In [42]:
```
import numpy as np
import pandas as pd

from sklearn.model_selection import KFold, RandomizedSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

from catboost import CatBoostRegressor, Pool
import lightgbm as lgb
from lightgbm import Dataset, LGBMRegressor

import xgboost as xgb
from xgboost import XGBRegressor
from xgboost import XGBRegressor, DMatrix
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
train = pd.read_csv('https://raw.githubusercontent.com/a-milenkin/Competi
test = pd.read_csv('https://raw.githubusercontent.com/a-milenkin/Competit
```

In [3]:
```
RANDOM_STATE = 42
```

In [4]:
```
results = [] #for saving results for each model
```

In [45]:
```
def train_model(algorithm,
                X,
                y,
                early_stopping_rounds,
                init_params = None,
                cat_features = None,
                random_seed = 2023
                ):

    scores = []
    models = []

    kf = KFold(n_splits=3, shuffle=True, random_state=random_seed)

    print(f'=========Training algorithm is {algorithm.__name__}==========

    for num_fold, (train_idx,val_idx) in  enumerate(kf.split(X)):
        X_train, X_eval = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_eval = y.iloc[train_idx], y.iloc[val_idx]

        if init_params is not None:
            model = algorithm(**init_params)
        else:
            model = algorithm()

        if algorithm.__name__ == 'CatBoostRegressor':
            train_dataset = Pool(X_train, y_train, cat_features=cat_featu
            eval_dataset = Pool(X_eval, y_eval, cat_features=cat_features

            model.fit(train_dataset,
                      eval_set=eval_dataset,
                      verbose=0,
                      early_stopping_rounds=early_stopping_rounds)
```

```
        elif algorithm.__name__ == 'LGBMRegressor':
            train_dataset = Dataset(X_train, y_train)
            eval_dataset = Dataset(X_eval, y_eval)

            model = lgb.train(params=init_params,
                              train_set=train_dataset,
                              valid_sets=(eval_dataset),
                              categorical_feature=cat_features,
                              )

        elif algorithm.__name__ == 'XGBRegressor':

            train_dataset = DMatrix(X_train, y_train)
            eval_dataset = DMatrix(X_eval, y_eval)

            model = xgb.train(params=init_params,
                              dtrain = train_dataset,
                              evals = [(train_dataset, 'dtrain'), (eval_d
                              verbose_eval = False,
                              early_stopping_rounds = early_stopping_roun


        # Сделайте предсказание на X_eval и посчитайте RMSE
        y_pred = model.predict(eval_dataset)
        score = np.sqrt(mean_squared_error(y_pred, y_eval))

        models.append(model)
        scores.append(score)

        print(f'Fold {num_fold} : Score {score}')

    mean_kfold_score = np.mean(scores) - np.std(scores)
    print('Mean RSME Score', mean_kfold_score)

    best_model = models[np.argmin(scores)]

    return mean_kfold_score, best_model
```

In [6]:
```
def tuning_hyperparams(algorithm,
                       X,
                       y,
                       init_params,
                       fit_params,
                       grid_params,
                       n_iter,
                       cv=3,
                       random_state=2023):

    estimator = algorithm(**init_params)

    model = RandomizedSearchCV(estimator=estimator,
                               param_distributions=grid_params,
                               n_iter=n_iter,
                               cv=cv,
                               scoring='neg_root_mean_squared_error',
                               n_jobs=-1,
                               verbose=0,
                               random_state=random_state)
```

```
    model.fit(X, y, **fit_params)

    return init_params | model.best_params_
```

