February 20, 2025

[1]:

```
#
import pandas as pd
import numpy as np

#  xlsx
#  'your_file.xlsx'
df = pd.read_excel(r"D:\\\Data2017 -2019_1.xlsx")

#
print(df.head())

#
print(df.info())
```

```
           X          Y  produkt  date Agrotech  Soil       Sum    Ca    Mg  \
0  59.423807  30.038469      0.1  2018        K     2.31  8.38  7.12  1.26
1  59.423821  30.038704      0.1  2018        K     2.31  8.38  7.12  1.26
2  59.423848  30.039059      0.1  2018        K    2.31  8.38  7.12  1.26
3  59.423862  30.039295      0.1  2018        K    2.31  8.38  7.12  1.26
4  59.423868  30.039537      0.1  2018        K    2.31  8.38  7.12  1.26

   pH_KCL    P    K     N   Org
0     5.4  327  188  0.31  0.89
1     5.4  327  188  0.31  0.89
2     5.4  327  188  0.31  0.89
3     5.4  327  188  0.31  0.89
4     5.4  327  188  0.31  0.89
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8182 entries, 0 to 8181
Data columns (total 15 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   X         8182 non-null   float64
 1   Y         8182 non-null   float64
 2   produkt   8182 non-null   float64
 3   date      8182 non-null   int64
 4   Agrotech  8182 non-null   object
 5   Soil      8171 non-null   object
 6          8182 non-null   float64
 7   Sum       8182 non-null   float64
 8   Ca        8182 non-null   float64
 9   Mg        8182 non-null   float64
 10  pH_KCL    8182 non-null   float64
 11  P         8182 non-null   int64
 12  K         8182 non-null   int64
 13  N         8182 non-null   float64
 14  Org       8182 non-null   float64
dtypes: float64(10), int64(3), object(2)
memory usage: 959.0+ KB
None
```

[2]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from catboost import CatBoostRegressor
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

#    MAPE
def calculate_mape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# 1.
def analyze_outliers(df, numeric_columns, z_score_threshold=3):
    outliers_info = {}
    all_outliers_indices = set()

    print("\n    :")
    for col in numeric_columns:
        z_scores = np.abs(stats.zscore(df[col]))
        outliers_mask = z_scores > z_score_threshold
        outliers_count = outliers_mask.sum()
        outliers_indices = df[outliers_mask].index

        outliers_info[col] = {
            'count': outliers_count,
            'percentage': (outliers_count / len(df)) * 100,
            'indices': outliers_indices
        }

        all_outliers_indices.update(outliers_indices)
        print(f"{col}:  {outliers_count}  ({outliers_info[col]['percentage']:.
↪2f}%)")

    print(f"\n    : {len(all_outliers_indices)}")
    print(f"   : {(len(all_outliers_indices) / len(df)) * 100:.2f}%")

    return list(all_outliers_indices), outliers_info

# 2.
df['Soil'] = df['Soil'].fillna('Unknown')

#
numeric_features = ['', 'Sum', 'Ca', 'Mg', 'pH_KCL', 'P', 'K', 'N', 'Org',␣
↪'produkt']
#
print("   :", (df['produkt'] == 0).sum())

#
df = df[df['produkt'] > 0]
print("     :", len(df))
#
print("\n   :")
print(df['produkt'].describe())

df = df[df['produkt'] > 0]

print("\n   :")
print(df['produkt'].describe())

#
print("\n  :", len(df))
outliers_indices, outliers_info = analyze_outliers(df, numeric_features,␣
↪z_score_threshold=3)

#
remove_outliers = True

if remove_outliers:
    df_clean = df.drop(outliers_indices)
    print(f"    : {len(df_clean)}")
    print(f" : {len(df) - len(df_clean)}")
else:
    df_clean = df
    print("   ")

# 3.
shuffle_index = np.random.permutation(len(df))
```

```
df_shuffled = df.iloc[shuffle_index].reset_index(drop=True)

features = ['Agrotech', 'Soil', '', 'Sum', 'Ca', 'Mg', 'pH_KCL', 'P', 'K', 'N',␣
 ↪'Org']
X = df_clean[features]
y = df_clean['produkt']

