#### **MODESOLA GIWA**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import os
from sklearn.model selection import train test split
from sklearn.linear model import LogisticRegression
from sklearn.metrics import confusion matrix, accuracy score
from sklearn.metrics import roc auc score
from google.colab import drive
drive.mount('/content/drive')
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force remount=True).
data dir= "/content/drive/MyDrive/Machine Learning"
file = "chronic kidney disease full.csv"
file = os.path.join(data dir, file)
dataset = pd.read csv(file)
dataset.head()
{"type": "dataframe", "variable name": "dataset"}
```

# **Exploratory Data Analysis**

```
dataset.dtypes
dataset.shape

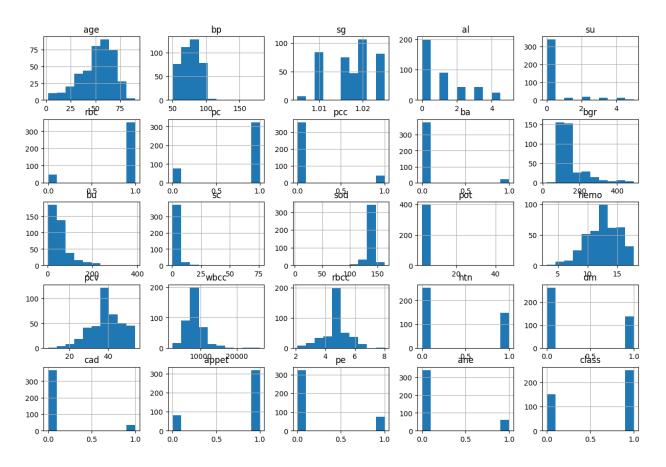
(400, 25)

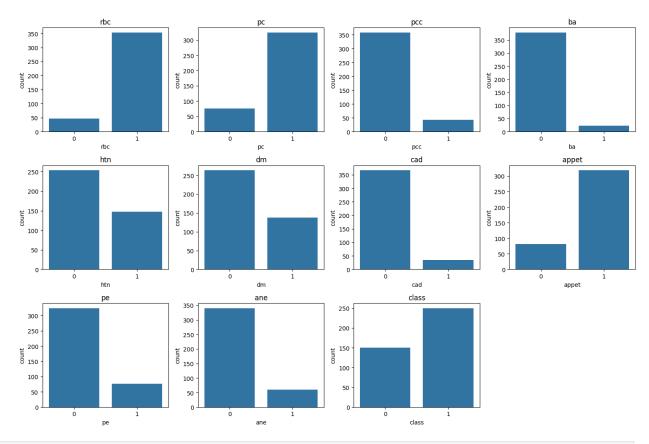
for column in dataset.columns:
    if dataset[column].dtype == 'object':
        mode_value = dataset[column].mode()[0]
        dataset[column].fillna(mode_value, inplace=True)
    else:
        mean_value = dataset[column].mean()
        dataset[column].fillna(mean_value, inplace=True)

replacement_mapping = {
    'normal': 1, 'abnormal': 0,
    'present': 1, 'notpresent': 0,
    'yes': 1, 'no': 0,
    'good': 1, 'poor': 0,
```

```
'ckd': 1, 'notckd': 0
}
categorical = [col for col, dtype in dataset.dtypes.items() if dtype
== 'object']
for column in categorical:
    dataset[column] = dataset[column].replace(replacement mapping)
# Histograms
dataset.hist(figsize=(15, 10))
plt.suptitle('Histograms of Numerical Variables', fontsize=16)
plt.show()
# Bar charts for categorical variables
categorical_vars = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad',
'appet', 'pe', 'ane', 'class']
plt.figure(figsize=(15, 10))
for i, col in enumerate(categorical_vars, 1):
    plt.subplot(3, 4, i)
    sns.countplot(x=col, data=dataset)
    plt.title(col)
plt.tight layout()
plt.show()
```

#### Histograms of Numerical Variables





f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(dataset.corr(), annot = True)

<Axes: >