In [7]:
```
target = ['target_reg']
features2drop = ['car_id','target_class']
columns_to_drop = target + features2drop
cat_features =  train.drop(columns=columns_to_drop).select_dtypes(include


filtered_features = [i for i in train.columns if (i not in features2drop)
num_features = [i for i in filtered_features if (i not in cat_features)]

print(f'filtered_features: {filtered_features}')
print(f'num_features: {num_features}')
print(f'cat_features: {cat_features}')
```

```
filtered_features: ['model', 'car_type', 'fuel_type', 'car_rating', 'year_
to_start', 'riders', 'year_to_work', 'target_reg', 'mean_rating', 'distanc
e_sum', 'rating_min', 'speed_max', 'user_ride_quality_median', 'deviation_
normal_count', 'user_uniq']
num_features: ['car_rating', 'year_to_start', 'riders', 'year_to_work', 't
arget_reg', 'mean_rating', 'distance_sum', 'rating_min', 'speed_max', 'use
r_ride_quality_median', 'deviation_normal_count', 'user_uniq']
cat_features: ['model', 'car_type', 'fuel_type']
```

In [33]:
```
X = train[filtered_features].drop(target, axis=1, errors="ignore")
y = train[target]
print(f"Filtered features after drop: {filtered_features}")
```

```
Filtered features after drop: ['model', 'car_type', 'fuel_type', 'car_rati
ng', 'year_to_start', 'riders', 'year_to_work', 'target_reg', 'mean_ratin
g', 'distance_sum', 'rating_min', 'speed_max', 'user_ride_quality_median',
'deviation_normal_count', 'user_uniq']
```

In [9]:
```
cb_init_params = {
    'loss_function': 'RMSE',
    'eval_metric': 'RMSE',
    'thread_count': -1,
    'task_type': 'CPU',
    'random_seed': RANDOM_STATE
}

cb_score, cb_model = train_model(
    algorithm=CatBoostRegressor,
    X=X,
    y=y,
    init_params=cb_init_params,
    early_stopping_rounds=50,
    random_seed=RANDOM_STATE,
    cat_features=cat_features
)
```

```
=========Training algorithm is CatBoostRegressor=============
Fold 0 : Score 11.926355935511776
Fold 1 : Score 11.882241144706034
Fold 2 : Score 11.36731068554355
Mean RSME Score 11.471524232962208
```

In [34]:
```
filtered_features_for_test = [feature for feature in filtered_features if
X_test = test[filtered_features_for_test]
```

```
cb_test_pred = cb_model.predict(X_test)
pd.DataFrame({'car_id': test['car_id'], 'target_reg': cb_test_pred}).to_c
```

In [11]:
```
results.append({
    'model_name': 'CatBoostRegressor',
    'tuning': False,
    'kfold_score': cb_score,
    'leaderboard_score': 'RMSE=11.9',
    'model': cb_model
})
```

In [12]:
```
cb_fit_params  = {
    'cat_features': cat_features,
    'verbose': 0,
    'early_stopping_rounds': 50}

cb_grid_params = {
    'depth': [4, 6, 8, 10],
    'learning_rate': [0.01, 0.05, 0.1],
    'l2_leaf_reg': [1, 3, 5],
    'iterations': [500, 1000],
    'rsm': [0.8, 0.9, 1.0],
    'border_count': [32, 64, 128]

}
```

In [13]:
```
cb_params_after_tuning = tuning_hyperparams(CatBoostRegressor,
                                            X=X,
                                            y=y,
                                            init_params=cb_init_params,
                                            fit_params=cb_fit_params,
                                            grid_params=cb_grid_params,
                                            n_iter=20
                                            )
cb_params_after_tuning
```

Out[13]:
```
{'loss_function': 'RMSE',
 'eval_metric': 'RMSE',
 'thread_count': -1,
 'task_type': 'CPU',
 'random_seed': 42,
 'rsm': 1.0,
 'learning_rate': 0.01,
 'l2_leaf_reg': 3,
 'iterations': 1000,
 'depth': 8,
 'border_count': 64}
```

