kidney_diseas_classification

March 11, 2025

[1]:
```
# !pip install seaborn
# !pip install xgboost
# !pip install matplotlib
# !pip install numpy
# !pip install pandas
# !pip3 install -U scikit-learn
# !pip install keras
# !pip install tensorflow
# !pip install openpyxl
# !pip install xlrd
```

0.0.1 Imports

[2]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.naive_bayes import BernoulliNB

from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')
```

Read the data

```
[3]: df = pd.read_csv("Kidney_disease.csv")
df.head()
```

```
[3]:    id   age    bp     sg   al   su     rbc        pc         pcc          ba  \
0   0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent  notpresent
1   1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent  notpresent
2   2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent
3   3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent
4   4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent

   … pcv    wc   rc  htn   dm  cad appet   pe  ane classification
0  …  44  7800  5.2  yes  yes   no  good   no   no            ckd
1  …  38  6000  NaN   no   no   no  good   no   no            ckd
2  …  31  7500  NaN   no  yes   no  poor   no  yes            ckd
3  …  32  6700  3.9  yes   no   no  poor  yes  yes            ckd
4  …  35  7300  4.6   no   no   no  good   no   no            ckd

[5 rows x 26 columns]
```

check for null values

```
[4]: df.isnull().sum().sort_values(ascending = False)
```

```
[4]: rbc               152
rc                130
wc                105
pot                88
sod                87
pcv                70
pc                 65
hemo               52
su                 49
sg                 47
al                 46
bgr                44
bu                 19
sc                 17
bp                 12
age                 9
pcc                 4
ba                  4
dm                  2
htn                 2
cad                 2
appet               1
ane                 1
pe                  1
```

```
id                  0
classification      0
dtype: int64
```

List of numerical and catagorical variables

[5]:
```
cols_numeric = ['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot',␣
 ↪'hemo', 'pcv', 'wc', 'rc']
cols_cat     = ['rbc', 'pc', 'pcc', 'ba', 'htn', 'dm', 'cad', 'appet', 'pe',␣
 ↪'ane', 'classification']
```

function to convert string columns to ints and floats

[6]:
```
def str_to_int(df, col):
  df[col] = df[col].replace(to_replace='\?', value=np.nan, regex=True)
  df[col] = df[col].replace(to_replace=np.nan, value=0, regex=True)
  df[col] = df[col].astype(int)
  df[col] = df[col].replace(to_replace=0, value=np.nan, regex=True)
  return df

def str_to_float(df, col):
  df[col] = df[col].replace(to_replace='\?', value=np.nan, regex=True)
  df[col] = df[col].replace(to_replace=np.nan, value=0.0, regex=True)
  df[col] = df[col].astype(float)
  df[col] = df[col].replace(to_replace=0.0, value=np.nan, regex=True)
  return df
```

[7]:
```
df = str_to_int(df, "pcv")
df = str_to_int(df, "wc")
df = str_to_float(df, "rc")
```

[8]:
```
df.head()
```

[8]:
```
   id   age    bp     sg   al   su     rbc        pc         pcc          ba  \
0   0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent  notpresent
1   1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent  notpresent
2   2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent
3   3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent
4   4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent

   …   pcv      wc   rc  htn   dm  cad  appet   pe  ane classification
0  …  44.0  7800.0  5.2  yes  yes   no   good   no   no            ckd
1  …  38.0  6000.0  NaN   no   no   no   good   no   no            ckd
2  …  31.0  7500.0  NaN   no  yes   no   poor   no  yes            ckd
3  …  32.0  6700.0  3.9  yes   no   no   poor  yes  yes            ckd
4  …  35.0  7300.0  4.6   no   no   no   good   no   no            ckd
```

```
[5 rows x 26 columns]
```

Fill missing values with mean of each numeric column and median for each catagorical column

[9]:
```python
for col_name in cols_numeric:
  df[col_name] = df[col_name].fillna((df[col_name].mean()))

df = df.fillna(df.mode().iloc[0])
```

Encoder catagorical varable into intergers

[10]:
```python
encoders = {}

