Step-1

```
import os
import numpy as np
import pandas as pd
import shap
shap.initjs()
import time
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import SelectKBest
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn import metrics
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
svc=SVC(probability=True, kernel='linear')
start = time.time()
```

Step-2

```
d_1=pd.read_excel(r"training_file.xlsx")
d_2=pd.read_excel(r"testing_file.xlsx")
```

Step-3

```
x_tr  = d_1.iloc[: , :-1]
y_tr1 = d_1.iloc[:,-1:]
x_ts  = d_2.iloc[: , :-1]
y_ts1 = d_2.iloc[:,-1:]
```

Step-4

```
x_tr = d_1[[selected features from fusion]
y_tr1= d_1.iloc[:,-1:]
x_ts = d_2[selected features from fusion]
y_ts1= d_1.iloc[:,-1:]
```

Step-5

```
x_tr=d_1[selected feature from SHAP]
y_tr1= d_1.iloc[:,-1:]
x_ts= d_2 [selected feature from SHAP]
y_ts1= d_2.iloc[:,-1:]
```

Step-6

```
x_tr= d_1[fusion and then SHAP to selected feature]
y_tr1= d_1.iloc[:,-1:]
x_ts = d_2 [fusion and then SHAP to selected feature]
y_ts1= d_1.iloc[:,-1:]
```

Step-7

```
(x_train, y_train),(x_test, y_test)=(x_tr,y_tr1),(x_ts,y_ts1)
model = SVC()
model = KNeighborsClassifier()
model = MultinomialNB()
model = LogisticRegression()
model = ExtraTreesClassifier()
history = model.fit(x_train, y_train)
y_pred = model.predict(x_test)
classification_report = classification_report(y_test, y_pred, output_dict=True)
end = time.time()
print(end - start, "seconds")
```

Step-8

```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(x_train)
shap.summary_plot(shap_values, x_train.values, max_display=10, plot_type="bar", class_names= label,
feature_names = x_train.columns)
```

Step-9

```
feature_names = x_train.columns)
f_value = f_classif(x_train, y_train)
feature_data = [(feature, value) for feature, value in zip(feature_names, f_value[0])]
sorted_feature_data = sorted(feature_data, key=lambda x: x[1], reverse=True)
df = pd.DataFrame(sorted_feature_data, columns=['Feature', 'Value'])
selector = VarianceThreshold()
selector.fit_transform(x_train)
feature_data = [(feature, variance) for feature, variance in zip(feature_names, selector.variances_)]
sorted_feature_data = sorted(feature_data, key=lambda x: x[1], reverse=True)
df = pd.DataFrame(feature_data, columns=['Feature', 'Variance'])
MI_score = mutual_info_classif(X_data, y_data, random_state=0)
feature_data = list(zip(feature_names, MI_score))
df = pd.DataFrame(feature_data, columns=['Feature', 'MI Score'])
skb = SelectKBest(score_func=f_classif, k=10)
x_train_new = skb.fit_transform(x_train, y_train)
selected_features = [feature_names[index] for index in skb.get_support(indices=True)]
df = pd.DataFrame({'Selected Features': selected_features})
sfs = SFS(estimator=lr, k_features=(1, 10), forward=True, scoring='accuracy', cv=5)
sfs = sfs.fit(x_train, y_train)
X_data_new = sfs.transform(X_data)
sbs_results = pd.DataFrame.from_dict(sbs.subsets_).T
sbs = SFS(estimator=lr, k_features=(1, 10),forward=False, scoring='accuracy', cv=5)
sbs = sbs.fit(x_train, y_train)
X_data_new = sbs.transform(X_data)
sbs_results = pd.DataFrame.from_dict(sbs.subsets_).T
model = RandomForestClassifier()
history = model.fit(x_train, y_train)
y_pred = model.predict(x_test)
feat_importances = pd.Series(model.feature_importances_, index= x_train.columns )
rf=feat_importances.nlargest(10)
model1= GradientBoostingClassifier()
history = model1.fit(x_train, y_train)
y_pred = model1.predict(x_test)
feature_importances = pd.Series(model1.feature_importances_, index= x_train.columns)
gb=feature_importances.nlargest(10)
```

Follow the instructions:
Here 4 Baseline model scenarios
B1: Performance Analysis of Machine Learning Classifiers
B2: Performance Analysis of Fusion-based Feature Selection
B3: Analysis of XAI-based Feature Selection
B4: Performance Analysis of sXFS

Follow the steps for B1:
1. Run the step 1
2. Run the step 2
3. Run the step 3
4. Run the step 7

Here choose the best model for the further case on the basis of highest accuracy.

Follow the steps for B2:
1. Run the step 1
2. Run the step 2
3. Run the step 3
4. Run the step 9

Make fusion of 10 most influential feature

5. Run the step 4
6. Run the step 7

Follow the steps for B3:
1. Run the step 1
2. Run the step 2
3. Run the step 3
4. Run the step 7
5. Run the step 8

Select most 50% feature based on model interpretability.

6. Run the step 5
7. Run the step 7

Follow the steps for B4:
1. Run the step 1
2. Run the step 2
3. Run the step 3
4. Run the step 9

Make fusion of 10 most influential feature

5. Run the step 4
6. Run the step 7
7. Run the step 8

Select most 50% feature based on model interpretability.

8. Run the step 6
9. Run the step 9