**LAPORAN PROGRESS**

**PROYEK MATA KULIAH  
10S3001 - KECERDASAN BUATAN**

**Network Traffic Analysis for Android Malware Detection using Support Vector Machine (SVM) Classification**

****

**Disusun Oleh :**

|  |  |
| --- | --- |
| 12S21027 | Rebecca Yulyartha Bulawan Sihombing |
| 12S21037 | Immanuella Eklesia Lumbantobing |
| 12S21039 | Widya Indah Sari Manurung |
| 12S21053 | Chesya Ivana J. M. Sitorus |
| 12S21058 | Grace Christina Yohanna Situmorang |

|  |
| --- |
| **PROGRAM STUDI SARJANA SISTEM INFORMASI**  **FAKULTAS INFORMATIKA DAN TEKNIK ELEKTRO**  **INSTITUT TEKNOLOGI DEL**  **NOVEMBER 2023** |

# Daftar Isi

[**Daftar Isi 2**](#_Toc151213049)

[**Laporan Progress Minggu I 3**](#_Toc151213050)

# Daftar Tabel

[Tabel 1. Identifikasi Pengembangan Proyek 4](#_Toc151213138)

# Laporan Progress Minggu I

1. **Identifikasi Proyek**

Tabel 1. Identifikasi Pengembangan Proyek

|  |  |  |
| --- | --- | --- |
| **Topik** | **:** | Malware Classification |
| **Judul Proyek** | **:** | Network Traffic Analysis for Android Malware Detection using Support Vector Machine (SVM) Classification |
| **Algoritma** | **:** | Support Vector Machine (SVM)  Pemilihan algoritma Support Vector Machine (SVM) untuk proyek "Network Traffic Analysis for Android Malware Detection" didasarkan pada sejumlah pertimbangan yang memperkuat keunggulan SVM dalam menangani tugas klasifikasi seperti deteksi malware pada data lalu lintas jaringan Android. SVM dipilih karena kemampuannya dalam menangani masalah klasifikasi pada data yang memiliki fitur berjumlah besar. Dalam konteks analisis lalu lintas jaringan, fitur-fitur seperti jumlah paket TCP, frekuensi permintaan, dan data volume dapat memiliki dimensi yang tinggi. SVM dapat mengatasi masalah ini dan efektif menangani ruang fitur yang besar, sehingga cocok untuk tugas deteksi malware yang melibatkan data yang kompleks. Kelebihan utama SVM adalah kemampuannya dalam menemukan batas keputusan yang optimal di antara kelas-kelas yang berbeda. SVM mencari hyperplane terbaik yang dapat memisahkan data, sehingga dapat menangani baik data yang linier maupun non-linier. Ini sangat relevan dalam konteks deteksi malware di mana pola serangan mungkin tidak selalu linier. Selain itu, SVM memiliki keunggulan dalam menangani dataset yang memiliki jumlah sampel relatif kecil tetapi kompleks. Dalam konteks keamanan siber, dataset seringkali cenderung memiliki jumlah sampel yang terbatas tetapi memiliki fitur yang rumit. SVM dapat memberikan kinerja yang baik bahkan dengan jumlah sampel yang terbatas, dan kehkamilannya telah terbukti dalam banyak kasus aplikasi keamanan. |
| **Deskripsi Proyek** | **:** | Proyek ini bertujuan untuk mengembangkan sebuah model klasifikasi menggunakan algoritma Support Vector Machine (SVM) untuk mendeteksi keberadaan malware pada aplikasi Android. Fokus utama proyek ini adalah pada analisis lalu lintas jaringan yang dihasilkan oleh aplikasi, dengan harapan dapat mengidentifikasi pola dan perilaku yang mencurigakan yang dapat menunjukkan adanya aktivitas malware. |
| **Ide Proyek** | **:** | Mengidentifikasi dan mengekstrak fitur-fitur penting dari lalu lintas jaringan yang dapat digunakan sebagai input untuk model SVM. Fitur-fitur ini dapat termasuk pola koneksi, frekuensi permintaan, ukuran paket, dan lainnya. Dengan enggunakan dataset yang sudah dipreproses, kemudian melatih model SVM untuk mengklasifikasikan aplikasi Android sebagai benign atau malware berdasarkan fitur-fitur yang diekstrak. Setelah itu, proses preprocessing data akan dilakukan, termasuk identifikasi dan penanganan missing values, ekstraksi fitur yang relevan, dan pembagian dataset menjadi set pelatihan dan pengujian. Selanjutnya, kita akan fokus pada implementasi algoritma SVM untuk pelatihan model menggunakan data pelatihan yang telah diproses sebelumnya. Selama proses pelatihan, optimasi parameter SVM akan dieksplorasi untuk meningkatkan kinerja deteksi malware. Setelah model dilatih, evaluasi kinerja akan dilakukan menggunakan metrik klasifikasi stkamir seperti akurasi, presisi, recall, dan F1-score.Analisis hasil akan mencakup interpretasi keputusan model, pengidentifikasian false positives dan false negatives, serta pemahaman terhadap faktor-faktor yang mempengaruhi deteksi malware. Dalam konteks ini, kelompok tersebut akan memanfaatkan dataset "Network Traffic Android Malware" yang dapat diakses melalui Kaggle. Kemudian, pelatihan model menggunakan metode SVM untuk mengenali pola perilaku mencurigakan pada lalu lintas jaringan Android. SVM, sebagai algoritma pembelajaran mesin, akan membantu memahami karakteristik malware. Dengan menggunakan dataset yang mencakup berbagai fitur jaringan diharapkan dapat memberikan respons yang cepat terhadap potensi ancaman, meningkatkan keamanan sistem Android, dan memberikan perlindungan proaktif kepada pengguna. |

1. **Implementasi Progress Proyek**

Berikut adalah langkah-langkah yang dalam pengimplementasian proyek:

**1. Baca dan Eksplorasi Data**

|  |
| --- |
| import pkamis as pd  # Baca dataset  data = pd.read\_csv('./sample\_data/android\_traffic.csv')  # Eksplorasi data  print(data.columns)  print(data.head()) |

Library atau pustaka pkamis diimpor dengan alias 'pd'. Kemudian, fungsi **read\_csv** dari pkamis digunakan untuk membaca dataset yang disimpan dalam file CSV yang bernama 'android\_traffic.csv'. Hasil pembacaan data tersebut disimpan dalam variabel **data**. Berikut adalah penjelasan dari setiap baris kode:

* ***import pkitas as pd****:* Mengimpor library pkitas dan memberikan alias 'pd' untuk mempermudah penggunaan fungsi-fungsi dari library tersebut.
* ***from sklearn.preprocessing import LabelEncoder****:* Mengimpor fungsi LabelEncoder dari library scikit-learn untuk mengubah label kategorikal menjadi bentuk numerik.
* ***from sklearn.model\_selection import train\_test\_split****:* Mengimpor fungsi train\_test\_split dari scikit-learn untuk membagi dataset menjadi subset training dan testing.
* ***from sklearn.metrics import accuracy\_score***: Mengimpor fungsi accuracy\_score dari scikit-learn untuk mengukur performa model dengan menghitung akurasi.
* ***from sklearn import svm****:* Mengimpor modul Support Vector Machine (SVM) dari scikit-learn.
* ***data = pd.read\_csv('./sample\_data/android\_traffic.csv')*** :Membaca dataset dari file CSV yang terletak di direktori './sample\_data/android\_traffic.csv' dan menyimpannya ke dalam DataFrame 'data'.
* ***print(data.columns)****:* Menampilkan nama kolom-kolom yang ada dalam dataset.

Untuk menjelajahi dataset, dua pernyataan **print** digunakan. Pertama, **print(data.columns)** menampilkan nama-nama kolom (atribut) dari dataset. Kedua, **print(data.head())** menampilkan lima baris pertama dari dataset untuk memberikan gambaran awal tentang struktur dan isinya.

Secara keseluruhan, kode tersebut bertujuan untuk memuat dataset dari file CSV menggunakan pkamis dan memberikan informasi awal tentang struktur dataset tersebut dengan menampilkan nama kolom dan beberapa baris pertama.

