Zulvanilta Firdaus_4112322022

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Mengimpor Dataset
df = pd.read_csv('loan_approval_dataset.csv')
df.head()
```

	Age	Income	Education_Level	Credit_Score	Loan_Amount	Loan_Purpose	Loan_Approval
0	56	24000	PhD	333	26892	Personal	0
1	46	90588	Master	316	26619	Home	1
2	32	113610	PhD	452	1281	Personal	1
3	60	117856	High School	677	28420	Personal	0
4	25	58304	PhD	641	16360	Car	0

Langkah berikutnya: Buat kode dengan df | Lihat plot yang direkomendasikan | New interactive sheet

2. Ekplorasi Data

```
#Mengecek Missing Value
df.info()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Age              500 non-null    int64
 1   Income           500 non-null    int64
 2   Education_Level  500 non-null    object
 3   Credit_Score     500 non-null    int64
 4   Loan_Amount      500 non-null    int64
 5   Loan_Purpose     500 non-null    object
 6   Loan_Approval    500 non-null    int64
dtypes: int64(5), object(2)
memory usage: 27.5+ KB
```

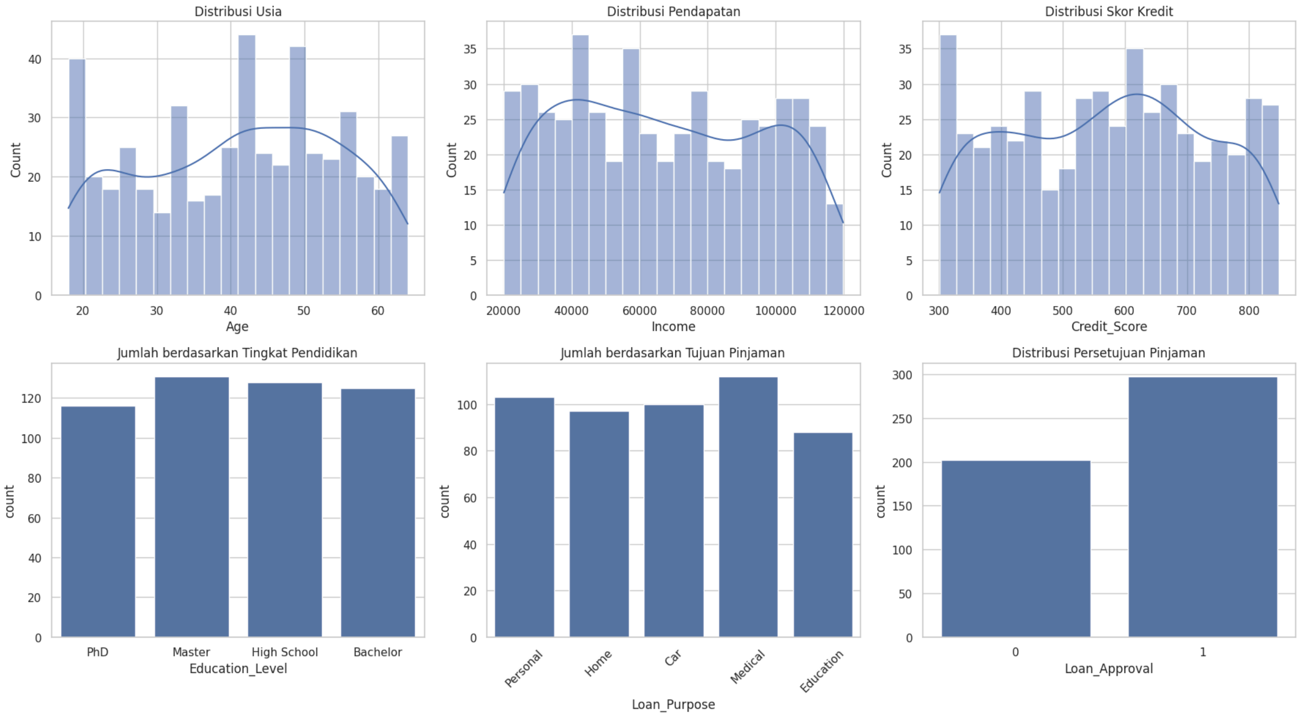
	0
Age	0
Income	0
Education_Level	0
Credit_Score	0
Loan_Amount	0
Loan_Purpose	0
Loan_Approval	0

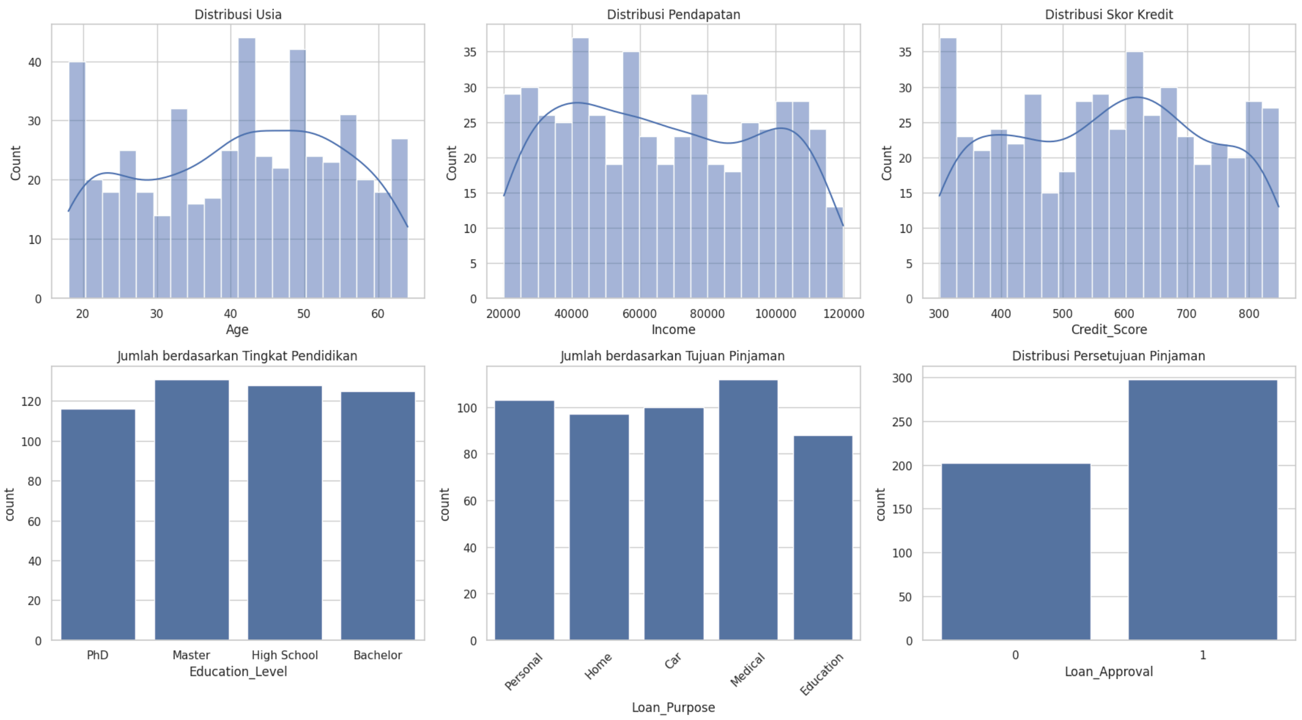
dtype: int64

Pada dataset Loan Approval tidak ditemukan adanya missing value

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set style
sns.set(style="whitegrid")

# Create subplots
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
```

```
# Histogram for numerical columns
sns.histplot(df["Age"], bins=20, kde=True, ax=axes[0, 0])
axes[0, 0].set_title("Distribusi Usia")

sns.histplot(df["Income"], bins=20, kde=True, ax=axes[0, 1])