#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# 4.     GridSearchCV
param_grid = {
    'learning_rate': [0.05, 0.1],
    'depth': [4, 6],
    'iterations': [500]
}

cat_features = ['Agrotech', 'Soil']

#    GridSearchCV
base_model = CatBoostRegressor(
    cat_features=cat_features,
    random_seed=42,
        iterations=1000,
    depth=6,
    learning_rate=0.03,
    verbose=False

)

# GridSearchCV
grid_search = GridSearchCV(
    estimator=base_model,
    param_grid=param_grid,
    cv=3,
    scoring='neg_root_mean_squared_error',
    n_jobs=-1
)

print("\n  ...")
#
grid_search.fit(X_train, y_train)

#
print("\n :")
print(grid_search.best_params_)
```

```python
#
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# 5.
def print_metrics(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    mape = calculate_mape(y_true, y_pred)

    print(f'\n  {model_name}:')
    print(f'RMSE: {rmse:.4f}')
    print(f'R2 Score: {r2:.4f}')
    print(f'MAPE: {mape:.2f}%')

print_metrics(y_test, y_pred, "CatBoost ( )")

# 6.
feature_importance = pd.DataFrame({
    'feature': features,
    'importance': best_model.feature_importances_
})
print("\n :")
print(feature_importance.sort_values(by='importance', ascending=False))
```

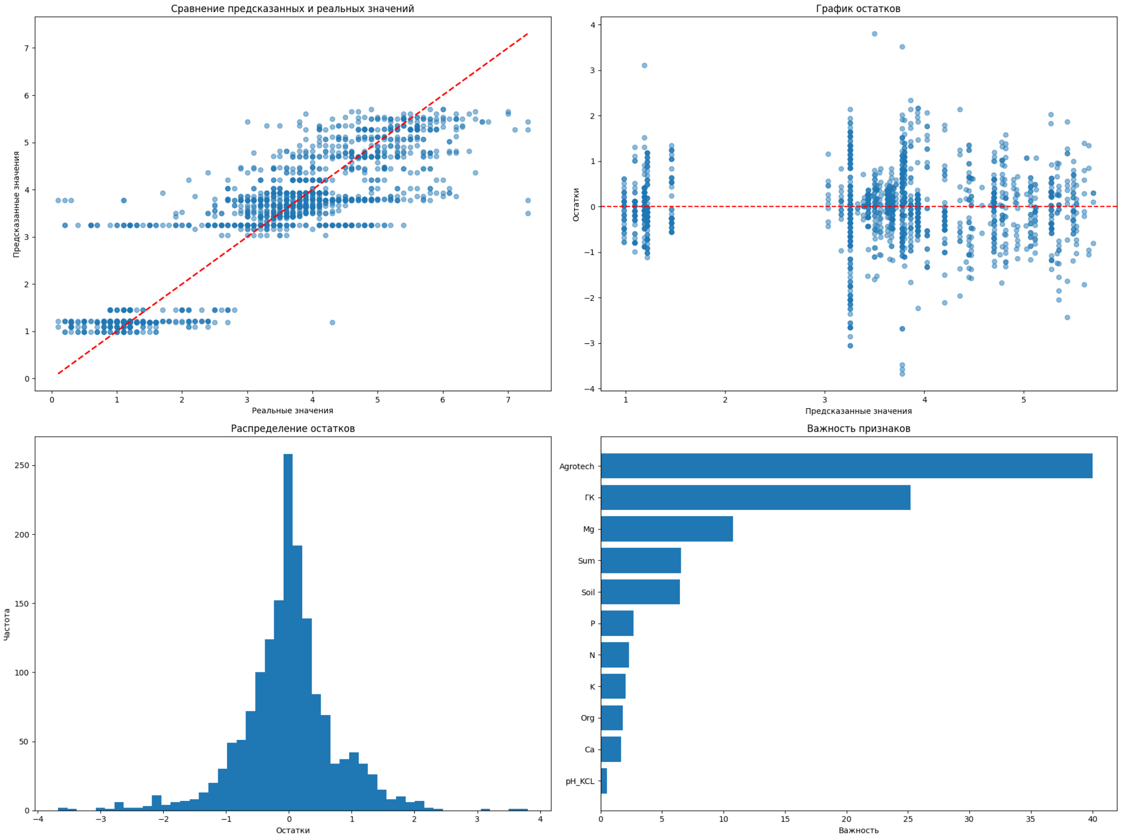
```
   : 0
     : 8182

