SIG788

Engineering AI solutions

Pass & Credit Task 1

Arunkumar Balaraman
S223919051

Contents

Target = Credit

Predictive Analysis of Chronic Kidney Disease Using Machine Learning Models

Introduction

The dataset pertains to the initial phase of **chronic kidney disease (CKD)** among patients in India. CKD is a severe health condition marked by a steady decline in kidney function over time. *Early detection of CKD is crucial* for effective management and slowing its progression. The dataset encompasses a variety of clinical and demographic factors that could impact CKD diagnosis, including age, blood pressure, sugar levels, red blood cell counts, among others. This makes it an all-encompassing source for constructing a predictive model.

Objective

The goal of creating a model with this dataset is to utilize machine learning methods to precisely forecast the occurrence of chronic kidney disease in patients. This predictive modelling can significantly contribute to early detection and treatment planning for individuals susceptible to CKD, thereby enhancing patient outcomes and alleviating the strain on healthcare systems.

By evaluating the performance of various machine learning models, such as **Decision Trees** and **Random Forests**, we attempt to pinpoint the most efficient method for diagnosing CKD based on the provided clinical and demographic data.

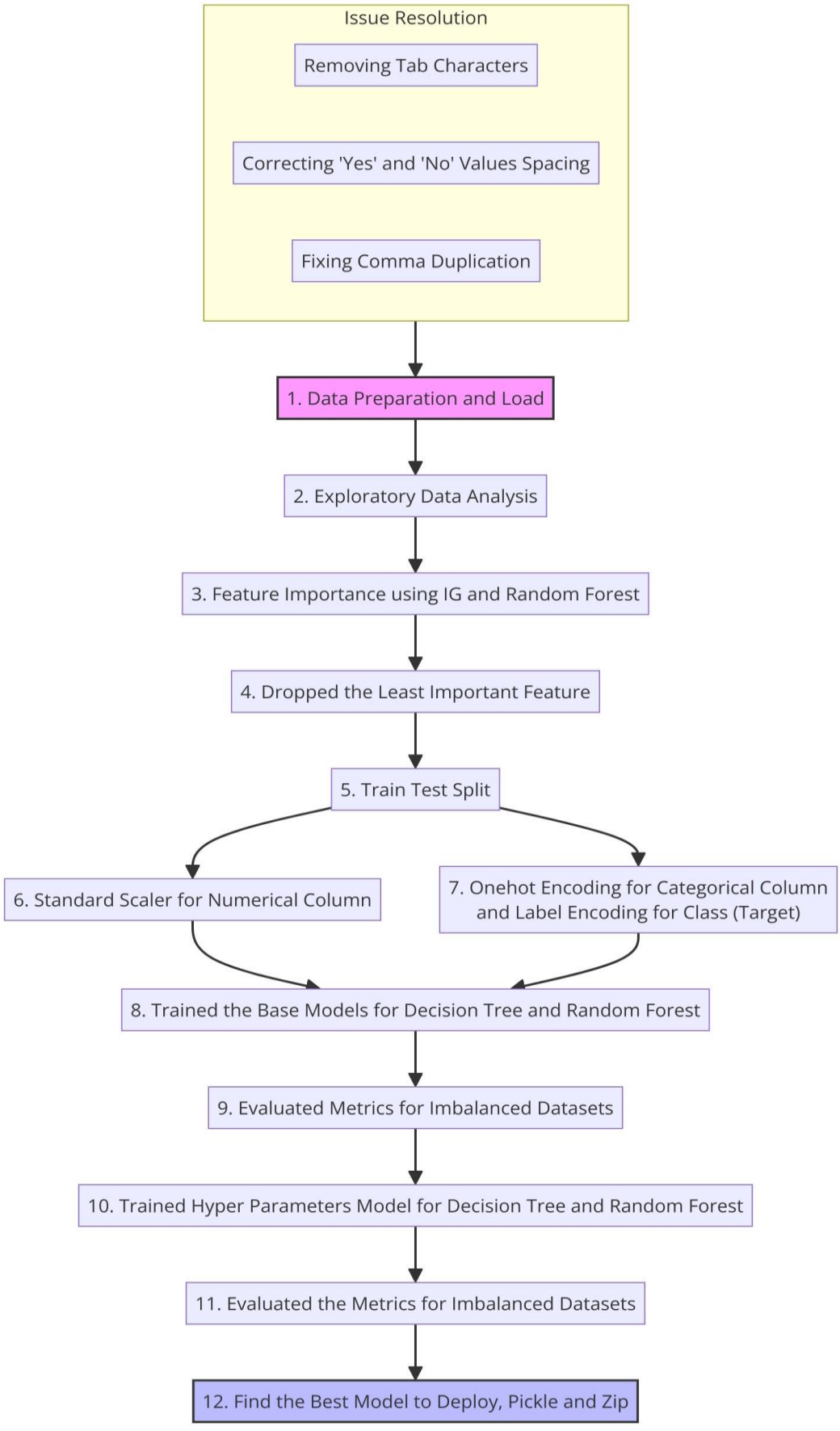
Approach:

The ML pipeline includes the following steps:

- Data Preparation and Loading
- Exploratory Data Analysis
- Feature Importance Evaluation using Information Gain and Random Forest
- Dropping the Least Important Feature
- Train-Test Split
- Standard Scaling for Numerical Columns
- One-Hot Encoding for Categorical Columns and Label Encoding for the Target
- Training Base Models for Decision Tree and Random Forest
- Evaluating Metrics for Imbalanced Datasets using F1 Score, ROC-AUC, Harmonic Mean, Geometric Mean
- Training Hyperparameter Models for Decision Tree and Random Forest
- Evaluating Metrics for Imbalanced Datasets using F1 Score, ROC-AUC, Harmonic Mean, Geometric Mean
- Identifying the Best Model for Deployment, Pickling, and Zipping

During the initial data loading, several issues were encountered and resolved by **removing tab characters**, correcting **spacing around "yes" and "no"** values, and **fixing duplicated commas**. The cleaned dataset, stored in a new file named modified_chronic_kidney_disease.arff.

ML Pipeline Flowchart



Issue Resolution in the data file

During the initial data loading, several issues were encountered and resolved by **removing tab characters**, correcting **spacing around "yes" and "no"** values, and **fixing duplicated commas**. The cleaned dataset, stored in a new file named modified_chronic_kidney_disease.arff.

Code:

```
In [3]: # Setting Up File to read
file_path = 'chronic_kidney_disease.arff'

# Read the file
with open(file_path, 'r') as file:
    file_content = file.readlines()

# Applying Correction to the data
file_content = [line.replace('\t', '') for line in file_content]  # Replace all tabs with nothing
file_content = [line.replace(' yes', 'yes').replace('yes ', 'yes') for line in file_content]  # Correct 'yes'
file_content = [line.replace(' no', 'no').replace('no ', 'no') for line in file_content]  # Correct 'no'
file_content = [line.replace(',,', ',') for line in file_content]  # Correct ",,"

# Save the modified file
modified_file = 'modified_chronic_kidney_disease.arff'
with open(modified_file, 'w') as file:
    file.writelines(file_content)
```

Observation:

Following steps are performed to fix the same:

- Removing Tab Characters: All tab characters were eliminated from the file to ensure uniform delimiter usage across the dataset.
- Correcting "Yes" and "No" Values Spacing: Rectified spacing problems around "yes" and "no" values.
- Fixing Comma Duplication: Handled occurrences of repeated commas which misrepresented the structure of the dataset.

By making these adjustments, stored the refined data in a new file **modified_chronic_kidney_disease.arff**. The cleaned dataset has been loaded for further analysis & which has **400 rows** and **25 columns**.

Data Preparation

Code:

```
In [4]: # load the ARFF file
data, meta = arff.loadarff(modified_file)
# Convert the data to a pandas DataFrame
df = pd.DataFrame(data)
# Convert byte strings to regular strings for object type columns
df_obj = df.select_dtypes([object])
df[df_obj.columns] = df_obj.apply(lambda x: x.str.decode('utf-8'))
# top 5 records
df.head()
```

Out[4]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc	rbcc	htn	dm	cad	appet	pe	ane	class
0	48.0	80.0	1.020	1	0	?	normal	notpresent	notpresent	121.0	...	44.0	7800.0	5.2	yes	yes	no	good	no	no	ckd
1	7.0	50.0	1.020	4	0	?	normal	notpresent	notpresent	NaN	...	38.0	6000.0	NaN	no	no	no	good	no	no	ckd
2	62.0	80.0	1.010	2	3	normal	normal	notpresent	notpresent	423.0	...	31.0	7500.0	NaN	no	yes	no	poor	no	yes	ckd
3	48.0	70.0	1.005	4	0	normal	abnormal	present	notpresent	117.0	...	32.0	6700.0	3.9	yes	no	no	poor	yes	yes	ckd
4	51.0	80.0	1.010	2	0	normal	normal	notpresent	notpresent	106.0	...	35.0	7300.0	4.6	no	no	no	good	no	no	ckd

5 rows × 25 columns

Observation:

The dataset has **400 rows** and **25 columns** and with size of **10,000**. As indicated in the instructions document "**chronic_kidney_disease.info.txt**".