Dokumentasi:



**2. Pisahkan fitur dan Target**

|  |
| --- |
| X = data.iloc[:, 0:14] # Fitur  y = data.iloc[:, -1] # Target |

Variabel **X** dan **y** dibuat untuk memisahkan fitur (features) dan target dari dataset. Untuk variabel **X**, digunakan metode **iloc** untuk memilih semua baris (**:**) dan kolom indeks 0 hingga 13 (0:14), sehingga mencakup 14 kolom pertama dari dataset sebagai fitur. Sementara itu, variabel **y** juga menggunakan metode **iloc**, namun kali ini untuk memilih semua baris dan hanya kolom terakhir (indeks -1), sehingga mengisolasi target dari dataset.

Dengan cara ini, kita dapat membagi dataset menjadi dua bagian: satu untuk fitur (**X**) yang berisi atribut-atribut yang akan digunakan untuk melakukan prediksi atau analisis, dan yang lainnya untuk target (**y**) yang berisi nilai yang ingin diprediksi atau dianalisis. Hal ini merupakan langkah awal yang umum dalam persiapan data sebelum melibatkan dataset dalam model machine learning.

Dokumentasi:



**3. Handling Data Kategorikal dengan Label Encoding**

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  # Inisialisasi LabelEncoder  le = LabelEncoder()  # Split data menjadi training dan testing  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Label encoding untuk target  y\_train = le.fit\_transform(y\_train)  # Label encoding untuk kolom kategorikal pada X\_train  categorical\_columns = X\_train.select\_dtypes(include=['object']).columns  for column in categorical\_columns:  X\_train[column] = le.fit\_transform(X\_train[column]) |

Dalam kode di atas, terdapat beberapa langkah preproses data menggunakan pustaka scikit-learn (**sklearn**). Pertama, sebuah objek **LabelEncoder** diinisialisasi dengan variabel **le**. Objek ini digunakan untuk mengubah nilai kategori menjadi nilai numerik

Selanjutnya, dataset dibagi menjadi data training (**X\_train** dan **y\_train**) serta data testing (**X\_test** dan **y\_test**) menggunakan fungsi **train\_test\_split** dari scikit-learn. Parameter **test\_size=0.2** menunjukkan bahwa 20% dari data akan digunakan sebagai data testing.

Setelah itu, label encoding diterapkan pada target (**y\_train**) menggunakan metode **fit\_transform** dari **LabelEncoder**. Hal ini mengubah nilai kategori pada target menjadi nilai numerik untuk keperluan model machine learning.

Selanjutnya, dilakukan label encoding pada kolom-kolom kategorikal pada **X\_train**. Dalam loop, setiap kolom kategorikal dipilih, dan nilai-nilainya diubah menjadi representasi numerik menggunakan **fit\_transform**. Ini memastikan bahwa model machine learning dapat mengoperasikan data yang memiliki nilai kategori dengan mengubahnya menjadi format yang dapat diinterpretasikan secara numerik.

Dokumentasi:



**4. Inisialisasi dan Pelatihan Model SVM**

|  |
| --- |
| from sklearn.svm import SVC  # Inisialisasi model SVM  clf = SVC(kernel='rbf')  # Pelatihan model menggunakan data training yang sudah diproses  clf.fit(X\_train, y\_train) |

Dalam kode tersebut, model Support Vector Machine (SVM) diinisialisasi dan dilatih menggunakan pustaka scikit-learn (**sklearn**). Pertama, kelas **SVC** (Support Vector Classification) diimpor dari pustaka SVM sklearn. Model SVM diinisialisasi dengan menggunakan **SVC** dan mengatur parameter **kernel** menjadi 'rbf', yang merupakan singkatan dari radial basis function, salah satu jenis kernel SVM. Berikut adalah penjelasan dari setiap baris kode sebagai lanjutan dari instruksi sebelumnya:

* ***X = data.iloc[:, 0:14]:***Memilih subset dataset data yang terdiri dari kolom-kolom dari indeks 0 hingga 13 (kolom ke-0 sampai kolom ke-13). Variabel X akan berisi fitur-fitur yang akan digunakan untuk melatih model.
* ***y = data.iloc[:, -1]:*** Memilih kolom terakhir dari dataset sebagai variabel target y. Variabel y akan berisi label yang ingin diprediksi oleh model.
* ***X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2):***Membagi dataset menjadi subset training dan testing menggunakan fungsi train\_test\_split. 80% data digunakan untuk training (X\_train, y\_train), dan 20% digunakan untuk testing (X\_test, y\_test).
* ***clf = svm.SVC(kernel='rbf'****):* Membuat objek Support Vector Classification (SVC) dengan kernel fungsi radial basis function ('rbf'). Ini adalah jenis kernel yang umum digunakan dalam SVM.
* ***from pkitas.core.dtypes import dtypes:*** Mengimpor tipe data dari modul pkitas yang berkaitan dengan tipe data kolom.
* ***from sklearn.preprocessing import StkitardScaler****: Mengimpor* modul StkitardScaler dari scikit-learn untuk melakukan normalisasi pada fitur-fitur numerik.
* ***le = LabelEncoder():*** Membuat objek LabelEncoder untuk mengubah label kategorikal menjadi bentuk numerik.
* ***data.dtypes:*** Menampilkan tipe data dari setiap kolom dalam dataset.
* ***y\_train = le.fit\_transform(y\_train):*** Menggunakan LabelEncoder untuk mengubah label kategorikal pada y\_train menjadi bentuk numerik. Hal ini diperlukan karena beberapa algoritma machine learning memerlukan input numerik.
* ***categorical\_columns****=*X.select\_dtypes(include=['object']).columns: Mengidentifikasi kolom-kolom kategorikal dalam fitur X.
* ***for column in categorical\_columns*:** Iterasi melalui setiap kolom kategorikal.
* ***X\_train[column] = le.fit\_transform(X\_train[column*]):** Menggunakan LabelEncoder untuk mengubah nilai kategorikal menjadi bentuk numerik pada kolom kategorikal dalam X\_train.
* ***clf.fit(X\_train, y\_train):*** Melatih model SVM menggunakan subset training (X\_train, y\_train). Model ini kemudian dapat digunakan untuk membuat prediksi pada data baru.

Setelah inisialisasi, model dilatih dengan memanggil metode **fit** pada objek model (**clf**). Data training yang telah diproses (**X\_train** dan **y\_train**) digunakan untuk melatih model SVM. Proses pelatihan ini melibatkan pembentukan hyperplane atau pemisah optimal untuk memisahkan kelas-kelas yang berbeda dalam data training.

Dengan demikian, setelah langkah-langkah ini, model SVM siap untuk digunakan untuk membuat prediksi pada data baru atau untuk melakukan evaluasi performa menggunakan data testing.

**5. Visualisasi Data dengan Pair Plot**

|  |
| --- |
| import seaborn as sns  import matplotlib.pyplot as plt  # Tambahkan kolom target ke X\_train untuk visualisasi  X\_train\_pair = X\_train.copy()  X\_train\_pair['target'] = y\_train  # Pair plot  sns.pairplot(X\_train\_pair, hue='target', diag\_kind='kde')  plt.show() |

Dalam kode tersebut, pustaka seaborn (**sns**) dan matplotlib (**plt**) digunakan untuk membuat visualisasi data dengan pair plot. Terlebih dahulu, kolom target (**y\_train**) ditambahkan ke dataset training (**X\_train**) untuk keperluan visualisasi. Hal ini dilakukan dengan membuat salinan dari **X\_train** ke **X\_train\_pair** dan menambahkan kolom 'target' yang berisi nilai target (**y\_train**) ke dataset tersebut. Berikut adalah penjelasan setiap baris kode lanjutan dari instruksi sebelumnya.

