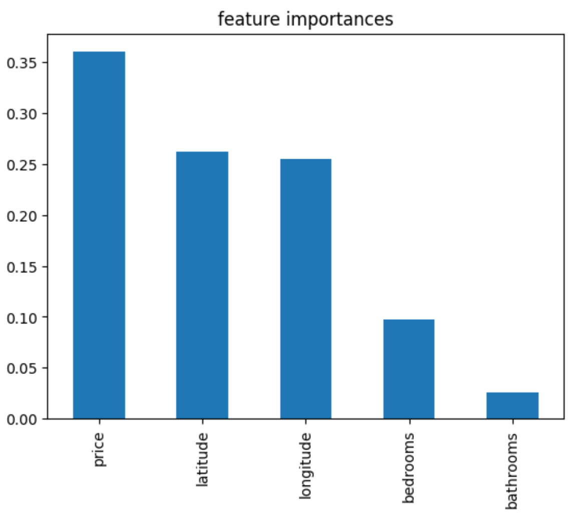
Отчет.

В ходе работы я провел эксперименты с feature importance и добавил все в библиотеку на python. Сравнил feature importance в реализации sklearn и R, а также сравнил sklearn с R-ranger.

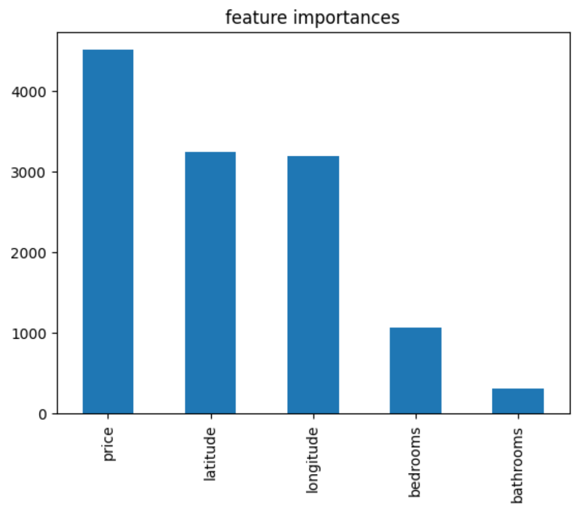
Эксперименты.

Использовал я датасет *rent.csv* из статьи, рассмотрел случаи, когда признаки: 1) разнородны, 2) присутствуют коррелированые признаки, 3) присутствуют нерелевантные признаки.

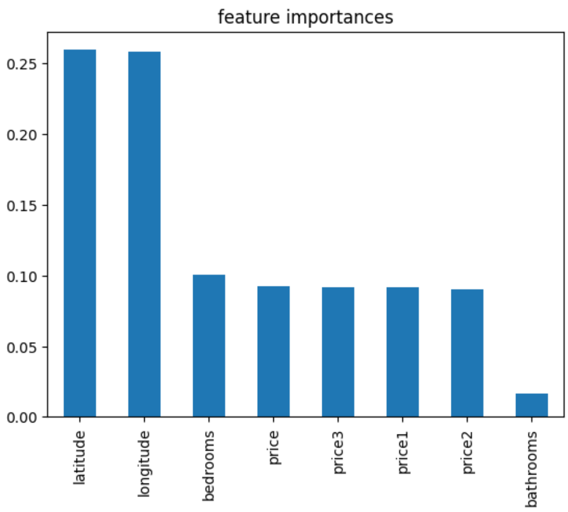
Подбор оптимальных гиперпараметров в реализации sklearn я осуществлял с помощью *Optuna*, и полученные параметры я использовал в реализации на R.