   :
count    8182.000000
mean        3.416634
std         1.570037
min         0.100000
25%         2.300000
50%         3.700000
75%         4.500000
max        12.500000
Name: produkt, dtype: float64

   :
count    8182.000000
mean        3.416634
std         1.570037
min         0.100000
25%         2.300000
50%         3.700000
```

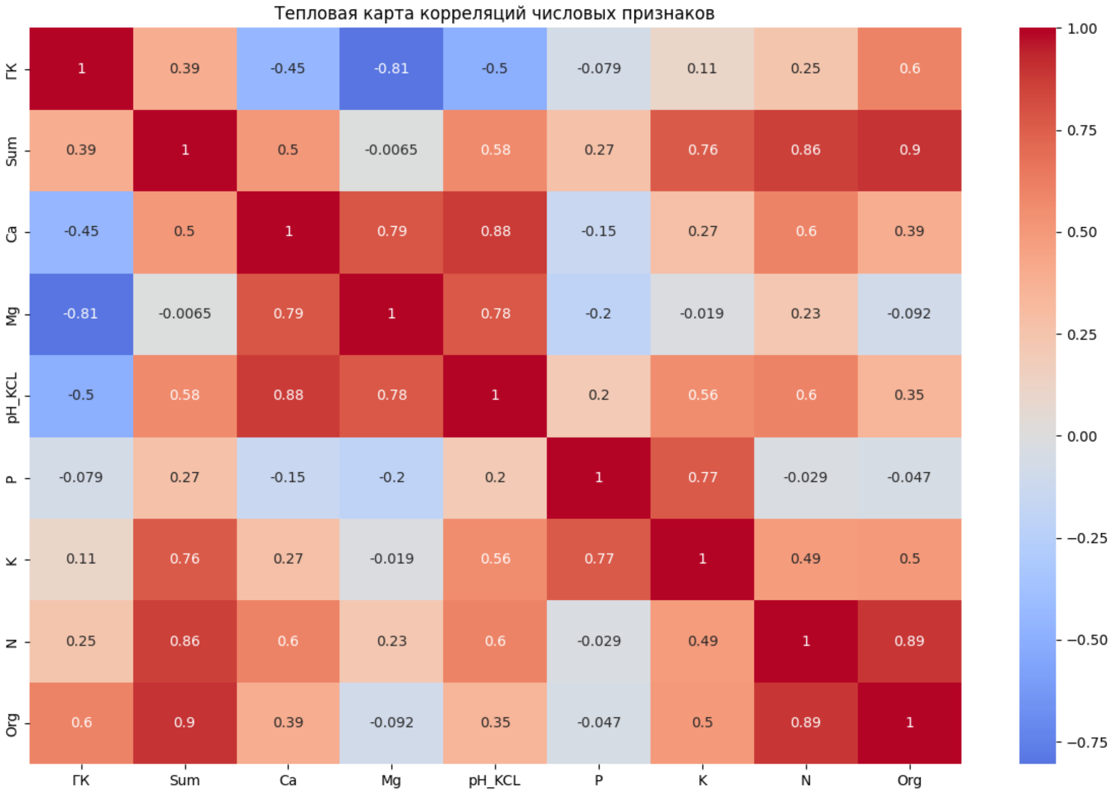
```
75%         4.500000
max        12.500000
Name: produkt, dtype: float64

  : 8182

    :
:  0  (0.00%)
Sum:  0  (0.00%)
Ca:  0  (0.00%)
Mg:  0  (0.00%)
pH_KCL:  0  (0.00%)
P:  0  (0.00%)
K:  0  (0.00%)
N:  0  (0.00%)
Org:  0  (0.00%)
produkt:  18  (0.22%)

    : 18
   : 0.22%
    : 8164
 : 18

  ...

