Risk Factor Prediction of Chronic Kidney Disease

Participants:

1. Kavya Kasthuri Dodle (kavyakasthuridodle@my.unt.edu)
   Role: Project Lead, Data Scientist
   Responsibilities:
   - Overall coordination and project management
   - Data gathering and preparation
   - Selection and use of algorithms
   - Analyzing the performance of models
   - Reporting and documentation
2. Vikas Burgupally (vikasburgupally@my.unt.edu)
   Role: Data Analyst, Researcher
   Responsibilities:
   - Cleaning and preparing data
   - Visualization and exploratory data analysis (EDA)
   - Engineering features and developing features iteratively
   - Model improvement and enhancement
   - Validation and testing of datasets
3. Latha Kolluri (lathakolluri@my.unt.edu)
   Role: Quality Assurance Specialist, Documentation Specialist
   Responsibilities:
   - Plan events and encourage communication
   - Participate in data gathering and preprocessing
   - Check the validity and veracity of data and models.
   - Test things out and make sure the outcomes are reliable
   - Identify and disclose any problems or contradictions.
   - Work together with the group to make sure the project's quality requirements are met.
4. Siva Srinivasa Rao (sivasrinivasaraochinni@my.unt.edu)
   Role: Machine Learning Specialist
   Responsibilities:
   - Learn about and use machine learning algorithms
   - Creation and improvement of models
   - Comparison of performance metrics
   - Provide assistance with feature engineering and data pretreatment.
   - Participate in team efforts for model selection and interpretation.

Communication Channels:

Zoom: We use Zoom for online meetings and conversations. Set up regular meetings to discuss difficulties, schedule tasks, and coordinate project progress.
WhatsApp: Use WhatsApp to stay in touch with teammates and share relevant information while providing timely updates.

Daily Meetings:

Wednesday, Friday, and Tuesday:
Time: 9:30–10:30 p.m.
Goal: Discuss the status, organize the work, deal with any problems, and give updates on everyone's efforts.
Time: One hour (This is not fixed. Can extend based on the requirement)

Sunday (if necessary)
Time: 9:30–10:30 p.m.
Goal: Hold an extra meeting on Sundays if it's necessary to discuss pressing issues, evaluate progress, or make plans for the coming week.
Time: One hour (This is not fixed. Can extend based on the requirement)

Abstract:

Chronic kidney disease (CKD) is a growing medical problem that impairs renal function and ultimately harms the kidneys. Cardiovascular infection and end-stage renal disease are two potentially fatal conditions resulting from CKD, which is currently fairly common. These might be avoided by identifying conditions early and treating those who are in danger. The responsibility of anticipating medical issues is exceedingly challenging. One of the most fatal diseases in the medical world is CKD. The prediction of risk factors is a crucial step in the initial stage before it is too late to identify CKD and eliminate risks. In this study, five algorithms are employed to predict the risk factors of CKD, including Naive Bayes, Random Forest, Simple Logistic Regression, Support Vector Machine, and Linear Regression Model. Five algorithms provide better and faster characterization execution when taking into account the ordered execution and investigations of these methods. The dataset is run through five different algorithms, and the classification of potential risk factors yields the best results.

Data specification:

The dataset used in the project is the Chronic Kidney Disease Data Set from the UCI Machine Learning Repository. This dataset contains information about patients with chronic kidney disease and is commonly used for classification tasks. It contains 400 rows.

Features:

1. age: Age of the patient (numerical)
2. blood pressure: Blood pressure measurement in mm/Hg (numerical)
3. specific gravity: Specific gravity of urine (categorical: normal, low, high)
4. albumin: Albumin levels in urine (categorical: negative, trace, 1, 2, 3, 4, 5)
5. sugar: Sugar levels in urine (categorical: negative, trace, 1, 2, 3, 4, 5)
6. red blood cells: Presence of red blood cells in urine (categorical: normal, abnormal)
7. pus cell: Presence of pus cells in urine (categorical: normal, abnormal)
8. pus cell clumps: Presence of pus cell clumps in urine (categorical: present, not present)
9. bacteria: Presence of bacteria in urine (categorical: present, not present)
10. blood glucose random: Random blood sugar measurement in mg/dL (numerical)
11. blood urea: Blood urea measurement in mg/dL (numerical)
12. serum creatinine: Serum creatinine measurement in mg/dL (numerical)
13. sodium: Sodium measurement in mEq/L (numerical)
14. potassium: Potassium measurement in mEq/L (numerical)
15. hemoglobin: Hemoglobin measurement in g/dL (numerical)
16. packed cell volume: Packed cell volume measurement (numerical)
17. white blood cell count: White blood cell count measurement in cells/cumm (numerical)
18. red blood cell count: Red blood cell count measurement in millions/cmm (numerical)
19. hypertension: Presence of hypertension (categorical: yes, no)
20. diabetes mellitus: Presence of diabetes mellitus (categorical: yes, no)
21. coronary artery disease: Presence of coronary artery disease (categorical: yes, no)

22. appetite: Patient's appetite status (categorical: good, poor)
23. pedal edema: Presence of pedal edema (categorical: yes, no)
24. anemia: Presence of anemia (categorical: yes, no)
25. class: Class label indicating the presence or absence of chronic kidney disease (categorical: yes, no)

The dataset contains both numerical and categorical features, and the class column represents the target variable. It is a supervised learning problem. The goal is to build a machine-learning model that can accurately predict the presence or absence of chronic kidney disease based on the given features.

Project Design:

Tools and Framework:

Python: The computer programming language used to put the code into action. Due to its ease of use and a robust ecosystem of libraries, Python is frequently employed in data analysis and machine learning.

