Project Report

Predicting Thyroid Cancer Re-occurrence

INDEX

Project Report

Abstract

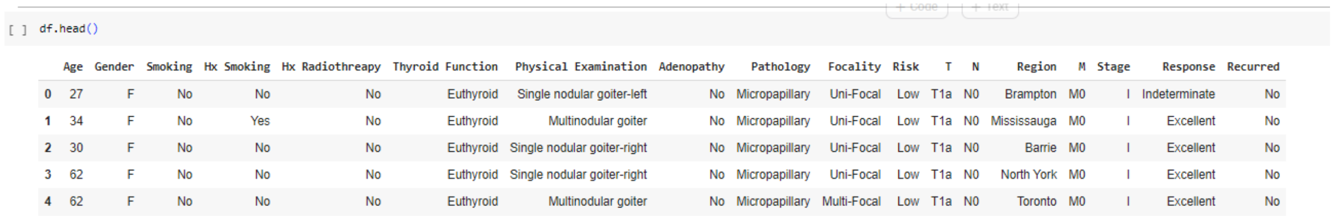
The project aimed to analyze data related to thyroid cancer patients to predict the recurrence of cancer. Various machine learning algorithms were employed for this purpose, including Random Forest Regression, Logistic Regression, and K-Nearest Neighbors Regression. The dataset underwent extensive preprocessing, including handling missing values, removing duplicates, correcting spelling mistakes, and encoding categorical variables. The performance of each model was evaluated using metrics such as Mean Absolute Error, Mean Squared Error, Root Mean Squared Error, and Accuracy.

Introduction:

Thyroid cancer recurrence is a significant concern for patients and clinicians. This project addresses the need for accurate prediction of recurrence using machine learning techniques. The dataset contains various features related to patient demographics, medical history, and tumor characteristics. The objective is to develop models that can effectively predict cancer recurrence based on these features. The methodology involves data preprocessing, model training using multiple algorithms, and performance evaluation.

Data Collection and Preprocessing:

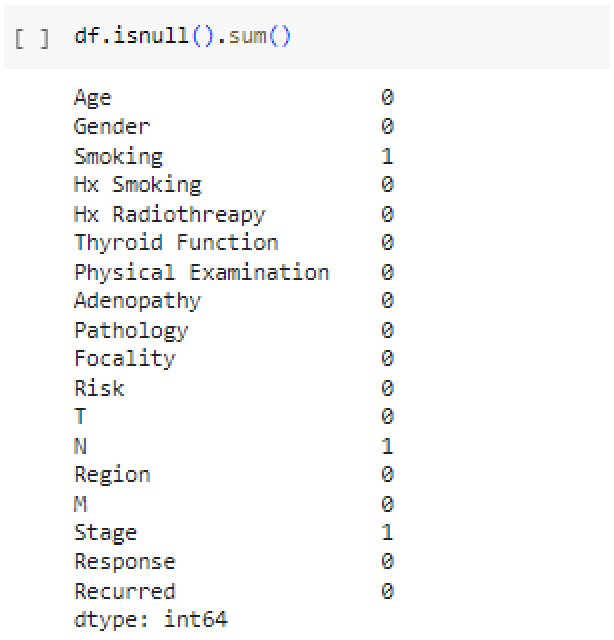
The dataset was collected from a reliable source and initially examined for its structure and missing values. Missing values were replaced with mode values after identifying them for respective columns. Duplicate rows were removed, and spelling mistakes in the 'Pathology' column were corrected. Categorical variables were encoded, and unnecessary columns were dropped.



```
[ ] df.head()
```

	Age	Gender	Smoking	Hx Smoking	Hx Radiothreapy	Thyroid Function	Physical Examination	Adenopathy	Pathology	Focality	Risk	T	N	Region	M	Stage	Response	Recurred
0	27	F	No	No	No	Euthyroid	Single nodular goiter-left	No	Micropapillary	Uni-Focal	Low	T1a	N0	Brampton	M0	I	Indeterminate	No
1	34	F	No	Yes	No	Euthyroid	Multinodular goiter	No	Micropapillary	Uni-Focal	Low	T1a	N0	Mississauga	M0	I	Excellent	No
2	30	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	Barrie	M0	I	Excellent	No
3	62	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	North York	M0	I	Excellent	No
4	62	F	No	No	No	Euthyroid	Multinodular goiter	No	Micropapillary	Multi-Focal	Low	T1a	N0	Toronto	M0	I	Excellent	No

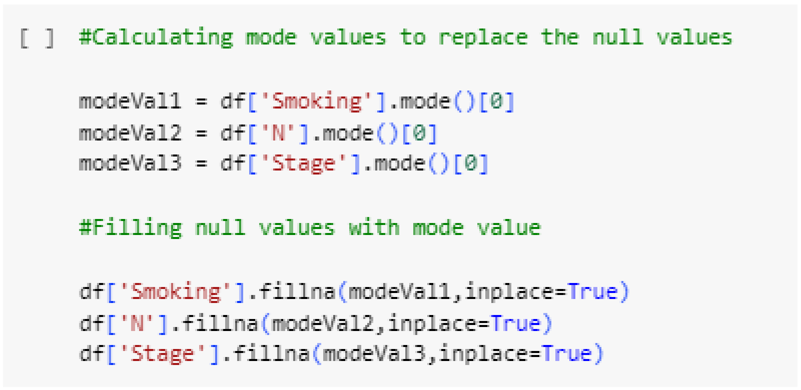
Figure 1 First five records of dataset



```
[ ]  df.isnull().sum()

     Age                    0
     Gender                 0
     Smoking                1
     Hx Smoking             0
     Hx Radiothreapy        0
     Thyroid Function       0
     Physical Examination   0
     Adenopathy             0
     Pathology              0
     Focality               0
     Risk                   0
     T                      0
     N                      1
     Region                 0
     M                      0
     Stage                  1
     Response               0
     Recurred               0
     dtype: int64
```

