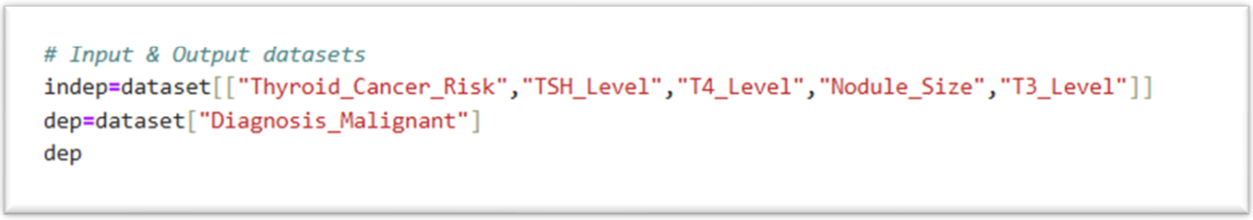
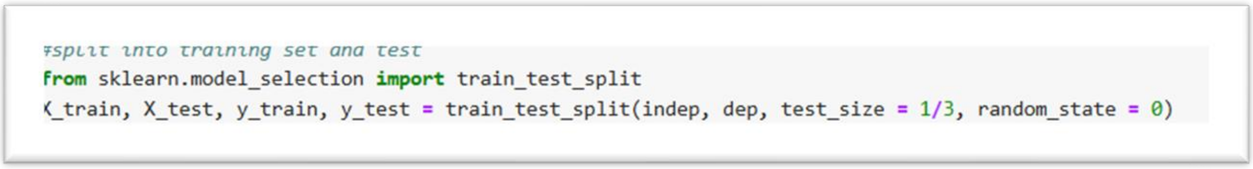
# Model Creation

1. **Defining datasets for Input and Output**



```
# Input & Output datasets
indep=dataset[["Thyroid_Cancer_Risk","TSH_Level","T4_Level","Nodule_Size","T3_Level"]]
dep=dataset["Diagnosis_Malignant"]
dep
```

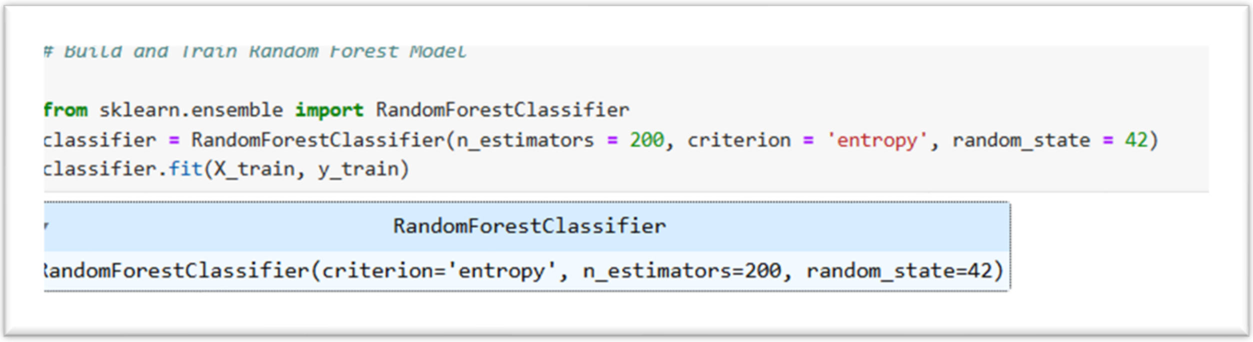
2. **Splitting Training and Test set**



```
#split into training set and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(indep, dep, test_size = 1/3, random_state = 0)
```

test_size = 1/3 → one third of the data will be used for testing, other for training.

3. **Build and Train the mode**

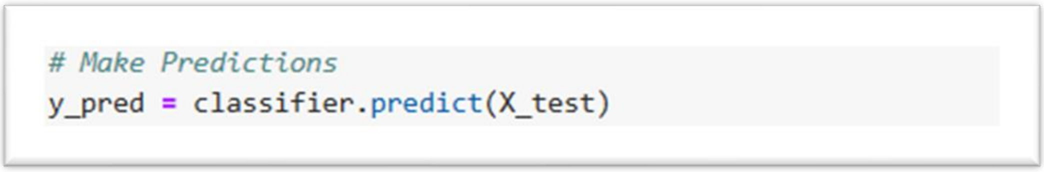


```
# Build and Train Random Forest Model

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 200, criterion = 'entropy', random_state = 42)
classifier.fit(X_train, y_train)
```

```
                    RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=200, random_state=42)
```

200 →no. of trees , entropy →splitting criteria
.fit(X_train, y_train) → trains the model using training data

4. **Making Predictions**



```
# Make Predictions
y_pred = classifier.predict(X_test)
```

Making predictions for test data by using trained model

## 5. Evaluating the Model

To assess the model thoroughly, the following metrics were calculated:

**✓ Accuracy**

Measures the percentage of correct predictions made by the model.