Pandas: Python's strong data analysis and manipulation library. Pandas offers many data preparation operations and data structures for working effectively with structured data, such as data frames.

NumPy: It is a foundational Python package for numerical computing. To execute calculations on numerical data, NumPy offers multidimensional array objects, mathematical functions, and linear algebra procedures.

Scikit-learn: This is pythons well-liked machine learning library. Regression, classification, clustering, and model evaluation are just a few of the machine learning tasks that Scikit-learns tools and methods can be used for. Additionally, it provides tools for choosing models and preprocessing data.

Seaborn and Matplotlib: These are Python visualization libraries. On top of Matplotlib, Seaborn offers a higher-level interface for producing visually appealing and educational statistics visuals. A variety of plots and visualizations can be made using the flexible charting toolkit Matplotlib.

Jupyter Notebook: Jupyter Notebook, an interactive notebook environment that enables integrating code, visualizations, and documentation, is where the code can be run.

A whole range of functions for data analysis, preprocessing, visualization, and machine learning modelling are offered by these tools and frameworks. They are ideal for a variety of data-driven initiatives because of their high level of flexibility and simplicity of usage.

The program's functionality can be summarised as follows:

Data Loading and Preprocessing:

We import all the necessary libraries that will be used in project then using the panda’s library, the code first loads the "kidney_disease.csv" dataset.

importing the libraries

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from numpy.random import RandomState
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, accuracy_score, confusion_matrix, classification_report
```

Loading the dataset

```
# Load the dataset
data = pd.read_csv('kidney_disease.csv').drop('id', axis=1)
data.dtypes
```

ut[3]:

```
age               float64
bp                float64
sg                float64
al                float64
su                float64
rbc                object
pc                 object
pcc                object
ba                 object
bgr               float64
bu                float64
sc                float64
sod               float64
pot               float64
hemo              float64
pcv                object
wc                 object
rc                 object
htn                object
dm                 object
cad                object
appet              object
pe                 object
ane                object
classification     object
dtype: object
```

As we can see most of the attributes have an object as their data type. So, we need to convert those fields. We convert those categorical attributes using label encoder in to numerical attributes as shown below.

Pre-processing the data

```
# Convert categorical variables to numerical
le = preprocessing.LabelEncoder()
categorical_cols = ["rbc","pc","pcc","ba","pcv","wc","rc","htn","dm","cad","appet","pe","ane","classification"]
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])
    print(le.classes_)
```

```
['abnormal' 'normal' nan]
['abnormal' 'normal' nan]
['notpresent' 'present' nan]
['notpresent' 'present' nan]
['\t43' '\t?' '14' '15' '16' '17' '18' '19' '20' '21' '22' '23' '24' '25'
 '26' '27' '28' '29' '30' '31' '32' '33' '34' '35' '36' '37' '38' '39'
 '40' '41' '42' '43' '44' '45' '46' '47' '48' '49' '50' '51' '52' '53'
 '54' '9' nan]
['\t6200' '\t8400' '\t?' '10200' '10300' '10400' '10500' '10700' '10800'
 '10900' '11000' '11200' '11300' '11400' '11500' '11800' '11900' '12000'
 '12100' '12200' '12300' '12400' '12500' '12700' '12800' '13200' '13600'
 '14600' '14900' '15200' '15700' '16300' '16700' '18900' '19100' '21600'
 '2200' '2600' '26400' '3800' '4100' '4200' '4300' '4500' '4700' '4900'
 '5000' '5100' '5200' '5300' '5400' '5500' '5600' '5700' '5800' '5900'
 '6000' '6200' '6300' '6400' '6500' '6600' '6700' '6800' '6900' '7000'
 '7100' '7200' '7300' '7400' '7500' '7700' '7800' '7900' '8000' '8100'
 '8200' '8300' '8400' '8500' '8600' '8800' '9000' '9100' '9200' '9300'
 '9400' '9500' '9600' '9700' '9800' '9900' nan]
['\t?' '2.1' '2.3' '2.4' '2.5' '2.6' '2.7' '2.8' '2.9' '3' '3.1' '3.2'
 '3.3' '3.4' '3.5' '3.6' '3.7' '3.8' '3.9' '4' '4.1' '4.2' '4.3' '4.4'
 '4.5' '4.6' '4.7' '4.8' '4.9' '5' '5.1' '5.2' '5.3' '5.4' '5.5' '5.6'
 '5.7' '5.8' '5.9' '6' '6.1' '6.2' '6.3' '6.4' '6.5' '8' nan]
['no' 'yes' nan]
['\tno' '\tyes' ' yes' 'no' 'yes' nan]
['\tno' 'no' 'yes' nan]
['good' 'poor' nan]
```

After converting the variables to numerical we can notice that their datatype changes to int32 as shown below.

```
1 data.dtypes
```

```
]: age                float64
   bp                 float64
   sg                 float64
   al                 float64
   su                 float64
   rbc                  int32
   pc                   int32
   pcc                  int32
   ba                   int32
   bgr                float64
   bu                 float64
   sc                 float64
   sod                float64
   pot                float64
   hemo               float64
   pcv                  int32
   wc                   int32
   rc                   int32
   htn                  int32
   dm                   int32
   cad                  int32
   appet                int32
   pe                   int32
   ane                  int32
   classification       int32
   dtype: object
```

Now we will check whether there are any null values. As we can see below in the 2nd column, we have a null value. So, we fill the null values with median.