The DataFrame has **null values** & few null values are represented as **'?'** which has been replaced with **np.nan**. The missing values are **imputed** with ***median*** *for numerical data* and ***mode*** *for categorical data*.

Below is the brief overview of the columns and their respective null percentages:

Age: 2.25%
Blood Pressure (bp): 3.00%
Specific Gravity (sg): 11.75%
Albumin (al): 11.50%
Sugar (su): 12.25%
Red Blood Cells (rbc): 38.00%
Pus Cell (pc): 16.25%
Pus Cell Clumps (pcc): 1.00%
Bacteria (ba): 1.00%
Blood Glucose Random (bgr): 11.00%
Blood Urea (bu): 4.75%
Serum Creatinine (sc): 4.25%
Sodium (sod): 21.75%
Potassium (pot): 22.00%
Hemoglobin (hemo): 13.00%
Packed Cell Volume (pcv): 17.75%
White Blood Cell Count (wbcc): 26.50%
Red Blood Cell Count (rbcc): 32.75%
Hypertension (htn): 0.50%
Diabetes Mellitus (dm): 0.50%
Coronary Artery Disease (cad): 0.50%
Appetite (appet): 0.25%
Pedal Edema (pe): 0.25%
Anemia (ane): 0.25%
Class: 0.00%

No duplicates were found in the DataFrame **before or after** the imputation.

Exploratory Data Analysis

Univariate Analysis

Describe

```
In [16]: df.describe().T
```

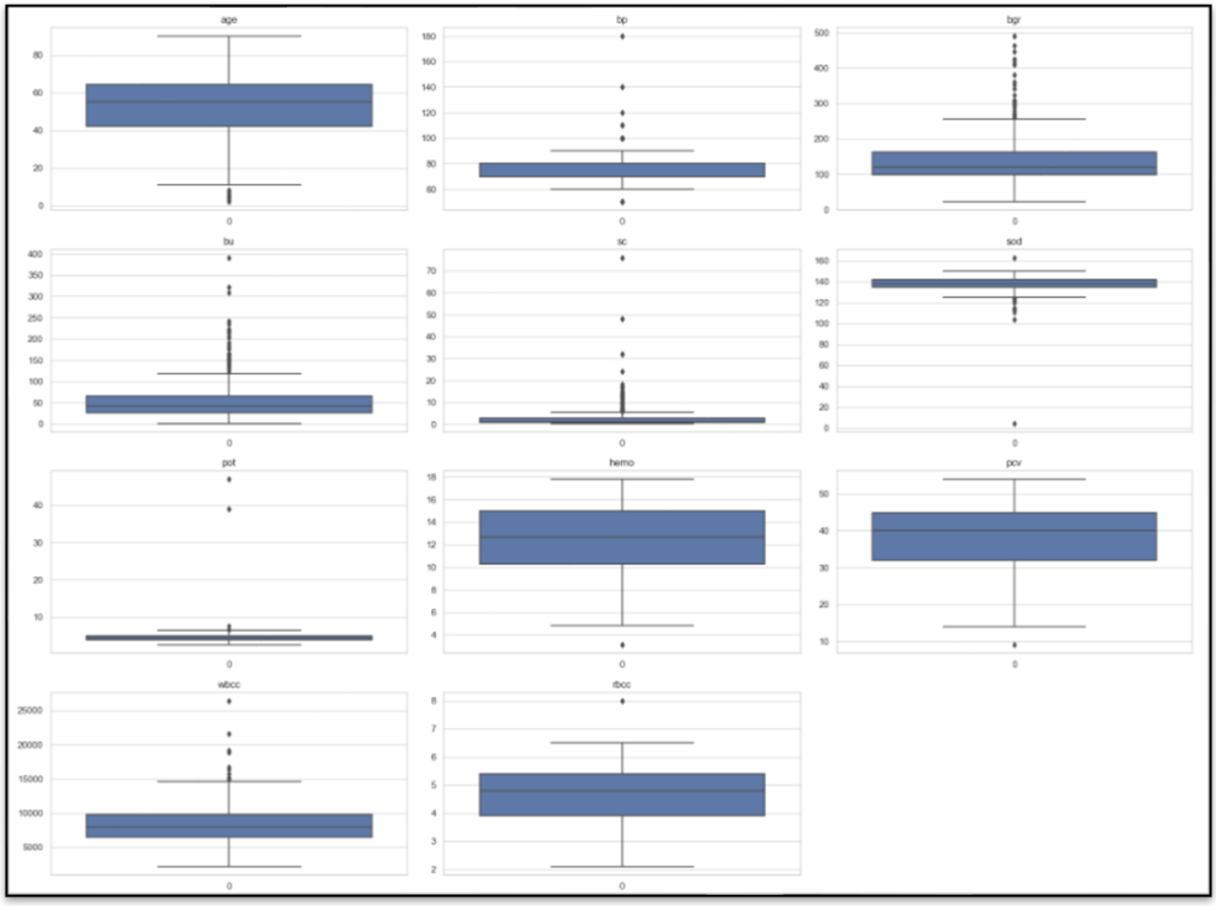
Out[16]:

	count	mean	std	min	25%	50%	75%	max
age	400.0	51.562500	16.982996	2.0	42.000	55.00	64.000	90.0
bp	400.0	76.575000	13.489785	50.0	70.000	80.00	80.000	180.0
bgr	400.0	145.062500	75.260774	22.0	101.000	121.00	150.000	490.0
bu	400.0	56.693000	49.395258	1.5	27.000	42.00	61.750	391.0
sc	400.0	2.997125	5.628886	0.4	0.900	1.30	2.725	76.0
sod	400.0	137.631250	9.206332	4.5	135.000	138.00	141.000	163.0
pot	400.0	4.577250	2.821357	2.5	4.000	4.40	4.800	47.0
hemo	400.0	12.542500	2.716490	3.1	10.875	12.65	14.625	17.8
pcv	400.0	39.082500	8.162245	9.0	34.000	40.00	44.000	54.0
wbcc	400.0	8298.500000	2529.593814	2200.0	6975.000	8000.00	9400.000	26400.0
rbcc	400.0	4.737750	0.841439	2.1	4.500	4.80	5.100	8.0

- The dataset represents a wide age range **(2-90 years),** with an average of **51.48** years.
- Blood pressure varies among individuals, with a mean of **76.47 mmHg**.
- Specific Gravity (sg) averages at **1.017**, indicating mostly normal kidney function. However, Albumin (al) and Sugar (su) levels vary widely **(0-5)**, suggesting varied kidney function.
- Blood Glucose Random (bgr) levels range from **22-490 mg/dl**, indicating diverse metabolic states.
- Blood Urea (bu) and Serum Creatinine (sc) show **significant variation**, highlighting potential kidney function issues.
- Sodium (sod) and Potassium (pot) levels are **within normal ranges**, but variability suggests potential **imbalances**.
- Hemoglobin (hemo), Packed Cell Volume (pcv), and White (wbcc) and Red Blood Cell Counts (rbcc) show wide ranges, important for diagnosing conditions like **anemia**.

Box Plot:

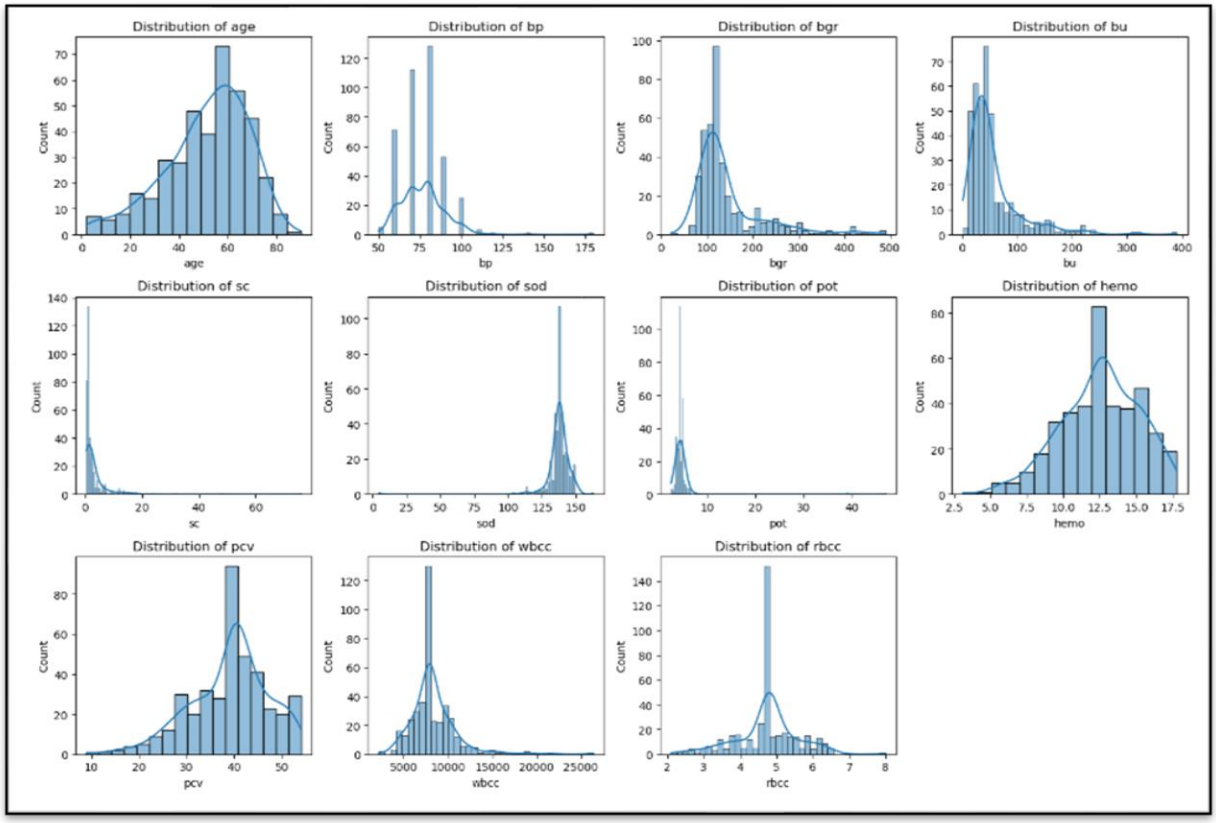
```
In [20]: # Setting the aesthetics for the plots
sns.set(style="whitegrid")

# Plotting box plots for numerical columns
plt.figure(figsize=(20, 15))
for i, col in enumerate(df_numerical, 1):
    plt.subplot(4, 3, i)
    sns.boxplot(data[col])
    plt.title(col)

plt.tight_layout()
plt.show()
```



