Berikut adalah contoh proyek dunia nyata yang menggabungkan data dari file Excel dan CSV untuk melakukan analisis data, mulai dari pembersihan data (data cleaning) hingga visualisasi. Proyek ini bisa diaplikasikan dalam berbagai konteks, misalnya analisis penjualan atau performa produk.

Contoh Kasus: Analisis Penjualan Produk

Deskripsi Proyek:

Proyek ini bertujuan untuk menganalisis data penjualan dari dua sumber:

1. **File Excel** yang berisi informasi produk seperti harga dan kategori.
2. **File CSV** yang berisi data transaksi penjualan dari berbagai toko.

Langkah-langkah yang akan dilakukan:

1. **Mengimpor data** dari Excel dan CSV.
2. **Data Cleaning** untuk menangani nilai yang hilang, duplikat, dan format yang tidak sesuai.
3. **Penggabungan data** dari kedua sumber untuk membentuk satu dataset yang komprehensif.
4. **Analisis data** untuk mendapatkan insight penting seperti total penjualan, produk terlaris, dll.
5. **Visualisasi data** untuk mempresentasikan hasil analisis.

1. Setup Lingkungan dan Installasi Library

Instalasi beberapa library Python yang diperlukan:

```
pip install pandas matplotlib seaborn openpyxl
```

2. Mengimpor Data dari Excel dan CSV

```
import pandas as pd

# Mengimpor data produk dari file Excel
product_data = pd.read_excel('data/products.xlsx')

# Mengimpor data transaksi dari file CSV
transaction_data = pd.read_csv('data/transactions.csv')
```

3. Data Cleaning

```
# Menghilangkan baris yang memiliki nilai kosong
product_data.dropna(inplace=True)
transaction_data.dropna(inplace=True)

# Menghapus duplikat, jika ada
product_data.drop_duplicates(inplace=True)
transaction_data.drop_duplicates(inplace=True)

# Pastikan tipe data sesuai (misalnya, kolom harga adalah numerik)
product_data['Price'] = pd.to_numeric(product_data['Price'],
errors='coerce')
```

4. Penggabungan Data

Gabungkan data produk dengan data transaksi berdasarkan `Product ID`:

```
# Menggabungkan data produk dan transaksi berdasarkan Product ID
merged_data = pd.merge(transaction_data, product_data, on='Product
ID')

# Menampilkan data yang telah digabungkan
print(merged_data.head())
```

5. Analisis Data

Misalnya, kita ingin mengetahui total penjualan per produk:

```
# Menambahkan kolom 'Total Sales' berdasarkan kuantitas dan harga
produk
merged_data['Total Sales'] = merged_data['Quantity'] *
merged_data['Price']

# Menghitung total penjualan per produk
sales_per_product = merged_data.groupby('Product Name')['Total
Sales'].sum().reset_index()

# Mengurutkan produk berdasarkan total penjualan
sales_per_product = sales_per_product.sort_values(by='Total Sales',
ascending=False)

print(sales_per_product.head())
```

6. Visualisasi Data

Visualisasi produk terlaris menggunakan `matplotlib` dan `seaborn`:

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

# Visualisasi produk terlaris
plt.figure(figsize=(10, 6))
sns.barplot(x='Total Sales', y='Product Name',
data=sales_per_product.head(10))
plt.title('Top 10 Best Selling Products')
plt.xlabel('Total Sales')
plt.ylabel('Product Name')
plt.show()
```

7. Menyimpan Hasil Analisis

Jika Anda ingin menyimpan hasil analisis ke dalam file Excel atau CSV:

```
# Menyimpan hasil analisis ke dalam file Excel
sales_per_product.to_excel('sales_analysis.xlsx', index=False)

