Analisis Penjualan di Cabang Toko Serba Ada dengan Algoritma Machine Learning

Rico Halim[#1], Hendra Bunyamin[#2]

[#]*Program Studi S1 Teknik Informatika, Universitas Kristen Maranatha*
Jalan Surya Sumantri No. 65, Bandung, Indonesia
[1]ricohalim999@gmail.com
[2]hendra.bunyamin@it.maranatha.edu

Abstract — **This research aims to analyze the factors influencing sales performance in a convenience store. The main focus of this study is to identify the factors affecting sales based on location zones and demographic characteristics. The research methodology involves collecting sales data from the convenience store over a period of time, as well as gathering data on store locations and zone demographics. Based on the collected data, various machine learning models such as multiple linear regression, Ridge Regression, Lasso, Elastic-Net, Bayesian Regression, Support Vector Machines (SVM), Gradient Boosting, and Random Forest are employed to analyze the relationship between the factors influencing sales. The scientific findings of this research provide a better understanding of the contributing factors to sales performance in the convenience store. These analysis results can guide store management in improving sales by optimizing the location and marketing strategies tailored to the local demographic characteristics.**

Keywords— **Factor analysis, Demographics, Sales performance, Location, Machine learning, Convenience store.**

I. PENDAHULUAN

Kinerja penjualan dari suatu toko serba ada dipengaruhi oleh berbagai faktor. Secara umum, faktor-faktor tersebut dibagi menjadi tiga, yaitu: zona lokasi, kompetisi, dan zona demografi.

Berkenaan dengan zona lokasi dan kompetisi toko serba ada, beberapa penelitian menyajikan bukti yang kontradiktif yang menyatakan bahwa kehadiran pesaing utama atau munculnya banyak pesaing mungkin secara visibilitas mempengaruhi kinerja dari toko serba ada [1], [2]. Di sisi lain, karakteristik ini mungkin dapat mengindikasikan sebuah lokasi dengan potensi ekonomi yang kuat dan daya beli yang mempengaruhi kinerja toko secara positif . Faktor lain yang terkait dengan kinerja toko serba ada adalah zona demografi yang meliputi: potensi pasar, karakteristik demografi, pertumbuhan dari suatu daerah, musim, luas toko, jumlah kategori barang dari toko serba ada, dan jumlah penjualan yang terjadi di dalam satu periode.

Dalam penelitian ini faktor-faktor yang mempengaruhi penjualan di cabang-cabang toko serba ada dianalisis. Selain analisis faktor-faktor yang mempengaruhi tersebut, penelitian ini juga membahas model-model *machine learning* yang akan digunakan untuk memprediksi penjualan di toko-toko tersebut.

Hasil lain yang juga diharapkan dari penelitian ini adalah pemodelan hubungan antara faktor-faktor yang mempengaruhi penjualan toko serba ada.

II. KAJIAN TEORI

A. Machine Learning

Metode-metode berbasis statistik yang diimplementasi dalam program komputer untuk menemukan pola dan faktor tertentu di dataset.

B. Regresi

Regresi adalah suatu metode statistik yang digunakan dalam berbagai bidang dan salah satunya terkait penjualan toko serba ada. Metode ini akan digunakan untuk menentukan kekuatan relasi antara prediktor penjualan toko (biasanya dilambangkan dengan *Y*) dan serangkaian prediktor-prediktor lain, seperti prediktor luas area toko, jumlah barang tersedia, dan jumlah pelanggan dalam periode satu bulan.

1) Regresi Linear Berganda

Regresi linier berganda merupakan bentuk umum dari regresi linier sederhana. Model persamaan regresi linier berganda menjelaskan hubungan satu variabel independen atau *response* (*Y*) dengan dua atau lebih variabel prediktor ($X_1, X_2, \ldots, X_n$).

2) Ordinary Least Squares

Linear Regression cocok dengan model linier dengan koefisien $b = (b_1, \ldots, b_n)$ untuk meminimalkan jumlah sisa kuadrat antara target yang diamati dalam himpunan data, dan target yang diprediksi oleh perkiraan linier.

3) Ridge Regression

Ridge Regression mengatasi masalah *overfitting* yang dialami dalam *Ordinary Least Squares* dengan cara menambahkan satu suku yang berfungsi sebagai penalti pada nilai koefisien atau biasa disebut dengan *l2-norm regularization*.

4) Lasso

Lasso adalah model regresi linier dengan koefisien yang sebagian besar bernilai nol. Koefisien-koefisien yang nol berguna dalam konteks untuk mengurangi *overfitting* karena koefisien-koefisien yang tak-nol apalagi bernilai besar akan memiliki kecenderungan *overfitting.*

5) Elastic-Net

Elastic-Net adalah model regresi linier yang dilatih dengan regularisasi *l1* dan *l2 norm regularization*.

6) Bayesian Regression

Teknik *Bayesian Regresision* dapat digunakan untuk memasukkan parameter regularisasi dalam prosedur estimasi, yaitu parameter ini disesuaikan terhadap data.

7) Support Vector Machines (SVM)

Support Vector Machine (SVM) adalah model *Machine Learning* yang kuat dan serbaguna, mampu melakukan klasifikasi linier atau nonlinier, regresi, dan bahkan deteksi outlier.

8) Random Forest

Random Forest adalah ansambel dari *Decision Tree* yang secara umum dilatih melalui metode *bagging*. Biasanya Random Forest memiliki *hyperparameter* max_samples adalah dengan ukuran *train set*.

9) Gradient Boosting

Gradient Boosting bekerja dengan menambahkan prediktor secara berurutan ke dalam ansambel, masing-masing melakukan koreksi pada pendahulunya. Metode ini mencoba melatih prediktor baru dengan menggunakan kesalahan residual yang dibuat oleh prediktor sebelumnya.

C. Dataset

Dataset adalah sekumpulan data yang biasanya disajikan dalam bentuk tabular atau tabel. Setiap kolom menjelaskan variabel tertentu yang mungkin menjadi prediktor dan setiap baris di *dataset* merupakan *instance* dengan variabel-variabel yang sudah bernilai.

