ML Code with Feature Selection – Explanation

1.Data Preprocessing before Feature selection

1.1 Removing Null values.

As we don't have any null values in the dataset Moving to next.

```
# First we need to check do we have any null values in the dataset
dataset.isnull().sum()
```

```
Age                   0
Gender                0
Country               0
Family_History        0
Radiation_Exposure    0
Iodine_Deficiency     0
Smoking               0
Obesity               0
Diabetes              0
TSH_Level             0
T3_Level              0
T4_Level              0
Nodule_Size           0
Thyroid_Cancer_Risk   0
Diagnosis             0
dtype: int64
```

1.2 Minimizing dataset for research

As the raw dataset has 2,12,691 rows , the calculations take too much time . So here we are minimizing the dataset to 30% from its original by selecting random rows.

Once we are done with feature selection we can train the model by entire dataset.

```
sample_df = dataset.sample(frac=0.3, random_state=42)
```

```
dataset = sample_df
dataset
```

	Age	Gender	Country	Family_History	Radiation_Exposure	Iodine_Deficiency	Smoking	Obesity	Diabetes	TSH_Level	T3_Level	T4_Level
82562	81	Male	Russia	No	No	Yes	No	Yes	No	0.81	0.72	4.57
101549	19	Female	India	Yes	No	No	Yes	No	No	9.90	2.38	6.17
97401	44	Male	Brazil	Yes	No	No	Yes	No	No	0.96	0.95	7.92
105415	56	Female	Nigeria	No	No	No	No	No	No	5.49	0.74	8.39
152387	86	Male	Nigeria	Yes	No	No	No	No	No	7.28	3.04	10.79
...	...	...	...	...	...	...	...	...	...	...	...	...
58464	70	Female	China	No	No	Yes	No	No	No	1.94	2.11	4.76
148788	34	Male	Russia	Yes	No	Yes	No	No	No	5.91	1.88	10.19
183932	83	Male	Germany	Yes	No	No	No	No	No	6.72	1.80	10.62
91945	56	Female	Nigeria	Yes	No	No	No	Yes	No	9.43	0.75	5.82
166584	72	Female	USA	No	No	No	No	No	No	6.29	0.59	10.73

63807 rows × 15 columns

Rows minimized to 63,807.

1.3 Encoding categorical columns to Numerical columns

I. Label encoding

Needed as Thyroid_Cancer_Risk column has Low, Medium, High values.

```python
# Label encoding for ordinal values
# Thyroid_Cancer_Risk - ordinal values changed into integers

risk_map = {'Low': 0, 'Medium': 1, 'High': 2}
dataset['Thyroid_Cancer_Risk'] = dataset['Thyroid_Cancer_Risk'].map(risk_map)
dataset
```

ountry	Family_History	Radiation_Exposure	Iodine_Deficiency	Smoking	Obesity	Diabetes	TSH_Level	T3_Level	T4_Level	Nodule_Size	Thyroid_Cancer_Risk	Diagnosis
Russia	No	No	Yes	No	Yes	No	0.81	0.72	4.57	0.72	0	Malignant
India	Yes	No	No	Yes	No	No	9.90	2.38	6.17	0.70	2	Benign
Brazil	Yes	No	No	Yes	No	No	0.96	0.95	7.92	3.79	1	Benign
Nigeria	No	No	No	No	No	No	5.49	0.74	8.39	4.75	0	Malignant
Nigeria	Yes	No	No	No	No	No	7.28	3.04	10.79	4.57	1	Benign
...	...	...	...	...	...	...	...	...	...	...	...	...
China	No	No	Yes	No	No	No	1.94	2.11	4.76	4.34	0	Benign
Russia	Yes	No	Yes	No	No	No	5.91	1.88	10.19	3.39	0	Benign
ermany	Yes	No	No	No	No	No	6.72	1.80	10.62	1.32	0	Benign
Nigeria	Yes	No	No	No	Yes	No	9.43	0.75	5.82	4.87	0	Benign
USA	No	No	No	No	No	No	6.29	0.59	10.73	4.47	1	Benign