In [14]:
```
cb_tuning_score, cb_tuning_model = train_model(CatBoostRegressor,
                X,
                y,
                early_stopping_rounds = 50,
                init_params = cb_params_after_tuning,
                cat_features = cat_features,
                random_seed = 2023
                )
```

```
=========Training algorithm is CatBoostRegressor=============
Fold 0 : Score 10.924275519995549
Fold 1 : Score 12.584374819700574
Fold 2 : Score 11.57913045597363
Mean RSME Score 11.013180769293538
```

In [15]:
```
tuning_cb_test_pred = cb_tuning_model.predict(X_test)
pd.DataFrame({'car_id': test['car_id'], 'target_reg': cb_test_pred}).to_c
```

In [16]:
```
results.append({
    'model_name': 'CatBoostRegressor',
    'tuning': True,
    'kfold_score': cb_tuning_score,
    'leaderboard_score': 'RMSE=11.9',
    'model': cb_tuning_model
})
```

In [17]:
```
cat_features
```

Out[17]:
```
['model', 'car_type', 'fuel_type']
```

6️⃣ LightGBMRegressor (goss).

In [18]:
```
# Сохраняем LabelEncoder для каждой колонки
encoders = {}
X_lgb = X.copy()

# Применяем LabelEncoder к каждой колонке в cat_features
for col in cat_features:
    le = LabelEncoder()
    X_lgb[col] = le.fit_transform(X[col])
    encoders[col] = le

X_test_lgb = X_test.copy()
for col in cat_features:
    le = encoders[col]
    X_test_lgb[col] = X_test[col].map(lambda s: le.transform([s])[0] if s
```

In [19]:
```
lgb_init_params = {
    'boosting_type': 'gbdt',
    'n_jobs': -1,
    'metric': 'RMSE',
    'objective': 'regression',
    'random_state': RANDOM_STATE,
    'verbosity': -1,
    'device': 'cpu'
}
```

In [20]:
```
lgb_score, lgb_model = train_model(LGBMRegressor,
                                   X_lgb,
                                   y,
                                   early_stopping_rounds=50,
                                   init_params=lgb_init_params,
                                   cat_features=cat_features,
                                   random_seed=RANDOM_STATE)
```

```
=========Training algorithm is LGBMRegressor=============
Fold 0 : Score 12.352780364094844
Fold 1 : Score 12.345895813022121
Fold 2 : Score 11.972325500395383
Mean RSME Score 12.045919564864533
```

In [21]:
```
lgb_test_pred = lgb_model.predict(X_test_lgb)
pd.DataFrame({'car_id': test['car_id'], 'target_reg': lgb_test_pred}).to_
```

In [23]:
```
results.append({
    'model_name': 'LGBMRegressor (goss)',
    'tuning': False,
    'mean_kfold_score': lgb_score,
    'leaderboard_score': 'RMSE=11.9',
    'model': lgb_model
})
```

Подбор гиперпараметров и обучение модели с новыми параметрами

In [28]:
```
from tqdm import tqdm
lgb_fit_params = {
    'eval_metric': 'rmse',
    'categorical_feature': cat_features
}

lgb_grid_params = {
    'max_depth': [3, 5, 7, 10, -1],
    'min_data_in_leaf': [10, 20, 30, 50],
    'num_leaves': [20, 31, 40, 50],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'feature_fraction': [0.6, 0.8, 1.0],
    'bagging_fraction': [0.6, 0.8, 1.0],
    'bagging_freq': [0, 5, 10],
    'lambda_l1': [0.0, 0.1, 1.0],
    'lambda_l2': [0.0, 0.1, 1.0],
    'boosting_type': ['gbdt', 'dart'],
    'n_estimators': [100, 200, 500]
}

lgb_params_after_tuning = tuning_hyperparams(
    algorithm=LGBMRegressor,
    X=X_lgb,
    y=y,
    init_params=lgb_init_params,
    grid_params=lgb_grid_params,
    fit_params=lgb_fit_params,
    n_iter=50,
    cv=5
)
lgb_params_after_tuning
```

