DAFTAR PUSTAKA

Adinugroho, R., 2022. Perbandingan Rasio Split Data Training dan Data Testing menggunakan Metode LSTM dalam Memprediksi Harga Indeks Saham Asia. Bachelor’s thesis, UIN Syarif Hidayatullah, Jakarta.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M., 2019. Optuna: A Next Generation Hyperparameter Optimization Framework. ACM SIGKDD. 2623-2631.

Deshtiara, D., 2023. Prediksi Suhu dan Kekuatan Angin pada Model El Nino menggunakan Algoritma Light Gradient Boosting Machine. Bachelor’s thesis, UIN Syarif Hidayatullah, Jakarta.

Fauzi, A., 2019. Forecasting Saham Syariah dengan menggunakan LSTM. Al-Masraf: Jurnal Lembaga Keuangan dan Perbankan. 4(1), 65-69.

Febriantoro, E., Setyati, E., dan Santoso, J., 2023. Pemodelan Prediksi Kuantitas Penjualan Mainan menggunakan LightGBM. SMARTICS Journal, 9(1).

Friedman, J. H., 2001. Greedy Function Approximation: A Gradient Boosting Machine. Annals of statistics, 1189-1232.

Herdianto. 2013. Prediksi Kerusakan Motor Induksi menggunakan Metode Jaringan Saraf Tiruan Backpropagation. Master thesis. Universitas Sumatera Utara, Medan.

Hermuningsih, S., 2019. Pengantar Pasar Modal Indonesia. STIM YKPN, Yogyakarta.

Ke, G., Meng, Q., Finlel, T., Wang, T., Chen, W., Ma, W., and Liu, T. Y., 2017. Lightgbm: A Highly Efficient Gradient Boosting Decision Tree. Neural Information Processing System, 30.

Makridakis, S., Wheelwright, S. C., dan McGee, V. E., 1999. Metode dan Aplikasi Peramalan. Erlangga, Jakarta.

Prihanditya, H. A., 2020. The Implementation of Z-Score Normalization and Boosting Techniques to Increase Accuracy of C4. 5 Algorithm in Diagnosing Chronic Kidney Disease. Journal of Soft Computing Exploration, 1(1), 63-69.

Probst, P., Boulesteix, A. L., and Bischl, B., 2019. Turnability: Importance of Hyperparameters of Machine Learning Algorithms. Journal of Machine Learning Research. 20(1), 1934-1965.

Rizky, P. S., Hirzi, R.H., dan Hidayaturrohman, U. (2022). Perbandingan Metode LightGBM dan XGBoost dalam Menangani Data dengan Kelas Tidak Seimbang. J Statistika: Jurnal Ilmiah Teori dan Aplikasi Statistika, 15(2), 2228-236

Rufo, D. D., Debelee, T. G., Ibenthal, A., and Negera, W. G., 2021. Diagnosis of Diabetes Mellitus Using Gradient Boosting Machine (LightGBM). Diagnostics. 11(9), 1714.

Siringoringo, Z., 2021. Prediksi Tingkat Inflasi Nasional menggunakan Metode Gated Recurrent Unit. Doctoral dissertation. Universitas Sumatera Utara, Medan.

Srinivas, P., and Katarya, R., 2022. Optuna Hyperparameter Optimization Framework for Predicting Cardiovascular Disease Using XGBoost. Biomedical Signal Processing and Control. 73, 103456.

Sukartaatmadja, I., Khim, S., dan Lestari, M. N., 2023. Faktor-faktor yang memengaruhi Harga Saham Perusahaan. Jurnal Ilmiah Manajemen Kesatuan. 11(1), 21-40.

Sun, X., Liu, M., and Sima, Z., 2020. A Novel Cryptocurrency Price Trend Forecasting Model Based on LightGBM. Finance Research Letters. 32, 101084.

Suyudi, M. A. D., Djamal, A. C., dan Maspupah, A., 2019. Prediksi Harga Saham menggunakan Metode Recurrent Neural Network. SNATI. 33-38.

Tussifah, S. A., 2020. Analisis Perbandingan Kinerja Model ARIMA, LSTM dan GRU pada Stock Price Forecasting. Bachelor's thesis. Universitas Islam Negeri Hidayatullah, Jakarta.