Figure 2 Missing values in dataset



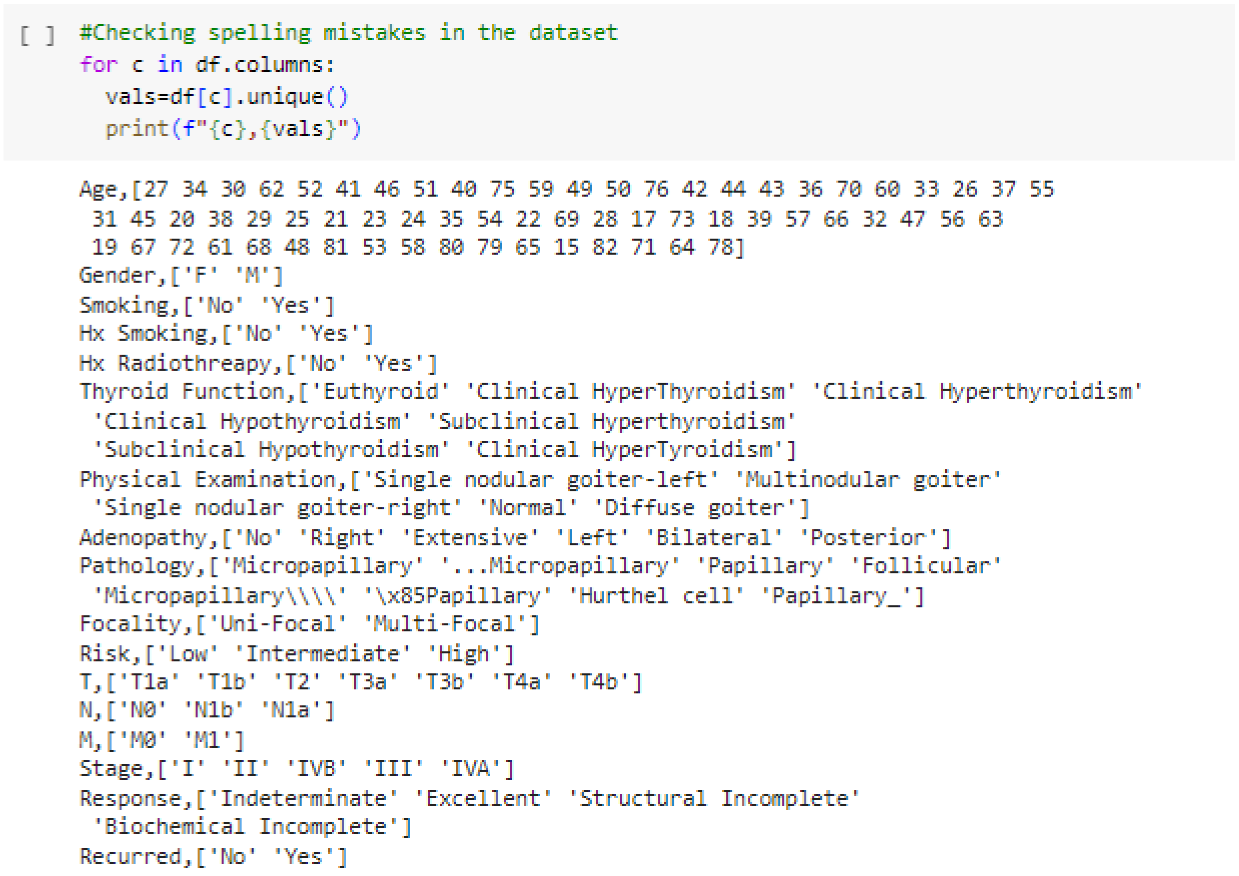
```
[ ]  #Calculating mode values to replace the null values

     modeVal1 = df['Smoking'].mode()[0]
     modeVal2 = df['N'].mode()[0]
     modeVal3 = df['Stage'].mode()[0]

     #Filling null values with mode value

     df['Smoking'].fillna(modeVal1,inplace=True)
     df['N'].fillna(modeVal2,inplace=True)
     df['Stage'].fillna(modeVal3,inplace=True)
```