 :
{'depth': 6, 'iterations': 500, 'learning_rate': 0.05}

  CatBoost ( ):
RMSE: 0.7501
R2 Score: 0.7664
MAPE: 31.37%

 :
     feature  importance
0   Agrotech   39.991141
2             25.191873
5         Mg   10.762833
3        Sum    6.534395
1       Soil    6.460281
7          P    2.692719
9          N    2.330991
8          K    2.027546
10       Org    1.801065
4         Ca    1.679669
6     pH_KCL    0.527487
```

[3]:

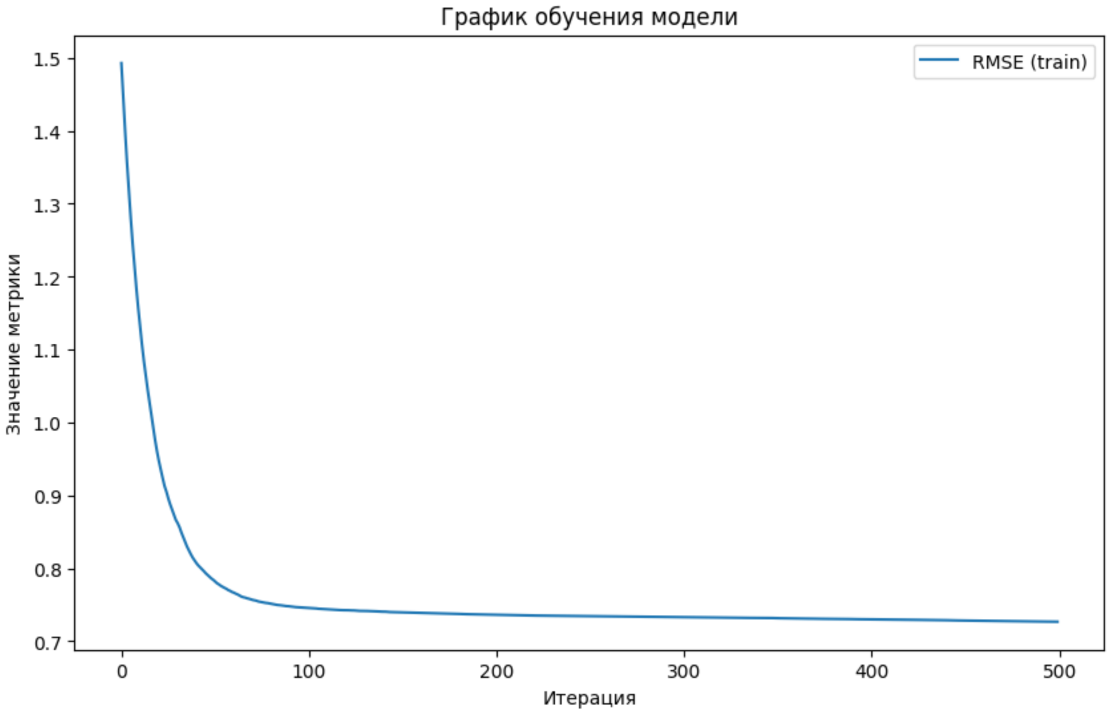
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

def visualize_model_results(model, X_train, X_test, y_train, y_test, y_pred,␣
 ↪features):
    #
    plt.figure(figsize=(20, 15))

    # 1.   vs
    plt.subplot(2, 2, 1)
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--',␣
 ↪lw=2)
    plt.xlabel(' ')
    plt.ylabel(' ')
    plt.title('    ')

    # 2.
    plt.subplot(2, 2, 2)
    residuals = y_test - y_pred
    plt.scatter(y_pred, residuals, alpha=0.5)
    plt.axhline(y=0, color='r', linestyle='--')
    plt.xlabel(' ')
    plt.ylabel('')
    plt.title(' ')

    # 3.
    plt.subplot(2, 2, 3)
    plt.hist(residuals, bins=50)
    plt.xlabel('')
    plt.ylabel('')
    plt.title(' ')

    # 4.
    importance = pd.DataFrame({
        'feature': features,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=True)

    plt.subplot(2, 2, 4)
    plt.barh(importance['feature'], importance['importance'])
    plt.xlabel('')
    plt.title(' ')

    plt.tight_layout()
    plt.show()

    # 5.
    plt.figure(figsize=(12, 8))
    numeric_cols = X_train.select_dtypes(include=[np.number]).columns
    correlation_matrix = X_train[numeric_cols].corr()
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
    plt.title('    ')
    plt.tight_layout()
    plt.show()

    # 6.    ( )
    if hasattr(model, 'evals_result_'):
        plt.figure(figsize=(10, 6))
        #   ,
        eval_metrics = model.evals_result_['learn'].keys()