Significant outliers in attributes such as bgr, bu, sc, sod, pot, and wbcc reflects natural variation or health conditions.
The IQR differences across parameters like bu and sc denote high variability among individuals.

Histogram and Skewness

```
In [17]: describe_df = df.describe()
summary_list = []

for column in describe_df.columns:
    mean = describe_df[column]['mean']
    std = describe_df[column]['std']
    min_val = describe_df[column]['min']
    max_val = describe_df[column]['max']

    median = describe_df[column]['50%']
    if mean > median:
        skewness = "right skewed"
    elif mean < median:
        skewness = "left skewed"
    else:
        skewness = "normally distributed"

    summary = f"{column}: has a mean of {mean:.2f} & standard deviation of {std:.2f}. Values range from {min_val} to {max_val} &
    summary_list.append(summary)

summary_list

Out[17]: ['age: has a mean of 51.56 & standard deviation of 16.98. Values range from 2.0 to 90.0 & left skewed.',
 'bp: has a mean of 76.58 & standard deviation of 13.49. Values range from 50.0 to 180.0 & left skewed.',
 'bgr: has a mean of 145.06 & standard deviation of 75.26. Values range from 22.0 to 490.0 & right skewed.',
 'bu: has a mean of 56.69 & standard deviation of 49.40. Values range from 1.5 to 391.0 & right skewed.',
 'sc: has a mean of 3.00 & standard deviation of 5.63. Values range from 0.4 to 76.0 & right skewed.',
 'sod: has a mean of 137.63 & standard deviation of 9.21. Values range from 4.5 to 163.0 & left skewed.',
 'pot: has a mean of 4.58 & standard deviation of 2.82. Values range from 2.5 to 47.0 & right skewed.',
 'hemo: has a mean of 12.54 & standard deviation of 2.72. Values range from 3.1 to 17.8 & left skewed.',
 'pcv: has a mean of 39.08 & standard deviation of 8.16. Values range from 9.0 to 54.0 & left skewed.',
 'wbcc: has a mean of 8298.50 & standard deviation of 2529.59. Values range from 2200.0 to 26400.0 & right skewed.',
 'rbcc: has a mean of 4.74 & standard deviation of 0.84. Values range from 2.1 to 8.0 & left skewed.']
```



Histogram and Skewness:

- 'age: has a mean of 51.56 & standard deviation of 16.98. Values range from 2.0 to 90.0 & left skewed.',
- 'bp: has a mean of 76.58 & standard deviation of 13.49. Values range from 50.0 to 180.0 & left skewed.',
- 'bgr: has a mean of 145.06 & standard deviation of 75.26. Values range from 22.0 to 490.0 & right skewed.',
- 'bu: has a mean of 56.69 & standard deviation of 49.40. Values range from 1.5 to 391.0 & right skewed.',
- 'sc: has a mean of 3.00 & standard deviation of 5.63. Values range from 0.4 to 76.0 & right skewed.',
- 'sod: has a mean of 137.63 & standard deviation of 9.21. Values range from 4.5 to 163.0 & left skewed.',
- 'pot: has a mean of 4.58 & standard deviation of 2.82. Values range from 2.5 to 47.0 & right skewed.',
- 'hemo: has a mean of 12.54 & standard deviation of 2.72. Values range from 3.1 to 17.8 & left skewed.',
- 'pcv: has a mean of 39.08 & standard deviation of 8.16. Values range from 9.0 to 54.0 & left skewed.',
- 'wbcc: has a mean of 8298.50 & standard deviation of 2529.59. Values range from 2200.0 to 26400.0 & right skewed.',
- 'rbcc: has a mean of 4.74 & standard deviation of 0.84. Values range from 2.1 to 8.0 & left skewed.'

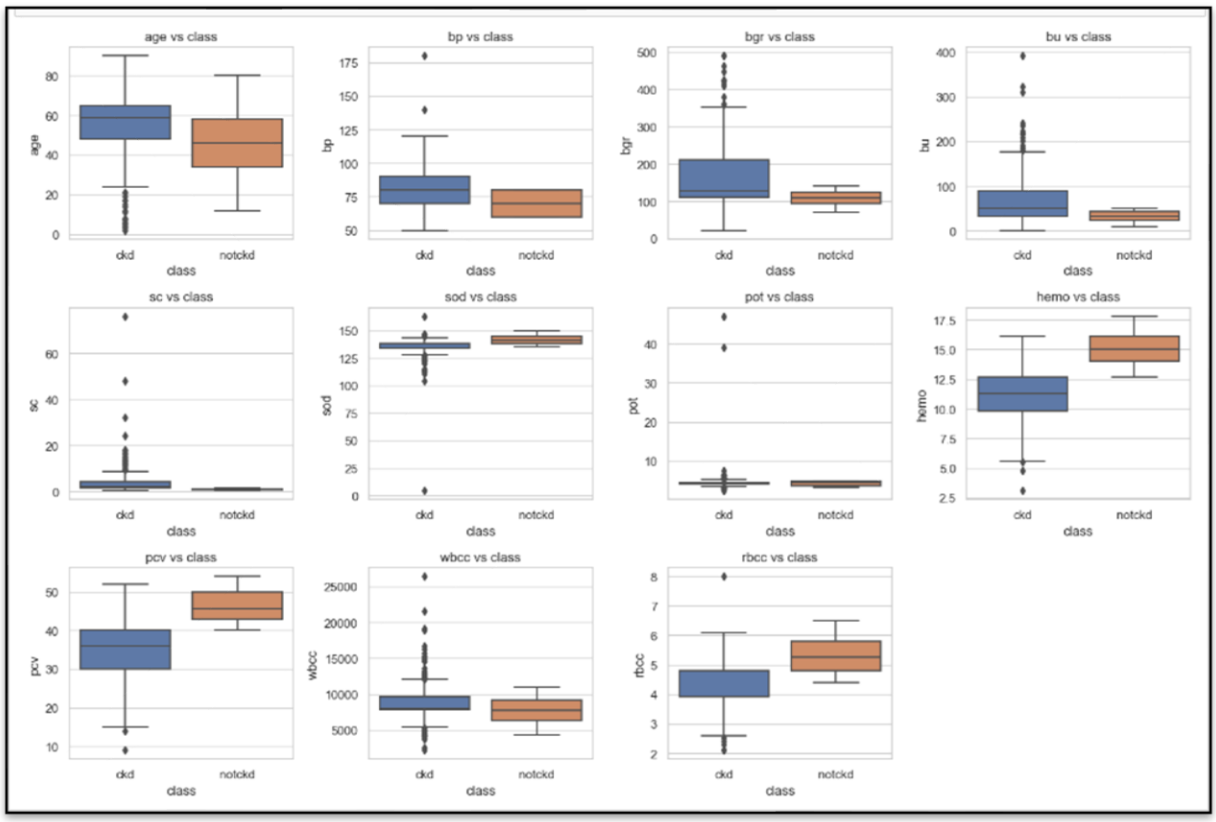
- Right-skewed distributions in al, su, bgr, bu, and sc indicate most values are low with a tail towards higher values. also, left-skewed distributions in age, bp, and hemo suggest a concentration of higher values.
- The non-normal distributions of sg, sod, and hemo reflect the complex of medical data.