# Menyimpan hasil analisis ke dalam file CSV
sales_per_product.to_csv('sales_analysis.csv', index=False)
```

Penjelasan:

1. **Data Cleaning:** Penting untuk menghilangkan nilai kosong dan duplikat yang dapat mengganggu analisis. Mengubah tipe data juga diperlukan untuk memastikan operasi matematika seperti perhitungan total penjualan berjalan dengan baik.
2. **Penggabungan Data:** Data dari dua sumber yang berbeda digabungkan untuk memberikan konteks penuh—misalnya, data transaksi tanpa informasi produk tidak akan cukup untuk analisis penjualan.
3. **Analisis:** Menggunakan agregasi (`groupby`) untuk meringkas data dan mendapatkan wawasan seperti produk mana yang paling laris.
4. **Visualisasi:** Digunakan untuk mempermudah interpretasi data dan mengomunikasikan hasil analisis dengan lebih efektif.

File dan Struktur Folder:

- **data/products.xlsx**: File Excel yang berisi informasi produk (`Product ID`, `Product Name`, `Category`, `Price`).
- **data/transactions.csv**: File CSV yang berisi data transaksi (`Transaction ID`, `Product ID`, `Quantity`, `Transaction Date`).
- **sales_analysis.xlsx**: File hasil analisis dalam format Excel.
- **sales_analysis.csv**: File hasil analisis dalam format CSV.

Dengan mengikuti langkah-langkah di atas, Anda dapat melakukan analisis lengkap menggunakan data dari berbagai sumber, dan menggunakannya untuk mengambil keputusan bisnis yang lebih baik.

Memilih estimator yang tepat di scikit-learn tergantung pada jenis masalah yang ingin kamu selesaikan, seperti apakah itu melibatkan klasifikasi, clustering, regresi, atau pengurangan dimensi. Berikut ringkasannya dalam bahasa yang mudah dipahami:

Klasifikasi

- **Jika kamu memiliki data yang sudah diberi label dan ingin memprediksi kategori:**
  - **< 100.000 sampel:** Mulai dengan **LinearSVC**. Jika data berupa teks, coba **NaiveBayes**. Jika bukan, pertimbangkan **KNeighborsClassifier** atau **SVC**.
  - **> 100.000 sampel:** Pertimbangkan **SGDClassifier**. Jika ini tidak berhasil, coba teknik **kernel approximation**.

Clustering

- **Jika kamu tidak memiliki data yang diberi label dan ingin menemukan pola:**
  - **Jumlah kategori diketahui:**
    - **< 10.000 sampel:** Mulai dengan **KMeans**. Jika tidak berhasil, pertimbangkan **SpectralClustering** atau **GaussianMixture**.
    - **> 10.000 sampel:** Gunakan **MiniBatchKMeans**.
  - **Jumlah kategori tidak diketahui:**
    - **< 10.000 sampel:** Coba **MeanShift** atau **VBGMM**.
    - **> 10.000 sampel:** scikit-learn mungkin tidak menyediakan solusi yang sesuai.

Regresi

- **Jika kamu ingin memprediksi nilai numerik:**
  - **< 100.000 sampel:**
    - Dimensi rendah: **Lasso** atau **ElasticNet**.
    - Dimensi tinggi: **RidgeRegression** atau **SVR** dengan kernel linear.
  - **> 100.000 sampel:** **SGDRegressor** direkomendasikan.

Pengurangan Dimensi

- **Jika kamu sedang menjelajahi data untuk mengidentifikasi fitur penting:**
  - **< 10.000 sampel:** Mulai dengan **PCA**. Jika tidak berhasil, coba **Isomap**, **SpectralEmbedding**, atau **LLE**.
  - **> 10.000 sampel:** Pertimbangkan teknik **kernel approximation**.

Kerangka ini akan membantumu menggunakan scikit-learn dengan efektif, memastikan kamu memilih estimator yang tepat sesuai dengan masalah yang ingin diselesaikan.

Untuk menentukan apakah estimator yang kamu pilih sudah tepat dalam menyelesaikan masalah tertentu, kamu perlu melakukan evaluasi dan analisis terhadap performa model tersebut. Berikut adalah langkah-langkah dan ciri-ciri yang bisa membantu kamu menentukan kecocokan estimator:

---

1. Evaluasi Performa Model dengan Metrik yang Sesuai

---

a. Pilih Metrik yang Tepat

- **Klasifikasi:**
  - **Akurasi:** Persentase prediksi yang benar dari total prediksi.
  - **Presisi:** Ketepatan prediksi positif model.
  - **Recall (Sensitivitas):** Kemampuan model dalam mendeteksi semua sampel positif.
  - **F1-Score:** Harmonik rata-rata dari presisi dan recall.
  - **ROC-AUC:** Mengukur kemampuan model dalam membedakan kelas.

- **Regresi:**
  - **Mean Absolute Error (MAE):** Rata-rata selisih absolut antara prediksi dan nilai aktual.
  - **Mean Squared Error (MSE):** Rata-rata kuadrat selisih antara prediksi dan nilai aktual.
  - **Root Mean Squared Error (RMSE):** Akar kuadrat dari MSE.
  - **R-squared (R^2):** Proporsi variansi yang dijelaskan oleh model.

b. Evaluasi dengan Data Uji

- **Split Data:** Bagi dataset menjadi data latih dan data uji.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

- **Latih Model dan Prediksi:**

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

- **Hitung Metrik:**

