Код для CatBoost с log-трансформацией и RMSEWithUncertainty

Установка и импорт библиотек

```
import subprocess
import sys
import numpy as np
import pandas as pd
from catboost import CatBoostRegressor, Pool
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

# Установка CatBoost если необходимо
try:
    from catboost import CatBoostRegressor
except ImportError:
    subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'catboost'])
    from catboost import CatBoostRegressor, Pool
```

1. Подготовка данных с log-трансформацией

```
# Создание датасета (тот же код, что и раньше)
np.random.seed(42)
n_samples = 1000

# Числовые признаки
area = np.random.normal(70, 20, n_samples)
area = np.clip(area, 30, 150)
rooms = np.random.poisson(2.5, n_samples) + 1
rooms = np.clip(rooms, 1, 5)
floor = np.random.randint(1, 16, n_samples)
distance_to_center = np.random.exponential(5, n_samples)
distance_to_center = np.clip(distance_to_center, 1, 20)
building_age = np.random.gamma(2, 8, n_samples)
building_age = np.clip(building_age, 1, 50)

# Категориальные признаки
districts = ['Центральный', 'Северный', 'Южный', 'Восточный', 'Западный']
district = np.random.choice(districts, n_samples)
building_types = ['Панельный', 'Кирпичный', 'Монолитный']
building_type = np.random.choice(building_types, n_samples)

# Создание целевой переменной с нелинейными зависимостями
base_price = 50000
```

```
district_multipliers = {'Центральный': 1.3, 'Северный': 1.0, 'Южный': 0.9,
                        'Восточный': 0.85, 'Западный': 1.1}
district_effect = np.array([district_multipliers[d] for d in district])

building_type_effects = {'Кирпичный': 1.1, 'Монолитный': 1.05, 'Панельный': 0.95}
building_type_effect = np.array([building_type_effects[bt] for bt in building_type])

# Нелинейные эффекты
area_nonlinear = area * 800 + (area ** 1.2) * 10
floor_effect = floor * 200 + np.where(floor > 10, 500, 0)
age_penalty = building_age * 300 + (building_age ** 1.5) * 20

price = (base_price + area_nonlinear + rooms * 5000 + floor_effect +
         -distance_to_center * 1500) * district_effect * building_type_effect - age_penal
price += np.random.normal(0, 8000, n_samples)
price = np.clip(price, 20000, 300000)

# КЛЮЧЕВОЕ ОТЛИЧИЕ: Логарифмическая трансформация
print(f"Распределение исходной цены - Шапиро-Уилк p-value: {stats.shapiro(price)[1]:.6f}"

log_price = np.log(price)
print(f"Распределение log(цены) - Шапиро-Уилк p-value: {stats.shapiro(log_price)[1]:.6f}"
print("Log-трансформация улучшила нормальность распределения")

# Создаем DataFrame
df = pd.DataFrame({
    'area': area, 'rooms': rooms, 'floor': floor,
    'distance_to_center': distance_to_center, 'building_age': building_age,
    'district': district, 'building_type': building_type,
    'price': price,  # исходная цена для сравнения
    'log_price': log_price  # трансформированная целевая переменная
})
```

2. Обучение CatBoost с RMSEWithUncertainty

```
# Подготовка данных
X = df.drop(['price', 'log_price'], axis=1)
y_original = df['price']  # исходная цена для оценки
y_log = df['log_price']   # log цена - наша целевая переменная

categorical_features = ['district', 'building_type']
X_train, X_test, y_log_train, y_log_test, y_orig_train, y_orig_test = train_test_split(
    X, y_log, y_original, test_size=0.2, random_state=42
)

# Создаем Pool объекты
train_pool = Pool(X_train, y_log_train, cat_features=categorical_features)
test_pool = Pool(X_test, y_log_test, cat_features=categorical_features)

# КЛЮЧЕВОЕ ОТЛИЧИЕ: Модель с RMSEWithUncertainty
catboost_model = CatBoostRegressor(
    iterations=400,
    learning_rate=0.1,
    depth=6,
    loss_function='RMSEWithUncertainty',  # Функция потерь с неопределенностью
```

```
    eval_metric='RMSEWithUncertainty',
    random_seed=42,
    verbose=100,
    early_stopping_rounds=50,
    use_best_model=True,
    posterior_sampling=True  # Включает virtual ensemble для uncertainty
)

# Обучение модели
catboost_model.fit(train_pool, eval_set=test_pool)

print(f"Обучение завершено. Количество деревьев: {catboost_model.tree_count_}")
```