for col_name in cols_cat:
  le = LabelEncoder()
  encoders[col_name] = le
  df[col_name] = le.fit_transform(df[col_name].values)
```

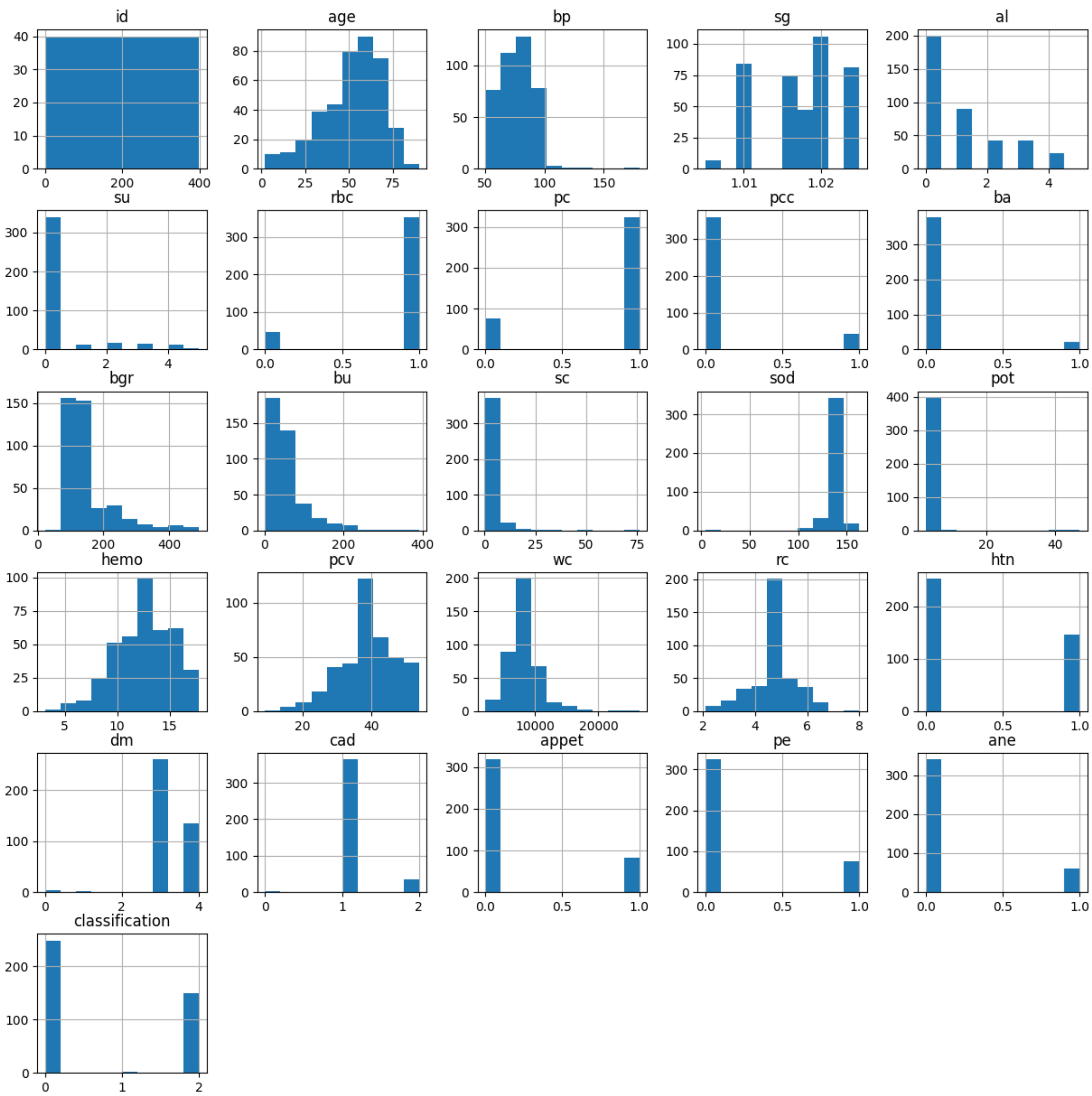
Varify missing values imputation

[11]:
```python
df.isnull().sum().sort_values(ascending = False)
```

[11]:
```
id               0
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
```

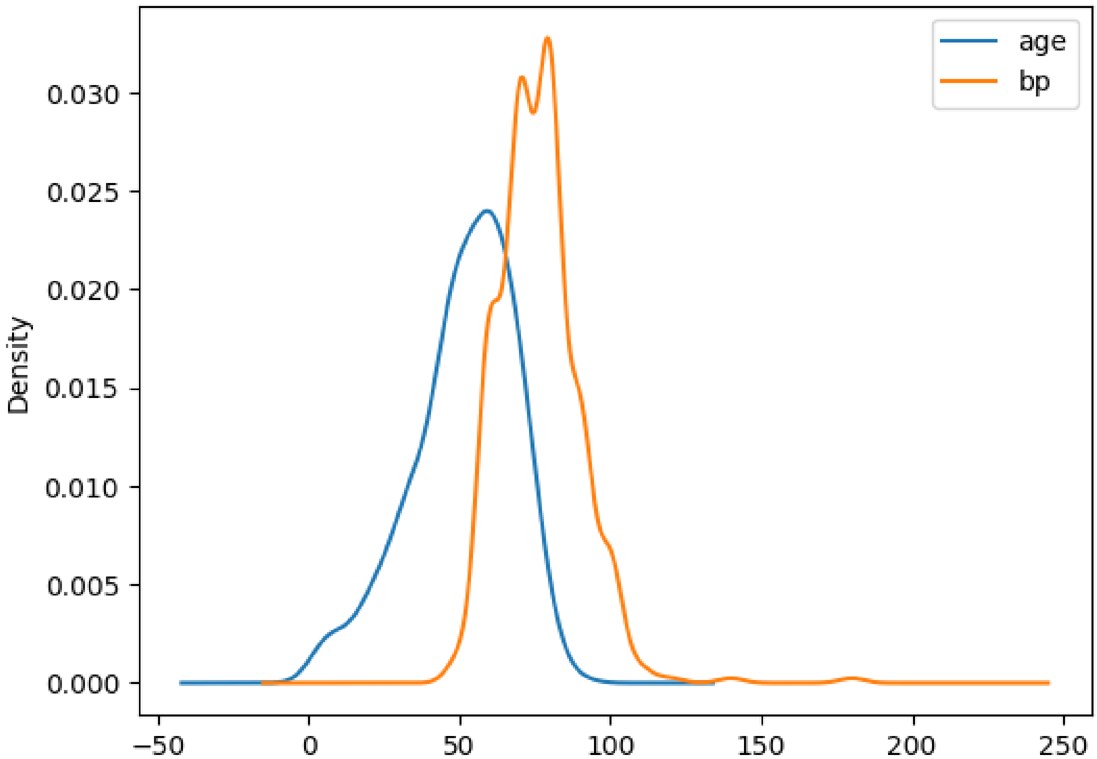
[12]:
```python
df.hist(figsize = (15, 15))
plt.show()
```



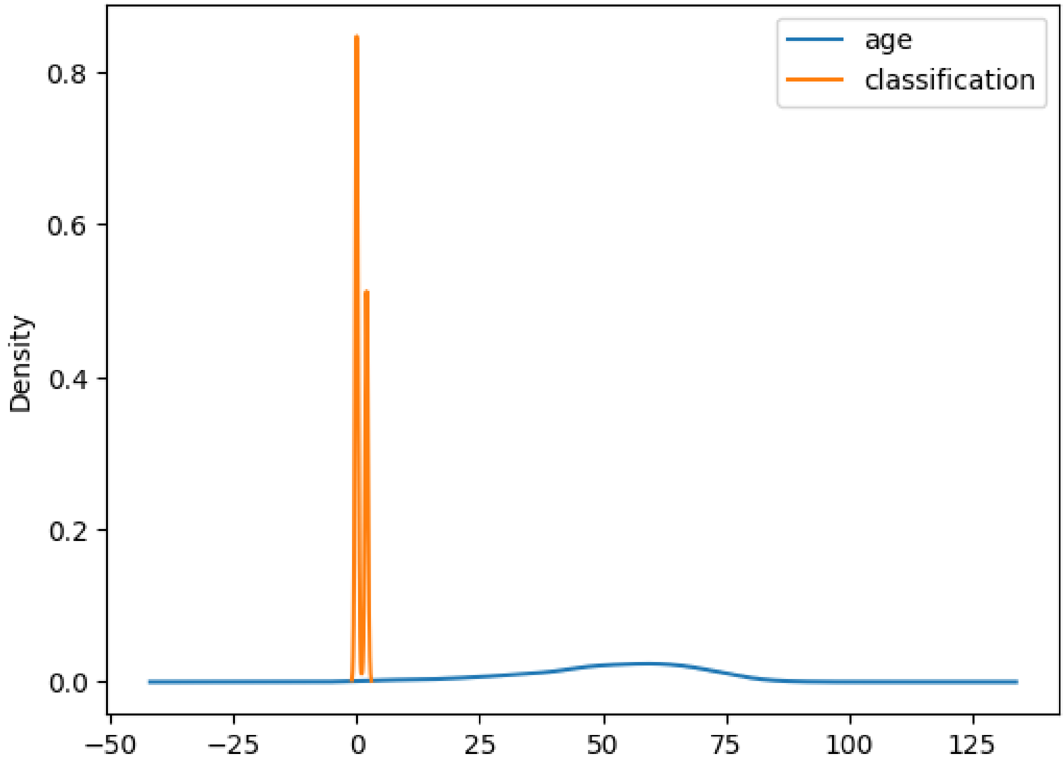
[13]:
```python
df.columns
```

[13]:
```
Index(['id', 'age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr',
       'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad',
       'appet', 'pe', 'ane', 'classification'],
      dtype='object')
```

[14]:
```
ax = df[["age", "bp"]].plot.kde()
```



[15]:
```
ax = df[["age", "classification"]].plot.kde()
```

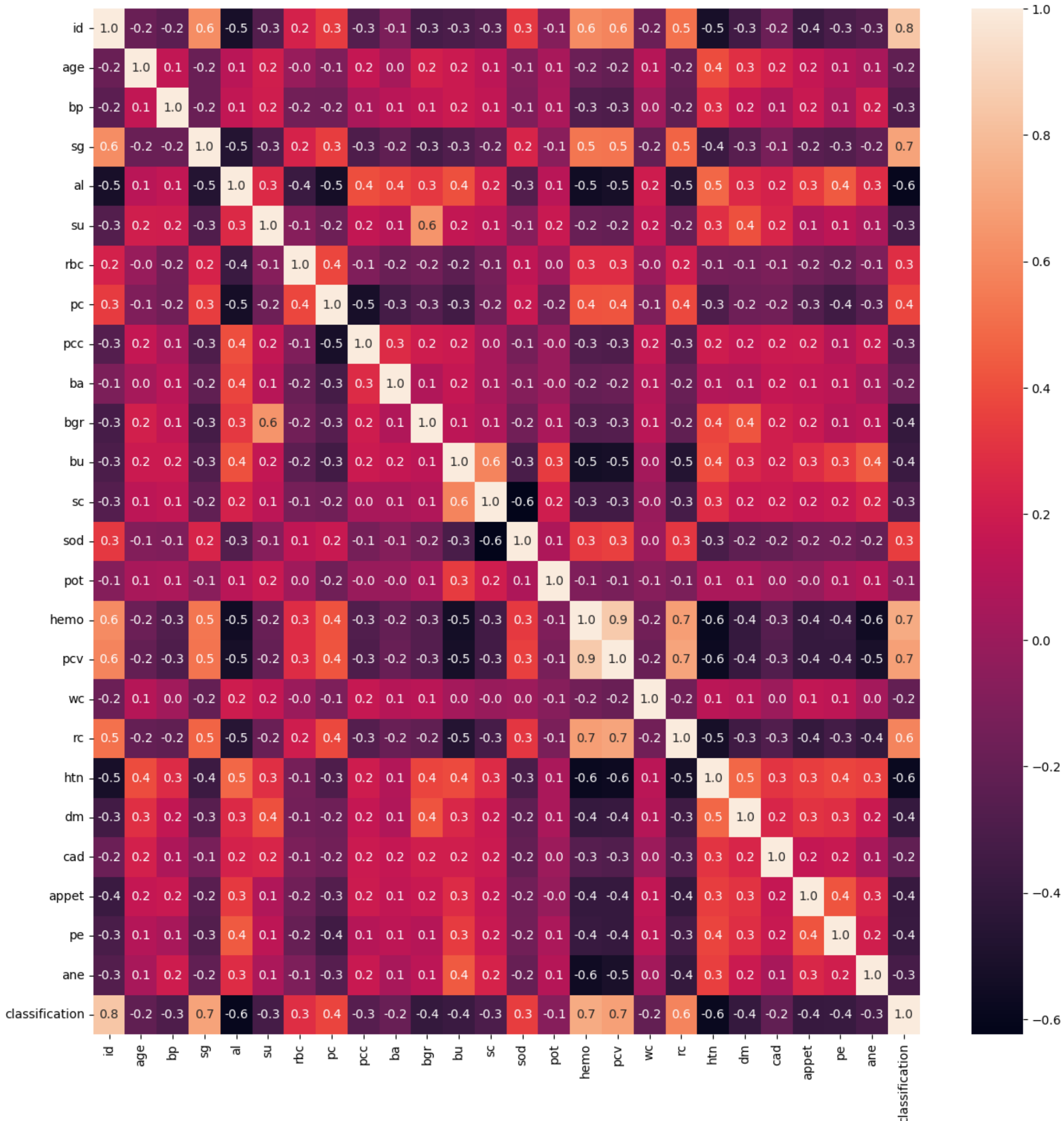


[ ]:

```
```

[16]:

```
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(), annot=True, fmt='.1f')
plt.show()
```



Separate the target varaible y and input variable X

[17]:
```
y = df["classification"].values
df = df.drop(["id", "classification"], axis=1, inplace=False)
X = df.values
```

Divide the dataset into training and test sets

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=42)
```

[19]:
```
table_acc = {}
table_p  = {}
table_r  = {}
table_f1  = {}
```

Apply random forest classifier

[20]:
```
clf = RandomForestClassifier(n_estimators=100, max_depth=100, random_state=0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
p = precision_score(y_test, y_pred, average='macro')
r = recall_score(y_test, y_pred, average='macro')
f1_sc = f1_score(y_test, y_pred, average='macro')

accuracy = np.round(accuracy, 3)*100
p = np.round(p, 3)*100
r = np.round(r, 3)*100
f1_sc = np.round(f1_sc, 3)*100

print("accuracy:", accuracy)
print("precision:", p)
print("recall:", r)
print("f1_score:", f1_sc)

table_acc["RF"] = accuracy
table_p["RF"] = p
table_r["RF"] = r
table_f1["RF"]  = f1_sc
```

```
accuracy: 100.0
precision: 100.0
recall: 100.0
f1_score: 100.0
```