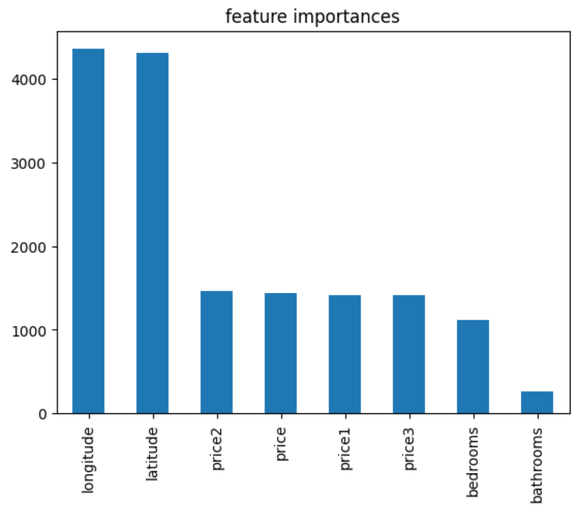
(a) Python: Исходный датасет



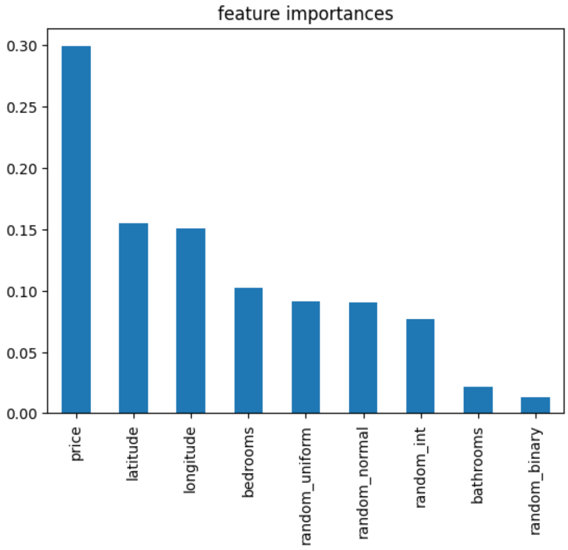
(a) R: Исходный датасет



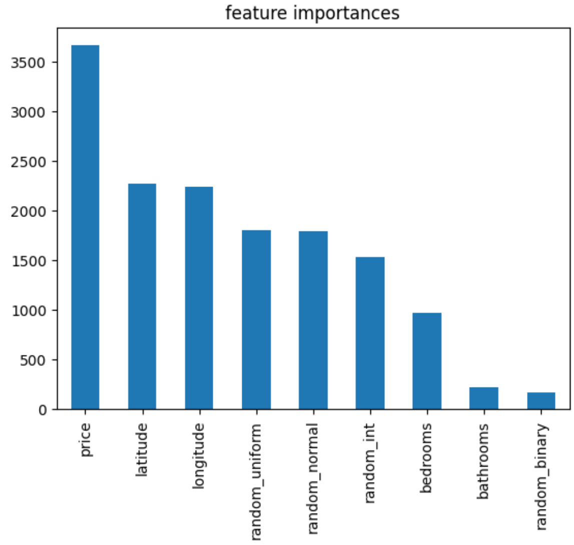
(b) Python: добавлены коррелированные признаки



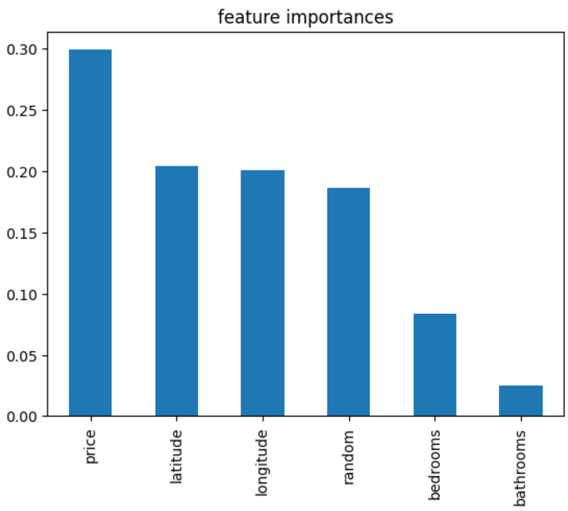
(b) R: добавлены коррелированные признаки



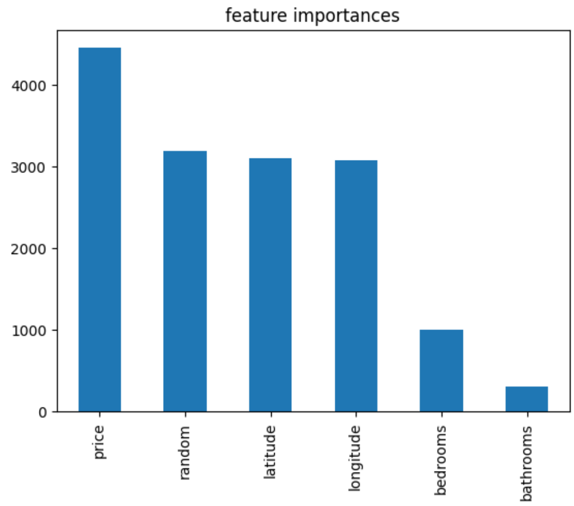
(c) Python: много нерелевантных признаков