3. Получение предсказаний с uncertainty

```
# Получаем предсказания: [mean, variance] для каждого наблюдения
train_predictions = catboost_model.predict(X_train)  # shape: (n_samples, 2)
test_predictions = catboost_model.predict(X_test)    # shape: (n_samples, 2)

# Извлекаем компоненты предсказаний в log-scale
y_log_test_pred_mean = test_predictions[:, 0]  # среднее в log-scale
y_log_test_pred_var = test_predictions[:, 1]   # дисперсия в log-scale

print(f"Предсказания получены:")
print(f"Средние log-предсказания: {y_log_test_pred_mean.min():.3f} - {y_log_test_pred_mea
print(f"Дисперсии: {y_log_test_pred_var.min():.6f} - {y_log_test_pred_var.max():.6f}")
```

4. Обратная трансформация в исходный масштаб

```
def log_to_original_scale_with_correction(log_mean, log_var):
    """
    Преобразует предсказания из логарифмического масштаба в исходный
    с коррекцией смещения для логнормального распределения

    Если log(Y) ~ Normal(μ, σ²), то:
    E[Y] = exp(μ + σ²/2)  - среднее с коррекцией смещения
    Median[Y] = exp(μ)    - медиана (без коррекции)
    """
    # Среднее с коррекцией смещения (bias correction)
    corrected_mean = np.exp(log_mean + log_var / 2)

    # Медиана (геометрическое среднее)
    median = np.exp(log_mean)

    return corrected_mean, median

def calculate_confidence_intervals_log_normal(log_mean, log_var, confidence_level=0.95):
    """
    Вычисляет доверительные интервалы для логнормального распределения
    """
    # Z-score для заданного уровня доверия
    alpha = 1 - confidence_level
    z_score = stats.norm.ppf(1 - alpha/2)
```

```
    # Стандартное отклонение в log-scale
    log_std = np.sqrt(log_var)

    # Границы ДИ в log-scale
    log_lower = log_mean - z_score * log_std
    log_upper = log_mean + z_score * log_std

    # Преобразуем в исходный масштаб (экспонента сохраняет порядок)
    lower_bound = np.exp(log_lower)
    upper_bound = np.exp(log_upper)

    return lower_bound, upper_bound

# Применяем трансформации
y_test_pred_mean_corrected, y_test_pred_median = log_to_original_scale_with_correction(
    y_log_test_pred_mean, y_log_test_pred_var
)

# Вычисляем доверительные интервалы
ci_lower_95, ci_upper_95 = calculate_confidence_intervals_log_normal(
    y_log_test_pred_mean, y_log_test_pred_var, confidence_level=0.95
)

ci_lower_80, ci_upper_80 = calculate_confidence_intervals_log_normal(
    y_log_test_pred_mean, y_log_test_pred_var, confidence_level=0.80
)

print("Обратная трансформация выполнена!")
print(f"Диапазон предсказанных цен: {y_test_pred_mean_corrected.min():.0f} - {y_test_pred
```