        for metric in eval_metrics:
            plt.plot(model.evals_result_['learn'][metric],
                    label=f'{metric} (train)')
            if 'test' in model.evals_result_:
                plt.plot(model.evals_result_['test'][metric],
                        label=f'{metric} (test)')
        plt.xlabel('')
        plt.ylabel(' ')
        plt.title('  ')
        plt.legend()
        plt.show()

    #
    print("\n :")
    for param_name, param_value in model.get_params().items():
        print(f"{param_name}: {param_value}")

    #
    print("\n  :")
    print(f" : {np.mean(residuals):.4f}")
    print(f"  : {np.std(residuals):.4f}")
    print(f" : {np.min(residuals):.4f}")
    print(f" : {np.max(residuals):.4f}")

    #
    print("\n  :")
    print(f"   (MAE): {np.mean(np.abs(residuals)):.4f}")
    print(f"  : {np.median(np.abs(residuals)):.4f}")

    #
    percentiles = [1, 5, 25, 50, 75, 95, 99]
```

```
    print("\n :")
    for p in percentiles:
        print(f"{p}- : {np.percentile(residuals, p):.4f}")

#
visualize_model_results(
    model=best_model,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    y_pred=y_pred,
    features=features
)

#
print("\n   :")
for cat_feature in ['Agrotech', 'Soil']:
    print(f"\n    {cat_feature}:")
    for unique_val in X_test[cat_feature].unique():
        mask = X_test[cat_feature] == unique_val
        if mask.sum() > 0:
            group_rmse = np.sqrt(mean_squared_error(y_test[mask], y_pred[mask]))
            group_r2 = r2_score(y_test[mask], y_pred[mask])
            group_mape = calculate_mape(y_test[mask], y_pred[mask])
            print(f"\n{unique_val}:")
            print(f"RMSE: {group_rmse:.4f}")
            print(f"R2: {group_r2:.4f}")
            print(f"MAPE: {group_mape:.2f}%")
            print(f" : {mask.sum()}")

#
best_model.save_model('catboost_model.cbm')
print("\n    'catboost_model.cbm'")
```