1) Data Preprocessing

Preprocessing mengacu pada transformasi yang diterapkan pada data sebelum data dimasukkan ke dalam algoritma *machine learning*.

III. ANALISIS DAN PERANCANGAN SISTEM

Tahapan dalam membangun model prediksi mencakup: *data cleaning*, *feature scaling*, memilih dan melatih model, melakukan *fine-tuning* pada model, dan mengevaluasi model pada *test set*.

A. Preprocessing Dataset

Pada tahap *preprocessing dataset*, terdapat beberapa hal yang perlu diperiksa, di antaranya adalah memeriksa *missing value*, menyeragamkan semua data ke format ‘int64’, memeriksa penamaan kolom di *dataset*, dan memeriksa satuan data di *dataset* untuk memastikan konsistensi.

1) Pemilihan Prediktor dan Response

Pemilihan prediktor dan *response* adalah proses memilih *subset* dari prediktor yang relevan dengan melakukan evaluasi berdasarkan kriteria tertentu.

2) Feature Scaling

Feature scaling adalah teknik untuk melakukan standarisasi prediktor independen yang ada di data dalam rentang tetap. Ini dilakukan selama pra-pemrosesan data.

B. Penentuan metrik evaluasinya

Dalam penentuan metrik evaluasi untuk mengukur kinerja model prediksi, terdapat beberapa metode yang umum digunakan. Salah satu metrik evaluasi yang sering digunakan adalah *Mean Square Error (MSE) dan Root Mean Squared Error (RMSE).*

1) Mean Square Error (MSE)

$$\text{MSE} = \frac{1}{m}\sum_{i=0}^{m}(X_i - Y_i)^2 \quad (10)$$

Mean Square Error (MSE) digunakan sebagai salah satu metrik evaluasi kinerja yang digunakan dalam statistik dan *machine learning* untuk mengukur seberapa dekat model prediksi dengan nilai aktual.

2) Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=0}^{m}(X_i - Y_i)^2} \quad (11)$$

Root Mean Squared Error (RMSE) adalah akar kuadrat dari *Mean Square Error* (MSE) dan menghasilkan hasil yang lebih mudah dipahami karena berada dalam satuan yang sama dengan data aktual.

C. Pembuatan Model Prediksi

Dalam pembuatan model prediksi, diperlukan beberapa tahap sebelum melakukan pemodelan.

1) Penentuan Hyperparameter

Mengevaluasi sebuah model dapat dilakukan dengan melatih setiap tipe dari model dan melakukan perbandingan atas seberapa baik model-model tersebut dalam penggunaan *test set*.

2) Randomized Search CV (5 fold)

Randomized Search adalah proses mengutak-atik *hyperparameter* sampai ditemukan kombinasi terbaik dari nilai *hyperparameter*.

3) Pelatihan Model

Setelah nilai *hyperparameter* terbaik ditemukan, model dilatih pada seluruh *training set* dengan nilai hyperparameter tersebut.

4) Fine-Tuning pada Model

Fine-tuning dilakukan dengan memeriksa performa model dan melakukan penyesuaian pada *hyperparameter* jika diperlukan. Proses ini dilakukan sampai ditemukan model dengan performa terbaik.

D. Evaluasi pada Test set

Evaluasi pada *test set* adalah tahap akhir dari pembuatan model prediksi, yang bertujuan untuk mengukur seberapa baik performa model pada data yang belum pernah dilihat sebelumnya.

IV. IMPLEMENTASI

A. Preprocessing Dataset

Tahap awal *preprocessing dataset* adalah melakukan impor *library* yang digunakan untuk analisis data.

```
import pandas as pd
import numpy as np
import statistics
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
```

Kode 4.1 Algoritma untuk melakukan impor *library*

1) Pandas

Pandas (pd) digunakan untuk memanipulasi dan menganalisis data terutama data tabular seperti tabel *spreadsheet*. *Library* ini menyediakan struktur data yang mudah digunakan dan memiliki banyak fungsi untuk memproses data.

2) NumPy

NumPy (np) digunakan untuk melakukan operasi matematika pada *array* atau matriks numerik.

3) *Statistics*

Statistics berfungsi untuk melakukan operasi statistik pada data numerik seperti menghitung rata-rata (mean) , akar kuadrat (sqrt), variansi (var), standar deviasi (std), dan korelasi (corr).

4) *Matplotlib*

Matplotlib digunakan untuk melakukan visualisasi data.

5) *Load Dataset*

Isi dari *dataset* dimuat dengan menggunakan fungsi "pd.read_csv()" dengan *file* "Stores.csv" sebagai argumen. Kemudian, *dataset* ditampilkan dengan menggunakan "data.head()", yang akan menampilkan 5 baris pertama dari *dataset*. Store_Area memiliki satuan Yard Square, Items_Available dalam satuan unit (Pcs), dan Store_Sales dalam mata uang dollar (Usd).

```
data = pd.read_csv('Stores.csv')
data.head()
```

	Store ID	Store_Area	Items_Available	Daily_Customer_Count	Store_Sales
0	1	1659	1961	530	66490
1	2	1461	1752	210	39820
2	3	1340	1609	720	54010
3	4	1451	1748	620	53730
4	5	1770	2111	450	46620

Kode 4.2 Menampilkan 5 baris pertama dalam Stores.csv

6) Memeriksa *Missing Value*

"Print(data.isnull().sum())" digunakan untuk menampilkan jumlah nilai null pada setiap kolom *dataset*. Tujuannya adalah untuk mengetahui apakah ada data yang hilang atau tidak.

```
print("Jumlah missing value pada setiap kolom:")
print(data.isnull().sum())
```

```
Jumlah missing value pada setiap kolom:
Store ID                0
Store_Area              0
Items_Available         0
Daily_Customer_Count    0
Store_Sales             0
dtype: int64
```

Kode 4.3 Tampilan pemeriksaan *missing value* pada kolom *dataset*

7) Memeriksa Tipe Kolom Data

```
data.describe()
data.dtypes
```

Kode 4.4 Tampilan pemeriksaan data

Pada Kode 4.4, Kolom data diperiksa menggunakan "data.describe()" untuk menampilkan daftar nama kolom dalam *dataset* beserta tipe datanya dan "data.dtypes" untuk menampilkan tipe data. Ini membantu memastikan bahwa semua kolom memiliki tipe data yang tepat dan sesuai dengan jenis datanya.