```
1 data.head(5)
```

6]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	48.0	80.0	1.020	1.0	0.0	2	1	0	0	121.0	...	32	72	31	1	4	1	0	0	0	0
1	7.0	50.0	1.020	4.0	0.0	2	1	0	0	NaN	...	26	56	46	0	3	1	0	0	0	0
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	...	19	70	46	0	4	1	1	0	1	0
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	20	62	18	1	3	1	1	1	1	0
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	23	68	25	0	3	1	0	0	0	0

5 rows × 25 columns

```
1 # filling null values
2 for col in data.columns:
3     data[col].fillna(data[col].median(),inplace=True)
4 data.head(5)
```

7]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	48.0	80.0	1.020	1.0	0.0	2	1	0	0	121.0	...	32	72	31	1	4	1	0	0	0	0
1	7.0	50.0	1.020	4.0	0.0	2	1	0	0	121.0	...	26	56	46	0	3	1	0	0	0	0
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	...	19	70	46	0	4	1	1	0	1	0
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	20	62	18	1	3	1	1	1	1	0
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	23	68	25	0	3	1	0	0	0	0

5 rows × 25 columns

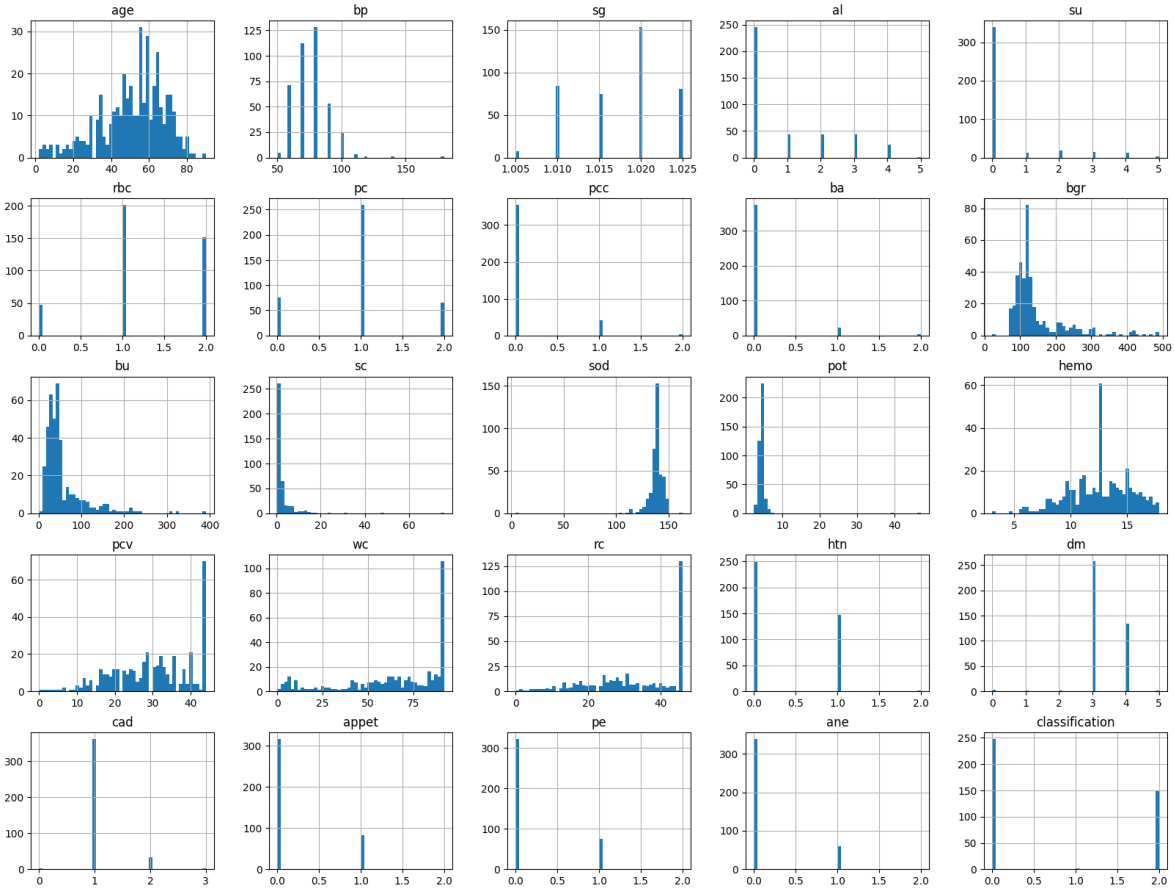
Now, we will be checking whether there are any null values using isnull() function. As, shown below there are 0 null values.