Сравнение предсказанных и реальных значений
Предсказанные значения
Реальные значения
0
1
2
3
4
5
6
7
График остатков
Остатки
Предсказанные значения
4
3
2
1
0
−1
−2
−3
−4
Распределение остатков
Частота
Остатки
0
50
100
150
200
250
−4
−3
−2
−1
0
1
2
3
4
Важность признаков
Agrotech
ГК
Mg
Sum
Soil
P
N
K
Org
Ca
pH_KCL
Важность
0
5
10
15
20
25
30
35
40



	ГК	Sum	Ca	Mg	pH_KCL	P	K	N	Org
ГК	1	0.39	-0.45	-0.81	-0.5	-0.079	0.11	0.25	0.6
Sum	0.39	1	0.5	-0.0065	0.58	0.27	0.76	0.86	0.9
Ca	-0.45	0.5	1	0.79	0.88	-0.15	0.27	0.6	0.39
Mg	-0.81	-0.0065	0.79	1	0.78	-0.2	-0.019	0.23	-0.092
pH_KCL	-0.5	0.58	0.88	0.78	1	0.2	0.56	0.6	0.35
P	-0.079	0.27	-0.15	-0.2	0.2	1	0.77	-0.029	-0.047
K	0.11	0.76	0.27	-0.019	0.56	0.77	1	0.49	0.5
N	0.25	0.86	0.6	0.23	0.6	-0.029	0.49	1	0.89
Org	0.6	0.9	0.39	-0.092	0.35	-0.047	0.5	0.89	1



```
 :
iterations: 500
learning_rate: 0.05
depth: 6
loss_function: RMSE
random_seed: 42
verbose: False
cat_features: ['Agrotech', 'Soil']

  :
 : -0.0105
  : 0.7500
 : -3.6772
 : 3.8020

  :
   (MAE): 0.5189
  : 0.3333

 :
1- : -2.2523
5- : -1.1702
25- : -0.3578
50- : 0.0010
75- : 0.3150
95- : 1.2477
99- : 1.8900

   :

    Agrotech:

X_1:
RMSE: 0.7468
R2: 0.0723
MAPE: 14.21%
 : 94

K:
RMSE: 0.4671
R2: 0.8653
MAPE: 20.56%
 : 408

TZ:
```

```
RMSE: 0.9659
R2: 0.4082
MAPE: 41.79%
 : 617

VI:
RMSE: 0.6263
R2: 0.8678
MAPE: 30.58%
 : 514

  Soil:

:
RMSE: 0.7493
R2: 0.7808
MAPE: 40.37%
 : 435

:
RMSE: 0.5874
R2: 0.8521
MAPE: 23.46%
 : 633

:
RMSE: 0.7089
R2: 0.5012
MAPE: 12.57%
 : 218

:
RMSE: 0.5419
R2: 0.7133
MAPE: 10.49%
 : 26

 :
RMSE: 1.1338
R2: 0.1972
MAPE: 64.17%
 : 225

:
RMSE: 0.7577
R2: 0.1625
MAPE: 14.76%
 : 70
 :
RMSE: 0.5459
R2: 0.0437
MAPE: 14.19%
 : 13

Unknown:
RMSE: 0.5918
R2: -2.6654
MAPE: 10.32%
 : 3

 :
RMSE: 0.6738
R2: 0.1217
MAPE: 11.99%
 : 10

    'catboost_model.cbm'
```

[4]:

```
import os
#    Graphviz
os.environ["PATH"] += os.pathsep + r"C:\Program Files\Graphviz\bin"

def visualize_trees(model, features, max_trees=3):
    """
             CatBoost
    """
    try:
        import graphviz

        #   dot
        try:
            from subprocess import run, PIPE
            result = run(['dot', '-V'], stdout=PIPE, stderr=PIPE)
            print("Graphviz version:", result.stderr.decode())
        except Exception as e:
            print("    Graphviz:", str(e))

        print(f"\n  {max_trees} ")

        #
        if not os.path.exists('tree_visualizations'):
            os.makedirs('tree_visualizations')

        #     dot
        graphviz.backend.executables = {'dot': r'C:\Program␣
↪Files\Graphviz\bin\dot.exe'}

        #
        for tree_idx in range(min(max_trees, model.tree_count_)):
            try:
                tree_graph = model.plot_tree(tree_idx)

                #        dot
                output_file = f'tree_visualizations/tree_{tree_idx}'
                tree_graph.render(filename=output_file,
                                format='png',
                                cleanup=True,
                                engine='dot')
                print(f" {tree_idx}   {output_file}.png")

            except Exception as e:
                print(f"    {tree_idx}: {str(e)}")

        #      ...

    except Exception as e:
        print(f"   : {str(e)}")

#
visualize_trees(best_model, features, max_trees=3)
```