**✓ Precision**

Indicates how many predicted malignant cases were actually malignant.
*Important in medical diagnosis to reduce false alarms.*

**✓ Recall (Sensitivity)**

Measures how many actual malignant cases the model successfully detected.
*Critical for catching true positive cancer cases.*

**✓ F1-Score**

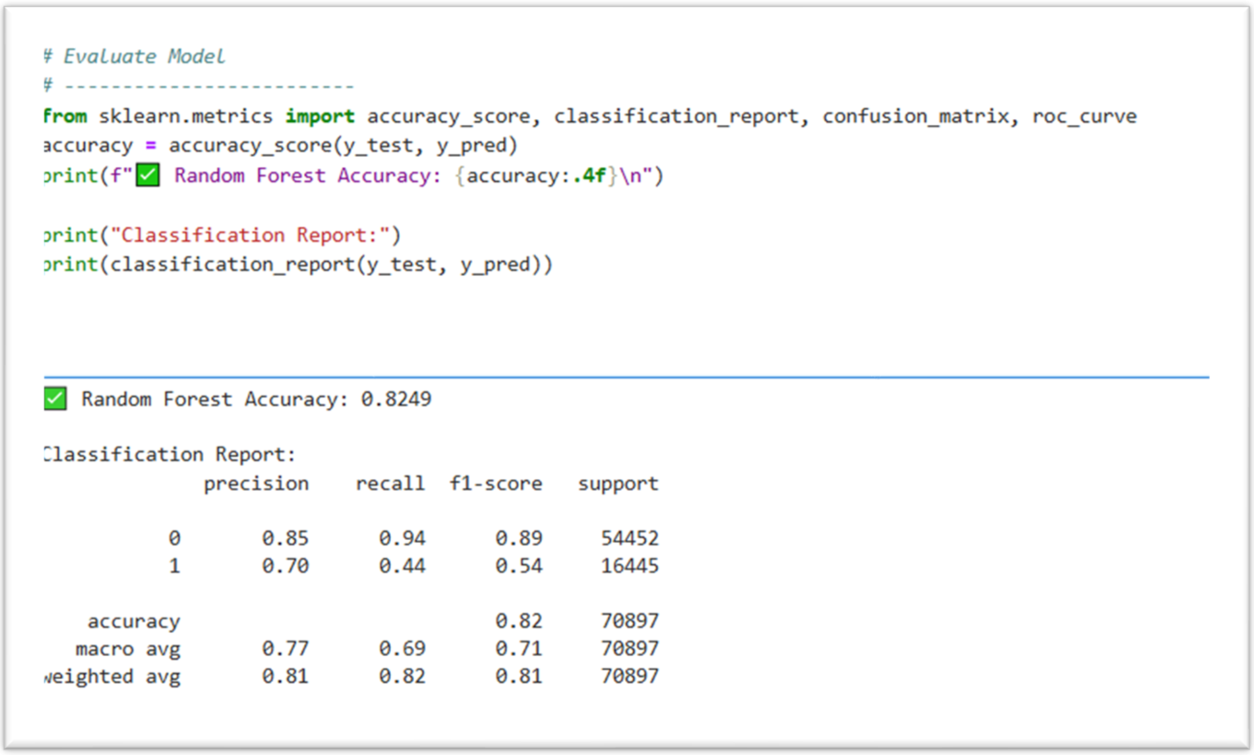
Harmonic mean of Precision and Recall.
*Balances both false positives and false negatives.*

**✓ ROC Curve (Receiver Operating Characteristic)**

Plots the trade-off between True Positive Rate and False Positive Rate.