II. One-Hot Encoding:

This will change the categorical values to binary values in the dataset.

```python
# One-Hot Encoding for categorical values to binary values
# All the categorical values will be changed to binary values in the dataset

dataset=pd.get_dummies(dataset,drop_first=True,dtype=int)
dataset
```

	Age	TSH_Level	T3_Level	T4_Level	Nodule_Size	Thyroid_Cancer_Risk	Gender_Male	Country_China	Country_Germany	Country_India	...	Country_South Korea	Co
82562	81	0.81	0.72	4.57	0.72	0	1	0	0	0	...	0	
101549	19	9.90	2.38	6.17	0.70	2	0	0	0	1	...	0	
97401	44	0.96	0.95	7.92	3.79	1	1	0	0	0	...	0	
105415	56	5.49	0.74	8.39	4.75	0	0	0	0	0	...	0	
152387	86	7.28	3.04	10.79	4.57	1	1	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
58464	70	1.94	2.11	4.76	4.34	0	0	1	0	0	...	0	
148788	34	5.91	1.88	10.19	3.39	0	1	0	0	0	...	0	
183932	83	6.72	1.80	10.62	1.32	0	1	0	1	0	...	0	
91945	56	9.43	0.75	5.82	4.87	0	0	0	0	0	...	0	
166584	72	6.29	0.59	10.73	4.47	1	0	0	0	0	...	0	

63807 rows × 23 columns

1.4 Saving the Preprocessed data to separate csv file.

```python
dataset.to_csv("thyroid_cancer_risk_final.csv", index=False)
```

Feature Selection

Trying with SelectKBest

```
def selectkbest(indep_X,dep_Y,n):
        test = SelectKBest(score_func=chi2, k=n)
        fit1= test.fit(indep_X,dep_Y)
        selectk_features = fit1.transform(indep_X)
        return selectk_features

def split_scalar(indep_X,dep_Y):
        X_train, X_test, y_train, y_test = train_test_split(indep_X, dep_Y, test_size = 0.25, random_state = 0)
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
        return X_train, X_test, y_train, y_test

def cm_prediction(classifier,X_test):
     y_pred = classifier.predict(X_test)

        # Making the Confusion Matrix
     from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_pred)

     from sklearn.metrics import accuracy_score
     from sklearn.metrics import classification_report

     Accuracy=accuracy_score(y_test, y_pred )

     report=classification_report(y_test, y_pred)
     return  classifier,Accuracy,report,X_test,y_test,cm
```

2.1.Created function for SelectKBest

- SelectKBest(score_func=chi2, k=n) will pick the **top n features** using the **Chi-Square test**.
- fit1.fit(indep_X,dep_Y) will learn which features are important.
- fit1.transform(indep_X) will keep only the best n features then return it.

2.2. Created a function for Split Scalar

- train_test_split(...) will Split the data into training (75%) and testing (25%).
- StandardScaler() - scales features so they are on the same range.
- fit_transform on training data - learn scaling values (mean, std) and apply.
- transform on test data - apply the same scaling.

2.3. Created a function for Prediction using confusion matrix

- Predict(X_test) - The model predict using test data.
- Confusion_matrix (y_test, y_pred) will create the confusion matrix
- Accuracy – Correct predictions saved in accuracy
- **Classification_report** – detailed report created for the predictions.

2.4. Created separate functions for below Algorithms

- Logistic Regression, Support Vector Machine (SVM) Linear Algorithm, SVM Non-Linear Algorithm, Naive Bayes, K-Nearest Neighbors, Decision Tree and Random Forest.
- It will create the instance of classifier and Trains the model with training data
- Then it will call prediction function and make prediction using test data and returns accuracy, report and confusion matrix.