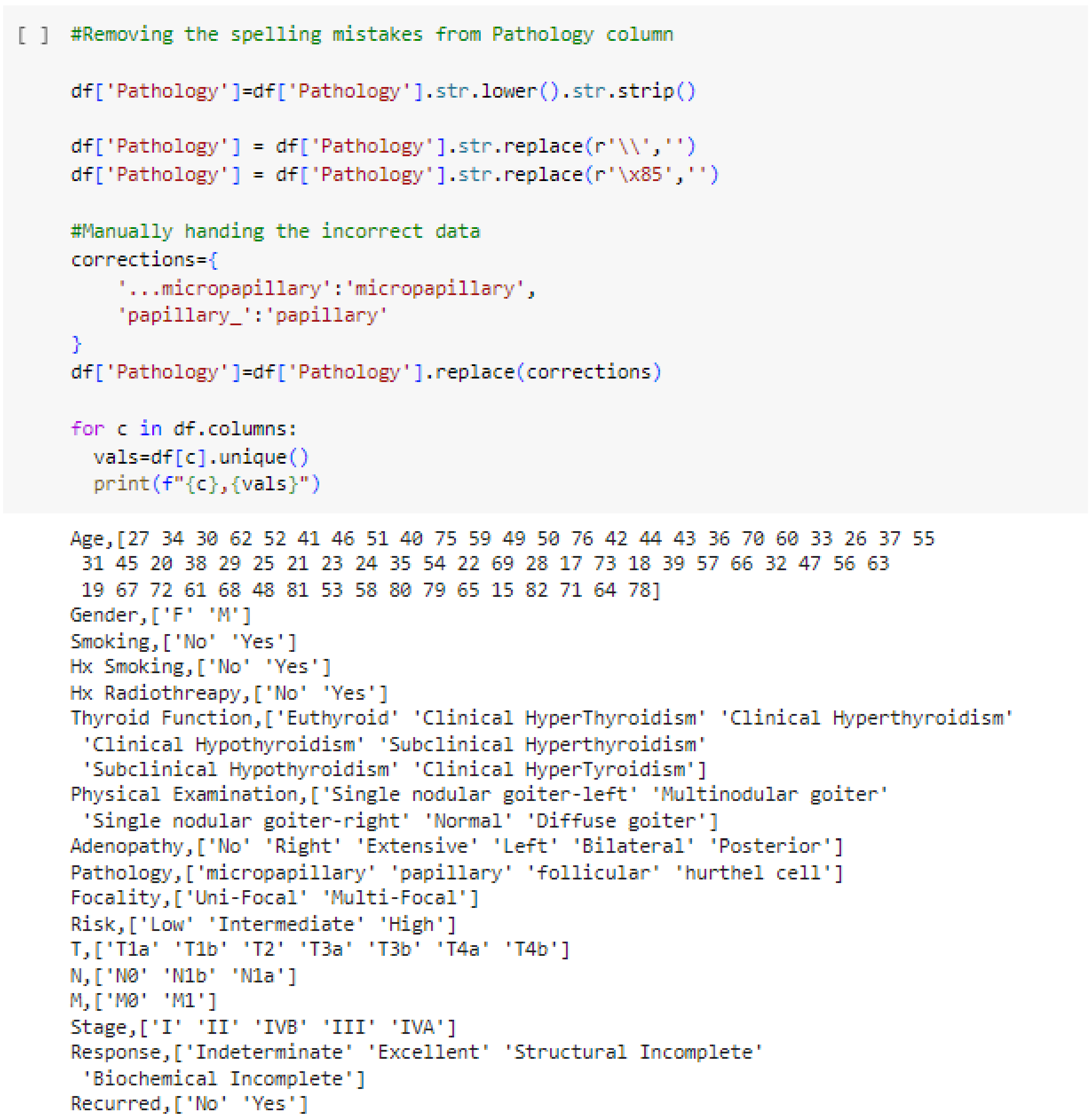
Figure 3 Handling missing values



```
[ ] #Checking spelling mistakes in the dataset
    for c in df.columns:
      vals=df[c].unique()
      print(f"{c},{vals}")
```

```
Age,[27 34 30 62 52 41 46 51 40 75 59 49 50 76 42 44 43 36 70 60 33 26 37 55
 31 45 20 38 29 25 21 23 24 35 54 22 69 28 17 73 18 39 57 66 32 47 56 63
 19 67 72 61 68 48 81 53 58 80 79 65 15 82 71 64 78]
Gender,['F' 'M']
Smoking,['No' 'Yes']
Hx Smoking,['No' 'Yes']
Hx Radiothreapy,['No' 'Yes']
Thyroid Function,['Euthyroid' 'Clinical HyperThyroidism' 'Clinical Hyperthyroidism'
 'Clinical Hypothyroidism' 'Subclinical Hyperthyroidism'
 'Subclinical Hypothyroidism' 'Clinical HyperTyroidism']
Physical Examination,['Single nodular goiter-left' 'Multinodular goiter'
 'Single nodular goiter-right' 'Normal' 'Diffuse goiter']
Adenopathy,['No' 'Right' 'Extensive' 'Left' 'Bilateral' 'Posterior']
Pathology,['Micropapillary' '...Micropapillary' 'Papillary' 'Follicular'
 'Micropapillary\\\\' '\x85Papillary' 'Hurthel cell' 'Papillary_']
Focality,['Uni-Focal' 'Multi-Focal']
Risk,['Low' 'Intermediate' 'High']
T,['T1a' 'T1b' 'T2' 'T3a' 'T3b' 'T4a' 'T4b']
N,['N0' 'N1b' 'N1a']
M,['M0' 'M1']
Stage,['I' 'II' 'IVB' 'III' 'IVA']
Response,['Indeterminate' 'Excellent' 'Structural Incomplete'
 'Biochemical Incomplete']
Recurred,['No' 'Yes']
```