**✓ AUC Score (Area Under Curve)**

Represents how well the model differentiates between malignant and benign cases.



```
# Evaluate Model
# --------------------------
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve
accuracy = accuracy_score(y_test, y_pred)
print(f"✅ Random Forest Accuracy: {accuracy:.4f}\n")

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
✅ Random Forest Accuracy: 0.8249

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.94      0.89     54452
           1       0.70      0.44      0.54     16445

    accuracy                           0.82     70897
   macro avg       0.77      0.69      0.71     70897
weighted avg       0.81      0.82      0.81     70897
```

Accuracy – 82%

**Precision :**

Precision tells you **how many of the positive predictions were actually correct**.

70% → Higher precision = fewer false alarms (fewer people wrongly predicted as at risk).

**Recall:**

Recall tells you **how many of the actual positive cases were correctly identified**.

44% → Higher recall = fewer missed actual risk cases (model is sensitive in catching positives).

**F1-Score :**

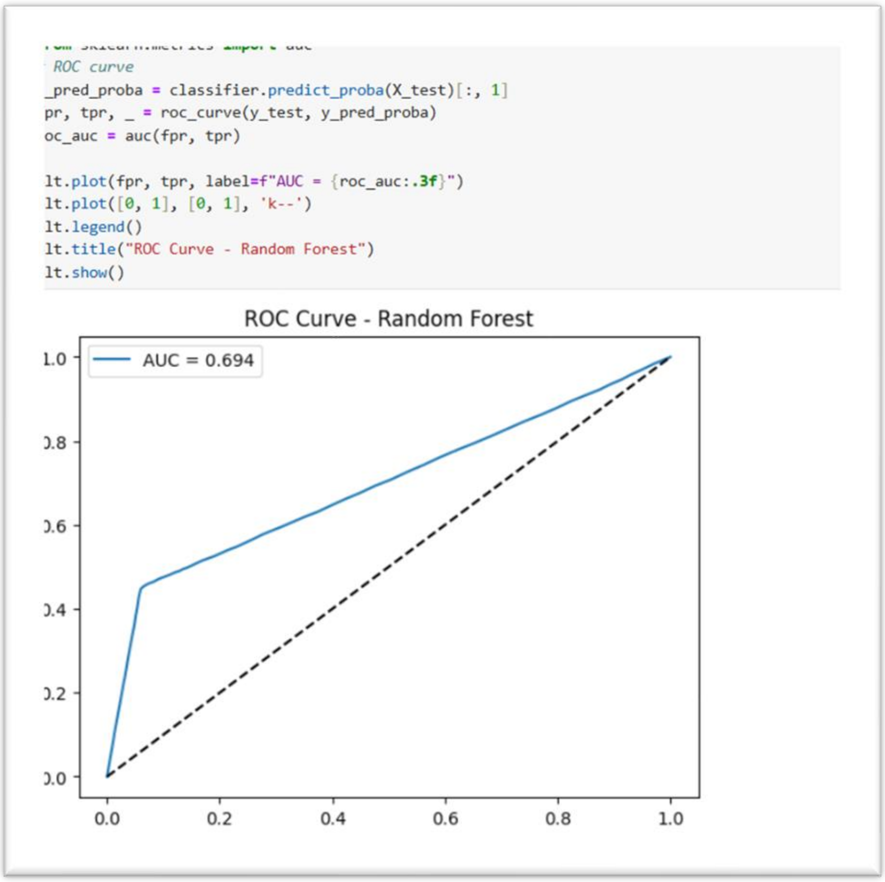
F1-score is the **harmonic mean** of precision and recall — it gives a single metric that balances both.

54% → Higher F1 means a good balance between catching actual positives and avoiding false alarms.

**Summary:**

- The **Random Forest Classifier** achieved an **accuracy of 82.49%** on the test dataset.
- **Precision** for class 0 (non-risk cases) is **0.85**, while for class 1 (risk cases) it is **0.70**.
- **Recall** for class 0 is high (**0.84**), but comparatively lower (**0.44**) for class 1, indicating that some high-risk cases were missed.
- The **weighted average F1-score** of **0.81** reflects an overall strong and balanced model performance.

6. **Evaluation - ROC**



```
om sklearn.metrics import auc
 ROC curve
_pred_proba = classifier.predict_proba(X_test)[:, 1]
pr, tpr, _ = roc_curve(y_test, y_pred_proba)
oc_auc = auc(fpr, tpr)

lt.plot(fpr, tpr, label=f"AUC = {roc_auc:.3f}")
lt.plot([0, 1], [0, 1], 'k--')
lt.legend()
lt.title("ROC Curve - Random Forest")
lt.show()
```

**ROC (Receiver Operating Characteristic)** curve is a **graphical plot** used to show the performance of a classification model **at different threshold values**.

- ❖ The ROC curve shows how the **model's sensitivity (True Positive Rate)** and **1 - specificity (False Positive Rate)** change as you move this threshold.
  - **X-axis:** False Positive Rate (FPR)
    → Percentage of healthy patients wrongly predicted as thyroid risk.
  - **Y-axis:** True Positive Rate (TPR)
    → Percentage of actual thyroid-risk patients correctly identified.
- ❖ Each point on the curve represents a model's performance at a different probability threshold.

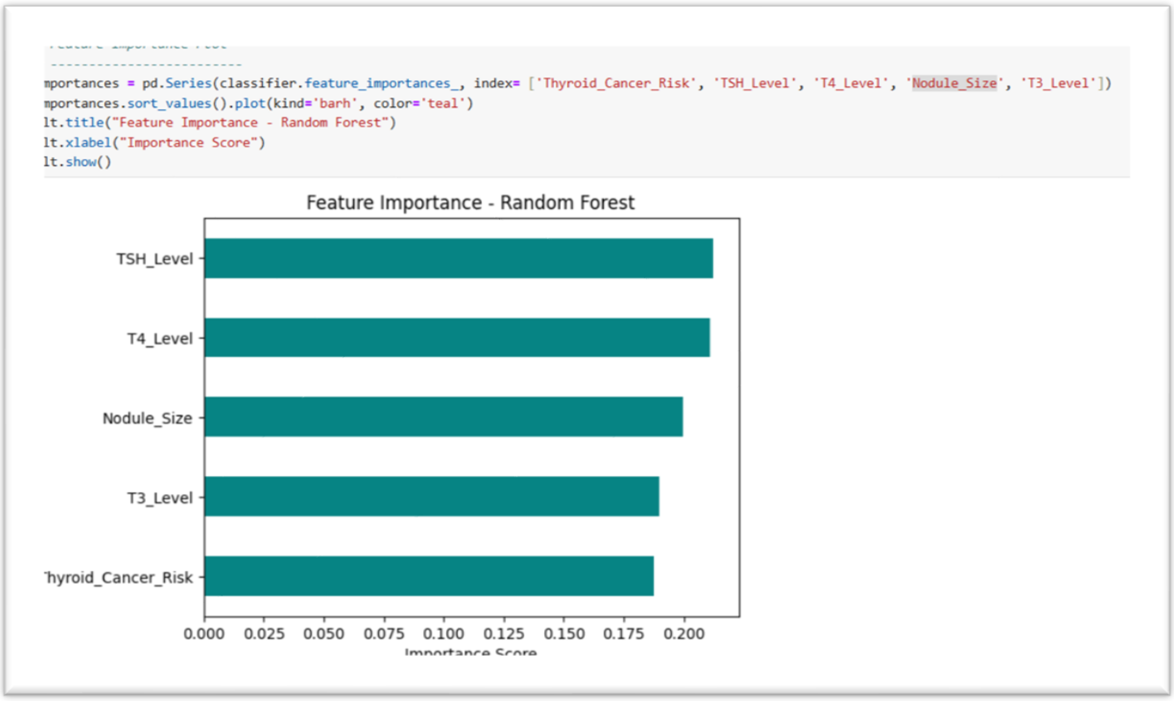
**AUC (Area Under the ROC Curve)** is a **numerical summary** of the ROC curve — it measures how well the model distinguishes between the two classes (risk vs. no risk).

- **AUC = 1.0 → Perfect model** (flawless distinction)
- **AUC = 0.5 → Random model** (like tossing a coin)
- **AUC between 0.7–0.8 → Fair**
- **AUC between 0.8–0.9 → Good**
- **AUC above 0.9 → Excellent**

- ❖ The model achieved an AUC score of **0.694**, indicating a **moderate discriminative ability**.
- ❖ This means that approximately **69% of the time**, the model correctly ranks a patient with thyroid risk higher than one without risk.

## 7. Feature Importance Plot

A **Feature Importance Plot** shows **how much each feature (input variable) contributes** to the model's predictions.



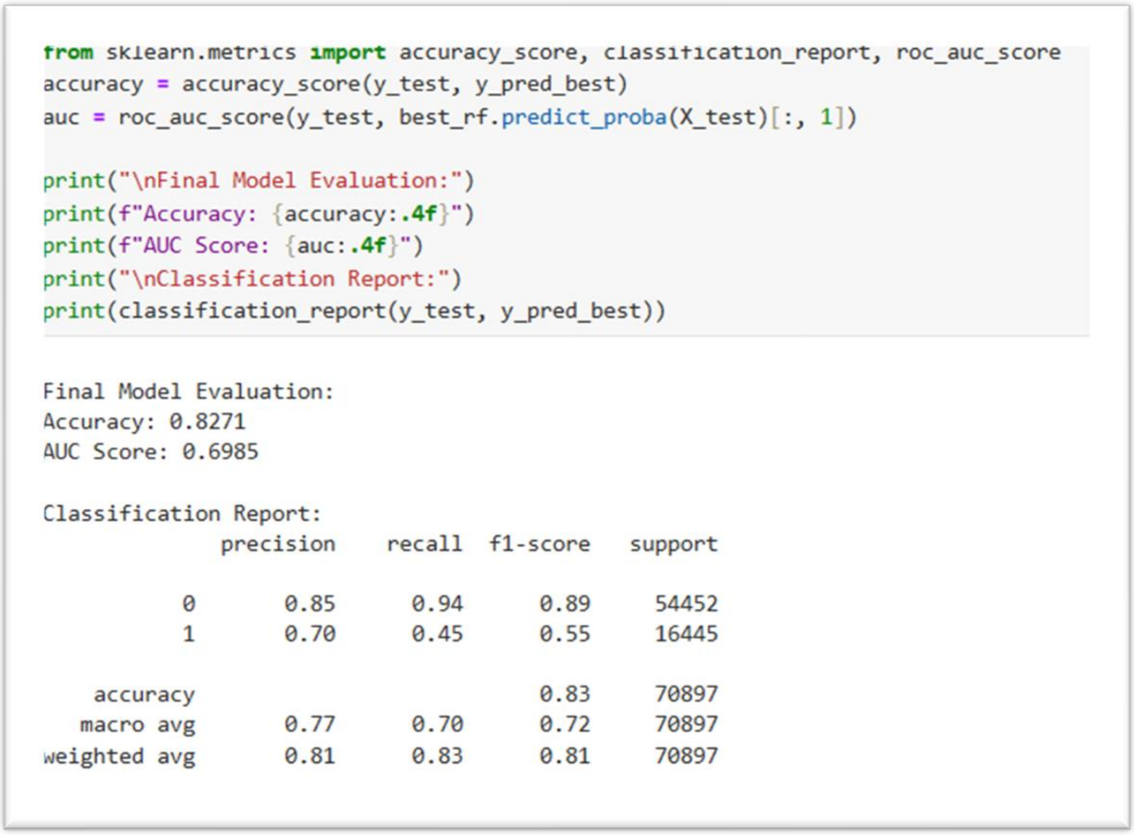
```
-------------------------
mportances = pd.Series(classifier.feature_importances_, index= ['Thyroid_Cancer_Risk', 'TSH_Level', 'T4_Level', 'Nodule_Size', 'T3_Level'])
mportances.sort_values().plot(kind='barh', color='teal')
lt.title("Feature Importance - Random Forest")
lt.xlabel("Importance Score")
lt.show()
```

Feature importance plot shows TSH_Level as the highest bar — it means the model relies **heavily** on TSH levels to decide the diagnosis.

## 8. Hyperparameter Tuning

To check whether the accuracy is increased with any other parameter we are using grid search .