B. Penerapan Pemilihan Prediktor

```
X = data[['Store_Area', 'Items_Available', 'Daily_Customer_Count']]
Y = data['Store_Sales']
```

Kode 4.5 Algoritma untuk mengelompokkan variabel Prediktor (X) dan *Response* (Y)

Pada Kode 4.5, dilakukan pemilihan prediktor yang akan digunakan dalam pemodelan. Untuk proyek ini, prediktor yang digunakan adalah Store_Area, Items_Available, dan Daily_Customer_Count.

C. Implementasi Train-Test Split pada Machine Learning

Pada Kode 4.6, data dibagi menjadi tiga buah set yaitu *training set*, *validation set*, dan *testing set* dengan menggunakan fungsi "train_test_split" dari *library* Scikit-learn. Fungsi ini digunakan untuk membagi dan menentukan ukuran data yang akan digunakan sebagai *testing set*.

```
From sklearn.model_selection import train_test_split
X_train_val, X_test, Y_train_val, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
X_train, X_val, Y_train, Y_val = train_test_split(X_train_val,
Y_train_val, test_size=0.2, random_state=42)
```

Kode 4.6 Implementasi *Train-Test Split*

Dalam implementasi Kode 4.6, variabel X_train_val dan Y_train_val merupakan gabungan dari X dan Y. Fungsi train_test_split digunakan dua kali untuk membagi data menjadi tiga set: *training set* (X_train dan Y_train), *validation set* (X_val dan Y_val), dan *testing set* (X_test dan Y_test). *Random state* digunakan untuk memastikan bahwa hasil eksperimen dapat direproduksi setiap kali kode program dijalankan.

D. Standarisasi Data

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
scale_pipeline = Pipeline([
 ('scale', StandardScaler())])
X_train_scale = scale_pipeline.fit_transform(X_train)
X_val_scale = scale_pipeline.transform(X_val)
X_test_scale = scale_pipeline.transform(X_test)
```

Kode 4.7 Algoritma untuk standarisasi data

Pada Kode 4.7 digunakan "StandardScaler" dari *library* Scikit-learn untuk melakukan *feature scaling* pada *dataset*. "StandardScaler" melakukan standarisasi dengan mengurangi rata-rata dan membagi standar deviasi dari setiap prediktor.

E. Pengujian Model Prediksi

Setelah melakukan *preprocessing*, pemisahan, dan standarisasi data, selanjutnya adalah melakukan pengujian model prediksi untuk mengukur seberapa baik performa dari suatu model.

1) Linear Regression

```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
```

Kode 4.8Pembuatan objek *Linear Regression*

Pada Kode 4.8, dilakukan pembuatan objek model *linear regression* yang akan digunakan untuk melakukan proses *training* pada data yang telah di-*preprocess* sebelumnya.

```
Linear_model.fit(X_train_scale, Y_train)
```

Kode 4.9 Pelatihan linear_model menggunakan X_train_scale

Fungsi fit() pada Kode 4.9 digunakan untuk melatih model dengan menggunakan *training set* yang telah diubah skala dan target variabel.

```
From sklearn.metrics import mean_squared_error
Y_pred_train = linear_model.predict(X_train_scale)
mse_linear_train = mean_squared_error(Y_train, Y_pred_train)
print("RMSE TRAIN:", np.sqrt(mse_linear_train))
RMSE TRAIN: 17589.442677811374
```

Kode 4.10 Prediksi linear_model pada *training set*

Pada Kode 4.10, prediksi nilai Y_train dilakukan dengan menggunakan model linear_model yang telah dilatih sebelumnya dengan menggunakan data X_train_scale. Setelah itu, dilakukan perhitungan nilai *mean squared error* (MSE) antara Y_train dan Y_pred_train. Setelah mendapatkan nilai *mean squared error* (MSE), nilai tersebut diakarkan untuk mendapatkan nilai *root mean squared error* (RMSE).

```
Y_pred_val = linear_model.predict(X_val_scale)
mse_linear_val = mean_squared_error(Y_val, Y_pred_val)
print("RMSE VAL:", np.sqrt(mse_linear_val))
```

```
RMSE VAL: 15894.535635479742
```

Kode 4.11 Prediksi linear_model pada *validation set*

Pada Kode 4.11, hasil *root mean squared error* (RMSE) untuk *validation set* adalah sebesar 15894.535635479742.

```
Y_pred_test = linear_model.predict(X_test_scale)
mse_linear_test = mean_squared_error(Y_test, Y_pred_test)
rmse_linear_test = np.sqrt(mse_linear_test)
print("RMSE TEST:", rmse_linear_test)
RMSE TEST: 16434.041128953024
```

Kode 4.12 Prediksi linear_model pada *test set*

Kode pada Kode 4.12, Hasil RMSE TEST yang dihasilkan adalah 16434.041128953024.Ridge Regression

```
from sklearn.linear_model import Ridge
ridge_model = Ridge()
```

Kode 4.13 Pembuatan objek *Ridge Regression*

Kode 4.13 melakukan impor Ridge dari pustaka Scikit-learn dan kemudian membuat objek model *Ridge Regression*. Objek ini akan digunakan dalam Kode 4.14 untuk melatih dan mengevaluasi model Regresi Ridge.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

hyperparameters_ridge = {
 'alpha': uniform(0, 100)}
random_search = RandomizedSearchCV(
 ridge_model, hyperparameters_ridge, n_iter=10,n_jobs=-1,
 cv=5, random_state=42)
random_search.fit(X_train, Y_train)
print("Best hyperparameter: ", random_search.best_params_)
Best hyperparameter: {'alpha': 95.07143064099162}
```

