Below is the detailed implementation for **Malware Detection Tasks** (Weeks 9-12). This includes structured code and explanations for preprocessing, exploratory data analysis, feature engineering, and machine learning models.

---

1. Problem Definition

Objective:

- Classify files into one of nine malware classes using `.asm` and `.bytes` file features.

Constraints:

- Handle large-scale data efficiently.
- Optimize performance to handle imbalanced class distributions.
- Models should generalize well and avoid overfitting.

---

2. Data Overview and Mapping

Load and Inspect Data

```
# Import necessary libraries
import pandas as pd

# Load data (replace with the actual file paths)
byte_data = pd.read_csv("byte_features.csv")  # Preprocessed features from .bytes files
asm_data = pd.read_csv("asm_features.csv")   # Preprocessed features from .asm files

# Merge both datasets
data = pd.merge(byte_data, asm_data, on="FileID")

# Check basic information
print(data.head())
print(data.info())
```

---

3. Exploratory Data Analysis

Class Distribution

```python
# Class distribution
print(data['MalwareClass'].value_counts())

# Plot class distribution
import matplotlib.pyplot as plt
data['MalwareClass'].value_counts().plot(kind='bar')
plt.title("Class Distribution")
plt.xlabel("Malware Classes")
plt.ylabel("Count")
plt.show()
```

Feature Extraction from `.bytes` Files

```python
from sklearn.feature_extraction.text import CountVectorizer

# Assuming the raw `.bytes` data is stored as strings in a column
vectorizer = CountVectorizer(max_features=5000, analyzer="word")
byte_features = vectorizer.fit_transform(byte_data['byte_text'])

# Convert to a DataFrame
byte_features_df = pd.DataFrame(byte_features.toarray(),
columns=vectorizer.get_feature_names_out())
```

Multivariate Analysis of Byte Features

```python
from sklearn.decomposition import PCA

# Apply PCA to reduce dimensions
pca = PCA(n_components=2, random_state=42)
byte_pca = pca.fit_transform(byte_features.toarray())

# Plot the PCA results
plt.scatter(byte_pca[:, 0], byte_pca[:, 1], c=data['MalwareClass'],
cmap='viridis', alpha=0.5)
plt.title("PCA of Byte Features")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label="Malware Class")
plt.show()
```

Train-Test Class Distribution

```python
from sklearn.model_selection import train_test_split

# Splitting the dataset
X = data.drop(columns=["FileID", "MalwareClass"])
y = data["MalwareClass"]

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

# Check distribution in train and test sets
print("Train Class Distribution:\n", y_train.value_counts(normalize=True))
print("Test Class Distribution:\n", y_test.value_counts(normalize=True))
```

---

4. ML Models Using Byte Files

Random Model

```python
import numpy as np
from sklearn.metrics import log_loss

# Generate random probabilities
num_classes = y.nunique()
random_probs = np.random.rand(len(y_test), num_classes)

# Normalize probabilities
random_probs = random_probs / random_probs.sum(axis=1, keepdims=True)

# Calculate Log Loss
random_logloss = log_loss(y_test, random_probs)
print(f"Random Model Log Loss: {random_logloss}")
```

K-Nearest Neighbors (K-NN)

```python
from sklearn.neighbors import KNeighborsClassifier

# Train K-NN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Evaluate K-NN
y_pred_knn = knn.predict_proba(X_test)
```

```
logloss_knn = log_loss(y_test, y_pred_knn)
print(f"K-NN Log Loss: {logloss_knn}")
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# Train Logistic Regression
lr = LogisticRegression(max_iter=1000, multi_class='multinomial')
lr.fit(X_train, y_train)

# Evaluate Logistic Regression
y_pred_lr = lr.predict_proba(X_test)
logloss_lr = log_loss(y_test, y_pred_lr)
print(f"Logistic Regression Log Loss: {logloss_lr}")
```

Random Forest and XGBoost

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Evaluate Random Forest
y_pred_rf = rf.predict_proba(X_test)
logloss_rf = log_loss(y_test, y_pred_rf)
print(f"Random Forest Log Loss: {logloss_rf}")
```

XGBoost

```
import xgboost as xgb

# Train XGBoost
xgb_model = xgb.XGBClassifier(objective='multi:softprob', n_estimators=100,
random_state=42)
xgb_model.fit(X_train, y_train)

# Evaluate XGBoost
y_pred_xgb = xgb_model.predict_proba(X_test)
```

```
logloss_xgb = log_loss(y_test, y_pred_xgb)
print(f"XGBoost Log Loss: {logloss_xgb}")
```

---

5. ASM File Feature Engineering

Extract Keywords and Opcodes

```
# Example keywords and opcodes
keywords = ['std::', ':dword', '.dll']
opcodes = ['jmp', 'mov', 'ret', 'push', 'pop']

# Count occurrences of each keyword and opcode
for keyword in keywords + opcodes:
    asm_data[keyword] = asm_data['asm_text'].str.count(keyword)
```

File Size Feature

```
# Add file size as a feature
asm_data['file_size'] = asm_data['asm_text'].apply(len)
```

T-SNE Analysis

```
from sklearn.manifold import TSNE

# Reduce dimensionality with T-SNE
tsne = TSNE(n_components=2, random_state=42)
asm_tsne = tsne.fit_transform(asm_data.drop(columns=['FileID', 
'MalwareClass']))

# Plot T-SNE results
plt.scatter(asm_tsne[:, 0], asm_tsne[:, 1], c=asm_data['MalwareClass'], 
cmap='tab10', alpha=0.7)
plt.title("T-SNE Analysis of ASM Features")
plt.xlabel("T-SNE Dimension 1")
plt.ylabel("T-SNE Dimension 2")
plt.colorbar(label="Malware Class")
plt.show()
```

---

6. Models on All Features

Combine Byte and ASM Features

```
combined_features = pd.concat([byte_features_df, asm_data.drop(columns=
['FileID', 'MalwareClass'])], axis=1)
```

Random Forest and XGBoost

Random Forest

```
rf_combined = RandomForestClassifier(n_estimators=100, random_state=42)
rf_combined.fit(combined_features.iloc[X_train.index], y_train)

# Evaluate
y_pred_rf_combined =
rf_combined.predict_proba(combined_features.iloc[X_test.index])
logloss_rf_combined = log_loss(y_test, y_pred_rf_combined)
print(f"Random Forest (Combined) Log Loss: {logloss_rf_combined}")
```

XGBoost

```
xgb_combined = xgb.XGBClassifier(objective='multi:softprob', n_estimators=100,
random_state=42)
xgb_combined.fit(combined_features.iloc[X_train.index], y_train)

# Evaluate
y_pred_xgb_combined =
xgb_combined.predict_proba(combined_features.iloc[X_test.index])
logloss_xgb_combined = log_loss(y_test, y_pred_xgb_combined)
print(f"XGBoost (Combined) Log Loss: {logloss_xgb_combined}")
```

Results Summary

```
results = pd.DataFrame({
    "Model": ["Random Model", "K-NN", "Logistic Regression", "Random Forest",
"XGBoost",
              "Random Forest (Combined)", "XGBoost (Combined)"],
    "Log Loss": [random_logloss, logloss_knn, logloss_lr, logloss_rf,
logloss_xgb,
                 logloss_rf_combined, logloss_xgb_combined]
})
print(results)
```

This implementation includes all tasks: EDA, feature extraction, preprocessing, and modeling for `.bytes` and `.asm` files. Let me know if you need further clarification!