RF cross validation¶

[21]:
```
clf = RandomForestClassifier(n_estimators=100, max_depth=100, random_state=0)
scores = cross_val_score(clf, X, y, cv=10)
print("mean accuracy for 10 fold cross validation",scores.mean() )
```

```
mean accuracy for 10 fold cross validation 0.985
```

Logistic Regression

[22]:
```
log_reg = LogisticRegression()
log_reg.fit(X_train,y_train)
y_pred = log_reg.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
p = precision_score(y_test, y_pred, average='macro')
r = recall_score(y_test, y_pred, average='macro')


f1_sc = f1_score(y_test, y_pred, average='macro')

accuracy = np.round(accuracy, 3)*100
p = np.round(p, 3)*100
r = np.round(r, 3)*100
f1_sc = np.round(f1_sc, 3)*100

print("accuracy:", accuracy)
print("precision:", p)
print("recall:", r)
print("f1_score:", f1_sc)

table_acc["LR"] = accuracy
table_p["LR"] = p
table_r["LR"] = r
table_f1["LR"]  = f1_sc
```

```
accuracy: 92.5
precision: 91.3
recall: 92.60000000000001
f1_score: 91.9
```

Logistic Regression CV

[23]:
```
log_reg = LogisticRegression()
scores = cross_val_score(log_reg, X, y, cv=10)
print("mean accuracy for 10 fold cross validation",scores.mean() )
```

```
mean accuracy for 10 fold cross validation 0.8949999999999999
```

KNN

[24]:
```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
y_pred = neigh.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
p = precision_score(y_test, y_pred, average='macro')
r = recall_score(y_test, y_pred, average='macro')
f1_sc = f1_score(y_test, y_pred, average='macro')

accuracy = np.round(accuracy, 3)*100
p = np.round(p, 3)*100
r = np.round(r, 3)*100
f1_sc = np.round(f1_sc, 3)*100
```

```
print("accuracy:", accuracy)
print("precision:", p)
print("recall:", r)
print("f1_score:", f1_sc)

table_acc["KNN"] = accuracy
table_p["KNN"] = p
table_r["KNN"] = r
table_f1["KNN"]  = f1_sc
```

```
accuracy: 93.8
precision: 92.4
recall: 95.19999999999999
f1_score: 93.4
```

KNN Cross Validation

[25]:
```
neigh = KNeighborsClassifier(n_neighbors=3)
scores = cross_val_score(neigh, X, y, cv=10)
print("mean accuracy for 10 fold cross validation",scores.mean() )
```

```
mean accuracy for 10 fold cross validation 0.7675
```

Apply XGBOOST model

[26]:
```
xgb_model = xgb.XGBClassifier()
# xgb_model = xgb.XGBRegressor(objective="reg:linear", random_state=42)
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
p = precision_score(y_test, y_pred, average='macro')
r = recall_score(y_test, y_pred, average='macro')
f1_sc = f1_score(y_test, y_pred, average='macro')


accuracy = np.round(accuracy, 3)*100
p = np.round(p, 3)*100
r = np.round(r, 3)*100
f1_sc = np.round(f1_sc, 3)*100

print("accuracy:", accuracy)
print("precision:", p)
print("recall:", r)
print("f1_score:", f1_sc)

table_acc["XGB"] = accuracy
table_p["XGB"] = p
```

```
table_r["XGB"] = r
table_f1["XGB"]  = f1_sc
```

```
accuracy: 98.8
precision: 98.3
recall: 99.0
f1_score: 98.6
```

Apply XGBOOST Cross Validation

[27]:
```
xgb_model = xgb.XGBClassifier()
scores = cross_val_score(xgb_model, X, y, cv=10)
print("mean accuracy for 10 fold cross validation",scores.mean() )
```

```
mean accuracy for 10 fold cross validation 0.9775
```

Apply Bernoulli Naive Bayes

[28]:
```
NB = BernoulliNB()
NB.fit(X_train, y_train)
y_pred = NB.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
p = precision_score(y_test, y_pred, average='macro')
r = recall_score(y_test, y_pred, average='macro')
f1_sc = f1_score(y_test, y_pred, average='macro')

accuracy = np.round(accuracy, 3)*100
p = np.round(p, 3)*100
r = np.round(r, 3)*100
f1_sc = np.round(f1_sc, 3)*100

print("accuracy:", accuracy)
print("precision:", p)
print("recall:", r)
print("f1_score:", f1_sc)

table_acc["NB"] = accuracy
table_p["NB"] = p
table_r["NB"] = r
table_f1["NB"]  = f1_sc
```

```
accuracy: 91.2
precision: 89.9
recall: 91.60000000000001
f1_score: 90.60000000000001
```

[29]:
```
#### Apply Bernoulli Naive Bayes Cross Validation
```

[30]:
```
NB = BernoulliNB()
scores = cross_val_score(NB, X, y, cv=10)
print("mean accuracy for 10 fold cross validation",scores.mean() )
```

```
mean accuracy for 10 fold cross validation 0.9199999999999999
```

Apply Decision Tree Classifier

[31]:
```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
p = precision_score(y_test, y_pred, average='macro')
r = recall_score(y_test, y_pred, average='macro')
f1_sc = f1_score(y_test, y_pred, average='macro')

accuracy = np.round(accuracy, 3)*100
p = np.round(p, 3)*100
r = np.round(r, 3)*100
f1_sc = np.round(f1_sc, 3)*100

print("accuracy:", accuracy)
print("precision:", p)
print("recall:", r)
print("f1_score:", f1_sc)

table_acc["DT"] = accuracy
table_p["DT"] = p
table_r["DT"] = r
table_f1["DT"]  = f1_sc
```

```
accuracy: 98.8
precision: 66.7
recall: 66.0
f1_score: 66.3
```

Apply Decision Tree Classifier Cross Validation

[32]:
```
clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, X, y, cv=10)
print("mean accuracy for 10 fold cross validation",scores.mean() )
```

```
mean accuracy for 10 fold cross validation 0.9574999999999999
```

0.0.2 Apply Neural Networks model

[33]:
```
from keras.layers import Dense

from tensorflow.keras import Model, Input
from keras.utils import to_categorical
from keras.optimizers import Adam
```

```
2025-03-11 11:56:02.677862: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-03-11 11:56:02.678625: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-03-11 11:56:02.680540: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-03-11 11:56:02.686303: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1741676162.697871  613659 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1741676162.701454  613659 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2025-03-11 11:56:02.713011: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

[34]:
```
y_train_1 = to_categorical(y_train, num_classes=3)
y_test_1  = to_categorical(y_test, num_classes=3)
```

[35]:
```
y_train_1.shape, X_train.shape
```

[35]:
```
((320, 3), (320, 24))
```

[36]:
```
# inp = Input(shape=(24,), name="input_1")
# x = Dense(100, name="Dense_1")(inp)
# x = Dense(10, name="Dense_2")(x)
# out = Dense(3, activation="softmax", name="output_1")(x)

# model  = Model(inputs = inp, outputs= out , name="Neural_network_model")
# opt = Adam(learning_rate=0.001)
```

```python
# model.compile(optimizer=opt, metrics=["acc"], loss="binary_crossentropy")
# print(model.summary())

# history_obj = model.fit(X_train, y_train_1, epochs=100,␣
↪validation_data=(X_test, y_test_1))


from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint

# Define the model
inp = Input(shape=(24,), name="input_1")
x = Dense(100, name="Dense_1")(inp)
x = Dense(10, name="Dense_2")(x)
out = Dense(3, activation="softmax", name="output_1")(x)

model = Model(inputs=inp, outputs=out, name="Neural_network_model")

# Compile the model
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, metrics=["acc"], loss="binary_crossentropy")

# Print the model summary
print(model.summary())

# Define the ModelCheckpoint callback
checkpoint_callback = ModelCheckpoint(
    filepath='best_model.keras',  # Path where the best model will be saved
    monitor='val_acc',          # Metric to monitor (validation accuracy)
    save_best_only=True,        # Save only the best model
    mode='max',                 # Mode to determine the best model (max for␣
↪accuracy)
    verbose=1                   # Print a message when the best model is saved
)
```

```
W0000 00:00:1741676163.682311  613659 gpu_device.cc:2344] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices…
```

Model: "Neural_network_model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 24)	0
Dense_1 (Dense)	(None, 100)	2,500
Dense_2 (Dense)	(None, 10)	1,010
output_1 (Dense)	(None, 3)	33

Total params: 3,543 (13.84 KB)

Trainable params: 3,543 (13.84 KB)

Non-trainable params: 0 (0.00 B)

```
None
```

[37]:

```
# history_obj = model.fit(X_train, y_train_1, epochs=100,␣
↪validation_data=(X_test, y_test_1))

# Train the model with the checkpoint callback
history_obj = model.fit(
    X_train, y_train_1,
    epochs=100,
    validation_data=(X_test, y_test_1),
    callbacks=[checkpoint_callback]  # Add the callback here
)

# Load the best model at the end
model = load_model('best_model.keras')
```

```
Epoch 1/100

Epoch 1: val_acc improved from -inf to 0.65000, saving model to best_model.keras
10/10            1s 17ms/step - acc:
0.4013 - loss: 319.5885 - val_acc: 0.6500 - val_loss: 137.1401
Epoch 2/100