Bivariate Analysis

Box Plot with Class

Code:

```
In [21]: # Plot Box plot
plt.figure(figsize=(15,10))

for i, column in enumerate(df_numerical.columns):
    plt.subplot(3, 4, i+1)
    sns.boxplot(x='class', y=column, data=df)
    plt.title(f'{column} vs class')

plt.tight_layout()
plt.show()
```

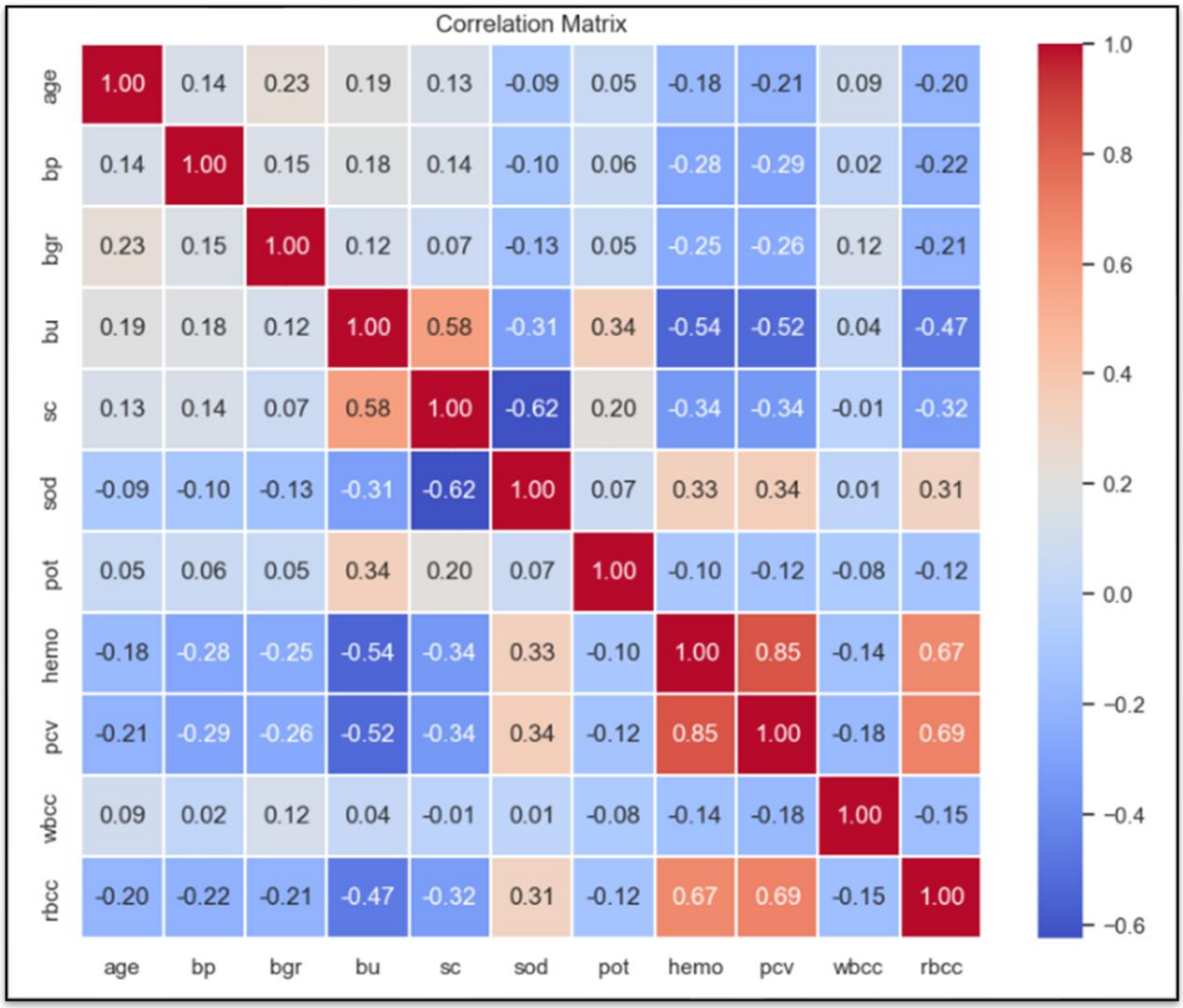


Box plots reveal distinct distributions for parameters like hemo, and pcv when compared against the disease presence class, suggesting these variables are key in distinguishing between classes.

Co-relation Analysis

Code:

```
In [22]: # Correlation Analysis
corr = df_numerical.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.05)
plt.title('Correlation Matrix')
plt.show()
```



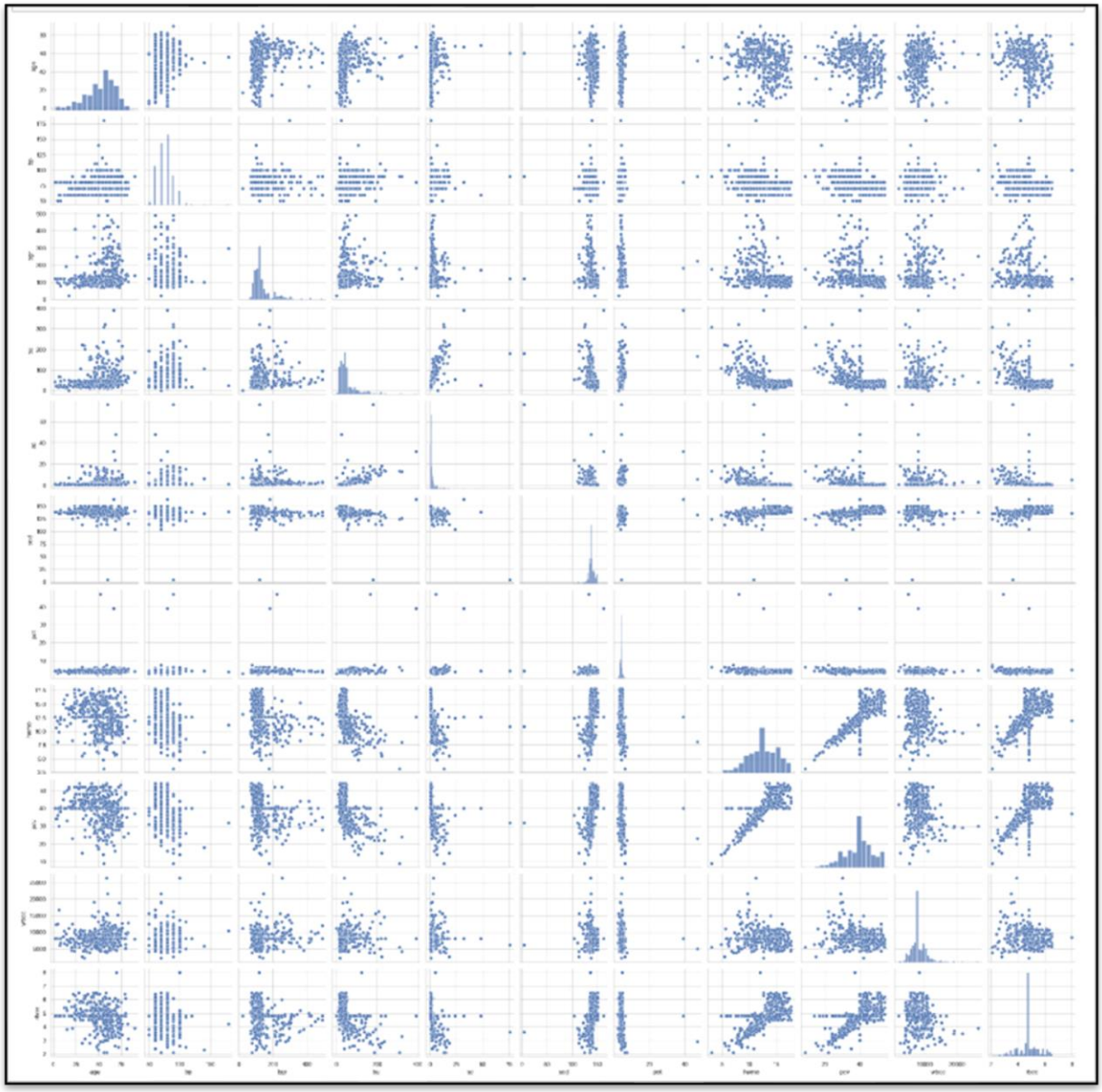
- Hemoglobin (hemo) and Packed Cell Volume (pcv) show a strong positive correlation (~0.90), indicating they are consistent indicators of each other's levels.

- Serum Creatinine (sc) and Sodium (sod) show a strong negative correlation (~-0.69), suggesting that as sc levels increase, sod levels decrease, potentially indicating kidney function impairment.

Pairplot Analysis:

Code:

```
In [24]: # Plot Pair plot
         sns.pairplot(df[df_numerical.columns])
         plt.show()
```



- Hemoglobin (hemo) and Packed Cell Volume (pcv) show a strong positive correlation, useful for diagnosing conditions like anemia or blood disorders.
- Age shows diverse patterns when plotted against various parameters, highlighting the complexity of age's interaction with physiological parameters and disease markers.
- Outliers are present in variables like bgr, Blood Urea (bu), and Serum Creatinine (sc), potentially indicating extreme cases or severe health conditions.

- Serum Creatinine (sc) and Blood Urea (bu) show a positive correlation, suggesting that their joint increase could signal kidney impairment or disease.

Target Class Distribution:

Code:

```
In [25]: # Target class distribution
         plt.figure(figsize=(6, 4))
         sns.countplot(x='class', data=df)
         plt.title('Distribution of Target Class')
         plt.show()
```

Target class distribution

The distribution of the target class in the dataset represents patients having chronic kidney disease (ckd) and those without (notckd). Specifically, there are **150 cases classified as ckd** and **250 cases** classified as **notckd**.

The dataset is imbalanced, with a larger number of notckd cases. This imbalance could influence the performance of a predictive model.