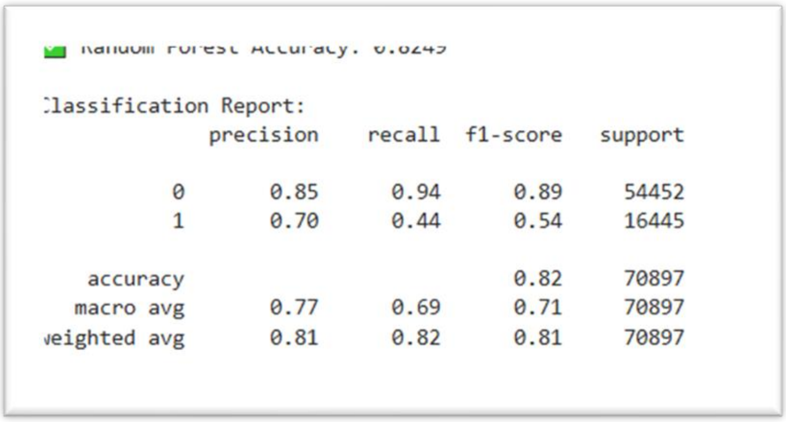
Grid Search tests different combinations of **hyperparameters** for your model and finds the one that gives **the best accuracy or performance metric** (like F1-score, ROC-AUC, etc.).



```python
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score
accuracy = accuracy_score(y_test, y_pred_best)
auc = roc_auc_score(y_test, best_rf.predict_proba(X_test)[:, 1])

print("\nFinal Model Evaluation:")
print(f"Accuracy: {accuracy:.4f}")
print(f"AUC Score: {auc:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_best))
```

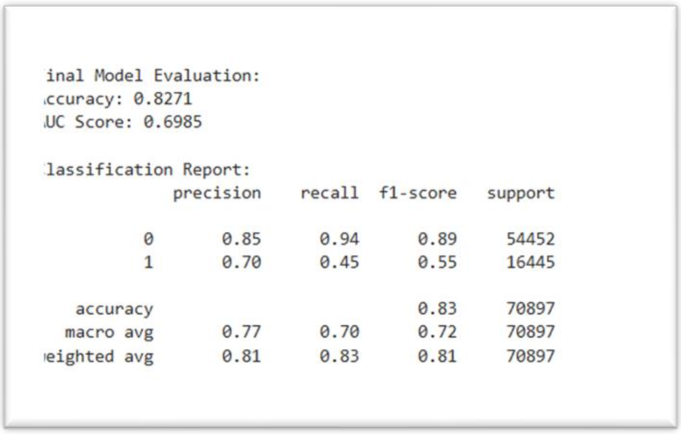
```
Final Model Evaluation:
Accuracy: 0.8271
AUC Score: 0.6985

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.94      0.89     54452
           1       0.70      0.45      0.55     16445

    accuracy                           0.83     70897
   macro avg       0.77      0.70      0.72     70897
weighted avg       0.81      0.83      0.81     70897
```

Before grid search



```
✅ Random Forest Accuracy: 0.8249

lassification Report:
              precision    recall  f1-score   support

           0       0.85      0.94      0.89     54452
           1       0.70      0.44      0.54     16445

    accuracy                           0.82     70897
   macro avg       0.77      0.69      0.71     70897
eighted avg       0.81      0.82      0.81     70897
```

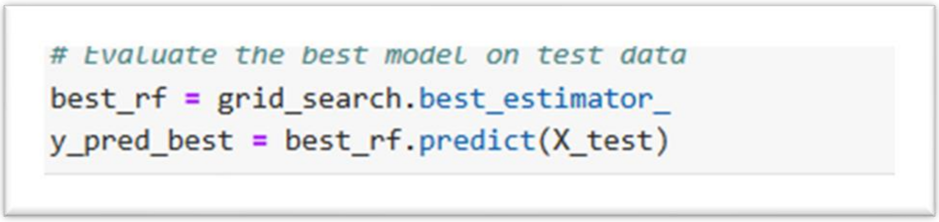
Grid search results



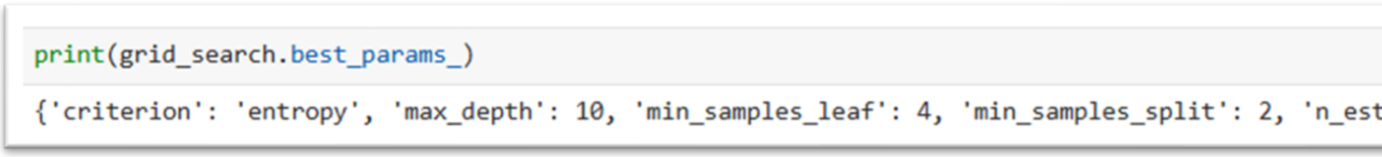
```
inal Model Evaluation:
ccuracy: 0.8271
UC Score: 0.6985

lassification Report:
              precision    recall  f1-score   support

           0       0.85      0.94      0.89     54452
           1       0.70      0.45      0.55     16445

    accuracy                           0.83     70897
   macro avg       0.77      0.70      0.72     70897
eighted avg       0.81      0.83      0.81     70897
```

- Grid Search helped fine-tune the model and resulted in consistent, slight improvements across key metrics including accuracy, AUC score, and performance on the malignant class.
- While the improvements are not large, the optimized model is more stable and reliable than the initial version.