Epoch 2: val_acc did not improve from 0.65000
10/10            0s 4ms/step - acc:
0.5881 - loss: 124.4577 - val_acc: 0.4000 - val_loss: 51.4881
Epoch 3/100

Epoch 3: val_acc did not improve from 0.65000
```

```
10/10             0s 4ms/step - acc:
0.5563 - loss: 39.3805 - val_acc: 0.6500 - val_loss: 11.5358
Epoch 4/100

Epoch 4: val_acc did not improve from 0.65000
10/10             0s 5ms/step - acc:
0.5163 - loss: 15.8989 - val_acc: 0.6500 - val_loss: 4.6956
Epoch 5/100

Epoch 5: val_acc did not improve from 0.65000
10/10             0s 5ms/step - acc:
0.5358 - loss: 11.8067 - val_acc: 0.4000 - val_loss: 15.9978
Epoch 6/100

Epoch 6: val_acc did not improve from 0.65000
10/10             0s 5ms/step - acc:
0.4863 - loss: 13.1489 - val_acc: 0.6500 - val_loss: 6.1265
Epoch 7/100

Epoch 7: val_acc did not improve from 0.65000
10/10             0s 4ms/step - acc:
0.5977 - loss: 13.1460 - val_acc: 0.6500 - val_loss: 11.4352
Epoch 8/100

Epoch 8: val_acc did not improve from 0.65000
10/10             0s 4ms/step - acc:
0.5794 - loss: 9.7217 - val_acc: 0.6250 - val_loss: 4.3380
Epoch 9/100

Epoch 9: val_acc improved from 0.65000 to 0.66250, saving model to
best_model.keras
10/10             0s 6ms/step - acc:
0.6106 - loss: 6.9028 - val_acc: 0.6625 - val_loss: 3.4173
Epoch 10/100

Epoch 10: val_acc improved from 0.66250 to 0.85000, saving model to
best_model.keras
10/10             0s 6ms/step - acc:
0.6371 - loss: 6.0367 - val_acc: 0.8500 - val_loss: 0.5471
Epoch 11/100

Epoch 11: val_acc improved from 0.85000 to 0.86250, saving model to
best_model.keras
10/10             0s 6ms/step - acc:
0.7153 - loss: 4.9934 - val_acc: 0.8625 - val_loss: 0.7634
Epoch 12/100

Epoch 12: val_acc did not improve from 0.86250
```

```
10/10              0s 5ms/step - acc:
0.6784 - loss: 5.4423 - val_acc: 0.6875 - val_loss: 3.0341
Epoch 13/100

Epoch 13: val_acc did not improve from 0.86250
10/10              0s 4ms/step - acc:
0.6377 - loss: 4.2717 - val_acc: 0.7125 - val_loss: 1.1455
Epoch 14/100

Epoch 14: val_acc did not improve from 0.86250
10/10              0s 5ms/step - acc:
0.6188 - loss: 3.5945 - val_acc: 0.5250 - val_loss: 6.6393
Epoch 15/100

Epoch 15: val_acc did not improve from 0.86250
10/10              0s 5ms/step - acc:
0.5334 - loss: 10.5569 - val_acc: 0.6500 - val_loss: 19.6416
Epoch 16/100

Epoch 16: val_acc did not improve from 0.86250
10/10              0s 4ms/step - acc:
0.6557 - loss: 16.1877 - val_acc: 0.6500 - val_loss: 10.6702
Epoch 17/100

Epoch 17: val_acc did not improve from 0.86250
10/10              0s 4ms/step - acc:
0.6203 - loss: 7.1682 - val_acc: 0.6250 - val_loss: 5.0156
Epoch 18/100

Epoch 18: val_acc did not improve from 0.86250
10/10              0s 4ms/step - acc:
0.5984 - loss: 6.7937 - val_acc: 0.7625 - val_loss: 2.5937
Epoch 19/100

Epoch 19: val_acc did not improve from 0.86250
10/10              0s 4ms/step - acc:
0.6429 - loss: 6.6665 - val_acc: 0.8500 - val_loss: 4.7072
Epoch 20/100

Epoch 20: val_acc improved from 0.86250 to 0.90000, saving model to
best_model.keras
10/10              0s 6ms/step - acc:
0.7521 - loss: 4.8118 - val_acc: 0.9000 - val_loss: 1.4927
Epoch 21/100

Epoch 21: val_acc did not improve from 0.90000
10/10              0s 4ms/step - acc:
0.7641 - loss: 1.8300 - val_acc: 0.8250 - val_loss: 1.3726
```

```
Epoch 22/100

Epoch 22: val_acc did not improve from 0.90000
10/10             0s 4ms/step - acc:
0.7679 - loss: 2.1860 - val_acc: 0.6875 - val_loss: 4.0646
Epoch 23/100

Epoch 23: val_acc did not improve from 0.90000
10/10             0s 4ms/step - acc:
0.6657 - loss: 4.3366 - val_acc: 0.7125 - val_loss: 4.4844
Epoch 24/100

Epoch 24: val_acc did not improve from 0.90000
10/10             0s 4ms/step - acc:
0.6866 - loss: 4.3282 - val_acc: 0.8500 - val_loss: 1.2412
Epoch 25/100

Epoch 25: val_acc did not improve from 0.90000
10/10             0s 4ms/step - acc:
0.7871 - loss: 2.3037 - val_acc: 0.8750 - val_loss: 1.1167
Epoch 26/100

Epoch 26: val_acc did not improve from 0.90000
10/10             0s 5ms/step - acc:
0.8273 - loss: 1.2539 - val_acc: 0.7125 - val_loss: 2.7600
Epoch 27/100

Epoch 27: val_acc improved from 0.90000 to 0.91250, saving model to
best_model.keras
10/10             0s 6ms/step - acc:
0.7628 - loss: 2.4514 - val_acc: 0.9125 - val_loss: 0.7743
Epoch 28/100

Epoch 28: val_acc did not improve from 0.91250
10/10             0s 4ms/step - acc:
0.8269 - loss: 2.2562 - val_acc: 0.6750 - val_loss: 3.5631
Epoch 29/100

Epoch 29: val_acc did not improve from 0.91250
10/10             0s 4ms/step - acc:
0.7367 - loss: 2.6686 - val_acc: 0.8750 - val_loss: 0.4648
Epoch 30/100

Epoch 30: val_acc did not improve from 0.91250
10/10             0s 4ms/step - acc:
0.8088 - loss: 1.4389 - val_acc: 0.8875 - val_loss: 0.7742
Epoch 31/100
```

```
Epoch 31: val_acc did not improve from 0.91250
10/10             0s 4ms/step - acc:
0.8304 - loss: 1.1794 - val_acc: 0.8625 - val_loss: 1.7577
Epoch 32/100

Epoch 32: val_acc did not improve from 0.91250
10/10             0s 5ms/step - acc:
0.7707 - loss: 2.0127 - val_acc: 0.7375 - val_loss: 2.4472
Epoch 33/100

Epoch 33: val_acc did not improve from 0.91250
10/10             0s 4ms/step - acc:
0.7804 - loss: 2.1923 - val_acc: 0.7375 - val_loss: 2.2234
Epoch 34/100

Epoch 34: val_acc did not improve from 0.91250
10/10             0s 5ms/step - acc:
0.7580 - loss: 2.0196 - val_acc: 0.8000 - val_loss: 1.4706
Epoch 35/100

Epoch 35: val_acc did not improve from 0.91250
10/10             0s 4ms/step - acc:
0.7404 - loss: 2.4910 - val_acc: 0.6625 - val_loss: 6.1145
Epoch 36/100

Epoch 36: val_acc did not improve from 0.91250
10/10             0s 5ms/step - acc:
0.7481 - loss: 4.1579 - val_acc: 0.9125 - val_loss: 2.5278
Epoch 37/100

Epoch 37: val_acc did not improve from 0.91250
10/10             0s 5ms/step - acc:
0.7646 - loss: 3.0304 - val_acc: 0.9125 - val_loss: 0.6425
Epoch 38/100

Epoch 38: val_acc improved from 0.91250 to 0.92500, saving model to
best_model.keras
10/10             0s 6ms/step - acc:
0.8224 - loss: 1.7679 - val_acc: 0.9250 - val_loss: 0.5745
Epoch 39/100

Epoch 39: val_acc did not improve from 0.92500
10/10             0s 4ms/step - acc:
0.8440 - loss: 1.1621 - val_acc: 0.8625 - val_loss: 0.6427
Epoch 40/100

Epoch 40: val_acc did not improve from 0.92500
10/10             0s 4ms/step - acc:
```

```
0.8592 - loss: 1.2051 - val_acc: 0.7625 - val_loss: 1.5969
Epoch 41/100

Epoch 41: val_acc did not improve from 0.92500
10/10            0s 4ms/step - acc:
0.7380 - loss: 1.8824 - val_acc: 0.7500 - val_loss: 1.7716
Epoch 42/100

Epoch 42: val_acc did not improve from 0.92500
10/10            0s 4ms/step - acc:
0.8206 - loss: 1.7863 - val_acc: 0.8750 - val_loss: 1.7830
Epoch 43/100

Epoch 43: val_acc did not improve from 0.92500
10/10            0s 5ms/step - acc:
0.8509 - loss: 1.8065 - val_acc: 0.8750 - val_loss: 2.4275
Epoch 44/100

