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"}
```

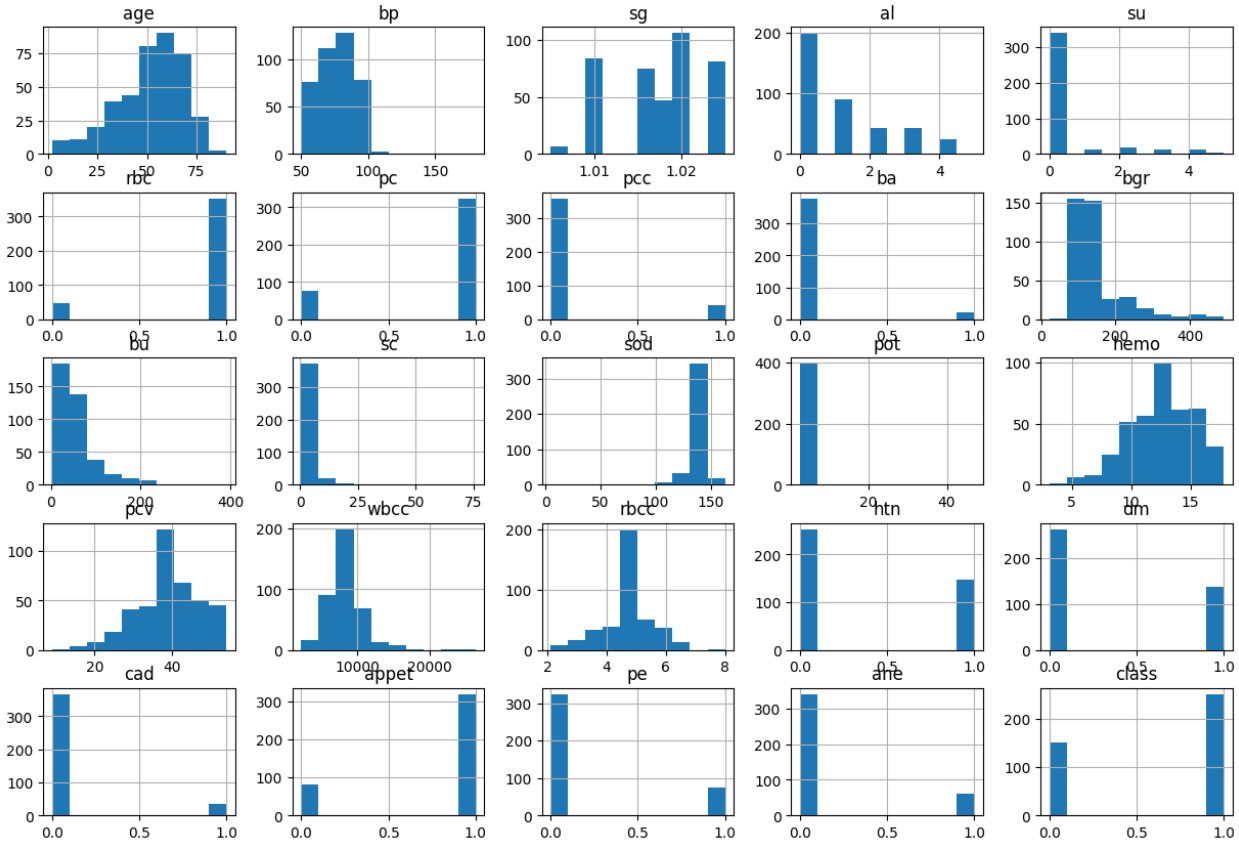
Exploratory Data Analysis

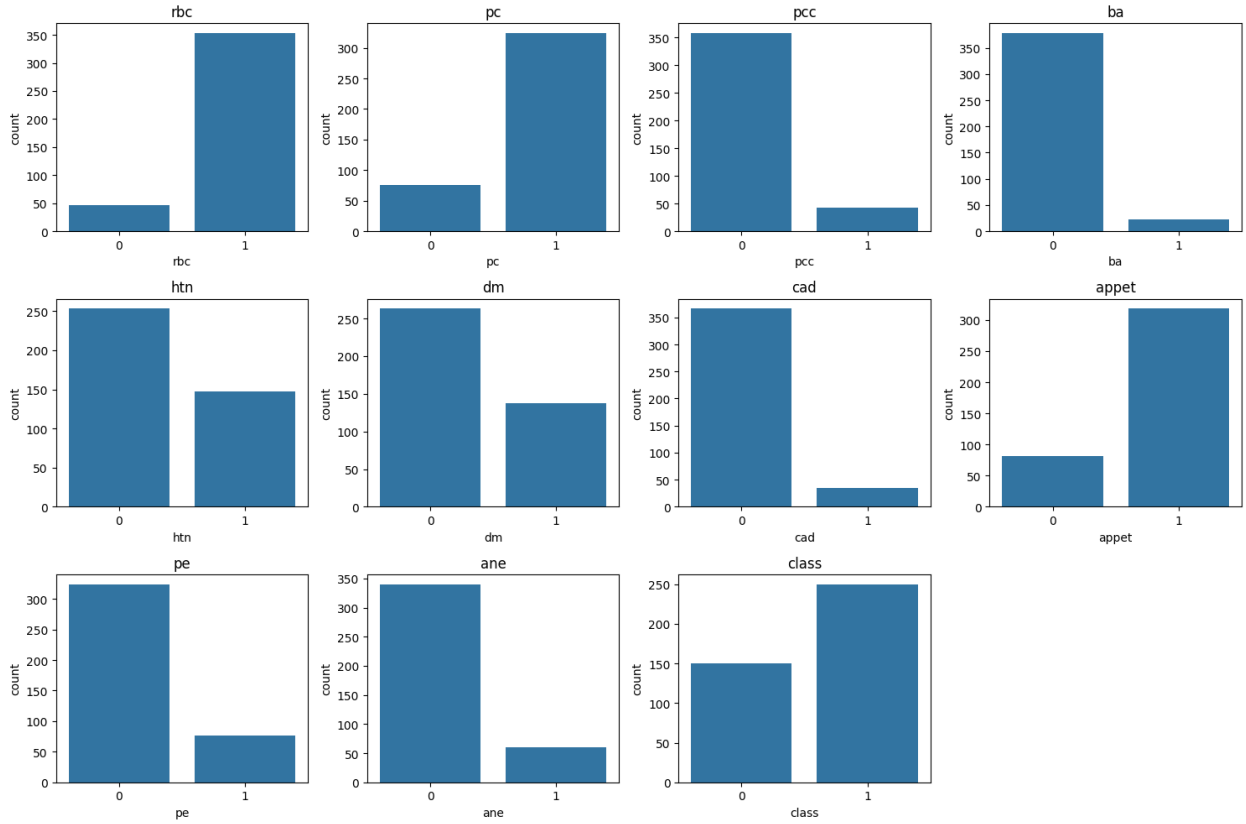
```
dataset.dtypes
dataset.shape
```

```
(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,
```

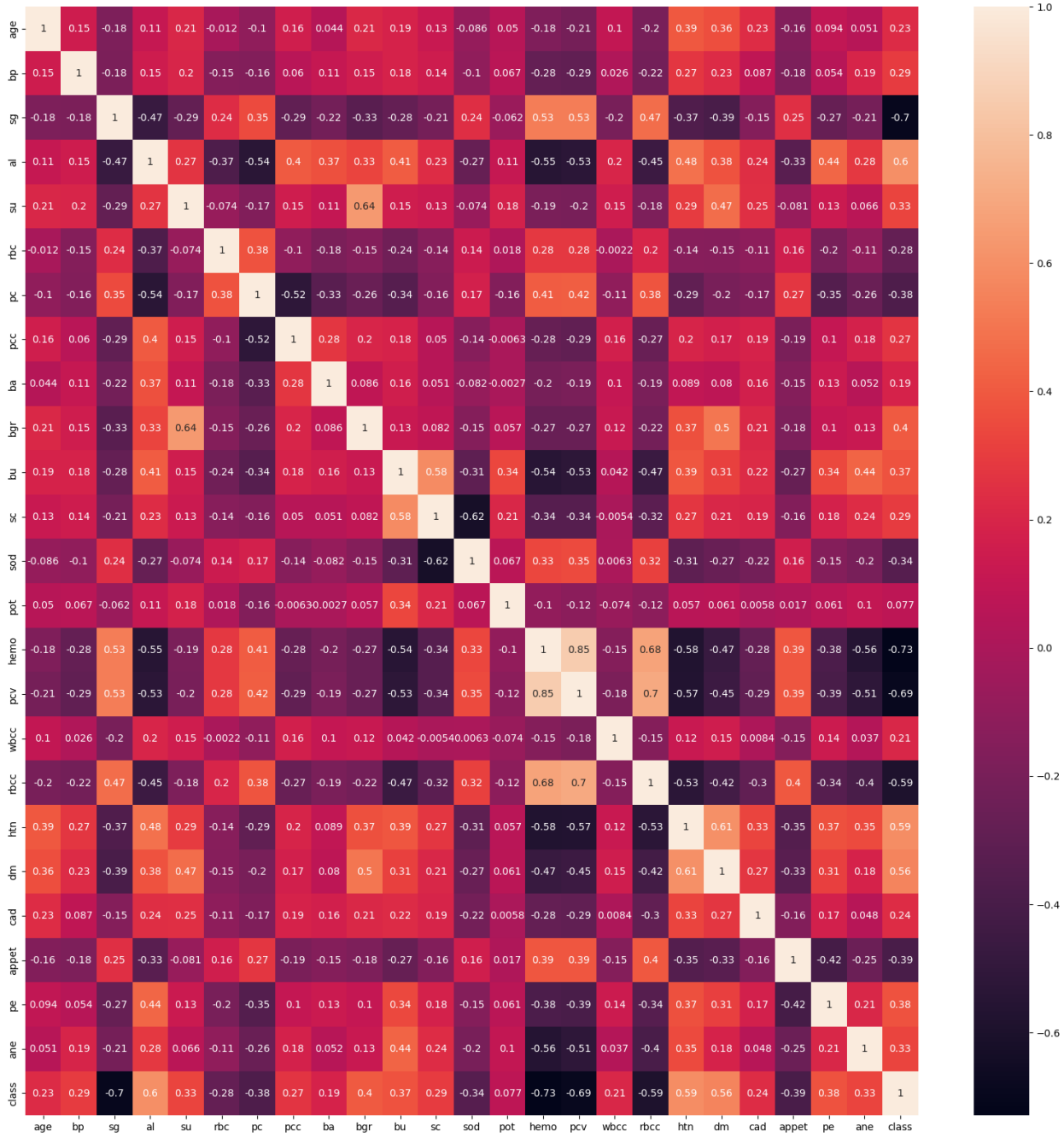
```
    'ckd': 1, 'notckd': 0
}
categorical = [col for col, dtype in dataset.dtypes.items() if dtype
== 'object']
for column in categorical:
    dataset[column] = dataset[column].replace(replacement_mapping)

# 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
age
bp
sg
al
su
rbc
pc
pcc
ba
bgr
bu
sc
sod
pot
hemo
pcv
wbcc
rbcc
htn
dm
cad
appet
pe
ane
class




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

```
<Axes: >
```




```
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

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
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   support

           0       0.97      1.00      0.99        38
           1       1.00      0.98      0.99        62

    accuracy                           0.99       100
   macro avg       0.99      0.99      0.99       100
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       1.00      1.00      1.00        38
           1       1.00      1.00      1.00        62
```

```
    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
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 62]]
Confusion Matrix (Random Forest): [[38  0]
 [ 0 62]]
Confusion Matrix (Logistic Regression): [[36  2]
 [ 4 58]]
AUC (KNN): 0.9919354838709677
AUC (Decision Tree): 1.0
AUC (Random Forest): 0.9999999999999999
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(
```