EARLY DETECTION OF CHRONIC KIDNEY DISEASE

TABLE OF CONTENTS:

ABSTRACT:

Chronic Kidney Disease (CKD) is a common health condition that causes significant morbidity and mortality worldwide. Early detection and intervention are critical for preventing the disease's progression and complications. This abstract describes a comprehensive approach to CKD detection that emphasizes the integration of clinical, biochemical, and technological advances.

The traditional CKD diagnosis paradigm is based on measuring glomerular filtration rate (GFR) and urinary albumin excretion. Recent advances in biomarker research, on the other hand, have revealed novel indicators, allowing for more sensitive and specific detection of early renal impairment. Markers such as neutrophil gelatinase-associated lipocalin (NGAL), kidney injury molecule-1 (KIM-1) and cystatin C show promise in detecting subtle renal dysfunction before conventional markers do.

In the age of digital health, the incorporation of artificial intelligence (AI) and machine learning algorithms for predictive modeling and risk stratification is gaining traction. These technologies use a variety of datasets, including electronic health records and genetic data, to identify patterns that indicate early CKD. These predictive models allow clinicians to identify high-risk patients and implement targeted interventions, preventing or delaying the progression of CKD.

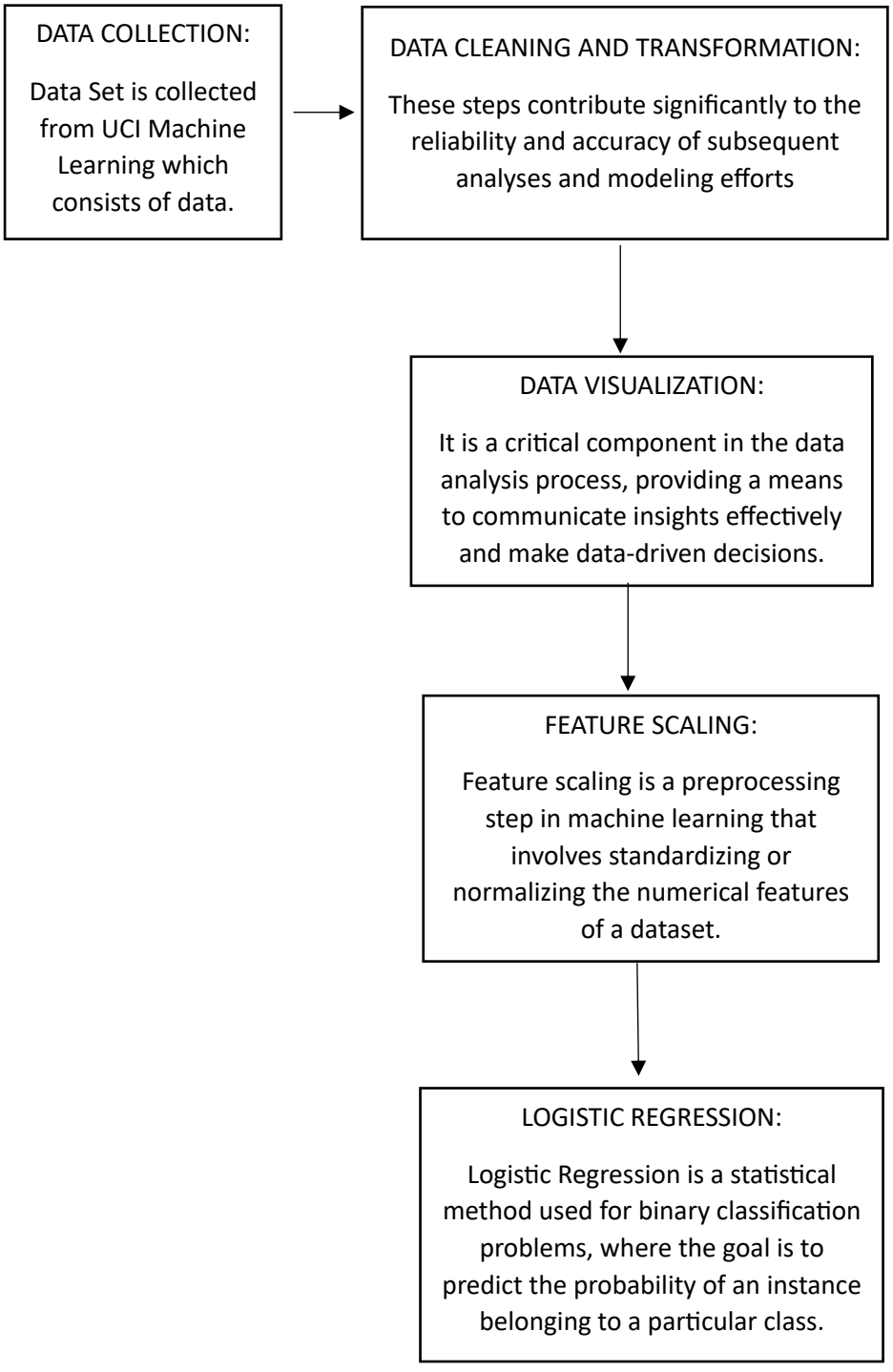
INTRODUCTION:

Chronic Kidney Disease (CKD) is a global health issue with a rising prevalence and significant public health implications. CKD progresses silently until advanced stages, when interventions become less effective and complications arise. Early detection of CKD is critical for implementing timely interventions that can slow or stop the progression of the disease, mitigate associated complications, and improve patient outcomes.

CKD has a significant burden, contributing to increased morbidity, mortality, and healthcare costs. Identifying individuals at risk or in the early stages of CKD allows for the implementation of preventive measures, lifestyle changes, and targeted therapeutic interventions. This overview discusses the significance of early detection in the context of CKD, highlighting key challenges and emerging strategies to improve our ability to identify and manage this condition in its early stages.

PROJECT METHODOLOGY

BLOCK DIAGRAM:



Data Collection:

Utilizing the UCI Machine Learning Repository for chronic kidney disease data collection involves extracting relevant features from datasets such as patient demographics, blood pressure, serum creatinine, and other clinical parameters.

Loading the downloaded csv file

```
#importing the downloaded data csv file
data=pd.read_csv('~/Downloads/kidney_disease.csv')
data
data
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	...	47	6700	4.9	no	no	no	good	no	no	notckd
396	396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	54	7800	6.2	no	no	no	good	no	no	notckd
397	397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	...	49	6600	5.4	no	no	no	good	no	no	notckd
398	398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	51	7200	5.9	no	no	no	good	no	no	notckd
399	399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	53	6800	6.1	no	no	no	good	no	no	notckd

400 rows × 26 columns

DATA CLEANING AND TRANSFORMATION:

- Data cleaning and transformation are crucial steps in the data preparation process before analysis or modeling. These steps help ensure that the data is accurate, consistent, and suitable for the intended purpose. Here's an overview of data cleaning and transformation.
- Effective data cleaning and transformation contribute significantly to the reliability and accuracy of subsequent analyses and machine learning models. These processes are often iterative, requiring continuous refinement as insights are gained and the data's nature becomes clearer.

In [6]:
```
#imputing mode in place of nan values by using simpleimputer