5. Анализ остатков в исходном масштабе

```
# Вычисляем остатки в исходном масштабе
residuals_corrected_mean = y_orig_test - y_test_pred_mean_corrected
residuals_median = y_orig_test - y_test_pred_median

# Метрики качества в исходном масштабе
r2_corrected = r2_score(y_orig_test, y_test_pred_mean_corrected)
rmse_corrected = np.sqrt(mean_squared_error(y_orig_test, y_test_pred_mean_corrected))
mae_corrected = mean_absolute_error(y_orig_test, y_test_pred_mean_corrected)

print("=== РЕЗУЛЬТАТЫ В ИСХОДНОМ МАСШТАБЕ ===")
print(f"R²: {r2_corrected:.4f}")
print(f"RMSE: {rmse_corrected:.2f} руб.")
print(f"MAE: {mae_corrected:.2f} руб.")

print(f"Остатки (скорректированное среднее):")
print(f"  Среднее: {residuals_corrected_mean.mean():.2f} руб.")
print(f"  Стд. отклонение: {residuals_corrected_mean.std():.2f} руб.")

# Анализ качества доверительных интервалов
coverage_95 = ((y_orig_test >= ci_lower_95) & (y_orig_test <= ci_upper_95)).mean()
coverage_80 = ((y_orig_test >= ci_lower_80) & (y_orig_test <= ci_upper_80)).mean()
```

```
print(f"Покрытие 95% ДИ: {coverage_95:.1%} (ожидается ~95%)")
print(f"Покрытие 80% ДИ: {coverage_80:.1%} (ожидается ~80%)")

avg_width_95 = (ci_upper_95 - ci_lower_95).mean()
avg_width_80 = (ci_upper_80 - ci_lower_80).mean()

print(f"Средняя ширина 95% ДИ: {avg_width_95:.0f} руб.")
print(f"Средняя ширина 80% ДИ: {avg_width_80:.0f} руб.")
```

6. Создание итогового DataFrame с результатами

```
# Создаем полный DataFrame с результатами для заказчика
results_df = pd.DataFrame({
    # Основные результаты
    'actual_price': y_orig_test.values,
    'predicted_mean_corrected': y_test_pred_mean_corrected,
    'predicted_median': y_test_pred_median,

    # Данные в log-scale для диагностики
    'log_predicted_mean': y_log_test_pred_mean,
    'log_predicted_variance': y_log_test_pred_var,

    # Остатки
    'residuals_corrected': residuals_corrected_mean,
    'residuals_median': residuals_median,

    # Доверительные интервалы
    'ci_95_lower': ci_lower_95,
    'ci_95_upper': ci_upper_95,
    'ci_80_lower': ci_lower_80,
    'ci_80_upper': ci_upper_80,

    # Исходные признаки для анализа
    'area': X_test['area'].values,
    'rooms': X_test['rooms'].values,
    'floor': X_test['floor'].values,
    'distance_to_center': X_test['distance_to_center'].values,
    'building_age': X_test['building_age'].values,
    'district': X_test['district'].values,
    'building_type': X_test['building_type'].values
})

# Добавляем вычисляемые поля для анализа
results_df['uncertainty_percent'] = (results_df['ci_95_upper'] - results_df['ci_95_lower'
results_df['in_ci_95'] = (results_df['actual_price'] >= results_df['ci_95_lower']) & (res
results_df['in_ci_80'] = (results_df['actual_price'] >= results_df['ci_80_lower']) & (res

results_df.to_csv('catboost_log_uncertainty_results.csv', index=False)
print("Результаты сохранены в 'catboost_log_uncertainty_results.csv'")
```

7. Примеры интерпретации для заказчика

```
def create_client_examples(results_df, n_examples=5):
    """
    Создает примеры предсказаний с доверительными интервалами для заказчика
    """
    examples = []

    for i in range(min(n_examples, len(results_df))):
        row = results_df.iloc[i]

        example = {
            'Характеристики': f"{row['area']:.0f} кв.м, {row['rooms']:.0f} комн., {row['f
            'Район': row['district'],
            'Тип дома': row['building_type'],
            'Фактическая цена': f"{row['actual_price']:,.0f} руб.",
            'Предсказанная цена': f"{row['predicted_mean_corrected']:,.0f} руб.",
            'Доверительный интервал 95%': f"{row['ci_95_lower']:,.0f} - {row['ci_95_upper
            'Доверительный интервал 80%': f"{row['ci_80_lower']:,.0f} - {row['ci_80_upper
            'Уровень неопределенности': f"±{row['uncertainty_percent']:.1f}%",
            'Попадание в 95% ДИ': '✓' if row['in_ci_95'] else '✗',
            'Попадание в 80% ДИ': '✓' if row['in_ci_80'] else '✗'
        }
        examples.append(example)

    return examples

# Создаем примеры для демонстрации
client_examples = create_client_examples(results_df, 5)

print("ПРИМЕРЫ ПРЕДСКАЗАНИЙ С ДОВЕРИТЕЛЬНЫМИ ИНТЕРВАЛАМИ:")
print("=" * 80)

for i, example in enumerate(client_examples, 1):
    print(f"КВАРТИРА {i}:")
    for key, value in example.items():
        print(f"  {key}: {value}")
    print()
```