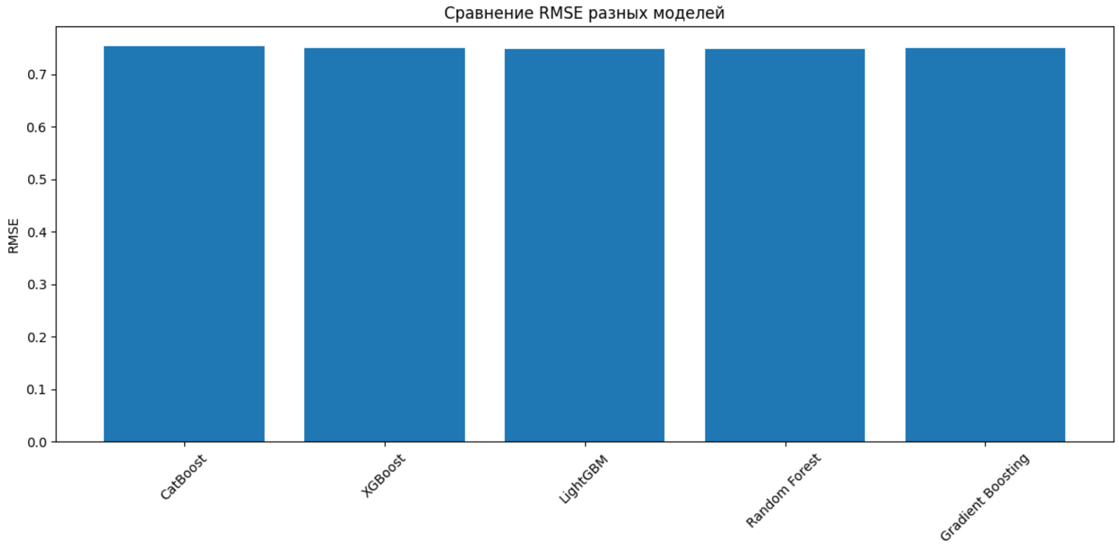
```
Graphviz version: dot - graphviz version 12.2.1 (20241206.2353)


  3
0   tree_visualizations/tree_0.png
1   tree_visualizations/tree_1.png
2   tree_visualizations/tree_2.png
```

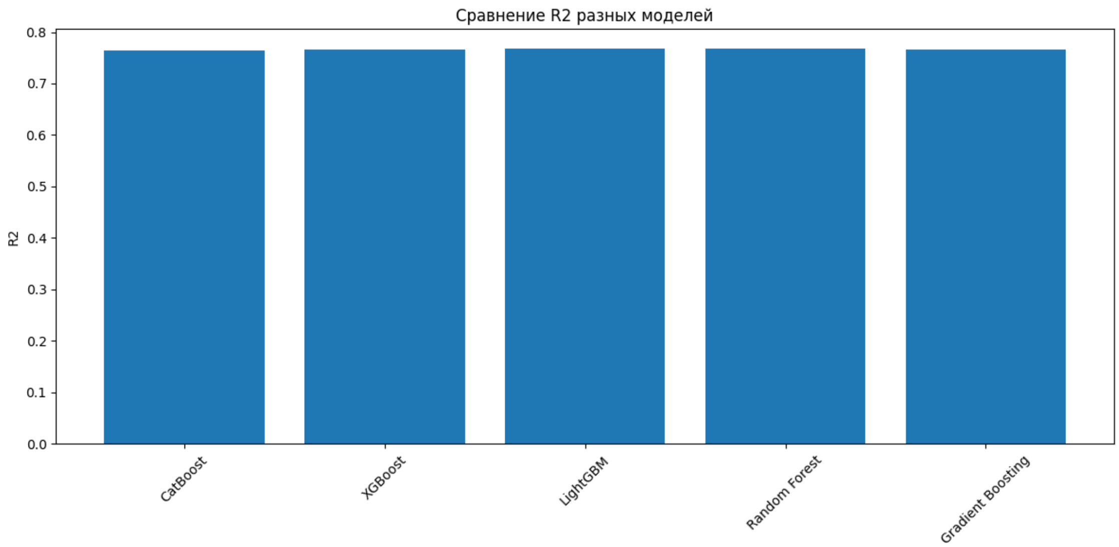
[5]:
```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import numpy as np

#
results = {}

# 1. CatBoost
catboost_model = CatBoostRegressor(
    cat_features=cat_features,
    iterations=2000,
    depth=8,
    learning_rate=0.03,
    verbose=False,
    random_seed=42
)

# 2. XGBoost
xgb_model = XGBRegressor(
    n_estimators=2000,
    max_depth=8,
    learning_rate=0.03,
    random_state=42,
    n_jobs=-1
)

# 3. LightGBM
lgb_model = LGBMRegressor(
    n_estimators=2000,
    max_depth=8,
    learning_rate=0.03,
    random_state=42,
    n_jobs=-1
)

# 4. Random Forest
rf_model = RandomForestRegressor(
    n_estimators=500,
    max_depth=8,
    random_state=42,
    n_jobs=-1
)

# 5. Gradient Boosting
gb_model = GradientBoostingRegressor(
    n_estimators=500,
    max_depth=8,
    learning_rate=0.03,
    random_state=42
)

#
models = {
    'CatBoost': catboost_model,
    'XGBoost': xgb_model,
    'LightGBM': lgb_model,
    'Random Forest': rf_model,
    'Gradient Boosting': gb_model
}

#       CatBoost
from sklearn.preprocessing import LabelEncoder

#
X_train_encoded = X_train.copy()
X_test_encoded = X_test.copy()

#
label_encoders = {}
for feature in cat_features:
    le = LabelEncoder()
    X_train_encoded[feature] = le.fit_transform(X_train[feature])
    X_test_encoded[feature] = le.transform(X_test[feature])
    label_encoders[feature] = le

#
for name, model in models.items():
    print(f"\n  {name}...")

    #
    if name == 'CatBoost':
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
    else:
        model.fit(X_train_encoded, y_train)
        y_pred = model.predict(X_test_encoded)

    #
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    mape = calculate_mape(y_test, y_pred)

    results[name] = {
        'RMSE': rmse,
        'R2': r2,
        'MAPE': mape
    }

    print(f"{name} :")
    print(f"RMSE: {rmse:.4f}")
    print(f"R2: {r2:.4f}")
    print(f"MAPE: {mape:.2f}%")
```

```python
#
import matplotlib.pyplot as plt

#   RMSE
plt.figure(figsize=(12, 6))
plt.bar([name for name in results.keys()],
        [results[name]['RMSE'] for name in results.keys()])
plt.title(' RMSE  ')
plt.xticks(rotation=45)
plt.ylabel('RMSE')
plt.tight_layout()
plt.show()

#   R2
plt.figure(figsize=(12, 6))
plt.bar([name for name in results.keys()],
        [results[name]['R2'] for name in results.keys()])
plt.title(' R2  ')
plt.xticks(rotation=45)
plt.ylabel('R2')
plt.tight_layout()
plt.show()

#  DataFrame
results_df = pd.DataFrame.from_dict(results, orient='index')
print("\n  :")
print(results_df)

#
results_df.to_csv('model_comparison_results.csv')
```

```
  CatBoost...
CatBoost :
RMSE: 0.7535
R2: 0.7642
MAPE: 31.31%

  XGBoost...
XGBoost :
RMSE: 0.7497
R2: 0.7666
MAPE: 31.19%

  LightGBM...
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000401 seconds.
You can set `force_row_wise=true` to remove the overhead.
```

```
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 82
[LightGBM] [Info] Number of data points in the train set: 6531, number of used
features: 11
LightGBM :
RMSE: 0.7483
R2: 0.7674
MAPE: 31.18%

  Random Forest...
Random Forest :
RMSE: 0.7491
R2: 0.7670
MAPE: 31.21%

  Gradient Boosting...
Gradient Boosting :
RMSE: 0.7502
R2: 0.7663
MAPE: 31.20%
```





```
  :
                       RMSE        R2       MAPE
CatBoost           0.753495  0.764226  31.305141
XGBoost            0.749670  0.766614  31.186627
LightGBM           0.748340  0.767441  31.177163
Random Forest      0.749088  0.766976  31.213115
Gradient Boosting  0.750208  0.766278  31.200033
```

[6]:

```python
#
tree_graph = best_model.plot_tree(
    tree_index=0,
    pool=None
)
tree_graph.render(filename='first_tree',
                 format='png',
                 cleanup=True)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 2
      1 #
----> 2 tree_graph = best_model.plot_tree(
      3     tree_index=0,
      4     pool=None
      5 )
      6 tree_graph.render(filename='first_tree',
      7                  format='png',
      8                  cleanup=True)
```

```
TypeError: CatBoost.plot_tree() got an unexpected keyword argument 'tree_index'
```

[ ]:

```

```