from sklearn.impute import SimpleImputer
impute_mode=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
data_imputed=pd.DataFrame(impute_mode.fit_transform(data))
data_imputed.columns=data.columns
data_imputed
```

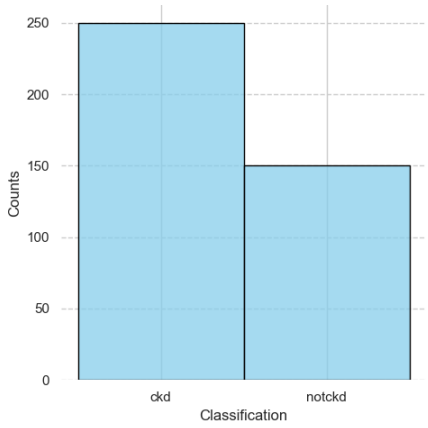
Out[6]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.02	1.0	0.0	normal	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.02	4.0	0.0	normal	normal	notpresent	notpresent	...	38	6000	5.2	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.01	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	5.2	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.01	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	395	55.0	80.0	1.02	0.0	0.0	normal	normal	notpresent	notpresent	...	47	6700	4.9	no	no	no	good	no	no	notckd
396	396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	54	7800	6.2	no	no	no	good	no	no	notckd
397	397	12.0	80.0	1.02	0.0	0.0	normal	normal	notpresent	notpresent	...	49	6600	5.4	no	no	no	good	no	no	notckd
398	398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	51	7200	5.9	no	no	no	good	no	no	notckd
399	399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	...	53	6800	6.1	no	no	no	good	no	no	notckd

400 rows × 26 columns

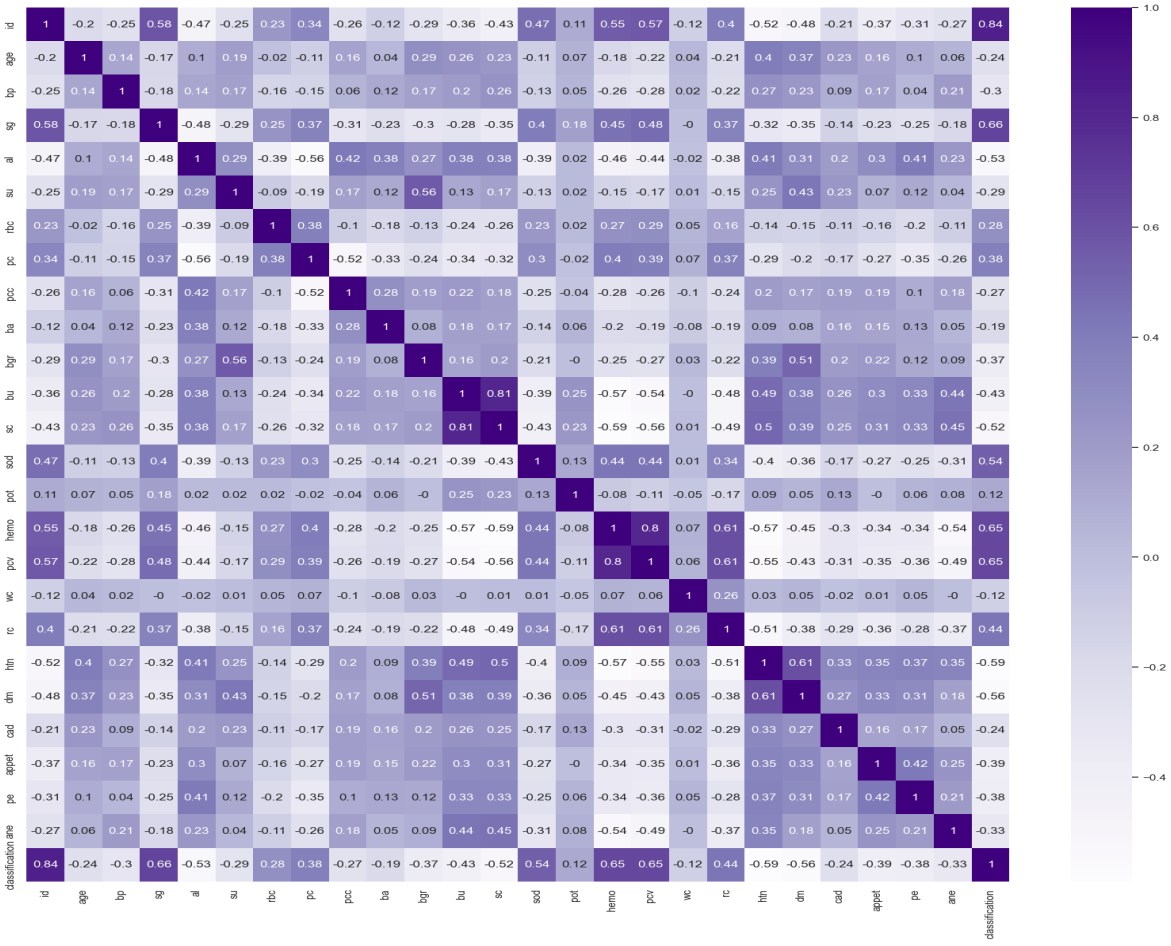
DATA VISUALIZATION:

- Data visualization is a powerful method of presenting information graphically to facilitate understanding, analysis, and communication of patterns, trends, and insights within data.
- Data visualization is a critical aspect of the data analysis process, providing a means to communicate insights effectively and make data-driven decisions. It combines art and science to transform raw data into meaningful and actionable information.
- Data visualization is a dynamic and evolving field, continuously incorporating new technologies and methodologies to enhance the representation of data. It plays a pivotal role in making data more accessible, interpretable, and actionable.



