# how to apply models in different conditions

## Classification

- K-Nearest Neighbour (KNN) Classification
- Support Vector Machine (SVM) Classification
- Naive Bayes Classification
- Decision Tree Classification
- Random Forest Classification

## Regression

- Linear Regression
- Multiple Linear Regression
- Polynomial Linear Regression
- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression

## Clustering

- K-Means Clustering
- Hierarchical Clustering

## process

- first we import all the libraries
- than we read our data
- than we do some data exploration before the analysis just to understand the data best
- so if we fuond some null values or if we want to chane them than we can do that
- we can change the data types of te data
- so there are some data exploration commands
- info, head, tail, isnull, notnull, unique, nunique, dtypes, inindex, columns, counts, value_counts and manny more
- now we wil check the correlation b/w the dpata and present that data on the heatmap by sns function
- noow we can drop thos ecolumns that aere not usefull for our data
- and we check corelation so if there are some features that are highly soretated than we can drop one of thatwo nifeature
- now its good two select the data and spliy that data into input and the out put like x and y
- now if ithe data is categorical tan we can do lablel encoding otherwise dont
- before the table encoding we select thge xtrean and y train variables and select the random state and test size than we do table encoding
- now after table encoding we do normalization
- and if the data is not in 2d state than we reeshape the data and then do normalization
- now after fit and transform the normalization to our xtrain and xtest data we will go for the model selection

# EDA(exploratry data analysis)

In [ ]
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]
```
df.info()
```

In [ ]
```
df.describe()
```

In [ ]
```
df.isnull()
```

In [ ]
```
df.corr()
```

In [ ]
```
df.dtypes
```

In [ ]
```
df.shape
```

In [ ]
```
df.columns
```

In [ ]
```
df.index
```

In [ ]
```
sns.countplot(df['c_name'])
plt.show()

sns.countplot(df.diagnosis)
plt.show()
```

In [ ]
```
sns.heatmap(df.c_name
```

In [ ]
```
plt.plot(df['c_name1'], df['c_name2'])
```

In [ ]
```
plt.figure(figsize=(15,9))
plt.scatter(df.c_name1, df.c_name2)
plt.show()
```

In [ ]
```
plt.figure(figsize=(15,9))
sns.jointplot(df.c_name1, df.c_name2)
plt.show()
```

In [ ]
```
sns.pairplot(df[['c_name1', 'c_name2', 'c_name3',
       'c_name4']])
plt.show()
```

In [ ]
```
df['c_name1', 'c_name2', 'c_name3',
       'c_name4'].plot()
```

# prepocessing

In [ ]
```
l = []
for i in df['diagnosis']:
    if i == 'M':
        l.append(1)
    else:
        l.append(0)

col= ['diagnosis']
for i in col:
    df[i] = le.fit_transform(df[i])
```

In [ ]
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [ ]
```
# select x and y
```

In [ ]
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.3, random_state=41)
```

In [ ]
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [ ]
```
# SO HERE WE GOT ONE D ARRAY  but for the normalization weeed the 2d arrayso we reshape our data

x = data.c_name1.values.reshape(-1,1) # or we can do this also
y = data.c_name2.values.reshape(-1,1)

x_train = np.array(x_train).reshape(-1,1)
x_test = np.array(x_test).reshape(-1,1)

scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

# use in every model

In [ ]
```
# y_test y_pred for all
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
```

# 1. linear regresion (only two features)

- multyple regression is same in which we use more then two features as input

In [ ]
```
# now we will import th modle to fit our data
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [ ]
```
lr.fit(x_train, y_train)
lr.coef_ # it will give mus the value of m which is slope
lr.intercept_ # it will give us the value of c which is intercept
```

In [ ]
```
# now we have x value m and c now we have to do is
# y = mx + c so all we have now we can predict the y values

y_pred = lr.predict(x_test)
```

In [ ]
```
plt.scatter(x_test, y_test, label = 'baby', c ='r')
plt.plot(x_test, y_pred, label= 'hubby', c= 'g')
plt.title('graph final')
plt.legend()
plt.show()
```

In [ ]
```
# now we can also check the various errors so we can that too
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test, y_pred))
```

In [ ]
```
# we hav e a command r2score to check the accuracy
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)

print('R sq: ', linear_reg.score(x, y))
print('Correlation: ', math.sqrt(linear_reg.score(x, y))) # we can use tis also
```

# Polynomial Regression

- same as linear regression

In [ ]
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

polynomial_regression = PolynomialFeatures(degree=4)
x_polynomial = polynomial_regression.fit_transform(x,y)

# so fit
linear_regression = LinearRegression()
linear_regression.fit(x_polynomial,y)
# Xb =
y_head2 = linear_regression.predict(x_polynomial)
```

In [ ]
```
from sklearn.metrics import r2_score
print("r_square score: ", r2_score(y,y_head2))
```

# Logistic regression

- knn and log we can use both for same prob

In [ ]
```
from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()
lgr.fit(x_train, y_train)
```

In [ ]
```
y_pred = lgr.predict(x_test)
```

In [ ]
```
# y_test y_pred for all
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
```

# knn

In [ ]
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
```

In [ ]
```
knn.fit(x_train, y_train)
```

In [ ]
```
y_pred = knn.predict(x_test)
```

In [ ]
```
# y_test y_pred for all
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
```

In [ ]
```
confusion_matrix(y_test, prediction)
```

# Naive Bayes Classification

In [ ]
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.3,random_state=1)

# Xb Naive bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train,y_train)

nb.score(x_test,y_test)
```

In [ ]
```
#Nb confusion matrix
y_pred = nb.predict(x_test)
cm = None
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true,y_pred)

# Nb cm visualization
import seaborn as sns

f, ax = plt.subplots(figsize =(5,5))
sns.heatmap(cm,annot = True,linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```

# decission tree

- everry thing is same in this model to used for categorical data

In [ ]
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
```

In [ ]
```
dtree.fit(x_train,y_train)
```

In [ ]
```
dtree.predict(x_test)
```

# Random forest

In [ ]
```
from sklearn.ensemble import RndomForestClassifier
rf = RndomForestClassifier(n_estimators = 51)
rf.fit(x_train,y_train)
rf.predict(x_test)
```

# SVM

## two types

### 1.linear

In [ ]
```
from sklearn.svm import SVC

svm = SVC(random_state = 1)
svm.fit(x_train,y_train)


from sklearn.svm import SVC

svm = SVC(kernal = 'linear')
svm.fit(x_train,y_train)
```

### 2.kerneled svm

In [ ]
```
from sklearn.preprocessing import StandardScaler
sc1 = StandardScaler()
x_scaled1 = sc1.fit_transform(x)
sc2 = StandardScaler()
y_scaled1 = sc2.fit_transform(y)


from sklearn.svm import SVR

svr_reg = SVR(kernel = 'rbf')
svr_reg.fit(x_scaled1,y_scaled1)

y_head = svr_reg.predict(x_scaled1)
```