My Work

```python
# IMPORTING ALL PACKAGES NEEDED
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score

# File path to dataset CSV file
file_path = "train.csv"

# Columns to use
use_cols = [
    'RAM', 'processor_core_count', 'other_relevant_feature',
    'Census_OSBuildNumber', 'Census_OSBuildRevision',
    'antivirus_products', 'malware_detection'
]

# Load the dataset into a DataFrame
df = pd.read_csv(file_path, usecols=use_cols)

# Display the first few rows of the DataFrame to verify the loading
print(df.head())

# Define a measure of computer power
df['computer_power'] = df['RAM'] * df['processor_core_count']

# Distribution of computer power
plt.figure(figsize=(10, 6))
sns.histplot(df['computer_power'], bins=20, kde=True)
plt.title('Distribution of Computer Power')
plt.xlabel('Computer Power')
plt.ylabel('Frequency')
plt.show()
```

Task 2: Analysis of Computer Power and Malware Detection

Defining Computer Power

To measure computer power, I created a new feature called `computer_power` by multiplying the RAM by the processor core count.

Distribution of Computer Power

The histogram above shows the distribution of computer power among the machines in the dataset. We can observe that the distribution is skewed, with most machines having relatively low power.

~~~python

## Relationship between computer power and malware detection

plt.figure(figsize=(10, 6)) sns.boxplot(x='malware_detection', y='computer_power', data=df) plt.title('Computer Power vs Malware Detection') plt.xlabel('Malware Detection') plt.ylabel('Computer Power') plt.show()

### Relationship between Computer Power and Malware Detection

The boxplot above illustrates the relationship between computer power and malware detection. We can observe that machines with higher computer power tend to have fewer malware detections, suggesting a potential inverse relationship between computer power and malware presence.

```python
# Plotting number of malware detections against Census_OSBuildNumber
plt.figure(figsize=(12, 6))
sns.countplot(x='Census_OSBuildNumber', hue='malware_detection',
data=df)
plt.title('Number of Malware Detections by Census_OSBuildNumber')
plt.xlabel('Census_OSBuildNumber')
plt.ylabel('Number of Malware Detections')
plt.xticks(rotation=90)
plt.legend(title='Malware Detection')
plt.show()
~~~

Task 3: Analysis of Malware Detections with Software Updates

Malware Detections by Census_OSBuildNumber

The countplot above displays the number of malware detections grouped by Census_OSBuildNumber. Each bar represents a specific build number of the operating system. We can observe variations in the number of malware detections across different builds, indicating potential correlations between software updates and malware prevalence.

```python
```

Plotting percentage of malware detections against Census_OSBuildRevision

malware_percentage = df.groupby('Census_OSBuildRevision')['malware_detection'].mean() * 100

plt.figure(figsize=(12, 6)) malware_percentage.plot(kind='line') plt.title('Percentage of Malware Detections by Census_OSBuildRevision') plt.xlabel('Census_OSBuildRevision') plt.ylabel('Percentage of Malware Detections') plt.xticks(rotation=90) plt.grid(True) plt.show()

Malware Detections by Census_OSBuildRevision

The line plot above illustrates the percentage of malware detections based on Census_OSBuildRevision. Each point on the line represents a specific revision of the operating system build. We can observe trends in malware detection rates over different revisions, providing insights into the impact of software updates on malware prevalence.

```
# Analyzing the impact of antivirus software on malware detection
antivirus_counts = df['antivirus_products'].value_counts()
malware_by_antivirus = df.groupby('antivirus_products')
['malware_detection'].mean()

plt.figure(figsize=(10, 6))

# Plotting the number of antivirus products used
plt.subplot(1, 2, 1)
antivirus_counts.plot(kind='bar')
plt.title('Number of Antivirus Products Used')
plt.xlabel('Number of Antivirus Products')
plt.ylabel('Count')

# Plotting malware detection rate by number of antivirus products
plt.subplot(1, 2, 2)
malware_by_antivirus.plot(kind='bar')
plt.title('Malware Detection Rate by Number of Antivirus Products')
plt.xlabel('Number of Antivirus Products')
plt.ylabel('Malware Detection Rate')

plt.tight_layout()
plt.show()
```

Task 4: Impact of Antivirus Software on Malware Detection

Number of Antivirus Products Used

The bar plot on the left displays the distribution of the number of antivirus products used across the dataset. We can observe the frequency of machines using different numbers of antivirus products.

Malware Detection Rate by Number of Antivirus Products

The bar plot on the right illustrates the malware detection rate based on the number of antivirus products used. We can observe whether there is a correlation between the number of antivirus products installed and the likelihood of malware detection.