Epoch 44: val_acc did not improve from 0.92500
10/10            0s 4ms/step - acc:
0.8014 - loss: 2.2883 - val_acc: 0.9125 - val_loss: 0.8756
Epoch 45/100

Epoch 45: val_acc did not improve from 0.92500
10/10            0s 5ms/step - acc:
0.7108 - loss: 3.8279 - val_acc: 0.7000 - val_loss: 3.1774
Epoch 46/100

Epoch 46: val_acc did not improve from 0.92500
10/10            0s 5ms/step - acc:
0.7199 - loss: 4.0440 - val_acc: 0.6875 - val_loss: 6.1329
Epoch 47/100

Epoch 47: val_acc did not improve from 0.92500
10/10            0s 4ms/step - acc:
0.6829 - loss: 5.6971 - val_acc: 0.8250 - val_loss: 1.5947
Epoch 48/100

Epoch 48: val_acc did not improve from 0.92500
10/10            0s 4ms/step - acc:
0.7473 - loss: 2.6920 - val_acc: 0.9000 - val_loss: 0.9820
Epoch 49/100

Epoch 49: val_acc did not improve from 0.92500
10/10            0s 5ms/step - acc:
0.7506 - loss: 3.2577 - val_acc: 0.7000 - val_loss: 3.1978
Epoch 50/100
```

```
Epoch 50: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8012 - loss: 1.9095 - val_acc: 0.9250 - val_loss: 0.6111
Epoch 51/100

Epoch 51: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8060 - loss: 1.6804 - val_acc: 0.9125 - val_loss: 1.2910
Epoch 52/100

Epoch 52: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8137 - loss: 2.4244 - val_acc: 0.9000 - val_loss: 0.7693
Epoch 53/100

Epoch 53: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.7733 - loss: 1.7535 - val_acc: 0.9250 - val_loss: 0.7271
Epoch 54/100

Epoch 54: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.7994 - loss: 1.9776 - val_acc: 0.8625 - val_loss: 1.1383
Epoch 55/100

Epoch 55: val_acc did not improve from 0.92500
10/10              0s 5ms/step - acc:
0.6269 - loss: 6.4842 - val_acc: 0.6625 - val_loss: 6.0059
Epoch 56/100

Epoch 56: val_acc did not improve from 0.92500
10/10              0s 5ms/step - acc:
0.6438 - loss: 6.1513 - val_acc: 0.8750 - val_loss: 1.6767
Epoch 57/100

Epoch 57: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8690 - loss: 1.8431 - val_acc: 0.9250 - val_loss: 0.7428
Epoch 58/100

Epoch 58: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8559 - loss: 1.0232 - val_acc: 0.9250 - val_loss: 0.7610
Epoch 59/100

Epoch 59: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8290 - loss: 1.2473 - val_acc: 0.7000 - val_loss: 3.6917
```

```
Epoch 60/100

Epoch 60: val_acc did not improve from 0.92500
10/10              0s 5ms/step - acc:
0.7012 - loss: 3.3312 - val_acc: 0.8500 - val_loss: 1.5702
Epoch 61/100

Epoch 61: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8123 - loss: 1.6071 - val_acc: 0.8375 - val_loss: 1.7127
Epoch 62/100

Epoch 62: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8230 - loss: 1.4339 - val_acc: 0.7875 - val_loss: 1.9652
Epoch 63/100

Epoch 63: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8737 - loss: 1.1136 - val_acc: 0.9125 - val_loss: 0.3362
Epoch 64/100

Epoch 64: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8817 - loss: 0.9245 - val_acc: 0.8625 - val_loss: 2.1488
Epoch 65/100

Epoch 65: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8297 - loss: 2.1640 - val_acc: 0.9000 - val_loss: 4.6981
Epoch 66/100

Epoch 66: val_acc did not improve from 0.92500
10/10              0s 5ms/step - acc:
0.8301 - loss: 3.4828 - val_acc: 0.9000 - val_loss: 1.0808
Epoch 67/100

Epoch 67: val_acc did not improve from 0.92500
10/10              0s 4ms/step - acc:
0.8095 - loss: 2.0298 - val_acc: 0.8875 - val_loss: 1.0005
Epoch 68/100

Epoch 68: val_acc improved from 0.92500 to 0.93750, saving model to
best_model.keras
10/10              0s 6ms/step - acc:
0.8284 - loss: 1.7448 - val_acc: 0.9375 - val_loss: 0.3206
Epoch 69/100
```

```
Epoch 69: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.8321 - loss: 1.1985 - val_acc: 0.7375 - val_loss: 3.6683
Epoch 70/100

Epoch 70: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.7211 - loss: 3.3667 - val_acc: 0.7500 - val_loss: 2.2215
Epoch 71/100

Epoch 71: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.7884 - loss: 2.2642 - val_acc: 0.9125 - val_loss: 0.4447
Epoch 72/100

Epoch 72: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.8943 - loss: 0.9631 - val_acc: 0.8875 - val_loss: 0.3540
Epoch 73/100

Epoch 73: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.8634 - loss: 0.7987 - val_acc: 0.8750 - val_loss: 0.5857
Epoch 74/100

Epoch 74: val_acc did not improve from 0.93750
10/10            0s 6ms/step - acc:
0.8689 - loss: 0.6929 - val_acc: 0.9375 - val_loss: 0.3705
Epoch 75/100

Epoch 75: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.9075 - loss: 1.0903 - val_acc: 0.9375 - val_loss: 0.3053
Epoch 76/100

Epoch 76: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.8945 - loss: 1.0610 - val_acc: 0.8750 - val_loss: 1.8160
Epoch 77/100

Epoch 77: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.9009 - loss: 1.5531 - val_acc: 0.9375 - val_loss: 0.6747
Epoch 78/100

Epoch 78: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.8808 - loss: 0.7454 - val_acc: 0.9250 - val_loss: 0.4743
```

```
Epoch 79/100

Epoch 79: val_acc did not improve from 0.93750
10/10             0s 4ms/step - acc:
0.8420 - loss: 0.8619 - val_acc: 0.8250 - val_loss: 1.4800
Epoch 80/100

Epoch 80: val_acc did not improve from 0.93750
10/10             0s 4ms/step - acc:
0.8327 - loss: 1.3967 - val_acc: 0.7500 - val_loss: 3.2671
Epoch 81/100

Epoch 81: val_acc did not improve from 0.93750
10/10             0s 4ms/step - acc:
0.7423 - loss: 2.3901 - val_acc: 0.8875 - val_loss: 0.3701
Epoch 82/100

Epoch 82: val_acc did not improve from 0.93750
10/10             0s 4ms/step - acc:
0.7460 - loss: 2.3952 - val_acc: 0.7750 - val_loss: 1.6107
Epoch 83/100

Epoch 83: val_acc did not improve from 0.93750
10/10             0s 5ms/step - acc:
0.8170 - loss: 1.8934 - val_acc: 0.7000 - val_loss: 3.2279
Epoch 84/100

Epoch 84: val_acc did not improve from 0.93750
10/10             0s 4ms/step - acc:
0.7976 - loss: 2.5554 - val_acc: 0.9000 - val_loss: 0.7931
Epoch 85/100

Epoch 85: val_acc did not improve from 0.93750
10/10             0s 5ms/step - acc:
0.8393 - loss: 1.8093 - val_acc: 0.9375 - val_loss: 0.4245
Epoch 86/100

Epoch 86: val_acc did not improve from 0.93750
10/10             0s 5ms/step - acc:
0.9163 - loss: 2.4669 - val_acc: 0.8750 - val_loss: 1.2572
Epoch 87/100

Epoch 87: val_acc did not improve from 0.93750
10/10             0s 4ms/step - acc:
0.8966 - loss: 1.5714 - val_acc: 0.9250 - val_loss: 0.6908
Epoch 88/100

Epoch 88: val_acc did not improve from 0.93750
```

```
10/10            0s 4ms/step - acc:
0.8733 - loss: 2.6624 - val_acc: 0.9250 - val_loss: 2.2571
Epoch 89/100

Epoch 89: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.8587 - loss: 2.3818 - val_acc: 0.9375 - val_loss: 1.0516
Epoch 90/100

Epoch 90: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.8668 - loss: 1.1922 - val_acc: 0.7875 - val_loss: 1.4917
Epoch 91/100

Epoch 91: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.8856 - loss: 0.9434 - val_acc: 0.9375 - val_loss: 0.5403
Epoch 92/100

Epoch 92: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.9036 - loss: 1.0650 - val_acc: 0.9125 - val_loss: 0.9049
Epoch 93/100

Epoch 93: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.8859 - loss: 1.2628 - val_acc: 0.8750 - val_loss: 1.5060
Epoch 94/100

Epoch 94: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.8693 - loss: 1.3977 - val_acc: 0.9125 - val_loss: 0.5206
Epoch 95/100

Epoch 95: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.8732 - loss: 0.9244 - val_acc: 0.6000 - val_loss: 2.8076
Epoch 96/100

Epoch 96: val_acc did not improve from 0.93750
10/10            0s 5ms/step - acc:
0.7782 - loss: 2.4530 - val_acc: 0.8750 - val_loss: 2.6428
Epoch 97/100

Epoch 97: val_acc did not improve from 0.93750
10/10            0s 4ms/step - acc:
0.8420 - loss: 2.8359 - val_acc: 0.9375 - val_loss: 0.6643
Epoch 98/100
```

