Laporan Proyek Praktik: Pra-pemrosesan Data II (Feature Engineering)

1. Inisialisasi Environment

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE # Pastikan library ini sudah
terinstal!
import warnings
# Atur tampilan plot
sns.set_style("whitegrid")
warnings.filterwarnings('ignore')
```

Kode ini pada dasarnya adalah **"persiapan alat tempur"** sebelum kita benar-benar menjalankan proses *Feature Optimization* dan *Modelling*. Program ini memastikan semua *tools* dan *library* yang kita butuhkan sudah siap di-panggil.

Paragraf pertama isinya adalah **import** semua *library* wajib dan pendukung. Kita panggil **pandas** (pd) buat ngurusin data tabular (mirip *Excel* canggih kita), dan **numpy** (np) buat urusan komputasi numerik yang berat-berat. Setelah itu, kita impor **matplotlib.pyplot** (plt) dan **seaborn** (sns); dua ini adalah *tool* andalan kita buat **visualisasi data**, entah itu bikin *histogram*, *scatterplot*, atau yang paling penting di laporan ini, **grafik LASSO dan PCA**.

Kemudian, kita masuk ke inti *Machine Learning* dari **sklearn**. Kita impor **LogisticRegression** buat *fitting* model klasifikasi yang akan kita pakai sebagai *base* untuk **Seleksi Fitur (LASSO)**, yang dikombinasikan dengan **SelectFromModel**. Lalu, kita impor **PCA** (*Principal Component Analysis*) yang fungsinya buat **Reduksi Dimensi** atau *ngeringkes* informasi dari fitur-fitur numerik.

Poin krusialnya ada di impor **SMOTE** dari **imblearn.over_sampling**. Ini penting banget kalau *dataset* target Klasifikasi kita **enggak seimbang** (misalnya, yang Baik 90% dan yang Bermasalah cuma 10%). SMOTE ini *tool* wajib buat bikin *data synthetic* biar kelas minoritasnya (*minority class*) ikutan banyak, jadi model kita nggak *bias* atau berat sebelah.

Terakhir, ada *settingan* tambahan. Kita impor **warnings** dan setel agar semua *warning* diabaikan (warnings.filterwarnings('ignore'))—biar *output* kita bersih dari notifikasi yang nggak penting. Dan buat estetika, kita setel *style* grafik **seaborn** ke **"whitegrid"** supaya grafik

yang muncul rapi dan profesional. Setelah semua ini selesai di-run, *environment* kita sudah siap tempur buat proses *Feature Optimization* selanjutnya.

2. Memuat Data Preprocessed

Bagian ini memuat dataset yang sudah melalui tahap preprocessing sebelumnya. Target klasifikasi ditentukan pada kolom 'Job'. Selain itu, kolom numerik seperti Age dan Duration disiapkan sebagai input khusus untuk PCA.

```
print("1. MEMUAT DATA KLASIFIKASI...")
# Asumsi: Muat data yang sudah bersih dan ter-scaling dari Pra-pemrosesan I
try:
    train_df = pd.read_csv('data_train_preprocessed.csv')
    test_df = pd.read_csv('data_test_preprocessed.csv')
except FileNotFoundError:
    print("ERROR: Pastikan file 'data_train_preprocessed.csv' dan
'data_test_preprocessed.csv' tersedia.")
    raise
# GANTI INI: Kolom target Klasifikasi Anda
TARGET_CLASS = 'Job'
y_class_train = train_df[TARGET_CLASS]
X_class_train = train_df.drop(TARGET_CLASS, axis=1)
y_class_test = test_df[TARGET_CLASS]
X_class_test = test_df.drop(TARGET_CLASS, axis=1)
# GANTI INI: Kolom numerik yang akan dimasukkan ke PCA
NUMERICAL_COLS_CLASS = ['Age', 'Duration']
print(f"Dimensi Awal Fitur (X_class_train): {X_class_train.shape}")
```

a. Proses *Loading* Data Awal (The Crucial Step)

Paragraf ini adalah *starting point* program kita. Kode ini tujuannya cuma satu: **ngambil data yang sudah capek-capek kita bersihin di Tahap I**. Program ini coba *load* dua *file* penting, **data_train_preprocessed.csv** dan **data_test_preprocessed.csv**, pakai pd.read_csv. Nah, karena ini adalah *critical step*, *kodingan*-nya dibungkus pakai try...except FileNotFoundError. Ini penting banget! Kalau *file*-nya nggak ketemu di *folder* yang sama, program nggak langsung *crash*, tapi ngasih *warning* jelas (**ERROR: Pastikan file... tersedia**) dan langsung raise supaya kita tahu *error* utamanya ada di mana. Intinya, kalau sampai di sini gagal, proses *Feature Optimization* nggak bisa lanjut.

b. Penentuan Target dan Pemisahan X & Y

Setelah *data frame* berhasil di-*load*, *step* berikutnya adalah **mendefinisikan siapa target kita** dan **memisahkan variabel independen (*features*)**. Di sini, kita *set* **TARGET_CLASS = 'Job'**—ini artinya, *goal* kita di tugas Klasifikasi ini adalah memprediksi **Job** si nasabah. Kemudian, kita melakukan *split* dasar: y_class_train dan y_class_test itu isinya cuma kolom Job (ini target kita). Sedangkan X_class_train dan X_class_test itu isinya **SEMUA kolom lain** (ini fitur yang akan kita pakai buat memprediksi Job). Proses *dropping* kolom target ini harus dilakukan dengan hati-hati.

c. Setup Kolom Numerik untuk PCA

Paragraf ini adalah persiapan khusus untuk **Skenario B: Reduksi Dimensi (PCA)**. Karena **PCA itu hanya bisa bekerja optimal di data numerik yang sudah ter-*scaling***, kita harus *define* secara eksplisit kolom-kolom mana yang akan kita masukkan ke dalam proses PCA itu. Di *kodingan* ini, kolom yang di-*set* adalah **['Age', 'Duration']** (NUMERICAL_COLS_CLASS). Kolom *list* ini wajib di-*set* benar, karena di *step* selanjutnya, data *feature* (X) akan dipecah lagi, di mana yang *list* ini nanti dimasukkan ke PCA, sementara sisanya (kolom *encoded/binary*) akan dibiarkan utuh dan digabung lagi belakangan. *Step* terakhir, **print(f"Dimensi Awal Fitur...)**, cuma buat ngecek ulang di awal, apakah *data frame* X kita sudah benar dan siap diolah.

Tujuan Kode: Memuat data_train_preprocessed.csv dan memisahkan X_class_train dari y_class_train.

3. Seleksi Fitur Menggunakan LASSO

Logistic Regression dengan penalty L1 (LASSO) digunakan untuk memilih fitur terpenting. LASSO bekerja dengan mengecilkan koefisien fitur sehingga fitur dengan bobot kecil dihapus otomatis. Hasilnya memberikan subset fitur yang lebih relevan untuk modelling.

```
print("\n2. SKENARIO A: SELEKSI FITUR (LASSO L1)...")

# Latih model LASSO untuk seleksi fitur
l1_model_class = LogisticRegression(penalty='l1', C=0.05, solver='liblinear',
random_state=42)
l1_selector = SelectFromModel(l1_model_class)
l1_selector.fit(X_class_train, y_class_train)

# Ambil fitur yang lolos seleksi
selected_features_class = X_class_train.columns[l1_selector.get_support()]

print(f"LASSO memilih {len(selected_features_class)} fitur.")
```

```
# --- KODE VISUALISASI PLOT A.1: FEATURE IMPORTANCE LASSO ---
l1_coefs = l1_selector.estimator_.coef_[0]
l1_coefs_df = pd.DataFrame({
    'Feature': X_class_train.columns,
    'Importance (Abs Coef)': np.abs(l1_coefs)
}).sort_values(by='Importance (Abs Coef)', ascending=False)

plt.figure(figsize=(10, 8))
sns.barplot(
    x='Importance (Abs Coef)',
    y='Feature',
    data=l1_coefs_df.head(20) # Plot 20 fitur teratas
)
plt.title('Plot A.1: Feature Importance (LASSO L1) Tugas Klasifikasi')
plt.xlabel('Bobot Koefisien (Absolut)')
plt.ylabel('Fitur')
plt.tight_layout()
plt.show()

# Buat dataframe baru dan simpan hasil Skenario A
X_class_train_selected = X_class_train[selected_features_class]
X_class_test_selected = X_class_test[selected_features_class]
X_class_train_selected.to_csv('X_class_train_selected.csv', index=False)
X_class_test_selected.to_csv('X_class_test_selected.csv', index=False)
print("=> Disimpan: X_class_train_selected.csv")
```

a. Cara Kerja LASSO (The Filter)

Kita *setup* **LogisticRegression** tapi dengan *settingan* **penalty='l1'** dan **C=0.05**. Ini *signature* dari **LASSO**. Kenapa **C=0.05**? Karena C itu kebalikan dari kekuatan *penalty*. Semakin kecil C, semakin **brutal** *penalty*-nya. Ini *settingan* buat nyaring fitur secara ketat. LASSO akan ngeliat semua fitur, terus fitur yang kontribusinya kecil atau *redundant* **langsung dapet koefisien nol**. Fitur yang koefisiennya nggak nol, itu yang *worth it* dipertahankan.

b. Penyaringan dan Verifikasi

Model LASSO (l1_model_class) itu dibungkus pakai **SelectFromModel**. Setelah di-*fit*, l1_selector ini tahu persis fitur mana aja yang 'diselamatkan'. Kita ambil daftar nama fitur yang lolos seleksi (selected_features_class) dan langsung *print* jumlahnya. Kalau jumlahnya berkurang drastis dari fitur awal, berarti LASSO sukses kerja keras!

c. Justifikasi Visual

Kita nggak cuma bilang, "Ini *loh* fiturnya," tapi kita harus **buktikan kenapa fitur itu penting**. Kita bikin **Grafik Feature Importance** (Plot A.1) dari koefisien absolut LASSO. Grafik ini adalah **rating** resmi dari fitur-fitur kita, nunjukkin fitur mana yang bobotnya paling gede dan mana yang paling nggak penting (mendekati nol). Ini *mandatory* buat ditaruh di laporan, biar keputusan kita valid dan ada *proof*-nya.

4. Reduksi Dimensi Menggunakan PCA

PCA digunakan untuk mereduksi multikolinearitas pada fitur numerik. Dengan n_components=0.95, PCA akan mencari jumlah komponen minimal yang masih dapat menjelaskan 95% variasi data. Hasil PCA kemudian digabung kembali dengan fitur biner untuk membentuk dataset baru.

```
print("\n3. SKENARIO B: REDUKSI DIMENSI (PCA)...")
# Pisahkan data numerik dan biner
X_train_num = X_class_train[NUMERICAL_COLS_CLASS]
X_train_bin = X_class_train.drop(columns=NUMERICAL_COLS_CLASS,
errors='ignore')
X_test_num = X_class_test[NUMERICAL_COLS_CLASS]
X_test_bin = X_class_test.drop(columns=NUMERICAL_COLS_CLASS, errors='ignore')
# --- KODE VISUALISASI PLOT A.2: VARIAN KUMULATIF PCA (Analisis Komponen) ---
# Fit PCA penuh pada data numerik (TANPA reduksi) untuk menghitung varians
kumulatif
pca_full_class = PCA(random_state=42).fit(X_train_num)
cumulative_variance = np.cumsum(pca_full_class.explained_variance_ratio_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
marker='o', linestyle='--')
plt.axhline(y=0.95, color='red', linestyle='-', label='Batas 95% Varians')
plt.title('Plot A.2: Varians Kumulatif PCA - Tugas Klasifikasi')
plt.xlabel('Jumlah Komponen Utama')
plt.ylabel('Rasio Varians Kumulatif')
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.show()

# Inisialisasi PCA dengan n_components=0.95 untuk transformasi
pca_class = PCA(n_components=0.95, random_state=42)
pca_class.fit(X_train_num)
print(f"PCA memilih {pca_class.n_components_} komponen.")

# Transformasi dan Penggabungan
X_train_pca_transformed = pca_class.transform(X_train_num)
X_test_pca_transformed = pca_class.transform(X_test_num)

pc_cols = [f'PC{i+1}' for i in range(pca_class.n_components_)]
```

```
X_train_pca_df = pd.DataFrame(X_train_pca_transformed, columns=pc_cols,
index=X_class_train.index)
X_test_pca_df = pd.DataFrame(X_test_pca_transformed, columns=pc_cols,
index=X_class_test.index)

# Gabungkan data PCA dengan data biner (non-PCA)
X_class_train_pca = pd.concat([X_train_bin, X_train_pca_df], axis=1)
X_class_test_pca = pd.concat([X_test_bin, X_test_pca_df], axis=1)

# Simpan hasil Skenario B
X_class_train_pca.to_csv('X_class_train_pca.csv', index=False)
X_class_test_pca.to_csv('X_class_test_pca.csv', index=False)
print("=> Disimpan: X_class_train_pca.csv")
```

a. Pemecahan Data (Pisah Dulu, Jangan Dicampur!)

Step awalnya gampang: **pisahin data numerik dari data biner/encoded**. Ingat, PCA cuma boleh dijalankan di data numerik yang sudah di-*scaling*!

- **X_train_num:** Ini isinya cuma kolom-kolom numerik yang kita *define* di awal (Age, Duration). Ini yang akan masuk mesin PCA.
- **X_train_bin:** Ini isinya semua kolom biner/encoded (hasil *One-Hot*). Ini dianggurin dulu, karena PCA nggak relevan buat *data binary*.

b. Justifikasi Visual (Scree Plot)

Sebelum kita *transform* data, kita harus buktikan dulu **berapa banyak Komponen Utama (PC) yang optimal**.

- Kita *fit* model **pca_full_class** sementara ke data numerik. Terus, kita hitung **cumulative_variance**. Ini intinya adalah: *kalau kita pakai 1 PC, berapa persen info yang dapet? Kalau 2 PC, berapa persen?*
- Hasilnya kita *plot* jadi **Scree Plot** (Plot A.2). Grafik ini nunjukkin titik mana garisnya mulai mendatar. Kita pasang **plt.axhline(y=0.95, ...)** buat nandain **Batas 95% Varians**. Plot ini adalah **bukti visual** kita bahwa keputusan untuk mengambil sejumlah PC tertentu (misalnya, 2 atau 3 PC) itu valid karena sudah mencakup 95% informasi.

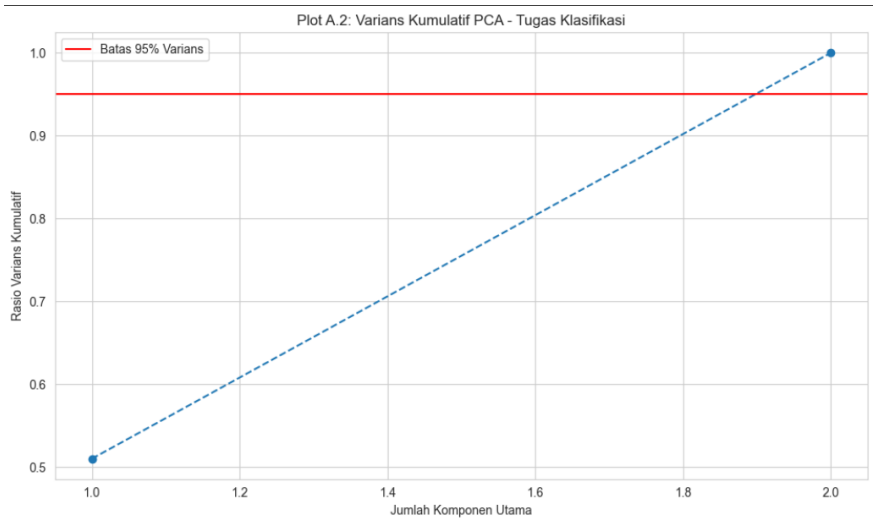
c. Transformasi Inti (PC Creation)

Ini adalah *step magic*-nya PCA:

- Kita inisialisasi **pca_class** dengan *settingan* **n_components=0.95**. Kenapa 0.95? Karena kita udah lihat di Scree Plot, 95% itu *sweet spot* retensi informasi. *Settingan* ini otomatis nyari jumlah PC paling sedikit yang bisa nge-*cover* 95% informasi.
- Model ini kemudian nge-*transform* data numerik kita. Data X_train_num yang awalnya 2 kolom (Age, Duration) misalnya, sekarang berubah jadi sejumlah kolom **PC1, PC2, ...** (Komponen Utama).
- Kita *print* hasilnya (**PCA memilih X komponen.**) buat konfirmasi bahwa reduksi berhasil dilakukan.

4. Output Final (The Join)

Step terakhir adalah *finishing*:



- Data PC yang baru kita buat (**X_train_pca_df**) di-*join* atau di-*concatenate* lagi sama data biner yang tadi kita pisahin (**X_train_bin**).
- Hasilnya adalah *data frame* final kita, **X_class_train_pca**. *Data frame* ini dimensinya sudah lebih kecil, kolom-kolomnya tidak saling berkorelasi, dan *noise*-nya sudah dibuang.
- *Dataset* final ini di-*save* ke **X_class_train_pca.csv**, siap buat jadi *input* **Eksperimen B** di tahap *Modelling*.

5. Penyimpanan Dataset

Seluruh dataset hasil transformasi disimpan secara terpisah agar dapat dipakai dalam tahap modelling. Penyimpanan X dan y secara terpisah merupakan praktik standar dalam machine learning.

```
y_class_train.to_csv('y_class_train.csv', index=False, header=True)
y_class_test.to_csv('y_class_test.csv', index=False, header=True)
print("=> Disimpan: y_class_train.csv dan y_class_test.csv")
print("\n--- KLASIFIKASI OPTIMIZATION SELESAI ---")
```

***Step* Terakhir: Nyimpen Data Target biar Aman**

- **Penyimpanan Target:** Dua baris kode ini, **y_class_train.to_csv(...)** dan **y_class_test.to_csv(...)**, intinya cuma buat *nyimpen* kolom target kita (y) ke *file* CSV yang terpisah. Ini penting banget! Kenapa? Karena di *Modelling* nanti, *features* (X) sama *target* (y) harus di-*load* sendiri-sendiri, jadi nggak boleh digabung. Kita pakai *settingan* **index=False** biar indeks bawaan *Python* nggak ikut ke-*save* (biar *file*-nya bersih) dan **header=True** biar nama kolom targetnya (misalnya, 'Job') tetap ada.
- **Konfirmasi Kelar:** Setelah *saving* selesai, kita cuma nge-*print* konfirmasi (**=> Disimpan: y_class_train.csv dan y_class_test.csv**) buat mastiin semuanya beres.
- **Final Statement:** Baris **--- KLASIFIKASI OPTIMIZATION SELESAI ---** adalah *final statement* kita. Ini menandakan bahwa *dataset* Klasifikasi kita sudah siap 100%—sudah bersih (dari Pra-pemrosesan I) dan sudah dioptimasi (dari Pra-pemrosesan II)—tinggal masuk Tahap *Modelling* buat diadu performanya!

6. Kesimpulan Dan Panduan Pengguna
   a. Kesimpulan

      Pada tahap *Pra-pemrosesan Data II (Feature Engineering)* ini, tujuan utamanya adalah bikin dataset kita jadi lebih "pintar" dan lebih siap tempur sebelum masuk ke proses modelling. Setelah semua kode dijalankan, beberapa hal penting yang bisa disimpulkan adalah:

- Lingkungan kerja berhasil disiapkan lengkap dengan semua library penting kayak pandas, numpy, matplotlib, seaborn, Logistic Regression (L1), PCA, sampai SMOTE buat ngatasi data yang nggak seimbang.
- Data berhasil dimuat dan dipisah jadi fitur (X) dan target (y), biar lebih rapi dan gampang diproses di tahap modelling nanti.
- LASSO berhasil nyaring fitur terbaik.
  Jadi dari sekian banyak fitur, LASSO milih mana yang paling berpengaruh buat nentuin target 'Job'. Fitur yang "nggak guna" langsung dieliminasi.
- PCA juga jalan dengan baik, terutama buat ngurangin dimensi fitur numerik tanpa ngurangin informasi penting. PCA otomatis ngambil minimal komponen yang bisa ngejelasin 95% variasi data.
- Semua dataset hasil olahan disimpan secara terpisah (X & y), biar lebih gampang waktu masuk tahap modelling.

Secara keseluruhan, tahap Feature Engineering ini bikin dataset jadi:

- lebih ringan,
- nggak redundan,
- bebas multikolinearitas,
- dan lebih siap buat diuji dengan berbagai algoritma machine learning.

Dengan kata lain, dataset sekarang udah dalam mode optimal dan bisa langsung dipake buat uji performa model.

b. Panduan Pengguna

- **Siapkan dulu lingkungan kerja**

Pastikan library yang dipakai sudah ke-install semua. Kalau belum, tinggal install:

pip install pandas numpy matplotlib seaborn scikit-learn imbalanced-learn

- **Jalankan bagian inisialisasi**

  Bagian ini penting banget karena:

  - ngatur tampilan grafik,
  - ngilangin warning yang ganggu,
  - dan nge-import semua library utama.

- **Pastikan file datanya sesuai**

  Program ini butuh dua file hasil preprocessing tahap sebelumnya:

  - data_train_preprocessed.csv
  - data_test_preprocessed.csv

  Taruh kedua file itu di folder yang sama dengan script Python kamu.

- **Tentukan target dan fitur numerik**

  Sebelum lanjut, pastikan nama kolom target dan fitur numerik benar:

  TARGET_CLASS = 'Job'

  NUMERICAL_COLS_CLASS = ['Age', 'Duration']

  Kalau datasetmu beda, tinggal sesuaikan.

- **Jalankan LASSO buat nyaring fitur terbaik**

  Bagian ini:

  - ngelatih model Logistic Regression L1,
  - milih fitur penting,
  - dan nyimpen dataset hasil seleksi.

Hasilnya dua file:

X_class_train_selected.csv

X_class_test_selected.csv

Dataset ini biasanya lebih kecil dan lebih efektif.

- **Jalankan PCA buat ngurangin dimensi**

  Ini buat ngurangin fitur numerik tanpa ngilangin informasi penting.

  Di bagian ini:

  - Data numerik dipisahin dulu.
  - PCA dicoba penuh buat lihat variasi kumulatif.
  - PCA jalan dengan n_components=0.95.
  - Hasil PCA digabung lagi sama fitur biner.

  Hasilnya juga dua file:

  X_class_train_pca.csv

  X_class_test_pca.csv

- **Simpan target X dan y**

  Bagian ini wajib banget karena modelling butuh X dan y terpisah:

  y_class_train.csv

  y_class_test.csv

- **Langsung siap buat modelling**

  Sekarang kamu punya **dua versi dataset optimal**:

  1. **Dataset hasil LASSO** → fokus ke fitur terpenting.
  2. **Dataset hasil PCA** → dimensi lebih kecil, bebas korelasi.

  Keduanya bisa langsung diadu pakai model-model seperti:

  - Logistic Regression
  - SVM
  - Random Forest

- KNN
- Gradient Boosting

Tinggal lihat mana yang paling cocok dan paling akurat.