(c) R: много нерелевантных признаков



(d) Python: 1 нерелевантный признак



(d) R: 1 нерелевантный признак

Как можно заметить, результаты в некоторых экспериментах неожиданные.

В примере (b) я скопировал признак $price$ 3 раза, и получилось, что важность этого признака уменьшилась. Вообще, при добавлении коррелированных признаков, важности этих признаков будут уменьшаться.

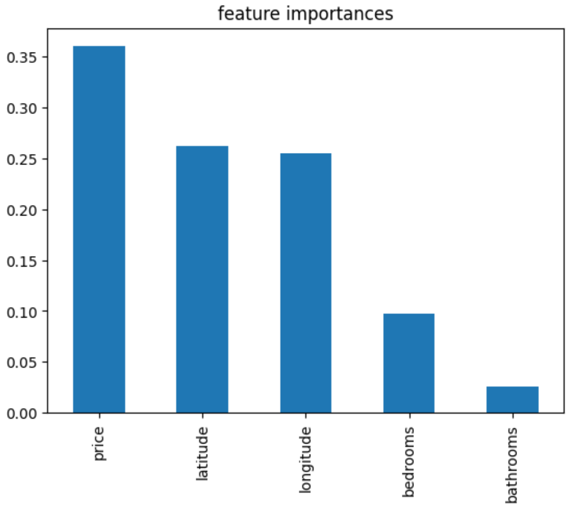
В примерах (c) и (d) я создал несколько признаков, которые никак не связаны с target.

Возникают вопросы, глядя на рисунки с нерелевантными признаками разного типа: 1) Почему эти признаки имеют важность? 2) Почему признаки $random\_uniform$, $random\_normal$, $random\_int$ имеют важность больше, чем признак $random\_binary$?

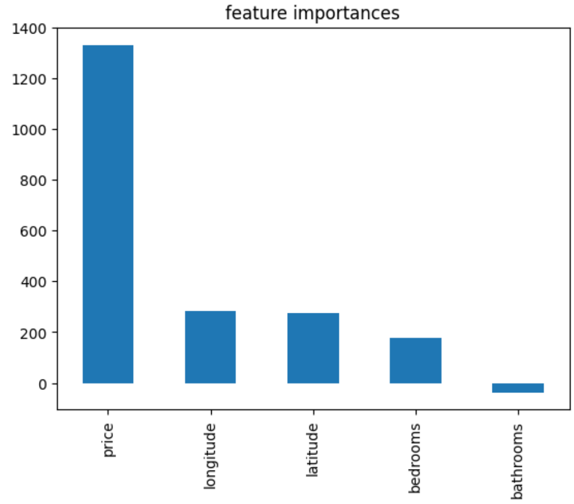
Ниже приведена таблица с оптимальными параметрами для каждого из примеров (a)-(d):

Пример	$n\_estimators$	max_depth	$min\_samples\_leaf$	$max\_features$
(a)	1760	26	5	sqrt
(b)	1289	14	3	1.0
(c)	1064	24	13	0.5
(d)	1096	19	3	0.3333333333333333