Kode 4.14 Hyperparameter tuning Ridge Regression

Kode pada Kode 4.14 digunakan untuk melakukan *hyperparameter tuning* pada model Ridge Regression. *Hyperparameter tuning* dilakukan dengan menggunakan algoritma "RandomizedSearchCV" dari *library* Scikit-learn. Pada kasus ini, *hyperparameter* yang akan dilakukan *tuning* adalah alpha. *Hyperparameter* alpha ditetapkan dengan menggunakan distribusi uniform dengan rentang nilai antara 0 dan 100.

```
ridge_model = Ridge(95.07143064099162)
ridge_model.fit(X_train_scale, Y_train)
```

Kode 4.15 Pelatihan ridge_model dengan best hyperparameter

Pada Kode 4.15, dilakukan inisialisasi objek ridge_model dengan mengatur nilai *hyperparameter* alpha menggunakan hasil terbaik dari proses *hyperparameter tuning* yang telah dilakukan sebelumnya. Selanjutnya, model dilatih menggunakan metode fit() pada objek ridge_model.

```
Y_pred_train = ridge_model.predict(X_train_scale)
mse_ridge_train = mean_squared_error(Y_train, Y_pred_train)
rmse_ridge_train = np.sqrt(mse_ridge_train)
print("RMSE TRAIN:", rmse_ridge_train )
RMSE TRAIN: 17592.059283344533
```

Kode 4.16 Prediksi ridge_model pada *training set*

Hasil RMSE TRAIN untuk model *Ridge* adalah 17592.059283344533.

```
Y_pred_val = ridge_model.predict(X_val_scale)
mse_ridge_val = mean_squared_error(Y_val, Y_pred_val)
rmse_ridge_val = np.sqrt(mse_ridge_val)
print("RMSE VAL:", rmse_ridge_val )
RMSE VAL: 15914.014924444582
```

Kode 4.17 Prediksi ridge_model pada *validation set*

Hasil RMSE VALIDATION untuk model Ridge adalah 15914.014924444582.

```
Y_pred_test = ridge_model.predict(X_test_scale)
mse_ridge_test = mean_squared_error(Y_test, Y_pred_test)
print("RMSE TEST:", np.sqrt(mse_ridge_test))
RMSE TEST: 16456.462555199978
```

Kode 4.18 Prediksi ridge_model pada *test set*

Pada kasus ini, didapatkan hasil RMSE TEST adalah sebesar 16456.462555199978.

2) Lasso Regression

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(max_iter=5000)
```

Kode 4.19 Pembuatan objek Lasso Regression

Parameter max_iter pada lasso_model merupakan jumlah iterasi maksimum yang akan dilakukan oleh algoritma optimasi saat melatih model.

```
hyperparameters_lasso = {
 'alpha': uniform(0,100)}

random_search = RandomizedSearchCV(
 lasso_model, hyperparameters_lasso,n_iter=10, n_jobs=-1,
 cv=5, random_state=42)
random_search.fit(X_train, Y_train)
print("Best hyperparameter: ", random_search.best_params_)
Best hyperparameter: {'alpha': 95.07143064099162}
```

Kode 4.20 *Hyperparameter tuning Lasso Regression*

Hyperparameter yang diuji adalah alpha dengan range nilai antara 1 dan 100 menggunakan fungsi uniform dari modul scipy.stats. Kemudian, objek RandomizedSearchCV dibuat dengan parameter model yang akan diuji (lasso_model), *hyperparameter* yang akan diuji (hyperparameters_lasso), jumlah iterasi (n_iter), jumlah pekerja (n_jobs), dan jumlah lipatan pada cross validation (cv).

```
lasso_model = Lasso( alpha = 95.07143064099162)
lasso_model.fit(X_train_scale, Y_train)
```

Kode 4.21 Pelatihan lasso_model dengan best *hyperparameter*

Selanjutnya membuat model *Lasso Regression* dengan nilai alpha yang sudah ditentukan menggunakan *hyperparameter tuning*.

```
Y_pred_train = lasso_model.predict(X_train_scale)
mse_lasso_train = mean_squared_error(Y_train, Y_pred_train)
print("RMSE TRAIN:", np.sqrt(mse_lasso_train))
RMSE TRAIN: 17591.851142719206
```

Kode 4.22 Prediksi lasso_model pada *training set*

Hasil RMSE TRAIN untuk model Lasso adalah 17591.851142719206.

```
Y_pred_val = lasso_model.predict(X_val_scale)
mse_lasso_val = mean_squared_error(Y_val, Y_pred_val)
print("RMSE VAL:", np.sqrt(mse_lasso_val))
RMSE VAL: 15913.704679182225
```

Kode 4.23 Prediksi lasso_model pada *validation set*

Dalam kasus ini, RMSE VALIDATION model *Lasso Regression* adalah 15913.704679182225.

```
Y_pred_test = lasso_model.predict(X_test_scale)
mse_lasso_test = mean_squared_error(Y_test, Y_pred_test)
print("RMSE TEST:", np.sqrt(mse_lasso_test))
RMSE TEST: 16445.31132357087
```

Kode 4.24 Prediksi lasso_model pada *test set*

Hasil evaluasi model *Lasso* pada *test set* menunjukkan RMSE sebesar 16,445.31132357087.

3) Elastic-Net Regression

```
from sklearn.linear_model import ElasticNet
elastic_model = ElasticNet(max_iter=5000)
```

Kode 4.25 Pembuatan objek *Elastic-Net Regression*

Kode 4.25 melakukan impor ElasticNet dari pustaka Scikit-learn dan kemudian membuat objek ElasticNet. Parameter max_iter=5000 mengatur jumlah iterasi maksimum yang dilakukan oleh algoritma optimasi dalam menemukan koefisien model yang optimal.

```
hyperparameters_elastic = {
 'alpha': uniform(0, 50),
 'l1_ratio' :[0.5]}
random_search = RandomizedSearchCV(
 elastic_model, hyperparameters_elastic, n_iter=10,n_jobs=-1,
 cv=5, random_state=42)
random_search.fit(X_train, Y_train)
print("Best hyperparameters: ", random_search.best_params_)
Best hyperparameters: {'alpha': 47.53571532049581, 'l1_ratio': 0.5}
```

Kode 4.26 *Hyperparameter tuning Elastic-Net Regression*

Kode 4.26 melakukan *hyperparameter tuning* menggunakan metode RandomizedSearchCV pada model *elastic*. *Hyperparameter* yang dilakukan *tuning* adalah alpha dan l1_ratio. Hyperparameter alpha ditentukan menggunakan distribusi uniform dengan rentang 0 hingga 50. Sedangkan *hyperparameter* l1_ratio memiliki nilai konstan 0.5.

```
elastic_model = ElasticNet(alpha = 47.53571532049581,
 l1_ratio = 0.5)
elastic_model.fit(X_train_scale, Y_train)
```

Kode 4.27 Pelatihan elastic_model dengan *best hyperparameter*

Selanjutnya yaitu membuat model ElasticNet dengan menggunakan nilai alpha dan l1_ratio yang telah ditentukan. Nilai alpha yang digunakan adalah 47.53571532049581, sedangkan nilai l1_ratio yang digunakan adalah 0.5.

```
Y_pred_train = elastic_model.predict(X_train_scale)
mse_elastic_train = mean_squared_error(Y_train, Y_pred_train)
print("RMSE TRAIN:", np.sqrt(mse_elastic_train))
RMSE TRAIN: 17628.742486432973
```

Kode 4.28 Prediksi elastic_model pada *training set*

Hasil RMSE TRAIN untuk model *Elastic* adalah 17628.742486432973.

```
Y_pred_val = elastic_model.predict(X_val_scale)
mse_elastic_val = mean_squared_error(Y_val, Y_pred_val)
print("RMSE VAL:", np.sqrt(mse_elastic_val))
RMSE VAL: 16122.698912915495
```

Kode 4.29 Prediksi elastic_model pada *validation set*

Dalam kasus ini, RMSE VALIDATION model *elastic-net regression* adalah 16122.698912915495.

```
Y_pred_test =elastic_model.predict(X_test_scale)
mse_elastic_test = mean_squared_error(Y_test, Y_pred_test)
print("RMSE TEST:", np.sqrt(mse_elastic_test))
RMSE TEST: 16502.529319601672
```

Kode 4.30 Prediksi elastic_model pada *test set*

4) Bayesian Ridge Regression

```
from sklearn.linear_model import BayesianRidge
bayesian_model = BayesianRidge()
```

Kode 4.31 Pembuatan objek Bayesian Ridge Regression

Kode 4.31 melakukan impor BayesianRidge dari *library* sklearn.linear_model dan membuat objek BayesianRidge dengan menggunakan parameter *default*.

```
hyperparameters_bayesian = {
 'alpha_1' : uniform(0, 100000),
 'alpha_2' : uniform(0, 100000),
 'lambda_1': [0.000001],
 'lambda_2': [0.000001]}
random_search = RandomizedSearchCV(
 bayesian_model, hyperparameters_bayesian, n_iter=10,n_jobs=-1,
 cv=5, random_state=42)

random_search.fit(X_train, Y_train)
print("Best hyperparameters: ", random_search.best_params_)
Best hyperparameters: {'alpha_1': 2058.4494295802447, 'alpha_2':
96990.98521619943, 'lambda_1': 1e-06, 'lambda_2': 1e-06}
```

Kode 4.32 *Hyperparameter tuning Bayesian Ridge Regression*

Hyperparameter yang dilakukan *tuning* adalah alpha_1, alpha_2, lambda_1, dan lambda_2. *Hyperparameter* alpha_1 dan alpha_2 ditentukan menggunakan distribusi uniform dengan rentang 0 hingga 100000. Sedangkan *hyperparameter* lambda_1 dan lambda_2 memiliki nilai konstan 0.000001.

```
bayesian_model = BayesianRidge(alpha_1 = 2058.4494295802447,
 alpha_2 = 96990.98521619943,
 lambda_1= 1e-06,
 lambda_2= 1e-06)
bayesian_model.fit(X_train_scale, Y_train)
```

Kode 4.33 Pelatihan bayesian_model dengan *best hyperparameter*

Pada Kode 4.44 ini dilakukan pembuatan model BayesianRidge baru dengan menggunakan nilai alpha_1, alpha_2, lambda_1, dan lambda_2 yang telah ditentukan pada proses *tuning hyperparameter* sebelumnya.

```
Y_pred_train = bayesian_model.predict(X_train_scale)
mse_bayesian_train = mean_squared_error(Y_train, Y_pred_train)
print("RMSE TRAIN:", np.sqrt(mse_bayesian_train))
RMSE TRAIN: 17592.10951707304
```

Kode 4.34 Prediksi bayesian_model pada *training set*

Hasil RMSE TRAIN untuk model *Bayesian Ridge Regression* adalah 17592.10951707304.

```
Y_pred_val = bayesian_model.predict(X_val_scale)
mse_bayesian_val = mean_squared_error(Y_val, Y_pred_val)
print("RMSE VAL:", np.sqrt(mse_bayesian_val))
RMSE VAL: 15915.376812269973
```

Kode 4.35 Prediksi bayesian_model pada *validation set*

Hasil RMSE VALIDATION untuk model *Bayesian Ridge Regression* adalah 15915.376812269973.

```
Y_pred_test = bayesian_model.predict(X_test_scale)
mse_bayesian_test = mean_squared_error(Y_test, Y_pred_test)
print("RMSE TEST:", np.sqrt(mse_bayesian_test))
RMSE TEST: 16456.590119056767
```

Kode 4.36 Prediksi bayesian_model pada *test set*

Pada kasus ini, didapatkan hasil RMSE model *Bayesian Ridge Regression* pada *test set* adalah sebesar 16456.590119056767.

*5) Support Vector Machine*s (SVM)

```
from sklearn.svm import SVR
svr_model = SVR()
```

Kode 4.37 Pembuatan objek Support Vector Machines

Pada Kode 4.37, dilakukan impor SVR dari *library* sklearn.svm dan membuat objek SVR baru dengan menggunakan parameter *default*.