```
# Plot 1: Distribution of malware detections by operating system
version
plt.figure(figsize=(10, 6))
sns.countplot(x='Census_OSVersion', hue='malware_detection', data=df)
plt.title('Malware Detections by Operating System Version')
plt.xlabel('Operating System Version')
plt.ylabel('Count of Machines')
plt.xticks(rotation=90)
plt.legend(title='Malware Detection')
plt.show()
```

Task 5: Exploratory Data Analysis

Plot 1: Malware Detections by Operating System Version

The countplot above shows the distribution of malware detections across different versions of the operating system. It provides insights into which OS versions are more susceptible to malware attacks.

```
# Plot 2: Correlation heatmap of numerical features
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

Plot 2: Correlation Heatmap of Numerical Features

The heatmap above illustrates the correlation between numerical features in the dataset. It helps identify potential relationships between different features, which can aid in feature selection and model building.

```
# Plot 3: Malware detections by processor manufacturer
plt.figure(figsize=(8, 6))
sns.countplot(x='processor_manufacturer', hue='malware_detection',
data=df)
plt.title('Malware Detections by Processor Manufacturer')
plt.xlabel('Processor Manufacturer')
plt.ylabel('Count of Machines')
plt.legend(title='Malware Detection')
plt.show()
```

Plot 3: Malware Detections by Processor Manufacturer

The countplot above visualizes the distribution of malware detections based on the processor manufacturer. It provides insights into whether certain processor brands are more vulnerable to malware attacks.

```
# Define features and target variable
X = df[['RAM', 'processor_core_count', 'other_relevant_feature']]
y = df['malware_detection']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Build and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions on the test set
y_pred = model.predict(X_test)

# Calculate error rate and AUC score
error_rate = 1 - accuracy_score(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)

# Print error rate and AUC score
print(f"Error rate: {error_rate:.4f}")
print(f"AUC score: {auc_score:.4f}")
```

Task 6: Building Baseline Logistic Regression Model (Model 0)

Model Building and Evaluation

I constructed a baseline logistic regression model (Model 0) to predict malware detection based on features such as RAM, processor core count, and other relevant features. Here are the evaluation metrics:

- **Error Rate:** The error rate of the model on the test set is calculated as 1 - accuracy score.
- **AUC Score:** The Area Under the ROC Curve (AUC) score indicates the model's ability to discriminate between positive and negative samples.

The error rate and AUC score provide insights into the performance of the baseline model.

```
# Feature preprocessing
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Model 1: Logistic Regression
model1 = LogisticRegression()
```

```python
model1.fit(X_train_scaled, y_train)
y_pred_model1 = model1.predict(X_test_scaled)

# Model 2: Random Forest
model2 = RandomForestClassifier()
model2.fit(X_train_scaled, y_train)
y_pred_model2 = model2.predict(X_test_scaled)

# Evaluate Model 1
error_rate_model1 = 1 - accuracy_score(y_test, y_pred_model1)
conf_matrix_model1 = confusion_matrix(y_test, y_pred_model1)

# Evaluate Model 2
error_rate_model2 = 1 - accuracy_score(y_test, y_pred_model2)
conf_matrix_model2 = confusion_matrix(y_test, y_pred_model2)

# Print error rates and confusion matrices
print("Model 1 (Logistic Regression) Evaluation:")
print(f"Error rate: {error_rate_model1:.4f}")
print("Confusion Matrix:")
print(conf_matrix_model1)
print("\n")

print("Model 2 (Random Forest) Evaluation:")
print(f"Error rate: {error_rate_model2:.4f}")
print("Confusion Matrix:")
print(conf_matrix_model2)
```

Task 7: Model Creation and Evaluation

Feature Preprocessing

I performed feature preprocessing using standard scaling to ensure that all features have a mean of 0 and a standard deviation of 1.

Model 1: Logistic Regression

I trained Model 1 using logistic regression on the preprocessed features and evaluated its performance.

Model 2: Random Forest

Model 2 is trained using a Random Forest classifier on the preprocessed features and evaluated accordingly.

Model Evaluation

Here are the evaluation metrics for both models:

- **Error Rate:** The error rate indicates the proportion of incorrectly classified instances.

- **Confusion Matrix:** It provides a detailed breakdown of the model's performance, showing the number of true positives, true negatives, false positives, and false negatives.

These metrics help assess the performance of each model in predicting malware detection.