```
g- 1 0.15 -0.18 0.11 0.21 -0.012 -0.1 0.16 0.044 0.21 0.19 0.13 -0.086 0.05 -0.18 -0.21 0.1 -0.2 0.39 0.36 0.23 -0.16 0.094 0.051 0.23
   0.15 1 0.18 0.15 0.2 0.15 0.16 0.06 0.11 0.15 0.18 0.14 0.15 0.06 0.11 0.15 0.18 0.14 0.15 0.067 0.28 0.29 0.026 0.22 0.27 0.23 0.087 0.18 0.054 0.19 0.29
97 - 0.18 0.18 1 0.47 0.29 0.24 0.35 0.29 0.22 0.33 0.28 0.21 0.24 0.062 0.53 0.53 0.53 0.53 0.53 0.20 0.47 0.37 0.39 0.15 0.25 0.27 0.21 0.7
та - 0.11 0.15 -0.47 1 0.27 -0.37 -0.54 0.4 0.37 0.33 0.41 0.23 -0.27 0.11 -0.55 -0.53 0.2 -0.45 0.48 0.38 0.24 -0.33 0.44 0.28
g - 0.21 0.2 -0.29 0.27 1 -0.074 -0.17 0.15 0.11 0.64 0.15 0.13 -0.074 0.18 -0.19 -0.2 0.15 -0.18 0.29 0.47 0.25 -0.081 0.13 0.066 0.33
g -0.012 -0.15 -0.24 -0.37 -0.074 1 0.38 -0.1 -0.18 -0.15 -0.24 -0.14 -0.14 -0.18 -0.28 -0.28 -0.022 -0.2 -0.14 -0.15 -0.11 -0.16 -0.2 -0.11 -0.28
   -0.1 -0.16 0.35 -0.54 -0.17 0.38 1 -0.52 -0.33 -0.26 -0.34 -0.16 0.17 -0.16 0.41 0.42 -0.11 0.38 -0.29 -0.2 -0.17 0.27 -0.35 -0.26 -0.38
g - 0.16 0.06 -0.29 0.4 0.15 -0.1 -0.52 1 0.28 0.2 0.18 0.05 -0.14 -0.0063 -0.28 0.29 0.16 -0.27 0.2 0.17 0.19 -0.19 0.1 0.18 0.27
g - 0.044 0.11 -0.22 0.37 0.11 -0.18 -0.33 0.28 1 0.086 0.16 0.051 -0.082 -0.0027 -0.2 -0.19 0.1 -0.19 0.089 0.08 0.16 -0.15 0.13 0.052 0.19
5 - 0.21 0.15 -0.33 0.33 0.64 -0.15 -0.26 0.2 0.086 1 0.13 0.082 -0.15 0.057 -0.27 -0.27 0.12 -0.22 0.37 0.5 0.21 -0.18 0.1 0.14 0.1
B - 0.19 0.18 -0.28 0.41 0.15 -0.24 -0.34 0.18 0.16 0.13 1 0.58 -0.31 0.34 -0.54 -0.53 0.042 -0.47 0.39 0.31 0.22 -0.27 0.34 0.44 0.37
N - 0.13 0.14 0.21 0.23 0.13 -0.14 -0.16 0.05 0.051 0.082 0.58 1 -0.62 0.21 -0.34 -0.34 -0.0054 -0.32 0.27 0.21 0.19 -0.16 0.18 0.24 0.29
8 -0.086 -0.1 0.24 -0.27 -0.074 0.14 0.17 -0.14 -0.082 -0.15 -0.31 -0.62 1 0.067 0.33 0.35 0.0063 0.32 -0.31 -0.27 -0.22 0.16 -0.15 -0.2 -0.34
g - 0.05 0.067 -0.062 0.11 0.18 0.018 -0.16 -0.00630.0027 0.057 0.34 0.21 0.067 1 -0.1 -0.12 -0.074 -0.12 0.057 0.061 0.0058 0.017 0.061 0.1 0.077
E - 0.18 - 0.28 | 0.53 | - 0.55 | - 0.19 | | 0.28 | 0.41 | - 0.28 | - 0.2 | - 0.27 | - 0.54 | - 0.34 | | 0.33 | - 0.1 | 1 | 0.85 | - 0.15 | 0.68 | - 0.58 | - 0.47 | - 0.28 | | 0.39 | - 0.38 | - 0.56 | - 0.73 |
E - 0.21 0.29 0.53 0.53 0.53 0.2 0.28 0.42 0.29 0.19 0.27 0.53 0.34 0.35 0.12 0.85 1 0.18 0.7 0.57 0.45 0.29 0.39 0.39 0.51 0.69
   0.1 0.026 0.2 0.2 0.15 0.0022 0.11 0.16 0.1 0.12 0.042 0.00540.0063 0.074 0.15 0.18 1 0.15 0.12 0.15 0.0084 0.15 0.14 0.037 0.21
   -0.2 -0.2 -0.47 -0.45 -0.18 -0.2 -0.38 -0.27 -0.19 -0.22 -0.47 -0.32 -0.32 -0.12 -0.68 -0.7 -0.15 -1 -0.53 -0.42 -0.3 -0.4 -0.34 -0.4 -0.59
   0.39 0.27 -0.37 0.48 0.29 -0.14 -0.29 0.2 0.089 0.37 0.39 0.27 -0.31 0.057 -0.58 -0.57 0.12 -0.53 1 0.61 0.33 -0.35 0.37 0.35 0.55
E - 0.36 0.23 -0.39 0.38 0.47 -0.15 -0.2 0.17 0.08 0.5 0.31 0.21 -0.27 0.061 -0.47 -0.45 0.15 -0.42 0.61
                                                                                                                 0.27 -0.33 0.31 0.18 0.56
- 0.23 0.087 -0.15 0.24 0.25 -0.11 -0.17 0.19 0.16 0.21 0.22 0.19 -0.22 0.0058 -0.28 -0.29 0.0084 -0.3 0.33 0.27 1
8 - 0.16 -0.18 0.25 -0.33 -0.081 0.16 0.27 -0.19 -0.15 -0.18 -0.27 -0.16 0.16 0.017 0.39 0.39 -0.15 0.4 -0.35 -0.33 -0.16 1 -0.42 -0.25 -0.39
g - 0.094 0.054 -0.27 0.44 0.13 -0.2 -0.35 0.1 0.13 0.1 0.34 0.18 -0.15 0.061 -0.38 -0.39 0.14 -0.34 0.37 0.31 0.17 -0.42 1 0.21 0.38
g - 0.051 0.19 -0.21 0.28 0.066 -0.11 -0.26 0.18 0.052 0.13 0.44 0.24 -0.2 0.1 -0.56 -0.51 0.037 -0.4 0.35 0.18 0.048 -0.25 0.21 1
   0.23 0.29 -0.7
                    0.6 0.33 -0.28 -0.38 0.27 0.19 0.4 0.37 0.29 -0.34 0.077 -0.73 -0.69 0.21 -0.59
                                    pc pcc ba bgr bu sc sod pot hemo pcv wbcc rbcc htn dm cad appet pe ane class
```

- 0.8

- 0.6

- 0.4

- 0.2

- 0.0

- -0.2

- -0.4

-0.6