```
Epoch 98: val_acc improved from 0.93750 to 0.96250, saving model to
best_model.keras
10/10             0s 6ms/step - acc:
0.8701 - loss: 1.9077 - val_acc: 0.9625 - val_loss: 0.2261
Epoch 99/100

Epoch 99: val_acc did not improve from 0.96250
10/10             0s 5ms/step - acc:
0.8318 - loss: 3.1353 - val_acc: 0.6750 - val_loss: 6.0692
Epoch 100/100

Epoch 100: val_acc did not improve from 0.96250
10/10             0s 5ms/step - acc:
0.6771 - loss: 6.3368 - val_acc: 0.8000 - val_loss: 2.4258
```

[38]:

```
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

accuracy = accuracy_score(y_test, y_pred)
p = precision_score(y_test, y_pred, average='macro')
r = recall_score(y_test, y_pred, average='macro')
f1_sc = f1_score(y_test, y_pred, average='macro')

accuracy = np.round(accuracy, 3)*100
p = np.round(p, 3)*100
r = np.round(r, 3)*100
f1_sc = np.round(f1_sc, 3)*100

print("accuracy:", accuracy)
print("precision:", p)
print("recall:", r)
print("f1_score:", f1_sc)

table_acc["NN"] = accuracy
table_p["NN"] = p
table_r["NN"] = r
table_f1["NN"]  = f1_sc
```

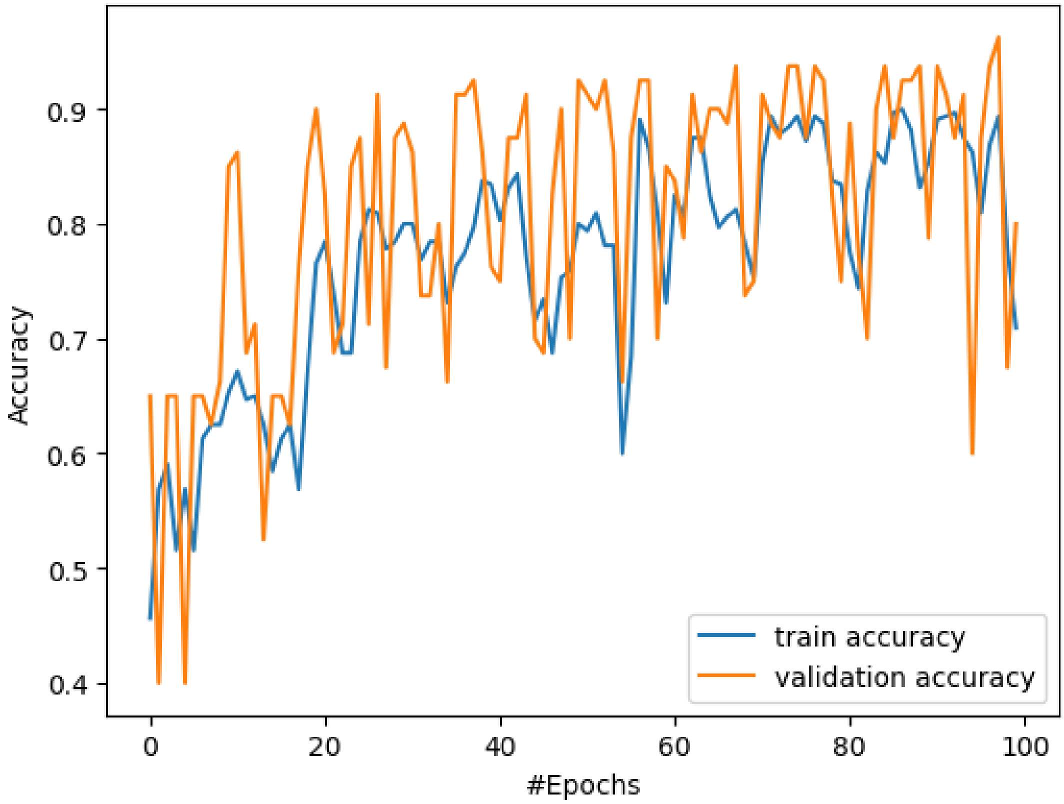
```
3/3             0s 12ms/step
accuracy: 96.2
precision: 95.6
recall: 96.3
f1_score: 95.89999999999999
```

[ ]:

```

```

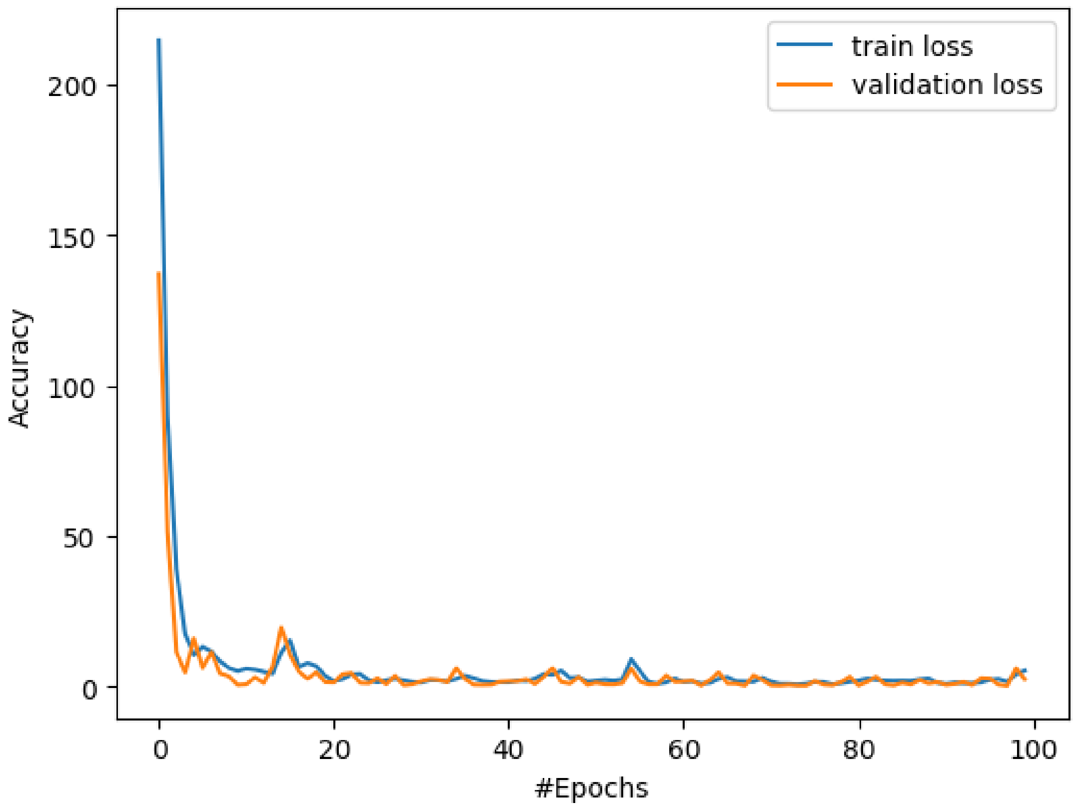
[39]:

```
plt.plot(history_obj.history["acc"], label="train accuracy")
plt.plot(history_obj.history["val_acc"], label="validation accuracy")
plt.xlabel("#Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[40]:

```
plt.plot(history_obj.history["loss"], label="train loss")
plt.plot(history_obj.history["val_loss"], label="validation loss")
plt.xlabel("#Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[41]:
```
def build_nn_model():
  inp = Input(shape=(24,), name="input_1")
  x = Dense(100, name="Dense_1")(inp)
  x = Dense(10, name="Dense_2")(x)
  out = Dense(3, activation="softmax", name="output_1")(x)
  model  = Model(inputs = inp, outputs= out , name="Neural_network_model")
  opt = Adam(learning_rate=0.001)
  model.compile(optimizer=opt, metrics=["acc"], loss="binary_crossentropy")
  ## loss='categorical_crossentropy'
  return model
```

[42]:
```
def k_fold_cross_validation(X, y, k, epochs):
  result_acc = []
  result_f1 = []
  fold_size = int(len(X)/k)
  prev = 0
  for i, next in enumerate(range(fold_size, len(X)+1, fold_size)):
    print("fold:", i+1)
    X_test = X[prev : next]
    X_train = np.array(list(X[:prev]) + list(X[next:]))
    y_test = y[prev : next]
```