8. Диагностика и улучшение модели

```
def diagnose_uncertainty_quality(results_df):
    """
    Диагностирует качество доверительных интервалов
    """
    diagnostics = {}

    # Проверка калибровки доверительных интервалов
    diagnostics['coverage_95'] = results_df['in_ci_95'].mean()
    diagnostics['coverage_80'] = results_df['in_ci_80'].mean()

    # Проверка на недо/переоценку неопределенности
    if diagnostics['coverage_95'] < 0.90:
        diagnostics['uncertainty_issue'] = "Недооценка неопределенности (ДИ слишком узкие
```

```
    elif diagnostics['coverage_95'] > 0.98:
        diagnostics['uncertainty_issue'] = "Переоценка неопределенности (ДИ слишком широк
    else:
        diagnostics['uncertainty_issue'] = "Неопределенность калибрована корректно"

    # Анализ ширины интервалов
    diagnostics['avg_relative_width'] = results_df['uncertainty_percent'].mean()

    # Корреляция между неопределенностью и ошибкой
    diagnostics['uncertainty_error_correlation'] = np.corrcoef(
        results_df['uncertainty_percent'],
        np.abs(results_df['residuals_corrected'])
    )[0, 1]

    return diagnostics

# Выполняем диагностику
diagnostics = diagnose_uncertainty_quality(results_df)

print("=== ДИАГНОСТИКА ДОВЕРИТЕЛЬНЫХ ИНТЕРВАЛОВ ===")
for key, value in diagnostics.items():
    print(f"{key}: {value}")
```

9. Продвинутые возможности

```
# Получение Virtual Ensemble предсказаний для дополнительного анализа
def get_virtual_ensemble_predictions(model, X_test, n_ensembles=10):
    """
    Получает предсказания virtual ensemble для анализа knowledge uncertainty
    """
    try:
        virtual_preds = model.virtual_ensembles_predict(
            X_test,
            prediction_type='TotalUncertainty',
            virtual_ensembles_count=n_ensembles
        )

        # virtual_preds содержит: [mean, knowledge_uncertainty, data_uncertainty]
        mean_preds = virtual_preds[:, 0]
        knowledge_uncertainty = virtual_preds[:, 1]
        data_uncertainty = virtual_preds[:, 2]

        return mean_preds, knowledge_uncertainty, data_uncertainty

    except Exception as e:
        print(f"Virtual ensemble недоступен: {e}")
        return None, None, None

# Пример использования virtual ensemble
try:
    ve_mean, ve_knowledge, ve_data = get_virtual_ensemble_predictions(catboost_model, X_t

    if ve_mean is not None:
        print("Virtual Ensemble результаты получены:")
        print(f"Knowledge uncertainty: {ve_knowledge.mean():.6f} ± {ve_knowledge.std():.6
```

```
        print(f"Data uncertainty: {ve_data.mean():.6f} ± {ve_data.std():.6f}")

        # Добавляем к результатам
        results_df['virtual_ensemble_mean'] = np.exp(ve_mean + ve_data/2)  # с коррекцией
        results_df['knowledge_uncertainty'] = ve_knowledge
        results_df['data_uncertainty'] = ve_data

except Exception as e:
    print(f"Virtual ensemble анализ не выполнен: {e}")

print("Анализ завершен!")
```