2.5. Created a function for SelectK

- It accepts inputs that hold accuracy values for different classifiers
- pd.DataFrame(index=['ChiSquare'], columns=['Logistic…] - Creates an empty dataframe with one row index - 'ChiSquare' & several col
- enumerate(dataframe.index): - is ['ChiSquare'] --- so this loop runs once

- enumerate(...) gives two values each iteration:
    - number → the integer position (0 for the first row)
    - index → the actual index label (the string 'ChiSquare')
- dataframe['Logistic'][index] = acclog[number] - Sets the cell at row 'ChiSquare' and column 'Logistic' to the value acclog[0].
- Acclog[0] – accuracy value for logistic is set to chiSquare row and logistic col of the dataframe.
- Like above all the accuracy values will be placed to specific columns.

2.6Assigning Input and Output Values

```
dataset1=pd.read_csv("thyroid_cancer_risk_final.csv",index_col=None)

df2=dataset1

df2 = pd.get_dummies(df2, drop_first=True)

indep_X=df2.drop('Diagnosis_Malignant', 1)
dep_Y=df2['Diagnosis_Malignant']
```

Input – All rows in df2 except Diagnosis_Malignant

Output – Diagnosis_Malignant

2.7.Calling a function selectkbest

```
kbest=selectkbest(indep_X,dep_Y,4)


acclog=[]
accsvml=[]
accsvmnl=[]
accknn=[]
accnav=[]
accdes=[]
accrf=[]
```

```
kbest
```

- Parameters are input ,output and n value (4 most important features)
- Empty Python lists created to store accuracy values for different classifiers

2.8.Training and Evaluation

```
X_train, X_test, y_train, y_test=split_scalar(kbest,dep_Y)


classifier,Accuracy,report,X_test,y_test,cm=logistic(X_train,y_train,X_test)
acclog.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=svm_linear(X_train,y_train,X_test)
accsvml.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=svm_NL(X_train,y_train,X_test)
accsvmnl.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=knn(X_train,y_train,X_test)
accknn.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=Navie(X_train,y_train,X_test)
accnav.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=Decision(X_train,y_train,X_test)
accdes.append(Accuracy)

classifier,Accuracy,report,X_test,y_test,cm=random(X_train,y_train,X_test)
accrf.append(Accuracy)

result=selectk_Classification(acclog,accsvml,accsvmnl,accknn,accnav,accdes,accrf)
```

- X_train, X_test, y_train, y_test = split_scalar(kbest, dep_Y) – split the dataset to training and test – kbest(input var selected features)
- classifier, Accuracy, report, X_test, y_test, cm = logistic(X_train, y_train, X_test)
- acclog.append(Accuracy)
    - Trains a **Logistic Regression** model.
    - Returns:
    - classifier → trained model
    - Accuracy → test accuracy
    - report → classification report
    - cm → confusion matrix

- Appends the accuracy to the list acclog.
- Like above all the model will be trained and tested and the accuracy value is appended to the list.
- result = selectk_Classification(acclog, accsvml, accsvmnl, accknn, accnav, accdes, accrf) – will create the dataframe using the accuracy values .

2.9.Analyzing Results

```
result
#6
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.824661	0.765296	0.824661	0.799085	0.809052	0.824661	0.824661

```
result
#5
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.824661	0.765296	0.824661	0.794634	0.814067	0.824661	0.824661

```
result
#4
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
ChiSquare	0.820649	0.765296	0.824661	0.793255	0.817139	0.824661	0.824661

Summary :

- Model accuracy remains consistent (~0.82) as the number of selected features increases from 4 to 6.
- Logistic Regression, Decision Tree, and Random Forest show the best and most stable performance across all feature sets.
- KNN and Naive Bayes exhibit slightly lower accuracy.

Feature Importance (Tree based models)

As we have the accuracy 82% from selectKBest, we are trying with one more feature selection method to check whether we can achieve more accuracy.

1. **Creating the function for Feature Importance**

```python
def featureImportanceFeature(indep_X, dep_Y, n):
        filist = []

        # Define tree-based models
        RF = RandomForestClassifier(n_estimators=100, random_state=42)
        DT = DecisionTreeClassifier(random_state=42)
        GB = GradientBoostingClassifier(n_estimators=100, random_state=42)
        ET = ExtraTreesClassifier(n_estimators=100, random_state=42)

        fimodellist = [RF, DT, GB, ET]

        for model in fimodellist:
            print(model)
            model.fit(indep_X, dep_Y)

            # Get feature importances
            importances = model.feature_importances_

            # Sort and select top n features
            importance_df = pd.DataFrame({
                'Feature': indep_X.columns,
                'Importance': importances
            }).sort_values(by='Importance', ascending=False)

            top_features = importance_df['Feature'].head(n).tolist()
            print(f"Top {n} features for {type(model).__name__}: {top_features}")

            # Transform dataset with selected features
            fi_features = indep_X[top_features].values
            filist.append(fi_features)

        return filist
```

- ➢ Created empty list – filist[]
- ➢ Defined the tree based models
- ➢ fimodellist = [RF, DT, GB, ET] - Stores all the models in a list
- ➢ In For loop – loops through each model in the list print its name.
- ➢ model.fit(indep_X, dep_Y) - Train the model on full dataset
- ➢ importances = model.feature_importances_ - built-in property which will give numerical score for every feature
- ➢ Created a DataFrame showing each feature and its importance score.
- ➢ Sorts the features in descending order, so the most important features come first.
- ➢ top_features = importance_df['Feature'].head(n).tolist() - Selecting the top n features based on their importance scores and storing it to a list
- ➢ fi_features = indep_X[top_features].values – we are storing the dataset containing only its most important features.-- .values→ get the values of those features .
- ➢ Appending it to the list

2. **Created other functions** – split scalar, cm_predictions and all other algorithms.

3. Created function for summarising accuracy results

```
def fi_classification(acclog, accsvml, accsvmnl, accknn, accnav, accdes, accrf):

    fidataframe = pd.DataFrame(index=['RandomForest','DecisionTree','GradientBoosting','ExtraTrees'],
                               columns=['Logistic','SVMl','SVMnl','KNN','Navie','Decision','Random'])

    for number, idex in enumerate(fidataframe.index):
        fidataframe['Logistic'][idex] = acclog[number]
        fidataframe['SVMl'][idex] = accsvml[number]
        fidataframe['SVMnl'][idex] = accsvmnl[number]
        fidataframe['KNN'][idex] = accknn[number]
        fidataframe['Navie'][idex] = accnav[number]
        fidataframe['Decision'][idex] = accdes[number]
        fidataframe['Random'][idex] = accrf[number]

    return fidataframe
```

- This function collects accuracy scores from different classifiers when using feature importance-based feature selection.
- Creates an empty **Pandas DataFrame** with:
- Rows =RandomForest, DecisionTree, GradientBoosting, ExtraTrees
- Columns = Logistic, SVMl, SVMnl, KNN, Navie, Decision, Random
- This DataFrame will store accuracy values for each combination.
- For loop – here iteration occurs for each tree based model(row)
- Dataframe will be populated with all the accuracy values at the end.

4. Assigning Input and Output Values

```
dataset1=pd.read_csv("thyroid_cancer_risk_final.csv",index_col=None)

df2=dataset1

df2 = pd.get_dummies(df2, drop_first=True)

indep_X=df2.drop('Diagnosis_Malignant', 1)
dep_Y=df2['Diagnosis_Malignant']
```

Input – All rows in df2 except Diagnosis_Malignant
Output – Diagnosis_Malignant

5. Calling featureImportanceFeature function

```
filist = featureImportanceFeature(indep_X, dep_Y, 5 )