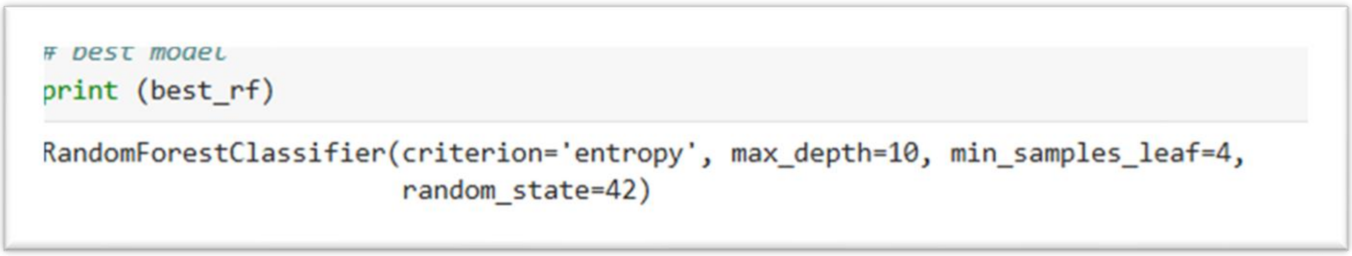
**9. Saving the best model**



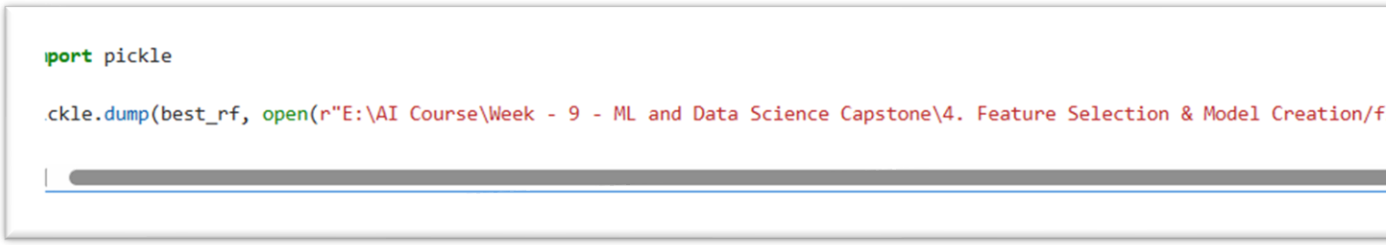
```
# Evaluate the best model on test data
best_rf = grid_search.best_estimator_
y_pred_best = best_rf.predict(X_test)
```



```
print(grid_search.best_params_)
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_est
```



```
# best model
print (best_rf)
RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_leaf=4,
                       random_state=42)
```

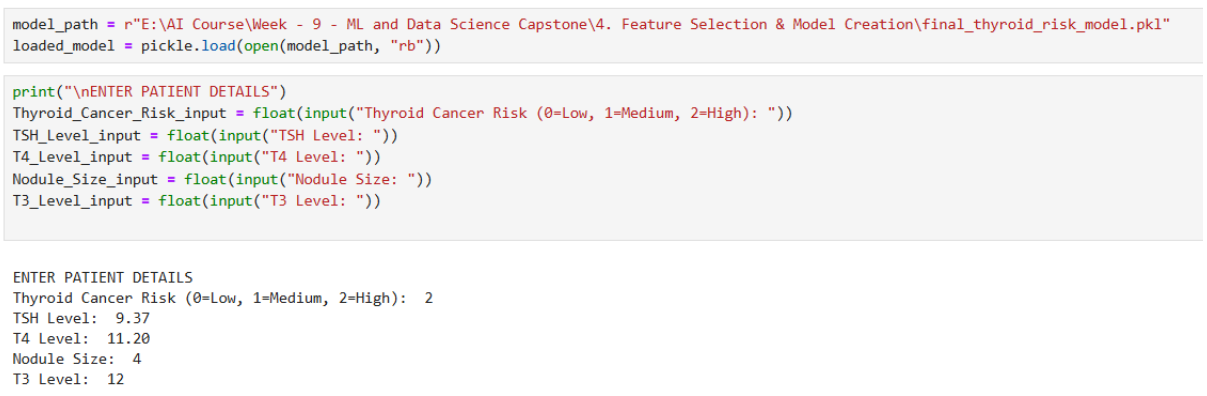


```
port pickle

ckle.dump(best_rf, open(r"E:\AI Course\Week - 9 - ML and Data Science Capstone\4. Feature Selection & Model Creation/f
```

Model saved by using pickle.