Figure 4 Checking spelling mistakes in dataset



```
[ ] #Removing the spelling mistakes from Pathology column

    df['Pathology']=df['Pathology'].str.lower().str.strip()

    df['Pathology'] = df['Pathology'].str.replace(r'\\','')
    df['Pathology'] = df['Pathology'].str.replace(r'\x85','')

    #Manually handing the incorrect data
    corrections={
        '...micropapillary':'micropapillary',
        'papillary_':'papillary'
    }
    df['Pathology']=df['Pathology'].replace(corrections)

    for c in df.columns:
      vals=df[c].unique()
      print(f"{c},{vals}")
```

```
Age,[27 34 30 62 52 41 46 51 40 75 59 49 50 76 42 44 43 36 70 60 33 26 37 55
 31 45 20 38 29 25 21 23 24 35 54 22 69 28 17 73 18 39 57 66 32 47 56 63
 19 67 72 61 68 48 81 53 58 80 79 65 15 82 71 64 78]
Gender,['F' 'M']
Smoking,['No' 'Yes']
Hx Smoking,['No' 'Yes']
Hx Radiothreapy,['No' 'Yes']
Thyroid Function,['Euthyroid' 'Clinical HyperThyroidism' 'Clinical Hyperthyroidism'
 'Clinical Hypothyroidism' 'Subclinical Hyperthyroidism'
 'Subclinical Hypothyroidism' 'Clinical HyperTyroidism']
Physical Examination,['Single nodular goiter-left' 'Multinodular goiter'
 'Single nodular goiter-right' 'Normal' 'Diffuse goiter']
Adenopathy,['No' 'Right' 'Extensive' 'Left' 'Bilateral' 'Posterior']
Pathology,['micropapillary' 'papillary' 'follicular' 'hurthel cell']
Focality,['Uni-Focal' 'Multi-Focal']
Risk,['Low' 'Intermediate' 'High']
T,['T1a' 'T1b' 'T2' 'T3a' 'T3b' 'T4a' 'T4b']
N,['N0' 'N1b' 'N1a']
M,['M0' 'M1']
Stage,['I' 'II' 'IVB' 'III' 'IVA']
Response,['Indeterminate' 'Excellent' 'Structural Incomplete'
 'Biochemical Incomplete']
Recurred,['No' 'Yes']
```

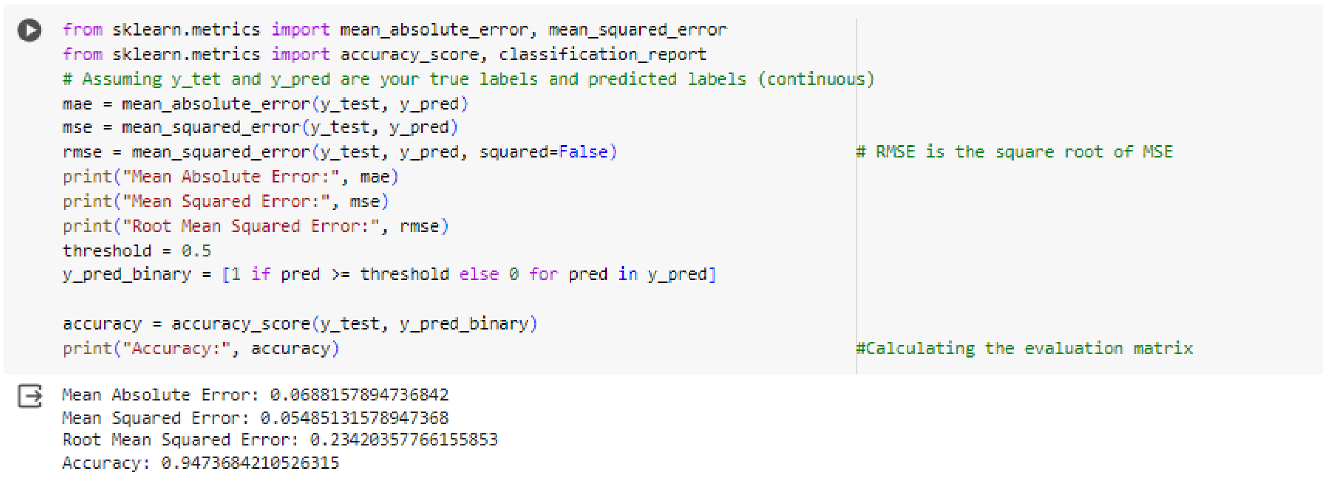
Figure 5 Removing spelling mistake

Methodology:

Machine learning algorithms such as Random Forest Regression, Logistic Regression, and K-Nearest Neighbors Regression were chosen for their suitability for the predictive task. The dataset was split into training and testing sets. Models were trained, validated, and evaluated using appropriate metrics. Parameter tuning was performed to optimize model performance.

Results:

Experimental results showed that Random Forest Regression achieved the highest accuracy among the models evaluated. Mean Absolute Error, Mean Squared Error, and Root Mean Squared Error were calculated for each model to assess prediction accuracy. Visualizations such as scatter plots and heatmaps were used to illustrate key findings and correlations between features.