```
X = dataset.drop('class', axis=1)
y = dataset['class']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=0)
```

## Logistic Regression

```
log reg = LogisticRegression()
log_reg.fit(X_train, y train)
/usr/local/lib/python3.10/dist-packages/sklearn/linear model/
logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
Increase the number of iterations (max iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear model.html#logistic-
regression
  n iter i = check optimize result(
LogisticRegression()
coefficients = log reg.coef [0]
feature names = X.columns
for feature, coef in zip(feature names, coefficients):
    print(f'{feature}: {coef}')
age: -0.007784038055200967
bp: 0.06714440058001413
sg: 0.0028941830107993313
al: 0.1348601429710441
su: 0.020673223443322532
rbc: -0.00828740352254335
pc: -0.011099720211091912
pcc: 0.006846771780651671
ba: 0.0009851925327361392
bgr: 0.06879442066527669
bu: 0.03024556416005083
sc: 0.10389047272820492
sod: 0.07069921341634061
pot: 0.0005915892491862929
hemo: -0.2314395171324182
pcv: -0.5151617229976999
wbcc: 7.44169463020974e-05
rbcc: -0.05387813347114533
htn: 0.03893917050921311
dm: 0.03564330724599753
cad: 0.006345748345832803
appet: -0.022600633062232978
```