Out[28]:
```
{'boosting_type': 'gbdt',
 'n_jobs': -1,
 'metric': 'RMSE',
 'objective': 'regression',
 'random_state': 42,
 'verbosity': -1,
 'device': 'cpu',
 'num_leaves': 20,
 'n_estimators': 500,
 'min_data_in_leaf': 10,
 'max_depth': 10,
 'learning_rate': 0.01,
 'lambda_l2': 1.0,
 'lambda_l1': 0.0,
 'feature_fraction': 0.8,
 'bagging_freq': 5,
 'bagging_fraction': 1.0}
```

In [29]:
```
lgb_tuning_score, lgb_tuning_model = train_model(
    algorithm=LGBMRegressor,
    X=X_lgb, y=y,
    init_params=lgb_params_after_tuning,
    early_stopping_rounds=50,
    cat_features=cat_features,
    random_seed=RANDOM_STATE
)
```

```
=========Training algorithm is LGBMRegressor=============
Fold 0 : Score 12.11997131214823
Fold 1 : Score 11.924731240074463
Fold 2 : Score 11.46437582241237
Mean RSME Score 11.561515817314175
```

In [31]:
```
lgb_tuning_test_pred = lgb_tuning_model.predict(X_test_lgb)
pd.DataFrame({'car_id': test['car_id'], 'target_reg': lgb_test_pred}).to_
```

7️⃣ XGBoostRegressor (dart).

In [43]:
```
encoders = {}
X_xgb = X.copy()

# Применяем LabelEncoder к каждой колонке в cat_features
for col in cat_features:
    le = LabelEncoder()
    X_xgb[col] = le.fit_transform(X_xgb[col])
    encoders[col] = le

X_test_xgb = X_test.copy()
for col in cat_features:
    le = encoders[col]
    X_test_xgb[col] = X_test_xgb[col].map(lambda s: le.transform([s])[0]
```

In [46]:
```
xgb_init_params = {
    'enable_categorical': True,
    'booster': 'dart',
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
```

```
    'random_state': RANDOM_STATE,
    'n_jobs': -1,
    'verbosity': 0,


    'rate_drop': 0.1,
    'skip_drop': 0.5,
}


xgb_score, xgb_model = train_model(
    algorithm=XGBRegressor,
    X=X_xgb, y=y,
    init_params=xgb_init_params,
    early_stopping_rounds=50,
    cat_features=cat_features,
    random_seed=RANDOM_STATE
)
```

```
=========Training algorithm is XGBRegressor=============
Fold 0 : Score 12.321854625996112
Fold 1 : Score 12.343150719988948
Fold 2 : Score 11.681670723608326
Mean RSME Score 11.808630406871782
```

In [48]:
```
xgb_test_pred = xgb_model.predict(DMatrix(X_test_xgb))

pd.DataFrame({'car_id': test['car_id'], 'target_reg': xgb_test_pred}).to_
```

In [49]:
```
results.append({
    'model_name': 'XGBRegressor (dart)',
    'tuning': False,
    'mean_kfold_score': xgb_score,
    'leaderboard_score': 'RMSE 12.2',
    'model': xgb_model
})
```

Подбор гиперпараметров и обучение модели с новыми параметрами

In [52]:
```
xgb_grid_params = {
    'max_depth': [3, 5, 7, 10],            # Глубина дерева
    'max_leaves': [0, 31, 50, 100],        # Максимальное количество лист
    'learning_rate': [0.01, 0.05, 0.1],    # Темп обучения
    'n_estimators': [100, 200, 300],       # Количество деревьев
    'subsample': [0.8, 0.9, 1.0],          # Доля выборки для каждого дер
    'colsample_bytree': [0.6, 0.8, 1.0],   # Доля признаков для каждого д
}


xgb_fit_params = {
    'verbose': False
}

xgb_params_after_tuning = tuning_hyperparams(algorithm=XGBRegressor,
                                             X=X_xgb, y=y,
                                             init_params=xgb_init_params,
                                             fit_params=xgb_fit_params,
```

```
                                                grid_params=xgb_grid_params,
                                                n_iter=30,
                                                cv=5,
                                                random_state=RANDOM_STATE
)

xgb_params_after_tuning
```