```
from sklearn.metrics import accuracy_score, mean_absolute_error

accuracy = accuracy_score(y_test, y_pred)  # Untuk klasifikasi
mae = mean_absolute_error(y_test, y_pred)  # Untuk regresi
```

- **Interpretasi Hasil:**
  - **Performa Tinggi:** Nilai metrik yang menunjukkan prediksi akurat dan konsisten.
  - **Performa Rendah:** Nilai metrik yang menunjukkan prediksi kurang akurat.

2. Validasi dengan Cross-Validation

- **Tujuan:** Memastikan model memiliki performa yang stabil dan tidak overfitting atau underfitting.

```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(model, X, y, cv=5)
print("Cross-validation scores:", cv_scores)
print("Mean CV score:", cv_scores.mean())
```

- **Ciri-ciri:**
  - **Stabil:** Skor konsisten di berbagai fold.
  - **Tidak Overfitting:** Performa baik pada data latih dan data uji.
  - **Tidak Underfitting:** Model mampu menangkap pola dalam data dengan baik.

3. Analisis Error dan Residual

- **Untuk Regresi:**
  - **Plot Residual:** Memeriksa distribusi error.

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

residuals = y_test - y_pred
sns.histplot(residuals, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals')
plt.show()
```

  - **Ciri-ciri Model Baik:**
    - Residual tersebar secara acak di sekitar nol.
    - Tidak ada pola yang jelas pada plot residual.
- **Untuk Klasifikasi:**
  - **Confusion Matrix:** Memeriksa detail prediksi benar dan salah.

```
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```

  - **Ciri-ciri Model Baik:**

    - Tingkat true positives dan true negatives tinggi.
    - Tingkat false positives dan false negatives rendah.

4. Bandingkan dengan Model Lain

- **Uji Beberapa Estimator:**
    - Coba beberapa algoritma berbeda dan bandingkan performanya.

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier()
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy}")
```

- **Pilih Model Terbaik:**
    - Pilih estimator dengan performa terbaik berdasarkan metrik evaluasi.

5. Perhatikan Kompleksitas dan Waktu Komputasi

- **Efisiensi:** Pastikan model tidak terlalu kompleks sehingga memakan waktu komputasi yang lama tanpa peningkatan performa yang signifikan.
- **Ciri-ciri:**
    - Waktu training dan prediksi yang wajar.
    - Skalabilitas terhadap dataset yang lebih besar.

6. Sesuaikan dengan Asumsi dan Karakteristik Data

- **Periksa Asumsi Model:**
  - Beberapa estimator memiliki asumsi tertentu (misalnya, linearitas, normalitas).
  - Pastikan data memenuhi asumsi-asumsi tersebut atau lakukan transformasi yang diperlukan.
- **Contoh:**
  - **Regresi Linear:** Mengasumsikan hubungan linear antara fitur dan target.
  - **Decision Tree:** Tidak terlalu terpengaruh oleh distribusi data dan dapat menangani data non-linear.

7. Lakukan Hyperparameter Tuning

- **Tujuan:** Mengoptimalkan performa model dengan mencari kombinasi parameter terbaik.

```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [100, 200], 'max_depth': [10, 20]}
grid_search = GridSearchCV(RandomForestClassifier(), param_grid,
cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)
```

- **Ciri-ciri Model Optimal:**
  - Mencapai performa terbaik pada data uji setelah tuning.
  - Menghindari overfitting dengan memilih parameter yang sesuai.

8. Validasi dengan Data Nyata

- **Uji di Lingkungan Nyata:**
  - Terapkan model pada data dunia nyata dan evaluasi performanya.
- **Feedback dan Penyesuaian:**

- Kumpulkan feedback dan lakukan penyesuaian jika diperlukan untuk memastikan model tetap relevan dan akurat.

Kesimpulan:
Estimator yang tepat ditandai dengan performa yang baik dan konsisten berdasarkan metrik evaluasi yang sesuai, mampu generalisasi dengan baik pada data baru, efisien secara komputasi, dan sesuai dengan karakteristik serta asumsi data yang digunakan. Proses evaluasi dan tuning yang sistematis akan membantu memastikan bahwa estimator yang dipilih adalah yang paling cocok untuk masalah yang ingin diselesaikan.

Jika setelah melalui semua langkah ini model masih belum memberikan hasil yang memuaskan, pertimbangkan untuk mencoba estimator lain atau mengumpulkan dan memproses data tambahan untuk meningkatkan kualitas model.

Untuk menjelajahi dataset **California Housing** yang diambil dari Kaggle, berikut adalah langkah-langkah yang bisa kamu lakukan menggunakan **Pandas** dan alat visualisasi seperti **Matplotlib** dan **Seaborn**:

1. Mengimpor Paket yang Diperlukan

- Mulai dengan mengimpor paket-paket yang diperlukan:

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

Jika belum terinstal, kamu bisa menginstal **Seaborn** dengan perintah berikut:

```
pip install seaborn
```

2. Membaca Dataset

- Baca file CSV yang berisi data perumahan California:

```
housing_data = pd.read_csv('datasets/house.csv')
```

- Gunakan fungsi `head()` untuk melihat beberapa baris pertama dari dataset:

```
print(housing_data.head())
```

3. Eksplorasi Dataset

- Lihat jumlah baris dan kolom dalam dataset dengan `shape`:

```
print(housing_data.shape)
```

- Tampilkan sampel acak dari data untuk melihat variasi data yang ada:

```
print(housing_data.sample(5))
```

- Gunakan `describe()` untuk mendapatkan statistik dasar tentang kolom-kolom numerik:

```
print(housing_data.describe())
```

- Eksplorasi kolom **categorical** `ocean_proximity` untuk melihat kategori yang ada:

```
print(housing_data['ocean_proximity'].unique())
```

4. Menangani Missing Values

- Jika dataset memiliki nilai yang hilang, kamu bisa menghapus baris yang memiliki missing values dengan `dropna()`:

```
housing_data = housing_data.dropna()
print(housing_data.shape)  # Lihat berapa banyak baris yang tersisa
setelah pembersihan
```

5. Visualisasi Data

- **Distribusi Data Numerik**: Visualisasikan distribusi dari kolom numerik menggunakan Seaborn:

```
sns.histplot(housing_data['median_income'], bins=30, kde=True)
plt.show()
```

- **Distribusi Data Kategorikal**: Tampilkan jumlah setiap kategori dalam `ocean_proximity`:

```
sns.countplot(data=housing_data, x='ocean_proximity')
plt.show()
```

- **Hubungan Antar Variabel**: Gunakan scatter plot untuk melihat hubungan antara pendapatan median dengan nilai rumah median:

```
sns.scatterplot(data=housing_data, x='median_income',
y='median_house_value', hue='ocean_proximity')
plt.show()
```

Insight dari Dataset

- Dataset ini memiliki 20.433 catatan setelah menangani missing values, dan 10 kolom (9 fitur dan 1 target).
- Kolom `ocean_proximity` adalah kolom kategorikal yang memiliki beberapa nilai unik seperti `near_bay`, `inland`, `island`, dll.
- Kolom numerik seperti `housing_median_age`, `total_rooms`, dan `median_income` memberikan detail statistik dasar, seperti rata-rata dan deviasi standar, yang bisa memberikan wawasan awal tentang distribusi data.
- Harga median rumah (`median_house_value`) mungkin dipengaruhi oleh `ocean_proximity` dan `median_income`.

Langkah-langkah ini memberikan eksplorasi dasar dari dataset California Housing dan memungkinkan kamu untuk memulai analisis lebih lanjut atau pembangunan model machine learning.

Memahami Regresi Linear

Regresi linear adalah teknik dasar dalam pembelajaran mesin dan statistika yang digunakan untuk memprediksi nilai numerik kontinu berdasarkan satu atau lebih fitur input.

Berikut adalah penjelasan mengenai konsep-konsep kunci dalam regresi linear:

Apa Itu Regresi?

Regresi digunakan untuk memprediksi nilai numerik (`Y`) berdasarkan fitur input (`X`). Contohnya termasuk memprediksi harga rumah, harga saham, atau perkiraan waktu tunggu. Dalam regresi, `Y` adalah variabel dependen, dan `X` adalah variabel independen (atau variabel penjelas). Tujuan dari regresi adalah menentukan hubungan antara `X` dan `Y`.

Regresi Linear yang Disederhanakan

Regresi linear mengasumsikan bahwa hubungan antara `X` dan `Y` dapat direpresentasikan sebagai garis lurus. Bentuk paling sederhana dari regresi linear melibatkan satu variabel independen (`X`) dan satu variabel dependen (`Y`).

Persamaan garis biasanya ditulis sebagai:

$$y = mx + c$$

- **m**: Kemiringan garis (berapa banyak Y berubah untuk setiap perubahan satuan pada X).
- **c**: Intersep Y (nilai Y ketika X adalah 0).

Menemukan Garis yang Paling Sesuai

Dalam regresi linear, garis yang paling sesuai adalah garis yang meminimalkan kesalahan antara titik data aktual dan garis itu sendiri. Kesalahan ini dihitung menggunakan metode **Least Square Error**.

Least Square Error

1. **Jarak Vertikal (Kesalahan)**: Untuk setiap titik data, jarak vertikal ke garis mewakili kesalahan (selisih antara Y yang diprediksi dan Y aktual).
2. **Jumlah Kesalahan Kuadrat**: Untuk menghindari kesalahan negatif yang membatalkan kesalahan positif, jarak-jarak ini dikuadratkan dan dijumlahkan.
3. **Minimisasi**: Tujuannya adalah menemukan garis di mana jumlah kesalahan kuadrat ini diminimalkan.

Secara matematis, kesalahan untuk setiap titik adalah:

$$Error = (yaktual - yprediksi)2$$

dimana:

$$yprediksi = mx + c$$

Menggunakan Scikit-Learn untuk Regresi Linear

Saat menggunakan pustaka seperti Scikit-learn untuk melakukan regresi linear, pustaka tersebut secara otomatis akan melakukan:

- Menghitung garis terbaik dengan meminimalkan least square error.
- Model ini kemudian dapat digunakan untuk membuat prediksi untuk titik data baru dengan hanya menerapkan persamaan garis.

Ringkasan

Regresi linear adalah teknik sederhana namun kuat untuk memodelkan hubungan antara variabel-variabel. Dengan memasang garis terbaik pada data Anda, Anda dapat membuat prediksi dan mendapatkan wawasan tentang bagaimana perubahan pada variabel independen mungkin mempengaruhi variabel dependen. Metode least square error adalah alat matematika kunci yang memastikan kecocokan terbaik.

Persiapan Data untuk Pembelajaran Mesin

Dalam demonstrasi ini, kita akan menggunakan estimator dari scikit-learn untuk membangun model pembelajaran mesin sederhana yang melakukan regresi linear. Kita akan melatih model kita menggunakan *dataset* California housing yang sudah kita eksplorasi sebelumnya dan menggunakannya untuk memprediksi harga rumah. Kita akan membangun solusi ini di dalam *Jupyter Notebook* dengan nama `LinearRegressionForPricePrediction`.

Langkah-Langkah:

1. **Impor Pustaka yang Diperlukan**:
   Pertama, kita akan mengimpor beberapa pustaka Python yang akan kita gunakan, seperti `Pandas` dan modul `pyplot` dari `Matplotlib`.
2. **Memuat Dataset**:
   Kita menggunakan `pd.read_csv` untuk membaca file CSV yang berisi *dataset* kita yang berada di `datasets/housing.csv`. Setelah data dimuat, kita akan membersihkan semua data yang memiliki nilai hilang dengan menggunakan fungsi `dropna`. Hasilnya, kita akan memiliki sekitar 20.433 *record* yang siap digunakan.

3. **Mengatasi Skewed Data**:
   Kita telah mengamati bahwa beberapa nilai harga rumah dalam *dataset* ini berkumpul di sekitar $500.000, yang tampaknya merupakan batas atas harga rumah. Hal ini dapat menyebabkan *skewed data* yang dapat memengaruhi pelatihan model kita. Oleh karena itu, kita akan menghapus *record* yang memiliki nilai `median_house_value` sebesar $500.001 dari *dataset* kita. Setelah dihapus, kita akan memiliki 19.475 *record* yang akan digunakan untuk membangun dan melatih model pembelajaran mesin kita.

4. **Mengonversi Data Kategorikal ke Bentuk Numerik**:
   Pada *dataset* ini, semua fitur memiliki nilai numerik kecuali kolom `ocean_proximity`, yang berisi nilai kategorikal atau diskrit. Karena model pembelajaran mesin hanya dapat bekerja dengan data numerik, kita perlu mengonversi nilai kategorikal ini menjadi bentuk numerik menggunakan teknik *one-hot encoding*. Kita akan menggunakan fungsi `pd.get_dummies` untuk mengonversi kolom `ocean_proximity` ke bentuk *one-hot* dan menghapus kolom asli dari *dataset* kita. Setelah dikonversi, jumlah kolom dalam *dataset* kita akan meningkat dari 10 menjadi 14.

5. **Mempersiapkan Fitur dan Target**:
   Selanjutnya, kita akan mempersiapkan fitur (X) dan target (Y) untuk model kita. Fitur (X) adalah semua kolom input kecuali `median_house_value`, sedangkan target (Y) adalah kolom `median_house_value` yang ingin kita prediksi. Kita akan membuat *dataframe* baru untuk X dengan menghapus kolom `median_house_value` dari *dataset* asli kita.

6. **Membagi Dataset menjadi Training Set dan Test Set**:
   Praktik umum dalam pelatihan model pembelajaran mesin adalah membagi *dataset* menjadi *training set* dan *test set*. *Training set* digunakan untuk melatih parameter model, sementara *test set* digunakan untuk mengukur kinerja model pada data yang belum pernah dilihat sebelumnya. Fungsi `train_test_split` dari pustaka scikit-learn memungkinkan kita untuk dengan cepat membagi *dataset* menjadi *training set* dan *test set*. Biasanya, kita melakukan pembagian 80/20, di mana 80% data asli digunakan untuk melatih model, dan 20% sisanya digunakan untuk menguji model. Data juga akan diacak untuk menghindari model mempelajari pola yang tidak diinginkan dalam data.

7. **Mengecek Bentuk Data**:
   Setelah melakukan pembagian, kita akan memeriksa bentuk dari X dan Y pada *training set* dan *test set*. Sebagai contoh, kita mungkin memiliki sekitar 15.580 *record* untuk melatih model kita dan sisanya, yaitu 3.895 *record*, akan digunakan untuk menguji model.

8. **Melatih Model Regresi Linear**:
   Sekarang kita siap untuk menggunakan estimator scikit-learn pertama kita untuk melakukan regresi linear, melatih model, dan menggunakannya untuk membuat prediksi.

Dengan persiapan ini, kita dapat melanjutkan ke langkah berikutnya yaitu melatih dan menguji model pembelajaran mesin kita.

Berikut adalah panduan langkah demi langkah dalam Bahasa Indonesia baku untuk melatih dan melakukan prediksi menggunakan model regresi linier dengan scikit-learn, menggunakan dataset harga rumah di California:

1. Impor Pustaka yang Diperlukan

Pertama, impor pustaka yang diperlukan untuk manipulasi data dan regresi linier.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
```

2. Muat dan Siapkan Data

Muat dataset dan lakukan persiapan data awal, termasuk menangani nilai yang hilang dan mengubah fitur kategorikal menjadi fitur numerik menggunakan one-hot encoding.

```
# Muat dataset
housing_data = pd.read_csv('datasets/housing.csv')

# Hapus nilai yang hilang
housing_data = housing_data.dropna()

# Hapus rekaman dengan nilai rumah maksimal untuk menghindari pengaruh
yang tidak diinginkan pada model
housing_data = housing_data[housing_data['median_house_value'] <
500001]

# Lakukan one-hot encoding pada kolom 'ocean_proximity'
housing_data = pd.get_dummies(housing_data, columns=
['ocean_proximity'])
```

```
# Pisahkan fitur (X) dan target (y)
X = housing_data.drop('median_house_value', axis=1)
y = housing_data['median_house_value']
```

3. Bagi Data Menjadi Set Pelatihan dan Uji

Bagi data menjadi set pelatihan dan set uji menggunakan pembagian 80/20.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

4. Latih Model Regresi Linier

Buat objek estimator `LinearRegression`, latih model menggunakan data pelatihan, dan lakukan normalisasi fitur.

```
# Buat model regresi linier
linear_model = LinearRegression(normalize=True)

# Latih model
linear_model.fit(X_train, y_train)
```

5. Evaluasi Model

Evaluasi model menggunakan data pelatihan dan lihat koefisien yang dihasilkan.

```
# Hitung R-squared pada data pelatihan
train_score = linear_model.score(X_train, y_train)
print(f"R-squared Pelatihan: {train_score:.2f}")

# Lihat koefisien model
coefficients = pd.Series(linear_model.coef_, index=X_train.columns)
coefficients.sort_values(ascending=False, inplace=True)
print(coefficients)
```

6. Lakukan Prediksi pada Data Uji

Lakukan prediksi menggunakan data uji dan evaluasi kinerja model.

```
# Prediksi pada data uji
y_pred = linear_model.predict(X_test)

# Hitung R-squared pada data uji
test_score = r2_score(y_test, y_pred)
print(f"R-squared Uji: {test_score:.2f}")
```

7. Visualisasikan Prediksi

Buat grafik scatter untuk memvisualisasikan nilai aktual vs. nilai prediksi.

```
# Grafik scatter dari nilai aktual vs. prediksi
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Harga Aktual")
plt.ylabel("Harga Prediksi")
plt.title("Harga Rumah Aktual vs. Prediksi")
plt.show()
```

8. Contohkan Prediksi dan Visualisasikan

Ambil sampel subset prediksi untuk memvisualisasikan kinerja model dengan lebih baik.

```
# Ambil sampel 100 prediksi dan atur ulang indeksnya
sample_data = pd.DataFrame({'Aktual': y_test, 'Prediksi':
y_pred}).sample(100).reset_index(drop=True)

# Buat grafik prediksi sampel
plt.plot(sample_data['Aktual'], label='Aktual', color='orange')
plt.plot(sample_data['Prediksi'], label='Prediksi', color='blue')
plt.legend()
plt.title("Harga Aktual vs. Prediksi pada Sampel")
plt.show()
```

Ringkasan

Anda telah berhasil membangun model regresi linier menggunakan scikit-learn, melatihnya dengan dataset harga rumah di California, dan melakukan prediksi. Anda juga telah mengevaluasi kinerja model dan memvisualisasikan hasilnya menggunakan nilai R-squared dan grafik scatter.