- Creating a heatmap typically involves using a dataset with numerical values, where you want to visualize the pairwise correlations between variables. For the purpose of demonstrating a heatmap, We created a synthetic dataset using the popular seaborn and libraries in Python.
- **np.random.randn** generates random values from a standard normal distribution to create synthetic numerical data.

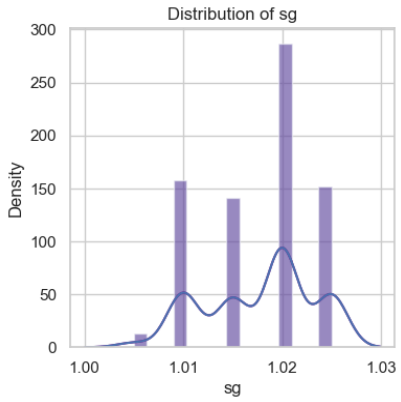
- **pd.DataFrame** creates a Data Frame with the synthetic data.
- **data.corr()** calculates the correlation matrix.
- **sns.heatmap** creates the heatmap using Seaborn.
- This is just a demonstration with synthetic data; in a real-world scenario.

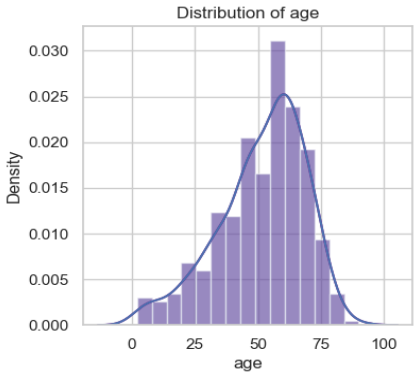


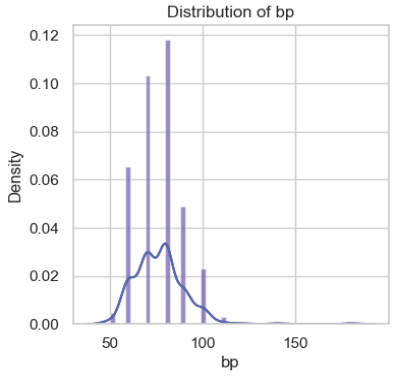
Plotting the bar graph of each and every attribute against the density:

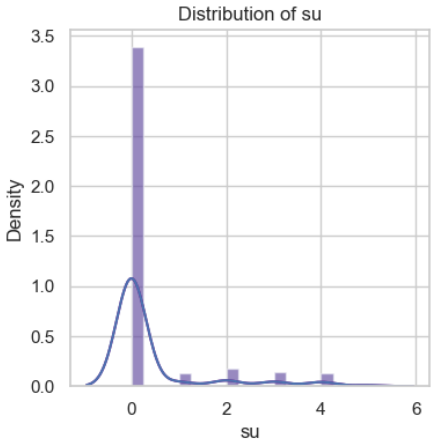
- Creating bar graphs for each attribute against the density information involves visualizing the distribution of each attribute in your dataset.
- Plotting a bar graph of each numerical attribute against its density can provide valuable insights into the distribution of data. While the code is using **sns.distplot** (distribution plot) rather than a bar graph, the purpose is similar—to visualizing the distribution of values within each numerical attribute.

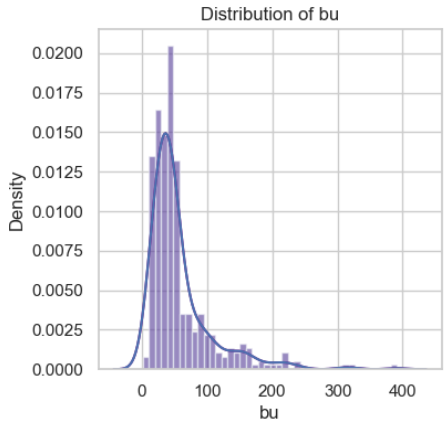
```
#plotting the bar graph of each and every attribute against the density
sns.set({'figure.figsize': (4, 4)})
for col in list(data_imputed.select_dtypes(exclude=['object']).columns)[1:]:#excuding the object datatype
    sns.set_theme(style="whitegrid")
    sns.distplot(data_imputed[col], color='purple')
    sns.distplot(data_imputed[col])