Yang, Y., Wu, Y., Wang, P., and Jiali, X., 2021. Stock Price Prediction Based on XGBoost and LightGBM. EDP Sciences. 275, 1040.

Yu, T., and Zhu, H., 2020. Hyper-parameter Optimization: A Review of Algorithms and Applications. ARXIV. 5689.

Zaidan, M., dan Garinas, W., 2021. Kajian Bahan Baku Mineral Nikel untuk Baterai Listrik di Daerah Sulawesi Tenggara. Jurnal Rekayasa Pertambangan. 1(1).

Zhang, Y., Zhu, C., and Wang, Q., 2020. LightGBM-based Model for Metro Passenger Volume Forecasting. IET Intelligent Transport Systems. 14(13), 1815-1823.

LAMPIRAN

Lampiran 1 : Dataset Harga Saham PT. Vale Indonesia Tbk.

Keseluruhan dataset yang digunakan dalam penelitian ini dapat diakses pada link berikut:

https://finance.yahoo.com/quote/INCO.JK/history

Lampiran 2: Source Code Python

Berikut lampiran *source code python* dalam membuat model *Light Gradient Boosting Machine:*

- *Library* yang digunakan

```
import os
import numpy as np
import pandas as pd
from google.colab import drive
from tqdm.auto import tqdm
import joblib
import time
import datetime
import matplotlib.pyplot as plt
from pathlib import Path
import matplotlib.style as style
import random
import seaborn as sns
from matplotlib import pyplot
from matplotlib.ticker import ScalarFormatter
import lightgbm as lgb
from lightgbm import LGBMRegressor
from lightgbm import early_stopping
from lightgbm import Dataset
from lightgbm import callback
import optuna
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error
from sklearn.datasets import make_regression
from sklearn.preprocessing import MinMaxScaler
from tabulate import tabulate
import math
import warnings
```

```
import logging
import json
```

- Import Dataset

```
data = pd.read_csv('/content/drive/MyDrive/Dataset/INCO.JK
(4).csv')
data.info()
```

- *Reduce Memory*

```
#Reduce Memory
def reduce_mem_usage(df):
    start_mem = df.memory_usage().sum() / 1024**2
    print('Penggunaan memori awal adalah {:.2f}
MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max <
np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max
< np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max
< np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max
< np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max
< np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and
c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
```

```
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Penggunaan memori setelah optimasi adalah: {:.2f} MB'.format(end_mem))
    print('Menurun hingga {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df

data = reduce_mem_usage(data)
```

- Data *Transform*

```
# Transform Data
data['Date'] = data['Date'].astype('datetime64[ns]')
```

```
data = pd.read_csv('/content/drive/MyDrive/Dataset/INCO.JK (4).csv', index_col='Date', parse_dates=['Date'])
data.head()
```

```
df = data.drop(['Adj Close'], axis=1)
df.head()
```

- *Missing Value*

```
# missing values?
df.isna().sum()
```

```
df.describe()
```

- Visualisasi masing-masing harga saham

```
#Grafik data open
plt.figure(figsize=(8,5))
plt.title('Grafik Data Open')
plt.plot(df['Open'])

plt.xlabel('Date')
plt.ylabel('Price')
```

```
currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()

#grafik data high
plt.figure(figsize=(8,5))
plt.title('Grafik Data High')
plt.plot(df['High'])

plt.xlabel('Date')
plt.ylabel('Price')

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()

#grafik data low
plt.figure(figsize=(8,5))
plt.title('Grafik Data Low')
plt.plot(df['Low'])

plt.xlabel('Date')
plt.ylabel('Price')

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()

#grafik data close
plt.figure(figsize=(8,5))
plt.title('Grafik Data Close')
plt.plot(df['Close'])

plt.xlabel('Date')
plt.ylabel('Price')

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()

#grafik data volume
plt.figure(figsize=(8,5))
plt.title('Grafik Data Volume')
plt.plot(df['Volume'])

plt.xlabel('Date')
```

```
plt.ylabel('Volume')

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()
```

- Penentuan *feature* dan *target*

```
# Menentukan fitur dan target
features = ['Open', 'High', 'Low', 'Volume']
target = ['Close']
```