acclog=[]
accsvml=[]
accsvmnl=[]
accknn=[]
accnav=[]
accdes=[]
accrf=[]
```

```
RandomForestClassifier(random_state=42)
Top 5 features for RandomForestClassifier: ['Thyroid_Cancer_Risk', 'TSH_Level', 'T4_Level', 'Nodule_Size', 'T3_Level']
DecisionTreeClassifier(random_state=42)
Top 5 features for DecisionTreeClassifier: ['Thyroid_Cancer_Risk', 'TSH_Level', 'T4_Level', 'Nodule_Size', 'T3_Level']
GradientBoostingClassifier(random_state=42)
Top 5 features for GradientBoostingClassifier: ['Thyroid_Cancer_Risk', 'T4_Level', 'Nodule_Size', 'T3_Level', 'TSH_Level']
ExtraTreesClassifier(random_state=42)
Top 5 features for ExtraTreesClassifier: ['Thyroid_Cancer_Risk', 'TSH_Level', 'T4_Level', 'Nodule_Size', 'T3_Level']
```

- Parameters are input ,output and n value (selects the top n features to be selected)
- Empty Python lists created to store accuracy values for different classifiers
- Output of the code shown above - five features which are selected displayed

6. Model Evaluation

```
for i in filist:
    X_train, X_test, y_train, y_test = split_scalar(i, dep_Y)

    classifier,Accuracy,report,X_test,y_test,cm = logistic(X_train,y_train,X_test)
    acclog.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm = svm_linear(X_train,y_train,X_test)
    accsvml.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm = svm_NL(X_train,y_train,X_test)
    accsvmnl.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm = knn(X_train,y_train,X_test)
    accknn.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm = Navie(X_train,y_train,X_test)
    accnav.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm = Decision(X_train,y_train,X_test)
    accdes.append(Accuracy)

    classifier,Accuracy,report,X_test,y_test,cm = random(X_train,y_train,X_test)
    accrf.append(Accuracy)

fi_result = fi_classification(acclog, accsvml, accsvmnl, accknn, accnav, accdes, accrf)
```

- For each dataset in filist (which contains top features chosen by different tree-based models), split it into training and testing sets.
- Train multiple classifiers and measure accuracy
- Save each classifier's accuracy in separate lists
- Use fi_classification() to combine all accuracies into a DataFrame

7. Results

```
fi_result
#5
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
RandomForest	0.824661	0.765296	0.824661	0.798458	0.824661	0.715898	0.806921
DecisionTree	0.824661	0.765296	0.824661	0.798458	0.824661	0.715898	0.806921
GradientBoosting	0.824661	0.765296	0.824661	0.798458	0.824661	0.716462	0.806795
ExtraTrees	0.824661	0.765296	0.824661	0.798458	0.824661	0.715898	0.806921

```
fi_result
#3
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
RandomForest	0.824661	0.765296	0.824661	0.796891	0.824661	0.708312	0.780404
DecisionTree	0.824661	0.765296	0.824661	0.796891	0.824661	0.708312	0.780404
GradientBoosting	0.824661	0.765296	0.824661	0.800527	0.824661	0.712324	0.778586
ExtraTrees	0.824661	0.765296	0.824661	0.796891	0.824661	0.708312	0.780404

```
fi_result
#7
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
RandomForest	0.824661	0.765296	0.824661	0.797894	0.824661	0.709566	0.805354
DecisionTree	0.824661	0.765296	0.824661	0.797204	0.824661	0.716462	0.808739
GradientBoosting	0.824661	0.765296	0.824661	0.797831	0.824661	0.712889	0.807046
ExtraTrees	0.824661	0.765296	0.824661	0.797894	0.824661	0.709566	0.805354

Summary:

- Model performance remains steady (~0.82 accuracy) across all tree-based feature selection techniques (Random Forest, Decision Tree, Gradient Boosting, Extra Trees).
- Logistic Regression , Navie and SVM models maintain consistent and strong performance regardless of the number of selected features.
- Decision Tree shows slightly lower accuracy compared to other models.

Conclusion:

"Both SelectKBest (Chi-Square) and Model-Based Feature Importance methods resulted in comparable model accuracies (~82%).
This indicates that the important predictive features are consistent across methods.
Ensemble-based models like Random Forest and Gradient Boosting provided slightly better stability and accuracy, making them suitable for final model selection."

Choosing Random Forest.

After feature selection, the top 5 most significant predictors were identified as:
Thyroid_Cancer_Risk, TSH_Level, T4_Level, Nodule_Size, and T3_Level.