```
# checking whether are any null values
data.isnull().sum()
```

```
age               0
bp                0
sg                0
al                0
su                0
rbc               0
pc                0
pcc               0
ba                0
bgr               0
bu                0
sc                0
sod               0
pot               0
hemo              0
pcv               0
wc                0
rc                0
htn               0
dm                0
cad               0
appet             0
pe                0
ane               0
classification    0
dtype: int64
```

To show how the variables in the dataset are distributed, we create histograms.

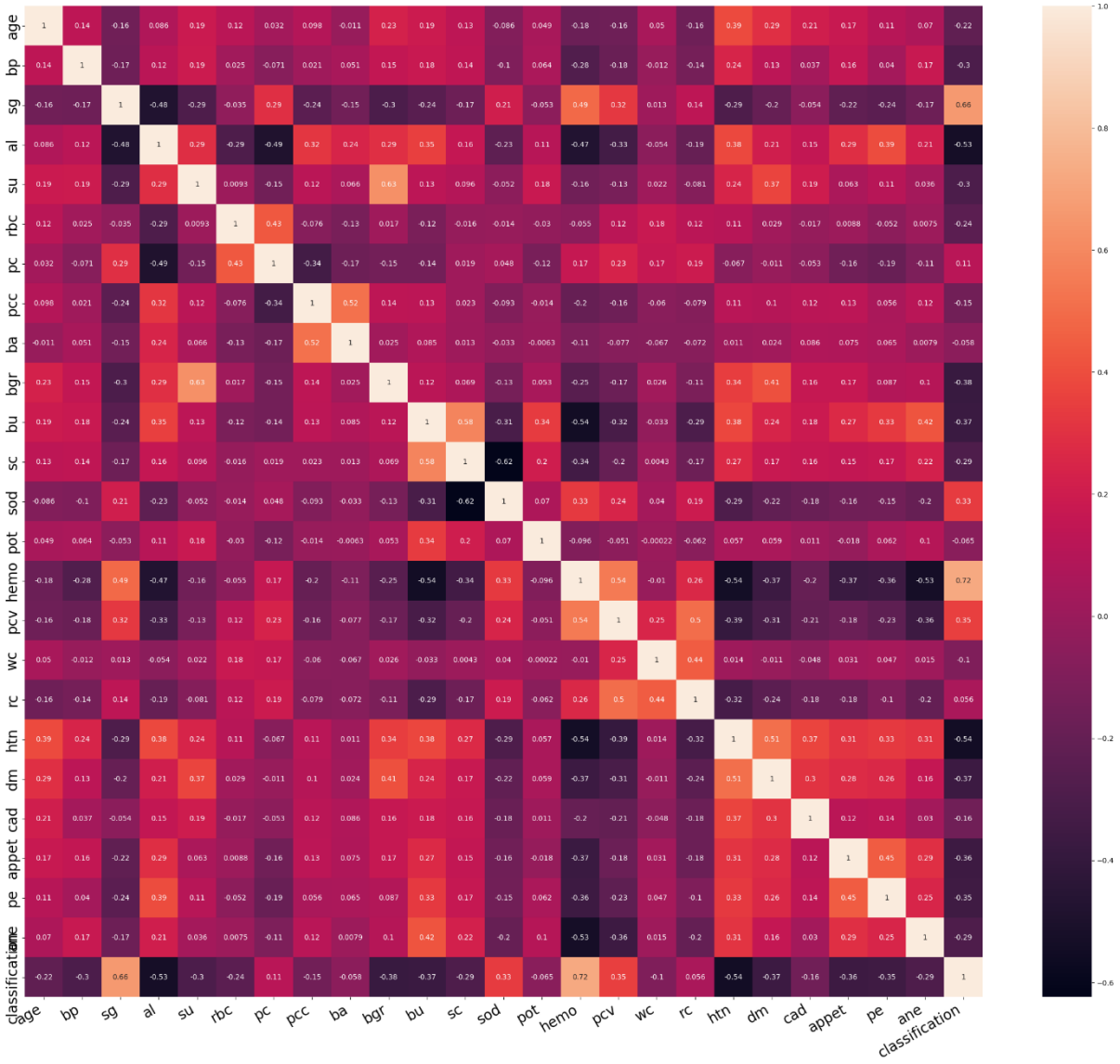
```
data.hist(bins=50,figsize=(20,15))
plt.show()
```



To see how different variables relate to one another, a correlation matrix is generated and shown as a heatmap. Each heatmap cell's colour denotes the degree and direction of the link. Darker colours are used to display negative correlations whereas lighter colours are used to display positive correlations. The real correlation coefficient is provided by the annotation values in each cell.

```
correlation_figure, correlation_axis = plt.subplots(figsize = (30,25))
corr_mtrx = data.corr()
correlation_axis = sns.heatmap(corr_mtrx, annot= True)

plt.xticks(rotation = 30, horizontalalignment = 'right', fontsize = 20)
plt.yticks(fontsize = 20)
plt.show()
```



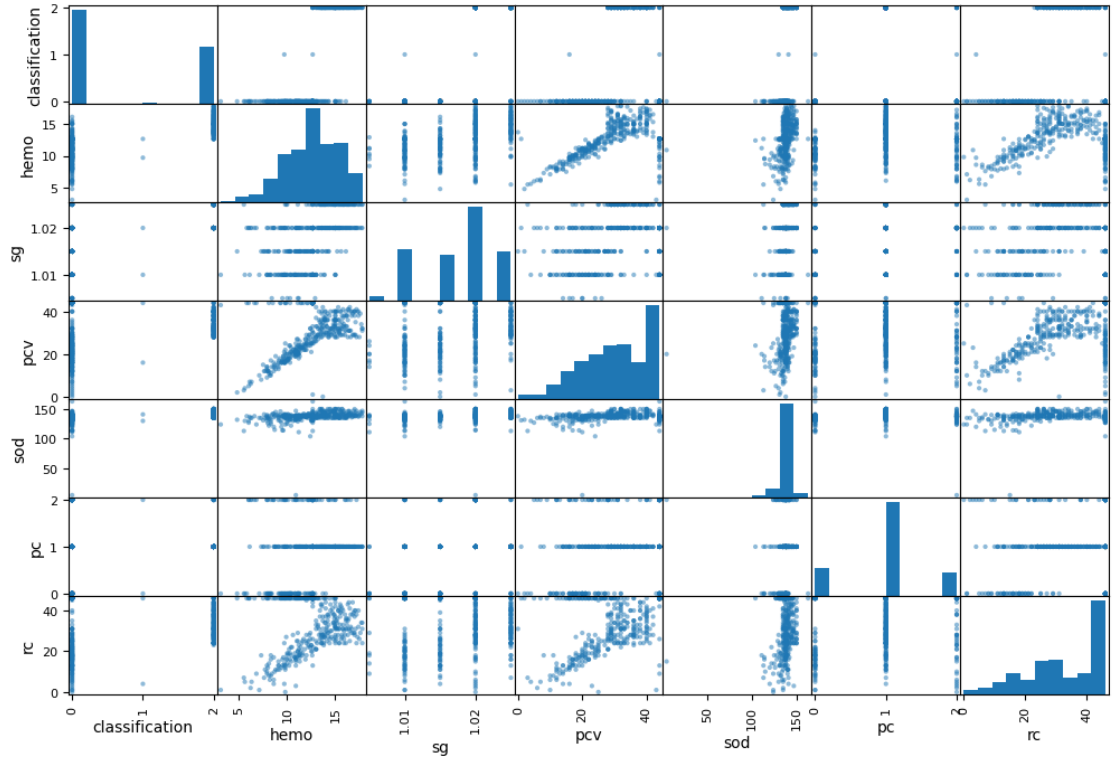
The dataset's columns are sorted in descending order after the correlation coefficients between the 'classification' column and all other columns are calculated. The higher the correlation coefficient, the stronger the linear relationship between the variables.