- Split Data

```
# Split data into train (80%) and test (20%)
def dataset(dataframe):
    train_size = int(len(dataframe)*0.8)
    train_dataset, test_dataset =
                                dataframe.iloc[:train_size],
                                dataframe.iloc[train_size:]
    return train_dataset, test_dataset

train_dataset, test_dataset = dataset(df)
```

```
# Membagi data menjadi set pelatihan dan pengujian
X_train = train_dataset[features]
y_train = train_dataset[target]

X_test = test_dataset[features]
y_test = test_dataset[target]

print(f"Ukuran set pelatihan: {len(X_train)}")
print(f"Ukuran set pengujian: {len(X_test)}")
```

```
# Membuat dataset LightGBM
train_data = lgb.Dataset(X_train, y_train)
test_data = lgb.Dataset(X_test, label=y_test,
reference=train_data)
```

```
# Plot train dan test data
plt.figure(figsize = (10, 5))
plt.rcParams['figure.dpi'] = 240
plt.plot(train_dataset.Close, linestyle="-")
plt.plot(test_dataset.Close, linestyle="-")
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price', fontsize=12)
plt.title('Plot Harga Saham', fontsize=12)
plt.legend(['Data Latih', 'Data Uji'], loc='upper left',
prop={'size': 10})
print('Dimension of train data: ', train_dataset.shape)
print('Dimension of test data: ', test_dataset.shape)
```

- Penentuan Parameter LightGBM

```
params = {
    'objective': 'regression',
    'metric': 'rmse',
    'boosting_type': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.1,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': -1
}

evals_result = {}
```

- *Training* Model

```
#Implementasi LightGBM
def train_model_LGBM(train_data, test_data, X_train,
categorical_feature_index=21):

#Train Model
  bst = lgb.train(
    params,
    train_data,
    valid_sets=[train_data, test_data],
    num_boost_round=100,
    feature_name=[f'f{i + 1}' for i in
range(X_train.shape[-1])],
    categorical_feature=[21],
    callbacks=[
```

```
        lgb.record_evaluation(evals_result)
    ]
  )

  bst.save_model('model.txt')

  y_pred = bst.predict(X_train,
num_iteration=bst.best_iteration)

  print('Plotting metrics recorded during training...')
  ax = lgb.plot_metric(evals_result, metric='rmse')
  plt.show()

  print('Plotting feature importances...')
  ax = lgb.plot_importance(bst, max_num_features=10)
  plt.show()

  print('Plotting 54th tree...')
  ax = lgb.plot_tree(bst, tree_index=53, figsize=(15, 15),
show_info=['split_gain'])
  plt.show()

  print('Plotting 54th tree with graphviz...')
  graph = lgb.create_tree_digraph(bst, tree_index=53,
name='Tree54')
  graph.render(view=True)

  print('Plot Predictid Matrix')
  y_test_sorted = y_train.sort_index()
  y_pred_sorted = pd.Series(y_pred,
index=y_test_sorted.index)

# Plotting
  plt.figure(figsize=(8, 5))
  plt.plot(y_test_sorted, color='blue', label='Actual
Prices')
  plt.plot(y_pred_sorted, color='red', label='Predicted
Prices')
  plt.title('Stock Price Prediction')
  plt.xlabel('Date')
  plt.ylabel('Stock Price')
  plt.legend()
  plt.show()

  print("Model Evaluation:")
  rmse = np.sqrt(mean_squared_error(y_train, y_pred))
  mape = mean_absolute_percentage_error(y_train, y_pred)
```

```
  print( 'RMSE: %2.2f' % rmse)
  print('MAPE: %.2f%%' % (mape*100))

  return bst, y_pred, rmse, mape
```

