Explain code step by step

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report

file_path = r"C:\Users\srush\Documents\Desktop\cl2\bi\data\human.txt"
df = pd.read_csv(file_path, sep='\t')
print(df.head())

sequences = df['sequence']
labels = df['class']

def get_kmers(seq, k=3):
    return [seq[i:i+k] for i in range(len(seq)-k+1)]

vectorizer = CountVectorizer(analyzer=get_kmers)
X = vectorizer.fit_transform(sequences)
y = labels

X_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())


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

rf = RandomForestClassifier(random_state=42).fit(X_train, y_train)
svm = SVC(kernel="linear").fit(X_train, y_train)

rf_pred, svm_pred = rf.predict(X_test), svm.predict(X_test)

def evaluate(name, y_true, y_pred):
    print(f"\n{name} Results:")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall:", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score:", f1_score(y_true, y_pred, average='weighted'))
    print(classification_report(y_true, y_pred))

evaluate("Random Forest", y_test, rf_pred)
evaluate("SVM", y_test, svm_pred)
```

```
def plot_cm(y_true, y_pred, title):
    sns.heatmap(confusion_matrix(y_true, y_pred), annot=True, fmt="d", cmap="Blues",
                xticklabels=["Healthy","Disease"], yticklabels=["Healthy","Disease"])
    plt.title(title); plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.show()

plot_cm(y_test, rf_pred, "Random Forest Confusion Matrix")
plot_cm(y_test, svm_pred, "SVM Confusion Matrix")

imp = rf.feature_importances_
idx = np.argsort(imp)[-10:][::-1]
plt.figure(figsize=(12,6))
plt.bar(range(10), imp[idx], tick_label=[X_df.columns[i] for i in idx])
plt.title("Top 10 Important k-mer Features"); plt.xticks(rotation=90); plt.show()

explain this code line by line
```

Absolutely 👍 Let's go through your code **step by step** — I'll explain **each line clearly**, including *what it does*, *why it's used*, and *how it fits into the overall machine learning workflow*.

🧩 Step 1: Importing the required libraries

```
python

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report
```

Explanation:

- **numpy (np)** → for numerical operations and arrays.
- **pandas (pd)** → for loading and manipulating tabular data.
- **matplotlib.pyplot** & **seaborn** → for visualizations (plots, confusion matrix).
- **CountVectorizer** → converts text data (DNA/protein sequences) into numerical feature vectors using token counts.
- **train_test_split** → splits data into training and testing sets.
- **RandomForestClassifier** → ensemble ML algorithm using multiple decision trees.
- **SVC** → Support Vector Machine classifier.
- **Metrics** → used for evaluating model performance.

🧬 Step 2: Loading the dataset

```
python

file_path = r"C:\Users\srush\Documents\Desktop\cl2\bi\data\human.txt"
df = pd.read_csv(file_path, sep='\t')
print(df.head())
```

Explanation:

- **file_path** → path to your dataset file (**`human.txt`**), containing tab-separated data.
- **pd.read_csv(..., sep='\t')** → reads the text file as a DataFrame, using tab as a separator.
- **df.head()** → prints the first 5 rows to inspect data.

✅ You likely have two columns:
`sequence` (DNA/protein sequence) and **`class`** (Healthy/Disease).

🔪 Step 3: Extracting sequences and labels

```
python

sequences = df['sequence']
labels = df['class']
```

Explanation:

- **sequences** → contains the biological sequences (strings of letters).
- **labels** → contains class labels for classification.

🧠 Step 4: Creating k-mers from sequences

```
python

def get_kmers(seq, k=3):
    return [seq[i:i+k] for i in range(len(seq)-k+1)]
```

Explanation:

- A **k-mer** is a substring of length **`k`** in the sequence (like a "word" in text).
- For example:
  Sequence = **`"ATGC"`** and k=3 → k-mers = **`["ATG", "TGC"]`**.
- This function slides a window of size **`k`** across the sequence to extract all possible k-mers.

🔡 Step 5: Converting sequences into numerical features

```
python

vectorizer = CountVectorizer(analyzer=get_kmers)
X = vectorizer.fit_transform(sequences)
y = labels
```

Explanation:

- **CountVectorizer** usually tokenizes words in text, but here you override its **`analyzer`** with your own function (**`get_kmers`**).
- It converts each sequence into a **bag-of-k-mers** representation — basically, it counts how many times each k-mer appears.
- **fit_transform()** learns all unique k-mers (vocabulary) and transforms sequences into a numeric matrix.
- **`X`** → feature matrix (rows = samples, columns = k-mer counts).
- **`y`** → target labels.

🗒️ Step 6: Convert feature matrix to DataFrame (for readability)

```
python

X_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
```

Explanation:

- **X.toarray()** → converts sparse matrix (default in scikit-learn) into a dense NumPy array.
- **columns=...** → assigns k-mer names as column headers.
- Now, you can view the features as a table (each k-mer as a column).

🔀 Step 7: Splitting data into train/test sets

```
python

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

Explanation:

- Splits dataset into:
  - **80% for training**
  - **20% for testing**
- **`random_state=42`** ensures reproducibility (same split every time).

