# Machine learning with python

# Individual assignment

Name: K. PRANVITH CHOWDARY

ID: SE22UCSE133

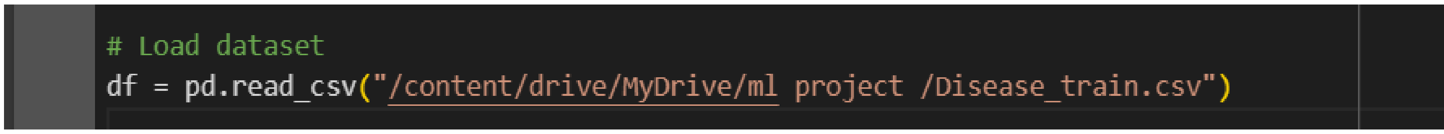
## REPORT

## Introduction:

- This report outlines the development of a machine learning model based on the dataset (disease_train.csv) and capable of predicting the presence of a disease. The dataset features are :
- `patient_id`: Unique identifier for each patient
- `feature_1` to `feature_n`: Anonymized patient features
- `diagnosis`: Target variable (0 = no disease, 1 = disease)

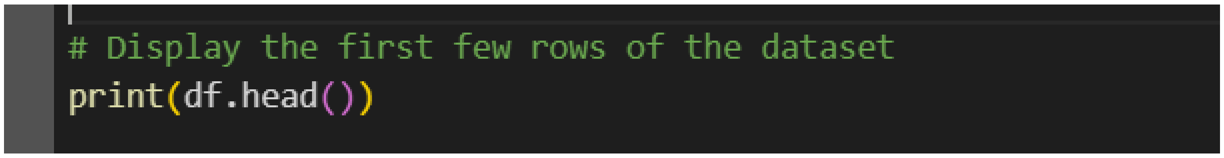
Data overview:

Firstly, loading the dataset:



```
# Load dataset
df = pd.read_csv("/content/drive/MyDrive/ml project /Disease_train.csv")
```

Secondly, displaying the first few rows to understand the content and features of the dataset:

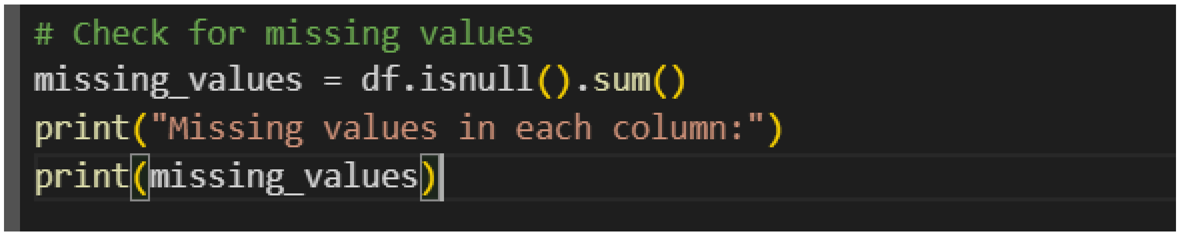


```
# Display the first few rows of the dataset
print(df.head())
```

## Data preprocessing:

Handling missing values:

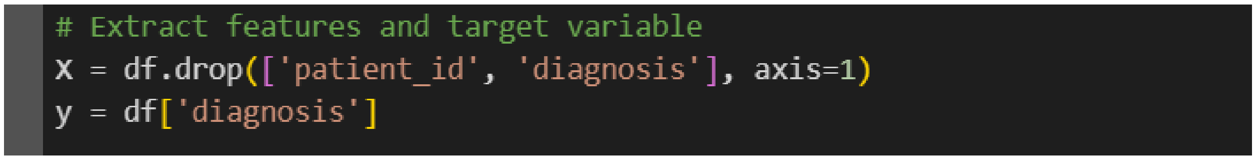
We check for the missing values in the dataset using the isnull function and print them:



```
# Check for missing values
missing_values = df.isnull().sum()
print("Missing values in each column:")
print(missing_values)
```

Feature and target variable extraction:

