ICRRS – Code Explanation

Step 1:

```
import numpy as np
import pandas as pd from sklearn.ensemble
import RandomForestClassifier from sklearn.model_selection
import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
```

What This Does?

NumPy → Handles numerical operations

Pandas → Reads and processes the dataset

RandomForestClassifier → Machine Learning model for classification

Train-Test Split → Splits the dataset for training & testing

Accuracy, ROC, and AUC Metrics → Used to evaluate model performance

Matplotlib → Used for visualization

Step 2: Loading the Dataset

```
workloads  = pd.read_csv("/content/drive/MyDrive/Copy of PP_recipes.csv")

workloads.dtypes
```

What This Does?

- Reads the **PP_recipes.csv** dataset using Pandas
- Checks the **data types** of each column

Step 3: Mounting Google Drive (For Google Colab Users)

```
from google.colab import drive

drive.mount('/content/drive')
```

What This Does?

- Mounts Google Drive in **Google Colab** to access files

Step 4: Handling Missing Data & Converting to Numeric

```
workloads['name_tokens'] = pd.to_numeric(workloads['name_tokens'], errors='coerce')

workloads['ingredient_tokens'] = pd.to_numeric(workloads['ingredient_tokens'], errors='coerce')

workloads['steps_tokens'] = pd.to_numeric(workloads['steps_tokens'], errors='coerce')

workloads['techniques'] = pd.to_numeric(workloads['techniques'], errors='coerce')

workloads['calorie_level'] = pd.to_numeric(workloads['calorie_level'], errors='coerce')

workloads['ingredient_ids'] = pd.to_numeric(workloads['ingredient_ids'], errors='coerce')

workloads = workloads.fillna(0).astype(np.int64, errors='ignore')
```

What This Does?

- Converts **text-based features into numeric values**
- Handles missing values using **fillna(0)**
- Converts the dataset to **int64 format**

What This Does?

- **X** → Takes first 7 columns as input features

- **Y** → Takes the 8th column as the target variable

Step 5: Splitting Features & Target Variable

```
X = workloads.iloc[:,0:7]

Y = workloads.iloc[:,7].values
```

What This Does?

- **X** → Takes first 7 columns as input features
- **Y** → Takes the 8th column as the target variable

Step 6: Splitting Data into Training & Testing Sets

```
train_x, test_x, train_y, test_y = train_test_split(X, Y, test_size=15)
```

What This Does?

- Splits the dataset into **Training Set (train_x, train_y)** & **Testing Set (test_x, test_y)**
- `test_size=15` → Uses **15% of the dataset for testing**

Step 7: Training the Random Forest Classifier

```
rfc = RandomForestClassifier(n_estimators=70, criterion='gini')

rfc.fit(train_x, train_y)
```

What This Does?

- Creates a **Random Forest Classifier**
  - `n_estimators=70` → Uses 70 decision trees
  - `criterion='gini'` → Splitting criteria for decision trees
- Fits the model on the **training dataset**

Step 8: Making Predictions

```
y_pred3 = rfc.predict(test_x)
```

What This Does?

- Uses the trained model to **predict values for test data**

Step 9: Evaluating Model Performance

```
print("Accuracy Score of Random Forest Classifier : ",
accuracy_score(y_pred3, test_y))
```

What This Does?

- Calculates the **accuracy score** of predictions

Step 10: Generating a Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(test_y, y_pred3))
```

What This Does?

- Displays precision, recall, F1-score, and support for each class

Possible Advanced Improvements

1. Feature Selection for Better Performance

- Use `SelectKBest` to select the most important features.

```
python
CopyEdit
from sklearn.feature_selection import
SelectKBest, f_classif
best_features =
SelectKBest(score_func=f_classif, k=5)
```

```
X_new = best_features.fit_transform(X, Y)
```

2. Hyperparameter Tuning

- Use `GridSearchCV` to find the best parameters.

```
python
CopyEdit
from sklearn.model_selection import
GridSearchCV
param_grid = {'n_estimators': [50, 100,
150], 'criterion': ['gini', 'entropy']}
grid =
GridSearchCV(RandomForestClassifier(),
param_grid, cv=5)
grid.fit(train_x, train_y)
print(grid.best_params_)
```

3. Try Another Model for Comparison

- Compare with **XGBoost**

```
python
CopyEdit
from xgboost import XGBClassifier
xgb = XGBClassifier(n_estimators=100)
xgb.fit(train_x, train_y)
```

4. Visualizing Feature Importance

- Check which features impact the model the most.

```
python
CopyEdit
importances = rfc.feature_importances_
plt.barh(X.columns, importances)
```

```
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.show()
```