Feature Importance

Feature importance is a practice of assigning scores to the independent features for a machine learning model based on their influence in predicting the target variable, scores would be indicated which features are more significant in model's predictions and which features could potentially be ignored without loosing too much predictive power.

Information Gain (IG) is a measure helps to identify which features are the most informative for predicting the target variable & used with decision tree algorithms and is based on the concept of entropy from information theory.

```
# Splitting X and y for Predicting Feature importance
X = df.drop('class', axis=1)
y = df['class']
```

```
# Creating copy
X_label_encoded = X.copy()

# Select Categorical Columns
df_cat = X_label_encoded.select_dtypes(include=['object']).columns

# applying label encoding for categorical Columns
for col in df_cat:
    le = LabelEncoder()
    # Fit label encoder and return encoded labels
    X_label_encoded[col] = le.fit_transform(X[col].astype(str))
```

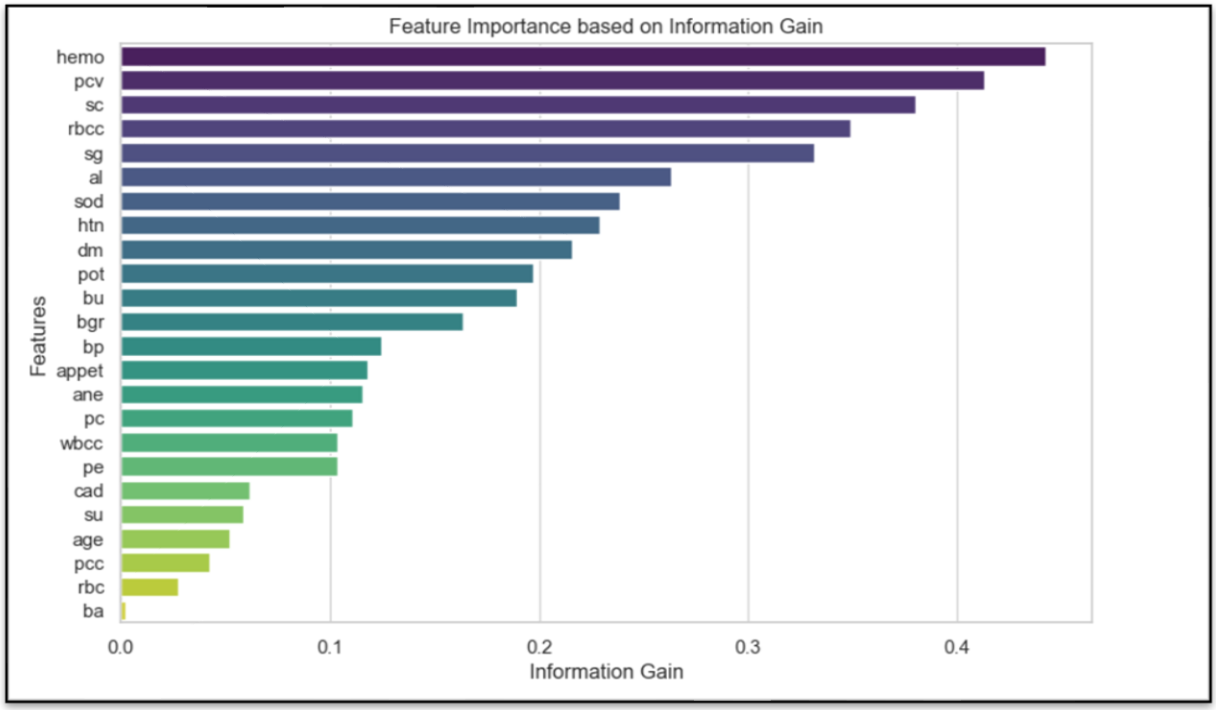
```
y = LabelEncoder().fit_transform(y)
```

Information Gain (IG) is a measure helps to identify which features are the most informative for predicting the target variable & used with decision tree algorithms and is based on the concept of entropy from information theory.

```
# Calculate Information Gain (mutual information) for each feature
info_gain = mutual_info_classif(X_label_encoded, y)
```

```
# Create a DataFrame for better visualization
info_gain_df = pd.DataFrame({'Feature': X_label_encoded.columns, 'Information Gain': info_gain}).sort_values(by='Information Gain
info_gain_df
```

Chart



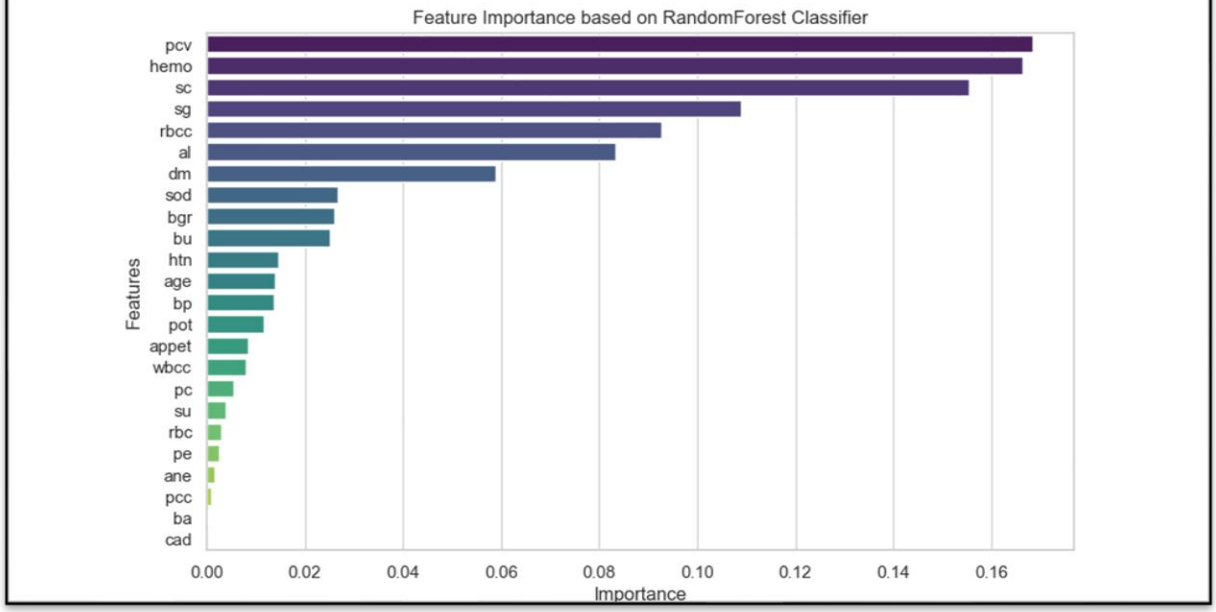
Random Forest uses a measure based on the decrease in node impurity & is Gini impurity or entropy in case of classification, and variance in case of regression. Every time a feature is used

to split data, the impurity of the child nodes is calculated. The difference between the impurity of the parent node and the weighted impurity of the child nodes gives us the impurity decrease. The feature importance for a feature is calculated as the average impurity decrease for that feature across all trees in the forest.

```
In [32]: X_train, X_test, y_train, y_test = train_test_split(X_label_encoded, y, test_size=0.2, random_state=42)

         # Train a RandomForestClassifier
         rf = RandomForestClassifier(n_estimators=100, random_state=42)
         rf.fit(X_train, y_train)

         # Get feature importance from the RandomForest model
         feature_importance_rf = pd.DataFrame({'Feature': X.columns, 'Importance': rf.feature_importances_}).sort_values(by='Importance',
         feature_importance_rf
```

Chart

```
In [33]: plt.figure(figsize=(10, 6))
         sns.barplot(x='Importance', y='Feature', data=feature_importance_rf, palette='viridis')
         plt.title('Feature Importance based on RandomForest Classifier')
         plt.xlabel('Importance')
         plt.ylabel('Features')
         plt.show()
```



Feature Importance:

High Importance Features: Hemoglobin (hemo), Packed Cell Volume (pcv), Serum Creatinine (sc), Specific Gravity (sg), and Red Blood Cell Count (rbcc) are consistently ranked high in importance by both Random Forest and Information Gain, indicating its importance for prediction.

Moderately Important Features: Albumin (al), Diabetes Mellitus (dm), and Sodium (sod) shows moderate importance with slight ranking variations, suggesting their significant contribution for prediction.

Low Importance: Hypertension (htn), Blood Glucose Random (bgr), and Blood Urea (bu) show some ranking variance and are generally considered of moderate to low importance. Age, Blood

Pressure (bp), and Potassium (pot) are ranked lower, indicating they might have less impact on the model.

No Importance: Pus Cell Clumps (pcc), Bacteria (ba), and Coronary Artery Disease (cad) consistently show little to no importance, suggesting it may not significantly influence the predictions and could be dropped without affecting performance significantly.

Dropping pcc & ba due to zero importance scores.

Train Test Split

Code:

```
In [34]: # Dropping the columns pcc and ba
         X = df.drop(columns=['class','pcc','ba'], axis=1)

In [35]: # Splitting the data into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,stratify=y)
         print("X_train shape:", X_train.shape)
         print("X_test shape:", X_test.shape)
         print("y_train shape:", y_train.shape)
         print("y_test shape:", y_test.shape)

         X_train shape: (320, 22)
         X_test shape: (80, 22)
         y_train shape: (320,)
         y_test shape: (80,)
```

Column Drop: Columns pcc and ba were dropped based on the feature importance.