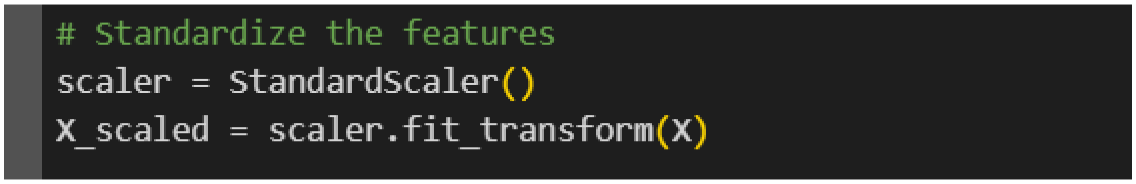
We extract the feature and target variable using the drop function i.e.
In features(X): we drop the patient_id and diagnosis columns,
In target variable(y): we drop the diagnosis column:



```
# Extract features and target variable
X = df.drop(['patient_id', 'diagnosis'], axis=1)
y = df['diagnosis']
```

Standardisation:

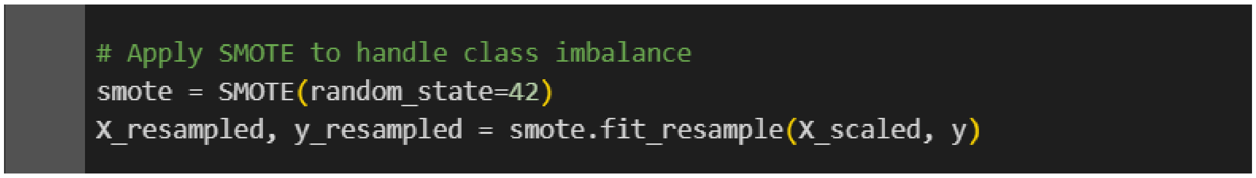
We standardise the features to contribute to the model equally



```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Handling unbalanced data by sampling:

We use smote (Synthetic Minority Over-sampling Technique) to balance the unbalanced data:

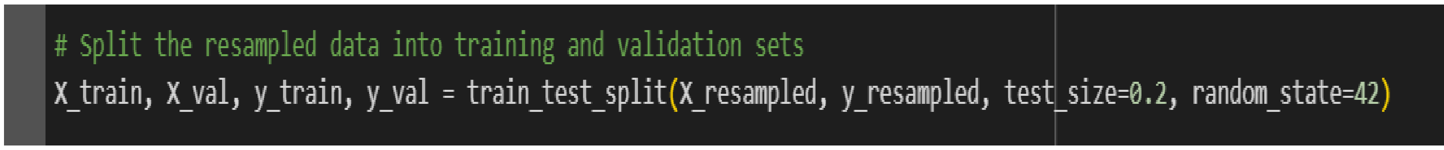


```
# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

## Model training:

## Splitting data:

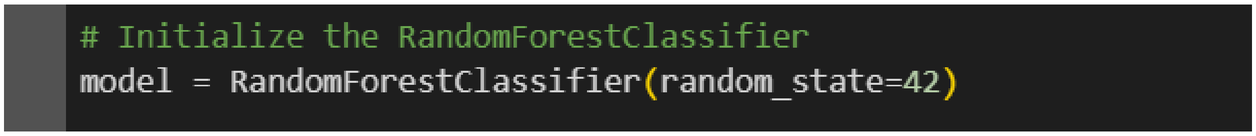
We split the features and target variable into training set and validation set:



```
# Split the resampled data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

## Random forest classifier:

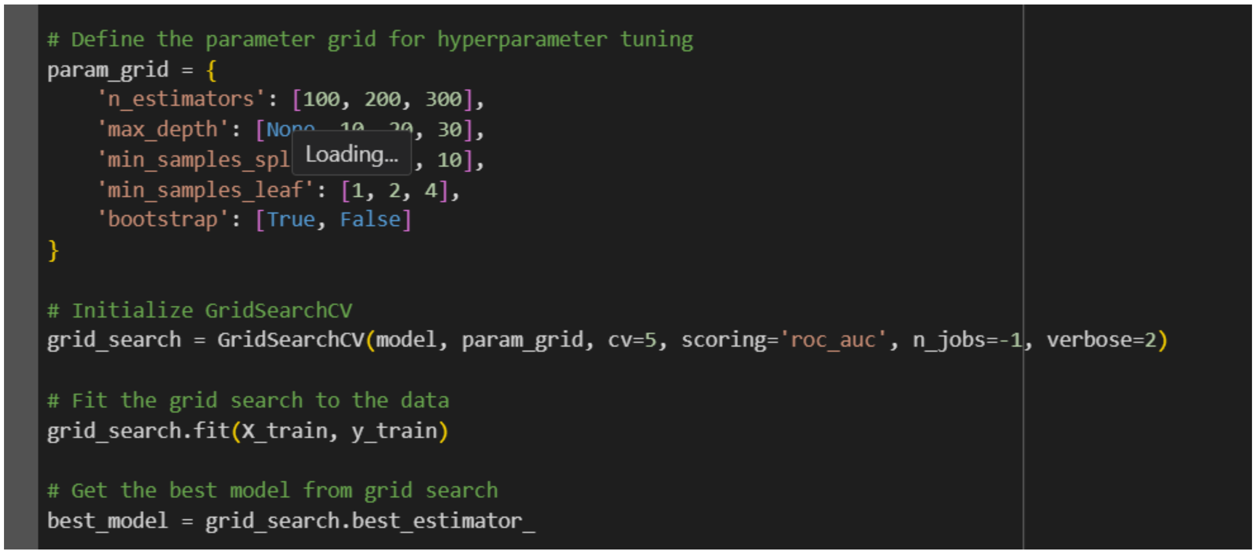
Using random forest classifier with a random state as 42 we train the model:



```
# Initialize the RandomForestClassifier
model = RandomForestClassifier(random_state=42)
```

## Hyper parameter tuning:

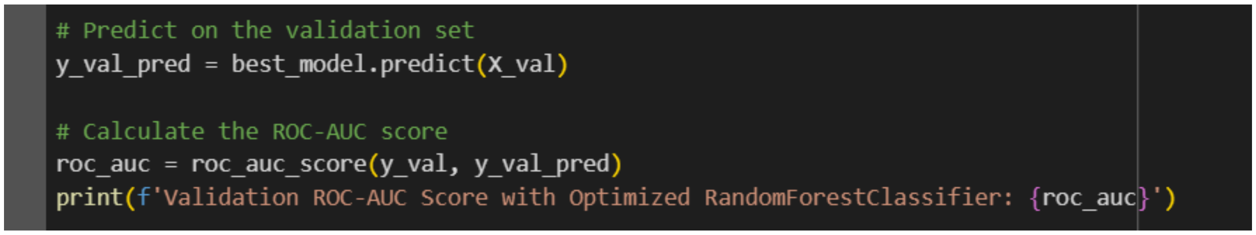
We optimize the model hyperparameters for the best performance of the model using different parameters and grid search:



```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_spl Loading... , 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='roc_auc', n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best model from grid search
best_model = grid_search.best_estimator_
```

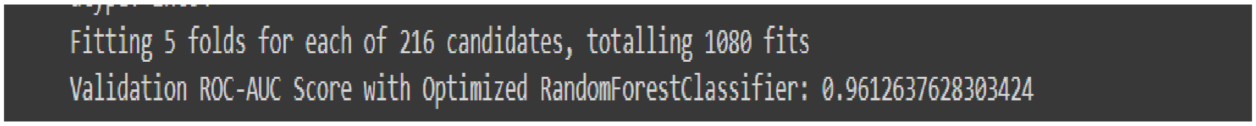
## Model evaluation:

The ROC-AUC score was calculated on the validation set to evaluate model performance:



```python
# Predict on the validation set
y_val_pred = best_model.predict(X_val)

# Calculate the ROC-AUC score
roc_auc = roc_auc_score(y_val, y_val_pred)
print(f'Validation ROC-AUC Score with Optimized RandomForestClassifier: {roc_auc}')
```

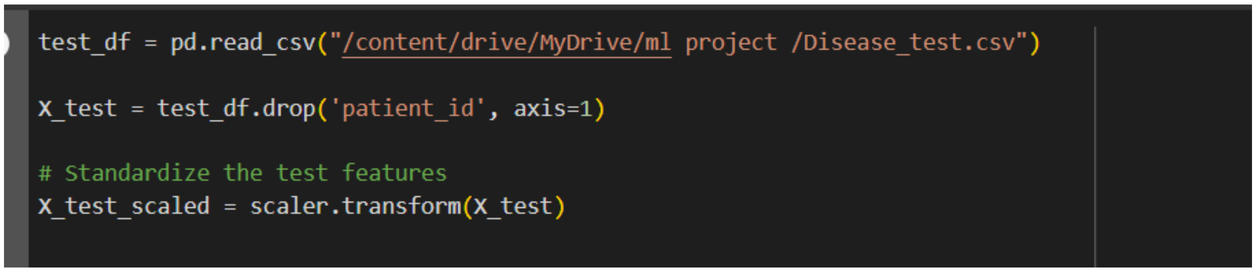
And the validation scores using ROC_AUC metric are 0.9612637628303424



```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Validation ROC-AUC Score with Optimized RandomForestClassifier: 0.9612637628303424
```

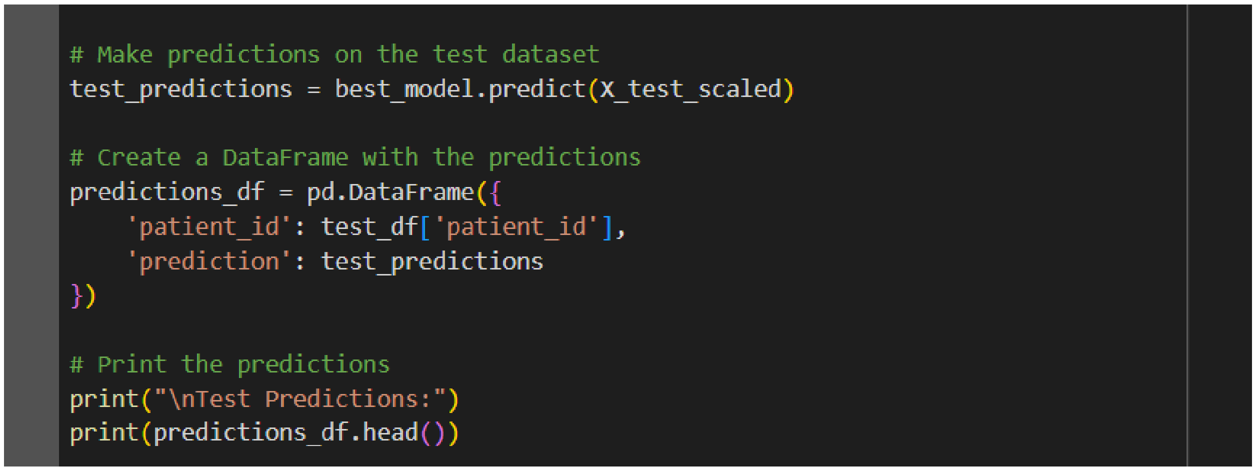
## Prediction:

Loading the test dataset and standardize the test dataset:



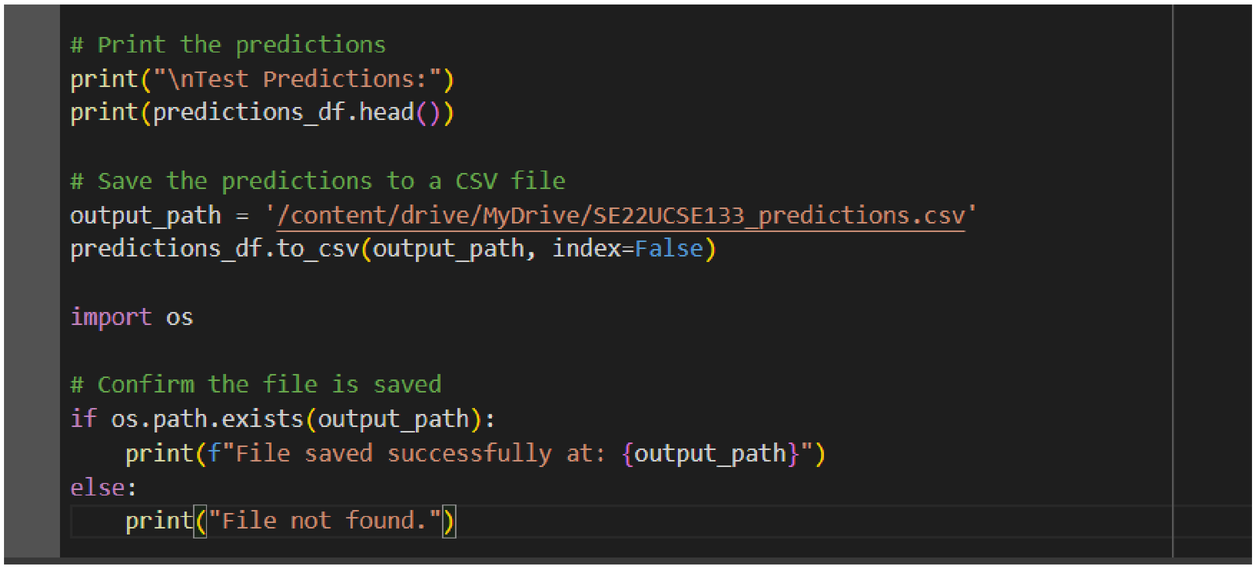
```python
test_df = pd.read_csv("/content/drive/MyDrive/ml project /Disease_test.csv")

X_test = test_df.drop('patient_id', axis=1)

# Standardize the test features
X_test_scaled = scaler.transform(X_test)
```

Generating the predictions on the test dataset:



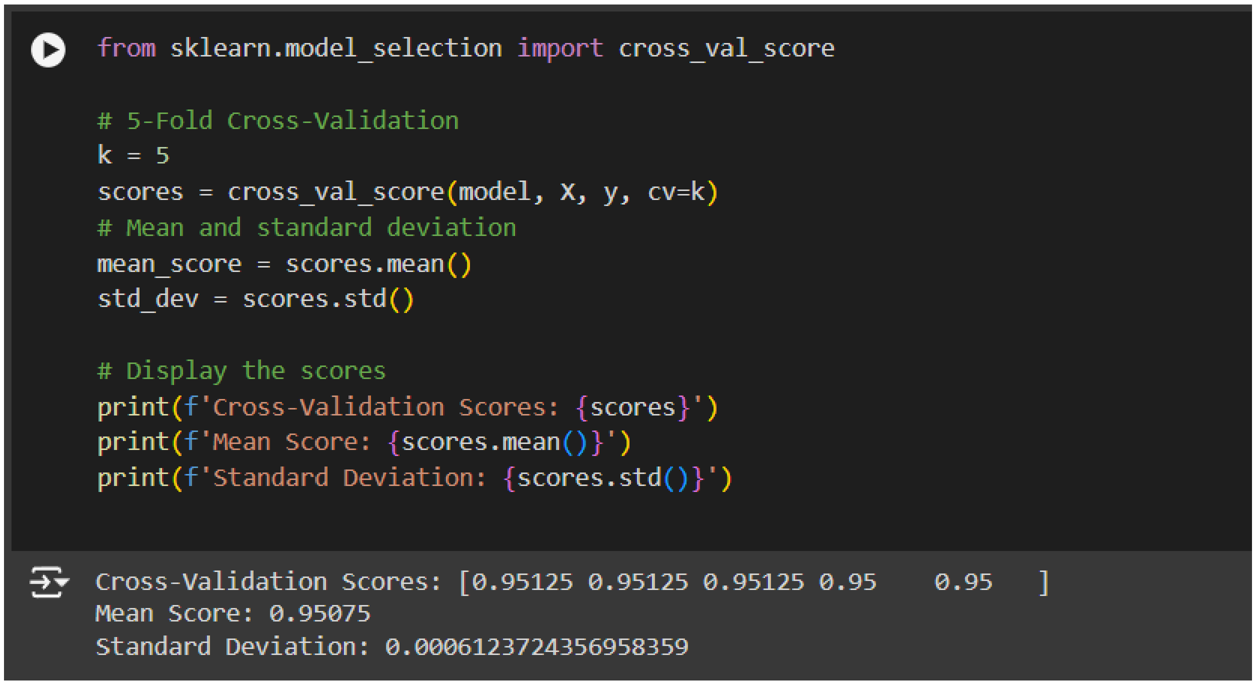
```
# Make predictions on the test dataset
test_predictions = best_model.predict(X_test_scaled)

# Create a DataFrame with the predictions
predictions_df = pd.DataFrame({
    'patient_id': test_df['patient_id'],
    'prediction': test_predictions
})

# Print the predictions
print("\nTest Predictions:")
print(predictions_df.head())
```

Saving the predictions in a CSV file named `student_<ID>_predictions.csv`



```
# Print the predictions
print("\nTest Predictions:")
print(predictions_df.head())

# Save the predictions to a CSV file
output_path = '/content/drive/MyDrive/SE22UCSE133_predictions.csv'
predictions_df.to_csv(output_path, index=False)

import os

# Confirm the file is saved
if os.path.exists(output_path):
    print(f"File saved successfully at: {output_path}")
else:
    print("File not found.")
```

now we perform cross validation to evaluate the scores and acsess the model's performance



```
from sklearn.model_selection import cross_val_score

# 5-Fold Cross-Validation
k = 5
scores = cross_val_score(model, X, y, cv=k)
# Mean and standard deviation
mean_score = scores.mean()
std_dev = scores.std()

# Display the scores
print(f'Cross-Validation Scores: {scores}')
print(f'Mean Score: {scores.mean()}')
print(f'Standard Deviation: {scores.std()}')
```

```
Cross-Validation Scores: [0.95125 0.95125 0.95125 0.95    0.95    ]
Mean Score: 0.95075
Standard Deviation: 0.0006123724356958359
```