```
from sklearn.metrics.pairwise import rbf_kernel
X_kernel = rbf_kernel(X_train_scale)
```

Kode 4.38 Pembuatan Matriks Kernel RBF untuk objek *Support Vector Machines*

Selanjutnya melakukan *impor* fungsi rbf_kernel dari *library* sklearn.metrics.pairwise. Fungsi rbf_kernel digunakan untuk menghitung matriks kernel RBF dari data X_train_scale. Matriks kernel ini dapat digunakan sebagai input untuk model SVR dengan kernel RBF untuk melakukan regresi pada data yang diberikan.

```
hyperparameters_svr = {
 'kernel': ['linear', 'poly', 'rbf', 'precomputed'],
 'gamma': ['scale', 'auto'] + list(np.arange(0, 10, 1)),
 'C': list(np.arange(0.1, 10, 1)),
 'epsilon': list(np.arange(0, 10, 1)),
 'shrinking': [True, False]}

random_search = RandomizedSearchCV(
 svr_model, hyperparameters_svr, n_iter=10,n_jobs=-1,
 cv=5, random_state=42)
random_search.fit(X_kernel, Y_train)
print("Best hyperparameters: ", random_search.best_params_)
Best hyperparameters: {'shrinking': False, 'kernel': 'linear', 'ga
mma': 6, 'epsilon': 5, 'C': 7.1
```

Kode 4.39 *Hyperparameter tuning Support Vector Machines*

Pada Kode 4.39, dilakukan *hyperparameter tuning* untuk model SVR dengan menggunakan metode RandomizedSearchCV.

Dalam hyperparameters_svr, dilakukan penentuan beberapa *hyperparameters* yang akan diuji, yaitu kernel, gamma, C, epsilon, dan shrinking.

```
svr_model = SVR(
 kernel='linear',
 gamma=6,
 C=7.1,
 epsilon=5,
 shrinking=False)
svr_model.fit(X_train_scale, Y_train)
```

Kode 4.40 Pelatihan svr_model dengan *best hyperparameter*

Dilakukan inisialisasi objek svr_model dari class SVR dengan *hyperparameter* yang sudah ditentukan pada Kode 4.39

Parameter kernel pada model SVR adalah salah satu *hyperparameter* yang mempengaruhi performa dari model. Dalam hal ini, digunakan kernel 'linear', yang artinya bahwa model akan memetakan data input ke dalam ruang prediktor yang sama dengan dimensi yang sama.

```
Y_pred_train = svr_model.predict(X_train_scale)
mse_svr_train = mean_squared_error(Y_train, Y_pred_train)
print("RMSE TRAIN:", np.sqrt(mse_svr_train))
RMSE TRAIN: 17636.09250559727
```

Kode 4.41 Prediksi svr_model pada *training set*

Hasil RMSE TRAIN untuk model SVR adalah 17636.09250559727.

```
Y_pred_val = svr_model.predict(X_val_scale)
mse_svr_val = mean_squared_error(Y_val, Y_pred_val)
print("RMSE VAL:", np.sqrt(mse_svr_val))
RMSE VAL: 16006.268573987105
```

Kode 4.42 Prediksi svr_model pada *validation set*

Hasil RMSE VALIDATION untuk model SVR adalah 16006.268573987105.

```
Y_pred_test = svr_model.predict(X_test_scale)
mse_svr_test = mean_squared_error(Y_test, Y_pred_test)
print("RMSE TEST:", np.sqrt(mse_svr_test))
RMSE TEST: 16519.108645397395
```

Kode 4.43 Prediksi svr_model pada *test set*

Pada kasus ini, didapatkan hasil RMSE model SVR pada *test set* adalah sebesar 16519.108645397395.4.6.7 *Random Forest Regressor*

6) Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor()
```

Kode 4.44 Pembuatan objek *Random Forest Regressor*

Melakukan pemanggilan kelas RandomForestRegressor dari *library* sklearn.ensemble. Kemudian, membuat objek rf_model dengan parameter *default*.

```
hyperparameters_rf = {
 'n_estimators': [100],
 'criterion': ['squared_error', 'absolute_error', 'friedman_mse',
'poisson'],
 'max_depth': [2,5],
 'min_samples_split': range(1,5),
 'min_samples_leaf': range(1,5),
 'max_features':['sqrt'] ,
 'max_leaf_nodes': [None],
 'bootstrap': [True],
 'oob_score': [True] }


random_search = RandomizedSearchCV(
 rf_model, hyperparameters_rf,n_iter=10, n_jobs=-1,
 cv=5, random_state=42)
random_search.fit(X_train, Y_train)
print("Best hyperparameters: ", random_search.best_params_)
Best hyperparameters: {'oob_score': True, 'n_estimators': 100, 'm
in_samples_split': 1, 'min_samples_leaf': 3, 'max_leaf_nodes': No
ne, 'max_features': 'sqrt', 'max_depth': 2, 'criterion': 'poisson
', 'bootstrap': True}
```

Kode 4.45 *Hyperparameter tuning Random Forest*

Dalam hyperparameters_rf, ditentukan beberapa *hyperparameters* yang akan diuji, yaitu n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, max_features, max_leaf_nodes, bootstrap, dan oob_score.

```
rf_model = RandomForestRegressor(n_estimators=100,
 criterion='poisson',
 max_depth=2,
 min_samples_split=1,
 min_samples_leaf=3,
 max_features='sqrt',
 max_leaf_nodes=None,
 bootstrap=True,
 oob_score=True,
 )
rf_model.fit(X_train_scale, Y_train)
RandomForestRegressor(criterion='poisson', max_depth=2, max_fea
tures='sqrt',
 min_samples_leaf=3, min_samples_split=1, oob_score=True)
```

Kode 4.46 Pelatihan rf_model dengan *best hyperparameter*

Selanjutnya membuat objek rf_model dengan menggunakan *hyperparameters* terbaik yang telah ditemukan sebelumnya.

```
Y_pred_train = rf_model.predict(X_train_scale)
mse_rf_train = mean_squared_error(Y_train, Y_pred_train)
```

```
print("RMSE TRAIN:", np.sqrt(mse_rf_train))
RMSE TRAIN: 17300.470517639256
```