```
corr_matrix = data.corr()
corr_matrix['classification'].sort_values(ascending=False)
```

```
classification    1.000000
hemo              0.724742
sg                0.657810
pcv               0.349749
sod               0.334153
pc                0.114674
rc                0.055589
ba               -0.057511
pot              -0.065448
wc               -0.102709
pcc              -0.149153
cad              -0.155347
age              -0.222985
rbc              -0.238042
sc               -0.292050
ane              -0.293151
bp               -0.296613
su               -0.296919
pe               -0.352622
appet            -0.359114
dm               -0.366016
bu               -0.371536
bgr              -0.378495
al               -0.531885
htn              -0.543271
Name: classification, dtype: float64
```

The pairwise correlations between the chosen columns are displayed in the scatter matrix plot, which is a grid of scatter plots. With one variable on the x-axis and the other on the y-axis, each scatter plot depicts the relationship between two variables. The scatter matrix's diagonal shows the histograms for each variable. We can explore the relationships between the variables in the dataset visually by making a scatter matrix plot.

```
attributes = ['classification','hemo','sg','pcv','sod','pc','rc']
scatter_matrix(data[attributes],figsize=(12,8))
```



By randomly selecting 70% of the dataset as the training set and using a random number generator, the code splits the data. The test set is given the remaining 30% of the data. The reproducibility of the splitting process is ensured by this method. The code chooses the rows from the initial dataset that are absent from the training set using the "~" operator and the "isin" function, so producing the test set.

```
# Spliting entire data into Train and Test Segments
rng = RandomState()

train = data.sample(frac=0.7, random_state=rng)
test = data.loc[~data.index.isin(train.index)]
```

```
train.head(5)
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
37	72.0	80.0	1.020	0.0	0.0	2	2	0	0	137.0	...	16	64	4	1	4	1	1	0	1	1
177	65.0	80.0	1.015	2.0	1.0	1	1	1	0	215.0	...	29	92	46	0	4	1	0	0	0	0
20	61.0	80.0	1.015	2.0	0.0	0	0	0	0	173.0	...	12	84	11	1	4	2	1	1	1	0
334	24.0	80.0	1.025	0.0	0.0	1	1	0	0	125.0	...	31	52	24	0	3	1	0	0	0	2
361	29.0	80.0	1.020	0.0	0.0	1	1	0	0	70.0	...	42	50	37	0	3	1	0	0	0	2

5 rows × 25 columns

```
test.head(5)
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	48.0	80.0	1.020	1.0	0.0	2	1	0	0	121.0	...	32	72	31	1	4	1	0	0	0	0
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	...	20	62	18	1	3	1	1	1	1	0
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	...	23	68	25	0	3	1	0	0	0	0
7	24.0	80.0	1.015	2.0	4.0	1	0	0	0	410.0	...	32	64	29	0	4	1	0	1	0	0
16	47.0	70.0	1.015	2.0	0.0	2	1	0	0	99.0	...	44	92	46	0	3	1	0	0	0	0

The 'label' variable is given the 'classification' column, which is taken from the 'data' DataFrame as this is the target variable. The 'classification' column is then removed from the 'data' DataFrame, creating a new DataFrame that does not have the 'classification' column.

```
data = train[attributes]
data.head(5)
```

	classification	hemo	sg	pcv	sod	pc	rc
37	1	9.7	1.020	16	141.0	2	4
177	0	13.2	1.015	29	138.0	1	46
20	0	7.7	1.015	12	135.0	0	11
334	2	15.4	1.025	31	136.0	1	24
361	2	13.7	1.020	42	138.0	1	37