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Density')
    plt.show()
```

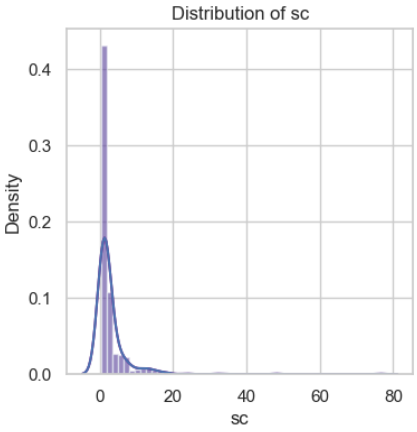

Distribution of sg
Density
sg
1.00
1.01
1.02
1.03

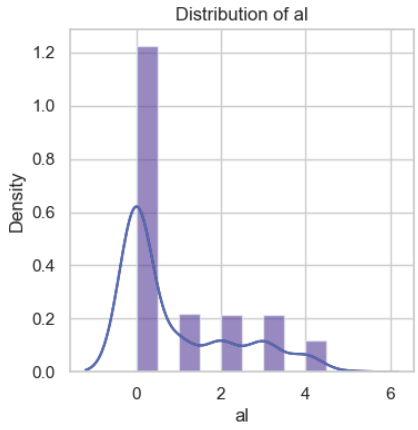

Distribution of age
Density
age

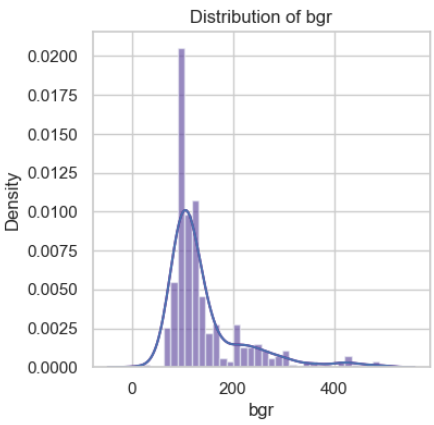

Distribution of bp
Density
bp


Distribution of su
Density
su


Distribution of bu
Density
bu

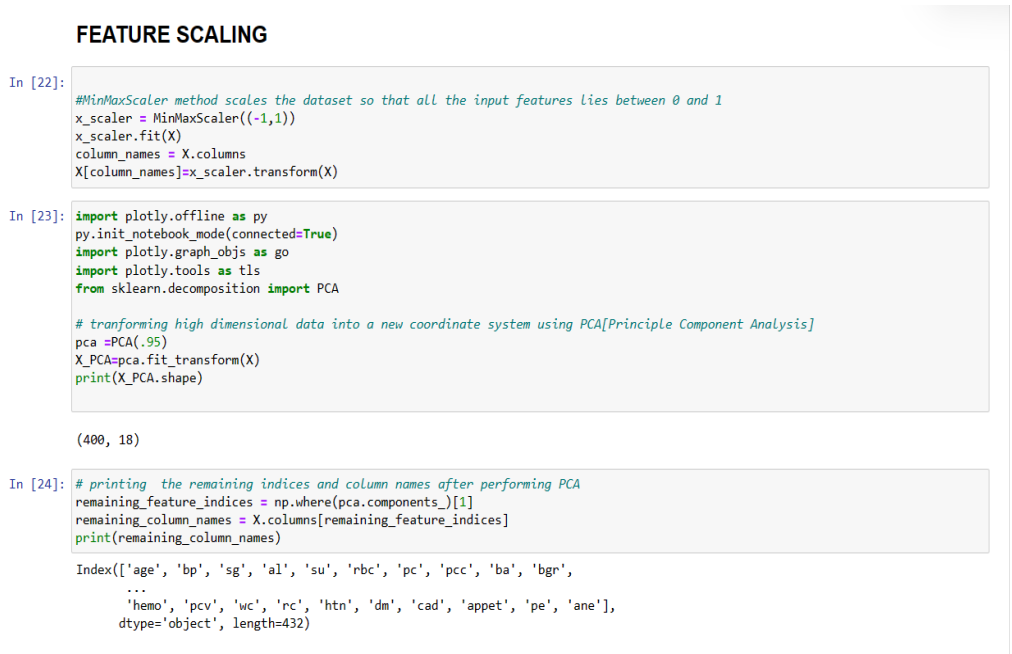

Distribution of sc
Density
sc


Distribution of al
Density
al


Distribution of bgr
Density
bgr

FEATURE SCALING:

- Feature scaling is a preprocessing technique used in machine learning to standardize or normalize the range of independent variables or features of a dataset. The goal is to ensure that all features have a similar scale, preventing certain features from dominating others during the training of machine learning models. Two common methods for feature scaling are normalization (Min-Max scaling) and standardization (Z-score normalization).
- Feature scaling is particularly important for algorithms that rely on distance measures or gradient-based optimization, such as k-nearest neighbors, support vector machines, and many variants of gradient descent.
- The choice of feature scaling method depends on the nature of the data and the requirements of the machine learning algorithm. Experimentation and testing different scaling methods can help identify the most suitable one for a specific task. In many cases, standardization is a safe choice and is widely used in practice.



FEATURE SCALING

In [22]:
```
#MinMaxScaler method scales the dataset so that all the input features lies between 0 and 1
x_scaler = MinMaxScaler((-1,1))
x_scaler.fit(X)
column_names = X.columns
X[column_names]=x_scaler.transform(X)
```

In [23]:
```
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
from sklearn.decomposition import PCA

# tranforming high dimensional data into a new coordinate system using PCA[Principle Component Analysis]
pca =PCA(.95)
X_PCA=pca.fit_transform(X)
print(X_PCA.shape)
```

```
(400, 18)
```

In [24]:
```
# printing  the remaining indices and column names after performing PCA
remaining_feature_indices = np.where(pca.components_)[1]
remaining_column_names = X.columns[remaining_feature_indices]
print(remaining_column_names)
```

```
Index(['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr',
       ...
       'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane'],
      dtype='object', length=432)
```

MinMaxScaler Initialization:

The MinMaxScaler method is initialized with a specified feature range of (-1, 1). This scaler is commonly used to normalize or scale the input features of a dataset, ensuring that all values fall within the specified range.

Scaling the Dataset:

The fit method of the MinMaxScaler is applied to the dataset X to compute the minimum and maximum values for each feature in the dataset.

Column Names Extraction:

The column names of the original dataset X are stored in the variable column_names.

Transforming Data with MinMaxScaler:

The original dataset X is transformed using the MinMaxScaler. The transformed data is then assigned back to the original dataset's columns.

Plotly Imports:

Plotly's offline mode is initialized, and necessary Plotly libraries (graph_objs and tools) are imported. Plotly is a visualization library used for creating interactive plots.

PCA Initialization:

An instance of the Principle Component Analysis (PCA) is created with the explained variance set to 95%. PCA is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while retaining as much variance as possible.

PCA Transformation:

The dataset X is transformed using the PCA fit-transform method, resulting in a new dataset X_PCA with a reduced number of features while preserving 95% of the variance.

Printing Transformed Data Shape:

The shape of the transformed dataset X_PCA is printed. In this case, the dataset has 400 samples and 18 principal components.

Remaining Feature Indices Extraction:

The indices of the remaining features after PCA are obtained using np.where(pca.components_). These indices represent the features that contribute significantly to the dataset after dimensionality reduction.

Remaining Column Names Extraction:

The remaining column names are extracted from the original dataset X based on the indices of the features that survived the PCA. These column names correspond to the features that still contribute meaningfully to the dataset.

Printing Remaining Column Names:

The remaining column names are printed, indicating the features that were retained after PCA.

Training and Testing the obtained Data:

Printing Shape of X_train:

print(X_train.shape) is used to display the shape of the training data (X_train). This typically outputs the number of samples and features in the training dataset.

Printing Shape of X_test:

print(X_test.shape) is used to display the shape of the testing data (X_test). Similar to the first point, this shows the number of samples and features in the testing dataset.

Printing Shape of y_train:

print(y_train.shape) is used to display the shape of the training labels (y_train). The shape usually indicates the number of samples in the training set.

Printing Shape of y_test:

print(y_test.shape) is used to display the shape of the testing labels (y_test). Similar to the third point, this shows the number of samples in the testing set.

These print statements are common in machine learning workflows and are useful for checking the dimensions of the datasets at various stages. Understanding the shapes of the training and testing data is crucial for ensuring that the data is correctly split and that the model is trained and tested on the appropriate sets. The shapes printed here would typically be in the form (number of samples, number of features) for the data (X) and (number of samples,) for the labels (y).

Building a Model using LOGISTIC REGRESSION:

- Logistic Regression is a statistical method used for binary classification, where the outcome variable is categorical and has two classes (0 or 1). Despite its name, logistic regression is primarily used for classification rather than regression.
- Logistic Regression is a fundamental algorithm in machine learning, widely used for binary classification tasks due to its simplicity and interpretability.
- One of the notable features of logistic regression is its interpretability. The coefficients associated with each feature provide insights into their impact on the predicted probability. The assumptions of logistic regression include a linear relationship between the features and the log-odds of the outcome.

Building a Model using LOGISTIC REGRESSION

In [27]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initializing and training the logistic regression model
model = LogisticRegression(random_state=7)
model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_scaled)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

#Printing the accuracy ,confusion matrix and classification report
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

Importing Necessary Libraries:

The code imports essential libraries from scikit-learn, including Standard Scaler for feature scaling, Logistic Regression for logistic regression modeling, and metrics like accuracy_score,

classification_report, and confusion_matrix for model evaluation.

Standardizing the Features:

A Standard Scaler is initialized to standardize the features. fit_transform is used to compute the mean and standard deviation of the training data (X_train) and then scale both the training and test sets (X_train_scaled and X_test_scaled, respectively).

Initializing Logistic Regression Model:

An instance of the logistic regression model is created with the LogisticRegression class. The parameter random_state is set to ensure reproducibility.

Training the Logistic Regression Model:

The logistic regression model is trained using the scaled training data (X_train_scaled) and corresponding labels (y_train) with the fit method.

Making Predictions:

Predictions are made on the scaled test set (X_test_scaled) using the predict method of the logistic regression model, resulting in y_pred.

Model Evaluation - Accuracy:

The accuracy of the model is calculated using the accuracy_score function, comparing the predicted labels (y_pred) with the true labels from the test set (y_test).

Model Evaluation - Confusion Matrix:

The confusion matrix is computed using the confusion_matrix function. The confusion matrix provides a detailed breakdown

of correct and incorrect predictions, useful for understanding model performance.

Model Evaluation - Classification Report:

The classification report is generated using the classification_report function. This report includes precision, recall, F1-score, and support for each class, offering a comprehensive assessment of model performance.

Printing Results:

The code prints the accuracy, confusion matrix, and classification report to the console, providing a summary of the logistic regression model's performance on the test set.

```
Accuracy: 0.975
Confusion Matrix:
[[48  2]
 [ 0 30]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        50
           1       0.94      1.00      0.97        30

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

For class 0:

Precision is 1.00 (or 100%), meaning all positive predictions for class 0 are correct.
Recall is 0.96 (or 96%), indicating that the model correctly captures 96% of the instances of class 0.
F1-score is 0.98, providing a balanced measure of precision and recall.
Support is 50, indicating that there are 50 instances of class 0 in the dataset.

For class 1:

Precision is 0.94 (or 94%), indicating that 94% of the positive predictions for class 1 are correct.
Recall is 1.00 (or 100%), meaning the model correctly captures all instances of class 1.
F1-score is 0.97, providing a balanced measure of precision and recall.
Support is 30, indicating that there are 30 instances of class 1 in the dataset.

For class 0:

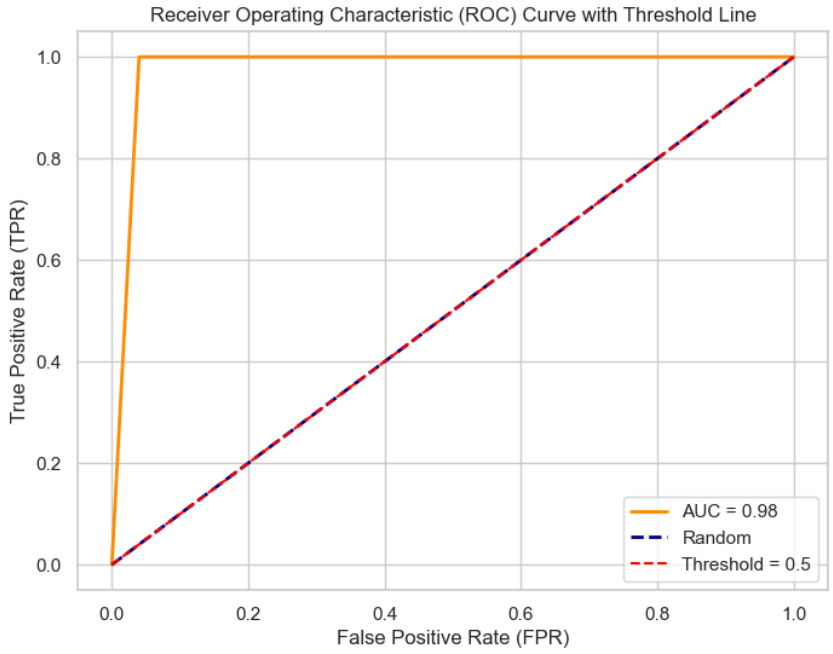
Precision is 1.00 (or 100%), meaning all positive predictions for class 0 are correct.
Recall is 0.96 (or 96%), indicating that the model correctly captures 96% of the instances of class 0.
F1-score is 0.98, providing a balanced measure of precision and recall.
Support is 50, indicating that there are 50 instances of class 0 in the dataset.

For class 1:

Precision is 0.94 (or 94%), indicating that 94% of the positive predictions for class 1 are correct.
Recall is 1.00 (or 100%), meaning the model correctly captures all instances of class 1.
F1-score is 0.97, providing a balanced measure of precision and recall.
Support is 30, indicating that there are 30 instances of class 1 in the dataset.

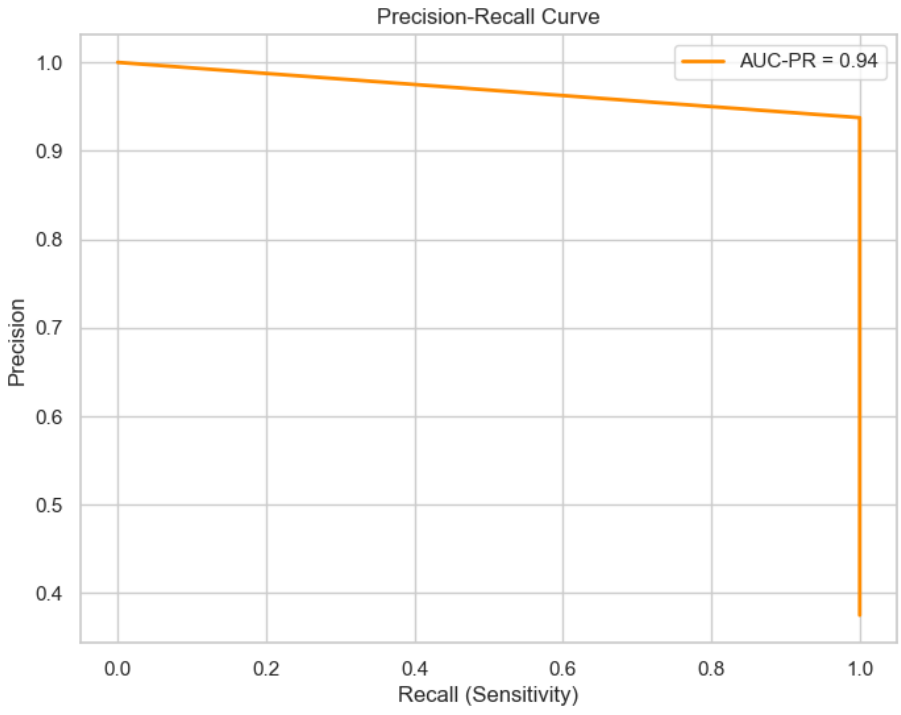
ROC CURVE:

Plotting a Receiver Operating Characteristic (ROC) curve is a common way to evaluate the performance of a binary classification model. The ROC curve is a graphical representation of the trade-off between sensitivity (true positive rate) and specificity (true negative rate) for different thresholds of a classification model.



Plotting Precision-Recall Curve:

A Precision-Recall curve is another way to evaluate the performance of a binary classification model, particularly when dealing with imbalanced datasets. It provides insights into the trade-off between precision and recall for different threshold values.



Precision:

Precision is the ratio of true positive predictions to the total predicted positives. It measures the accuracy of the positive predictions made by the model.
Precision = TP / (TP + FP)
For class 0: Precision = 1.00 (or 100%)
For class 1: Precision = 0.94 (or 94%)

Recall (Sensitivity or True Positive Rate):

Recall is the ratio of true positive predictions to the total actual positives. It measures the model's ability to capture all the positive instances.
Recall = TP / (TP + FN)
For class 0: Recall = 0.96 (or 96%)
For class 1: Recall = 1.00 (or 100%)

Pickle File:

A pickle file is a serialized form of a Python object that allows you to save and load complex data structures, such as lists, dictionaries, and custom objects, in a binary format. The pickle module in Python provides a way to serialize and deserialize Python objects.

Loading the model into a pickel file

```
In [33]: import pickle
         from sklearn.linear_model import LogisticRegression  # Replace with your model

         model = LogisticRegression()
         model.fit(X_train, y_train)

         model_path = 'model.pkl'
         with open(model_path, 'wb') as file:
             pickle.dump(model, file)
```

This pickled model file can later be loaded and used for predictions or further analysis. Keep in mind that the pickled file should be used with caution, especially if it comes from an untrusted source, as it has the potential to execute arbitrary code during the loading process.

Predicting the final result for an unknown data values:

Predicting the final result for an unknown data values

In [34]:
```
new_data = np.array([60, 70, 1.02, 1, 100, 40, 1.2, 3.2, 15, 44, 1000, 3.9, 0, 0, 0, 0, 0, 0])
new_data_reshaped = new_data.reshape(1, -1)

# Scaling the reshaped data
new_data_scaled = scaler.transform(new_data_reshaped)

prediction = model.predict(new_data_scaled)

# printing the result
if prediction == 1:
    print("The person is predicted to have CKD.")
else:
    print("The person is predicted not to have CKD.")
```

```
The person is predicted to have CKD.
```

In [35]:
```
new_data = np.array([60, 70, 1.02, 1, 500, 40, 1.2, 3.2, 15, 44, 1000, 3.9, 0, 0, 1, 1, 1, 0])
new_data_reshaped = new_data.reshape(1, -1)

# Scaling the reshaped data
new_data_scaled = scaler.transform(new_data_reshaped)


prediction = model.predict(new_data_scaled)

# printing the result
if prediction == 1:
    print("The person is predicted to have CKD.")
else:
    print("The person is predicted not to have CKD.")
```

```
The person is predicted not to have CKD.
```

Insights:

Elevated levels of specific gravity, hemoglobin, packed cell volume (PCV), and sodium may be indicative of chronic kidney disease (CKD), but they do not necessarily confirm a kidney infection. CKD is typically diagnosed through a combination of clinical assessments, blood tests (such as serum creatinine and estimated glomerular filtration rate), and urinalysis.

Future Scope:

- **Integration of Machine Learning Algorithms:** Incorporate advanced machine learning algorithms to enhance the model's predictive capabilities. This could involve training the model on larger and more diverse datasets to improve accuracy and account for a broader range of patient characteristics.
- **Real-Time Monitoring and Telehealth Integration:** Develop a system for real-time monitoring of relevant health parameters, allowing for continuous assessment and early detection of changes in kidney function. Integration with telehealth platforms could facilitate remote patient monitoring, improving accessibility and timely intervention.

Conclusion:

By using data and fancy computer techniques, we can find it earlier and take better care of people. It's like teamwork between computers and doctors to make sure everyone gets the right help at the right time.