Imbalanced Dataset: Stratification is used during data split to ensure balanced representation of each target class in the **train** and **test** sets, crucial for **avoiding bias** in classification tasks.

Split Ratio: Divided into **80%** for training and **20%** for testing.

Dataset Sizes:

X_train shape: (320, 22)
X_test shape: (80, 22)
y_train shape: (320,)
y_test shape: (80,)

Feature Scaling

Code:

```
In [37]: # Identifying categorical and numerical columns
         categorical_cols = X_train.select_dtypes(include=['object', 'bool']).columns.tolist()
         numerical_cols = X_train.select_dtypes(include=['float64', 'int64']).columns.tolist()

         # Preprocessing for numerical data
         numerical_transformer = Pipeline(steps=[('scaler', StandardScaler())])

         # Preprocessing for categorical data
         categorical_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle_unknown='ignore'))])

         # Bundle preprocessing for numerical and categorical data
         preprocessor = ColumnTransformer(
             transformers=[
                 ('num', numerical_transformer, numerical_cols),
                 ('cat', categorical_transformer, categorical_cols)])
```

```
In [38]: # Define the models
decision_tree_model = Pipeline(steps=[('preprocessor', preprocessor),
                                      ('classifier', DecisionTreeClassifier(random_state=42))])

random_forest_model = Pipeline(steps=[('preprocessor', preprocessor),
                                      ('classifier', RandomForestClassifier(random_state=42))])

# Encoding the target variable since it's categorical
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

Preprocessing Summary:

- **Feature Segregation:** Features were divided into numerical and categorical types for Scaling.
- **Standard Scaling:** Applied StandardScaler for Numerical features & standardizing features by removing the mean and scaling to unit variance.
- **One-Hot Encoding**: Applied OneHotEncoder for categorical features converting them into a binary matrix representation.
- **Integration: Preprocessing** steps were combined using **ColumnTransformer**, allowing concurrent application of numerical and categorical transformers to their **respective columns**. This ensures correct preprocessing, enhances **reproducibility** and **efficiency.**
- **Label Encoding:** Applied to the target variable (y) to convert categorical labels into a numeric format

Base Model:

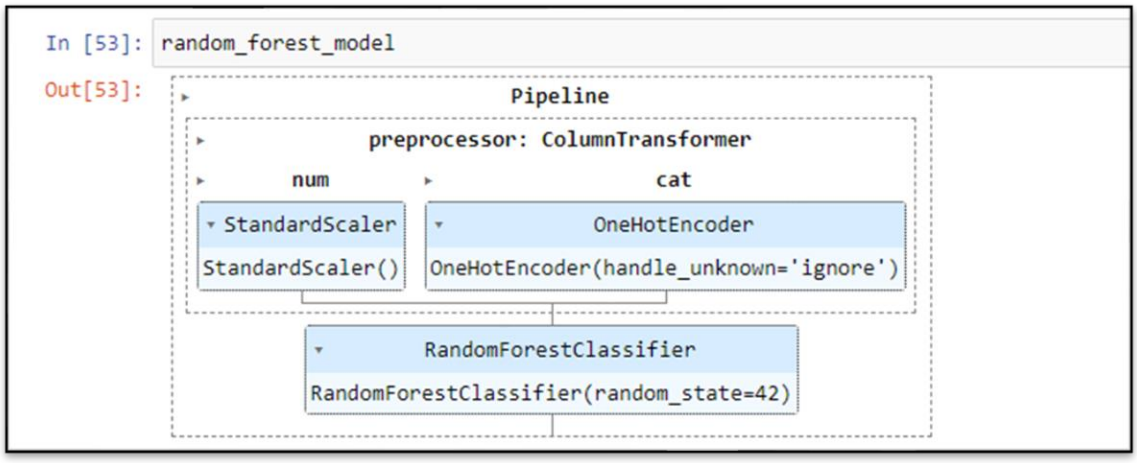
Code:

```
In [38]: # Define the models
decision_tree_model = Pipeline(steps=[('preprocessor', preprocessor),
                                      ('classifier', DecisionTreeClassifier(random_state=42))])

random_forest_model = Pipeline(steps=[('preprocessor', preprocessor),
                                      ('classifier', RandomForestClassifier(random_state=42))])

# Encoding the target variable since it's categorical
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Fit the models
decision_tree_model.fit(X_train, y_train_encoded)
random_forest_model.fit(X_train, y_train_encoded)
```

Decision tree Model:

```
In [52]: decision_tree_model
Out[52]:
Pipeline
  preprocessor: ColumnTransformer
    num
      StandardScaler
      StandardScaler()
    cat
      OneHotEncoder
      OneHotEncoder(handle_unknown='ignore')
  DecisionTreeClassifier
  DecisionTreeClassifier(random_state=42)
```

Random Forest Model:



- **Reproducibility:** A random state of 42 was set for two models to ensure consistent results across multiple runs.
- **Baseline Models**: Trained two models, Decision Tree Classifier and Random Forest Classifier, integrated into pipelines with preprocessing steps for feature scaling and encoding.
- **Pipelines**: Used to encapsulate the entire process from raw data preprocessing to model training in a single workflow, simplifying the code and minimizing data leakage risk.
- **Decision Tree Classifier Pipeline**: Includes a preprocessor for handling numerical and categorical features, followed by the classifier itself.
- **Random Forest Classifier Pipeline**: Includes a preprocessor for handling numerical and categorical features, followed by the classifier itself

Base Model Evaluation:

Code:

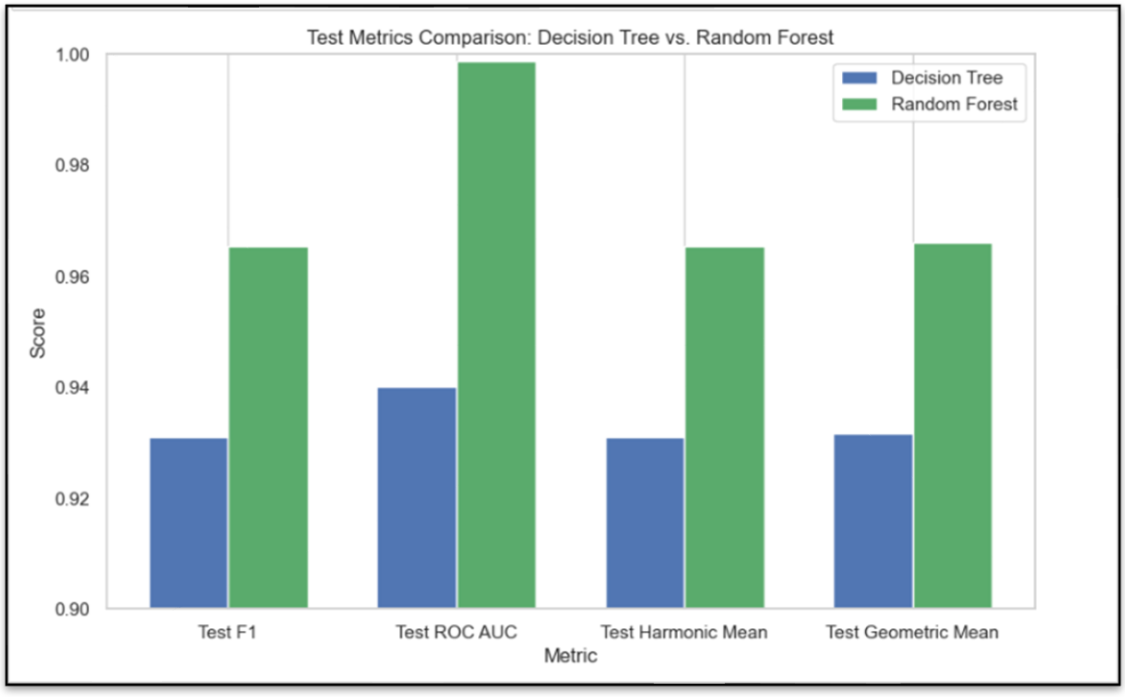
```
In [36]: # Function to calculate metrics
def calculate_metrics(model, X_train, y_train, X_test, y_test):
    # Getting predictions and probabilities
    train_preds = model.predict(X_train)
    test_preds = model.predict(X_test)
    train_probs = model.predict_proba(X_train)[:, 1]  # Probability of positive class
    test_probs = model.predict_proba(X_test)[:, 1]

    # Calculating metrics
    metrics = {
        'Train F1': f1_score(y_train, train_preds),
        'Test F1': f1_score(y_test, test_preds),
        'Train ROC AUC': roc_auc_score(y_train, train_probs),
        'Test ROC AUC': roc_auc_score(y_test, test_probs),
        'Train Precision': precision_score(y_train, train_preds),
        'Test Precision': precision_score(y_test, test_preds),
        'Train Recall': recall_score(y_train, train_preds),
        'Test Recall': recall_score(y_test, test_preds),
        'Train Harmonic Mean': hmean([precision_score(y_train, train_preds), recall_score(y_train, train_preds)]),
        'Test Harmonic Mean': hmean([precision_score(y_test, test_preds), recall_score(y_test, test_preds)]),
        'Train Geometric Mean': gmean([precision_score(y_train, train_preds), recall_score(y_train, train_preds)]),
        'Test Geometric Mean': gmean([precision_score(y_test, test_preds), recall_score(y_test, test_preds)]),
    }

    return metrics, train_preds, test_preds
```

Classification Report & Confusion Matrix