```
label = data['classification']
data = data.drop('classification',axis=1)
data.head(5)
```

	hemo	sg	pcv	sod	pc	rc
37	9.7	1.020	16	141.0	2	4
177	13.2	1.015	29	138.0	1	46
20	7.7	1.015	12	135.0	0	11
334	15.4	1.025	31	136.0	1	24
361	13.7	1.020	42	138.0	1	37

The train_test_split function from sci-kit-learn is used to divide the dataset into training and testing sets. 70% of the data are in the training set, and the other 30% are in the testing set.

Two arguments are provided to the display_results function: y_test (the actual target values)

Split the data into train and test sets

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.2, random_state=42)
```

```
def display_results(y_test,y_pred):
    # Print the Confusion Matrix and slice it into four pieces
    cm = confusion_matrix(y_test,y_pred)
    # visualize confusion matrix with seaborn heatmap
    cm_matrix = pd.DataFrame(data=cm)
    print("Model Accuracy:",accuracy_score(y_test,y_pred))
    sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

and y_pred (the anticipated target values). The confusion matrix is calculated using the confusion_matrix function, and the heatmap plot produced by sns.heatmap is used to visualize the confusion matrix. Additionally, the accuracy_score function's model accuracy calculation is printed by the function.

1. Linear Regression:

Linear regression is a supervised learning algorithm used for predicting continuous numerical values. By applying a linear equation to the data, it models the relationship between the input features and the goal variable.

Implementation: The code trains a linear regression model using the LinearRegression class from sklearn.linear_model module. It predicts the target variable for the test data using the predict() function and fits the model to the training data using the fit() method.

Evaluation Metrics: As evaluation metrics for the linear regression model, the code computes and outputs Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R2).

1. Linear Regression

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
# Linear Regression
reg_model = LinearRegression()
reg_model.fit(X_train, y_train)
reg_pred = reg_model.predict(X_test)
```

```
# Calculate metrics
mse = mean_squared_error(y_test, reg_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, reg_pred)
r2 = r2_score(y_test, reg_pred)

# Print metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
```

```
Mean Squared Error (MSE): 0.2541119322263674
Root Mean Squared Error (RMSE): 0.5040951618755802
Mean Absolute Error (MAE): 0.4216466193233008
R-squared (R²): 0.7336580817306524
```

2. Logistic Regression:

For binary classification tasks, the supervised learning algorithm logistic regression is utilized. A logistic function is fitted to the data to model the association between the input features and the binary target variable.

Implementation: To train a logistic regression model, the code makes use of the LogisticRegression class found in the sklearn.linear_model module. The fit() method is used to fit the model to the training data, while the predict() method is used to forecast the target variable for the test data.

Metrics for evaluation for the logistic regression model, the code computes and displays the accuracy score and classification report, which includes precision, recall, F1-score, and support.

2. Logistic Regression

```
# Logistic Regression
clf_model = LogisticRegression(random_state=0, max_iter=2000, solver='liblinear')
clf_model.fit(X_train, y_train)
clf_pred = clf_model.predict(X_test)
# Calculate accuracy and display results
clf_accuracy = accuracy_score(y_test, clf_pred)
print("Logistic Regression Accuracy:", clf_accuracy)
print(classification_report(y_test, clf_pred))
```

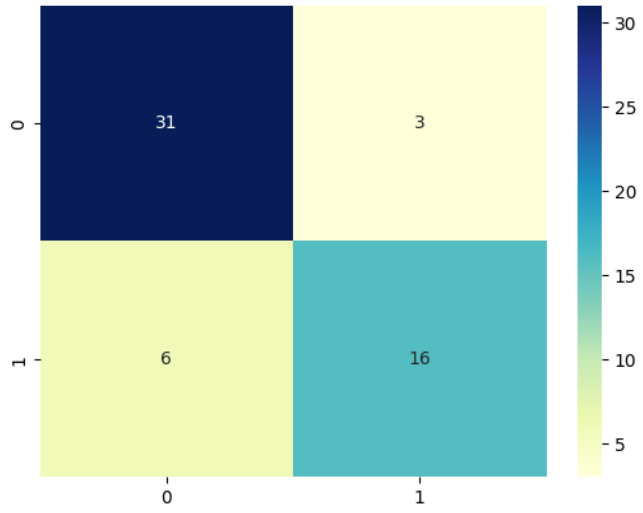
```
Logistic Regression Accuracy: 0.8392857142857143
              precision    recall  f1-score   support

           0       0.84      0.91      0.87        34
           2       0.84      0.73      0.78        22

    accuracy                           0.84        56
   macro avg       0.84      0.82      0.83        56
weighted avg       0.84      0.84      0.84        56
```

```
display_results(y_test,clf_pred)
```

```
Model Accuracy: 0.8392857142857143
```



3. Navie Bayes:

Naive Bayes is a supervised learning technique that relies on the Bayes theorem and presupposes feature independence. It is frequently employed for classification problems and is effective with both category and textual data.

Implementation: The code trains a Naive Bayes model using the GaussianNB class from the sklearn.naive_bayes module. It predicts the target variable for the test data using the predict() function and fits the model to the training data using the fit() method.

Evaluation Metrics: The code generates a classification report and calculates the accuracy score for the Naive Bayes model.

3. Navie Bayes

]:
```
# Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_pred = nb_model.predict(X_test)
# Calculate accuracy and display results
nb_accuracy = accuracy_score(y_test, nb_pred)
print("Naive Bayes Accuracy:", nb_accuracy)
print(classification_report(y_test, nb_pred))
```

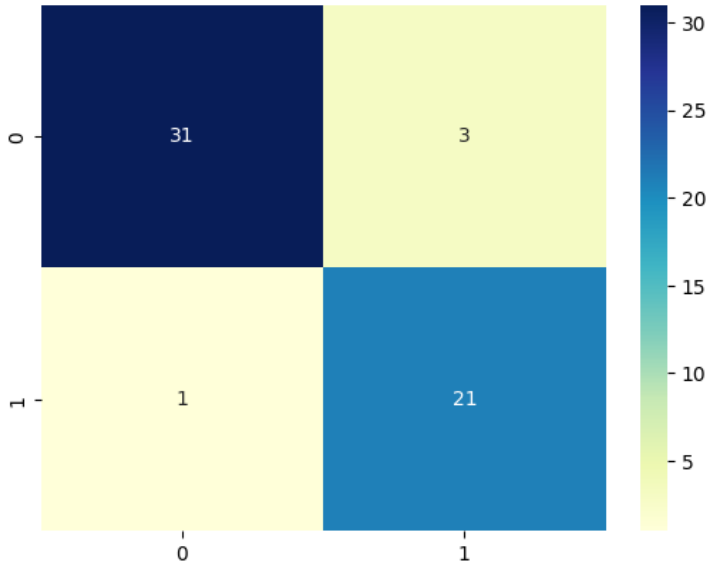
```
Naive Bayes Accuracy: 0.9285714285714286
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        34
           2       0.88      0.95      0.91        22

    accuracy                           0.93        56
   macro avg       0.92      0.93      0.93        56
weighted avg       0.93      0.93      0.93        56
```