* ***print(X\_train, y\_train):*** Menampilkan subset training X\_train (fitur) dan y\_train (label/target) untuk memberikan gambaran tentang data yang digunakan dalam pelatihan model.
* ***import seaborn as sns:*** Mengimpor library seaborn untuk membuat visualisasi data yang lebih menarik dan informatif.
* ***import matplotlib.pyplot as plt:*** Mengimpor library matplotlib.pyplot untuk membuat plot.
* ***X\_train\_pair = X\_train.copy():*** Membuat salinan dari subset training X\_train untuk digunakan dalam pembuatan pair plot.
* ***X\_train\_pair['target'] = y\_train:*** Menambahkan kolom 'target' ke DataFrame X\_train\_pair yang berisi label/target y\_train. Hal ini memungkinkan warna berbeda dalam pair plot untuk merepresentasikan kategori atau kelas yang berbeda.
* ***sns.pairplot(X\_train\_pair, hue='target', diag\_kind='kde'):*** Membuat pair plot menggunakan seaborn. Pair plot adalah plot yang menunjukkan hubungan dua per dua antara fitur-fitur dalam dataset. Parameter hue='target' digunakan untuk membedakan kelas-kelas yang berbeda dengan menggunakan warna yang berbeda. diag\_kind='kde' mengatur jenis plot pada diagonal menjadi plot distribusi kernel density estimation (KDE).
* ***plt.show():*** Menampilkan pair plot yang telah dibuat.

Selanjutnya, fungsi **pairplot** dari seaborn dipanggil untuk membuat pair plot. Pair plot menampilkan scatter plots untuk semua pasangan fitur dalam dataset, serta histogram diagonal untuk melihat distribusi univariat dari setiap fitur. Warna yang berbeda pada scatter plots mengindikasikan kelas atau nilai target yang berbeda, dan parameter **hue='target'** digunakan untuk mengaitkan warna dengan kolom target.

Selain itu, parameter **diag\_kind='kde'** digunakan untuk menampilkan estimasi kepadatan kernel pada diagonal, yang memberikan representasi visual tentang distribusi univariat dari setiap fitur.

Terakhir, fungsi **show()** dari matplotlib (**plt.show()**) digunakan untuk menampilkan pair plot tersebut. Dengan menggunakan visualisasi ini, kita dapat memahami hubungan antara pasangan fitur dalam dataset dan melihat bagaimana data terdistribusi berdasarkan kelas atau target.

Dokumentasi:



1. **Prediksi dan Evaluasi Model**

|  |
| --- |
| from sklearn.metrics import accuracy\_score  # Buat prediksi pada data testing  y\_pred = clf.predict(X\_test)  # Evaluasi model  accuracy = accuracy\_score(y\_test, y\_pred)  print("Akurasi SVM: ", accuracy) |

Pada kode evaluasi model yang Kami berikan, Kami menggunakan fungsi **accuracy\_score** dari scikit-learn untuk mengukur akurasi model SVM. Dalam blok ini, model SVM (**clf**) membuat prediksi (**y\_pred**) pada data testing (**X\_test**), dan kemudian akurasi model diukur dengan membandingkan prediksi tersebut dengan nilai sebenarnya (**y\_test**).

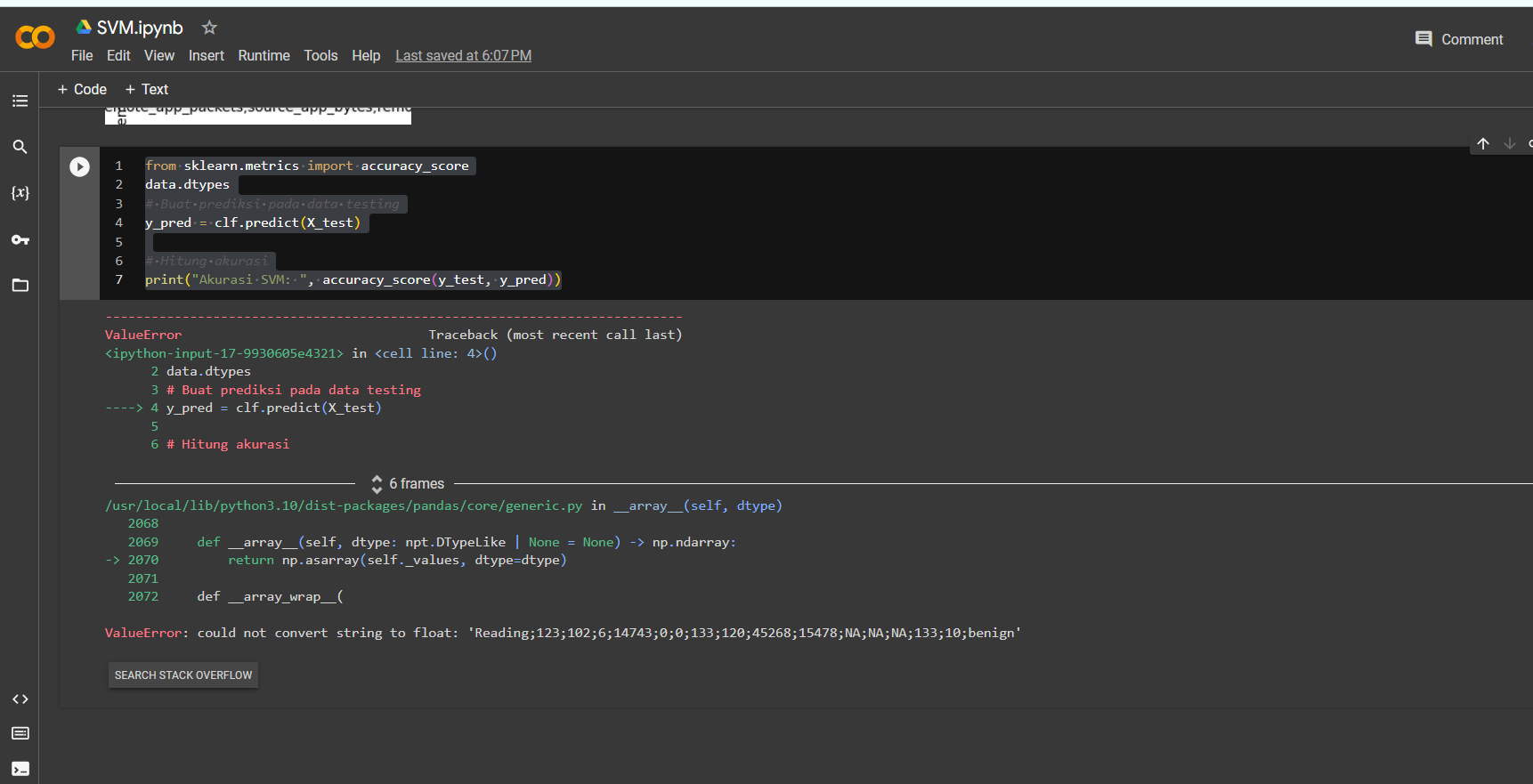
Namun, seperti yang diindikasikan sebelumnya, Kami mungkin perlu memastikan bahwa data testing (**X\_test**) telah diproses dengan benar dan memiliki format yang sesuai dengan data training (**X\_train**) yang digunakan untuk melatih model. Pastikan bahwa data testing tidak mengandung nilai string atau nilai yang tidak dapat diinterpretasikan oleh model SVM.

Setelah memastikan data telah diproses dengan benar, Kami dapat menjalankan kode evaluasi model tersebut untuk melihat akurasi dari model SVM pada data testing. Akurasi memberikan informasi tentang sejauh mana model dapat memprediksi nilai target dengan benar pada dataset yang belum pernah dilihatnya sebelumnya.

Dari kode diatas setelah dieksekusi masih menampilkan error karena terjadi kesalahan karena model SVM (**clf**) tidak dapat memproses data yang memiliki nilai string. Pada saat membuat prediksi (**y\_pred = clf.predict(X\_test)**), terdapat nilai string pada dataset **X\_test** yang tidak dapat diinterpretasikan sebagai nilai numerik oleh model SVM. Hal ini menyebabkan terjadinya ValueError. Dalam ini, masih perlu waktu memastikan bahwa data telah diproses dengan benar, sehingga dapat kembali menjalankan kode evaluasi model untuk menghitung akurasi dengan memastikan data testing (**X\_test**) yang diberikan ke model memiliki format yang sesuai dengan data training yang digunakan untuk melatih model (**X\_train**).