Out[52]:
```
{'enable_categorical': True,
 'booster': 'dart',
 'objective': 'reg:squarederror',
 'eval_metric': 'rmse',
 'random_state': 42,
 'n_jobs': -1,
 'verbosity': 0,
 'rate_drop': 0.1,
 'skip_drop': 0.5,
 'subsample': 0.8,
 'n_estimators': 300,
 'max_leaves': 0,
 'max_depth': 5,
 'learning_rate': 0.05,
 'colsample_bytree': 1.0}
```

In [54]:
```
xgb_tuning_score, xgb_tuning_model = train_model(
    algorithm=XGBRegressor,
    X=X_xgb, y=y,
    init_params=xgb_params_after_tuning,
    early_stopping_rounds=50,
    cat_features=cat_features,
    random_seed=RANDOM_STATE
)
```

```
=========Training algorithm is XGBRegressor=============
Fold 0 : Score 14.235536385595745
Fold 1 : Score 15.125368915878502
Fold 2 : Score 14.183297274894453
Mean RSME Score 14.082423882612263
```

In [55]:
```
tuning_xgb_test_pred = xgb_tuning_model.predict(DMatrix(X_test_xgb))

pd.DataFrame({'car_id': test['car_id'], 'target_reg': tuning_xgb_test_pre
```

In [56]:
```
results.append({
    'model_name': 'XGBRegressor (dart)',
    'tuning': True,
    'mean_kfold_score': xgb_tuning_score,
    'leaderboard_score': 'RMSE 14.5',
    'model': xgb_tuning_model
})
```

In [57]:
```
results
```

Out[57]:
```
[{'model_name': 'CatBoostRegressor',
  'tuning': False,
  'kfold_score': 11.471524232962208,
  'leaderboard_score': 'RMSE=11.9',
  'model': <catboost.core.CatBoostRegressor at 0x106c33280>},
 {'model_name': 'CatBoostRegressor',
  'tuning': True,
  'kfold_score': 11.013180769293538,
  'leaderboard_score': 'RMSE=11.9',
  'model': <catboost.core.CatBoostRegressor at 0x106c328c0>},
 {'model_name': 'LGBMRegressor (goss)',
  'tuning': False,
  'mean_kfold_score': 12.045919564864533,
  'leaderboard_score': 'RMSE=11.9',
  'model': <lightgbm.basic.Booster at 0x17613ec20>},
 {'model_name': 'XGBRegressor (dart)',
  'tuning': False,
  'mean_kfold_score': 11.808630406871782,
  'leaderboard_score': 'RMSE 12.2',
  'model': <xgboost.core.Booster at 0x30e91e920>},
 {'model_name': 'XGBRegressor (dart)',
  'tuning': True,
  'mean_kfold_score': 14.082423882612263,
  'leaderboard_score': 'RMSE 14.5',
  'model': <xgboost.core.Booster at 0x30e9e1d20>}]
```

In [58]:
```
best_cb_model = cb_model
best_cb_model.save_model('best_cb_model.cbm')

best_lgb_model = lgb_model
best_lgb_model.save_model('best_lgb_model.mod')


best_xgb_model = xgb_model
best_xgb_model.save_model('best_xgb_model.json')
```

In [61]:
```
final_pred = (cb_model.predict(X_test) + lgb_model.predict(X_test_lgb) +
```

In [62]:
```
final_pred
```

Out[62]:
```
array([45.08659275, 33.12411749, 32.16873894, ..., 35.14593856,
       46.32541362, 48.44395433])
```

In [63]:
```
pd.DataFrame({'car_id': test['car_id'], 'target_reg': final_pred}).to_csv
```

In [ ]:
```

```