]:
```
display_results(y_test,nb_pred)
```

```
Model Accuracy: 0.9285714285714286
```



4. Random Forest:

Random Forest is an ensemble learning technique that blends different decision trees to generate predictions. For classification and regression applications, it is a flexible and strong method.

Application: To train a random forest model, the code makes use of the RandomForestClassifier class from the sklearn.ensemble module. The fit() method is used to fit the model to the training data, while the predict() method is used to forecast the target variable for the test data.

Metrics for evaluation the random forest model's accuracy score and classification report are computed and printed using display_results.

4. Random Forest

```
# Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
# Calculate accuracy and display results
rf_accuracy = accuracy_score(y_test, rf_pred)
print("Random Forest Accuracy:", rf_accuracy)
print(classification_report(y_test, rf_pred))
```

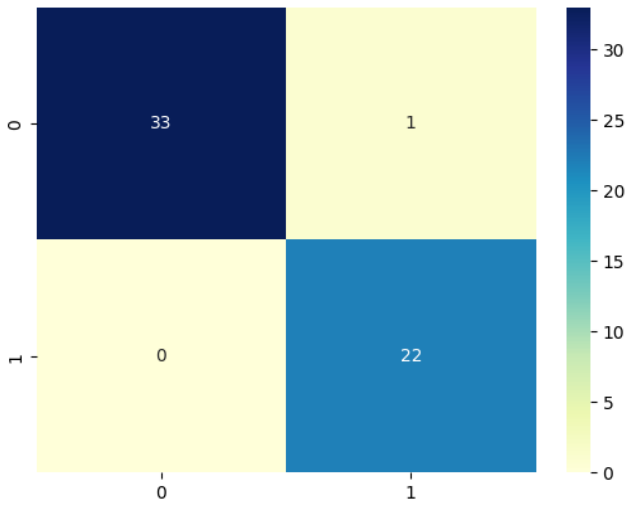
```
Random Forest Accuracy: 0.9821428571428571
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        34
           2       0.96      1.00      0.98        22

    accuracy                           0.98        56
   macro avg       0.98      0.99      0.98        56
weighted avg       0.98      0.98      0.98        56
```

```
display_results(y_test,rf_pred)
```

```
Model Accuracy: 0.9821428571428571
```



5. Support Vector Machine:

Support Vector Machine is a supervised learning method used for regression and classification tasks. It locates a hyperplane in a high-dimensional feature space that categorises the data points.

Implementation: To train an SVM model, the code makes use of the SVC class from the sklearn.svm module. The fit() method is used to fit the model to the training data, while the predict() method is used to forecast the target variable for the test data.

Evaluation Metrics: The SVM model's accuracy score and classification report are computed and printed using display results function.

5. Support Vector Machine (SVM)

```
from sklearn.svm import SVC
from sklearn import svm

svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train,y_train)
svm_model_score = round(svm_model.score(X_train, y_train) * 100, 2)
svm_pred = svm_model.predict(X_test)
# Calculate accuracy and display results
svm_accuracy = accuracy_score(y_test, svm_pred)
print("Support Vector Machine Accuracy:",svm_accuracy)
print(classification_report(y_test, svm_pred))
```

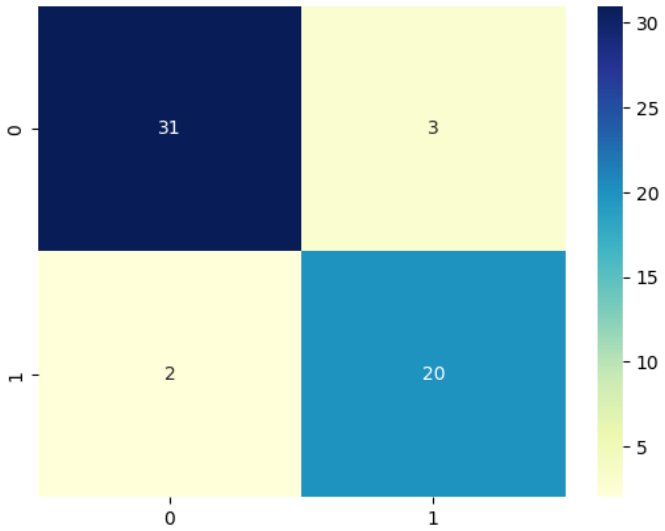
```
Support Vector Machine Accuracy: 0.9107142857142857
              precision    recall  f1-score   support

           0       0.94      0.91      0.93        34
           2       0.87      0.91      0.89        22

    accuracy                           0.91        56
   macro avg       0.90      0.91      0.91        56
weighted avg       0.91      0.91      0.91        56
```

```
display_results(y_test,svm_pred)
```

```
Model Accuracy: 0.9107142857142857
```



The below code compares and displays all of the models' accuracy scores in tabular style.

The models are sorted according to their accuracy ratings, enabling the user to choose the best model.