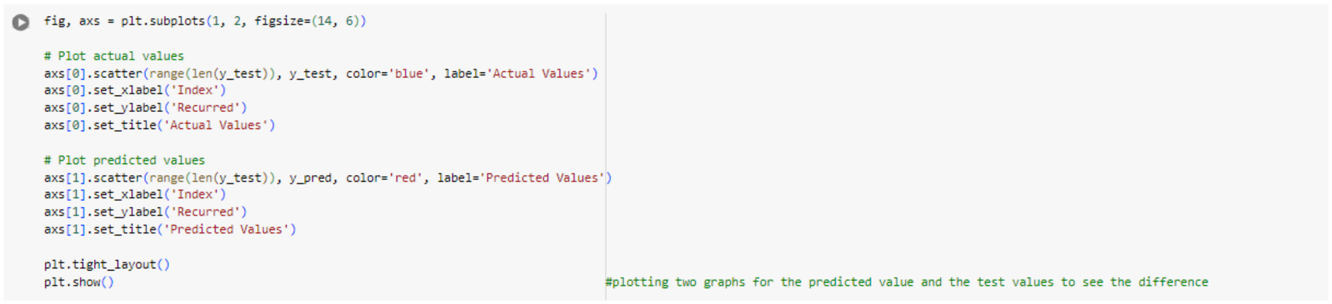


```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import accuracy_score, classification_report
# Assuming y_tet and y_pred are your true labels and predicted labels (continuous)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)                    # RMSE is the square root of MSE
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
threshold = 0.5
y_pred_binary = [1 if pred >= threshold else 0 for pred in y_pred]

accuracy = accuracy_score(y_test, y_pred_binary)
print("Accuracy:", accuracy)                                                #Calculating the evaluation matrix
```

```
Mean Absolute Error: 0.0688157894736842
Mean Squared Error: 0.05485131578947368
Root Mean Squared Error: 0.23420357766155853
Accuracy: 0.9473684210526315
```

Figure 6 Evaluation matrix of Random Forest Model



```python
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plot actual values
axs[0].scatter(range(len(y_test)), y_test, color='blue', label='Actual Values')
axs[0].set_xlabel('Index')
axs[0].set_ylabel('Recurred')
axs[0].set_title('Actual Values')

# Plot predicted values
axs[1].scatter(range(len(y_test)), y_pred, color='red', label='Predicted Values')
axs[1].set_xlabel('Index')
axs[1].set_ylabel('Recurred')
axs[1].set_title('Predicted Values')

plt.tight_layout()
plt.show()                                                   #plotting two graphs for the predicted value and the test values to see the difference
```

Figure 7 Plotting graph for predicted and test values for Random Forest Model

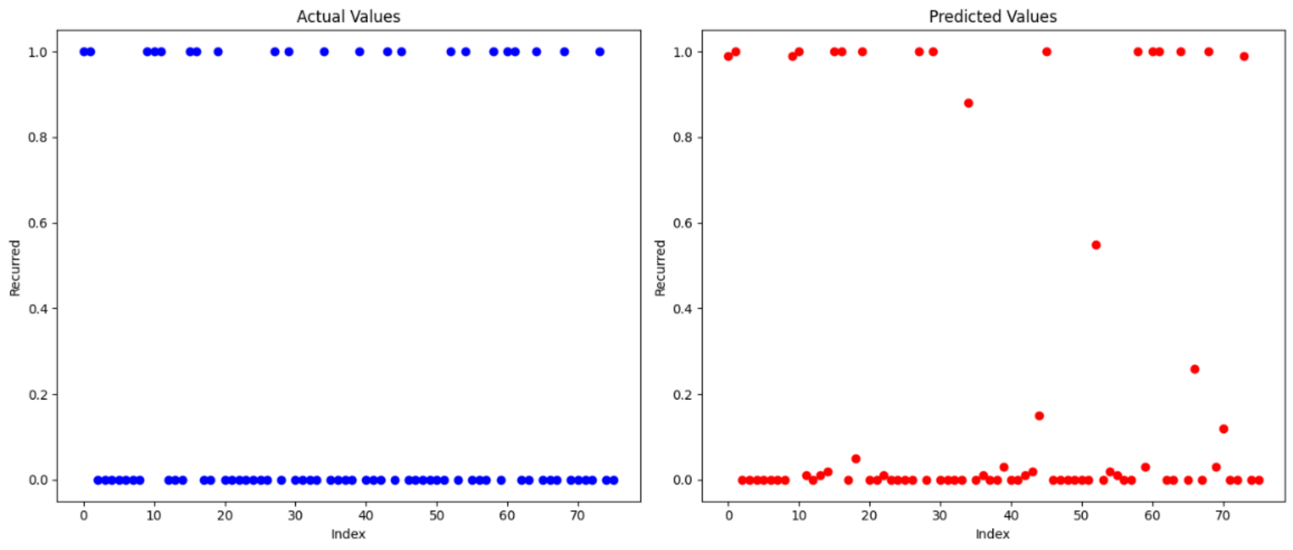
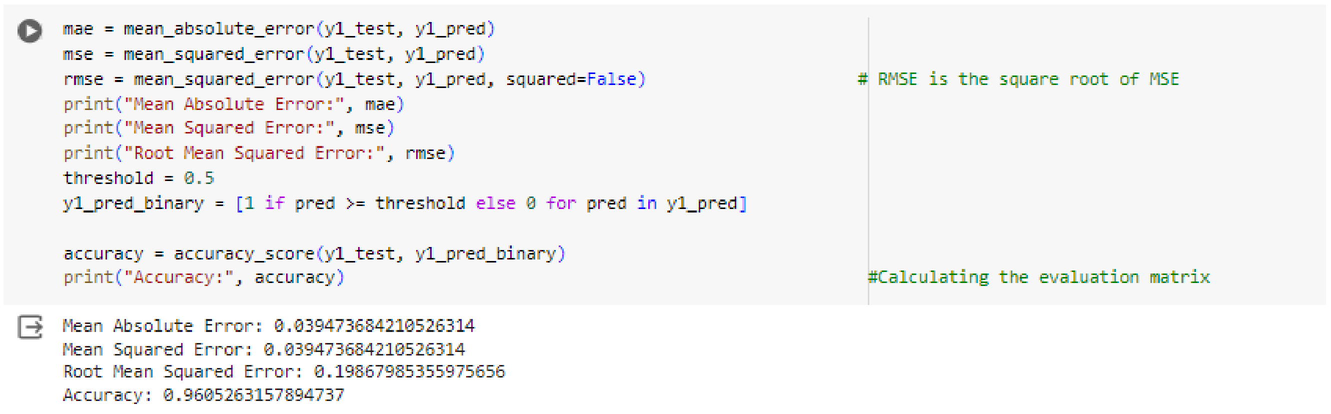