- *Hyperparameter Tuning*

```
# Definisi fungsi objektif untuk Optuna
def objective(trial):
    params = {
        'objective': 'regression',
        'metric': 'rmse',
        'boosting_type': 'gbdt',
        'num_leaves': trial.suggest_int('num_leaves', 2,
256),
        'learning_rate':
trial.suggest_loguniform('learning_rate', 0.005, 0.5),
        'feature_fraction':
trial.suggest_uniform('feature_fraction', 0.1, 1.0),
        'bagging_fraction':
trial.suggest_uniform('bagging_fraction', 0.1, 1.0),
        'bagging_freq': trial.suggest_int('bagging_freq',
1, 10),
        'n_estimators': trial.suggest_int('n_estimators',
100, 1000, step=50),
        'verbose': -1
    }

    evals_result = {}

    bst = lgb.LGBMRegressor(**params)
    bst.fit(X_train, y_train)

    y_pred = bst.predict(X_train)
    mse = mean_squared_error(y_train, y_pred)
    rmse = np.sqrt(mse)

    return rmse
```

```
# Optimasi hyperparameter dengan Optuna
optuna.logging.set_verbosity(optuna.logging.ERROR)
```

```
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100)

best_params = study.best_params
best_score = study.best_value

print('Number of finished trials:', len(study.trials))
print('Best trial:', study.best_params)
print('Best score:', study.best_value)
```

```
# Model akhir dengan parameter terbaik
def LGBM_with_optuna(X_train, y_train):
  model = lgb.LGBMRegressor(**best_params)
  model.fit(X_train, y_train)

  warnings.filterwarnings("default")

# Prediksi harga saham
  prediction = model.predict(X_train)

# Plot pentingnya fitur untuk model terbaik dengan optuna
  print('Plotting metrics recorded during training...')
  lgb.plot_importance(model, figsize=(10, 6),
title='Feature Importance (Best Model)')
  plt.show()

  print('Plotting 54th tree...')
  ax = lgb.plot_tree(model, tree_index=53, figsize=(15,
15), show_info=['split_gain'])
  plt.show()

  print('Plot Predictid Matrix')
  test_sorted = y_train.sort_index()
  prediction_sorted = pd.Series(prediction,
index=test_sorted.index)

# Plotting
  plt.figure(figsize=(10, 6))
  plt.plot(test_sorted, color='blue', label='Actual
Prices')
  plt.plot(prediction_sorted, color='red',
label='Predicted Prices')
  plt.title('Stock Price Prediction')
  plt.xlabel('Date')
  plt.ylabel('Stock Price')
```

```
  plt.legend()
  plt.show()

  print("Model Evaluation:")
  # Evaluasi model
  mse_optuna = mean_squared_error(y_train, prediction)
  rmse_optuna = np.sqrt(mse_optuna)
  mape_optuna = mean_absolute_percentage_error(y_train,
prediction)

# Print hasil
  print(f"Best Parameters: {best_params}")
  print('RMSE: %2.2f' % rmse_optuna)
  print('MAPE: %.2f%%' % (mape_optuna*100))

  return model, predictions, rmse_optuna, mape_optuna
```

- Hasil Prediksi dengan Data Uji

```
# Model akhir dengan parameter terbaik
def final_model_LGBM(X_test, y_test):
  final_model = lgb.LGBMRegressor(**best_params)
  final_model.fit(X_test, y_test)

  warnings.filterwarnings("default")

# Prediksi harga saham
  predictions = final_model.predict(X_test)

# Plot pentingnya fitur untuk model terbaik dengan optuna
  print('Plotting metrics recorded during training...')
  lgb.plot_importance(final_model, figsize=(10, 6),
title='Feature Importance (Best Model)')
  plt.show()

  print('Plotting 54th tree...')
  ax = lgb.plot_tree(final_model, tree_index=53,
figsize=(15, 15), show_info=['split_gain'])
  plt.show()

  print('Plot Predictid Matrix')
  test_sorted = y_test.sort_index()
  prediction_sorted = pd.Series(predictions,
index=test_sorted.index)
```

```
# Plotting
  plt.figure(figsize=(10, 6))
  plt.plot(test_sorted, color='blue', label='Actual
Prices')
  plt.plot(prediction_sorted, color='red',
label='Predicted Prices')
  plt.title('Stock Price Prediction')
  plt.xlabel('Date')
  plt.ylabel('Stock Price')
  plt.legend()
  plt.show()

  print(' Data Prediction :')
  print(prediction_sorted)

  print('Data Actual : ')
  print(test_sorted)

  print("Model Evaluation:")
  # Evaluasi model
  mse_final = mean_squared_error(y_test, predictions)
  rmse_final = np.sqrt(mse_final)
  mape_final = mean_absolute_percentage_error(y_test,
predictions)

# Print hasil
  print(f"Best Parameters: {best_params}")
  print('RMSE: %2.2f' % rmse_final)
  print('MAPE: %.2f%%' % (mape_final*100))

  return final_model, predictions, rmse_final, mape_final
```