🌲 Step 8: Training the models

```
python

rf = RandomForestClassifier(random_state=42).fit(X_train, y_train)
svm = SVC(kernel="linear").fit(X_train, y_train)
```

Explanation:

- **RandomForestClassifier** → trains multiple decision trees and averages their predictions to reduce overfitting.
- **SVC(kernel="linear")** → trains a linear Support Vector Machine classifier.
- Both models are trained using the same training data.

🎯 Step 9: Making predictions

```
python

rf_pred, svm_pred = rf.predict(X_test), svm.predict(X_test)
```

Explanation:

- Each model predicts labels for test data.
- **`rf_pred`** → predictions from Random Forest.
- **`svm_pred`** → predictions from SVM.

📊 Step 10: Evaluation function

```
python

def evaluate(name, y_true, y_pred):
    print(f"\n{name} Results:")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred, average='weighted'))
    print("Recall:", recall_score(y_true, y_pred, average='weighted'))
    print("F1 Score:", f1_score(y_true, y_pred, average='weighted'))
    print(classification_report(y_true, y_pred))
```

Explanation:

This helper function prints standard performance metrics:

- **Accuracy** → overall correctness.
- **Precision** → how many predicted positives are truly positive.
- **Recall** → how many actual positives were detected.
- **F1 Score** → harmonic mean of precision and recall.

- **classification_report()** → detailed per-class summary.

📃 Step 11: Evaluate both models

```
python

evaluate("Random Forest", y_test, rf_pred)
evaluate("SVM", y_test, svm_pred)
```

Explanation:

- Calls the **`evaluate()`** function for both models to display performance metrics.

🧩 Step 12: Plot confusion matrices

```
python

def plot_cm(y_true, y_pred, title):
    sns.heatmap(confusion_matrix(y_true, y_pred), annot=True, fmt="d", cmap="Blues",
                xticklabels=["Healthy","Disease"], yticklabels=["Healthy","Disease"])
    plt.title(title); plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.show()
```

Explanation:

- **confusion_matrix()** → shows how many samples were correctly or incorrectly classified.
- **sns.heatmap()** → visualizes it as a color-coded matrix.
- X/Y labels help you interpret prediction errors.

📈 Step 13: Visualize confusion matrices

```
python

plot_cm(y_test, rf_pred, "Random Forest Confusion Matrix")
plot_cm(y_test, svm_pred, "SVM Confusion Matrix")
```

Explanation:

- Plots confusion matrices for both classifiers to visually compare performance.

🌟 Step 14: Analyze important features

```
python

imp = rf.feature_importances_
idx = np.argsort(imp)[-10:][::-1]
plt.figure(figsize=(12,6))
```

```
plt.bar(range(10), imp[idx], tick_label=[X_df.columns[i] for i in idx])
plt.title("Top 10 Important k-mer Features"); plt.xticks(rotation=90); plt.show()
```

Explanation:

- **rf.feature_importances_** → gives importance scores for each k-mer (how useful each is for prediction).
- **np.argsort(imp)** → sorts indices of importances.
- **[-10:][::-1]** → selects top 10 most important features (in descending order).
- The bar chart shows which k-mers are most influential for classification.

🧠 Summary

Step	Concept	Purpose
1	Import libraries	Set up ML and plotting tools
2	Load dataset	Read sequence data
3	Extract columns	Separate input/output
4	Generate k-mers	Break sequences into fixed-length tokens
5	Vectorize	Convert text into numeric data
6	DataFrame	View and handle features easily
7	Split data	Prepare training and test sets
8	Train models	Learn patterns from data
9	Predict	Get model outputs
10–11	Evaluate	Measure performance
12–13	Plot confusion matrices	Visualize classification results
14	Feature importance	Understand which k-mers matter most

Would you like me to also **explain how to optimize this code** (like choosing best `k`, tuning Random Forest, or balancing classes)? It can help improve accuracy.