axes[0, 1].set_title("Distribusi Pendapatan")

sns.histplot(df["Credit_Score"], bins=20, kde=True, ax=axes[0, 2])
axes[0, 2].set_title("Distribusi Skor Kredit")

# Count plot for categorical columns
sns.countplot(x="Education_Level", data=df, ax=axes[1, 0])
axes[1, 0].set_title("Jumlah berdasarkan Tingkat Pendidikan")

sns.countplot(x="Loan_Purpose", data=df, ax=axes[1, 1])
axes[1, 1].set_title("Jumlah berdasarkan Tujuan Pinjaman")
axes[1, 1].tick_params(axis='x', rotation=45)

sns.countplot(x="Loan_Approval", data=df, ax=axes[1, 2])
axes[1, 2].set_title("Distribusi Persetujuan Pinjaman")

# Adjust layout
plt.tight_layout()
plt.show()
# Create subplots
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

# Histogram for numerical columns
sns.histplot(df["Age"], bins=20, kde=True, ax=axes[0, 0])
axes[0, 0].set_title("Distribusi Usia")

sns.histplot(df["Income"], bins=20, kde=True, ax=axes[0, 1])
axes[0, 1].set_title("Distribusi Pendapatan")

sns.histplot(df["Credit_Score"], bins=20, kde=True, ax=axes[0, 2])
axes[0, 2].set_title("Distribusi Skor Kredit")

# Count plot for categorical columns
sns.countplot(x="Education_Level", data=df, ax=axes[1, 0])
axes[1, 0].set_title("Jumlah berdasarkan Tingkat Pendidikan")

sns.countplot(x="Loan_Purpose", data=df, ax=axes[1, 1])
axes[1, 1].set_title("Jumlah berdasarkan Tujuan Pinjaman")
axes[1, 1].tick_params(axis='x', rotation=45)

sns.countplot(x="Loan_Approval", data=df, ax=axes[1, 2])
axes[1, 2].set_title("Distribusi Persetujuan Pinjaman")

# Adjust layout
plt.tight_layout()
plt.show()
```





2. Pemrosesan Data

```
# Melakukan encoding pada fitur kategorikal
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Memisahkan fitur dan target
X = df.drop('Loan_Approval', axis=1)  # Fitur
y = df['Loan_Approval']  # Target

# Melakukan Encoding pada Fitur Kategorikal
label_encoder = LabelEncoder()
categorical_features = ['Education_Level', 'Loan_Purpose']  # Ganti dengan nama kolom kategorikal yang sesuai

for feature in categorical_features:
```

```
    X[feature] = label_encoder.fit_transform(X[feature])

# Melakukan Feature Scaling pada Fitur Numerik
scaler = StandardScaler()
numerical_features = ['Age', 'Income', 'Credit_Score', 'Loan_Amount']  # Ganti dengan nama kolom numerik yang sesuai
X[numerical_features] = scaler.fit_transform(X[numerical_features])

# Menampilkan data setelah Feature Scaling
print("\nData Setelah Feature Scaling:\n", X.head())

# Membagi Dataset menjadi Training Set (80%) dan Testing Set (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=36)

# Menampilkan bentuk dari dataset training dan testing
print("Shape of X_train:", X_train.shape)  # Harusnya (jumlah_training, jumlah_fitur)
print("Shape of X_test:", X_test.shape)    # Harusnya (jumlah_testing, jumlah_fitur)
print("Shape of y_train:", y_train.shape)  # Harusnya (jumlah_training,)
print("Shape of y_test:", y_test.shape)    # Harusnya (jumlah_testing,)
```

```
Data Setelah Feature Scaling:
         Age    Income  Education_Level  Credit_Score  Loan_Amount  \
0  1.100655 -1.496205                3     -1.500286     0.026245
1  0.353029  0.809486                2     -1.606833     0.006629
2 -0.693647  1.606651                3     -0.754461    -1.813972
3  1.399705  1.753674                1      0.655712     0.136035
4 -1.216985 -0.308387                3      0.430084    -0.730507

   Loan_Purpose
0             4
1             2
2             4
3             4
4             0
Shape of X_train: (400, 6)
Shape of X_test: (100, 6)
Shape of y_train: (400,)
Shape of y_test: (100,)
```

3.Pemilihan dan Training Model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Melatih Model Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Memprediksi dan mengevaluasi model Logistic Regression
y_pred_logistic = logistic_model.predict(X_test)
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
print("Akurasi Logistic Regression:", accuracy_logistic)
print("Laporan Klasifikasi Logistic Regression:\n", classification_report(y_test, y_pred_logistic))

# Melatih Model Random Forest
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=36)
random_forest_model.fit(X_train, y_train)

# Memprediksi dan mengevaluasi model Random Forest
y_pred_rf = random_forest_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Akurasi Random Forest:", accuracy_rf)
print("Laporan Klasifikasi Random Forest:\n", classification_report(y_test, y_pred_rf))
```

```
Akurasi Logistic Regression: 0.63
Laporan Klasifikasi Logistic Regression:
               precision    recall  f1-score   support

           0       0.33      0.03      0.05        36
           1       0.64      0.97      0.77        64

    accuracy                           0.63       100
   macro avg       0.49      0.50      0.41       100
weighted avg       0.53      0.63      0.51       100
```

```
Akurasi Random Forest: 0.53
Laporan Klasifikasi Random Forest:
              precision    recall  f1-score   support

           0       0.24      0.14      0.18        36
           1       0.61      0.75      0.67        64

    accuracy                           0.53       100
   macro avg       0.42      0.44      0.42       100
weighted avg       0.47      0.53      0.49       100
```

Adapun alasan mengapa memilih Logistic Regression dan Random Forest untuk menganalisis data pinjaman karena kedua metode ini sangat cocok untuk menangani masalah yang melibatkan lebih dari dua kategori. Logistic Regression, meskipun awalnya dirancang untuk memprediksi dua hasil (misalnya, disetujui atau tidak disetujui), dapat disesuaikan untuk memprediksi beberapa kategori dengan cara yang sederhana dan mudah dipahami. Hal ini dapat memberikan gambaran mengenai seberapa besar kemungkinan suatu pinjaman akan disetujui berdasarkan berbagai faktor.

Sedangkan untuk Random Forest ialah metode yang lebih kuat yang menggunakan banyak pohon keputusan untuk membuat prediksi. Hal ini dapat membantu meningkatkan akurasi dan mengurangi risiko kesalahan, terutama ketika data yang digunakan cukup kompleks.

```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Menentukan parameter yang ingin dicari untuk Logistic Regression
param_grid_logistic = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Nilai regularisasi
    'penalty': ['l1', 'l2'],        # Jenis regularisasi
    'solver': ['liblinear' , 'saga']         # Solver yang digunakan
}


# Menggunakan Grid Search untuk menemukan kombinasi terbaik
grid_search_logistic = GridSearchCV(estimator=logistic_model, param_grid=param_grid_logistic, cv=5, scoring='accuracy')
grid_search_logistic.fit(X_train, y_train)

# Menampilkan kombinasi hyperparameter terbaik untuk Logistic Regression
print("Kombinasi Hyperparameter Terbaik untuk Logistic Regression:")
print(grid_search_logistic.best_params_)
```

```
Kombinasi Hyperparameter Terbaik untuk Logistic Regression:
{'C': 0.001, 'penalty': 'l1', 'solver': 'saga'}
```

Adapun kombinasi hyperparameter terbaik ialah {'C': 0.001, 'penalty': 'l1', 'solver': 'saga'}

```python
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score, f1_score # Import precision

# Menggunakan model terbaik setelah tuning untuk Logistic Regression
best_logistic_model = grid_search_logistic.best_estimator_
y_pred_best_logistic = best_logistic_model.predict(X_test)

# Menghitung metrik untuk Logistic Regression setelah tuning
accuracy_logistic_after = accuracy_score(y_test, y_pred_best_logistic)
precision_logistic_after = precision_score(y_test, y_pred_best_logistic)
recall_logistic_after = recall_score(y_test, y_pred_best_logistic)
f1_logistic_after = f1_score(y_test, y_pred_best_logistic)

# Menampilkan hasil untuk Logistic Regression setelah tuning
print("\nMetric Evaluasi untuk Logistic Regression (Setelah Tuning):")
print(f"Akurasi: {accuracy_logistic_after:.4f}")
print(f"Precision: {precision_logistic_after:.4f}")
print(f"Recall: {recall_logistic_after:.4f}")
print(f"F1-Score: {f1_logistic_after:.4f}")
```

```

Metric Evaluasi untuk Logistic Regression (Setelah Tuning):
Akurasi: 0.6400
Precision: 0.6400
Recall: 1.0000
F1-Score: 0.7805
```

```
# Tabel perbandingan sebelum dan sesudah tunning
accuracy_logistic_before = accuracy_score(y_test, y_pred_logistic)
precision_logistic_before = precision_score(y_test, y_pred_logistic)
recall_logistic_before = recall_score(y_test, y_pred_logistic)
f1_logistic_before = f1_score(y_test, y_pred_logistic)

results = {
    'Metric': ['Akurasi', 'Precision', 'Recall', 'F1-Score'],
    'Sebelum Tuning': [accuracy_logistic_before, precision_logistic_before, recall_logistic_before, f1_logistic_before],
    'Setelah Tuning': [accuracy_logistic_after, precision_logistic_after, recall_logistic_after, f1_logistic_after]
}

results_df = pd.DataFrame(results)

# Menampilkan tabel
print("\nTabel Perbandingan Hasil Tuning:")
print(results_df)
```

```
Tabel Perbandingan Hasil Tuning:
      Metric  Sebelum Tuning  Setelah Tuning
0    Akurasi        0.630000        0.640000
1  Precision        0.639175        0.640000
2     Recall        0.968750        1.000000
3   F1-Score        0.770186        0.780488
```

Meskipun peningkatan akurasi hanya 1%, hasil tuning menunjukkan perbaikan yang lebih signifikan dalam metrik lain, terutama recall yang mencapai 100%. Ini menunjukkan bahwa tuning berhasil meningkatkan performa model dalam hal kemampuan untuk mendeteksi