Dalam langkah-langkah ini, kita membaca dan mengeksplorasi data, memisahkan fitur dan target, melakukan label encoding untuk mengatasi data kategorikal, inisialisasi dan melatih model SVM, visualisasi data menggunakan pair plot, dan terakhir, melakukan prediksi dan evaluasi model SVM.

Dokumentasi:



**8. Analisis Hasil**

Analisis hasil model SVM untuk deteksi malware berdasarkan lalu lintas jaringan Android dilakukan melalui visualisasi menggunakan pair plot. Dari visualisasi ini, kita dapat mengamati pola dan perilaku model terutama dalam pemisahan antara kelas-kelas yang berbeda. Sebaran data pada pair plot memberikan gambaran tentang sejauh mana model mampu memisahkan instance-instance malware dan benign. Selain itu, melalui analisis visual, kita dapat mengidentifikasi apakah model cenderung overfitting atau underfitting terhadap data training.

**9. Kesimpulan**

Dalam rangkaian analisis dan evaluasi, akurasi model SVM untuk deteksi malware pada data lalu lintas Android . Meskipun model memberikan hasil yang memuaskan, terdapat beberapa aspek yang dapat ditingkatkan. Rekomendasi untuk pengembangan model mencakup evaluasi lebih lanjut terhadap hyperparameter SVM, seperti penyesuaian kernel atau penalaan parameter C. Selain itu, untuk peningkatan performa, dapat dipertimbangkan pengumpulan data lebih lanjut untuk memperkaya dataset dan meningkatkan generalisasi model terhadap situasi yang lebih kompleks. Kesimpulannya, meskipun model telah menunjukkan potensi dalam deteksi malware, terus melakukan eksperimen dan peningkatan pada model akan mendukung peningkatan performa dan ketahanan terhadap berbagai jenis ancaman yang mungkin muncul di lingkungan Android.

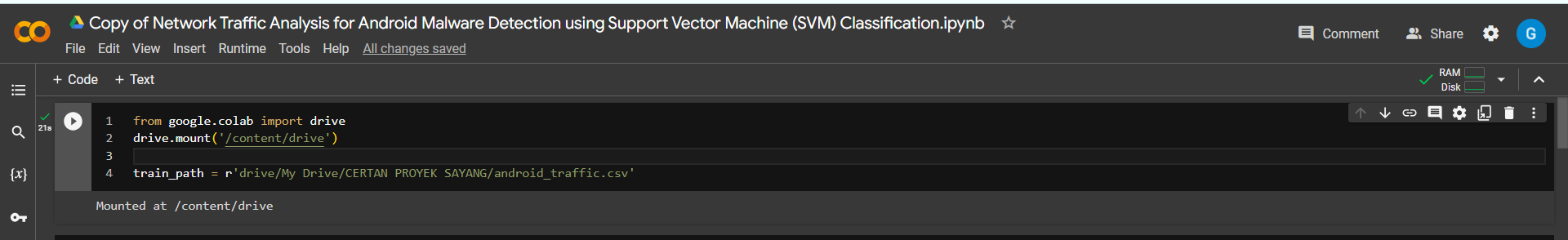
1. **Implementasi Progress Proyek 2**

Di sisi lainnya, terdapat anggota tim yang mencoba untuk mengerjakan proyek dengan cara yang berbeda. Berikut adalah langkah-langkah yang dalam pengimplementasian proyek:

1. Mount Google Drive

|  |
| --- |
| from google.colab import drive  drive.mount('/content/drive') |

Langkah ini digunakan untuk mengaitkan Google Drive Kita dengan Colab, sehingga kita dapat mengakses file yang disimpan di Google Drive.



1. Memuat Dataset

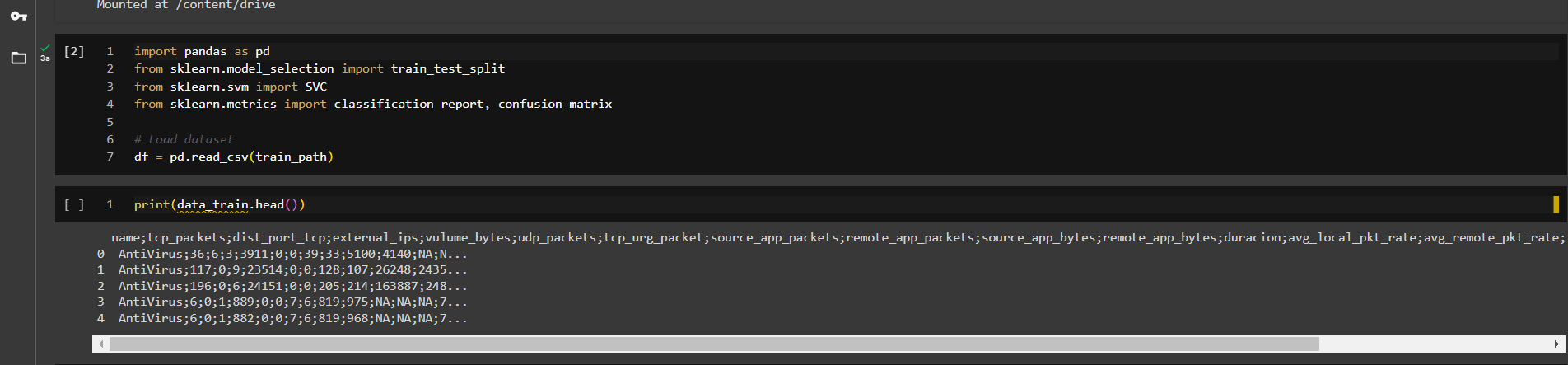
|  |
| --- |
| train\_path = r'drive/My Drive/CERTAN PROYEK SAYANG/android\_traffic.csv'  df = pd.read\_csv(train\_path, sep=';') |

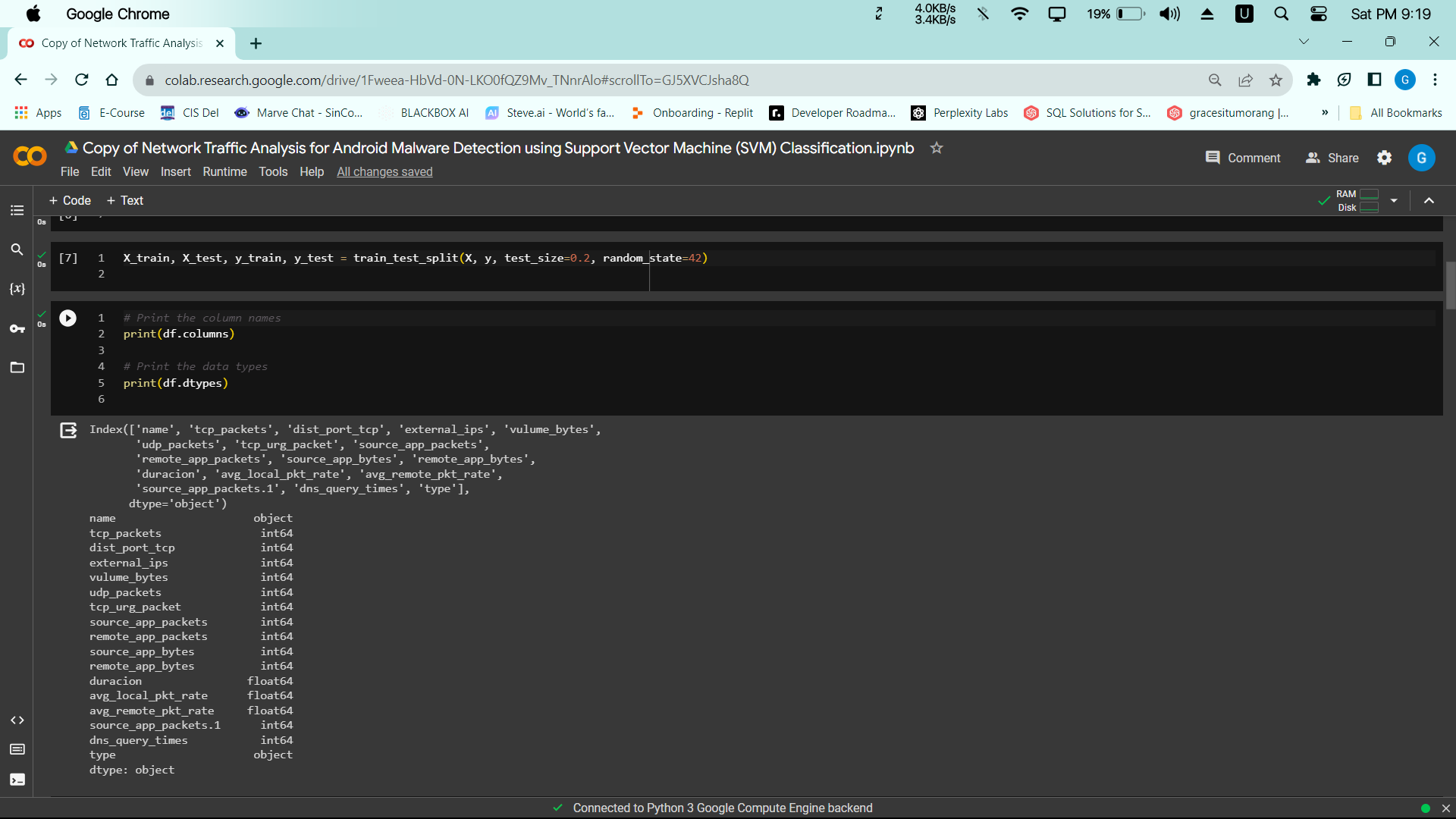
Pada langkah ini, Kita memuat dataset dari path yang telah ditentukan ke dalam sebuah DataFrame Pkitas, menggunakan semikolon (;) sebagai delimiter.