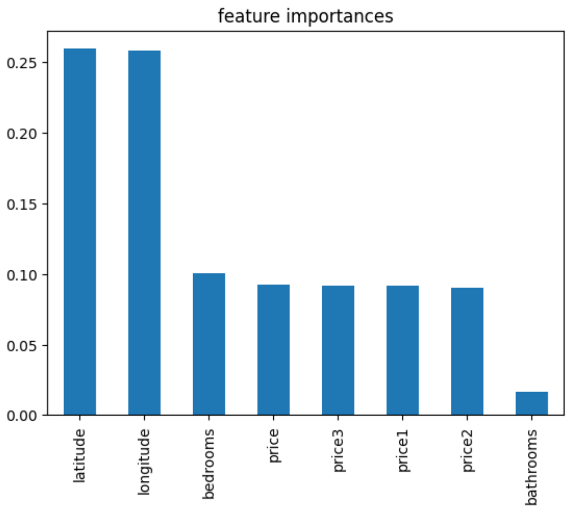
Sklearn vs R-ranger



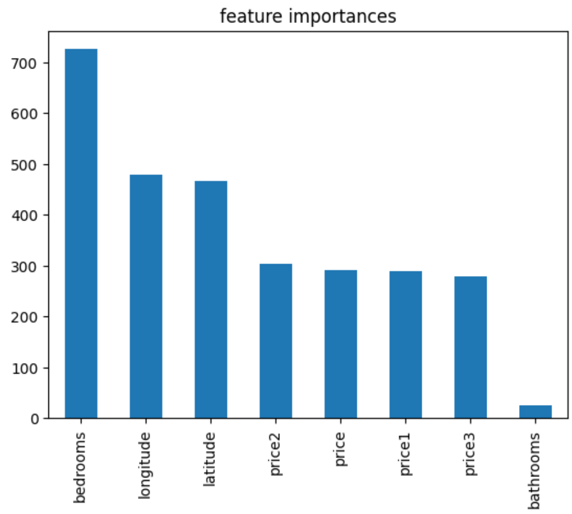
(a) Sklearn: Исходный датасет



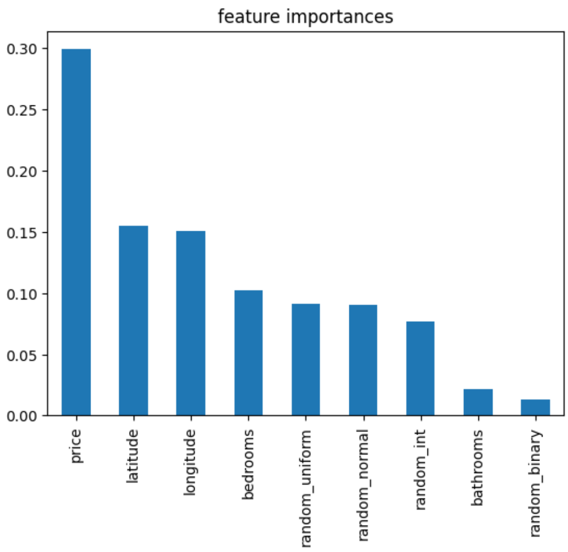
(a) R-ranger: Исходный датасет



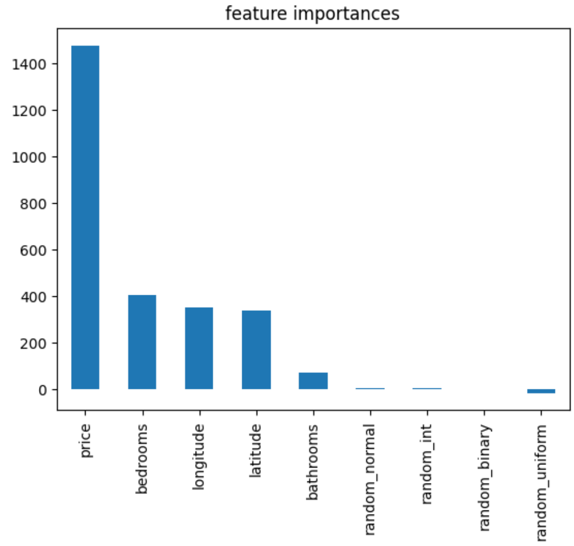
(b) Sklearn: добавлены коррелированные признаки



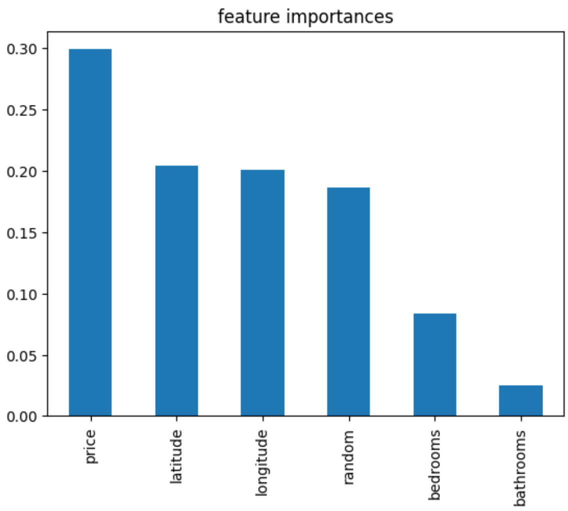
(b) R-ranger: добавлены коррелированные признаки