```
    y_train = np.array(list(y[:prev]) + list(y[next:]) )
    y_train_1 = to_categorical(y_train, num_classes=3)
    y_test_1  = to_categorical(y_test, num_classes=3)
    model = build_nn_model()
    hist = model.fit(X_train, y_train_1, epochs=epochs,␣
↪validation_data=(X_test, y_test_1), verbose=0)
    ##############################
    y_pred = model.predict(X_test)
    y_pred = np.argmax(y_pred, axis=1)
    accuracy = accuracy_score(y_test, y_pred)*100
    f1_sc = f1_score(y_test, y_pred, average='macro')*100
    result_acc.append(accuracy)
    result_f1.append(f1_sc)
    ############################
    prev = next
  return np.array(result_acc), np.array(result_f1)
```

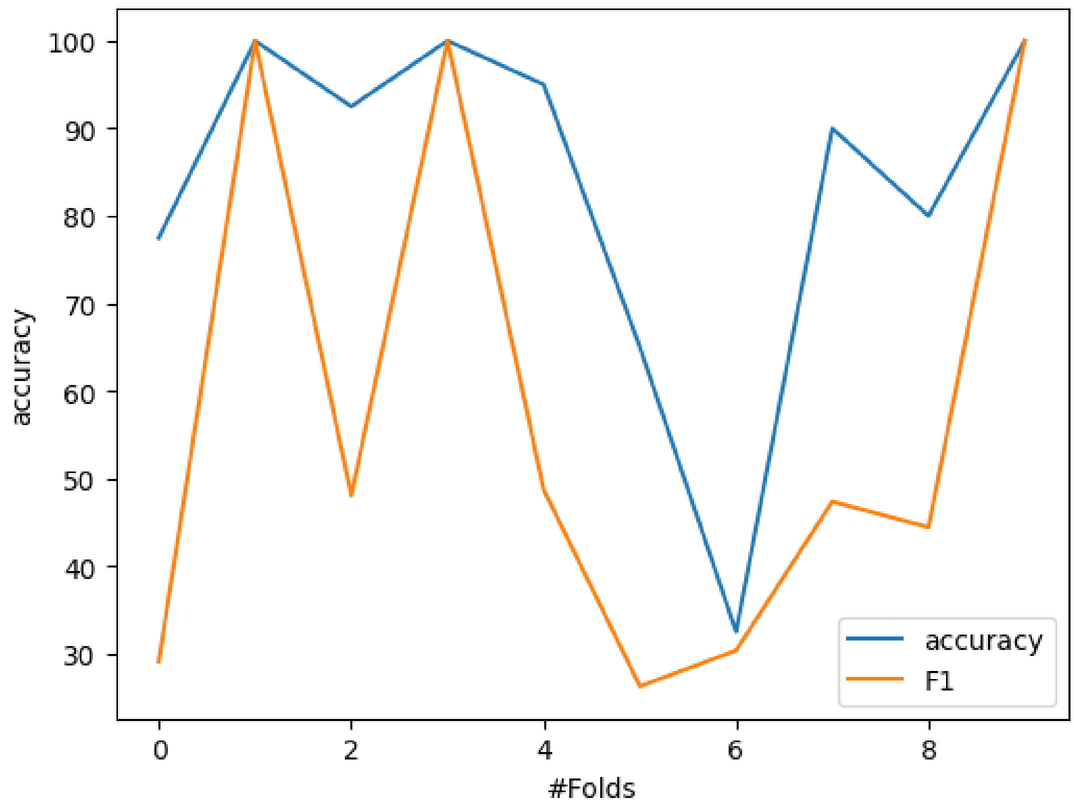
[43]:
```
k = 10 ### number of folds for k fold cross validation
epochs = 200

acc, f1 = k_fold_cross_validation(X, y, k, epochs)
```

```
fold: 1
2/2              0s 24ms/step
fold: 2
WARNING:tensorflow:5 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7c048d32dfc0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 7 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7c048d32dfc0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/2              0s 24ms/step
fold: 3
```

```
2/2              0s 24ms/step
fold: 4
2/2              0s 26ms/step
fold: 5
2/2              0s 24ms/step
fold: 6
2/2              0s 24ms/step
fold: 7
2/2              0s 26ms/step
fold: 8
2/2              0s 25ms/step
fold: 9
2/2              0s 25ms/step
fold: 10
2/2              0s 25ms/step
```

[44]:

```
plt.plot(acc, label="accuracy")
plt.plot(f1, label="F1")
plt.xlabel("#Folds")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```

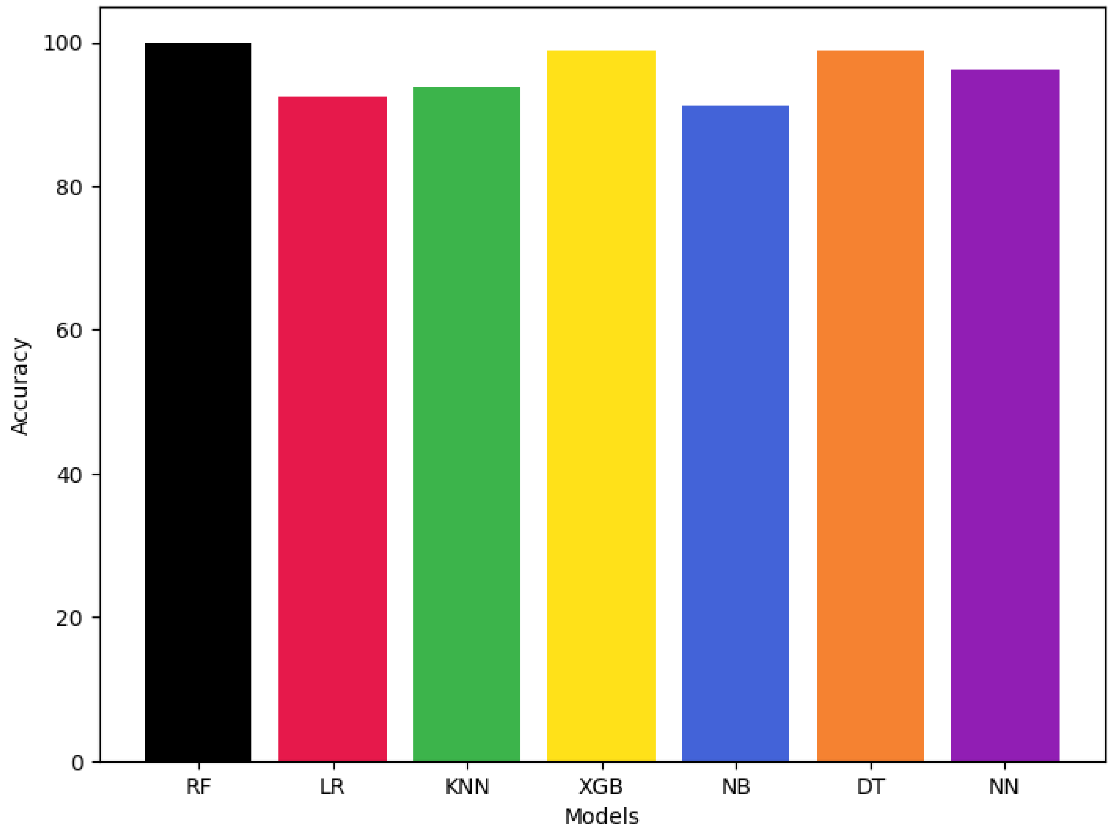


[45]:
```
mean_acc = acc.mean()
print("mean accuacy for", k, "fold cross validaton", mean_acc)

mean_f1 = f1.mean()
print("mean F1 for", k, "fold cross validaton", mean_f1)
```

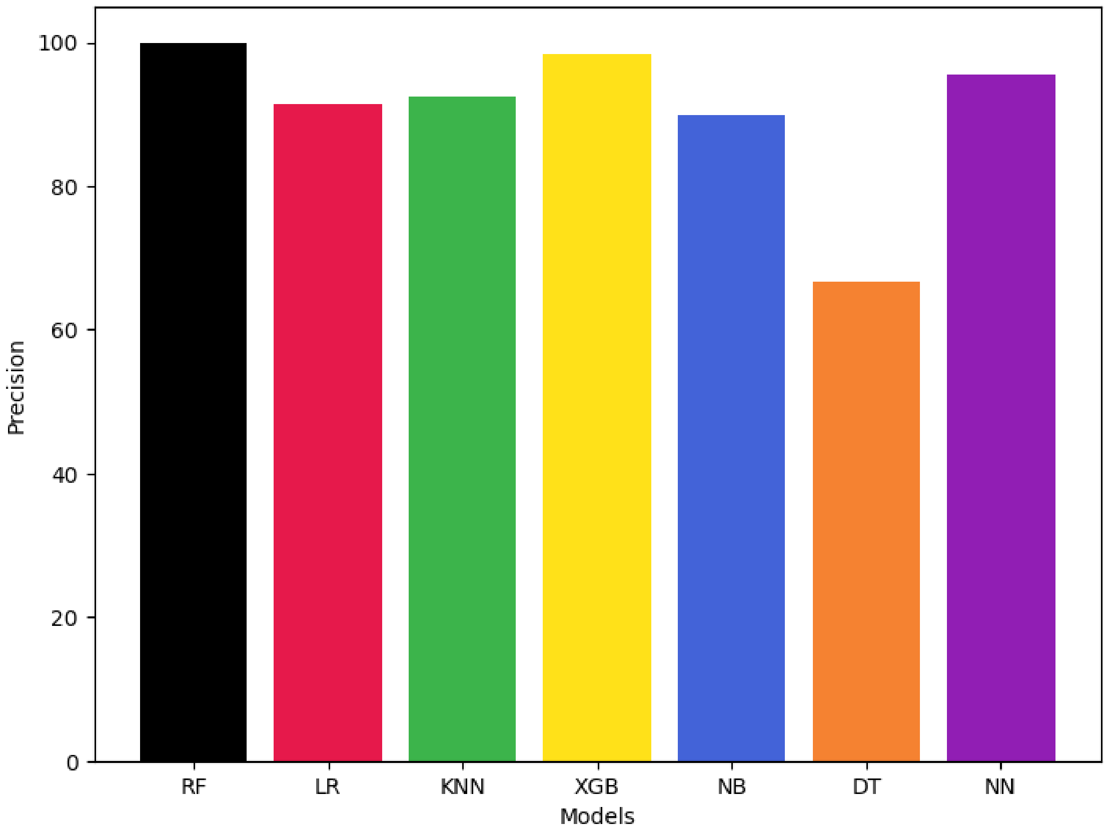
```
mean accuacy for 10 fold cross validaton 83.25
mean F1 for 10 fold cross validaton 57.432087458584625
```