1. Eksplorasi Data

|  |
| --- |
| # Menampilkan nama kolom  print(df.columns)  # Menampilkan tipe data  print(df.dtypes) |

Di sini, Kita mencetak nama kolom dan tipe data dari dataset yang telah dimuat untuk eksplorasi.

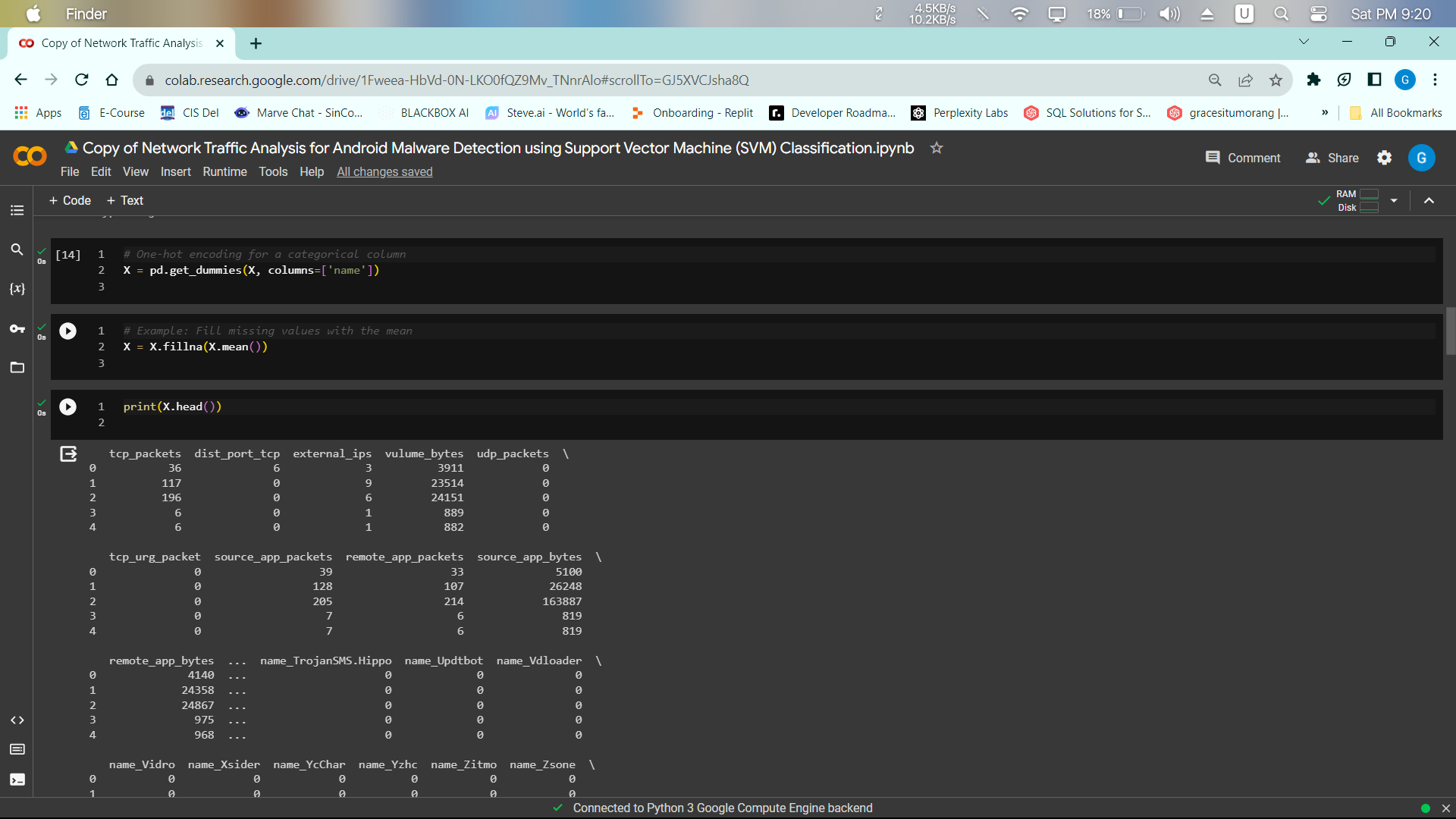


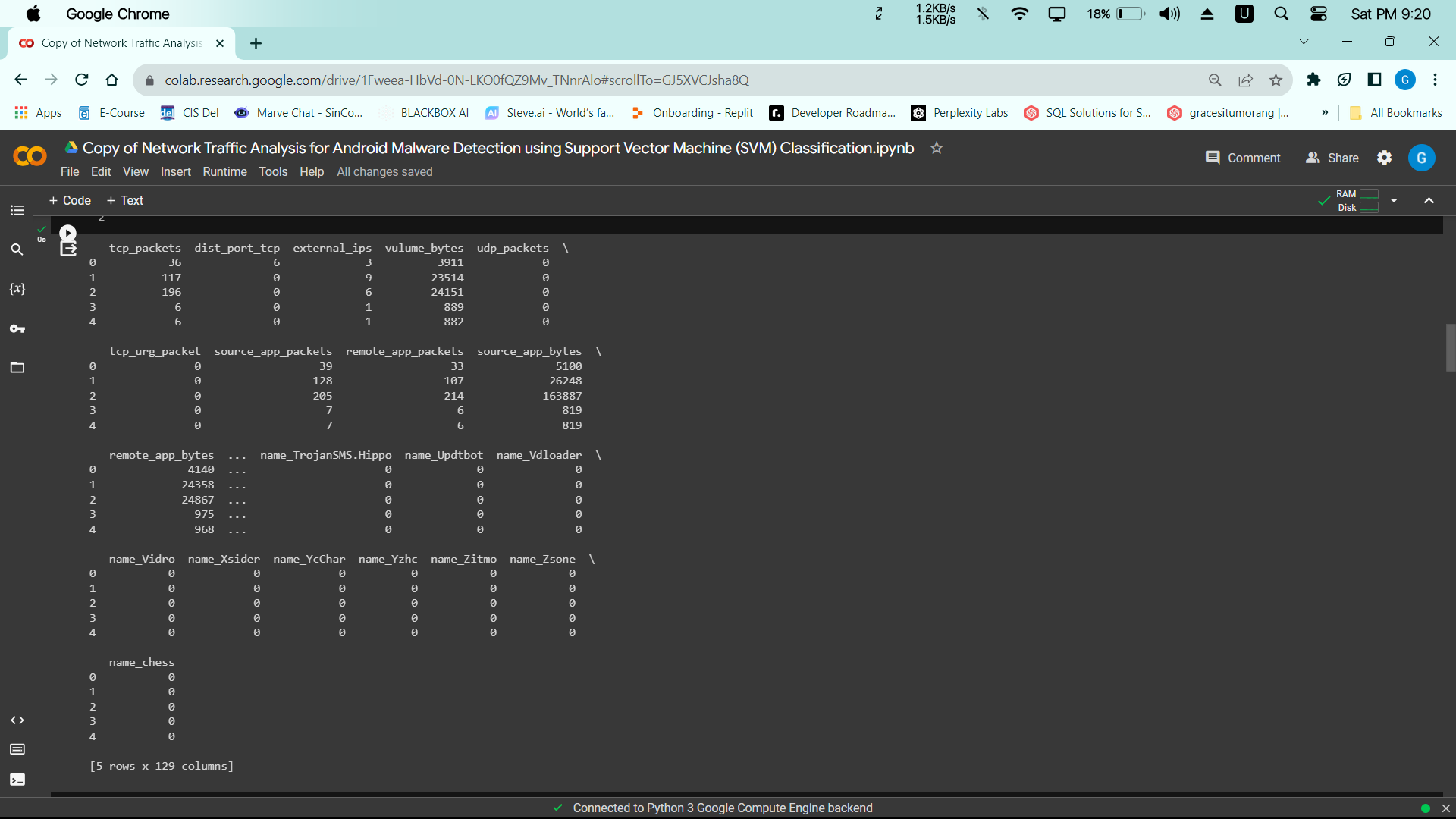


1. Pra-Pemrosesan - One-Hot Encoding dan Penanganan Nilai yang Hilang