```
pe: 0.03390091953083066
ane: 0.010183193934151446

y_pred_lr = log_reg.predict(X_test)
```

### Random Forest

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X train = sc.fit transform(X train)
X test = sc.transform(X test)
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification report, accuracy score
# Initialize the Random Forest classifier
rf = RandomForestClassifier(n estimators=100, random state=42)
# Train the model
rf.fit(X train, y train)
# Predict on the testing set
y_pred_rf = rf.predict(X_test)
# Evaluate the model
classification report rf = classification report(y test, y pred rf)
accuracy rf = accuracy score(y test, y pred rf)
print(classification_report_rf)
print("Accuracy:", accuracy rf)
              precision
                           recall f1-score
                                               support
           0
                             1.00
                   1.00
                                        1.00
                                                    38
           1
                   1.00
                                        1.00
                             1.00
                                                    62
                                        1.00
                                                   100
    accuracy
                   1.00
                             1.00
                                        1.00
                                                   100
   macro avg
weighted avg
                   1.00
                             1.00
                                        1.00
                                                   100
Accuracy: 1.0
```

## **KNN**

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n neighbors=5)
knn.fit(X_train, y_train)
# Evaluate the model
from sklearn.metrics import classification report, accuracy score
y pred knn = knn.predict(X test)
print(classification_report(y_test, y_pred_knn))
print("Accuracy:", accuracy score(y test, y pred knn))
              precision
                           recall f1-score
           0
                   0.97
                              1.00
                                        0.99
                                                    38
           1
                   1.00
                              0.98
                                        0.99
                                                    62
    accuracy
                                        0.99
                                                   100
                   0.99
                              0.99
                                        0.99
                                                   100
   macro avq
weighted avg
                   0.99
                              0.99
                                        0.99
                                                   100
Accuracy: 0.99
```

## **Decision Trees**

```
from sklearn.tree import DecisionTreeClassifier
# Initialize the Decision Tree classifier
dt = DecisionTreeClassifier(random state=42)
# Train the model
dt.fit(X_train, y_train)
# Predict on the testing set
y pred dt = dt.predict(X test)
# Evaluate the model
classification report dt = classification report(y test, y pred dt)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(classification report dt)
print("Accuracy:", accuracy_dt)
              precision
                           recall f1-score
                                              support
           0
                                                    38
                   1.00
                             1.00
                                       1.00
           1
                   1.00
                             1.00
                                       1.00
                                                    62
```

```
100
                                      1.00
    accuracy
                                      1.00
                   1.00
                            1.00
                                                  100
   macro avg
weighted avg
                   1.00
                             1.00
                                      1.00
                                                  100
Accuracy: 1.0
from sklearn.metrics import confusion matrix, roc auc score,
roc curve, auc
# Confusion Matrices
cm knn = confusion matrix(y test, y pred knn)
cm_dt = confusion_matrix(y_test, y_pred_dt)
cm_rf = confusion_matrix(y_test, y_pred_rf)
cm lr = confusion matrix(y test, y pred lr)
# AUC Scores - Only if the output probabilities can be obtained,
otherwise skip
auc_knn = roc_auc_score(y_test, knn.predict_proba(X_test)[:, 1])
auc_dt = roc_auc_score(y test, dt.predict proba(X test)[:, 1])
auc rf = roc auc score(y test, rf.predict proba(X test)[:, 1])
auc lr = roc auc score(y test, log reg.predict proba(X test)[:, 1])
# Display results
print("Confusion Matrix (KNN):", cm_knn)
print("Confusion Matrix (Decision Tree):", cm dt)
print("Confusion Matrix (Random Forest):", cm rf)
print("Confusion Matrix (Logistic Regression):", cm lr)
print("AUC (KNN):", auc_knn)
print("AUC (Decision Tree):", auc dt)
print("AUC (Random Forest):", auc_rf)
print("AUC (Logistic Regression):", auc lr)
Confusion Matrix (KNN): [[38 0]
 [ 1 61]]
Confusion Matrix (Decision Tree): [[38 0]
 [ 0 6211
Confusion Matrix (Random Forest): [[38 0]
 [ 0 6211
Confusion Matrix (Logistic Regression): [[36 2]
 [ 4 58]]
AUC (KNN): 0.9919354838709677
AUC (Decision Tree): 1.0
AUC (Logistic Regression): 0.9804753820033957
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
LogisticRegression was fitted with feature names
  warnings.warn(
```