10. Визуализация результатов

```
import matplotlib.pyplot as plt

def plot_uncertainty_analysis(results_df):
    """
    Создает визуализацию анализа неопределенности
    """
    fig, axes = plt.subplots(2, 3, figsize=(18, 12))
    fig.suptitle('Анализ CatBoost с Log-трансформацией и RMSEWithUncertainty', fontsize=1

    # 1. Predicted vs Actual
    axes[0, 0].scatter(results_df['predicted_mean_corrected'], results_df['actual_price']
    min_val = min(results_df['predicted_mean_corrected'].min(), results_df['actual_price'
    max_val = max(results_df['predicted_mean_corrected'].max(), results_df['actual_price'
    axes[0, 0].plot([min_val, max_val], [min_val, max_val], 'r--', lw=2)
    axes[0, 0].set_xlabel('Предсказанная цена (руб.)')
    axes[0, 0].set_ylabel('Фактическая цена (руб.)')
    axes[0, 0].set_title('Предсказанные vs Фактические цены')
    axes[0, 0].grid(True, alpha=0.3)

    # 2. Residuals plot
    axes[0, 1].scatter(results_df['predicted_mean_corrected'], results_df['residuals_corr
    axes[0, 1].axhline(y=0, color='r', linestyle='--')
    axes[0, 1].set_xlabel('Предсказанная цена (руб.)')
    axes[0, 1].set_ylabel('Остатки (руб.)')
    axes[0, 1].set_title('Остатки vs Предсказанные значения')
    axes[0, 1].grid(True, alpha=0.3)

    # 3. Confidence intervals (first 20 observations)
    sample_df = results_df.head(20)
    y_pos = range(len(sample_df))

    # 95% CI
    axes[0, 2].errorbar(sample_df['predicted_mean_corrected'], y_pos,
                        xerr=[sample_df['predicted_mean_corrected'] - sample_df['ci_95_low
                              sample_df['ci_95_upper'] - sample_df['predicted_mean_correct
                        fmt='bo', alpha=0.7, label='95% ДИ')

    # Actual values
    axes[0, 2].scatter(sample_df['actual_price'], y_pos, color='red', s=50, label='Фактич

    axes[0, 2].set_xlabel('Цена (руб.)')
```

```
    axes[0, 2].set_ylabel('Наблюдение')
    axes[0, 2].set_title('Доверительные интервалы (первые 20)')
    axes[0, 2].legend()
    axes[0, 2].grid(True, alpha=0.3)

    # 4. Residuals histogram
    axes[1, 0].hist(results_df['residuals_corrected'], bins=30, alpha=0.7, edgecolor='bla
    axes[1, 0].axvline(x=0, color='r', linestyle='--')
    axes[1, 0].set_xlabel('Остатки (руб.)')
    axes[1, 0].set_ylabel('Частота')
    axes[1, 0].set_title('Распределение остатков')
    axes[1, 0].grid(True, alpha=0.3)

    # 5. Residuals by district
    districts = results_df['district'].unique()
    district_residuals = [results_df[results_df['district'] == d]['residuals_corrected'].
                          for d in districts]
    axes[1, 1].boxplot(district_residuals, labels=districts)
    axes[1, 1].axhline(y=0, color='r', linestyle='--')
    axes[1, 1].set_xlabel('Район')
    axes[1, 1].set_ylabel('Остатки (руб.)')
    axes[1, 1].set_title('Остатки по районам')
    axes[1, 1].grid(True, alpha=0.3)
    plt.setp(axes[1, 1].get_xticklabels(), rotation=45)

    # 6. Uncertainty vs Error
    axes[1, 2].scatter(results_df['uncertainty_percent'],
                       np.abs(results_df['residuals_corrected']), alpha=0.6)
    axes[1, 2].set_xlabel('Неопределенность (%)')
    axes[1, 2].set_ylabel('Абсолютная ошибка (руб.)')
    axes[1, 2].set_title('Неопределенность vs Ошибка')
    axes[1, 2].grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

# Создаем визуализацию
plot_uncertainty_analysis(results_df)
```

Ключевые особенности этого подхода:

1. **Log-трансформация целевой переменной** улучшает нормальность распределения
2. **RMSEWithUncertainty** предоставляет оценки дисперсии для каждого предсказания
3. **Коррекция смещения** при обратной трансформации учитывает свойства логнормального распределения
4. **Доверительные интервалы** вычисляются корректно для логнормального распределения
5. **Интерпретация в исходном масштабе** делает результаты понятными для заказчика
6. **Virtual Ensemble** (если доступен) дает дополнительную информацию о неопределенности

Этот подход особенно эффективен для данных с правосторонней асимметрией (как цены недвижимости) и когда требуется количественная оценка неопределенности предсказаний.