[46]:
```
res = pd.DataFrame({"acc":table_acc, "F1":table_f1})
res = pd.DataFrame({"acc":table_acc, "P":table_p, "R":table_r, "F1":table_f1})
res.to_excel("res.xlsx")
```

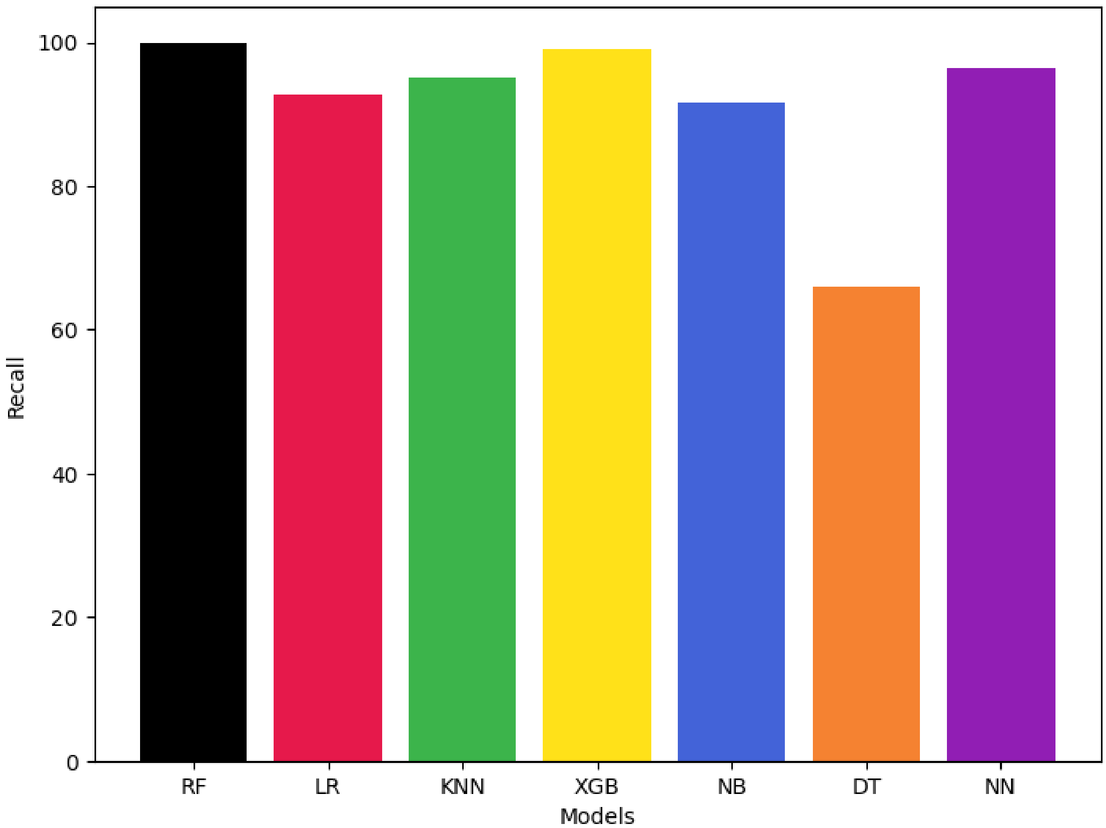
[47]:
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
labels = list(res.index)
acc = res["acc"]
# colors=['black', 'red', 'green', 'blue', 'cyan', 'black', 'blue', '#eeefff',␣
↪'orange', 'green']

colors =  ['#000000', '#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231',␣
↪'#911eb4',
          '#46f0f0', '#f032e6', '#bcf60c', '#fabebe', '#008080', '#e6beff',␣
↪'#9a6324', '#fffac8',
          '#800000', '#aaffc3', '#808000', '#ffd8b1', '#000075', '#808080',␣
↪'#ffffff']
ax.bar(labels, acc, color = colors[:len(acc)])
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.show()
```



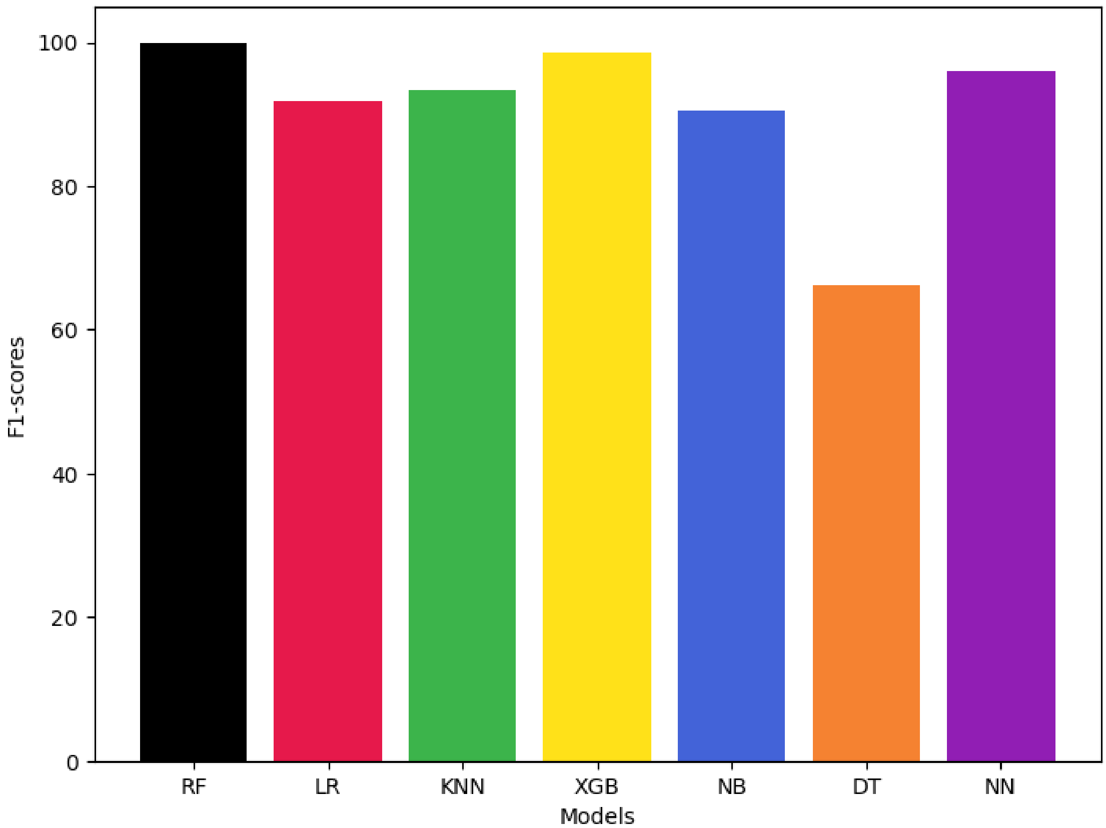
[48]:
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
labels = list(res.index)
acc = res["P"]
# colors=['black', 'red', 'green', 'blue', 'cyan', 'black', 'blue', '#eeefff',
 ↪'orange', 'green']

colors =  ['#000000', '#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231',
 ↪'#911eb4',
          '#46f0f0', '#f032e6', '#bcf60c', '#fabebe', '#008080', '#e6beff',
 ↪'#9a6324', '#fffac8',
          '#800000', '#aaffc3', '#808000', '#ffd8b1', '#000075', '#808080',
 ↪'#ffffff']
ax.bar(labels, acc, color = colors[:len(acc)])
plt.xlabel("Models")
plt.ylabel("Precision")
plt.show()
```



[49]:
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
labels = list(res.index)
acc = res["R"]
# colors=['black', 'red', 'green', 'blue', 'cyan', 'black', 'blue', '#eeefff',␣
↪'orange', 'green']

colors =  ['#000000', '#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231',␣
↪'#911eb4',
          '#46f0f0', '#f032e6', '#bcf60c', '#fabebe', '#008080', '#e6beff',␣
↪'#9a6324', '#fffac8',
          '#800000', '#aaffc3', '#808000', '#ffd8b1', '#000075', '#808080',␣
↪'#ffffff']
ax.bar(labels, acc, color = colors[:len(acc)])
plt.xlabel("Models")
plt.ylabel("Recall")
plt.show()
```



[50]:
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
labels = list(res.index)
acc = res["F1"]
# colors=['black', 'red', 'green', 'blue', 'cyan', 'black', 'blue', '#eeefff',␣
↪'orange', 'green']

colors =  ['#000000', '#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231',␣
↪'#911eb4',
          '#46f0f0', '#f032e6', '#bcf60c', '#fabebe', '#008080', '#e6beff',␣
↪'#9a6324', '#fffac8',
          '#800000', '#aaffc3', '#808000', '#ffd8b1', '#000075', '#808080',␣
↪'#ffffff']
ax.bar(labels, acc, color = colors[:len(acc)])
plt.xlabel("Models")
plt.ylabel("F1-scores")
plt.show()
```



100
80
60
40
20
0
F1-scores
RF
LR
KNN
XGB
NB
DT
NN
Models

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: