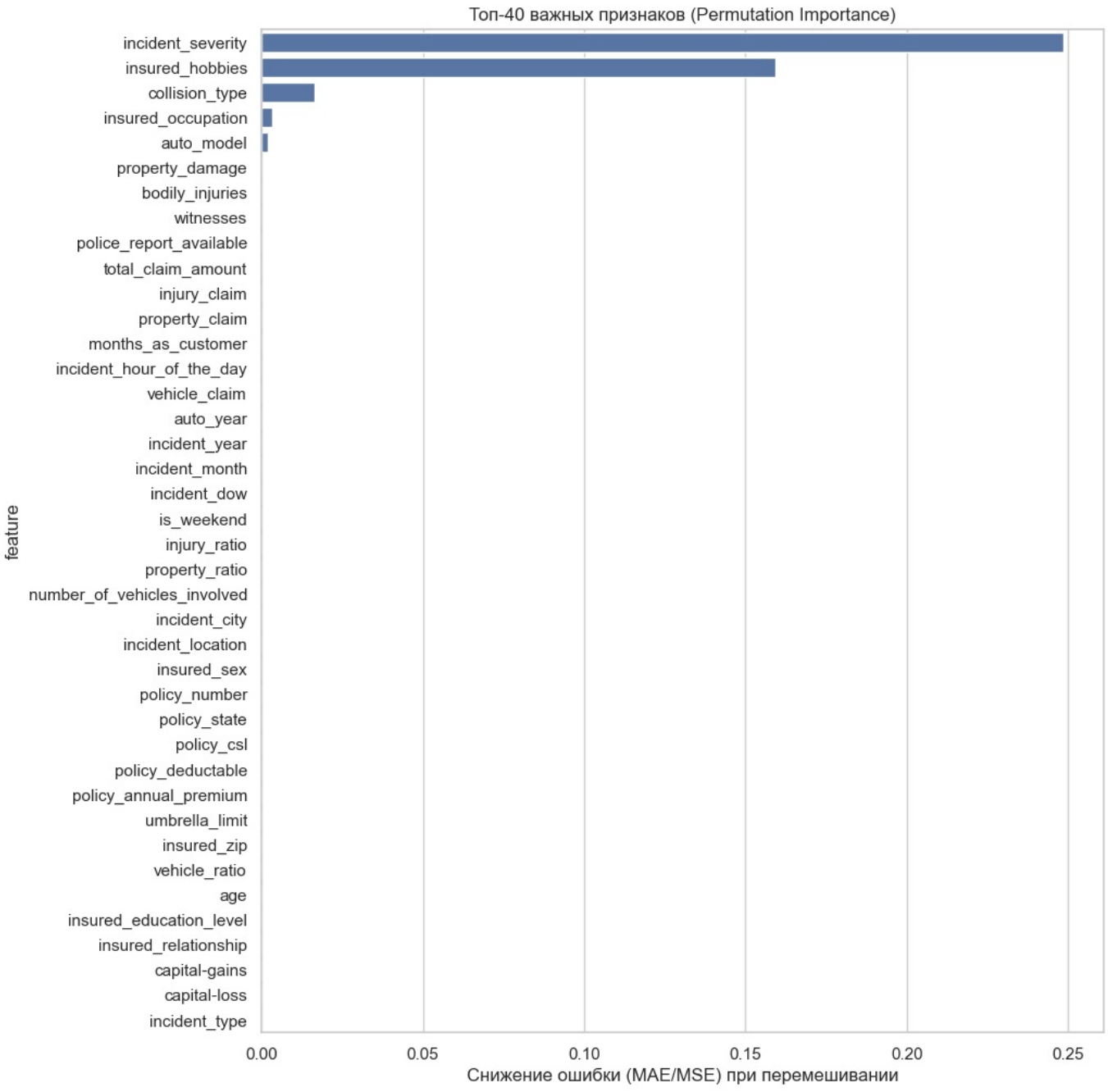
In [42]:
```
model_pipe = Pipeline([
    ('ct', ct),
    ('model', MyDecisionTreeClassifier())
])

param_grid = {
    'model__max_depth': [5, 8, 10, 12, 15, None],
    'model__min_samples_leaf': [2, 4, 8, 10],
    'model__criterion': ['gini', 'entropy']
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(
    model_pipe,
    param_grid,
    cv=cv,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print(f"Лучшие параметры: {grid.best_params_}")

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)
```

```
results = {'F1-score': f1, 'ROC-AUC': roc}

print("-" * 30)
print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
print("="*50)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Лучшие параметры: {'model__criterion': 'gini', 'model__max_depth': 5, 'model__min_samples_leaf': 10}
------------------------------
F1-score (класс 1): 0.6179
ROC-AUC: 0.8781
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.88      0.88       188
           1       0.62      0.61      0.62        62

    accuracy                           0.81       250
   macro avg       0.75      0.75      0.75       250
weighted avg       0.81      0.81      0.81       250

==================================================
```

In [43]:
```
graphics(y_test_i, y_pred, y_prob)
feature_important(grid, X_test_i, y_test_i)
```




Топ-40 важных признаков (Permutation Importance)
incident_severity
insured_hobbies
collision_type
insured_occupation
auto_model
property_damage
bodily_injuries
witnesses
police_report_available
total_claim_amount
injury_claim
property_claim
months_as_customer
incident_hour_of_the_day
vehicle_claim
auto_year
incident_year
incident_month
incident_dow
is_weekend
injury_ratio
property_ratio
number_of_vehicles_involved
incident_city
incident_location
insured_sex
policy_number
policy_state
policy_csl
policy_deductable
policy_annual_premium
umbrella_limit
insured_zip
vehicle_ratio
age
insured_education_level
insured_relationship
capital-gains
capital-loss
incident_type
feature
0.00
0.05
0.10
0.15
0.20
0.25
Снижение ошибки (MAE/MSE) при перемешивании

Моя модель получила даже лучший скор, чем имплементированная в scikit-learn, поскольку чаще угадывает честных. Но в реальной жизни эта модель была бы хуже, по выясненным ранее причинам. Вероятно это связано с отсутствием прунинга

	Base Decision Tree	Decision Tree	My Decision Tree
ROC_AUC	0.7447	0.8611	0.8781
F1 (1 класс)	0.86	0.89	0.88
F1 (2 класс)	0.61	0.73	0.62

Вывод

Решающее дерево хорошо тем, что мы можем легко понять, каким образом модель принимает решение. Кроме того, не нужно запариваться с нормализацией данных. При этом результаты, которые мы получаем на тестовой выборке очень хорошие. Единственная проблема заключается в том, что имплементация довольно сложная под капотом.

Регрессия

Задача: предсказать зарплату игрока НХЛ по его статистике с использованием модели случайного леса.

Будем использовать MAE, так как она очень наглядная, будет легко понять, насколько долларов ошибается модель. В качестве дополнительной метрики будем использовать R^2, чтобы смотреть насколько отличается предсказание от среднего.

Baseline Random Forest

In [14]:
```
import kagglehub
import pandas as pd


path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")

df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)
df.head()
```

Out[14]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	1
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2

5 rows × 154 columns

In [15]:
```
df['Born'] = pd.to_datetime(df['Born'], format='%y-%m-%d')
df_clean = df.fillna(0)
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean = df_clean.drop("Born", axis=1)
```

In [16]:
```
from sklearn.model_selection import train_test_split

TARGET_NAME = "Salary"
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

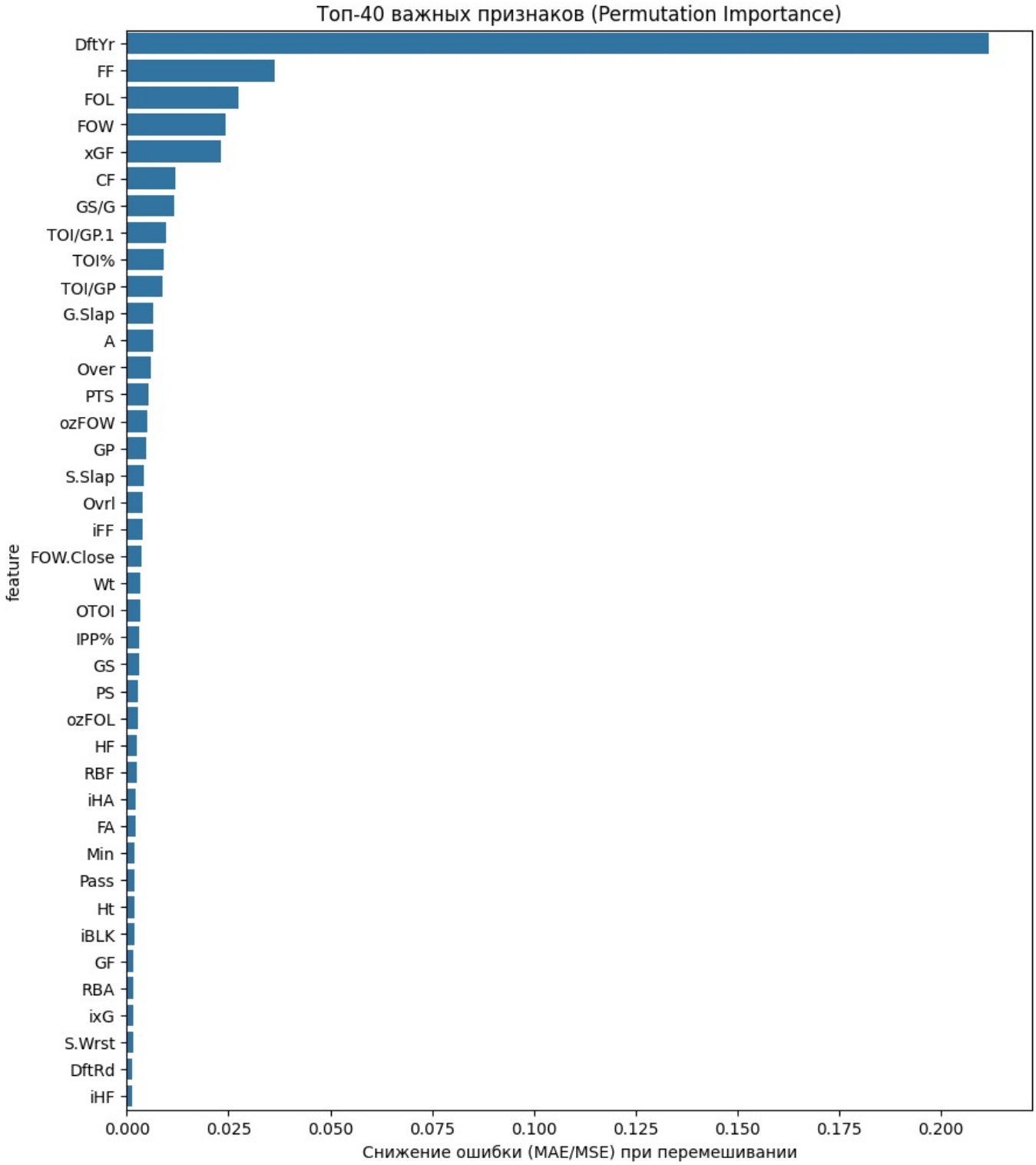
In [17]:
```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

dt = RandomForestRegressor(random_state=42, max_depth=None, min_samples_leaf=1)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, np.full(len(y_test), y_train.mean()))
```

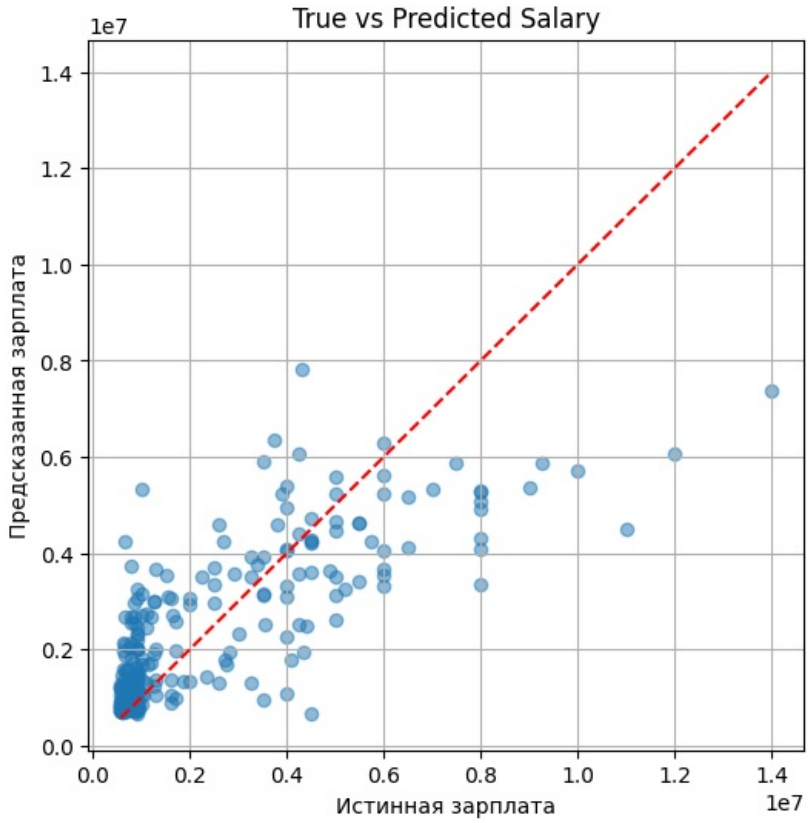
```
print("--- Decision Forest Regressor ---")
print(f"MAE (в тех же единицах, что и y): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

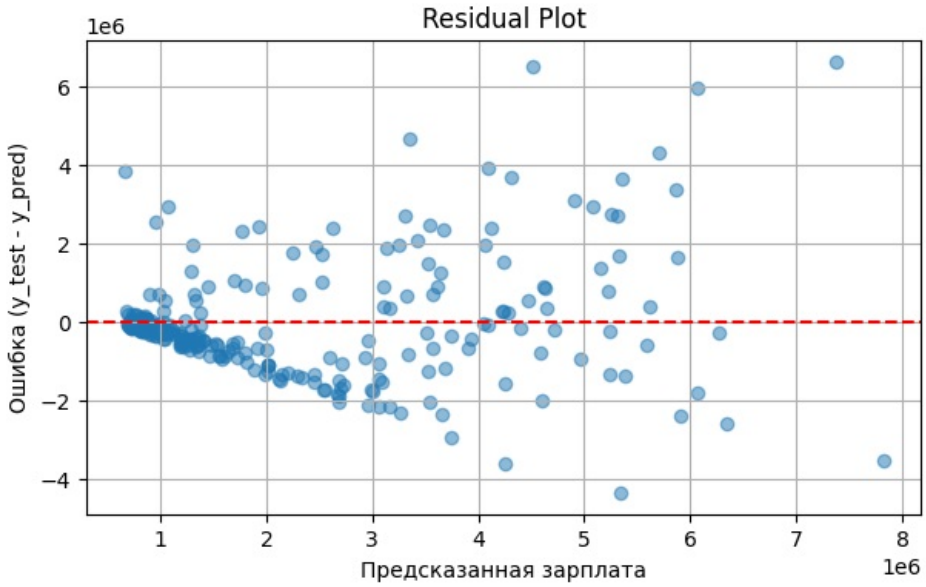
```
--- Decision Forest Regressor ---
MAE (в тех же единицах, что и y): 973296.880
R2 Score: 0.604
```

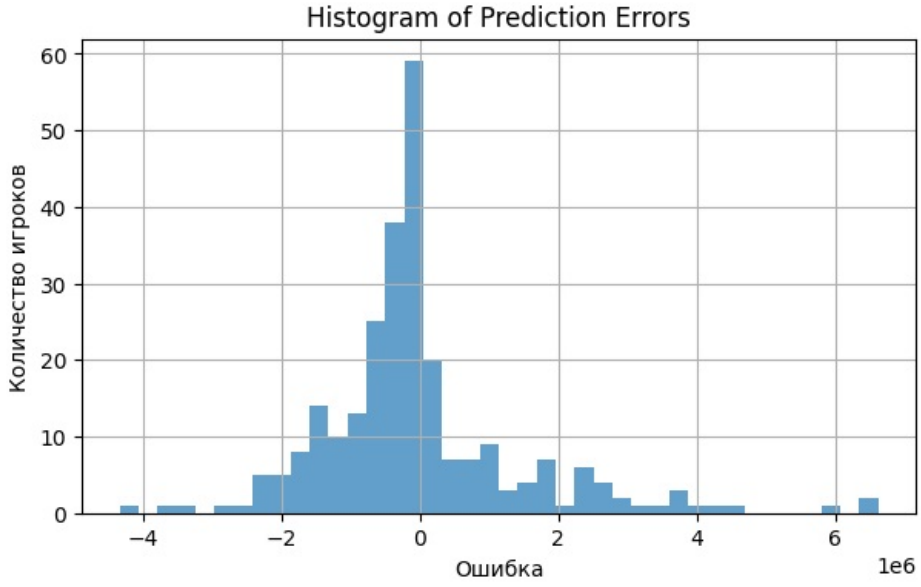
In [18]:

```
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.inspection import permutation_importance

def important_features(model, X_test, y_test):
    perm_result = permutation_importance(
        model,
        X_test,
        y_test,
        n_repeats=10,
        random_state=52,
        n_jobs=-1
    )

    perm_df = pd.DataFrame({
        'feature': X_test.columns,
        'importance': perm_result.importances_mean
    })

    top_40_perm = perm_df.sort_values(by='importance', ascending=False).head(40)

    plt.figure(figsize=(10, 12))
    sns.barplot(data=top_40_perm, x='importance', y='feature')
    plt.title("Топ-40 важных признаков (Permutation Importance)")
    plt.xlabel("Снижение ошибки (MAE/MSE) при перемешивании")
    plt.show()

def true_vs_predicted(y_test, y_pred):
    plt.figure(figsize=(6,6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
    plt.xlabel("Истинная зарплата")
    plt.ylabel("Предсказанная зарплата")
    plt.title("True vs Predicted Salary")
    plt.grid(True)
    plt.show()

def resudial(y_test, y_pred):
    residuals = y_test - y_pred
    plt.figure(figsize=(7,4))
    plt.scatter(y_pred, residuals, alpha=0.5)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel("Предсказанная зарплата")
    plt.ylabel("Ошибка (y_test - y_pred)")
    plt.title("Residual Plot")
    plt.grid(True)
    plt.show()
def histogramm_error(y_test, y_pred):
    plt.figure(figsize=(7,4))
    residuals = y_test - y_pred
    plt.hist(residuals, bins=40, alpha=0.7)
    plt.title("Histogram of Prediction Errors")
    plt.xlabel("Ошибка")
    plt.ylabel("Количество игроков")
    plt.grid(True)
    plt.show()


important_features(dt, X_test, y_test)
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```


Топ-40 важных признаков (Permutation Importance)
feature
DftYr
FF
FOL
FOW
xGF
CF
GS/G
TOI/GP.1
TOI%
TOI/GP
G.Slap
A
Over
PTS
ozFOW
GP
S.Slap
Ovrl
iFF
FOW.Close
Wt
OTOI
IPP%
GS
PS
ozFOL
HF
RBF
iHA
FA
Min
Pass
Ht
iBLK
GF
RBA
ixG
S.Wrst
DftRd
iHF
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Снижение ошибки (MAE/MSE) при перемешивании



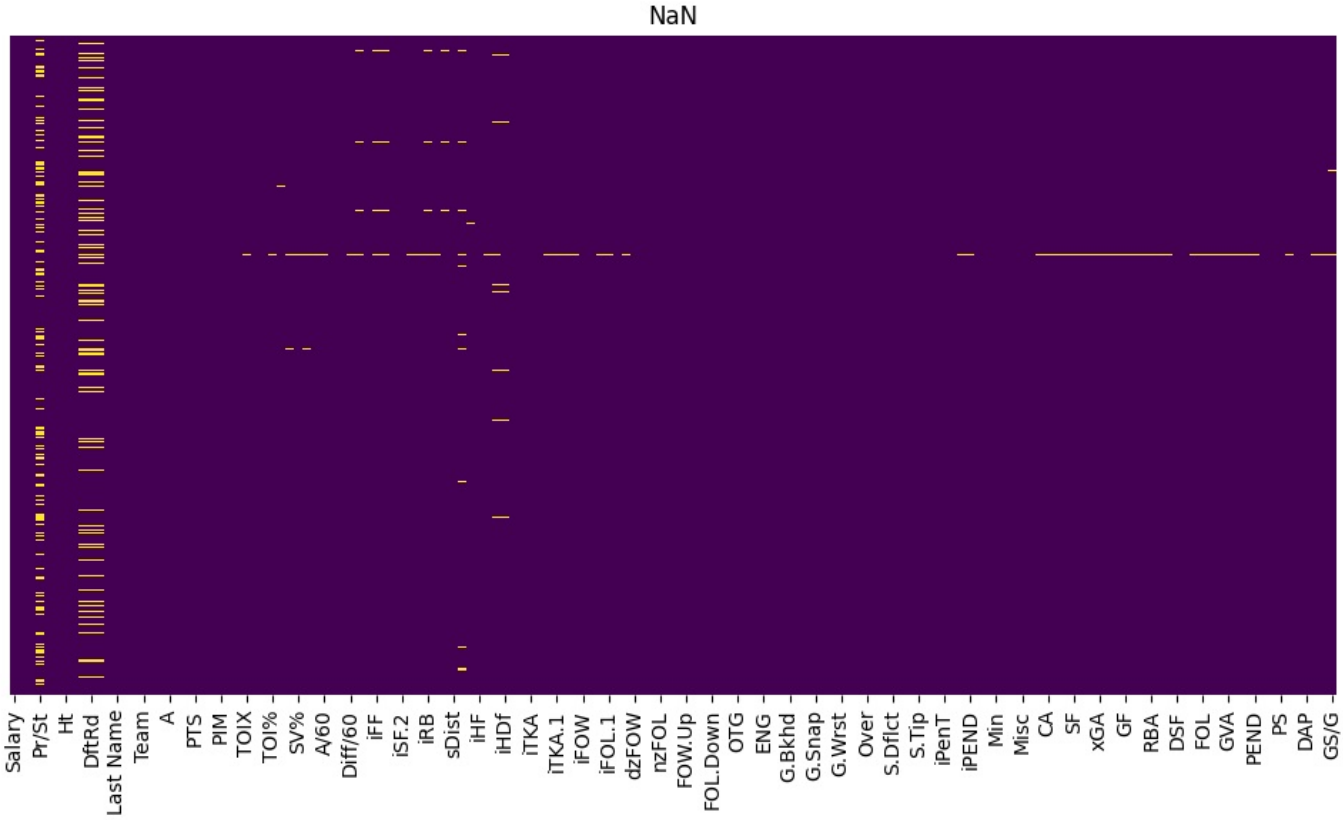




Случайный лес более адаптивная и продвинутая модель по сравнении с решающим деревом. Даже при необработанных данных она даёт неплохой результат. Видим, что точки более сконцентрированы рядом с осью, однако большие зарплаты лесу тяжело предугадать: если не было в обучающей выборке, значит она просто не может дать подобный результат.

Improved Random Forest

In [19]:
```
import kagglehub
from matplotlib import pyplot as plt
import seaborn as sns

path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")
df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [20]:
```
num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
df[num_cols].describe().T
```

Out[20]:

	count	mean	std	min	25%	50%	75%	max
Salary	874.0	2.325289e+06	2.298253e+06	575000.00	742500.00	925000.00	3700000.00	14000000.00
Ht	874.0	7.308238e+01	2.105485e+00	66.00	72.00	73.00	75.00	81.00
Wt	874.0	2.008432e+02	1.506008e+01	157.00	190.00	200.00	210.00	265.00
DftYr	749.0	2.008708e+03	4.380158e+00	1990.00	2006.00	2010.00	2012.00	2016.00
DftRd	749.0	2.742323e+00	1.988358e+00	1.00	1.00	2.00	4.00	9.00
...	...	...	...	...	...	...	...	...
Grit	874.0	1.267815e+02	1.016121e+02	0.00	41.00	114.00	190.00	622.00
DAP	874.0	9.215675e+00	7.815029e+00	0.00	4.60	7.60	12.00	61.00
Pace	873.0	1.089439e+02	8.899877e+00	75.00	104.70	109.20	113.90	175.70
GS	873.0	2.187331e+01	2.198638e+01	-4.30	2.60	15.70	35.40	104.70
GS/G	872.0	3.401606e-01	2.925900e-01	-0.81	0.14	0.31	0.53	1.28

144 rows × 8 columns

In [21]:

```
import pandas as pd
import numpy as np

num_cols = df.select_dtypes(include=["int64", "float64"]).columns

corr_matrix = df[num_cols].corr()

corr_pairs = corr_matrix.unstack().reset_index()
corr_pairs.columns = ['feature_1', 'feature_2', 'correlation']

corr_pairs = corr_pairs[corr_pairs['feature_1'] < corr_pairs['feature_2']]

corr_pairs = corr_pairs.reindex(
    corr_pairs['correlation'].abs().sort_values(ascending=False).index
)

top40 = corr_pairs.head(40)

display(top40)
upper = np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
upper_corr = corr_matrix.where(upper)
to_drop = [col for col in upper_corr.columns if any(upper_corr[col] > 0.998)]

df_clean = df.drop(columns=to_drop)

df_clean.head()
```

	feature_1	feature_2	correlation
4931	iSF.1	iSF.2	0.999996
8267	iFOL	iFOL.1	0.999981
8122	iFOW	iFOW.1	0.999979
6381	iHF	iHF.1	0.999957
2321	TOI	TOIX	0.999952
7398	iBLK	iBLK.1	0.999948
4787	iSF	iSF.2	0.999937
4786	iSF	iSF.1	0.999931
7108	iGVA	iGVA.1	0.999907
7253	iTKA	iTKA.1	0.999843
2611	TOI/GP	TOI/GP.1	0.999289
15807	CF	FF	0.999216
4351	iCF	iCF.1	0.998819
16097	FF	SF	0.998722
10424	FOW.Close	iFOW	0.998613
10426	FOW.Close	iFOW.1	0.998603
16242	FA	SA	0.998366
10571	FOL.Close	iFOL.1	0.998221
10569	FOL.Close	iFOL	0.998215
15952	CA	FA	0.998114
15809	CF	SF	0.997739
15954	CA	SA	0.995524
4642	iFF	iSF.1	0.995443
4643	iFF	iSF.2	0.995427
4641	iFF	iSF	0.995415
17108	SCA	xGA	0.995015
16963	SCF	xGF	0.994495
1002	GP	OTOI	0.993385
13008	Wide	iMiss	0.993229
16244	FA	xGA	0.992949
15956	CA	xGA	0.992542
10136	FOW.Down	iFOW	0.992496
10138	FOW.Down	iFOW.1	0.992464
16099	FF	xGF	0.991109
2177	Shifts	TOIX	0.991098
16532	SA	xGA	0.991094
4496	iCF.1	iFF	0.990960
2899	TOI%	TOI/GP.1	0.990941
10281	FOL.Down	iFOL	0.990924
2898	TOI%	TOI/GP	0.990914

Out[21]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	1(
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2(

5 rows × 137 columns

In [22]:
```
df_features = df_clean.copy()

df_features['Born'] = pd.to_datetime(df_features['Born'], format='%y-%m-%d')
reference_date = pd.Timestamp('2016-10-01')
df_features['Age'] = (reference_date - df_features['Born']).dt.days / 365.25

df_features['Experience'] = reference_date.year - df_features['DftYr']
df_features['Age_squared'] = df_features['Age'] ** 2
df_features['G_per_GP'] = df_features['G'] / df_features['GP'].replace(0, 1)
df_features['A_per_GP'] = df_features['A'] / df_features['GP'].replace(0, 1)
df_features['PTS_per_GP'] = df_features['PTS'] / df_features['GP'].replace(0, 1)
df_features['Is_Drafted'] = df_features['DftYr'].notna().astype(int)
df_features['Physical_Impact'] = df_features['Wt'] * df_features['Ht']

features_to_drop = ['Born', 'Last Name', 'First Name', 'Nat', 'Pr/St', 'City']
df_features = df_features.drop(features_to_drop, axis=1)
df_features['Match'].value_counts()
```

Out[22]:
```
Match
0    870
1      4
Name: count, dtype: int64
```

In [23]:
```
has_nan = df_features.isnull().any()

columns_with_nan = has_nan[has_nan].index.tolist()

print("Столбцы, содержащие хотя бы один NaN:")
columns_with_nan
```

```
Столбцы, содержащие хотя бы один NaN:
```

Out[23]:
```
['DftYr',
 'DftRd',
 'Ovrl',
 'TOI%',
 'IPP%',
 'SH%',
 'SV%',
 'PDO',
 'F/60',
 'A/60',
 'Diff/60',
 'iCF',
 'iFF',
 'iSF',
 'ixG',
 'iSCF',
 'iRB',
 'iRS',
 'iDS',
 'sDist.1',
 'Pass',
 'iHA',
 'iHDf',
 'BLK%',
 '%FOT',
 'iPENT',
 'iPEND',
 'CF',
 'CA',
 'xGF',
 'xGA',
 'SCF',
 'SCA',
 'GF',
 'GA',
 'RBF',
 'RBA',
 'RSF',
 'RSA',
 'FOW',
 'FOL',
 'HF',
 'HA',
 'GVA',
 'TKA',
 'PENT',
 'PEND',
 'OTOI',
 'Pace',
 'GS',
 'GS/G',
 'Experience']
```

Тут обработка выбросов помогла улучшить score

In [24]:
```
from scipy.stats.mstats import winsorize
TARGET_NAME = 'Salary'

X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]

X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(
    X_i, y_i, test_size=0.2, random_state=42
)

y_train_i = winsorize(y_train_i, limits=[0.01, 0.01])
```

In [25]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

cat_cols = X_i.select_dtypes(include=['object', 'category']).columns.tolist()

cat_branch = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

num_cols = X_i.select_dtypes(include=["int64", "float64"]).columns

num_branch = Pipeline([
```

```
    ('imputer', SimpleImputer(strategy='median')),
])

ct = ColumnTransformer(
    transformers=[
        ("cat_proc", cat_branch, cat_cols),
        ("num_proc", num_branch, num_cols),
    ], remainder='drop'
)
```

In [27]:

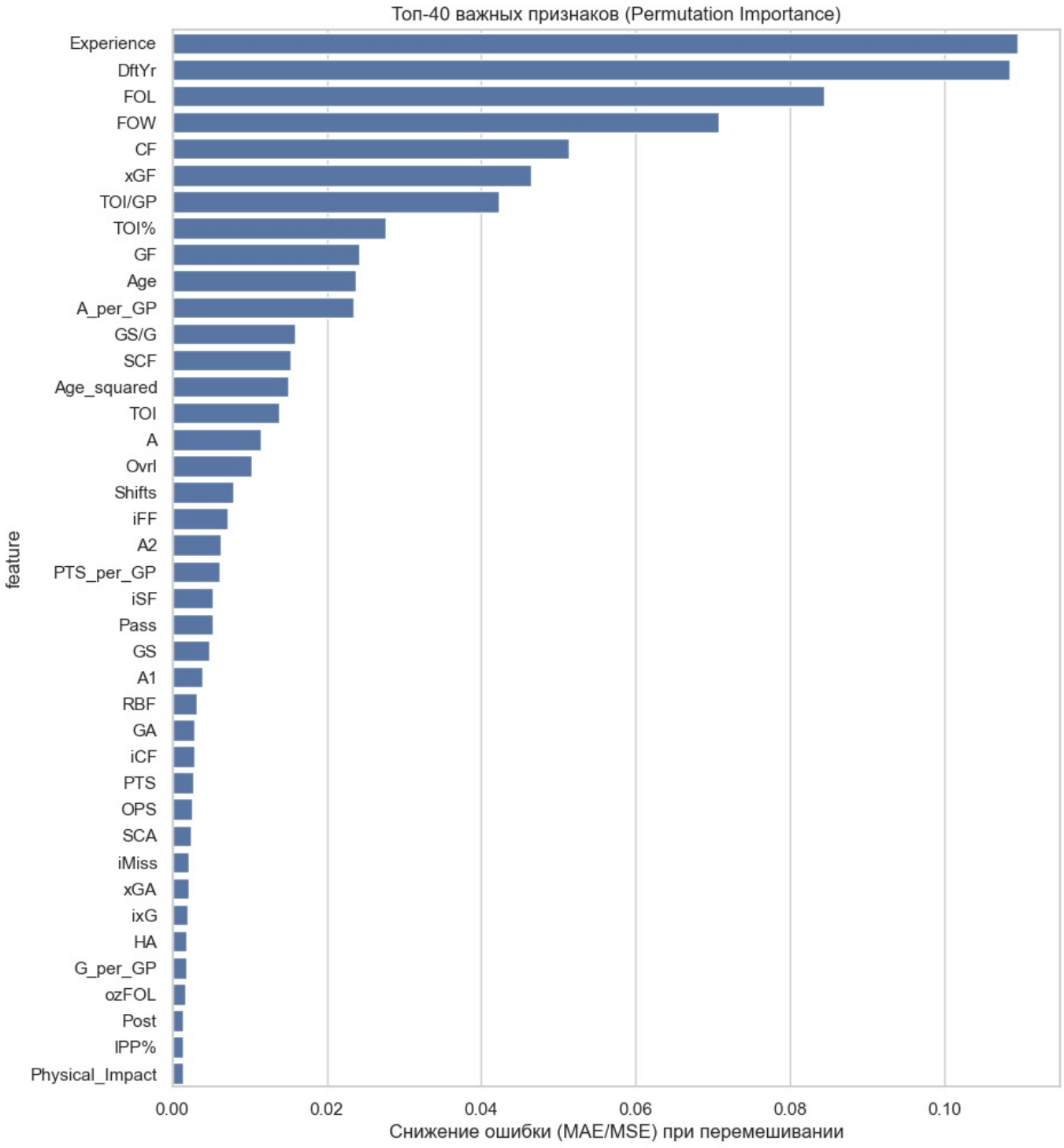
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.compose import TransformedTargetRegressor
import numpy as np

full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=RandomForestRegressor(random_state=52),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'model_wrapper__regressor__n_estimators': [200, 400, 800],

    'model_wrapper__regressor__max_depth': [3, 5, 7],
    'model_wrapper__regressor__min_samples_leaf': [7, 10, 15],
    'model_wrapper__regressor__min_samples_split': [2, 5, 10],

    'model_wrapper__regressor__max_features': ['sqrt', 'log2', 0.5],

    'model_wrapper__regressor__bootstrap': [True, False],
    'model_wrapper__regressor__max_samples': [0.6, 0.8, None],
}

grid = RandomizedSearchCV(
    full_pipeline,
    param_grid,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1,
    n_iter=250
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
y_pred_train = grid.best_estimator_.predict(X_train_i)

mae = mean_absolute_error(y_test_i, y_pred)
train_mae = mean_absolute_error(y_train_i, y_pred_train)
r2 = r2_score(y_test_i, y_pred)

print(f"\n--- Random Forest Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"Train MAE: {train_mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
Fitting 5 folds for each of 250 candidates, totalling 1250 fits
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/model_selection/_validation.py:516: FitFailedWarning:
445 fits failed out of a total of 1250.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
445 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/model_selection/_validation.py", line 859, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/base.py", line 1365, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/pipeline.py", l
```

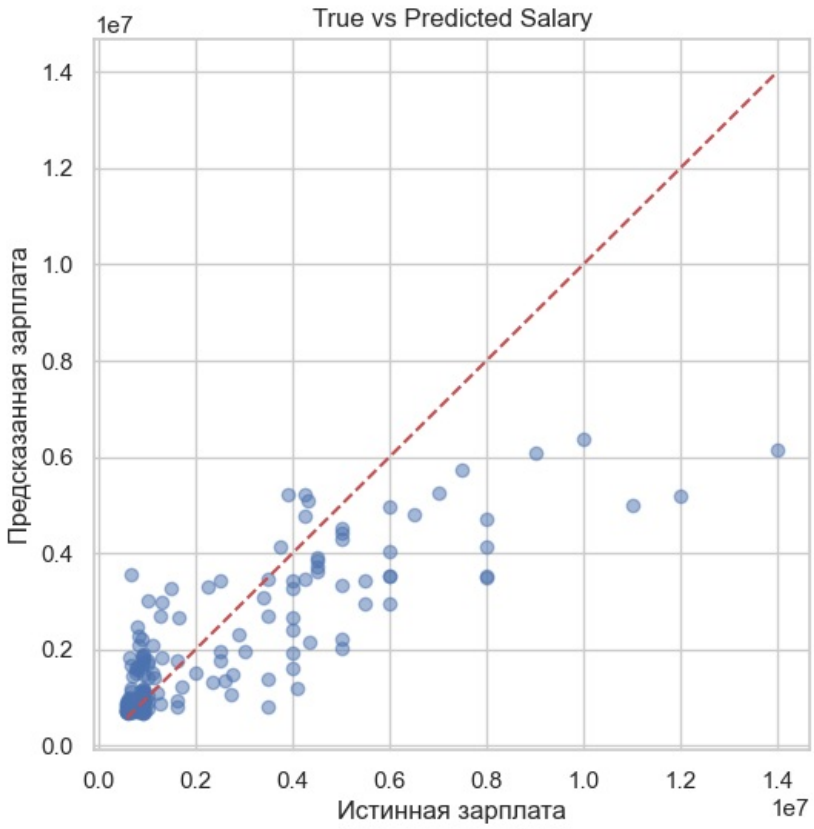
```
ine 663, in fit
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/base.py", line
1365, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/compose/_target
.py", line 293, in fit
    self.regressor_.fit(X, y_trans, **routed_params.regressor.fit)
  File "/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/base.py", line
1365, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/ensemble/_fores
t.py", line 430, in fit
    raise ValueError(
ValueError: `max_sample` cannot be set if `bootstrap=False`. Either switch to `bootstrap=True` or set `max_sampl
e=None`.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/model_selection/_search
.py:1135: UserWarning: One or more of the test scores are non-finite: [ -978653.26883628  -896834.9412678  -1102
220.41131317  -990379.05510381
  -969618.47712101               nan  -970581.80710492               nan
 -1074483.50068273               nan  -970394.17573753  -988863.18992738
  -999517.78004625  -903601.89732986  -914621.81150138  -890492.30085835
  -904606.2472704  -1049049.9313279   -908263.20121119               nan
               nan               nan -1012256.53100781               nan
  -978653.26883628  -994782.20520229  -949110.0944828   -916159.57712467
 -1030762.5546746   -930214.93888234  -954477.76348141               nan
  -954477.76348141  -967677.82323949  -948934.40286485               nan
  -987720.59670486 -1037073.3787189                nan               nan
 -1057183.92782646  -886052.81603346               nan  -901545.28332536
               nan -1033904.58740981  -988777.81058285  -969618.47712101
               nan               nan -1049049.9313279   -969947.19850948
  -904606.2472704   -982508.38747475               nan -1047300.90973719
 -1031952.12489304 -1057183.92782646               nan               nan
 -1029186.70093456  -950694.88379609 -1064062.88466879 -1011760.18541459
               nan               nan               nan  -901870.04780652
 -1006623.92911111               nan               nan  -901413.2326126
 -1007607.6332976   -914154.24126284 -1049482.44890482               nan
 -1101491.80153289               nan  -948610.80705984               nan
 -1047182.22436865 -1029946.40173386 -1102220.41131317               nan
 -1043081.22024761  -970535.33074236 -1041512.57504627 -1110064.51068785
               nan  -897131.17402029               nan               nan
               nan  -949838.24725154  -948817.93178854  -982508.38747475
               nan  -914636.96859582  -978653.26883628 -1110366.19938626
               nan               nan -1010593.47005659 -1104194.17975577
 -1110064.51068785 -1057183.92782646 -1100452.58375474               nan
               nan -1024373.96807625 -1047477.66701009  -997953.45248372
 -1014621.82738794  -974170.38046517               nan               nan
 -1076220.15188958  -982364.98558557 -1024605.08776405 -1049049.9313279
               nan               nan               nan -1033904.58740981
  -970535.33074236               nan               nan  -988777.81058285
 -1102252.63225624               nan -1050164.52839106               nan
 -1043644.81443302               nan  -929586.1611688  -1050836.33545259
  -993359.76755492               nan               nan -1033904.58740981
               nan -1012380.0708728  -1047139.07215046 -1098446.92965924
  -910478.28393565               nan  -987720.59670486  -886052.81603346
               nan -1100262.74313692  -905344.42830387  -970581.80710492
               nan               nan               nan               nan
  -903601.89732986 -1103327.8603698                nan  -927725.08749172
               nan               nan -1055453.133724    -896481.00926566
               nan  -950698.81061638  -983814.98129851               nan
 -1043047.41440949 -1048930.75346809               nan  -905318.99451392
  -954645.18732966               nan               nan -1103194.06316652
  -994737.88315341               nan  -990379.05510381  -991020.43505979
  -908362.86370822  -904856.04189909 -1042213.95876579               nan
               nan -1108207.78752047 -1054619.56938937               nan
               nan -1045988.22400674 -1003636.30434662               nan
               nan -1031952.12489304 -1064062.88466879  -948260.83764094
               nan               nan               nan -1106881.04399056
               nan  -912896.01737393               nan  -950698.81061638
 -1102252.63225624  -988513.68813444  -982465.77751934  -920053.12965377
  -911621.35042405  -909579.15903522 -1104194.17975577               nan
 -1032719.66678798 -1050779.39568996  -972548.00319566               nan
               nan  -947519.34288686 -1103990.87500506  -903075.85485267
               nan  -887249.00246169               nan               nan
 -1036749.43587703  -986727.35323473               nan -1034884.31603017
 -1053765.06245545  -970601.61045639               nan               nan
               nan  -970535.33074236  -983814.98129851 -1055025.22238954
  -986137.31530143               nan  -901413.2326126                nan
               nan -1044300.18331242               nan  -952257.08394253
```

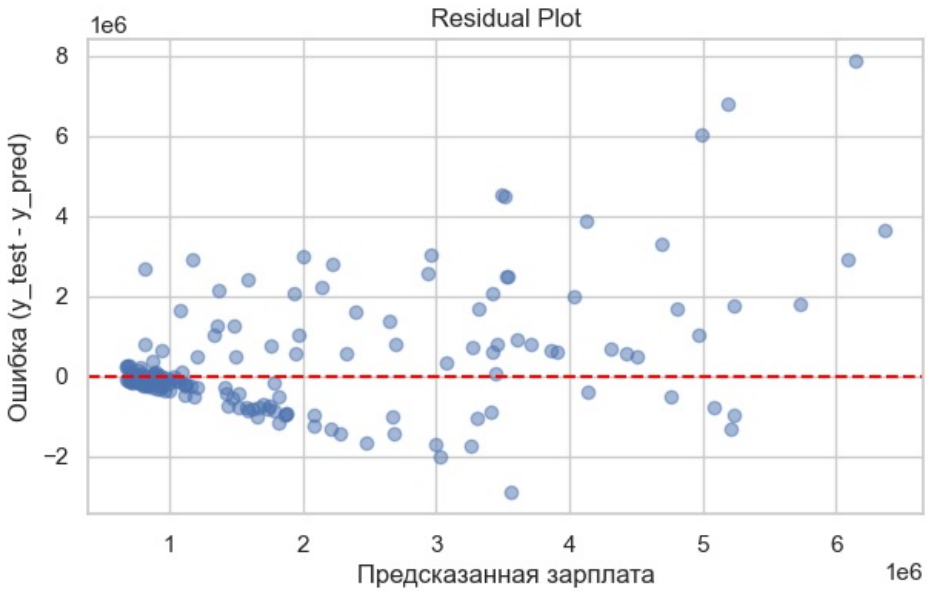
```
  -917838.52419749 -1103660.02271622 -1055902.66735392  -916159.57712467
 -1050836.33545259 -1055555.78468428]
  warnings.warn(
Лучшие параметры: {'model_wrapper__regressor__n_estimators': 200, 'model_wrapper__regressor__min_samples_split':
5, 'model_wrapper__regressor__min_samples_leaf': 10, 'model_wrapper__regressor__max_samples': None, 'model_wrapp
er__regressor__max_features': 0.5, 'model_wrapper__regressor__max_depth': 7, 'model_wrapper__regressor__bootstra
p': False}

--- Random Forest Regressor ---
MAE: 857029.069
Train MAE: 511353.482
R2 Score: 0.638
```

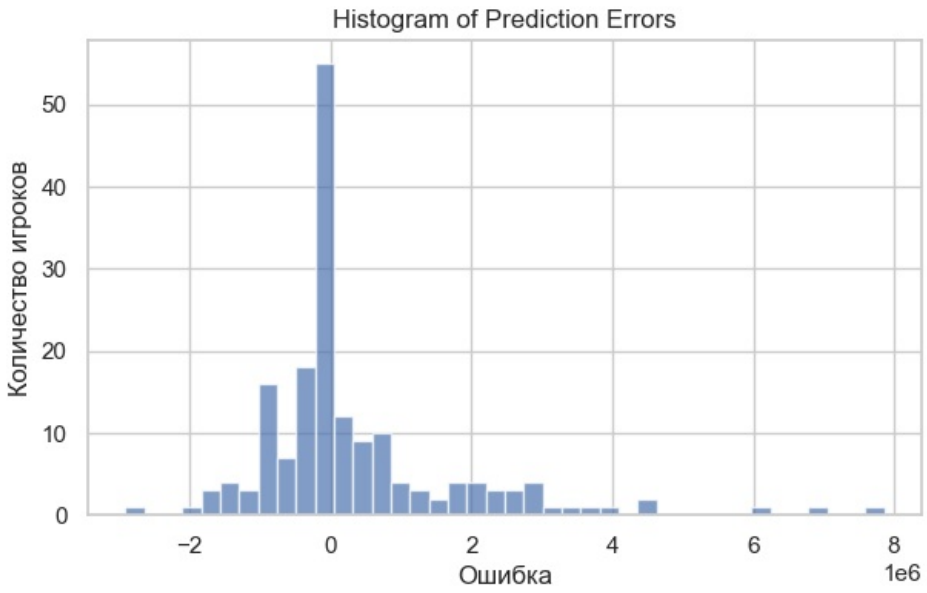
In [ ]:
```
important_features(grid.best_estimator_, X_test_i, y_test_i)
```



In [ ]:
```
true_vs_predicted(y_test_i, y_pred)
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```







Решающее дерево оказывается немного хуже обычной линейной регрессии. Деревьям и лесу тяжело видимо работать с такими данными. Данных немного, поэтому в тестовой выборке он может не увидеть некоторых игроков с большими зарплатами, как

следствие, лесу становиться плохо при работе с очень большими зарплатами, он склонен их занижать, в то время как модели линейной регрессии могут лучше обрабатывать такие выбросы.

Стоит отметить, что при меньших зарплатах, лесные модели показывают лучший скор, что видно по графику выше, предсказания явно более кучные относительно главной диагонали.

In [ ]:
```
import pandas as pd
import numpy as np

original_float_format = pd.get_option('display.float_format')

pd.set_option('display.float_format', '{:.2f}'.format)

best_model = grid.best_estimator_

y_full_pred = best_model.predict(X_i)

results_df = pd.DataFrame({
    'True_Salary': y_i,
    'Predicted_Salary': y_full_pred
})

results_df['Absolute_Error'] = np.abs(results_df['True_Salary'] - results_df['Predicted_Salary'])

test_indices = X_test.index
final_results = pd.merge(
    df[['First Name', 'Last Name']],
    results_df,
    left_index=True,
    right_index=True
)

final_results_sorted = final_results.sort_values(by='Absolute_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---")
print(final_results_sorted.head(10).to_string())

pd.set_option('display.float_format', original_float_format)
```

```
--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---
    First Name Last Name  True_Salary  Predicted_Salary  Absolute_Error
626       Anze   Kopitar  14000000.00        6142609.56      7857390.44
817     Steven   Stamkos   9500000.00        1962838.71      7537161.29
103    Patrick      Kane  13800000.00        6557234.23      7242765.77
496   Jonathan     Toews  13800000.00        6775986.57      7024013.43
542       Shea     Weber  12000000.00        5191312.11      6808687.89
208       P.K.    Subban  11000000.00        4986752.04      6013247.96
837       Ryan  O'Reilly  11000000.00        5318298.35      5681701.65
564       Ryan  Callahan   6500000.00        1975552.41      4524447.59
868       Loui  Eriksson   8000000.00        3495885.94      4504114.06
497     Andrew      Ladd   8000000.00        3518267.35      4481732.65
```

In [ ]:
```
import pandas as pd
final_results['Signed_Error'] = final_results['Predicted_Salary'] - final_results['True_Salary']

overestimated_players = final_results[final_results['Signed_Error'] > 0].copy()

top_overestimated = overestimated_players.sort_values(by='Signed_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---")

print(top_overestimated[['First Name', 'Last Name',
                         'True_Salary', 'Predicted_Salary', 'Signed_Error']].head(10).to_string())
```

```
--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---
    First Name   Last Name  True_Salary  Predicted_Salary  Signed_Error
849        Sam      Gagner     650000.0      3.559721e+06  2.909721e+06
532      Radim      Vrbata    1000000.0      3.713533e+06  2.713533e+06
830    Patrick       Eaves    1000000.0      3.522240e+06  2.522240e+06
665     Justin     Schultz    1400000.0      3.621041e+06  2.221041e+06
110       Luke      Schenn    1000000.0      3.025774e+06  2.025774e+06
150       Eric       Staal    3500000.0      5.524733e+06  2.024733e+06
5        Brian    Campbell    1500000.0      3.260718e+06  1.760718e+06
659     Artemi     Panarin     925000.0      2.678042e+06  1.753042e+06
687     Dennis  Seidenberg    1000000.0      2.747500e+06  1.747500e+06
44        Paul       Byron    1300000.0      2.994608e+06  1.694608e+06
```

My implementation

In [ ]:
```
import numpy as np
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
from sklearn.utils import check_random_state

class MyRandomForestRegressor(BaseEstimator, RegressorMixin):
    def __init__(
        self,
        n_estimators=100,
        max_depth=None,
        min_samples_leaf=1,
        max_features="sqrt",
        bootstrap=True,
        random_state=None
    ):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
        self.max_features = max_features
        self.bootstrap = bootstrap
        self.random_state = random_state
        self.trees_ = []
        self.feature_subsets_ = []

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        n_samples, n_features = X.shape

        rng = check_random_state(self.random_state)

        if self.max_features == "sqrt":
            n_sub_features = int(np.sqrt(n_features))
        elif self.max_features == "log2":
            n_sub_features = int(np.log2(n_features))
        elif isinstance(self.max_features, int):
            n_sub_features = self.max_features
        elif self.max_features is None:
            n_sub_features = n_features
        else:
            raise ValueError("Invalid max_features")

        for _ in range(self.n_estimators):
            if self.bootstrap:
                sample_idx = rng.choice(n_samples, n_samples, replace=True)
            else:
                sample_idx = np.arange(n_samples)

            X_sample = X[sample_idx]
            y_sample = y[sample_idx]

            feature_idx = rng.choice(n_features, n_sub_features, replace=False)
            self.feature_subsets_.append(feature_idx)

            X_sample_small = X_sample[:, feature_idx]

            tree = DecisionTreeRegressor(
                max_depth=self.max_depth,
                min_samples_leaf=self.min_samples_leaf
            )
            tree.fit(X_sample_small, y_sample)

            self.trees_.append(tree)

        return self

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)

        predictions = []

        for tree, feat_idx in zip(self.trees_, self.feature_subsets_):
            X_small = X[:, feat_idx]
            predictions.append(tree.predict(X_small))

        return np.mean(predictions, axis=0)
```

In [ ]:

```
dt = MyRandomForestRegressor(random_state=42, max_depth=None, min_samples_leaf=1)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
```

```
base_mean_error = mean_absolute_error(y_test, np.full(len(y_test), y_train.mean()))

print("--- Decision Forest Regressor ---")
print(f"MAE (в тех же единицах, что и y): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
--- Decision Forest Regressor ---
MAE (в тех же единицах, что и y): 1128442.685
R2 Score: 0.478
```

In [ ]:

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.compose import TransformedTargetRegressor
import numpy as np

full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=MyRandomForestRegressor(),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'model_wrapper__regressor__max_depth': [3, 5],
    'model_wrapper__regressor__min_samples_leaf': [10, 15],
}

grid = GridSearchCV(
    full_pipeline,
    param_grid,
    cv=3,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
y_pred_train = grid.best_estimator_.predict(X_train_i)

mae = mean_absolute_error(y_test_i, y_pred)
train_mae = mean_absolute_error(y_train_i, y_pred_train)
r2 = r2_score(y_test_i, y_pred)

print(f"\n--- Decision Tree Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"Train MAE: {train_mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
Лучшие параметры: {'model_wrapper__regressor__max_depth': 5, 'model_wrapper__regressor__min_samples_leaf': 15}

--- Decision Tree Regressor ---
MAE: 1108498.463
Train MAE: 911186.010
R2 Score: 0.424
```

Мой лес для регрессии получился сильно хуже, чем у scikit-learn. Это базовая имплементация, которая просто берёт среднее по всем деревьям, а библиотечная реализация использует дополнительно weighted averaging, out-of-bag estimation, subspace sampling, fast impurity reduction. И прунинга нет, конечно.

	Base Random Forest	Random Forest	My Random Forest
MAE	973296.880	857029.069	1134829.680
R2	0.604	0.638	0.405

Классификация

Задача: вычислить мошенника на страховых выплатах с использованием модели случайного леса

Для выполнения лабораторной работы были выбраны метрики F1-score и ROC-AUC, так как исследуемый датасет является несбалансированным. Метрика Accuracy в данном случае неинформативна, так как модель, предсказывающая всем класс ‘0’ (не фрод), может иметь высокую Accuracy, но будет бесполезна. F1-score позволит контролировать баланс между ложными срабатываниями и пропуском мошенников.

Baseline

In [ ]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                "buntyshah/auto-insurance-claims-data/versions/1",
                                "insurance_claims.csv")
df
```

Out[ ]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 40 columns

In [ ]:
```
df_clean = df.copy()

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
df_clean["property_damage"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
```

In [ ]:
```
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean.head()
```

Out[ ]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium
0	328	48	521585	940	2	1	1000	1406.91
1	228	42	342868	635	1	1	2000	1197.22
2	134	29	687698	413	2	0	2000	1413.14
3	256	41	227811	19	0	1	2000	1415.74
4	228	44	367455	922	0	2	1000	1583.91

5 rows × 40 columns

In [ ]:
```
from sklearn.model_selection import train_test_split
drop_dates = ["policy_bind_date", "incident_date"]
df_clean = df_clean.drop(drop_dates, axis=1)
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

In [ ]:
```
from sklearn.metrics import classification_report, f1_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
```

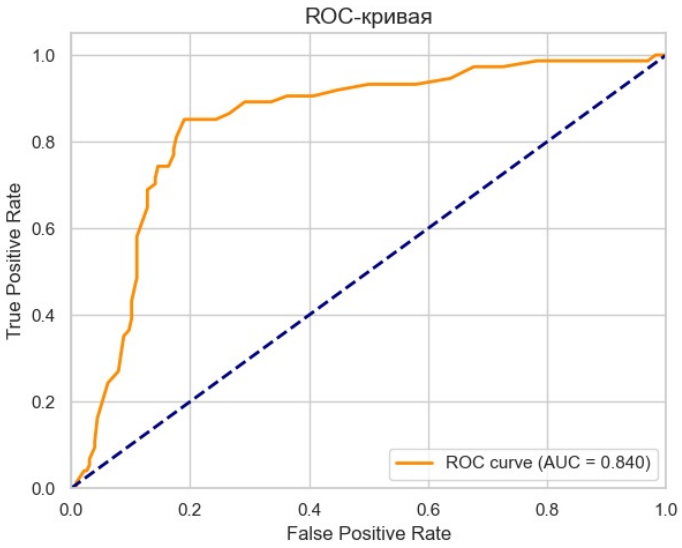
```
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

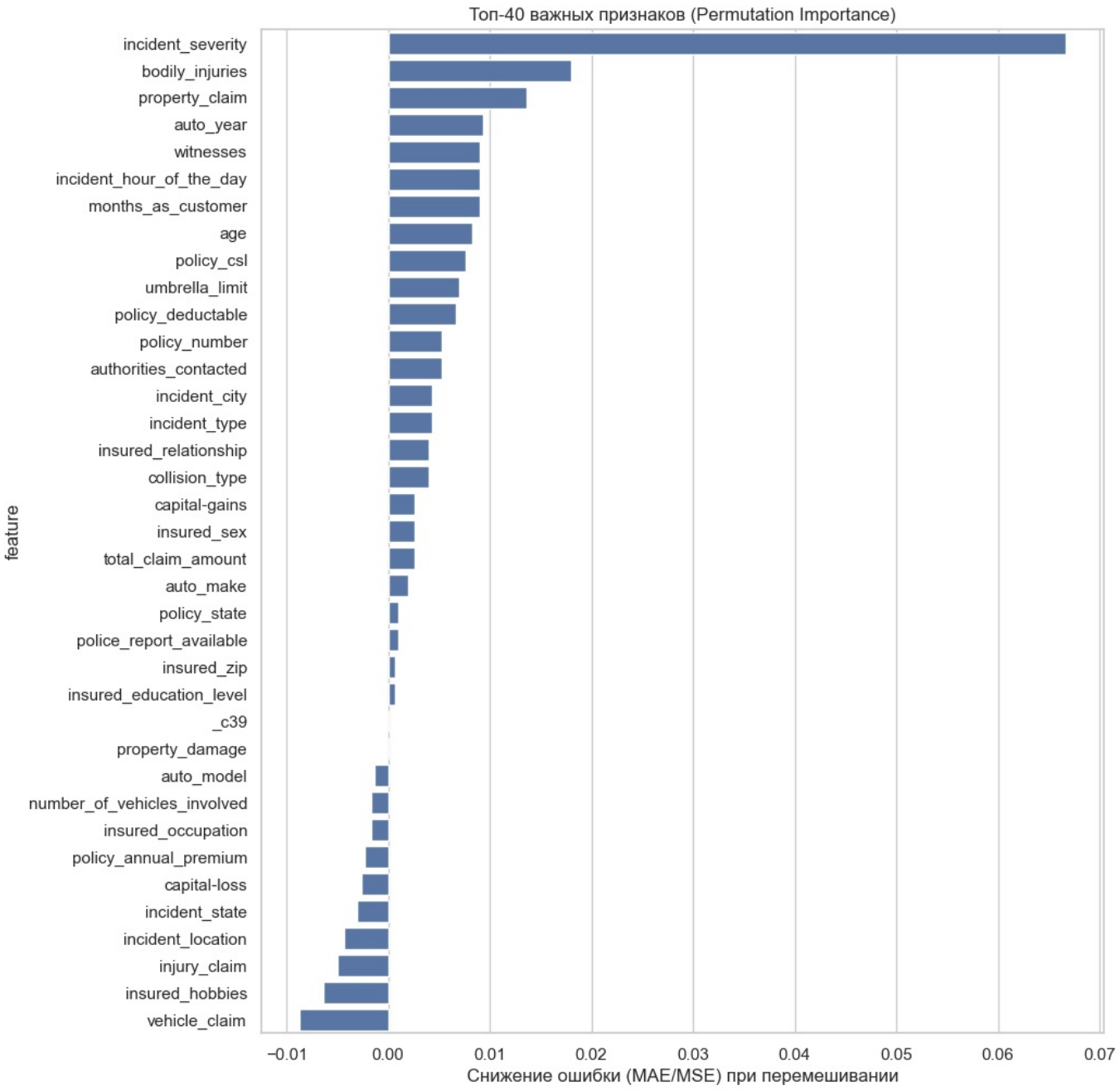
```
F1-score (класс 1): 0.4333
ROC-AUC: 0.8404
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86       226
           1       0.57      0.35      0.43        74

    accuracy                           0.77       300
   macro avg       0.69      0.63      0.65       300
weighted avg       0.75      0.77      0.75       300

==================================================

ROC-AUC: 0.8404
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86       226
           1       0.57      0.35      0.43        74

    accuracy                           0.77       300
   macro avg       0.69      0.63      0.65       300
weighted avg       0.75      0.77      0.75       300

==================================================
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc
def graphics(y_test, y_pred, y_prob):
    sns.set(style="whitegrid")
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", cbar=False,
                xticklabels=['Предск: 0', 'Предск: 1'],
                yticklabels=['Факт: 0', 'Факт: 1'])
    plt.title("Матрица ошибок", fontsize=14)
    plt.ylabel("Реальность")
    plt.xlabel("Предсказание")

    plt.subplot(1, 2, 2)
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC-кривая', fontsize=14)
    plt.legend(loc="lower right")

    plt.tight_layout()
    plt.show()
def feature_important(model, X_test, y_test):

    perm_result = permutation_importance(
        model,
        X_test,
        y_test,
        n_repeats=10,
        random_state=42,
        n_jobs=-1
    )

    perm_df = pd.DataFrame({
        'feature': X_test.columns,
        'importance': perm_result.importances_mean
    })

    top_40_perm = perm_df.sort_values(by='importance', ascending=False).head(40)
```

```
    plt.figure(figsize=(10, 12))
    sns.barplot(data=top_40_perm, x='importance', y='feature')
    plt.title("Топ-40 важных признаков (Permutation Importance)")
    plt.xlabel("Снижение ошибки (MAE/MSE) при перемешивании")
    plt.show()
graphics(y_test, y_pred, y_prob)
feature_important(model, X_test, y_test)
```

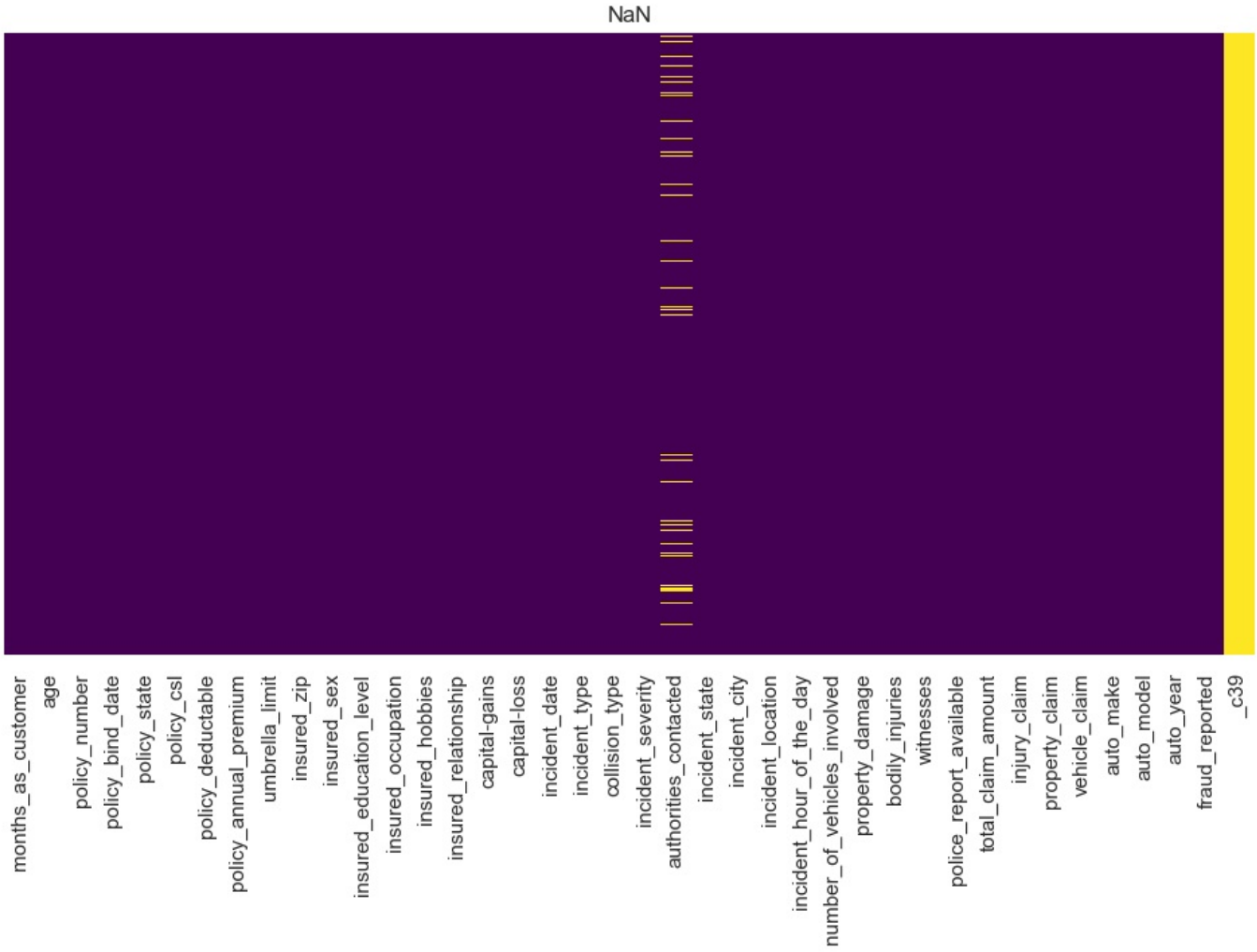




Случайный лес показывает неплохие метрики, однако он склонен приписывать всем честность. Нам такое не нравится, модель должна лучше определять недоминирующий класс.

Improved Decision Tree

In [ ]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                    "buntyshah/auto-insurance-claims-data/versions/1",
                                    "insurance_claims.csv")
df.head()

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [ ]:
```
df_clean = df.copy()
df_clean["fraud_reported"].value_counts()
```

Out[ ]:
```
fraud_reported
N    753
Y    247
Name: count, dtype: int64
```

In [ ]:
```
display(df_clean["police_report_available"].unique())
display(df_clean["property_damage"].unique())
```

```
array(['YES', '?', 'NO'], dtype=object)
array(['YES', '?', 'NO'], dtype=object)
```

In [ ]:
```
df_clean = df_clean.drop(columns=["_c39"])

df_clean["authorities_contacted"] = df_clean["authorities_contacted"].fillna("No Contact")

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0, '?': -1})
df_clean["property_damage"] = df_clean["property_damage"].map({'YES': 1, 'NO': 0, '?': -1})

dates_cols = ["policy_bind_date", "incident_date"]
```

```
for c in dates_cols:
    df_clean[c] = pd.to_datetime(df_clean[c])

df_clean
```

Out[ ]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 39 columns

In [ ]:
```
num_cols = df_clean.select_dtypes(include=["int64", "float64"]).columns.tolist()
df_clean[num_cols].describe().T
```

Out[ ]:

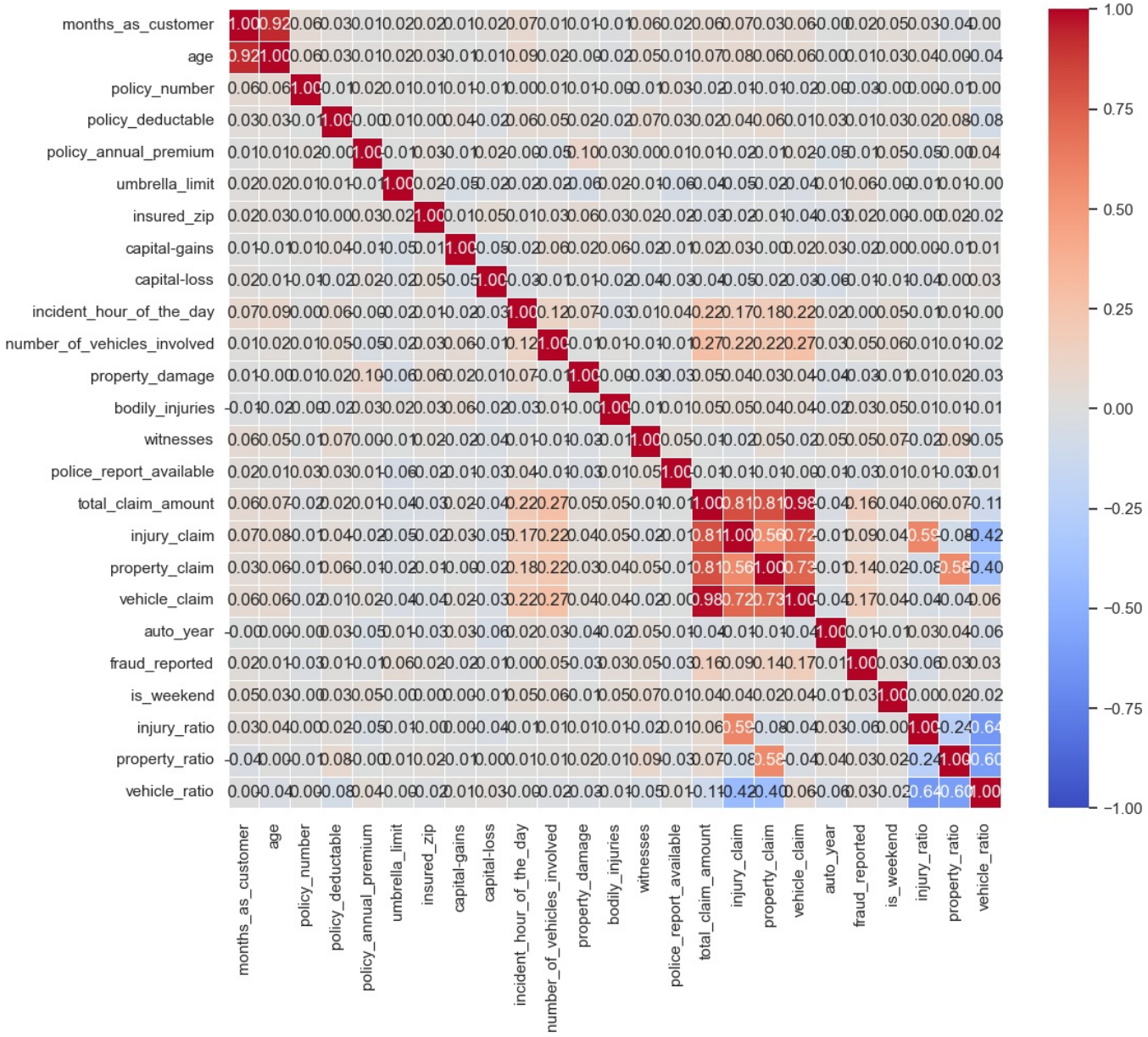
	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
property_damage	1000.0	-5.800000e-02	8.119700e-01	-1.00	-1.0000	0.0	1.000	1.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
police_report_available	1000.0	-2.900000e-02	8.104417e-01	-1.00	-1.0000	0.0	1.000	1.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00
fraud_reported	1000.0	2.470000e-01	4.314825e-01	0.00	0.0000	0.0	0.000	1.00

In [ ]:
```
import numpy as np

df_pr = df_clean.copy()

median_value = df_pr.loc[df_pr['umbrella_limit'] != -100000, 'umbrella_limit'].median()
df_pr.loc[df_pr['umbrella_limit'] == -100000, 'umbrella_limit'] = median_value
```

In [ ]:
```
df_features = df_pr.copy()

df_clean["policy_tenure_months"] = ((df_clean["incident_date"] - df_clean["policy_bind_date"]).dt.days / 30).as

df_features["incident_year"] = df_features["incident_date"].dt.year
df_features["incident_month"] = df_features["incident_date"].dt.month
```

```
df_features["incident_dow"] = df_features["incident_date"].dt.dayofweek
df_features["is_weekend"] = df_features["incident_dow"].isin([5, 6]).astype(int)

df_features["injury_ratio"] = df_features["injury_claim"] / (df_features["total_claim_amount"] + 1e-3)
df_features["property_ratio"] = df_features["property_claim"]  / (df_features["total_claim_amount"] + 1e-3)
df_features["vehicle_ratio"] = df_features["vehicle_claim"] / (df_features["total_claim_amount"] + 1e-3)

drop_dates = ["policy_bind_date", "incident_date"]
df_features = df_features.drop(drop_dates, axis=1)
```

In [ ]:
```
num_cols = df_features.select_dtypes(include=["int64", "float64"]).columns.tolist()

plt.figure(figsize=(12, 10))
correlation_matrix = df_features[num_cols].corr()

sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            vmin=-1, vmax=1,
            linewidths=0.5)

plt.show()
```



In [ ]:
```
from sklearn.model_selection import train_test_split
X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_i, y_i, test_size=0.25, random_state=42, stratify=
```

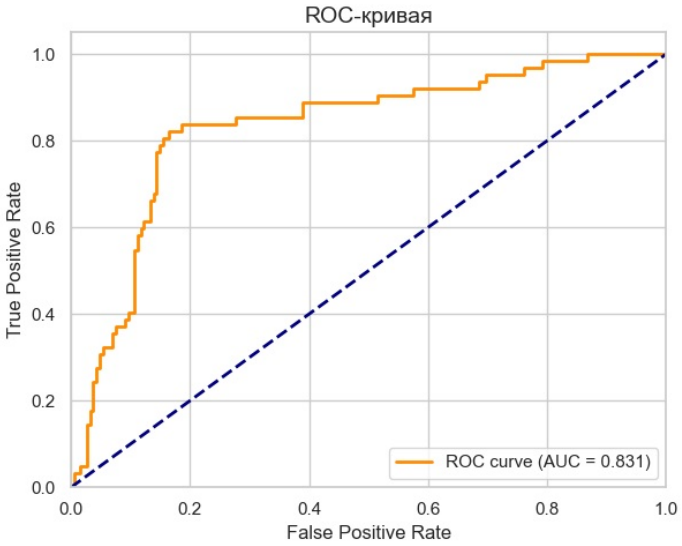
Я использовал BalancedRandomForestClassifier, оно даёт лучший скор на несбалансированных классах.

In [ ]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
```

```python
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import f1_score, roc_auc_score, classification_report
from imblearn.ensemble import BalancedRandomForestClassifier

from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE


num_cols = X_train_i.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = X_train_i.select_dtypes(include=["object", "category"]).columns.tolist()

categorical_pipe = Pipeline([
    ("onehot", OneHotEncoder(
        handle_unknown='infrequent_if_exist',
        sparse_output=False,
        min_frequency=2
    ))
])

ct = ColumnTransformer([
    ("cat", categorical_pipe, cat_cols)
])

model_pipe = Pipeline([
    ('ct', ct),
    ('model', BalancedRandomForestClassifier(random_state=42, n_estimators=500))
])

param_grid = {
    'model__n_estimators': [5, 50, 55],
    'model__max_depth': [10, 15, 20, None],
    'model__max_features': ['sqrt'],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4, 8],
    'model__class_weight': ['balanced'],
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(
    model_pipe,
    param_grid,
    cv=cv,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print(f"Лучшие параметры: {grid.best_params_}")

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print("-" * 30)
print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
print("="*50)
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
Лучшие параметры: {'model__class_weight': 'balanced', 'model__max_depth': 15, 'model__max_features': 'sqrt', 'mo
del__min_samples_leaf': 1, 'model__min_samples_split': 5, 'model__n_estimators': 50}
------------------------------
F1-score (класс 1): 0.7083
ROC-AUC: 0.8314
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.84      0.88       188
           1       0.62      0.82      0.71        62

    accuracy                           0.83       250
   macro avg       0.78      0.83      0.80       250
weighted avg       0.86      0.83      0.84       250

================================================
```

In [ ]:
```
graphics(y_test_i, y_pred, y_prob)
feature_important(grid, X_test_i, y_test_i)
```




```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[79], line 2
      1 graphics(y_test_i, y_pred, y_prob)
----> 2 feature_important(grid, X_test_i, y_test_i)

Cell In[68], line 34, in feature_important(model, X_test, y_test)
     32 def feature_important(model, X_test, y_test):
---> 34     perm_result = permutation_importance(
     35         model,
     36         X_test,
     37         y_test,
     38         n_repeats=10,
     39         random_state=42,
     40         n_jobs=-1
     41     )
     43     perm_df = pd.DataFrame({
     44         'feature': X_test.columns,
     45         'importance': perm_result.importances_mean
     46     })
     48     top_40_perm = perm_df.sort_values(by='importance', ascending=False).head(40)

File ~/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/utils/_param_validation.py:218, i
n validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    212 try:
    213     with config_context(
    214         skip_parameter_validation=(
    215             prefer_skip_nested_validation or global_skip_validation
    216         )
    217     ):
--> 218         return func(*args, **kwargs)
    219 except InvalidParameterError as e:
    220     # When the function is just a wrapper around an estimator, we allow
    221     # the function to delegate validation to the estimator, but we replace
    222     # the name of the estimator by the name of the function in the error
    223     # message to avoid confusion.
    224     msg = re.sub(
    225         r"parameter of \w+ must be",
    226         f"parameter of {func.__qualname__} must be",
    227         str(e),
```

```
    228     )

File ~/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/inspection/_permutation_importance.py:288, in permutation_importance(estimator, X, y, scoring, n_repeats, n_jobs, random_state, sample_weight, max_samples)
    285 scorer = check_scoring(estimator, scoring=scoring)
    286 baseline_score = _weights_scorer(scorer, estimator, X, y, sample_weight)
--> 288 scores = Parallel(n_jobs=n_jobs)(
    289     delayed(_calculate_permutation_scores)(
    290         estimator,
    291         X,
    292         y,
    293         sample_weight,
    294         col_idx,
    295         random_seed,
    296         n_repeats,
    297         scorer,
    298         max_samples,
    299     )
    300     for col_idx in range(X.shape[1])
    301 )
    303 if isinstance(baseline_score, dict):
    304     return {
    305         name: _create_importances_bunch(
    306             baseline_score[name],
   (...)    310         for name in baseline_score
    311     }

File ~/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/utils/parallel.py:82, in Parallel.__call__(self, iterable)
     73 warning_filters = warnings.filters
     74 iterable_with_config_and_warning_filters = (
     75     (
     76         _with_config_and_warning_filters(delayed_func, config, warning_filters),
   (...)     80     for delayed_func, args, kwargs in iterable
     81 )
---> 82 return super().__call__(iterable_with_config_and_warning_filters)

File ~/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/joblib/parallel.py:2072, in Parallel.__call__(self, iterable)
   2066 # The first item from the output is blank, but it makes the interpreter
   2067 # progress until it enters the Try/Except block of the generator and
   2068 # reaches the first `yield` statement. This starts the asynchronous
   2069 # dispatch of the tasks to the workers.
   2070 next(output)
-> 2072 return output if self.return_generator else list(output)

File ~/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/joblib/parallel.py:1682, in Parallel._get_outputs(self, iterator, pre_dispatch)
   1679     yield
   1681     with self._backend.retrieval_context():
-> 1682         yield from self._retrieve()
   1684 except GeneratorExit:
   1685     # The generator has been garbage collected before being fully
   1686     # consumed. This aborts the remaining tasks if possible and warn
   1687     # the user if necessary.
   1688     self._exception = True

File ~/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/joblib/parallel.py:1800, in Parallel._retrieve(self)
   1789 if self.return_ordered:
   1790     # Case ordered: wait for completion (or error) of the next job
   1791     # that have been dispatched and not retrieved yet. If no job
   (...)   1795     # control only have to be done on the amount of time the next
   1796     # dispatched job is pending.
   1797     if (nb_jobs == 0) or (
   1798         self._jobs[0].get_status(timeout=self.timeout) == TASK_PENDING
   1799     ):
-> 1800         time.sleep(0.01)
   1801         continue
   1803 elif nb_jobs == 0:
   1804     # Case unordered: jobs are added to the list of jobs to
   1805     # retrieve `self._jobs` only once completed or in error, which
   (...)   1811     # timeouts before any other dispatched job has completed and
   1812     # been added to `self._jobs` to be retrieved.

KeyboardInterrupt:
```

Лесу удалось хуже распознать мошенника. Даже использование балансного леса не помогло, ROC_AUC оказался хуже чем при использовании обычного решающего дерева.

Думаю, что это можно объяснить следующим образом. Как мы раньше видили, на то, является ли человек мошенником или нет влияет не так много факторов. Обучение каждого из деревьев леса происходит на случайной подвыборке, в которую этот сигнал

может не попасть, поэтому он будет размыт другими факторами. Одиночное дерево же в свою очередь обучается на всей выборке, поэтому ему в этом плане легче.

My implementation

In [ ]:
```
import numpy as np
from collections import Counter
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
from sklearn.utils.multiclass import unique_labels

class Node:
    def __init__(self, feature_index=None, threshold=None, left=None, right=None,
                 value=None, proba=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
        self.proba = proba

class MyDecisionTreeClassifier(ClassifierMixin, BaseEstimator):
    def __init__(self, max_depth=None, min_samples_leaf=1, min_samples_split=2, criterion='gini'):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
        self.criterion = criterion
        self.root = None

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)
        self.n_classes_ = len(self.classes_)
        self.n_features_ = X.shape[1]

        self.root = self._build_tree(X, y)
        return self

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)
        return np.array([self._make_prediction(x, self.root) for x in X])

    def predict_proba(self, X):
        check_is_fitted(self)
        X = check_array(X)

        probas = []
        for x in X:
            node_probs = self._get_node_proba(x, self.root)
            row_proba = [node_probs.get(c, 0.0) for c in self.classes_]
            probas.append(row_proba)

        return np.array(probas)

    def _build_tree(self, X, y, depth=0):
        num_samples, num_features = X.shape
        unique_classes = len(np.unique(y))

        if (self.max_depth is not None and depth >= self.max_depth) or \
           (num_samples < self.min_samples_split or num_samples < self.min_samples_leaf * 2) or \
           (unique_classes == 1):
            return self._create_leaf_node(y)

        best_split = self._get_best_split(X, y, num_features)

        if best_split["gain"] > 0:
            left_subtree = self._build_tree(best_split["X_left"], best_split["y_left"], depth + 1)
            right_subtree = self._build_tree(best_split["X_right"], best_split["y_right"], depth + 1)
            return Node(
                feature_index=best_split["feature_index"],
                threshold=best_split["threshold"],
                left=left_subtree,
                right=right_subtree
            )

        return self._create_leaf_node(y)

    def _create_leaf_node(self, y):
        counts = Counter(y)
        most_common = counts.most_common(1)[0][0]
```

```
        total = len(y)
        probs = {cls: count / total for cls, count in counts.items()}
        return Node(value=most_common, proba=probs)

    def _get_best_split(self, X, y, num_features):
        best_split = {"gain": -1, "feature_index": None, "threshold": None}
        max_info_gain = -float("inf")
        parent_impurity = self._calculate_impurity(y)

        for feature_index in range(num_features):
            feature_values = X[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            for threshold in possible_thresholds:
                left_mask = feature_values <= threshold
                right_mask = ~left_mask
                if np.sum(left_mask) < self.min_samples_leaf or np.sum(right_mask) < self.min_samples_leaf:
                    continue

                y_left, y_right = y[left_mask], y[right_mask]
                n = len(y)
                n_l, n_r = len(y_left), len(y_right)
                child_impurity = (n_l / n) * self._calculate_impurity(y_left) + \
                                 (n_r / n) * self._calculate_impurity(y_right)
                info_gain = parent_impurity - child_impurity

                if info_gain > max_info_gain:
                    max_info_gain = info_gain
                    best_split = {
                        "feature_index": feature_index,
                        "threshold": threshold,
                        "X_left": X[left_mask],
                        "y_left": y_left,
                        "X_right": X[right_mask],
                        "y_right": y_right,
                        "gain": info_gain
                    }
        return best_split

    def _calculate_impurity(self, y):
        if len(y) == 0: return 0
        counts = np.unique(y, return_counts=True)[1]
        probabilities = counts / len(y)
        if self.criterion == 'gini':
            return 1 - np.sum(probabilities ** 2)
        elif self.criterion == 'entropy':
            return -np.sum(probabilities * np.log2(probabilities + 1e-9))
        else:
            raise ValueError("Unknown criterion. Use 'gini' or 'entropy'")

    def _make_prediction(self, x, node):
        if node.value is not None: return node.value
        if x[node.feature_index] <= node.threshold:
            return self._make_prediction(x, node.left)
        return self._make_prediction(x, node.right)

    def _get_node_proba(self, x, node):
        if node.proba is not None: return node.proba
        if x[node.feature_index] <= node.threshold:
            return self._get_node_proba(x, node.left)
        return self._get_node_proba(x, node.right)

class MyRandomForestClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self,
                 n_estimators=100,
                 max_depth=None,
                 min_samples_leaf=1,
                 criterion='gini',
                 max_features="sqrt",
                 bootstrap=True,
                 random_state=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
        self.criterion = criterion
        self.max_features = max_features
        self.bootstrap = bootstrap
        self.random_state = random_state

        self.trees_ = []
        self.features_per_tree_ = []

    def fit(self, X, y):
        X, y = check_X_y(X, y)
```

```
        self.classes_ = unique_labels(y)
        self.n_classes_ = len(self.classes_)
        self.n_samples_, self.n_features_ = X.shape

        rng = np.random.default_rng(self.random_state)

        if self.max_features == "sqrt":
            self.max_features_ = int(np.sqrt(self.n_features_))
        elif self.max_features == "log2":
            self.max_features_ = int(np.log2(self.n_features_))
        elif isinstance(self.max_features, int):
            self.max_features_ = self.max_features
        elif isinstance(self.max_features, float):
            self.max_features_ = int(self.max_features * self.n_features_)
        else:
            self.max_features_ = self.n_features_

        self.trees_ = []
        self.features_per_tree_ = []

        for _ in range(self.n_estimators):
            if self.bootstrap:
                indices = rng.integers(0, self.n_samples_, size=self.n_samples_)
            else:
                indices = np.arange(self.n_samples_)

            X_sample = X[indices]
            y_sample = y[indices]

            feature_subset = rng.choice(
                self.n_features_,
                size=self.max_features_,
                replace=False
            )
            self.features_per_tree_.append(feature_subset)

            X_sub = X_sample[:, feature_subset]

            tree = MyDecisionTreeClassifier(
                max_depth=self.max_depth,
                min_samples_leaf=self.min_samples_leaf,
                criterion=self.criterion
            )
            tree.fit(X_sub, y_sample)
            self.trees_.append(tree)

        return self

    def predict_proba(self, X):
        check_is_fitted(self)
        X = check_array(X)

        probas = np.zeros((len(X), self.n_classes_))

        for tree, features in zip(self.trees_, self.features_per_tree_):
            X_sub = X[:, features]
            probas += tree.predict_proba(X_sub)

        probas /= self.n_estimators
        return probas

    def predict(self, X):
        check_is_fitted(self)
        proba = self.predict_proba(X)
        class_indices = np.argmax(proba, axis=1)
        return self.classes_[class_indices]
```

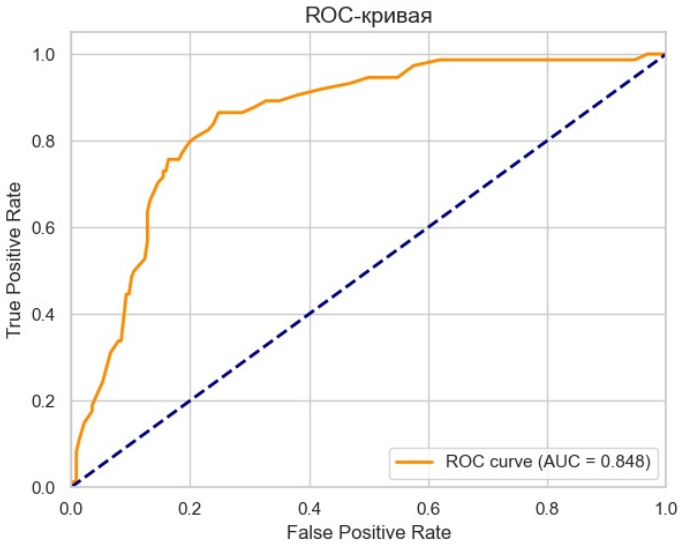
In [ ]:

```
from sklearn.metrics import classification_report, f1_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)
```

```
F1-score (класс 1): 0.4237
ROC-AUC: 0.8485
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.92      0.86       226
           1       0.57      0.34      0.42        74

    accuracy                           0.77       300
   macro avg       0.69      0.63      0.64       300
weighted avg       0.75      0.77      0.75       300

==================================================
```

In [ ]:
```
graphics(y_test, y_pred, y_prob)
```

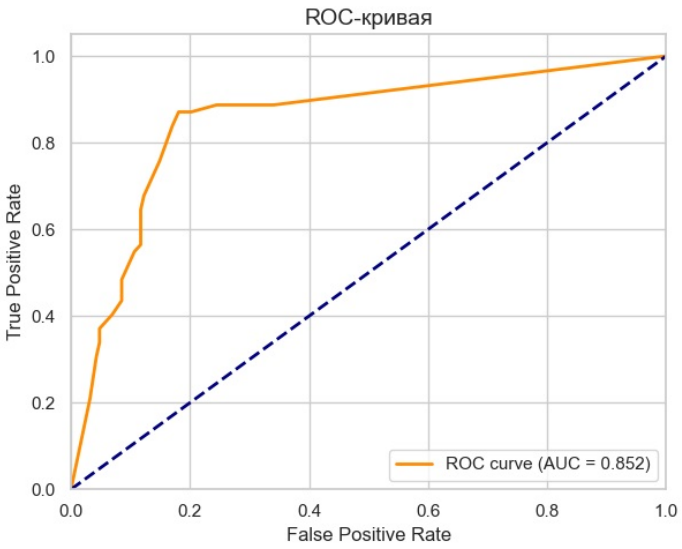


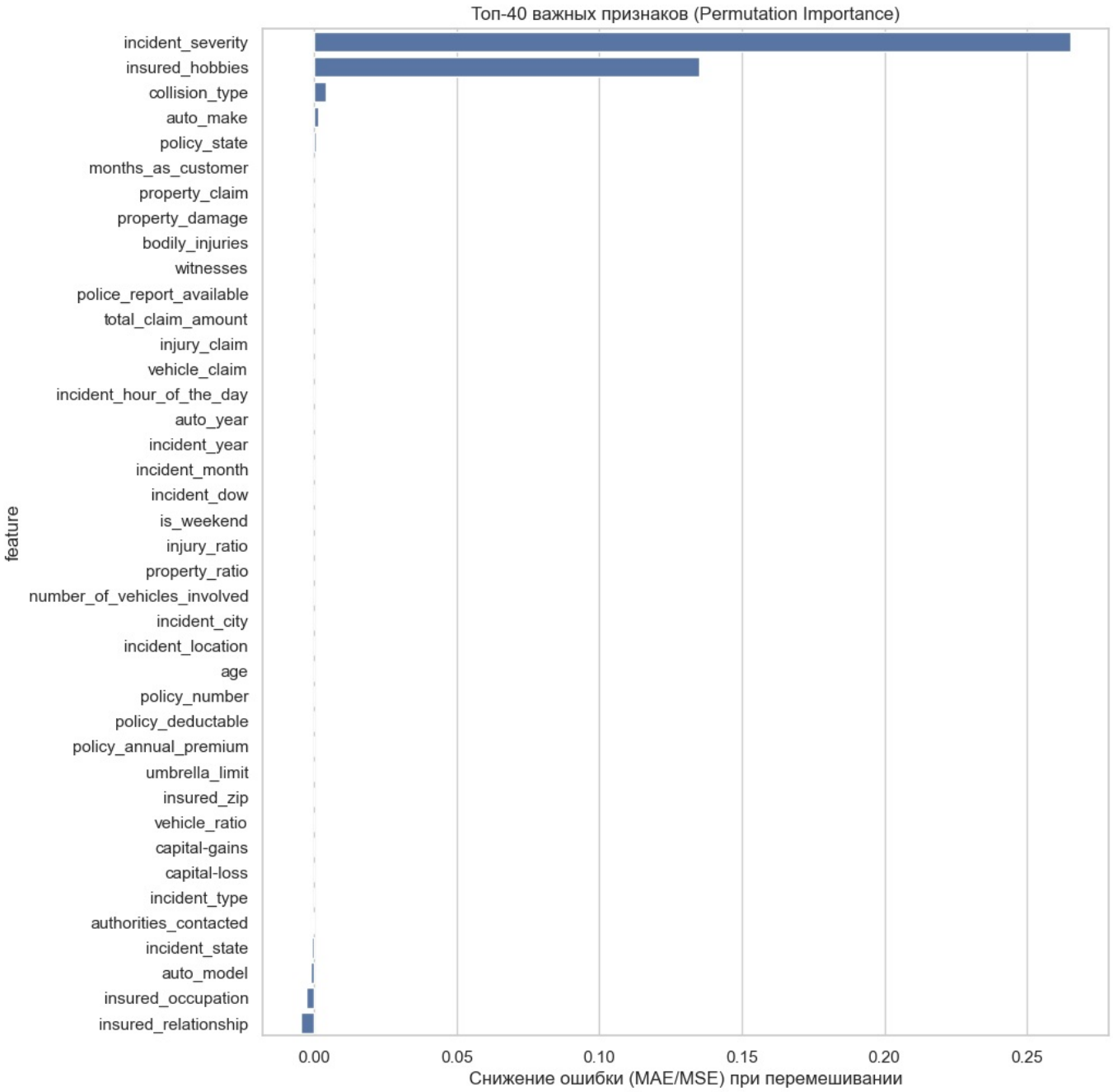
In [ ]:
```
model_pipe = Pipeline([
    ('ct', ct),
    ('model', MyDecisionTreeClassifier())
])

param_grid = {
    'model__max_depth': [10, 15, 20, None],
    'model__min_samples_leaf': [1, 2, 4, 8],
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid = GridSearchCV(
    model_pipe,
    param_grid,
    cv=cv,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print(f"Лучшие параметры: {grid.best_params_}")

best_model = grid.best_estimator_

y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print("-" * 30)
print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
```

```
print("="*50)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Лучшие параметры: {'model__max_depth': 15, 'model__min_samples_leaf': 8}
------------------------------
F1-score (класс 1): 0.5862
ROC-AUC: 0.8515
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       188
           1       0.63      0.55      0.59        62

    accuracy                           0.81       250
   macro avg       0.74      0.72      0.73       250
weighted avg       0.80      0.81      0.80       250

==================================================
```

In [ ]:
```
graphics(y_test_i, y_pred, y_prob)
feature_important(grid, X_test_i, y_test_i)
```





Собственная реализация работает чуть хуже, поскольку нет поддержки задания min_samples_split и class_weight, последний особенно сильно влияет на результат, поскольку классы в датасете несбалансированны.

	Base Random Forest	Random Forest	My Random Forest
ROC_AUC	0.8398	0.8314	0.8515
F1 (1 класс)	0.85	0.88	0.88
F1 (2 класс)	0.39	0.71	0.59

Вывод

Использование случайного леса в регрессии показывает результат около того, что показывала линейная регрессия, однако модель склонна переобучаться при неверно подобранных параметрах. В задаче классификации результат оказался лучше, модель показала очень даже неплохие результаты.

Регрессия

Задача: предсказать зарплату игрока НХЛ по его статистике с использованием модели случайного леса.

Будем использовать MAE, так как она очень наглядная, будет легко понять, насколько долларов ошибается модель. В качестве дополнительной метрики будем использовать R^2, чтобы смотреть насколько отличается предсказание от среднего.

Baseline Random Forest

In [1]:
```
import kagglehub
import pandas as pd
import numpy as np

path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")

df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)
df.head()
```

```
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: I
Progress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_i
nstall.html
  from .autonotebook import tqdm as notebook_tqdm
```

Out[1]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	14
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2'

5 rows × 154 columns

In [2]:
```
df['Born'] = pd.to_datetime(df['Born'], format='%y-%m-%d')
df_clean = df.fillna(0)
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean = df_clean.drop("Born", axis=1)
```

In [3]:
```
from sklearn.model_selection import train_test_split

TARGET_NAME = "Salary"
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

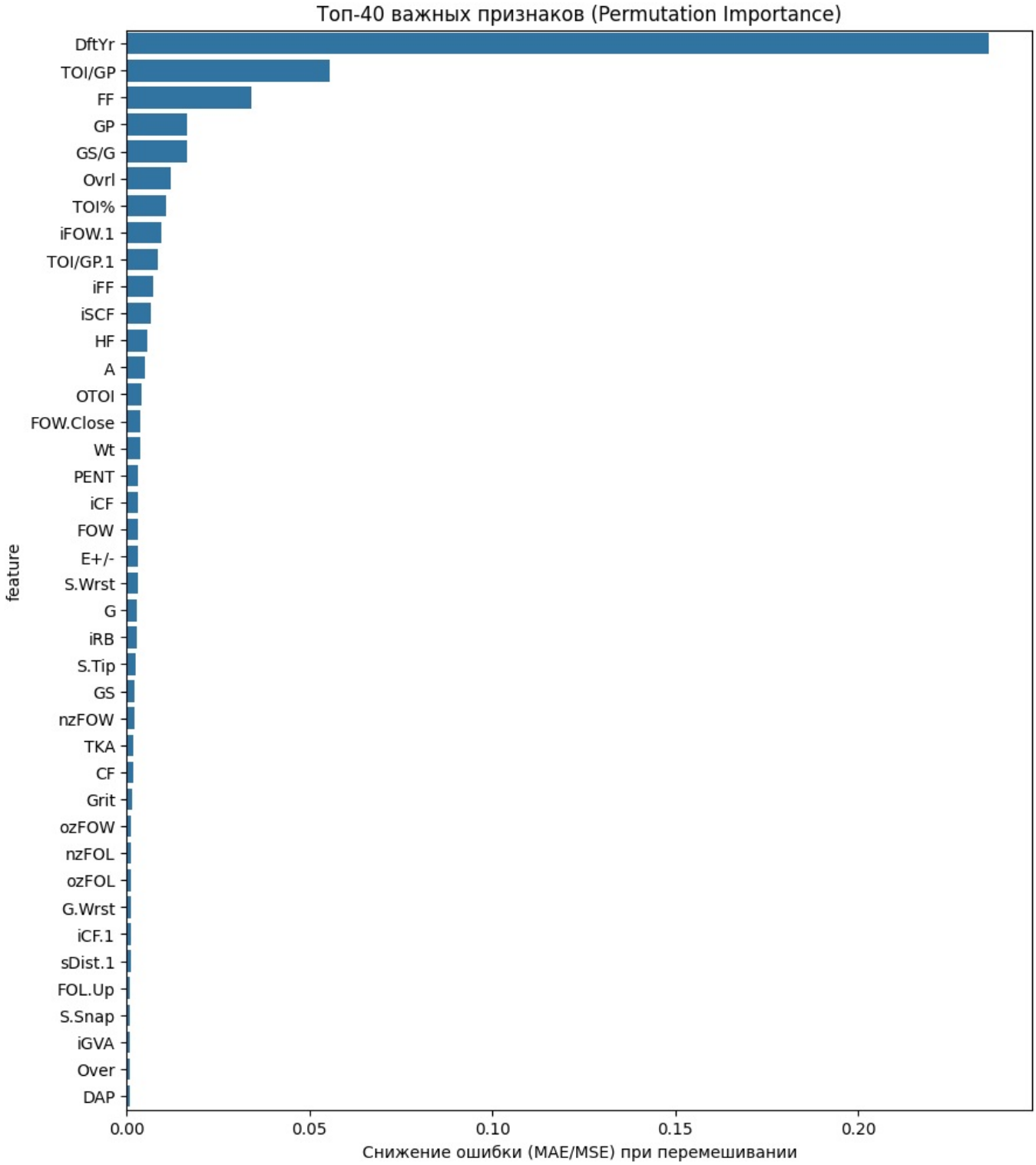
In [4]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import seaborn as sns

model = GradientBoostingRegressor()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

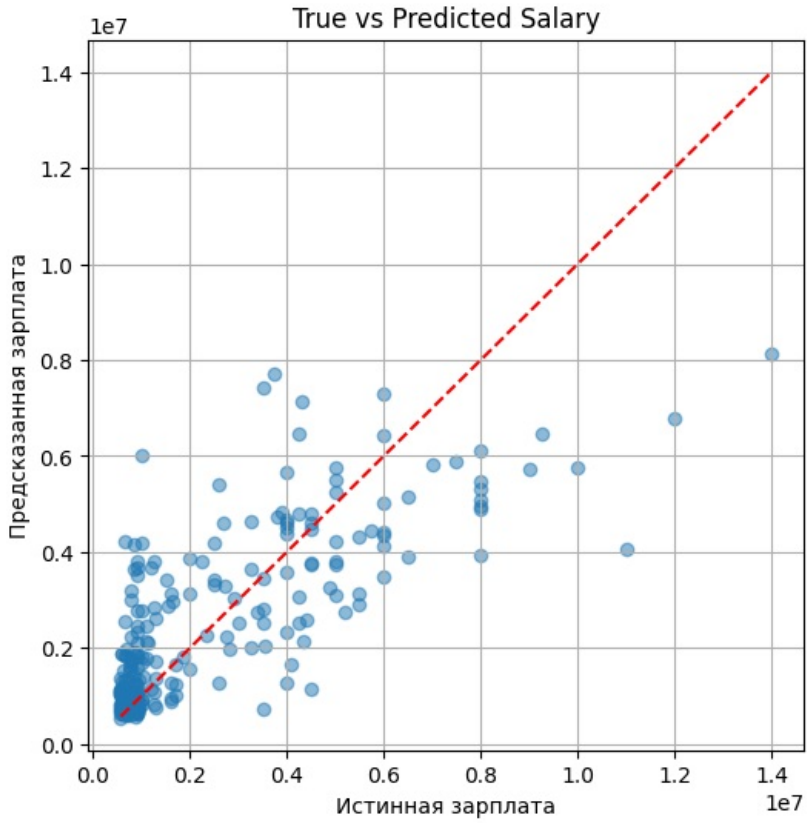
```
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, [y_train.mean()]*len(y_test))

print(f"--- Baseline LinearRegreesion (Raw Data) ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

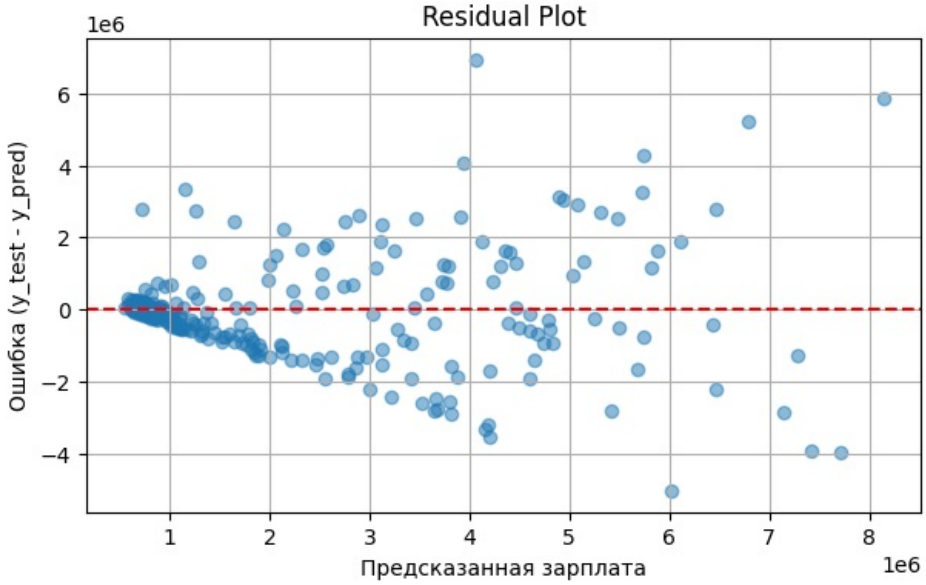
```
--- Baseline LinearRegreesion (Raw Data) ---
MAE (Ошибка в долларах): 962253.788
R2 Score: 0.600
```

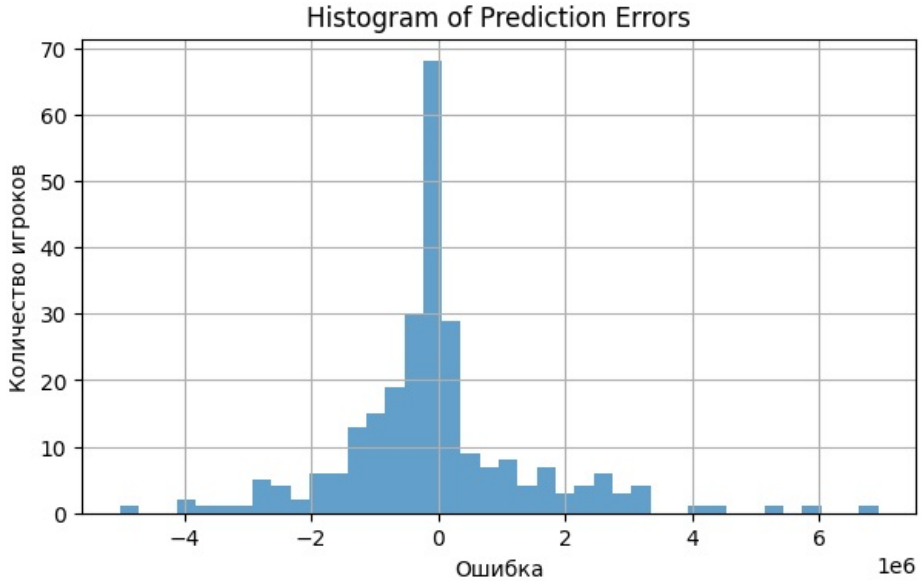
In [5]:

```
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.inspection import permutation_importance

def important_features(model, X_test, y_test):
    perm_result = permutation_importance(
        model,
        X_test,
        y_test,
        n_repeats=10,
        random_state=52,
        n_jobs=-1
    )

    perm_df = pd.DataFrame({
        'feature': X_test.columns,
        'importance': perm_result.importances_mean
    })

    top_40_perm = perm_df.sort_values(by='importance', ascending=False).head(40)

    plt.figure(figsize=(10, 12))
    sns.barplot(data=top_40_perm, x='importance', y='feature')
    plt.title("Топ-40 важных признаков (Permutation Importance)")
    plt.xlabel("Снижение ошибки (MAE/MSE) при перемешивании")
    plt.show()

def true_vs_predicted(y_test, y_pred):
    plt.figure(figsize=(6,6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
    plt.xlabel("Истинная зарплата")
    plt.ylabel("Предсказанная зарплата")
    plt.title("True vs Predicted Salary")
    plt.grid(True)
    plt.show()

def resudial(y_test, y_pred):
    residuals = y_test - y_pred
    plt.figure(figsize=(7,4))
    plt.scatter(y_pred, residuals, alpha=0.5)
    plt.axhline(0, color='red', linestyle='--')
    plt.xlabel("Предсказанная зарплата")
    plt.ylabel("Ошибка (y_test - y_pred)")
    plt.title("Residual Plot")
    plt.grid(True)
    plt.show()
def histogramm_error(y_test, y_pred):
    plt.figure(figsize=(7,4))
    residuals = y_test - y_pred
    plt.hist(residuals, bins=40, alpha=0.7)
    plt.title("Histogram of Prediction Errors")
    plt.xlabel("Ошибка")
    plt.ylabel("Количество игроков")
    plt.grid(True)
    plt.show()

important_features(model, X_test, y_test)
true_vs_predicted(y_test, y_pred)
resudial(y_test, y_pred)
histogramm_error(y_test, y_pred)
```


Топ-40 важных признаков (Permutation Importance)
DftYr
TOI/GP
FF
GP
GS/G
Ovrl
TOI%
iFOW.1
TOI/GP.1
iFF
iSCF
HF
A
OTOI
FOW.Close
Wt
PENT
iCF
FOW
E+/-
S.Wrst
G
iRB
S.Tip
GS
nzFOW
TKA
CF
Grit
ozFOW
nzFOL
ozFOL
G.Wrst
iCF.1
sDist.1
FOL.Up
S.Snap
iGVA
Over
DAP
feature
0.00
0.05
0.10
0.15
0.20
Снижение ошибки (MAE/MSE) при перемешивании



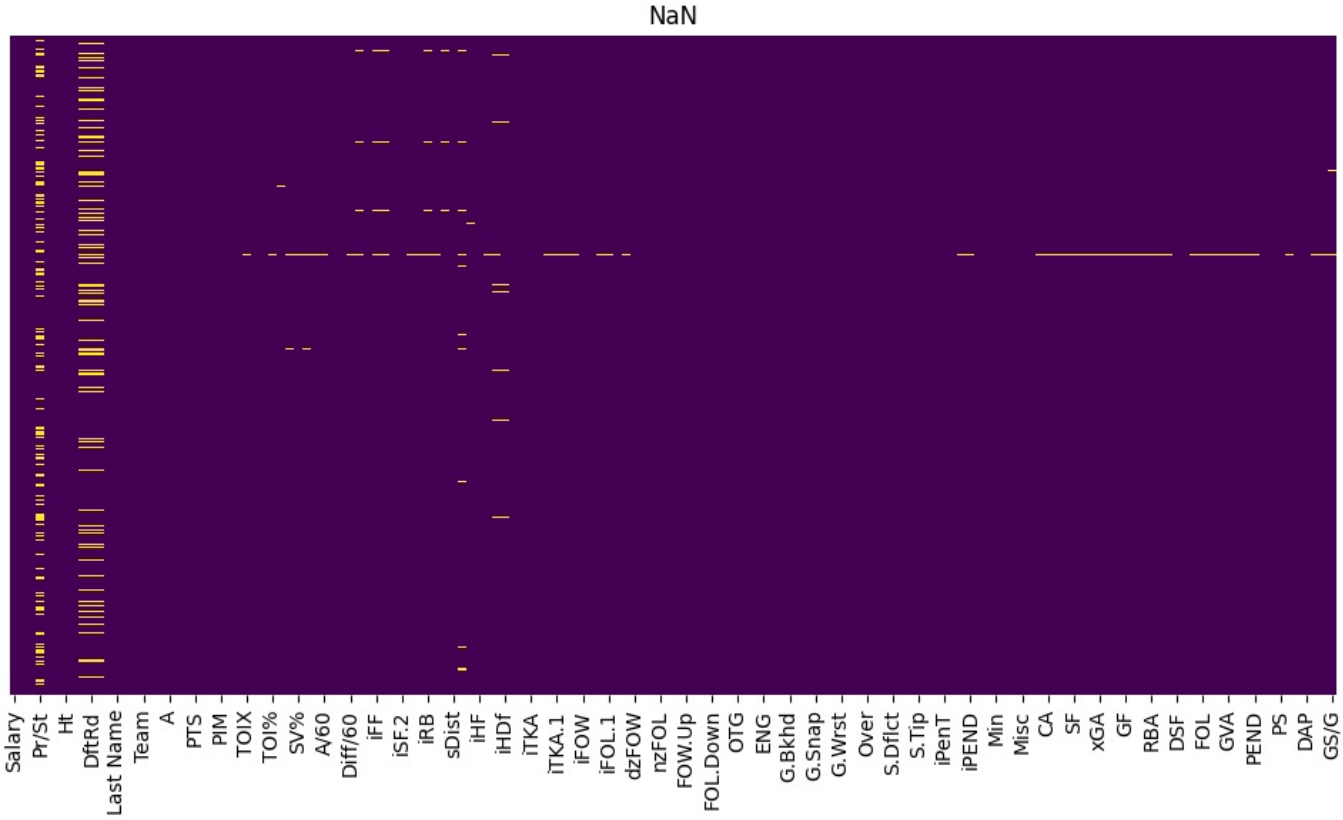




Бэйзлайн градиентного бустинга выдал очень приличный результат. Модель лучше определяет зарплаты более дорогих хоккеистов, однако эта модель стала худше справляется с более дешевыми, более распыленными стали точки вокруг главной диагонали.

Improved Gradient Boosting

In [6]:
```
import kagglehub
from matplotlib import pyplot as plt
import seaborn as sns

path = kagglehub.dataset_download(
    "camnugent/predict-nhl-player-salaries/versions/2"
)

df1 = pd.read_csv(path + "/train.csv")
df2 = pd.read_csv(path + "/test.csv")
salary = pd.read_csv(path + "/test_salaries.csv")
df2['Salary'] = salary['Salary'].values
df2 = df2[df1.columns]
df = pd.concat([df1, df2], ignore_index=True)

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [7]:
```
num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
df[num_cols].describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Salary	874.0	2.325289e+06	2.298253e+06	575000.00	742500.00	925000.00	3700000.00	14000000.00
Ht	874.0	7.308238e+01	2.105485e+00	66.00	72.00	73.00	75.00	81.00
Wt	874.0	2.008432e+02	1.506008e+01	157.00	190.00	200.00	210.00	265.00
DftYr	749.0	2.008708e+03	4.380158e+00	1990.00	2006.00	2010.00	2012.00	2016.00
DftRd	749.0	2.742323e+00	1.988358e+00	1.00	1.00	2.00	4.00	9.00
...	...	...	...	...	...	...	...	...
Grit	874.0	1.267815e+02	1.016121e+02	0.00	41.00	114.00	190.00	622.00
DAP	874.0	9.215675e+00	7.815029e+00	0.00	4.60	7.60	12.00	61.00
Pace	873.0	1.089439e+02	8.899877e+00	75.00	104.70	109.20	113.90	175.70
GS	873.0	2.187331e+01	2.198638e+01	-4.30	2.60	15.70	35.40	104.70
GS/G	872.0	3.401606e-01	2.925900e-01	-0.81	0.14	0.31	0.53	1.28

144 rows × 8 columns

In [8]:

```
import pandas as pd
import numpy as np

num_cols = df.select_dtypes(include=["int64", "float64"]).columns

corr_matrix = df[num_cols].corr()

corr_pairs = corr_matrix.unstack().reset_index()
corr_pairs.columns = ['feature_1', 'feature_2', 'correlation']

corr_pairs = corr_pairs[corr_pairs['feature_1'] < corr_pairs['feature_2']]

corr_pairs = corr_pairs.reindex(
    corr_pairs['correlation'].abs().sort_values(ascending=False).index
)

top40 = corr_pairs.head(40)

display(top40)
upper = np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
upper_corr = corr_matrix.where(upper)
to_drop = [col for col in upper_corr.columns if any(upper_corr[col] > 0.998)]

df_clean = df.drop(columns=to_drop)

df_clean.head()
```

	feature_1	feature_2	correlation
4931	iSF.1	iSF.2	0.999996
8267	iFOL	iFOL.1	0.999981
8122	iFOW	iFOW.1	0.999979
6381	iHF	iHF.1	0.999957
2321	TOI	TOIX	0.999952
7398	iBLK	iBLK.1	0.999948
4787	iSF	iSF.2	0.999937
4786	iSF	iSF.1	0.999931
7108	iGVA	iGVA.1	0.999907
7253	iTKA	iTKA.1	0.999843
2611	TOI/GP	TOI/GP.1	0.999289
15807	CF	FF	0.999216
4351	iCF	iCF.1	0.998819
16097	FF	SF	0.998722
10424	FOW.Close	iFOW	0.998613
10426	FOW.Close	iFOW.1	0.998603
16242	FA	SA	0.998366
10571	FOL.Close	iFOL.1	0.998221
10569	FOL.Close	iFOL	0.998215
15952	CA	FA	0.998114
15809	CF	SF	0.997739
15954	CA	SA	0.995524
4642	iFF	iSF.1	0.995443
4643	iFF	iSF.2	0.995427
4641	iFF	iSF	0.995415
17108	SCA	xGA	0.995015
16963	SCF	xGF	0.994495
1002	GP	OTOI	0.993385
13008	Wide	iMiss	0.993229
16244	FA	xGA	0.992949
15956	CA	xGA	0.992542
10136	FOW.Down	iFOW	0.992496
10138	FOW.Down	iFOW.1	0.992464
16099	FF	xGF	0.991109
2177	Shifts	TOIX	0.991098
16532	SA	xGA	0.991094
4496	iCF.1	iFF	0.990960
2899	TOI%	TOI/GP.1	0.990941
10281	FOL.Down	iFOL	0.990924
2898	TOI%	TOI/GP	0.990914

Out[8]:

	Salary	Born	City	Pr/St	Cntry	Nat	Ht	Wt	DftYr	DftRd	...	PEND	OPS	DPS	PS	OTOI	Grit	DAP	Pace	
0	925000.0	97-01-30	Sainte-Marie	QC	CAN	CAN	74	190	2015.0	1.0	...	1.0	0.0	-0.2	-0.2	40.03	1	0.0	175.7	-(
1	2250000.0	93-12-21	Ottawa	ON	CAN	CAN	74	207	2012.0	1.0	...	98.0	-0.2	3.4	3.2	2850.59	290	13.3	112.5	1(
2	8000000.0	88-04-16	St. Paul	MN	USA	USA	72	218	2006.0	1.0	...	70.0	3.7	1.3	5.0	2486.75	102	6.6	114.8	3(
3	3500000.0	92-01-07	Ottawa	ON	CAN	CAN	77	220	2010.0	1.0	...	22.0	0.0	0.4	0.5	1074.41	130	17.5	105.1	5
4	1750000.0	94-03-29	Toronto	ON	CAN	CAN	76	217	2012.0	1.0	...	68.0	-0.1	1.4	1.3	3459.09	425	8.3	99.5	2(

5 rows × 137 columns

In [9]:
```
df_features = df_clean.copy()

df_features['Born'] = pd.to_datetime(df_features['Born'], format='%y-%m-%d')
reference_date = pd.Timestamp('2016-10-01')
df_features['Age'] = (reference_date - df_features['Born']).dt.days / 365.25

df_features['Experience'] = reference_date.year - df_features['DftYr']
df_features['Age_squared'] = df_features['Age'] ** 2
df_features['PTS_per_GP'] = df_features['PTS'] / df_features['GP'].replace(0, 1)
df_features['Is_Drafted'] = df_features['DftYr'].notna().astype(int)
df_features['Physical_Impact'] = df_features['Wt'] * df_features['Ht']

features_to_drop = ['Born', 'Last Name', 'First Name', 'Nat', 'Pr/St', 'City']
df_features = df_features.drop(features_to_drop, axis=1)
df_features['Match'].value_counts()
```

Out[9]:
```
Match
0    870
1      4
Name: count, dtype: int64
```

In [10]:
```
has_nan = df_features.isnull().any()

columns_with_nan = has_nan[has_nan].index.tolist()

print("Столбцы, содержащие хотя бы один NaN:")
columns_with_nan
```

```
Столбцы, содержащие хотя бы один NaN:
```

Out[10]:
```
['DftYr',
 'DftRd',
 'Ovrl',
 'TOI%',
 'IPP%',
 'SH%',
 'SV%',
 'PDO',
 'F/60',
 'A/60',
 'Diff/60',
 'iCF',
 'iFF',
 'iSF',
 'ixG',
 'iSCF',
 'iRB',
 'iRS',
 'iDS',
 'sDist.1',
 'Pass',
 'iHA',
 'iHDf',
 'BLK%',
 '%FOT',
 'iPENT',
 'iPEND',
 'CF',
 'CA',
 'xGF',
 'xGA',
 'SCF',
 'SCA',
 'GF',
 'GA',
 'RBF',
 'RBA',
 'RSF',
 'RSA',
 'FOW',
 'FOL',
 'HF',
 'HA',
 'GVA',
 'TKA',
 'PENT',
 'PEND',
 'OTOI',
 'Pace',
 'GS',
 'GS/G',
 'Experience']
```

Тут обработку выбросов не добавляю. Результаты с ней ухудшились.

In [11]:
```
TARGET_NAME = 'Salary'

X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]

X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(
    X_i, y_i, test_size=0.3, random_state=42
)
```

In [12]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

cat_cols = X_i.select_dtypes(include=['object', 'category']).columns.tolist()

cat_branch = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

num_cols = X_i.select_dtypes(include=["int64", "float64"]).columns

num_branch = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
])
```

```
ct = ColumnTransformer(
    transformers=[
        ("cat_proc", cat_branch, cat_cols),
        ("num_proc", num_branch, num_cols),
    ], remainder='drop'
)
```

В данном случае я использовать реализацию градиентного бустинга из scikit-learn, перебрал внутренние параметры дерева, штраф регуляризации и прочее. Использовал RandomSearch, потому что Grid ОЧЕНЬ долго обучался конечно.

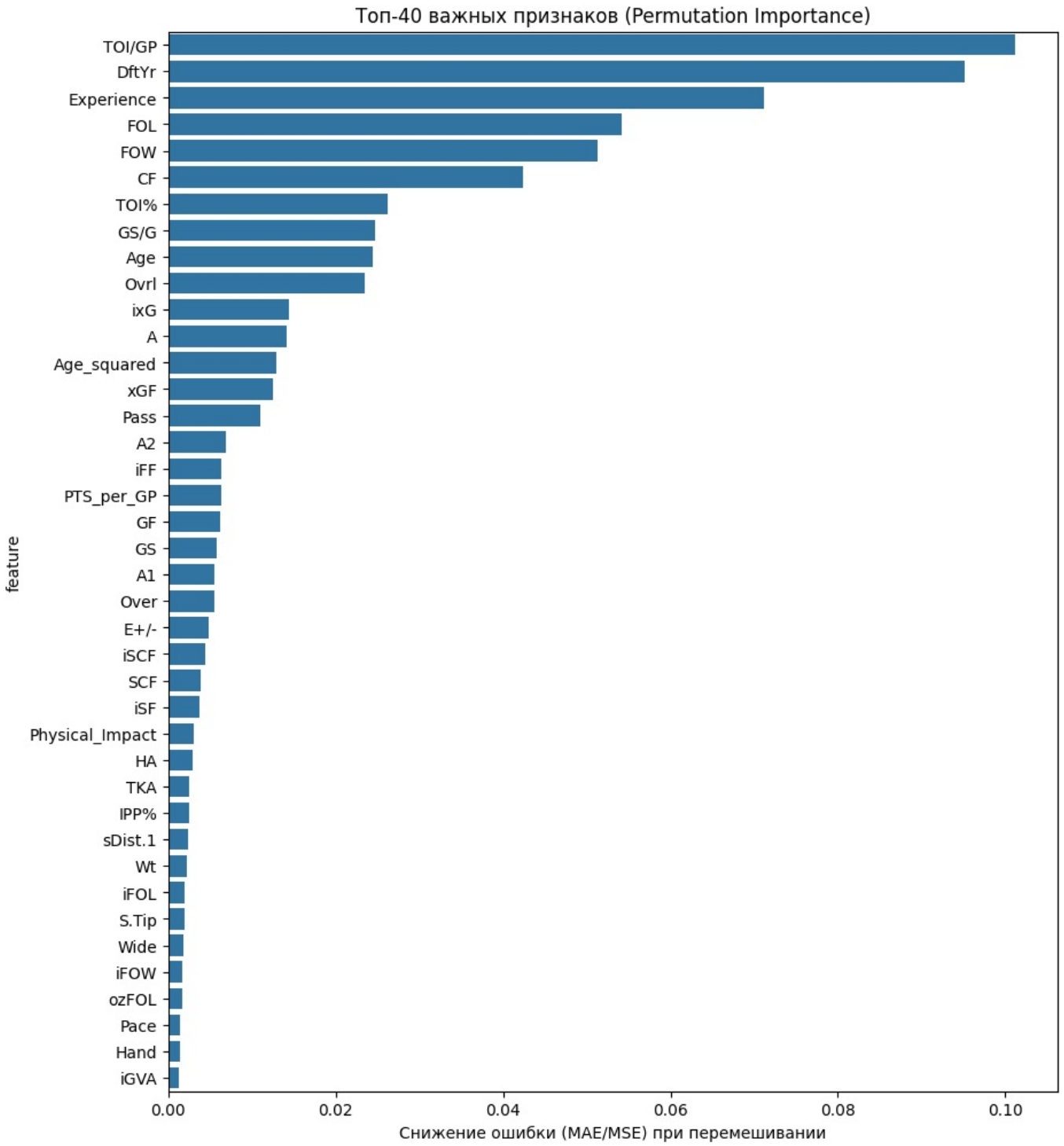
In [13]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=GradientBoostingRegressor(random_state=42),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

from scipy.stats import randint, uniform

param_dist = {
    'model_wrapper__regressor__n_estimators': randint(200, 800),
    'model_wrapper__regressor__learning_rate': uniform(0.01, 0.2),
    'model_wrapper__regressor__max_depth': randint(3, 7),
    'model_wrapper__regressor__min_samples_leaf': randint(7, 15),
    'model_wrapper__regressor__subsample': uniform(0.6, 0.4),
}


scoring = {
    'MAE': 'neg_mean_absolute_error',
    'R2': 'r2'
}
grid = RandomizedSearchCV(
    full_pipeline,
    param_distributions=param_dist,
    n_iter=30,
    cv=3,
    scoring=scoring,
    refit='MAE',
    n_jobs=-1,
    verbose=1,
    random_state=42
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
y_pred_train = grid.best_estimator_.predict(X_train_i)

mae = mean_absolute_error(y_test_i, y_pred)
train_mae = mean_absolute_error(y_train_i, y_pred_train)
r2 = r2_score(y_test_i, y_pred)

print(f"\n--- Gradient Boosting Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"Train MAE: {train_mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

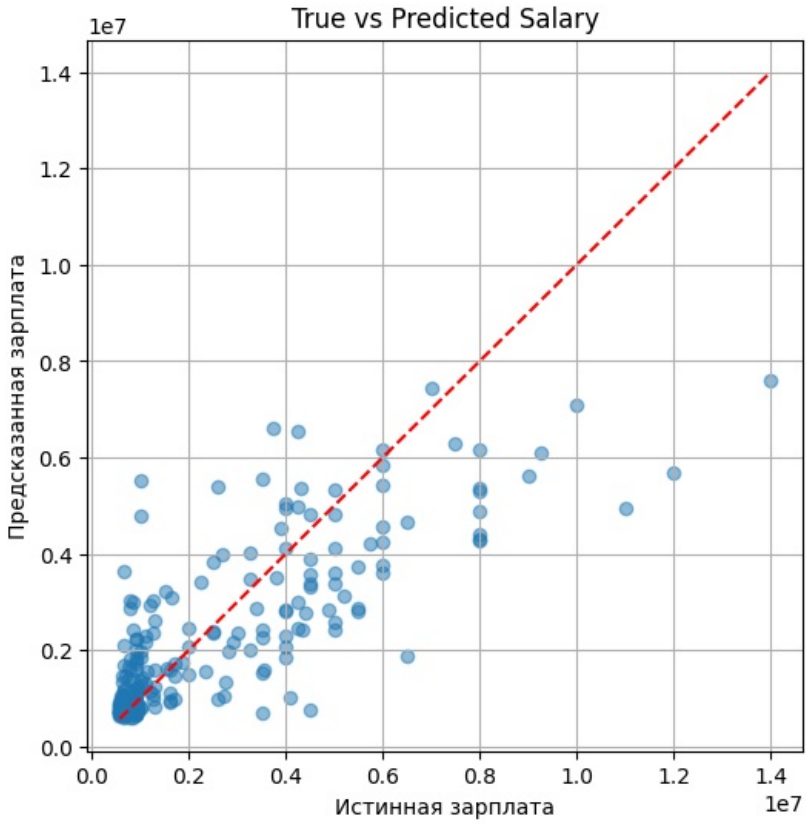
```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Лучшие параметры: {'model_wrapper__regressor__learning_rate': np.float64(0.014612485008283152), 'model_wrapper__
regressor__max_depth': 5, 'model_wrapper__regressor__min_samples_leaf': 9, 'model_wrapper__regressor__n_estimato
rs': 710, 'model_wrapper__regressor__subsample': np.float64(0.6557975442608167)}

--- Gradient Boosting Regressor ---
MAE: 850100.263
Train MAE: 182388.563
R2 Score: 0.640
```

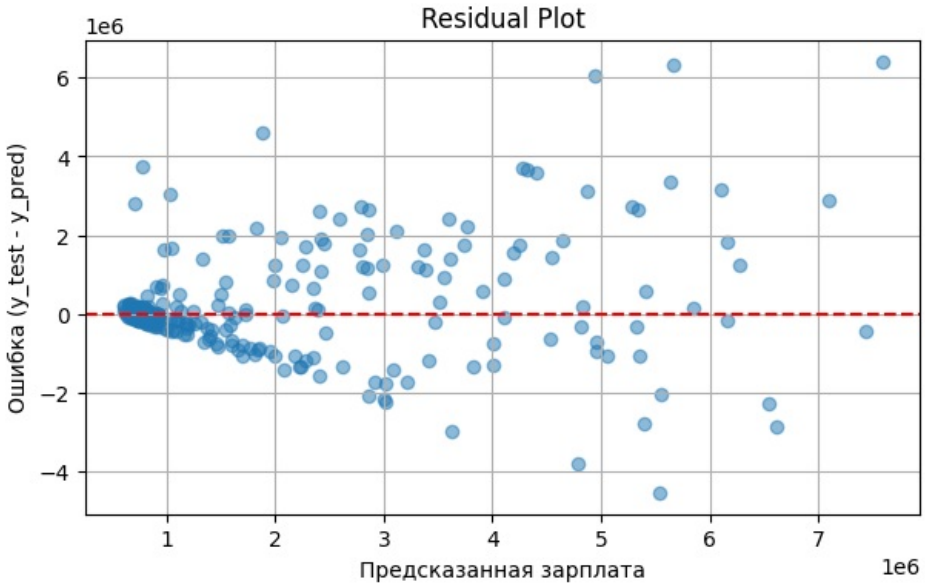
Кажется, бустинг переучился. Но метрика получилась лучшей из всех моделей, поэтому я думаю, что это не очень критично.

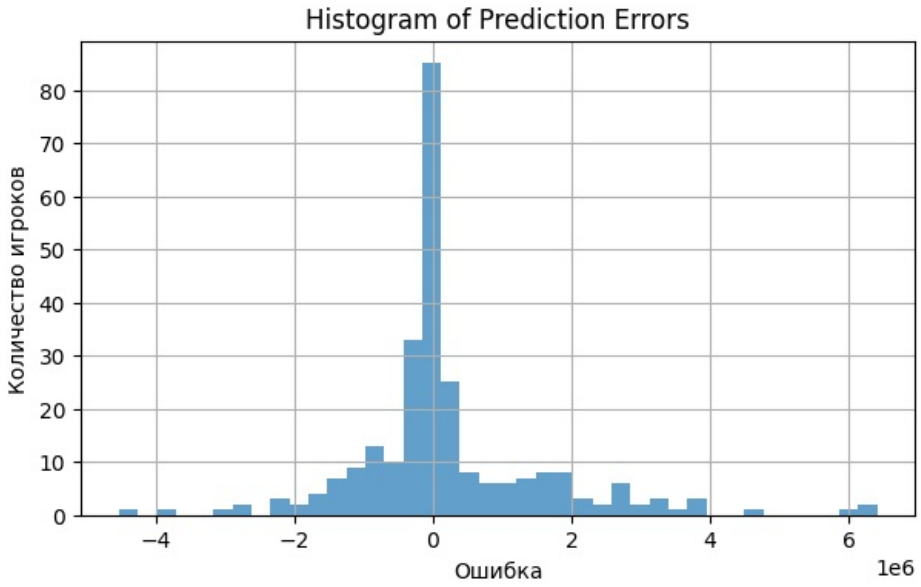
In [14]:
```
important_features(grid.best_estimator_, X_test_i, y_test_i)
true_vs_predicted(y_test_i, y_pred)
```

```
resudial(y_test_i, y_pred)
histogramm_error(y_test_i, y_pred)
```




True vs Predicted Salary
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Предсказанная зарплата
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e7
Истинная зарплата


Residual Plot
1e6
6
4
2
0
−2
−4
Ошибка (y_test - y_pred)
1
2
3
4
5
6
7
1e6
Предсказанная зарплата



Бустинг показал лучшую ключевую метрику, он неплохо справляется как с дорогими, так и с дешёвыми хоккеистами. Единственное, что смущает -- это переобучение.

In [15]:
```python
import pandas as pd
import numpy as np

original_float_format = pd.get_option('display.float_format')

pd.set_option('display.float_format', '{:.2f}'.format)

best_model = grid.best_estimator_

y_full_pred = best_model.predict(X_i)

results_df = pd.DataFrame({
    'True_Salary': y_i,
    'Predicted_Salary': y_full_pred
})

results_df['Absolute_Error'] = np.abs(results_df['True_Salary'] - results_df['Predicted_Salary'])

test_indices = X_test.index
final_results = pd.merge(
    df[['First Name', 'Last Name']],
    results_df,
    left_index=True,
    right_index=True
)

final_results_sorted = final_results.sort_values(by='Absolute_Error', ascending=False)

print("\n--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---")
print(final_results_sorted.head(10).to_string())

pd.set_option('display.float_format', original_float_format)
```

```
--- Топ-10 игроков с наибольшей ошибкой (Train & Test) ---
    First Name  Last Name  True_Salary  Predicted_Salary  Absolute_Error
626       Anze    Kopitar  14000000.00        7588145.65      6411854.35
542       Shea      Weber  12000000.00        5668397.58      6331602.42
208       P.K.     Subban  11000000.00        4943771.39      6056228.61
721       Sean    Monahan   6500000.00        1887067.06      4612932.94
532      Radim     Vrbata   1000000.00        5533651.13      4533651.13
110       Luke     Schenn   1000000.00        4790963.70      3790963.70
620     Clarke  MacArthur   4500000.00         773747.60      3726252.40
497     Andrew       Ladd   8000000.00        4284315.37      3715684.63
787      Derek     Stepan   8000000.00        4320848.47      3679151.53
148   Vladimir  Tarasenko   8000000.00        4399066.98      3600933.02
```

In [16]:
```python
import pandas as pd
final_results['Signed_Error'] = final_results['Predicted_Salary'] - final_results['True_Salary']

overestimated_players = final_results[final_results['Signed_Error'] > 0].copy()

top_overestimated = overestimated_players.sort_values(by='Signed_Error', ascending=False)
```

```
print("\n--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---")

print(top_overestimated[['First Name', 'Last Name',
                         'True_Salary', 'Predicted_Salary', 'Signed_Error']].head(10).to_string())
```

```
--- Топ-10 игроков с наибольшей переоценкой зарплаты (Predicted > True) ---
    First Name Last Name  True_Salary  Predicted_Salary  Signed_Error
532      Radim    Vrbata    1000000.0      5.533651e+06  4.533651e+06
110       Luke    Schenn    1000000.0      4.790964e+06  3.790964e+06
849        Sam    Gagner     650000.0      3.623715e+06  2.973715e+06
584    Vincent  Trocheck    3750000.0      6.619976e+06  2.869976e+06
440     Thomas     Vanek    2600000.0      5.391430e+06  2.791430e+06
286     Victor    Hedman    4250000.0      6.539313e+06  2.289313e+06
412       Mark  Barberio     800000.0      3.021040e+06  2.221040e+06
316   Jonathan    Drouin     832500.0      3.009321e+06  2.176821e+06
302        Tom     Pyatt     800000.0      2.864126e+06  2.064126e+06
326     Martin    Hanzal    3500000.0      5.546232e+06  2.046232e+06
```

Ожидания не оправдались, градиентный бустинг оказался хуже для этой задачи, чем HuberLinearRegression. Последняя меньше склонна к переобучению, даёт лучший MAE, да и в целом проще.

My implementation

In [17]:
```
import numpy as np
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.tree import DecisionTreeRegressor
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted

class MyGradientBoostingRegressor(RegressorMixin, BaseEstimator):
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3, min_samples_leaf=1):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf

    def fit(self, X, y):
        X, y = check_X_y(X, y, ensure_2d=True)
        self.initial_pred_ = np.mean(y)
        current_prediction = np.full(y.shape, self.initial_pred_)
        self.estimators_ = []

        for m in range(self.n_estimators):
            residual = y - current_prediction
            tree = DecisionTreeRegressor(
                max_depth=self.max_depth,
                min_samples_leaf=self.min_samples_leaf
            )
            tree.fit(X, residual)

            tree_predictions = tree.predict(X)

            current_prediction += self.learning_rate * tree_predictions
            self.estimators_.append(tree)

        return self

    def predict(self, X):
        check_is_fitted(self)
        X = check_array(X)

        final_prediction = np.full(X.shape[0], self.initial_pred_)
        for tree in self.estimators_:
            tree_predictions = tree.predict(X)
            final_prediction += self.learning_rate * tree_predictions
        return final_prediction
```

In [18]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import seaborn as sns

model = GradientBoostingRegressor()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
base_mean_error = mean_absolute_error(y_test, [y_train.mean()]*len(y_test))

print(f"--- My gradient boosting (Raw Data) ---")
print(f"MAE (Ошибка в долларах): {mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
--- My gradient boosting (Raw Data) ---
MAE (Ошибка в долларах): 959400.336
R2 Score: 0.603
```

In [19]:
```python
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.compose import TransformedTargetRegressor
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

full_pipeline = Pipeline([
    ('preprocessor', ct),
    ('model_wrapper', TransformedTargetRegressor(
        regressor=MyGradientBoostingRegressor(),
        func=np.log1p,
        inverse_func=np.expm1
    ))
])

param_grid = {
    'model_wrapper__regressor__n_estimators': [500],
    'model_wrapper__regressor__max_depth': [3, 5, 7],
    'model_wrapper__regressor__learning_rate': [0.005],
}

scoring = {
    'MAE': 'neg_mean_absolute_error',
    'R2': 'r2'
}
grid = RandomizedSearchCV(
    full_pipeline,
    param_grid,
    cv=3,
    scoring=scoring,
    refit='MAE',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

y_pred = grid.best_estimator_.predict(X_test_i)
y_pred_train = grid.best_estimator_.predict(X_train_i)

mae = mean_absolute_error(y_test_i, y_pred)
train_mae = mean_absolute_error(y_train, y_pred_train)
rmse = np.sqrt(mean_squared_error(y_test_i, y_pred))
r2 = r2_score(y_test_i, y_pred)
base_mean_error = mean_absolute_error(y_test_i, [y_train.mean()] * len(y_test_i))

print(f"\n--- Gradient Boosting Regressor ---")
print(f"MAE: {mae:.3f}")
print(f"Train MAE: {train_mae:.3f}")
print(f"R2 Score: {r2:.3f}")
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
/Users/zloyaloha/development/ai-frameworks/.venv_ai/lib/python3.11/site-packages/sklearn/model_selection/_search.py:317: UserWarning: The total space of parameters 3 is smaller than n_iter=10. Running 3 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
Лучшие параметры: {'model_wrapper__regressor__n_estimators': 500, 'model_wrapper__regressor__max_depth': 3, 'model_wrapper__regressor__learning_rate': 0.005}

--- Gradient Boosting Regressor ---
MAE: 915465.798
Train MAE: 719921.153
R2 Score: 0.570
```

Собственная имплементация градиентного бустинга показала чуть более плохой скор, чем из scikit-learn. Собственная имплементация не учитывает некоторые гиперпараметры, с которыми работает библиотечная модель, что может влиять на результат. В целом разница в 15 тысяч -- это в пределах погрешности.

	Base Gradient Boosting	Gradient Boosting	My Random Forest
MAE	963131.492	850100.263	915546.804
R2	0.604	0.560	0.569

Классификация

Задача: вычислить мошенника на страховых выплатах с использованием модели случайного леса

Для выполнения лабораторной работы были выбраны метрики F1-score и ROC-AUC, так как исследуемый датасет является несбалансированным. Метрика Accuracy в данном случае неинформативна, так как модель, предсказывающая всем класс ‘0’ (не фрод), может иметь высокую Accuracy, но будет бесполезна. F1-score позволит контролировать баланс между ложными срабатываниями и пропуском мошенников.

Baseline

In [20]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                "buntyshah/auto-insurance-claims-data/versions/1",
                                "insurance_claims.csv")
df
```

Out[20]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 40 columns

In [21]:
```
df_clean = df.copy()

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
df_clean["property_damage"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0}).fillna(-1)
```

In [22]:
```
df_clean = df_clean.drop(columns=["_c39"])
```

In [23]:
```
date_cols = ["policy_bind_date", "incident_date"]
for c in date_cols:
    df_clean[c] = pd.to_datetime(df_clean[c], errors="coerce")
```

In [24]:
```
for col in df_clean.select_dtypes(include=['object']).columns:
    df_clean[col] = df_clean[col].astype('category').cat.codes
df_clean.head()
```

Out[24]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium
0	328	48	521585	2014-10-17	2	1	1000	1406.91
1	228	42	342868	2006-06-27	1	1	2000	1197.22
2	134	29	687698	2000-09-06	2	0	2000	1413.14
3	256	41	227811	1990-05-25	0	1	2000	1415.74
4	228	44	367455	2014-06-06	0	2	1000	1583.91

5 rows × 39 columns

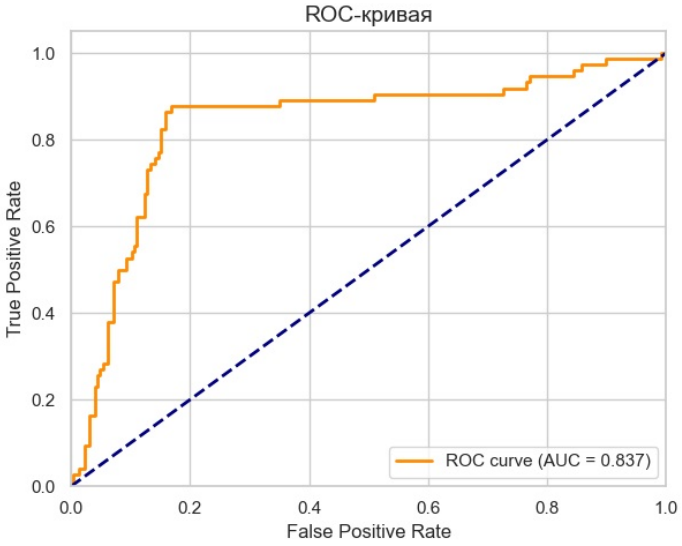
In [25]:
```
from sklearn.model_selection import train_test_split
drop_dates = ["policy_bind_date", "incident_date"]
df_clean = df_clean.drop(drop_dates, axis=1)
X = df_clean.drop(TARGET_NAME, axis=1)
y = df_clean[TARGET_NAME]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

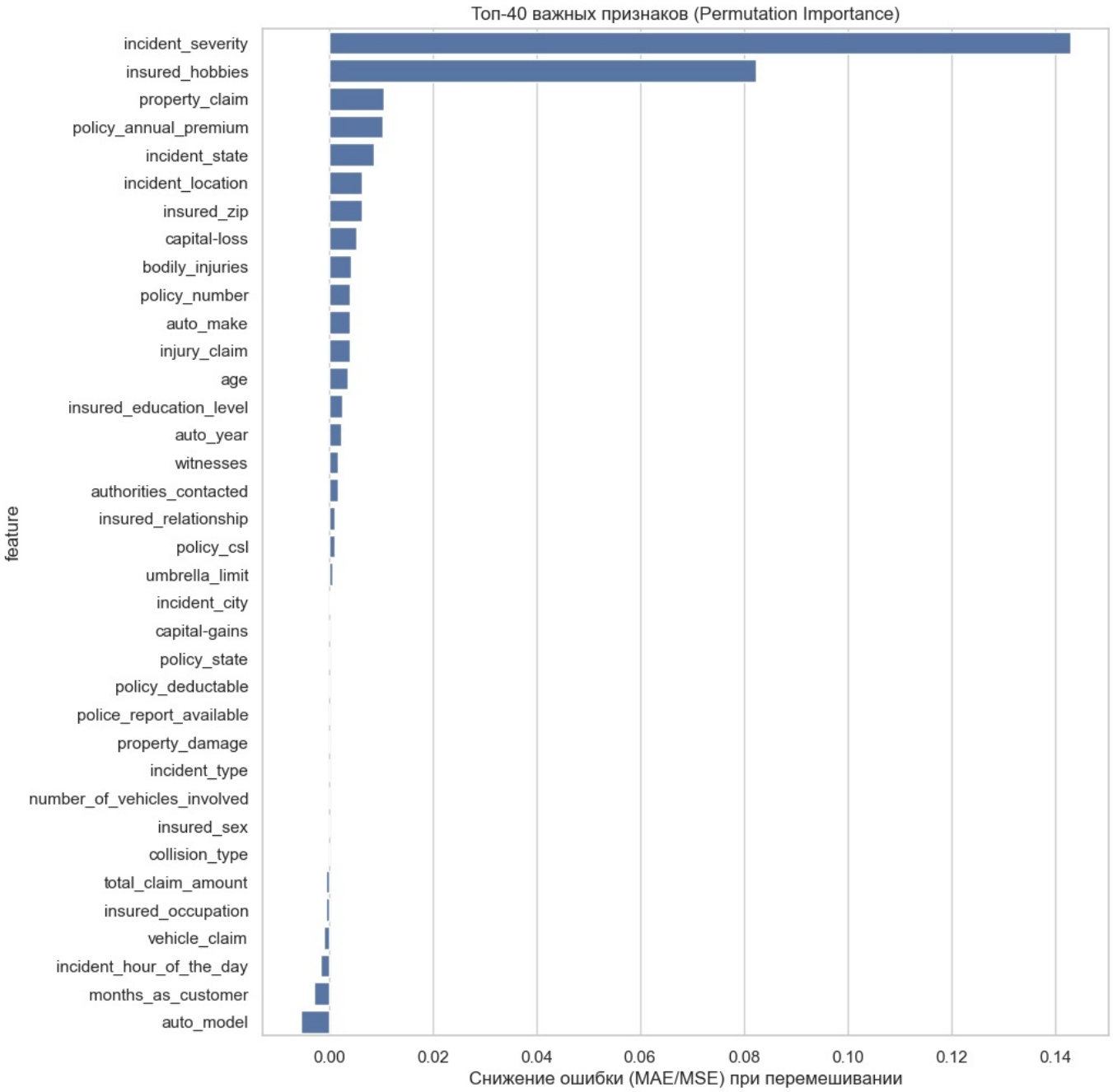
In [26]:
```
from sklearn.metrics import classification_report, f1_score, roc_auc_score, confusion_matrix, roc_curve
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]

f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
F1-score (класс 1): 0.6579
ROC-AUC: 0.8368
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       226
           1       0.64      0.68      0.66        74

    accuracy                           0.83       300
   macro avg       0.77      0.78      0.77       300
weighted avg       0.83      0.83      0.83       300

==================================================
Confusion Matrix:
[[198  28]
 [ 24  50]]
```

In [27]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc
def graphics(y_test, y_pred, y_prob):
    sns.set(style="whitegrid")
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", cbar=False,
                xticklabels=['Предск: 0', 'Предск: 1'],
                yticklabels=['Факт: 0', 'Факт: 1'])
    plt.title("Матрица ошибок", fontsize=14)
    plt.ylabel("Реальность")
    plt.xlabel("Предсказание")

    plt.subplot(1, 2, 2)
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC-кривая', fontsize=14)
    plt.legend(loc="lower right")

    plt.tight_layout()
    plt.show()
def feature_important(model, X_test, y_test):

    perm_result = permutation_importance(
        model,
        X_test,
        y_test,
        n_repeats=10,
        random_state=42,
        n_jobs=-1
    )
```

```
    perm_df = pd.DataFrame({
        'feature': X_test.columns,
        'importance': perm_result.importances_mean
    })

    top_40_perm = perm_df.sort_values(by='importance', ascending=False).head(40)

    plt.figure(figsize=(10, 12))
    sns.barplot(data=top_40_perm, x='importance', y='feature')
    plt.title("Топ-40 важных признаков (Permutation Importance)")
    plt.xlabel("Снижение ошибки (MAE/MSE) при перемешивании")
    plt.show()
graphics(y_test, y_pred, y_prob)
feature_important(model, X_test, y_test)
```

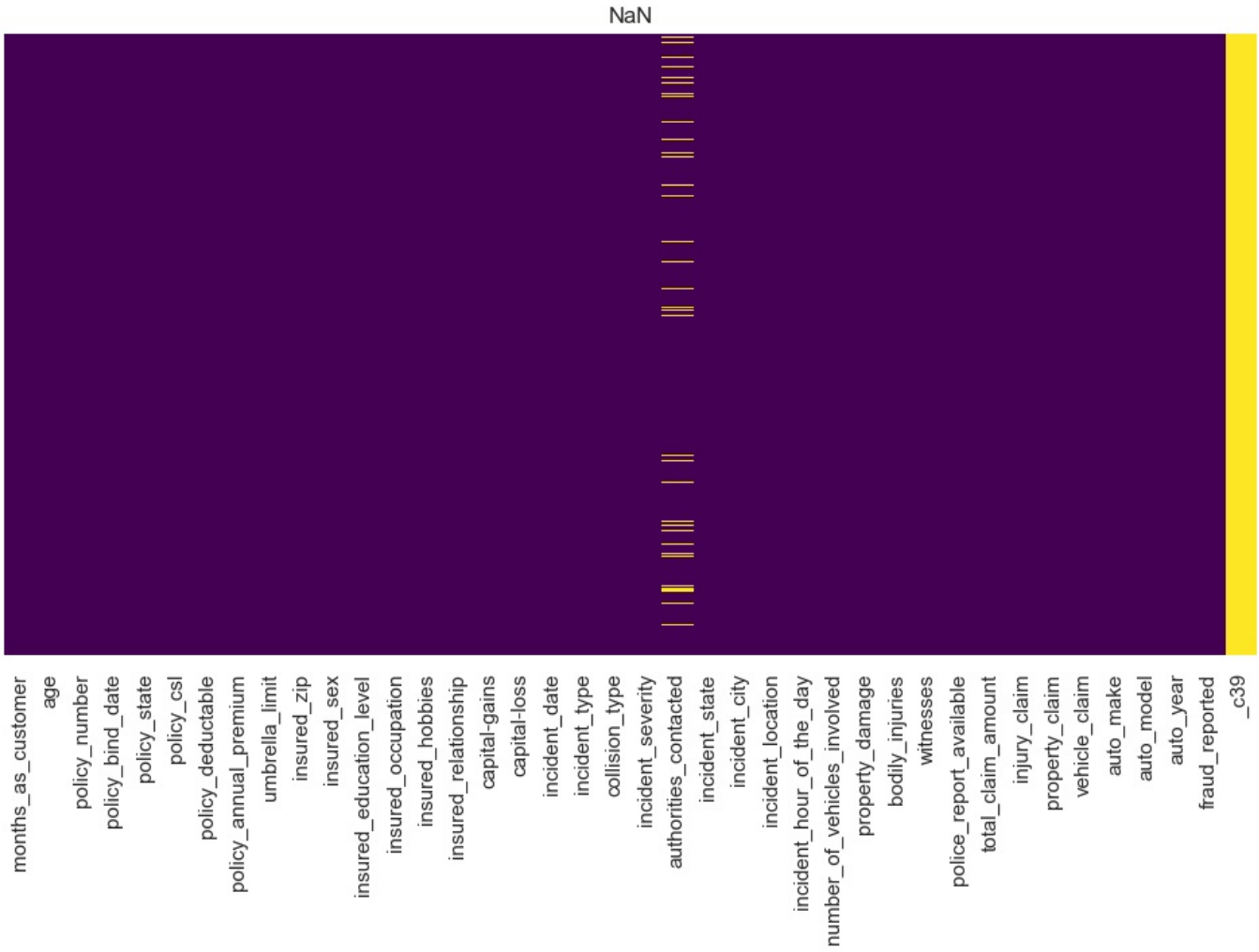




Бустинг нашёл больше мошенников. Сработал лучше, нашёл больше мошенников даже без предобработки данных.

Improved Gradien Boosting

In [28]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

df = kagglehub.dataset_load(KaggleDatasetAdapter.PANDAS,
                                    "buntyshah/auto-insurance-claims-data/versions/1",
                                    "insurance_claims.csv")
df.head()

nulls = df.isna().sum().sort_values(ascending=False)
null_pct = (nulls / len(df)).mul(100).round(2)

plt.figure(figsize=(12,6))
sns.heatmap(df.isna(), cbar=False, yticklabels=False, cmap="viridis")
plt.title("NaN")
plt.show()
```



In [29]:
```
df_clean = df.copy()
df_clean["fraud_reported"].value_counts()
```

Out[29]:
```
fraud_reported
N    753
Y    247
Name: count, dtype: int64
```

In [30]:
```
display(df_clean["police_report_available"].unique())
display(df_clean["property_damage"].unique())
```

```
array(['YES', '?', 'NO'], dtype=object)
array(['YES', '?', 'NO'], dtype=object)
```

In [31]:
```
df_clean = df_clean.drop(columns=["_c39"])

df_clean["authorities_contacted"] = df_clean["authorities_contacted"].fillna("No Contact")

TARGET_NAME = "fraud_reported"
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({'Y': 1, 'N': 0})
df_clean["police_report_available"] = df_clean["police_report_available"].map({'YES': 1, 'NO': 0, '?': -1})
df_clean["property_damage"] = df_clean["property_damage"].map({'YES': 1, 'NO': 0, '?': -1})

dates_cols = ["policy_bind_date", "incident_date"]
for c in date_cols:
    df_clean[c] = pd.to_datetime(df_clean[c])

df_clean
```

Out[31]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 39 columns

In [32]:
```
num_cols = df_clean.select_dtypes(include=["int64", "float64"]).columns.tolist()
df_clean[num_cols].describe().T
```

Out[32]:

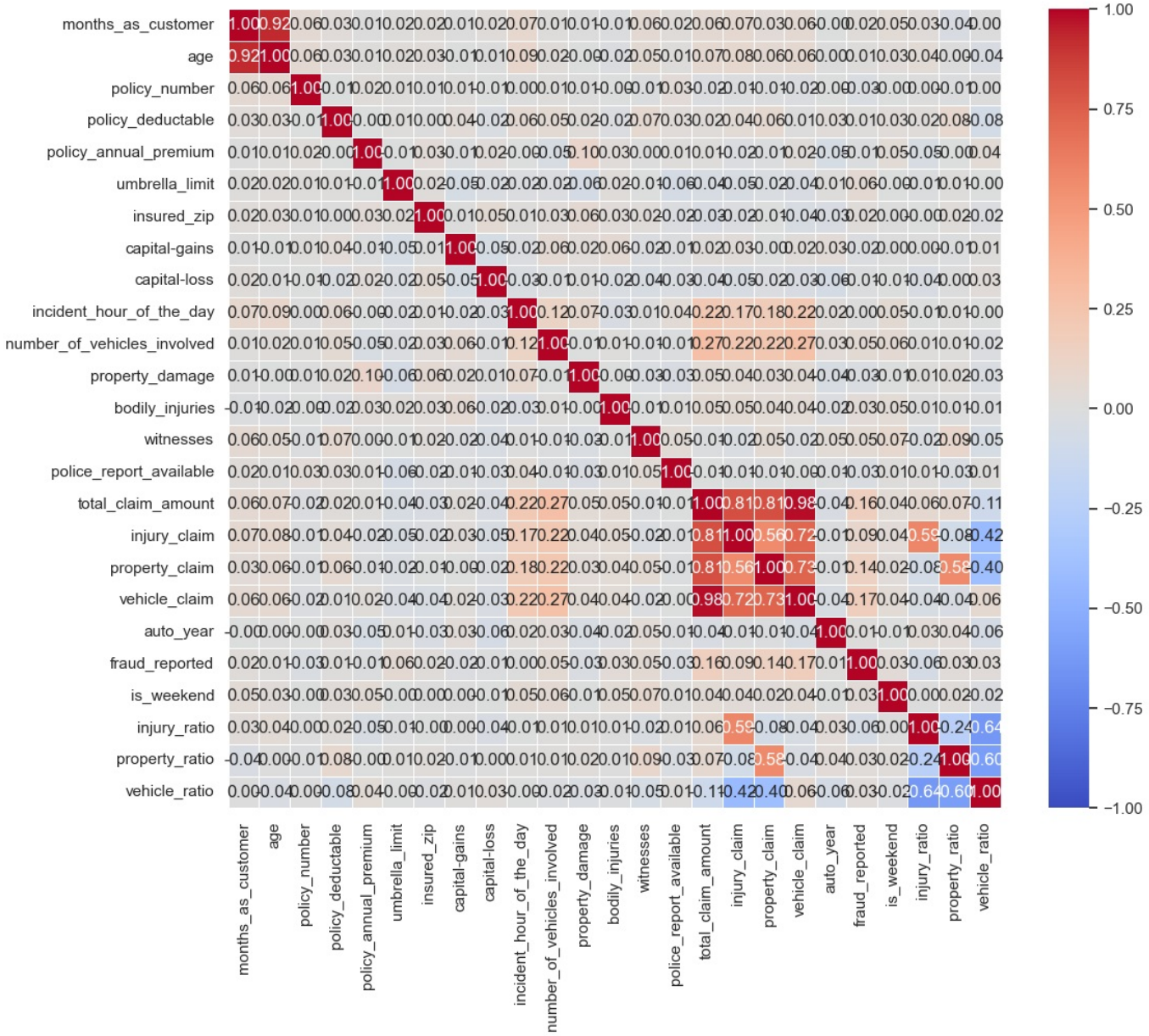
	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
property_damage	1000.0	-5.800000e-02	8.119700e-01	-1.00	-1.0000	0.0	1.000	1.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
police_report_available	1000.0	-2.900000e-02	8.104417e-01	-1.00	-1.0000	0.0	1.000	1.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00
fraud_reported	1000.0	2.470000e-01	4.314825e-01	0.00	0.0000	0.0	0.000	1.00

In [33]:
```
import numpy as np

df_pr = df_clean.copy()

median_value = df_pr.loc[df_pr['umbrella_limit'] != -100000, 'umbrella_limit'].median()
df_pr.loc[df_pr['umbrella_limit'] == -100000, 'umbrella_limit'] = median_value
```

In [34]:
```
df_features = df_pr.copy()

df_clean["policy_tenure_months"] = ((df_clean["incident_date"] - df_clean["policy_bind_date"]).dt.days / 30).as

df_features["incident_year"] = df_features["incident_date"].dt.year
df_features["incident_month"] = df_features["incident_date"].dt.month
df_features["incident_dow"] = df_features["incident_date"].dt.dayofweek
df_features["is_weekend"] = df_features["incident_dow"].isin([5, 6]).astype(int)

df_features["injury_ratio"] = df_features["injury_claim"] / (df_features["total_claim_amount"] + 1e-3)
df_features["property_ratio"] = df_features["property_claim"]  / (df_features["total_claim_amount"] + 1e-3)
```

```
df_features["vehicle_ratio"] = df_features["vehicle_claim"] / (df_features["total_claim_amount"] + 1e-3)

drop_dates = ["policy_bind_date", "incident_date"]
df_features = df_features.drop(drop_dates, axis=1)
```

In [35]:
```
num_cols = df_features.select_dtypes(include=["int64", "float64"]).columns.tolist()

plt.figure(figsize=(12, 10))
correlation_matrix = df_features[num_cols].corr()

sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            vmin=-1, vmax=1,
            linewidths=0.5)

plt.show()
```



In [36]:
```
from sklearn.model_selection import train_test_split
X_i = df_features.drop(TARGET_NAME, axis=1)
y_i = df_features[TARGET_NAME]
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_i, y_i, test_size=0.25, random_state=42, stratify
```

Я пробовал и модель и scikit-learn, и xgboost, но последняя показала лучшие результаты. В пайплайне уже по традиции отсутствует нормализация данных, кроме целевой переменной, которая логарифмируется. Также тут я попробовал использовать внутренние инструменты XGBoost, чтобы учесть дисбаланс классов.

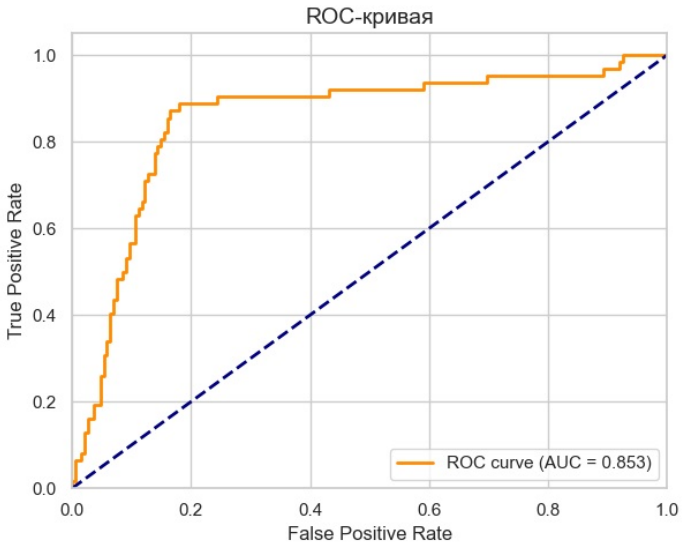
In [37]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import f1_score, roc_auc_score, classification_report
from xgboost import XGBClassifier
```

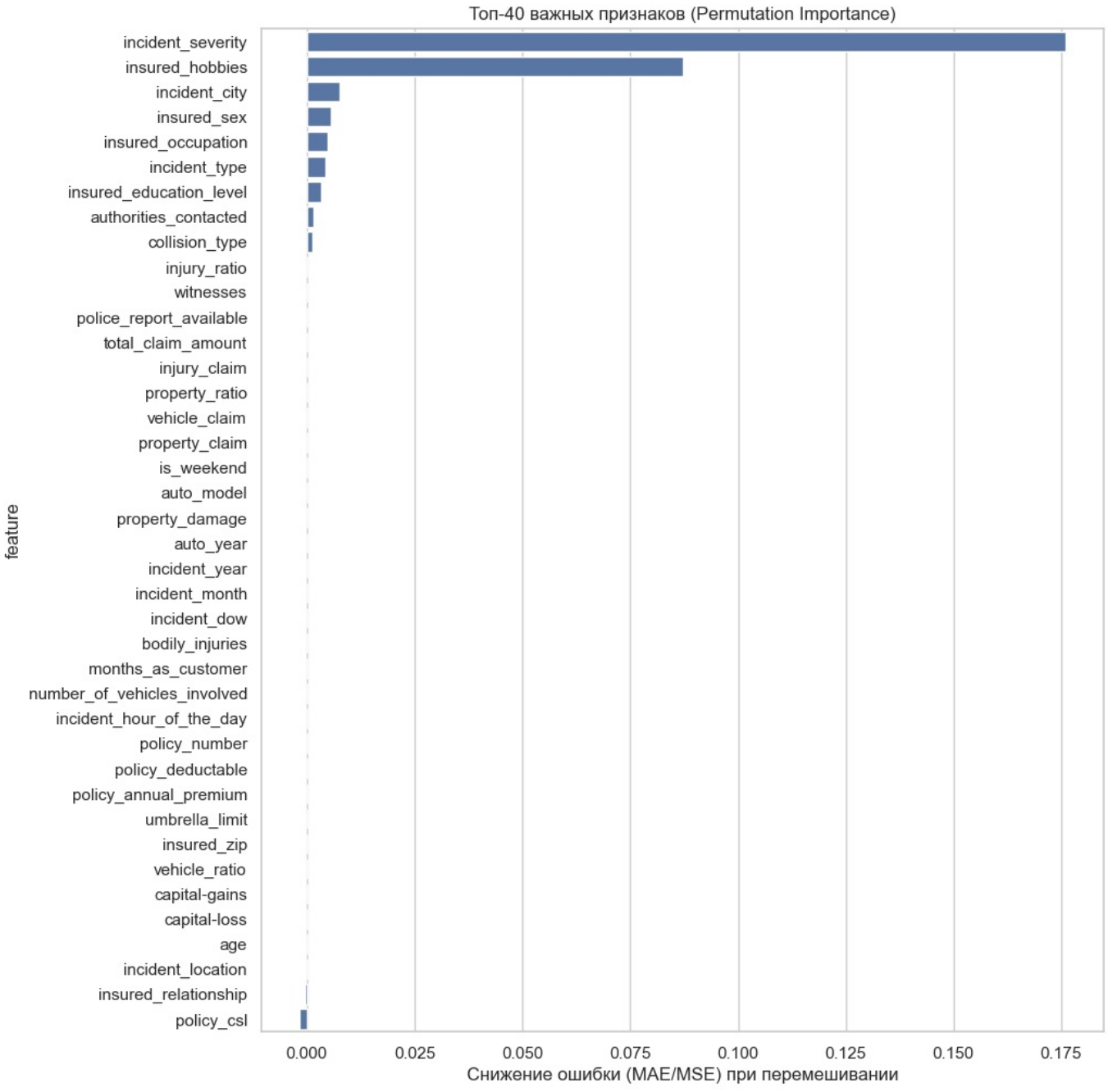
```
from collections import Counter

num_cols = X_train_i.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = X_train_i.select_dtypes(include=["object", "category"]).columns.tolist()

categorical_pipe = Pipeline([
    ("onehot", OneHotEncoder(handle_unknown='infrequent_if_exist', sparse_output=False, min_frequency=2))
])

ct = ColumnTransformer([
    ("cat", categorical_pipe, cat_cols)
])

counter = Counter(y_train)
scale_pos_weight = counter[0] / counter[1]

model_pipe = Pipeline([
    ('ct', ct),
    ('model', XGBClassifier(
        random_state=42,
        eval_metric='logloss',
        scale_pos_weight=scale_pos_weight
    ))
])

param_grid = {
    'model__n_estimators': [100, 300],
    'model__max_depth': [3, 5, 7],
    'model__learning_rate': [0.05, 0.1],
    'model__subsample': [0.7, 0.8, 1.0],
    'model__colsample_bytree': [0.6, 0.8, 1.0],
    'model__gamma': [0, 1, 5],
    'model__reg_alpha': [0, 0.1, 1],
    'model__reg_lambda': [1, 1.5, 2],
}

cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

grid = RandomizedSearchCV(
    model_pipe,
    param_grid,
    cv=cv,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

best_model = grid.best_estimator_
y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

print("-"*30)
print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-"*30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Лучшие параметры: {'model__subsample': 0.8, 'model__reg_lambda': 1, 'model__reg_alpha': 1, 'model__n_estimators'
: 300, 'model__max_depth': 5, 'model__learning_rate': 0.05, 'model__gamma': 1, 'model__colsample_bytree': 1.0}
------------------------------
F1-score (класс 1): 0.7101
ROC-AUC: 0.8525
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.86      0.89       188
           1       0.64      0.79      0.71        62

    accuracy                           0.84       250
   macro avg       0.79      0.82      0.80       250
weighted avg       0.86      0.84      0.85       250
```

In [38]:
```
import matplotlib.pyplot as plt
```

```
import pandas as pd
from sklearn.inspection import permutation_importance

graphics(y_test_i, y_pred, y_prob)
feature_important(grid.best_estimator_, X_test_i, y_test_i)
```





Улучшенный бустинг определил больше мошенников, чем случайный лес, нашёл более сложные зависимости по признакам, однако это всё еще хуже, чем обычное дерево. Видимо зависимости, что оно находит, не влияют на количество мошенников, но только увеличивают шанс быть честным человеком. "Грубое" решающее дерево таким не страдает, поэтому F1 на класс

мошенников у него выше.

My implementation

In [39]:
```python
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.tree import DecisionTreeRegressor
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
from sklearn.utils.multiclass import unique_labels

class MyGradientBoostingClassifier(ClassifierMixin, BaseEstimator):
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3, min_samples_leaf=1):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
        self.estimators = []
        self.initial_pred = None

    def _sigmoid(self, logit):
        logit = np.clip(logit, -15, 15)
        return 1 / (1 + np.exp(-logit))

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        self.classes_ = unique_labels(y)
        if len(self.classes_) > 2:
            raise ValueError("This implementation supports binary classification only.")
        p_initial = np.mean(y)
        p_initial = np.clip(p_initial, 1e-15, 1 - 1e-15)
        self.initial_pred = np.log(p_initial / (1 - p_initial))
        current_logits = np.full(y.shape, self.initial_pred)
        self.estimators = []

        for m in range(self.n_estimators):
            current_probabilities = self._sigmoid(current_logits)
            residual = y - current_probabilities
            tree = DecisionTreeRegressor(
                max_depth=self.max_depth,
                min_samples_leaf=self.min_samples_leaf,
                random_state=42
            )
            tree.fit(X, residual)

            tree_predictions = tree.predict(X)
            current_logits += self.learning_rate * tree_predictions
            self.estimators.append(tree)

        return self

    def predict_proba(self, X):
        check_is_fitted(self)
        X = check_array(X)
        final_logits = np.full(X.shape[0], self.initial_pred)
        for tree in self.estimators:
            tree_predictions = tree.predict(X)
            final_logits += self.learning_rate * tree_predictions

        proba_class_1 = self._sigmoid(final_logits)
        proba_class_0 = 1 - proba_class_1
        return np.vstack((proba_class_0, proba_class_1)).T

    def predict(self, X):
        probas = self.predict_proba(X)[:, 1]
        return np.array([1 if p > 0.5 else 0 for p in probas])
```

In [40]:
```python
model = MyGradientBoostingClassifier()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]

f1 = f1_score(y_test, y_pred)
roc = roc_auc_score(y_test, y_prob)

results = {'F1-score': f1, 'ROC-AUC': roc}

print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-" * 30)
```

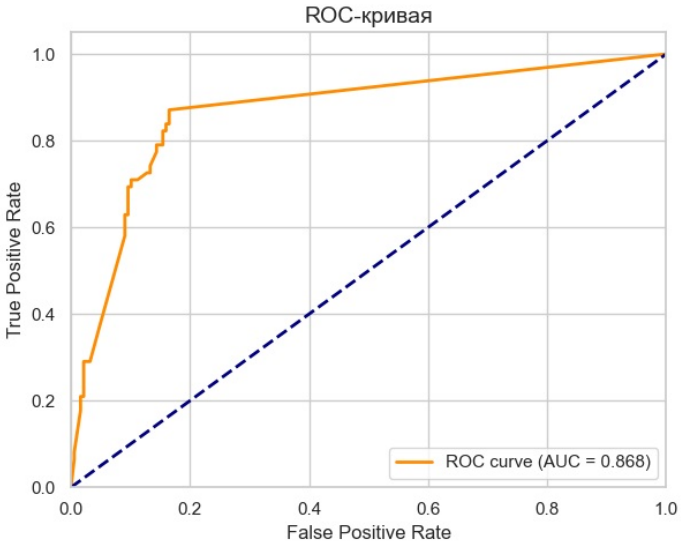
```
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("="*50)

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

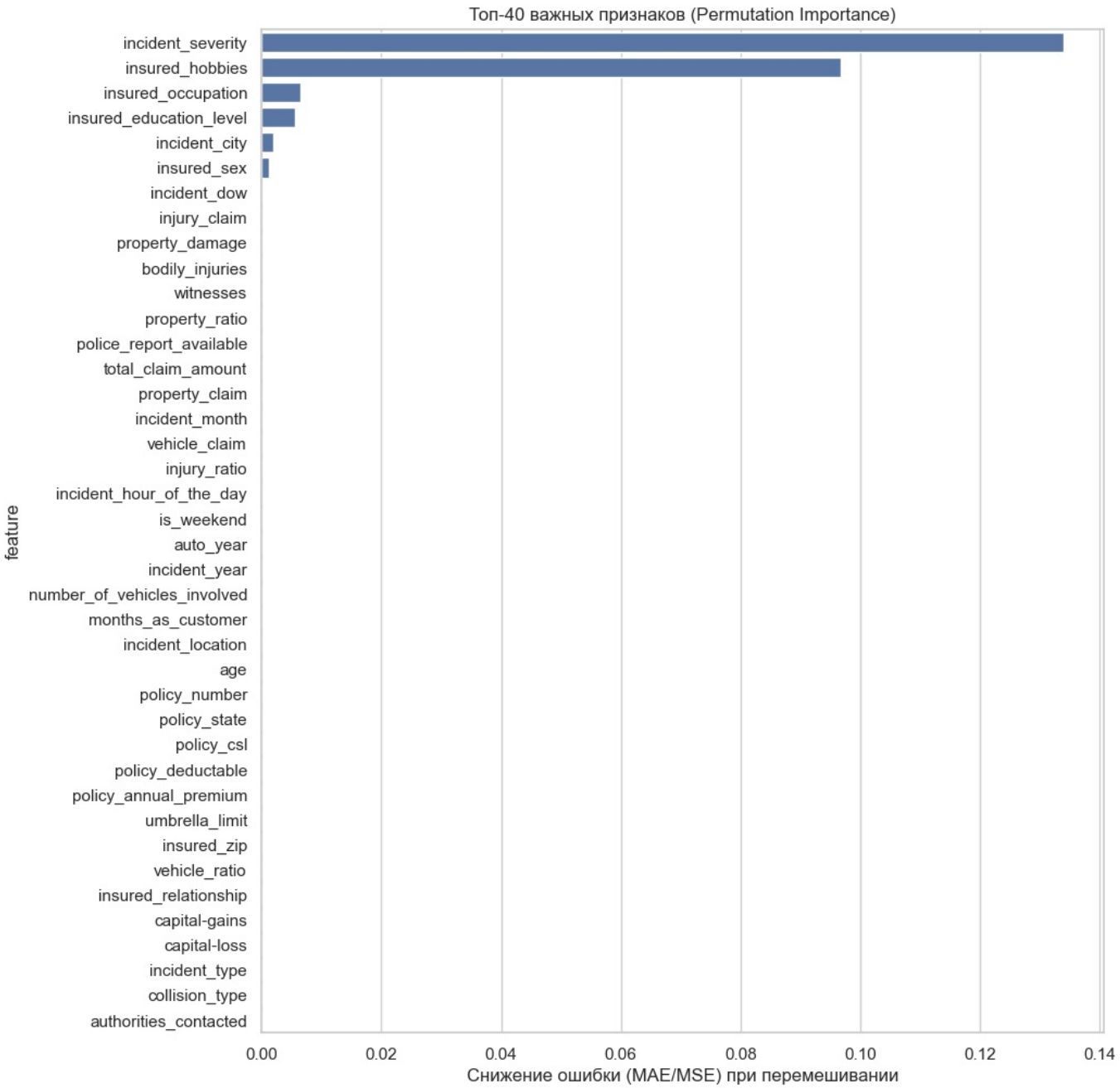
```
F1-score (класс 1): 0.6790
ROC-AUC: 0.8565
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       226
           1       0.62      0.74      0.68        74

    accuracy                           0.83       300
   macro avg       0.77      0.80      0.78       300
weighted avg       0.84      0.83      0.83       300

==================================================
Confusion Matrix:
[[193  33]
 [ 19  55]]
```

In [41]:
```
model_pipe = Pipeline([
    ('ct', ct),
    ('model', MyGradientBoostingClassifier())
])

param_grid = {
    'model__n_estimators': [100, 300],
    'model__max_depth': [3, 5, 7],
    'model__learning_rate': [0.05, 0.1],
}

cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

grid = RandomizedSearchCV(
    model_pipe,
    param_grid,
    cv=cv,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_i, y_train_i)

print("Лучшие параметры:", grid.best_params_)

best_model = grid.best_estimator_
y_pred = best_model.predict(X_test_i)
y_prob = best_model.predict_proba(X_test_i)[:, 1]

f1 = f1_score(y_test_i, y_pred)
roc = roc_auc_score(y_test_i, y_prob)

print("-"*30)
print(f"F1-score (класс 1): {f1:.4f}")
print(f"ROC-AUC: {roc:.4f}")
print("-"*30)
print("Classification Report:")
print(classification_report(y_test_i, y_pred))
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Лучшие параметры: {'model__n_estimators': 100, 'model__max_depth': 3, 'model__learning_rate': 0.05}
------------------------------
F1-score (класс 1): 0.6261
ROC-AUC: 0.8677
------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       188
           1       0.68      0.58      0.63        62

    accuracy                           0.83       250
   macro avg       0.77      0.75      0.76       250
weighted avg       0.82      0.83      0.82       250
```

In [42]:
```
graphics(y_test_i, y_pred, y_prob)
feature_important(grid.best_estimator_, X_test_i, y_test_i)
```





Собственная реализация градиентного бустинга выдаёт примерно такие результаты, что и версия из XGBoost

	Base Random Forest	Random Forest	My Random Forest
ROC_AUC	0.8380	0.8497	0.8677
F1 (1 класс)	0.88	0.88	0.89
F1 (2 класс)	0.65	0.67	0.63

Вывод

При выполнении этой лабораторной работы я понял, что самая мощная модель не значит, что она будет самая лучшая. Для обоих датасетов и задач менее продвинутые и сложные модели показали лучший результат.

Для задачи классификации бустинг еще показывал результат примерно в рамках погрешности, то для регресси он не смог достаточно точно научиться предсказывать зарплату игроков, которая гораздно выше среднего.

Отчёт

Классификация

Базовые модели зачастую страдают от того, что классы в задаче несбалансированны: для многих моделей F1 метрика второго класса сильно просаживается. Безоговорочно лучшей моделью оказалось решающее дерево, поскольку по сумме метрик оно выдало наилучший результат. Кроме того, она выдала наилучший результат по метрике F1 для второго класса, что является ключевым в задаче. Если нужно быстрое бизнес решение с приемлимыми метриками, я бы использовал градиентный бустинг, который даже при отстутствии тюнинга выдал приличные результаты.

	ROC_AUC	F1 (1 класс)	F1 (2 класс)
Baseline KNN	0.4674	0.81	0.06
Improved KNN	0.8545	0.86	0.67
My KNN	0.8545	0.86	0.67
Baseline Linear Classifier	0.6118	0.86	0.00
Linear Classifier	0.8513	0.88	0.62
My Linear Classifier	0.8542	0.89	0.71
Baseline Decision Tree	0.7447	0.86	0.61
Decision Tree	0.8611	0.89	0.73
My Decision Tree	0.8781	0.88	0.62
Baseline Random Forest	0.8398	0.85	0.39
Random Forest	0.8314	0.88	0.71
My Random Forest	0.8515	0.88	0.59
Baseline Gradient Boosting	0.8380	0.88	0.65
Gradient Boosting	0.8497	0.88	0.67
My Gradient Boosting	0.8677	0.89	0.63

Регрессия

По метрике R^2 лидирует Linear Regression (0.645), совсем немного обгоняя Gradient Boosting (0.640) и Random Forest (0.641). По метрике MAE выигрывает Gradient Boosting (850100) и Random Forest (851915), что говорит о том, что в среднем он ошибается меньше, хотя линейная регрессия лучше объясняет общую дисперсию данных.

Линейная регрессия показала себя неожиданно хорошо. Это может указывать на то, что зависимость в данных имеет преимущественно линейный характер, либо данных недостаточно для раскрытия потенциала сложных ансамблей.

KNN показал худшие результаты ($R^2 < 0.48$). Для задач регрессии на данных высокой размерности этот алгоритм часто неэффективен.

Разница между Base моделями и настроенными огромна. Например, для базовой линейной регрессии имеем R^2 0.324, а для настроенной — 0.644.

Лучше всего при отсутствии тюнинга показал себя градиентный бустинг, который я бы рекомендовал использовать при нежелании тратить время на тюнинг.

	MAE	R2
Base KNN	1173197.566	0.399
KNN	1069096.869	0.492
My KNN	1069096.869	0.492
Base Linear Regression	1381895.059	0.324
Linear Regression	874617.127	0.645

	MAE	R2
My Linear Regression	892750.102	0.622
Base Decision Tree	1220967.395	0.322
Decision Tree	865388.517	0.635
My Decision Tree	870330.527	0.632
Base Random Forest	973296.880	0.604
Random Forest	857029.069	0.638
My Random Forest	1104519.680	0.446
Base Gradient Boosting	963131.492	0.604
Gradient Boosting	850100.263	0.640
My gradient boosting	915417.103	0.570

Вывод

Для Регрессии: лучшими моделями оказались градиентный бустинг или линейную регрессию.

Для Классификации: Для классификации я бы рекомендовал использовать дерево решений.

Собственные реализации моделей показали близкое качество к библиотечным версиям. Этот анализ демонстрирует, как важно учитывать структуру данных, их распределение, наличие выбросов и дисбаланс классов при выборе подходящего алгоритма. Не существует одной универсальной модели, подходящей для обеих задач, для обоих датасетов.