```
In [39]: print('confusion matrix - Decision Tree')
         print(confusion_dt)
         print('Classification report - Decision Tree')
         print(report_dt)

confusion matrix - Decision Tree
[[49  1]
 [ 3 27]]
Classification report - Decision Tree
              precision    recall  f1-score   support

           0       0.94      0.98      0.96        50
           1       0.96      0.90      0.93        30

    accuracy                           0.95        80
   macro avg       0.95      0.94      0.95        80
weighted avg       0.95      0.95      0.95        80
```

```
In [40]: print('confusion matrix - Random Forest')
         print(confusion_rf)
         print('Classification report - Random Forest')
         print(report_rf)

confusion matrix - Random Forest
[[50  0]
 [ 2 28]]
Classification report - Random Forest
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        50
           1       1.00      0.93      0.97        30

    accuracy                           0.97        80
   macro avg       0.98      0.97      0.97        80
weighted avg       0.98      0.97      0.97        80
```

Chart:



Model Comparison

Overfitting Analysis:

Decision Tree: Achieves a perfect F1 score of 1.0 on training data, dropping to 0.931 on the test set, indicating potential overfitting. Precision remains high, but recall drops from 1.0 in training to 0.9 in testing.

Random Forest: Also achieves a perfect F1 score of 1.0 on training data, with a slight decrease to 0.966 on the test set. This model maintains perfect precision and high recall, suggesting less overfitting than the Decision Tree.

Performance on Imbalanced Datasets:

Decision Tree: Favors the majority class (0), with more false negatives than false positives. The Test ROC AUC of 0.94 suggests room for improvement in handling the minority class.

Random Forest: has very good performance with no false positives and few false negatives, demonstrating strong ability to handle the imbalanced dataset. The Test ROC AUC of 0.999 indicates exceptional class differentiation ability.

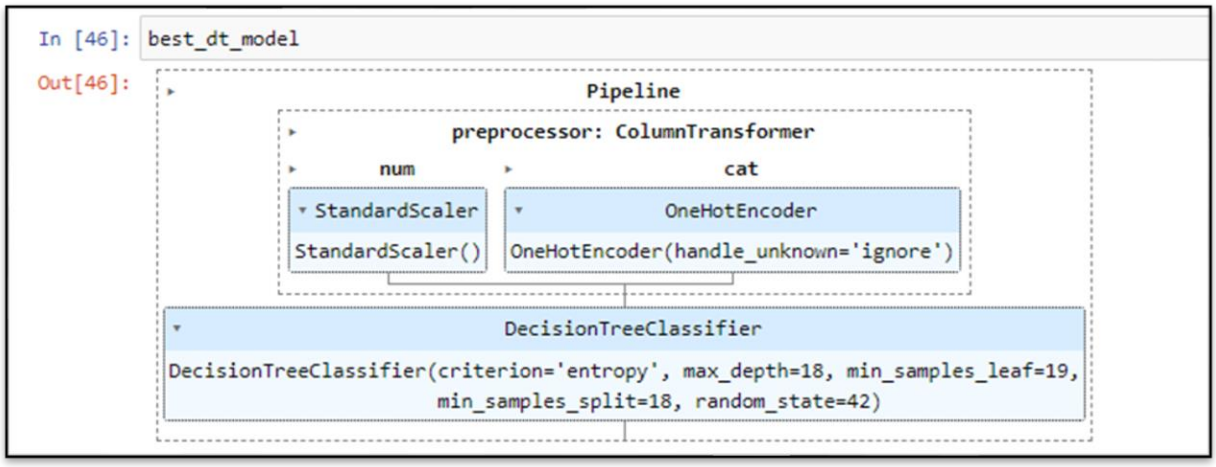
Conclusion: The Random Forest model outperforms the Decision Tree indicating better generalization, stronger handling of the imbalanced dataset and higher precision, recall, and F1 scores. The Decision Tree, while performing well, shows signs of overfitting and weaker handling of imbalance.

Hyper Parameters Tuned Models

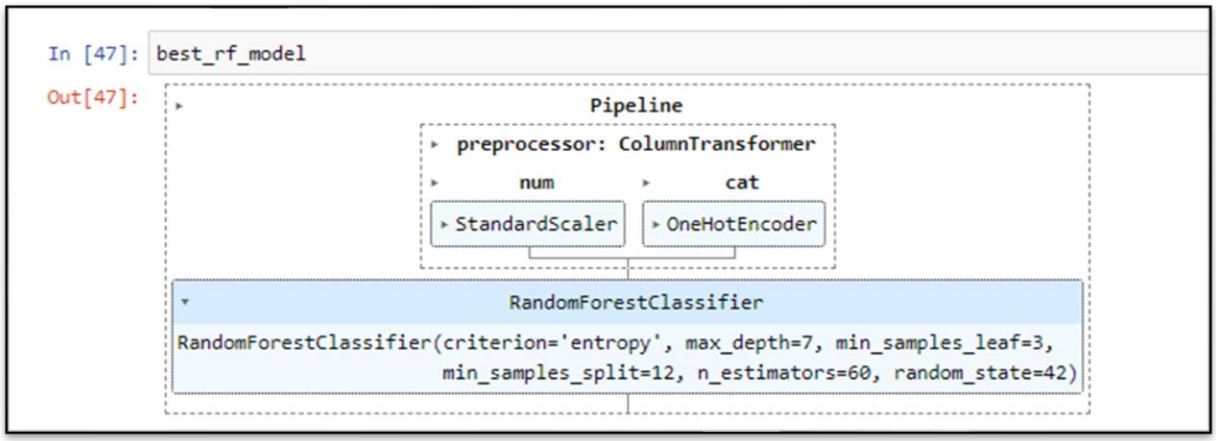
Code:

```
In [45]: from sklearn.model_selection import RandomizedSearchCV
         import numpy as np

         # Hyperparameter grid for Decision Tree
         dt_param_grid = {
             'classifier__max_depth': np.arange(3, 20),
             'classifier__min_samples_split': np.arange(2, 20),
             'classifier__min_samples_leaf': np.arange(1, 20),
             'classifier__criterion': ['gini', 'entropy']
         }

         # Hyperparameter grid for Random Forest
         rf_param_grid = {
             'classifier__n_estimators': np.arange(10, 200, 10),
             'classifier__max_depth': np.arange(3, 20),
             'classifier__min_samples_split': np.arange(2, 20),
             'classifier__min_samples_leaf': np.arange(1, 20),
             'classifier__criterion': ['gini', 'entropy']
         }

         # Setup RandomizedSearchCV for Decision Tree
         dt_random_search = RandomizedSearchCV(decision_tree_model, dt_param_grid, n_iter=10, cv=5, scoring='roc_auc', random_state=42, n_

         # Setup RandomizedSearchCV for Random Forest
         rf_random_search = RandomizedSearchCV(random_forest_model, rf_param_grid, n_iter=10, cv=5, scoring='roc_auc', random_state=42, n_

         # Fit the models
         dt_random_search.fit(X_train, y_train_encoded)
         rf_random_search.fit(X_train, y_train_encoded)

