```
# -*- coding: utf-8 -*-
"""Breast Cancer: Malignant or Benign
.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1ZcDcJh3CZeYGY2FwtSZew1d-4qnIv9y_

## Golden
Task
##Breast Cancer Maligant or Benign

##importing required
packages
"""

import numpy as np
import pandas as pd

import os
for
dirname,_,filenames in os.walk('/kaggle/input'):
    for filename in filenames:

print(os.path.join(dirname, filename))

import warnings
import os

# Commented out IPython
magic to ensure Python compatibility.
import numpy as np
import pandas as pd
import
matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns
from imblearn.over_sampling
import SMOTE

"""##Machine learning Models"""

from
sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import
LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import
SVC
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from sklearn.metrics import
accuracy_score, confusion_matrix,
classification_report

"""##ANN"""

import tensorflow as tf
from
tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense,
Dropout, BatchNormalization
from keras.utils import plot_model
#import keras_tuner as kt
from
keras.callbacks import EarlyStopping

"""##Importing
Dataset

"""
```

```
bc =
pd.read_csv("/content/breast-cancer.csv")

"""##EDA"""


df =
bc.copy()

df.info()

df.isna().sum()

df.duplicated().sum()

df.describe()

df.head()

df.drop
(columns=['id'],axis = 1,inplace = True)

df['diagnosis'] = df['diagnosis'].map({'M':0,
'B':1})

df.head()

df.tail()

df['diagnosis'].value_counts()

X =
df.drop(columns=['diagnosis'], axis = 1)
Y =
df['diagnosis']

X.shape

X.corr()

Y.shape

Y.value_counts()

X.head()

Y.head()

""
"##Visualisation of Data"""

plt.figure(figsize=(10,
7))
sns.heatmap(X.corr(), linewidth=1.0, cmap =
sns.cubehelix_palette(as_cmap=True))

sns.pairplot(X, corner =
True)

sns.boxplot(Y.value_counts())

df['diagnosis'].value_counts().plot(kind = 'pie', autopct
= '%1.1f%%')

"""##Model Building"""

X_train, X_test, y_train,
y_test = train_test_split(X, Y, test_size=0.2,
random_state=1)

X_train.shape

X_test.shape

y_train.shape
```

```
scaler =
StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled =
scaler.transform(X_test)

X_train_scaled

"""##Logistic
Regression"""

lr = LogisticRegression()

lr.fit(X_train_scaled,
y_train)

pred_lr = lr.predict(X_test_scaled)

pred_lr

y_test

score_lr =
accuracy_score(y_test, pred_lr)

score_lr

print(classification_report(y_test, pred_lr))

cm_lr
= confusion_matrix(y_test, pred_lr)
sns.heatmap(cm_lr,
annot=True)

"""##Decision Tree Classifier"""

dtc =
DecisionTreeClassifier()

par_dtc = {'criterion':['gini', 'entropy', 'loss_loss'],

'max_depth':[2,4,6,8],
           'min_samples_split':[2,4,6,8],
          'max_features':
['auto', 'sqrt', 'log2'],
           'ccp_alpha': [0.1, 0.01, 0.001]}

gcv_dtc =
GridSearchCV(estimator=dtc, param_grid=par_dtc, cv=5, verbose=1)

gcv_dtc.fit(X_train_scaled,
y_train)

gcv_dtc.best_params_

dtc_new = DecisionTreeClassifier(ccp_alpha=0.001,
criterion='gini', max_depth=6, max_features= 'auto', min_samples_split=6
)

dtc_new.fit(X_train_scaled, y_train)

pred_dtc = dtc_new.predict(X_test_scaled)

score_dtc =
accuracy_score(y_test, pred_dtc)

score_dtc

print(classification_report(y_test,
pred_dtc))

cm_dtc = confusion_matrix(y_test, pred_dtc)
sns.heatmap(cm_dtc,
```

```
annot=True)

"""##SVC ( Support Vector Classifier )"""

svc =
SVC()

svc.fit(X_train_scaled, y_train)

pred_svc = svc.predict(X_test_scaled)

score_svc =
accuracy_score(y_test, pred_svc)

score_svc

print(classification_report(y_test,
pred_svc))

cm_svc = confusion_matrix(y_test, pred_svc)
sns.heatmap(cm_svc,
annot=True)

"""##ANN Model"""

model =
Sequential()

model.add(Dense(128, activation = 'relu', input_shape =
(30,)))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(64,
activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(6
4,
activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))


model.add(Dense(
16, activation='relu'))
model.add(BatchNormalization())


model.add(Dense(1,
activation='sigmoid'))

model.compile(optimizer='adam', loss = 'binary_crossentropy',
metrics=['accuracy'])

calls = EarlyStopping(patience=5, verbose=1)

res = model.fit(x =
X_train_scaled, y = y_train, validation_split=0.2, epochs=100, verbose=True,
callbacks=calls)

res.history.keys()

pred_ann = model.predict(X_test_scaled)

pred_ann =
pd.Series(model.predict(X_test_scaled).flatten())
pred_ann = (pred_ann >
0.5).astype(int)

model.evaluate(X_test_scaled, y_test)

cm_ann = confusion_matrix(y_test,
pred_ann)
```

```
sns.heatmap(cm_ann, annot=
True)

plt.plot(res.history['accuracy'])
plt.plot(res.history['val_accuracy'])
plt.title('Accur
acy and Validation Accuracy')
plt.legend(['Train',
'Test'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()

plt.plot(res.history['loss'])

plt.plot(res.history['val_loss'])
plt.title('Loss and Validation Loss')
plt.legend(['Train',
'Test'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```