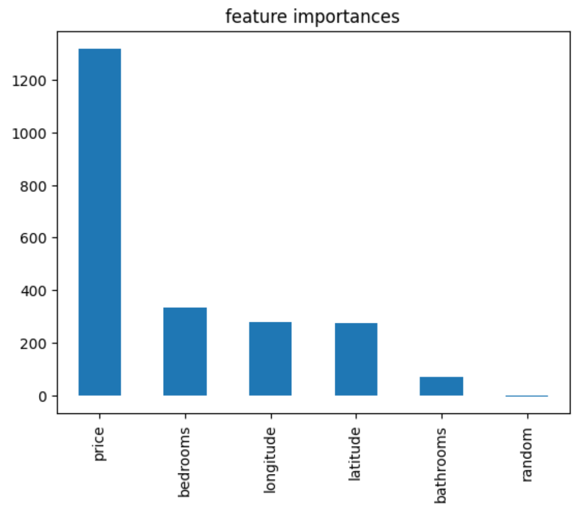
(c) Sklearn: много нерелевантных признаков



(c) R-ranger: много нерелевантных признаков



(d) Sklearn: 1 нерелевантный признак



(d) R-ranger: 1 нерелевантный признак

Код

```
from .python_implementations import (
    sklearn_importance,
    objective_classifier
)

from .r_implementations import (
    r_randomforest_importance,
    clean_feature_names
)

from .pic import (
    picture
)


__version__ = "1.0.0"
__all__ = ['sklearn_importance', 'picture',
           'r_randomforest_importance']
```

Листинг 1: Файл __init__.py

```
import matplotlib.pyplot as plt


def picture(fi):
    fi.plot(kind='bar')
    plt.title('feature importances')
    plt.show()
    return
```

Листинг 2: Файл pic.py

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import optuna
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


def objective_classifier(trial, X_train, y_train):
    n_estimators = trial.suggest_int('n_estimators', 50, 2000)
    max_depth = trial.suggest_int('max_depth', 5, 30, log=True)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 40)
    max_features = trial.suggest_categorical(
        'max_features', ["sqrt", 0.25, 1/3, 0.5, 0.7, 1.0])

    model = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_leaf=min_samples_leaf,
        max_features=max_features,
        random_state=42,
        oob_score=True,
        bootstrap=True,
        n_jobs=-1
    )

    model.fit(X_train, y_train)
```

```python

    return model.oob_score_


def sklearn_importance(X_train, y_train):
    study = optuna.create_study(direction='maximize')
    study.optimize(lambda trial: objective_classifier(
        trial, X_train, y_train), n_trials=100)
    best_params = study.best_params
    rf = RandomForestClassifier(
        **best_params,
        random_state=42,
        oob_score=True,
        bootstrap=True,
        n_jobs=-1
    )
    print(best_params)
    rf.fit(X_train, y_train)
    fi = pd.Series(rf.feature_importances_, index=X_train.columns)
    return fi.sort_values(ascending=False)
```

Листинг 3: Файл python_implementations.py

```python
from rpy2.robjects.vectors import StrVector
import pandas as pd
import numpy as np
import re
import rpy2.robjects as robjects
from rpy2.robjects import pandas2ri
from rpy2.robjects.packages import importr
import warnings
import traceback
from rpy2 import robjects


def clean_feature_names(names):
    """                                                                          ,
                                R."""
    cleaned_names = [name.replace('.', '_').replace(
        '-', '_').replace(' ', '_') for name in names]
    return cleaned_names


def r_randomforest_importance(X, y, data_path=None, n_estimators=1760,
    max_depth=26, min_samples_leaf=5, max_features='sqrt'):
    try:
        base = importr('base')
        utils = importr('utils')
        randomForest = importr('randomForest')

        original_features = list(X.columns)
        cleaned_features = clean_feature_names(original_features)

        df_for_r = X.copy()
        df_for_r.columns = cleaned_features
        df_for_r['target'] = y.values

        with robjects.conversion.localconverter(robjects.default_converter +
    pandas2ri.converter) as cv:
            r_df = robjects.conversion.py2rpy(df_for_r)

```