Figure 8 Plotting graph for predicted and test values for Random Forest Model

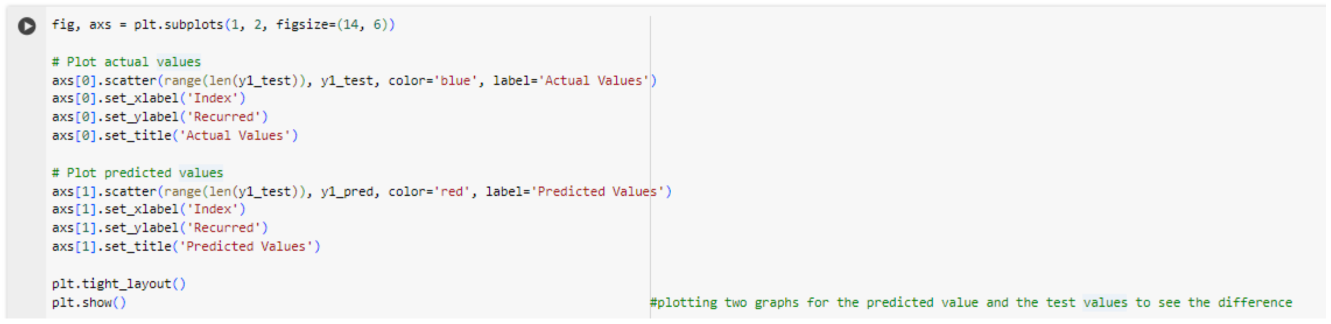


```
mae = mean_absolute_error(y1_test, y1_pred)
mse = mean_squared_error(y1_test, y1_pred)
rmse = mean_squared_error(y1_test, y1_pred, squared=False)                # RMSE is the square root of MSE
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
threshold = 0.5
y1_pred_binary = [1 if pred >= threshold else 0 for pred in y1_pred]

accuracy = accuracy_score(y1_test, y1_pred_binary)
print("Accuracy:", accuracy)                                               #Calculating the evaluation matrix
```

```
Mean Absolute Error: 0.039473684210526314
Mean Squared Error: 0.039473684210526314
Root Mean Squared Error: 0.19867985355975656
Accuracy: 0.9605263157894737
```

Figure 9 Evaluation matrix for Logistic Regression Model



```
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plot actual values
axs[0].scatter(range(len(y1_test)), y1_test, color='blue', label='Actual Values')
axs[0].set_xlabel('Index')
axs[0].set_ylabel('Recurred')
axs[0].set_title('Actual Values')

# Plot predicted values
axs[1].scatter(range(len(y1_test)), y1_pred, color='red', label='Predicted Values')
axs[1].set_xlabel('Index')
axs[1].set_ylabel('Recurred')
axs[1].set_title('Predicted Values')

plt.tight_layout()
plt.show()                                                  #plotting two graphs for the predicted value and the test values to see the difference
```

Figure 10 Plotting graph for predicted and test values for Logistic Regression Model

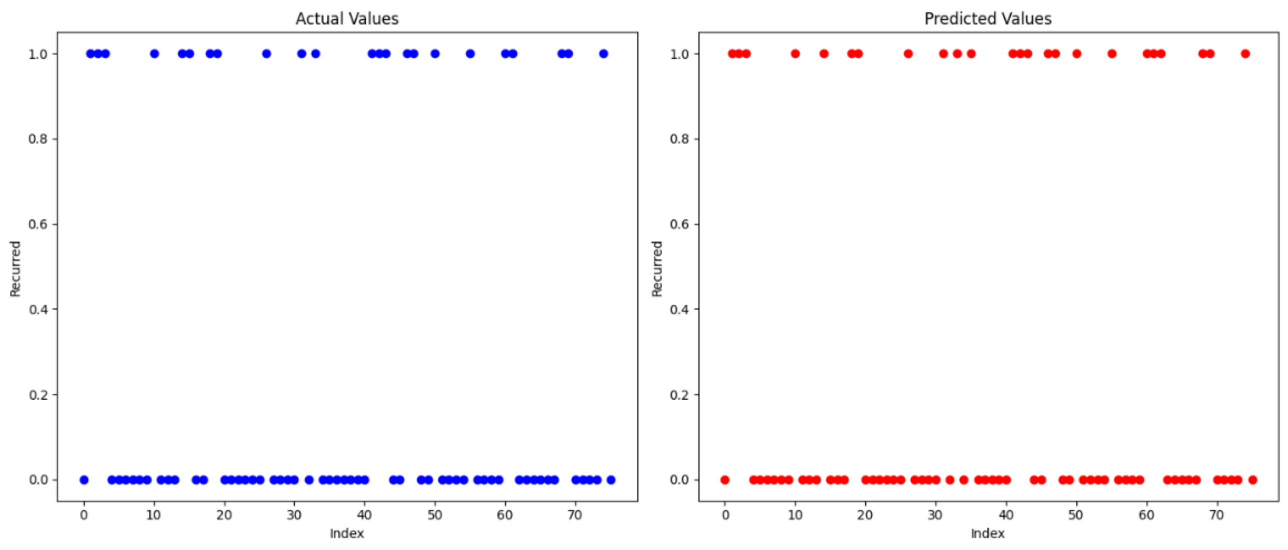
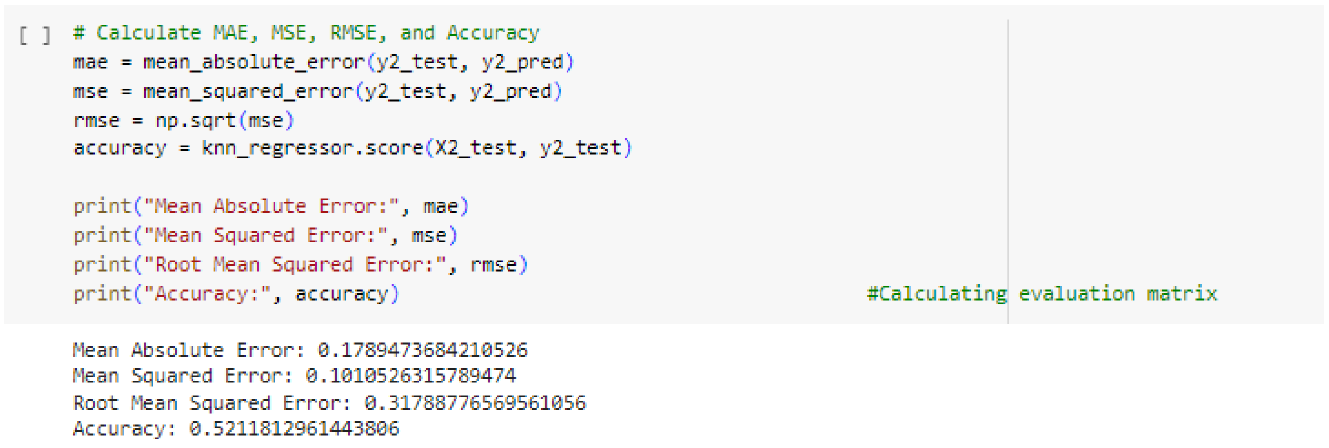


Figure 11 Plotting graph for predicted and test values for Logistic Regression Model

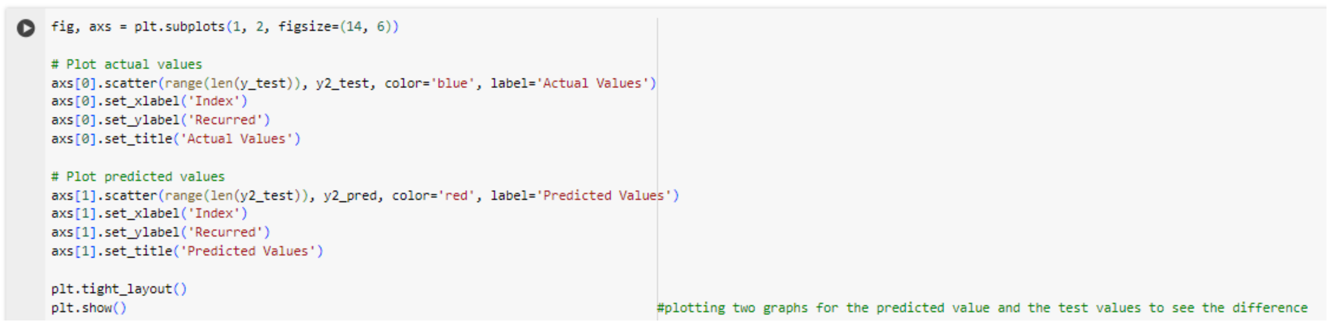


```
[ ] # Calculate MAE, MSE, RMSE, and Accuracy
    mae = mean_absolute_error(y2_test, y2_pred)
    mse = mean_squared_error(y2_test, y2_pred)
    rmse = np.sqrt(mse)
    accuracy = knn_regressor.score(X2_test, y2_test)

    print("Mean Absolute Error:", mae)
    print("Mean Squared Error:", mse)
    print("Root Mean Squared Error:", rmse)
    print("Accuracy:", accuracy)                                  #Calculating evaluation matrix

    Mean Absolute Error: 0.1789473684210526
    Mean Squared Error: 0.1010526315789474
    Root Mean Squared Error: 0.31788776569561056
    Accuracy: 0.5211812961443806
```

Figure 12 Evaluation Matrix for RNN



```
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plot actual values
axs[0].scatter(range(len(y_test)), y2_test, color='blue', label='Actual Values')
axs[0].set_xlabel('Index')
axs[0].set_ylabel('Recurred')
axs[0].set_title('Actual Values')

# Plot predicted values
axs[1].scatter(range(len(y2_test)), y2_pred, color='red', label='Predicted Values')
axs[1].set_xlabel('Index')
axs[1].set_ylabel('Recurred')
axs[1].set_title('Predicted Values')

plt.tight_layout()
plt.show()                                                    #plotting two graphs for the predicted value and the test values to see the difference
```

Figure 13 Plotting graph for predicted and test values for RNN

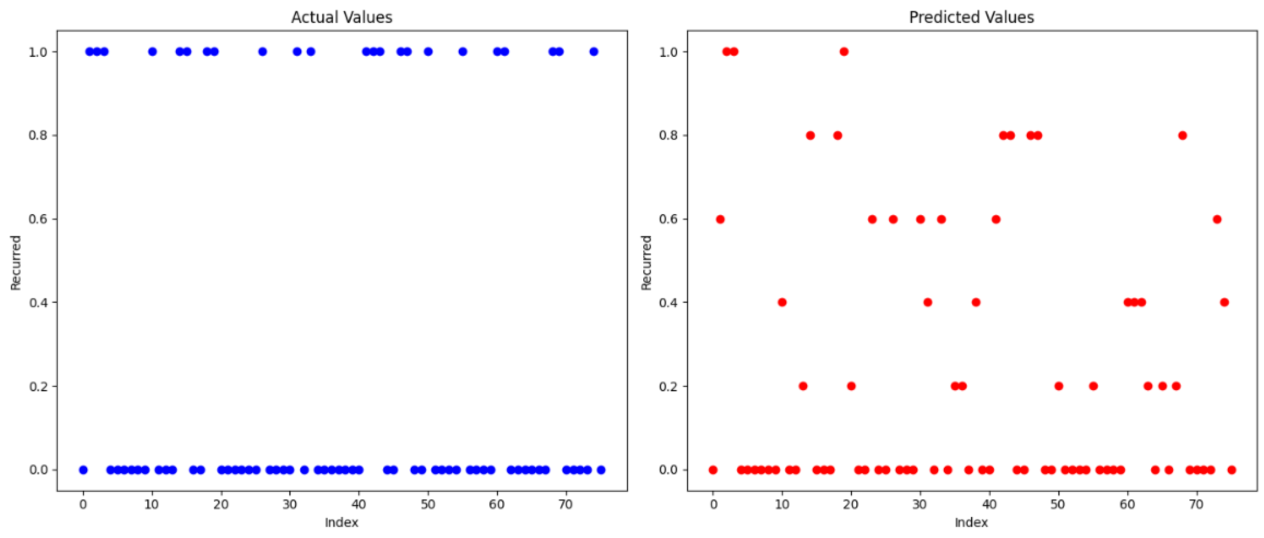


Figure 14 Plotting graph for predicted and test values for RNN

Discussion:

The results indicate that machine learning models can effectively predict thyroid cancer recurrence based on patient data. The strengths and weaknesses of each model were analyzed, and insights were provided into areas for further improvement. A comparison with prior work demonstrated the contribution of the project to existing knowledge in the field.

Conclusion:

In conclusion, the project successfully developed machine learning models for predicting thyroid cancer recurrence. The objectives were achieved, and the models demonstrated promising performance. Recommendations for future work include exploring additional features, optimizing model parameters further, and validating the models on larger datasets.

References:

Sabato, C. (2023, December 11). *How to read CSV files using Pandas: Step-By-Step*. CodeFatherTech. https://codefather.tech/blog/pandas-read-csv/

Sabato, C. (2023b, December 11). *How to read CSV files using Pandas: Step-By-Step*. CodeFatherTech. https://codefather.tech/blog/pandas-read-csv/

Machine Learning Plus. (2022, March 8). *Pandas read_csv() – How to read a csv file in Python*. https://www.machinelearningplus.com/pandas/pandas-read_csv-completed/

Pykes, K. (2023, February 27). *pandas read csv() Tutorial: Importing Data*. https://www.datacamp.com/tutorial/pandas-read-csv

Nik. (2023, April 5). *Pandas read_csv() – Read CSV and Delimited Files in Pandas*. Datagy. https://datagy.io/pandas-read_csv/

Appendices:

Git-hub Repository:

https://github.com/sourav1920/Thyroid-Cancer-Model

PY File:

Thyroid Cancer Prediction.ipynb