## 10. Deployment File

Created a separate file for deployment. Will predict the output for User Input.

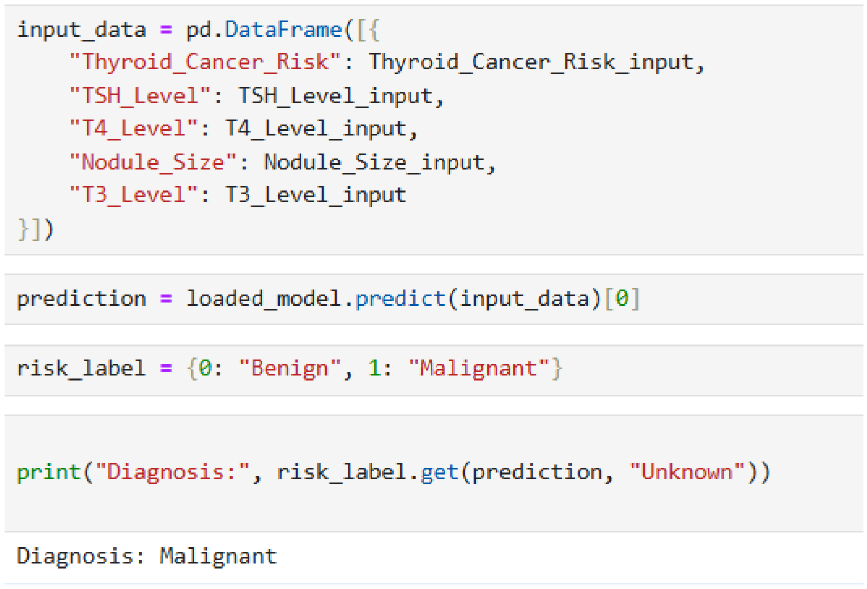


```
model_path = r"E:\AI Course\Week - 9 - ML and Data Science Capstone\4. Feature Selection & Model Creation\final_thyroid_risk_model.pkl"
loaded_model = pickle.load(open(model_path, "rb"))
```

```
print("\nENTER PATIENT DETAILS")
Thyroid_Cancer_Risk_input = float(input("Thyroid Cancer Risk (0=Low, 1=Medium, 2=High): "))
TSH_Level_input = float(input("TSH Level: "))
T4_Level_input = float(input("T4 Level: "))
Nodule_Size_input = float(input("Nodule Size: "))
T3_Level_input = float(input("T3 Level: "))
```

```
ENTER PATIENT DETAILS
Thyroid Cancer Risk (0=Low, 1=Medium, 2=High):  2
TSH Level:  9.37
T4 Level:  11.20
Nodule Size:  4
T3 Level:  12
```

User Input provided.



```
input_data = pd.DataFrame([{
    "Thyroid_Cancer_Risk": Thyroid_Cancer_Risk_input,
    "TSH_Level": TSH_Level_input,
    "T4_Level": T4_Level_input,
    "Nodule_Size": Nodule_Size_input,
    "T3_Level": T3_Level_input
}])
```

```
prediction = loaded_model.predict(input_data)[0]
```

```
risk_label = {0: "Benign", 1: "Malignant"}
```

```
print("Diagnosis:", risk_label.get(prediction, "Unknown"))
```

```
Diagnosis: Malignant
```

- Got the input in a dataframe.
- By using the loaded model predicting the input dataframe & saving to prediction.
- Used dictionary called risk_label to convert the model’s prediction (0 or 1) into a readable diagnosis.

- The dictionary **maps numbers to text**:
  - **0 → Benign**
  - **1 → Malignant**

  - `.get(prediction)` looks for the prediction (0 or 1) in the dictionary.

  - If the prediction exists → it returns the correct label ("Benign" or "Malignant").

  - If the prediction is not found → it returns **"Unknown"** instead of giving an error.

## <u>Prediction Result:</u>

Here it predicted the patient has Malignant Nodule.