```python
        formula_str = "target ~ " + " + ".join(cleaned_features)
        formula = robjects.Formula(formula_str)

        print(f"Training R Random Forest with formula: {formula_str}")

        n_samples = X.shape[0]
        n_features = len(cleaned_features)
        if max_features == 'sqrt':
            max_features = round(np.sqrt(n_features))
        else:
            max_features = round(n_features * max_features)

        rf_result = randomForest.randomForest(
            formula,
            data=r_df,
            ntree=n_estimators,
            nodesize=min_samples_leaf,
            maxnodes=2**max_depth,
            mtry=max_features,
            importance=True
        )

        importance_r = randomForest.importance(rf_result, type=2, scale=
    False)

        importance_matrix = np.array(importance_r)

        feature_names_r = list(robjects.r['rownames'](importance_r))

        print(f"Importance matrix shape: {importance_matrix.shape}")
        print(f"Feature names from R: {feature_names_r}")

        if importance_matrix.ndim == 2:
            importance_values = importance_matrix[:, 0]
        else:
            importance_values = importance_matrix

        importance_series = pd.Series(
            importance_values, index=original_features)

        print("Successfully computed variable importance")
        print(importance_series)

        return importance_series.sort_values(ascending=False)

    except Exception as e:
        warnings.warn(f"R implementation failed: {e}")
        import traceback
        traceback.print_exc()
        return None


def r_ranger_importance_air(X, y, n_estimators=1760, max_depth=26,
    min_samples_leaf=5, max_features='sqrt'):
    try:
        base = importr('base')
        ranger = importr('ranger')

        original_features = list(X.columns)

```

```
        cleaned_features = clean_feature_names(original_features)

        df_for_r = X.copy()
        df_for_r.columns = cleaned_features

        df_for_r['target'] = y.astype(str).values

        name_map = dict(zip(cleaned_features, original_features))

        with robjects.conversion.localconverter(robjects.default_converter + pandas2ri.converter):
            r_df = robjects.conversion.py2rpy(df_for_r)

        r_target = base.factor(r_df.rx2('target'))

        r_cleaned_features = StrVector(cleaned_features)

        r_df_features_only = r_df.rx(True, r_cleaned_features)

        r_df = base.cbind(r_df_features_only, target=r_target)

        formula_str = "target ~ " + " + ".join(cleaned_features)
        formula = robjects.Formula(formula_str)

        print(f"Training R Ranger Forest with formula: {formula_str}")

        target_type = robjects.r['class'](r_df.rx2('target'))
        target_levels = robjects.r['levels'](r_df.rx2('target'))
        print(f"Target type in R: {list(target_type)}")
        print(f"Target levels: {list(target_levels)}")

        n_features = len(cleaned_features)
        if max_features == 'sqrt':
            mtry_val = round(np.sqrt(n_features))
        elif isinstance(max_features, float):
            mtry_val = round(n_features * max_features)
        else:
            mtry_val = max_features

        rf_result = ranger.ranger(
            formula,
            data=r_df,
            num_trees=n_estimators,
            min_node_size=min_samples_leaf,
            mtry=mtry_val,
            importance="impurity_corrected",
            max_depth=max_depth,
            classification=True
        )

        task_type = rf_result.rx2('treetype')[0]
        print(f"Ranger task type: {task_type}")

        importance_r = rf_result.rx2('variable.importance')
        importance_values = np.array(importance_r)
        feature_names_r = list(robjects.r['names'](importance_r))

        importance_series = pd.Series(importance_values, index=feature_names_r)
        importance_series.index = importance_series.index.map(name_map)
```

```

        print("Successfully computed variable importance for CLASSIFICATION"
    )

        return importance_series.sort_values(ascending=False)

    except Exception as e:
        warnings.warn(f"Ranger implementation failed: {e}")
        traceback.print_exc()
        return None
```

Листинг 4: Файл r_implementations.py