As we can see Random Forest has a high accuracy of 98% when compared to other models.

Comparing all the models

34]:

```
# Comparing all the models
models = pd.DataFrame({
    'Model': [ 'Linear Regression', 'Logistic Regression', 'Naive Bayes','Random Forest','SVM Model'],
    'Model Accuracy': [r2, clf_accuracy, nb_accuracy, rf_accuracy, svm_accuracy]})
models.sort_values(by='Model Accuracy', ascending=False)
```

Out[34]:

	Model	Model Accuracy
3	Random Forest	0.982143
2	Naive Bayes	0.928571
4	SVM Model	0.910714
1	Logistic Regression	0.839286
0	Linear Regression	0.733658

Important Functions/Methods:

display_results(y_test, y_pred): This function computes the accuracy score for the expected results and displays the confusion matrix. To clearly explain the model's performance, it uses a heatmap to visualize the confusion matrix.

LinearRegression.fit(X_train, y_train): This technique allows the linear regression model to learn the correlation between the input characteristics and the target variable by fitting it to the training data.

LogisticRegression.fit(X_train, y_train): Using this approach, the logistic regression model is fitted to the training data so that it can identify underlying patterns and predict outcomes using the input features.

GaussianNB.fit(X_train, y_train): Using the training data, this method trains the Naive Bayes model, allowing it to understand the probability distribution of the features and predict the future based on the input data.

RandomForestClassifier.fit(X_train, y_train): This technique trains numerous decision trees and combines their predictions to produce precise classifications, fitting the random forest model to the training data.

SVC.fit(X_train, y_train): This technique finds the ideal hyperplane that categorises the data points into different groups by training the Support Vector Machine model on the training data.

Project Milestones:

1. Milestone: Loading and Preprocessing the Dataset

- Loading the kidney disease dataset
- Converting categorical variables to numerical values
- Handling missing values by filling them with the median

2. Milestone: Exploratory Data Analysis (EDA)

- Visualizing data through histograms
- Creating a correlation matrix and heatmap
- Generating scatter plots for selected attributes

3. Milestone: Splitting Data into Training and Testing Sets

- Using a random state to ensure reproducibility
- Splitting the data into 70% for training and 30% for testing

4. Milestone: Implementing Machine Learning Algorithms

- Linear Regression: Training a linear regression model and evaluating metrics
- Logistic Regression: Training a logistic regression model and evaluating metrics
- Naive Bayes: Training a naive Bayes model and evaluating metrics
- Random Forest: Training a random forest model and evaluating metrics
- Support Vector Machine (SVM): Training an SVM model and evaluating metrics

5. Milestone: Evaluation and Comparison

- Calculating evaluation metrics such as accuracy, mean squared error, and mean absolute error for each model
- Generating a confusion matrix and displaying it as a heatmap
- Printing a classification report for each algorithm
- Comparing the accuracy of different models

6. Milestone: Testing on Separate Test Dataset

- Using the trained models to make predictions on the separate test dataset
- Displaying the evaluation metrics and confusion matrix for the test dataset

Throughout the project, incremental features were added, including:

- Loading and preprocessing the dataset
- Conducting exploratory data analysis
- Implementing and evaluating different machine learning algorithms
- Generating evaluation metrics and confusion matrix
- Testing the models on a separate test dataset

These milestones represent key steps in the project, gradually building upon each other to analyze the kidney disease dataset and assess the performance of various machine-learning models.

Project Results:

The project's objectives included the analysis of a dataset related to kidney disease and the comparison of the effectiveness of several machine-learning models. The outcomes demonstrated that, among the evaluated models, random forest and navie bayes had the highest accuracy. The classification of kidney problems could be predicted by the models with a respectable degree of accuracy.

]:

	Model	Model Accuracy
3	Random Forest	0.982143
2	Naive Bayes	0.928571
4	SVM Model	0.910714
1	Logistic Regression	0.839286
0	Linear Regression	0.733658

The following actions can be carried out with more time and/or information:

- Investigate additional feature transformations or combinations to increase the models' capacity for prediction.
- Hyperparameter tuning: To identify the optimal combination, optimize the models' hyperparameters using methods like grid search or random search.
- Ensemble methods: Use ensemble techniques like stacking or boosting to aggregate the output of various models and maybe improve performance.
- Cross-validation: Use cross-validation to improve the accuracy of your model performance estimates and to lessen data fitting.
- Increase the amount of data you collect to assist the models generalize and produce more precise predictions.
- Analyse feature importance carefully to determine the characteristics that are most useful in predicting kidney disease.

The performance of the models can be further improved by putting these recommendations into practice, leading to more precise forecasts and a greater comprehension of the variables impacting renal disease.

Repository / Archive:

https://github.com/Kavya9199/INFO_5502.git

Appendix: code of the project “Chronic_Kidney_Disease_Prediction”

kidney_disease_code

Reference Material:

M. A. Islam, S. Akter, M. S. Hossen, S. A. Keya, S. A. Tisha and S. Hossain, "Risk Factor Prediction of Chronic Kidney Disease based on Machine Learning Algorithms," 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, 2020, pp. 952-957, doi: 10.1109/ICISS49785.2020.9315878.

Wang, W., Chakraborty, G., & Chakraborty, B. (2020). Predicting the Risk of Chronic Kidney Disease (CKD) Using Machine Learning Algorithm. *Applied Sciences*, *11*(1), 202. MDPI AG. Retrieved from http://dx.doi.org/10.3390/app11010202

mygreatlearning.com was first indexed by Google in July 2019 https://www.mygreatlearning.com/blog/most-used-machine-learning-algorithms-in-python/