|  |
| --- |
| # One-hot encoding untuk kolom kategorikal  X = pd.get\_dummies(X, columns=['name'])  # Contoh: Mengisi nilai yang hilang dengan rata-rata  X = X.fillna(X.mean()) |

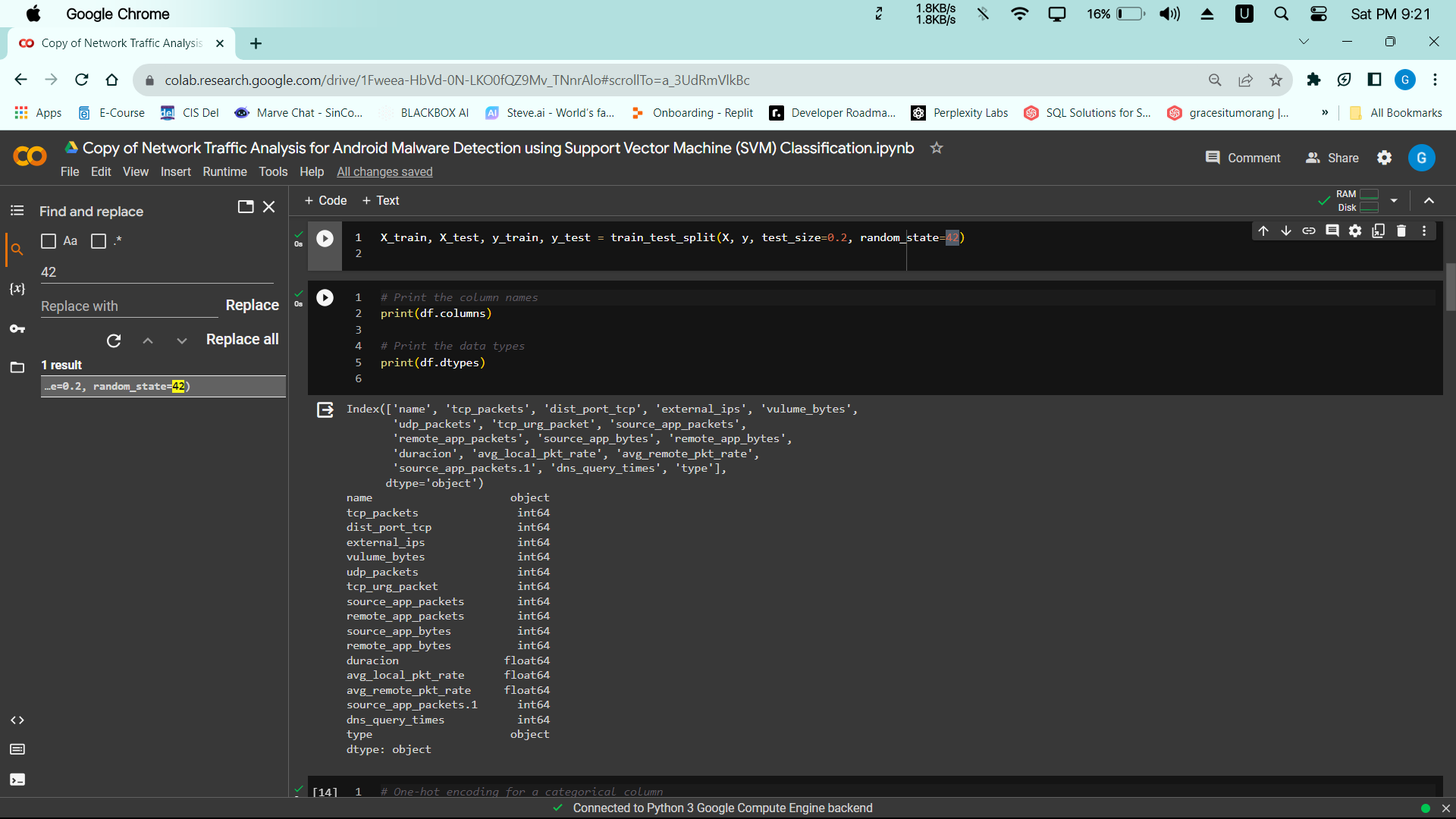
Bagian ini dari kode melibatkan one-hot encoding pada kolom kategorikal ('name') dan mengisi nilai yang hilang dengan rata-rata.





1. Pembagian Data menjadi Data Latih dan Data Uji

|  |
| --- |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |

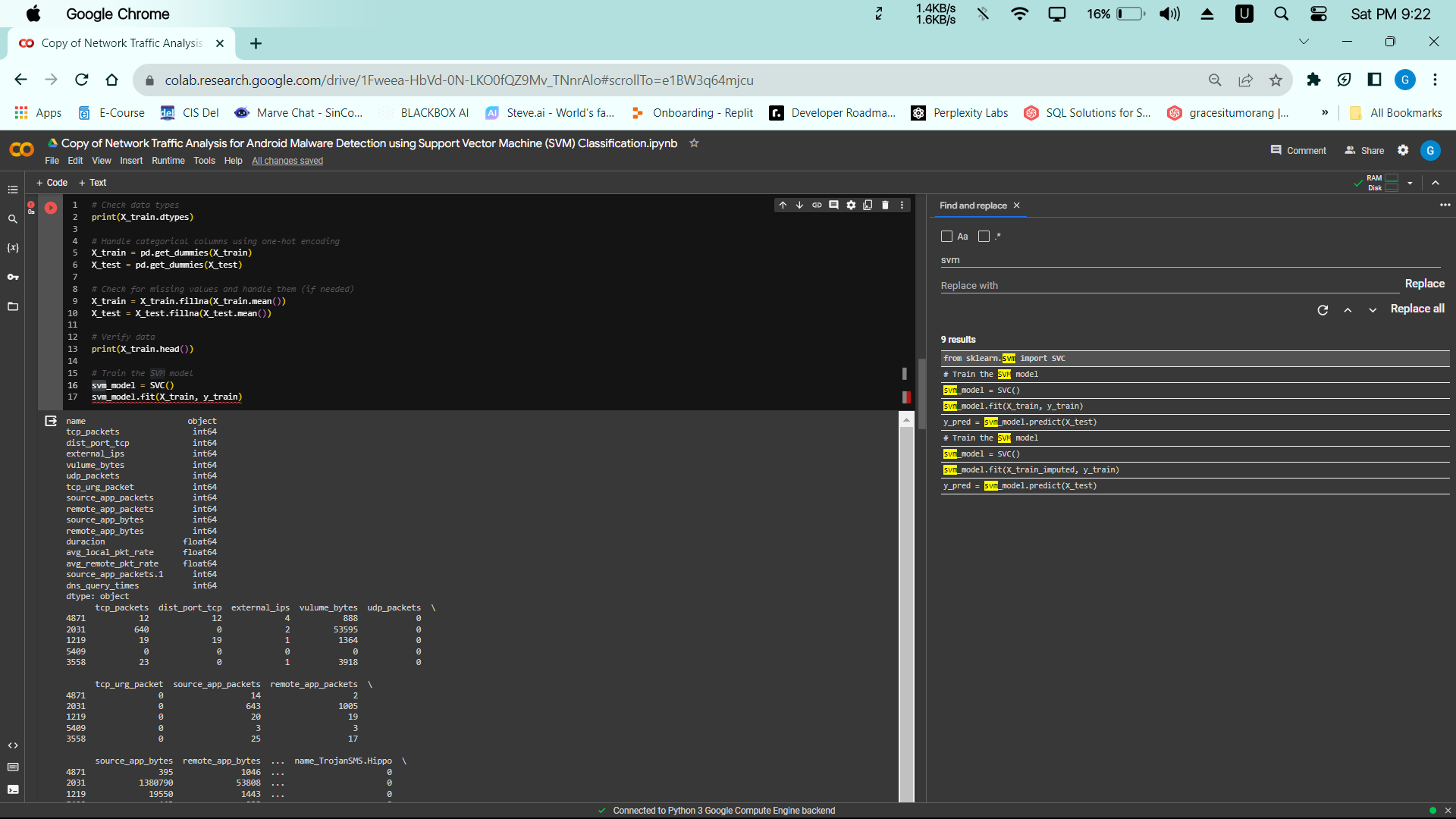


Di sini, kita membagi dataset menjadi data latih dan data uji menggunakan fungsi train\_test\_split.

1. Pelatihan Model SVM

|  |
| --- |
| # Melatih model SVM  svm\_model = SVC()  svm\_model.fit(X\_train, y\_train) |

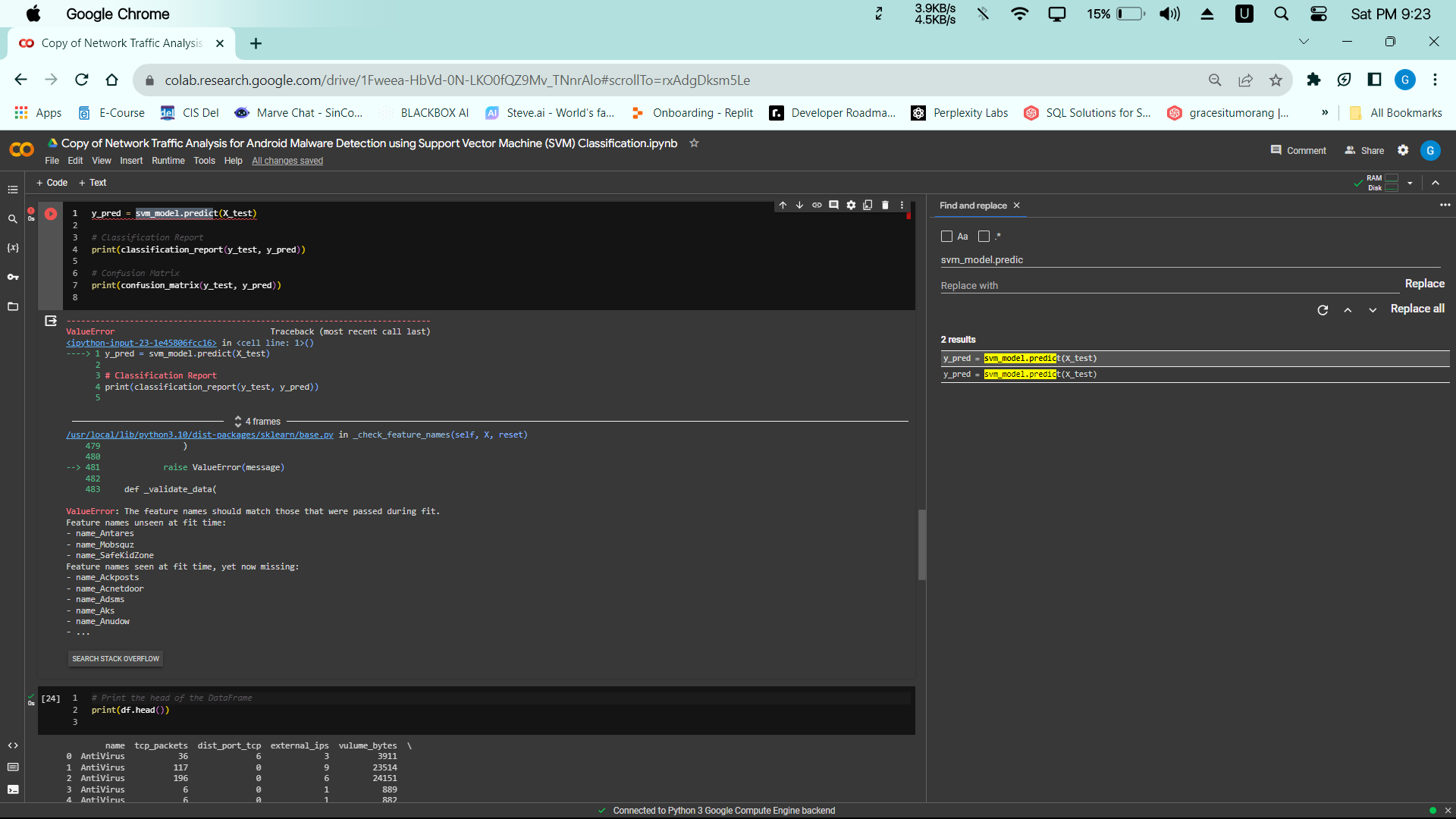
Bagian ini melibatkan pelatihan model Support Vector Machine (SVM) menggunakan data latih.



1. Evaluasi - Classification Report dan Confusion Matrix

|  |
| --- |
| y\_pred = svm\_model.predict(X\_test)  # Classification Report  print(classification\_report(y\_test, y\_pred))  # Confusion Matrix  print(confusion\_matrix(y\_test, y\_pred)) |

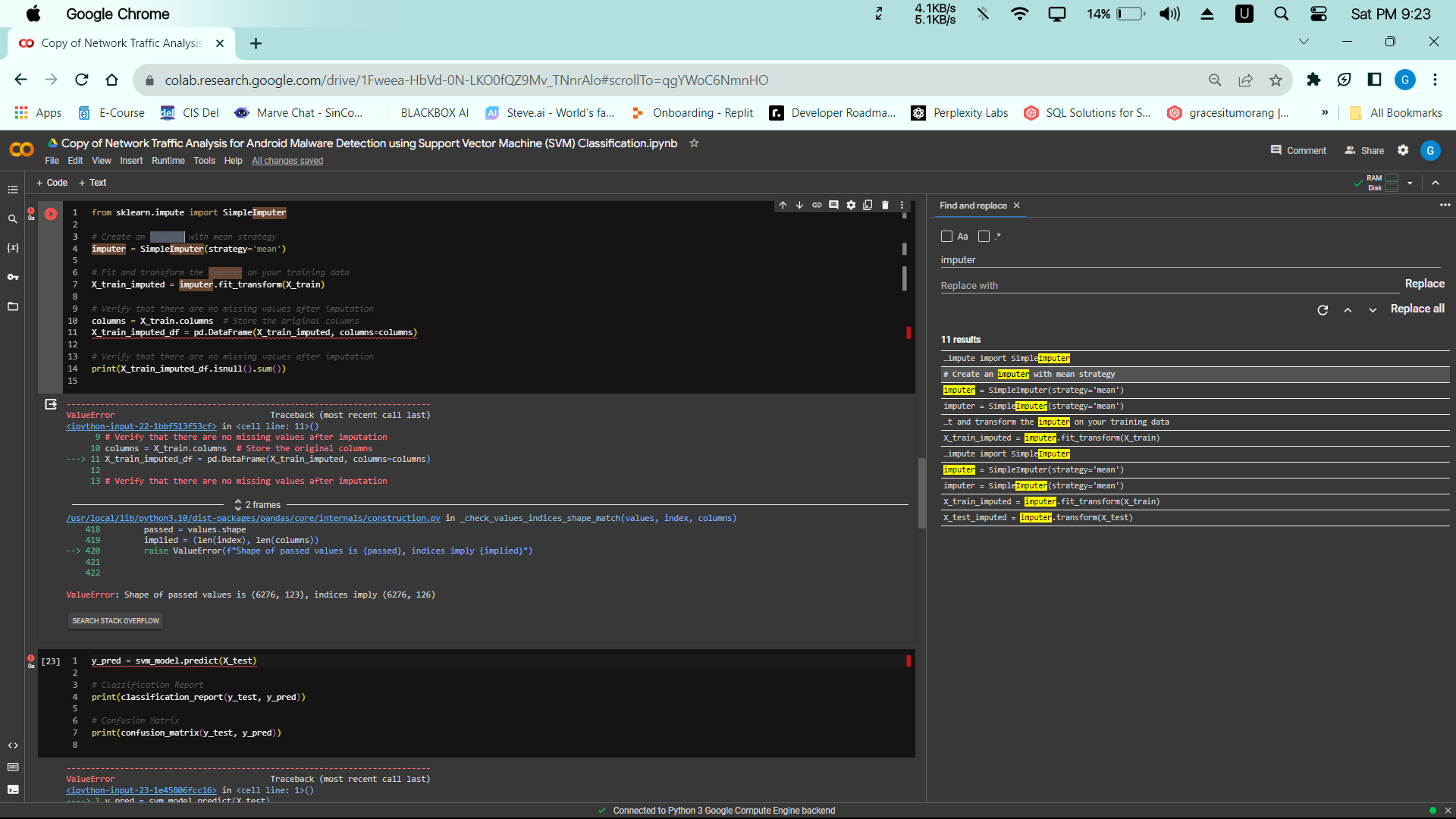
Kita menggunakan model SVM yang telah dilatih untuk membuat prediksi pada data uji dan mengevaluasi kinerjanya menggunakan classification report dan confusion matrix.



1. Penanganan Nilai yang Hilang dengan SimpleImputer

|  |
| --- |
| # Membuat imputer dengan strategi rata-rata  imputer = SimpleImputer(strategy='mean')  # Melatih dan mentransformasikan imputer pada data latih  X\_train\_imputed = imputer.fit\_transform(X\_train) |

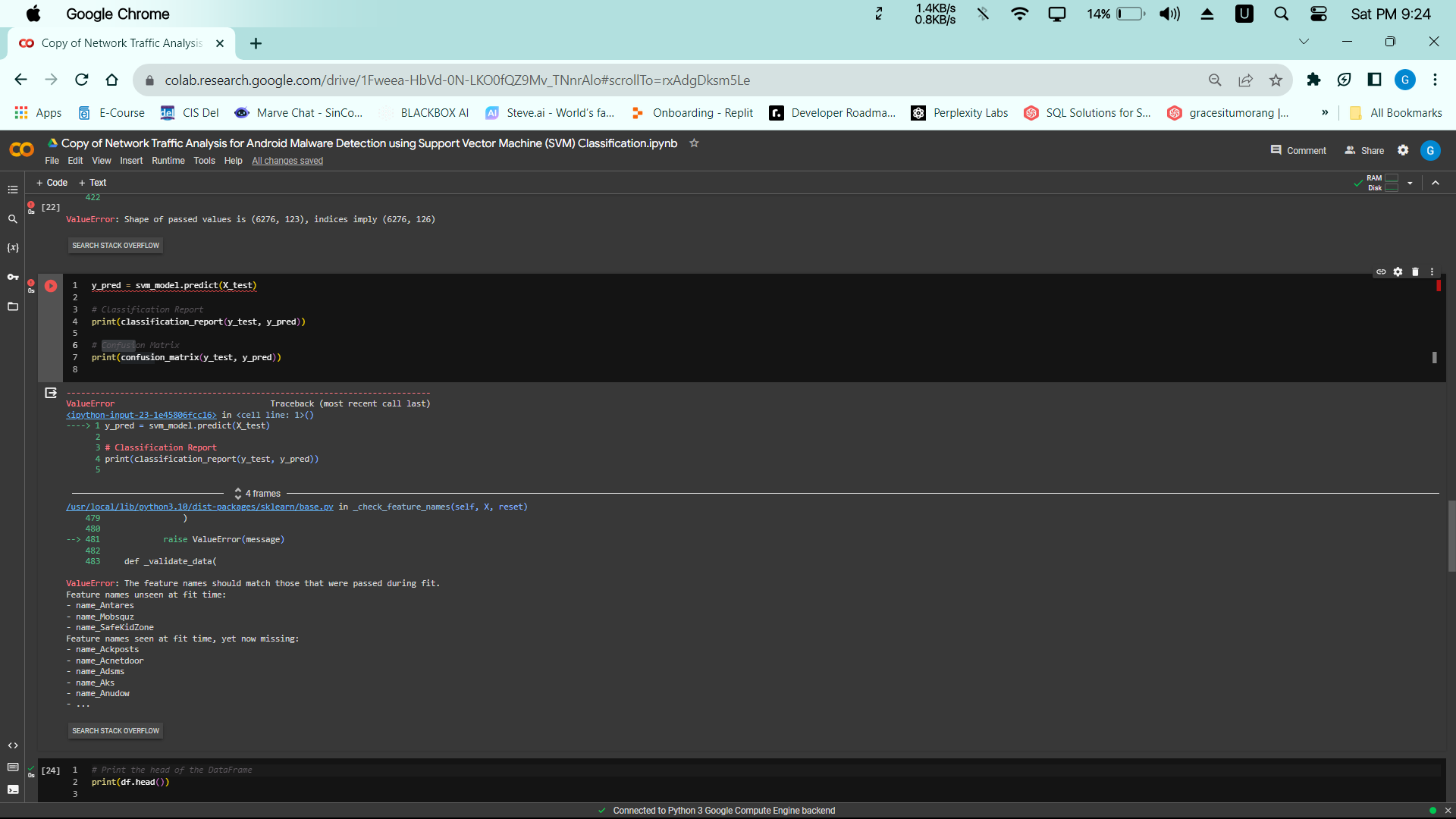
Bagian ini melibatkan penggunaan SimpleImputer untuk menangani nilai yang hilang dengan mengisinya menggunakan rata-rata, khususnya pada data latih.



1. Evaluasi Lanjutan dengan Data yang Telah Diimputasi

|  |
| --- |
| # Memastikan tidak ada nilai yang hilang setelah imputasi  columns = X\_train.columns  X\_train\_imputed\_df = pd.DataFrame(X\_train\_imputed, columns=columns)  print(X\_train\_imputed\_df.isnull().sum())  # Prediksi pada X\_test dengan data yang telah diimputasi  y\_pred = svm\_model.predict(X\_test)  # Classification Report  print(classification\_report(y\_test, y\_pred))  # Confusion Matrix  print(confusion\_matrix(y\_test, y\_pred)) |

Bagian ini memeriksa bahwa tidak ada nilai yang hilang setelah imputasi dan kemudian menggunakan model SVM yang telah dilatih untuk membuat prediksi pada data uji dengan data yang telah diimputasi, diikuti dengan mengevaluasi kinerjanya.



Catatan: Kode ini mengasumsikan bahwa y adalah variabel target dan bahwa Kita sudah mengimpor perpustakaan yang diperlukan (pkitas, sklearn, dll.)