Kode 4.47 Prediksi rf_model pada *training set*

Hasil RMSE TRAIN untuk model *Random Forest* adalah 17300.470517639256.

```
Y_pred_val = rf_model.predict(X_val_scale)
mse_rf_val = mean_squared_error(Y_val, Y_pred_val)
print("RMSE VAL:", np.sqrt(mse_rf_val))
RMSE VAL: 16036.303752435615
```

Kode 4.48 Prediksi rf_model pada *validation set*

Hasil RMSE VALIDATION untuk model *Random Forest* adalah 16036.303752435615.

```
Y_pred_test = rf_model.predict(X_test_scale)
mse_rf_test = mean_squared_error(Y_test, Y_pred_test)
print("RMSE TEST:", np.sqrt(mse_rf_test))
RMSE TEST: 16425.377052182943
```

Kode 4.49 Prediksi rf_model pada *test set*

Pada kasus ini, didapatkan hasil RMSE model *Random Forest* pada *test set* adalah sebesar 16425.377052182943.

7) Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
gb_model = GradientBoostingRegressor()
```

Kode 4.50 Pembuatan objek *Gradient Boosting Regressor*

Pada Kode 4.50, dilakukan pembuatan kelas GradientBoostingRegressor dari modul sklearn.ensemble. Kemudian, dilakukan pembuatan sebuah objek gb_model yang akan digunakan untuk membangun model gradient boosting r*egressor*.

```
hyperparameters_gb = {
 'learning_rate': [0.005, 0.01, 0.05],
 'n_estimators': [100, 150, 200],
 'subsample': [0.1],
 'min_samples_split': [2, 5, 10],
 'min_samples_leaf': [1, 2, 4],
 'max_depth': [2, 3, 4],
 'max_features': ['sqrt'],
 'alpha': [0.9],
 'max_leaf_nodes': [None],
 'criterion': ['squared_error']}
random_search = RandomizedSearchCV(
 gb_model, hyperparameters_gb, n_iter=10,n_jobs=-1,
 cv=5, random_state=42 )

random_search.fit(X_train, Y_train)
print("Best hyperparameters: ", random_search.best_params_)
Best hyperparameters: {'subsample': 0.1, 'n_estimators': 100, '
min_samples_split': 2, 'min_samples_leaf': 2, 'max_leaf_nodes':
 None, 'max_features': 'sqrt', 'max_depth': 2, 'learning_rate':
 0.005, 'criterion': 'squared_error', 'alpha': 0.9}
```

Kode 4.51 *Hyperparameter tuning Gradient Boosting Regressor*

Dalam hyperparameters_gb, beberapa *hyperparameters* yang ditentukan, yaitu learning_rate, n_estimators, subsample, min_samples_split, min_samples_leaf, max_depth, max_features, alpha, max_leaf_nodes, dan criterion.

```
from sklearn.metrics import mean_squared_error
gb_model = GradientBoostingRegressor(
 learning_rate=0.005,
 n_estimators=100,
 subsample=0.5,
 min_samples_split=2,
 min_samples_leaf=2,
 max_depth=3,
 max_features='sqrt',
 alpha=0.9,
 max_leaf_nodes=None,
 criterion='squared_error')
gb_model.fit(X_train_scale,Y_train)
GradientBoostingRegressor(criterion='squared_error', learning_rate
=0.005,
 max_features='sqrt', min_samples_leaf=2,
 subsample=0.5)
```

Kode 4.52 Pelatihan gb_model dengan *best hyperparameter*

Pada Kode 4.52, dilakukan pembuatan objek gb_model dengan *hyperparameters* terbaik yang telah ditemukan melalui proses *hyperparameter tuning*. Setelah objek gb_model dibuat, dilakukan pelatihann dengan memanggil metode fit() pada objek gb_model dengan memasukkan *training set* yang telah dilakukan *scaling* X_train_scale dan Y_train.

```
Y_pred_train = gb_model.predict(X_train_scale)
mse_gb_train = mean_squared_error(Y_train, Y_pred_train)
print("RMSE TRAIN:", np.sqrt(mse_gb_train))
RMSE TRAIN: 17326.40162777696
```

Kode 4.53 Prediksi gb_model pada *training set*

Hasil RMSE TRAIN untuk model *Gradient Boosting* adalah 17326.40162777696.

```
Y_pred_val = gb_model.predict(X_val_scale)
mse_gb_val = mean_squared_error(Y_val, Y_pred_val)
print("RMSE VAL:", np.sqrt(mse_gb_val))
RMSE VAL: 16063.746504475981
```

Kode 4.54 Prediksi gb_model pada *validation set*

Hasil RMSE VALIDATION untuk model *Gradient Boosting* adalah 16063.746504475981.

```
Y_pred_test = gb_model.predict(X_test_scale)
mse_gb_test = mean_squared_error(Y_test, Y_pred_test)
print("RMSE TEST:", np.sqrt(mse_gb_test))
RMSE TEST: 16487.569983449084
```

Kode 4.55 Prediksi gb_model pada *test set*

Hasil RMSE model *Gradient Boosting* pada *test set* adalah sebesar 16487.569983449084.

F. Evaluasi Hasil Prediksi

TABEL I
PARAMETER YANG DIGUNAKAN PADA SETIAP MODEL

MODEL	SETTING
Ridge Regression (RR)	alpha = 95.07143064099162
Lasso (L)	alpha = 95.07143064099162
Elastic-Net (EN)	alpha = 47.53571532049581, l1_ratio = 0.5

Bayesian Ridge Regression (BRR)	alpha_1 = 2058.4494295802447, alpha_2 = 96990.98521619943, lambda_1 = 0.000001, lambda_2 = 0.000001
Support Vector Machines (SVM)	shrinking = False, kernel = linear, gamma = 6, epsilon = 5, C = 7.1
Random Forest (RF)	oob_score = True, n_estimators = 100, min_samples_split = 1, min_samples_leaf = 3, max_leaf_nodes = None, max_features = sqrt, max_depth = 2, criterion = poisson, bootstrap = True
Gradient Boosting (GB)	subsample = 0.1, n_estimators = 100, min_samples_split'= 2, min_samples_leaf = 2, max_leaf_nodes = None, max_features = sqrt, max_depth = 2, learning_rate = 0.005, criterion = squared_error, alpha: 0.9

TABEL II
HASIL RMSE PADA SETIAP MODEL

MODEL			
	TRAIN	VALIDATION	TEST
LR	17589.442677811374	15894.535635479742	16434.041128953024
RR	17592.059283344533	15914.014924444582	16456.462555199978
L	17591.851142719206	15913.704679182225	16445.31132357087
EN	17628.742486432973	16122.698912915495	16502.529319601672
BRR	17592.10951707304	15915.376812269973	16456.590119056767
SVR	17636.09250559727	16006.268573987105	16519.108645397395
RF	17300.470517639256	16036.303752435615	16425.377052182943

GB	17326.40162777696	16063.746504475981	16487.569983449084

Pada sub bab, ini dilakukan evaluasi terhadap performa model terbaik dari masing-masing algoritma yang telah diuji sebelumnya dengan menggunakan *Root Mean Squared Error* (RMSE) pada *test set*. Pada tabel 4.3, model *Random Forest* memiliki RMSE paling rendah pada *test set*, yang berarti model tersebut menunjukkan performa terbaik.

G. Analisis Hasil Prediksi

```
importance_score = rf_model.feature_importances_

sorted_indices = np.argsort(importance_score)[::-1]

for i in sorted_indices:
 print(X_train.columns[i], importance_score[i])
Items_Available 0.3365749160343698
Daily_Customer_Count 0.3340140288286099
Store_Area 0.32941105513702035
```

Kode 4.56 Feature importances

Pada Kode 4.56, dilakukan perhitungan *importance score* untuk model *machine learning* yang terbaik yaitu model *Random Forest*. *Importance score* menghitung pentingnya variabel prediktor dalam mempengaruhi nilai *response* (Y), dan digunakan untuk mengukur signifikansi kontribusi variabel dalam memprediksi nilai target.

TABEL III
SKOR *FEATURE IMPORTANCE* BERDASARKAN *IMPURITY*

Nama Prediktor	Skor Feature Importance berdasarkan Impurity
Items_Available	0.3365749160343698
Daily_Customer_Count	0.3340140288286099
Store_Area	0.32941105513702035

V. KESIMPULAN DAN SARAN

A. Simpulan

Hasil penelitian menunjukkan bahwa model regresi *random forest* memberikan performa yang lebih baik dari model-model lainnya. Meskipun model *random forest* memberikan hasil terbaik, nilai RMSE pada model tersebut masih cukup tinggi, yaitu mencapai sekitar 16.000. Hal ini menunjukkan bahwa ada faktor-faktor lain yang dapat mempengaruhi penjualan dan faktor-faktor tersebut tidak terdapat dalam dataset.

Selanjutnya, prediktor-prediktor model *random forest* dievaluasi berdasarkan *impurity* dan hasil evaluasi mencatat bahwa variabel "Items Available" menempati urutan pertama dan selanjutnya diikuti oleh "Daily Customer Count", dan "Store Area" dalam kategori prediktor yang memiliki pengaruh paling besar terhadap prediksi jumlah penjualan.

Hasil eksperimen yang menguji kombinasi prediktor terhadap model *random forest* menunjukkan bahwa prediktor “Items Available” memiliki performa yang terbaik dibandingkan dengan kombinasi lainnya yang telah diuji coba dalam penelitian ini. Hal ini menunjukkan bahwa semakin banyak variasi item yang tersedia di toko serba ada, maka jumlah penjualan akan meningkat.

B. Saran

Untuk pengembangan lebih lanjut, dapat dilakukan penambahan informasi yang lebih detail seperti informasi tentang waktu penjualan (*dataset time series*). Hal ini dapat memberikan informasi yang lebih lengkap tentang pola penjualan dan dapat memperbaiki kinerja model. Selain itu, dapat pula dilakukan analisis lanjutan seperti analisis faktor untuk mengeksplorasi hubungan antar variabel dan menemukan pola-pola yang lebih kompleks dalam data penjualan.

DAFTAR PUSTAKA

[1] N. B.-P. A. B.-V. J. &. M.-V. F. Roig-Tierno, "The retail site location decision process using GIS and the analytical hierarchy process," 2013.

[2] A. L. &. P. D. Keel, "The effects of adjacent competitors and promotion on brand sales," *Journal of Consumer Marketing*, 2015.

[3] I. &. M. M. J. El Naqa, "What is Machine Learning? Machine Learning in Radiation Oncology," pp. 3-11, 2015.

[4] G. C. R. &. P. S. T, "Prediction of Sales Value in Online shopping using Linear Regression," *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, 2018.

[5] I. M. Yuliara, "Regresi Linier Berganda," 2016.

[6] J. &. M. W. J. Tolles, "Logistic Regression," *JAMA*, vol. 5, p. 316, 2016.

[7] S. M. Hendra Bunyamin, "https://hbunyamin.github.io/machine-learning/Normal_equation/," 10 November 2022. [Online]. [Accessed 10 November 2022].

[8] "Linear Models," 2007, 2007. [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares. [Accessed 13 November 2022].

[9] Qastack.id, "What is exactly meant by a dataset," Qastack.id, [Online]. Available: https://qastack.id/stats/244388/what-is-exactly-meant-by-a-data-set. [Accessed 14 November 2022].

[10] G. e. al., "Big Data Analytics," 2016.

[11] A. Géron, "Hands-on Machine Learning with Scikiit-Learn, Keras, and Tensor Flow," 2019.

[12] C. e. al., "PeerJ Comput," 2021.

[13] Geek, "Feature Scaling," Geeksforgeeks, 25 Agustus 2021. [Online]. Available: https://www.geeksforgeeks.org/ml-feature-scaling-part-1/. [Accessed 18 November 2022].