         # Best estimators
         best_dt_model = dt_random_search.best_estimator_
         best_rf_model = rf_random_search.best_estimator_
```

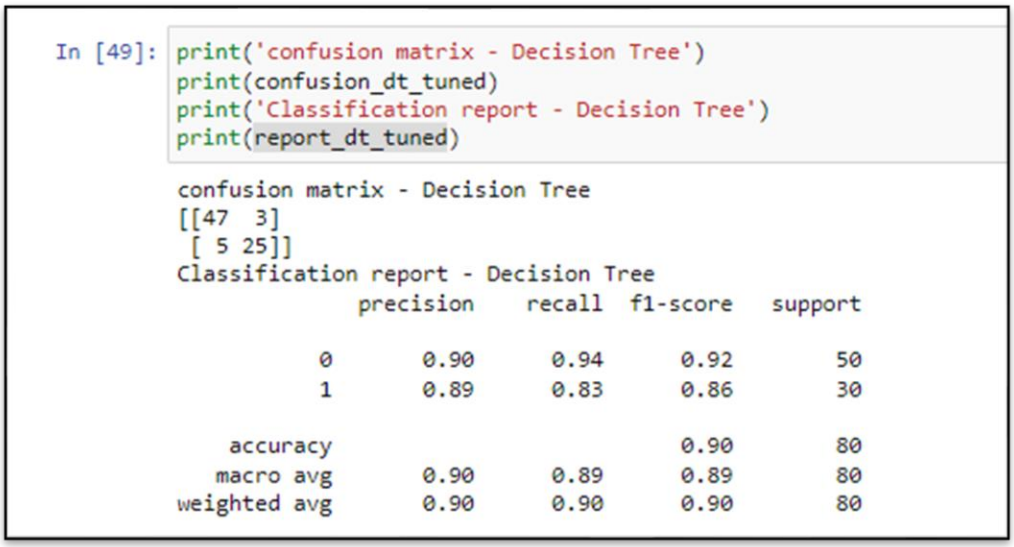
Best Models:
Decision Tree:



Random Forest:



Confusion Matrix and Classification Report
Decision Tree Tuned Model

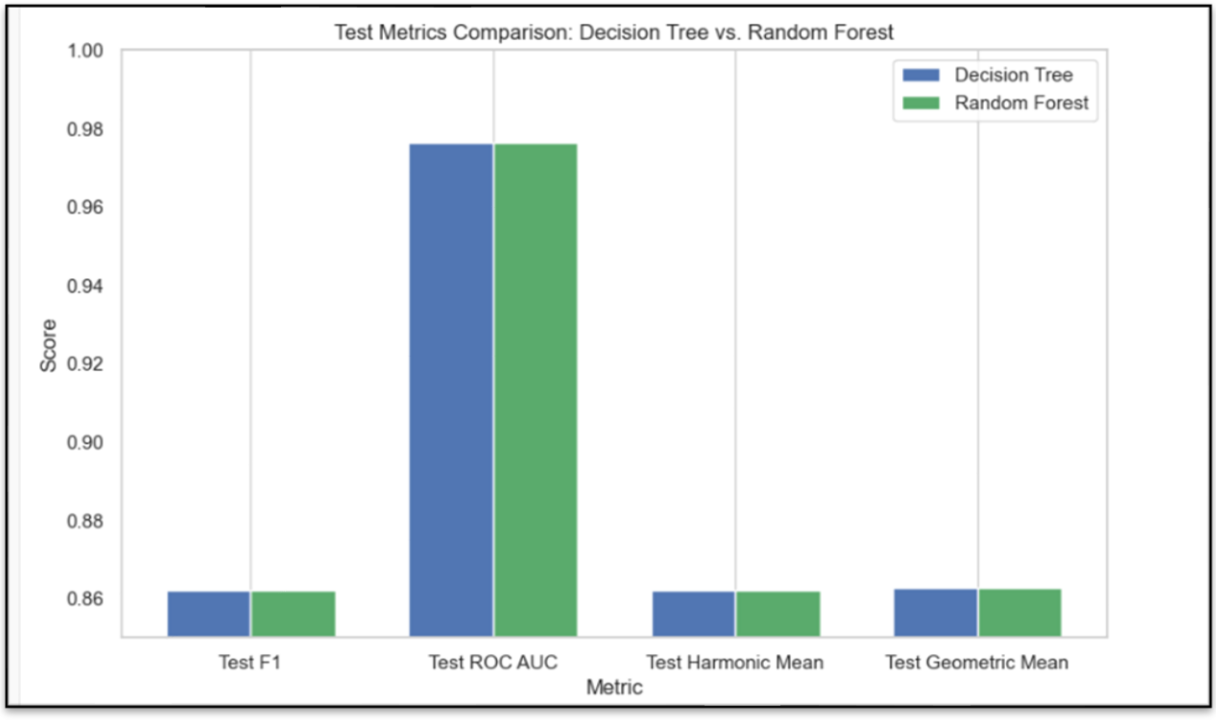


```
In [49]: print('confusion matrix - Decision Tree')
         print(confusion_dt_tuned)
         print('Classification report - Decision Tree')
         print(report_dt_tuned)

confusion matrix - Decision Tree
[[47  3]
 [ 5 25]]
Classification report - Decision Tree
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        50
           1       0.89      0.83      0.86        30

    accuracy                           0.90        80
   macro avg       0.90      0.89      0.89        80
weighted avg       0.90      0.90      0.90        80
```

Random Forest Tuned Model:

```
In [50]: print('confusion matrix - Random Forest')
         print(confusion_rf_tuned)
         print('Classification report - Random Forest')
         print(report_dt_tuned)

confusion matrix - Random Forest
[[49  1]
 [ 3 27]]
Classification report - Random Forest
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        50
           1       0.89      0.83      0.86        30

    accuracy                           0.90        80
   macro avg       0.90      0.89      0.89        80
weighted avg       0.90      0.90      0.90        80
```

Chart:



Model Comparison

Overfitting Analysis:

- Decision Tree Base Model: Achieves perfect 1.0 scores on training metrics, which drop on the test set, suggesting potential overfitting.

- Random Forest Base Model: Displays perfect training scores with high test scores, indicating slight overfitting, but less than the Decision Tree due to superior test performance.
- Decision Tree Tuned Model: Shows reduced training metrics compared to its base model, suggesting less overfitting. However, the decrease in test metrics indicates a trade-off between generalization and overall performance.
- Random Forest Tuned Model: Exhibits high training and test metrics, indicating excellent generalization and slight overfitting due to near-perfect scores.

Performance on Imbalanced Datasets:

- Decision Tree Base Model: Displays high precision and recall for both classes, slightly favouring the majority class.
- Random Forest Base Model: Demonstrates excellent precision and recall, slightly outperforming the Decision Tree in handling the minority class.
- Tuned Models: Show a decrease in performance metrics compared to their base models. The Decision Tree Tuned Model shows a significant drop in recall for the minority class, indicating decreased performance on the imbalanced dataset. The Random Forest Tuned Model maintains high precision but sees a **drop in recall** for the minority class.

Final Model pickle

The Random Forest **Base Model** is recommended as the final model for this scenario due to its high performance on both the training and test sets, excellent **precision** and **recall** values for the **minority class**, and strong balance between high performance and generalization capability on the **imbalanced dataset**.

While tuning aims to **improve model performance** and generalization, in this case, the **Random Forest Base Model** already provides a strong balance.

Code:

```
In [57]: # Save the Random Forest Base model to a pickle file
         joblib.dump(random_forest_model, 'random_forest_model.pkl')

         # Create a new zip file containing the pickle file
         with zipfile.ZipFile('random_forest_model.zip', 'w') as myzip:
             myzip.write('random_forest_model.pkl', 'random_forest_model.pkl')
```

Folder Structure Files:

Conclusion:

- Predictive analysis, machine learning models like Random Forest will help in early detection of CKD.
- This approach enhances patient care by enabling timely interventions and personalized treatment plans.
- The study's success highlights machine learning's potential in medical diagnostics and its ability to handle complex datasets.

Next Steps:

- **Model Preparation**: Create a scoring script for predictions and define an Azure ML environment with all necessary **Python dependencies**. Also, create an inference configuration specifying the **Docker image** and scoring script.
- **Model Deployment**: Deploy the model as a **web service** in Azure Container Instances (ACI) or Azure Kubernetes Service (AKS) using the Model.deploy() function.
- **Model Utilization**: Test the deployed model by obtaining the **REST endpoint** of the **web service** and **sending data for prediction**. Then, consume the endpoint in application for real-time predictions.

References

- Great Learning. (2024). **Introduction to Data Frames in pandas**. Available at: https://olympus.mygreatlearning.com/courses/97366/pages/4-dot-1-introduction-to-data-frames-in-pandas

- Great Learning. (2024). **Outliers, Missing, Censored, and Incorrect Data**. Available at: https://olympus.mygreatlearning.com/courses/97366/pages/4-dot-5-outliers-missing-censored-and-incorrect-data

- Great Learning Olympus. (2023). **Site for PGP for Artificial intelligence and machine learning study materials**. Available at: https://olympus.mygreatlearning.com/courses/74001

- Scikit-Learn. (No Date). **Hyper parameter tuning**. Available at: https://scikit-learn.org/stable/modules/grid_search.html

- Sruthi E R. (2024, Jan 03). **Understand Random Forest Algorithms With Examples (Updated 2024)** Available at: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

- Brownlee, J. (No Date). **Metrics for Imbalanced Classification Books**. Available at: https://machinelearningmastery.com/imbalanced-classification-with-python/

- Jain, A. (2018). **Hyperparameters and Parameters in Machine Learning: What's the difference?** Avialble at: https://www.analyticsvidhya.com/blog/2018/08/difference-between-parameters-and-hyperparameters/

- Goyal, P. (2018). **Feature Importance and Why It's Important**. Avialble at: https://towardsdatascience.com/feature-importance-and-why-its-important-c46d326e81d2

- Patel, K. (2018). **Why, How and When to Scale your Features**. Available at: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

- Brownlee, J. (October 16, 2019). **Information Gain and Mutual Information for Machine Learning. Machine Learning Mastery.** Available at: https://machinelearningmastery.com/information-gain-and-mutual-information/

- Sethi, A. (March 6, 2020). **One Hot Encoding vs. Label Encoding using Scikit-Learn**. Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/

- scikit-learn. **1.10. Decision Trees**. Available at: https://scikit-learn.org/stable/modules/tree.html

- Kim, C. (July 14, 2022). **Decision Tree Classifier with Scikit-Learn from Python**. Medium. Available at: https://medium.com/@chyun55555/decision-tree-classifier-with-scikit-learn-from-python-e83f38079fea

- Doshi, N. (March 19, 2022). **5 Most Important Metrics for Model Evaluation in Machine Learning. Towards Data Science**. Available at: https://towardsdatascience.com/5-most-important-metrics-for-model-evaluation-in-machine-learning-c74fc9d0609f

- Microsoft Developer (Jul 10, 2020) **How do you deploy a machine learning model as a web service within Azure** Available at: https://www.youtube.com/watch?v=n8h6_Expf38