```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving faces_target.npy to faces_target (1).npy
```

Dataset-1

Face recognition-Identifying and verifying people in a photograph by their face.

Congratulations for your new Job...!!!

You see many new faces in your new office. Sometime it is very disrespectful not to address a person by his or her name. Hence Datascientist in you is planning to make a model which will help you to identify all new colleague's name by their photograph.

There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). The image is quantized to 256 grey levels and stored as unsigned 8-bit integers; the loader will convert these to floating point values on the interval [0, 1].

The “target” for this database is an integer from 0 to 39 indicating the identity of the person pictured; however, with only 10 examples per class.

Image Credit: AT&T Laboratories Cambridge.

Step-1: Load the files given below in colab.

Target_data: https://drive.google.com/file/d/1_dZ2-ea3N32DKyr3sZisnandpatFy2BA/view?usp=sharing

Features_data: https://drive.google.com/file/d/1LAjKw0O73XwSeNvmUqrHmwBFcIxVit0q/view?usp=sharing

Step-2: Check the shape of your dataset.

Step-3: Write a code to plot first 40 image.Comment Your observation.

Step-4: Write the code to plot all image whose index difference is 10. (like 10,20,30).

Step-5: Use train-test split to split the dataset by keeping random state=10 and test size =0.2.

Step-6: Check the shape of X_train,Y_train,X_test,Y_test.

Step-7: Train your model using Logistic regression,Softmax regression and KNN Classifier estimator.

Step-8: Create a dataframe which shows all the estimator with corresponding accuracy.

Step-9: Plot the confusion matrix for the result obtained using KNN classifier.

Based on the above step Please answer the following Question:

Que 1) What is the shape of your dataset.

Que 2) How many labels are there in the dataset.

Que 3) How many examples are there with each class.

Que-4) What is the shape of X_train and Y_train.

Que-5) Convert x_train and x_test shape into (a,b) type.

Que-7) What is the accuracy you got with logistic regression and KNN classsification.

Que-8) What is the f1_score you got with softmax regression.

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')


?train_test_split


from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving faces.npy to faces (1).npy
Saving faces_target.npy to faces_target (1).npy
```

```
pics=np.load("faces.npy")
pics.shape
```

```
(400, 64, 64)
```

```
pics[::1]
```

```
array([[[0.3181818 , 0.40082645, 0.49173555, ..., 0.40082645,
         0.3553719 , 0.30991736],
        [0.30991736, 0.3966942 , 0.47933885, ..., 0.40495867,
         0.37603307, 0.30165288],
        [0.26859504, 0.34710744, 0.45454547, ..., 0.3966942 ,
         0.37190083, 0.30991736],
        ...,
        [0.1322314 , 0.09917355, 0.08264463, ..., 0.13636364,
         0.14876033, 0.15289256],
        [0.11570248, 0.09504132, 0.0785124 , ..., 0.1446281 ,
         0.1446281 , 0.1570248 ],
        [0.11157025, 0.09090909, 0.0785124 , ..., 0.14049587,
         0.14876033, 0.15289256]],

       ...,

       [[0.5       , 0.53305787, 0.607438  , ..., 0.28512397,
         0.23966943, 0.21487603],
        [0.49173555, 0.5413223 , 0.60330576, ..., 0.29752067,
         0.20247933, 0.20661157],
        [0.46694216, 0.55785125, 0.6198347 , ..., 0.29752067,
         0.17768595, 0.18595041],
        ...,
        [0.03305785, 0.46280992, 0.5289256 , ..., 0.17355372,
         0.17355372, 0.1694215 ],
        [0.1570248 , 0.5247934 , 0.53305787, ..., 0.16528925,
         0.1570248 , 0.18595041],
        [0.45454547, 0.5206612 , 0.53305787, ..., 0.17768595,
         0.14876033, 0.19008264]],

       [[0.21487603, 0.21900827, 0.21900827, ..., 0.71487606,
         0.71487606, 0.6942149 ],
        [0.20247933, 0.20661157, 0.20661157, ..., 0.7107438 ,
         0.7066116 , 0.6942149 ],
        [0.2107438 , 0.20661157, 0.20661157, ..., 0.6859504 ,
         0.69008267, 0.6942149 ],
        ...,
        [0.2644628 , 0.25619835, 0.2603306 , ..., 0.5413223 ,
         0.57438016, 0.59090906],
        [0.26859504, 0.2644628 , 0.26859504, ..., 0.56198347,
         0.58264464, 0.59504133],
        [0.27272728, 0.26859504, 0.27272728, ..., 0.57438016,
         0.59090906, 0.60330576]],

       [[0.5165289 , 0.46280992, 0.28099173, ..., 0.5785124 ,
         0.5413223 , 0.60330576],
        [0.5165289 , 0.45041323, 0.29338843, ..., 0.58264464,
         0.553719  , 0.5785124 ],
        [0.5165289 , 0.44214877, 0.29338843, ..., 0.59917355,
         0.5785124 , 0.54545456],
        ...,
        [0.39256197, 0.41322315, 0.38842976, ..., 0.33471075,
         0.37190083, 0.3966942 ],
        [0.39256197, 0.38429752, 0.40495867, ..., 0.3305785 ,
```

```
 0.35950413, 0.37603307],
[0.3677686 , 0.40495867, 0.3966942 , ..., 0.35950413,
 0.3553719 , 0.38429752]]], dtype=float32)
```

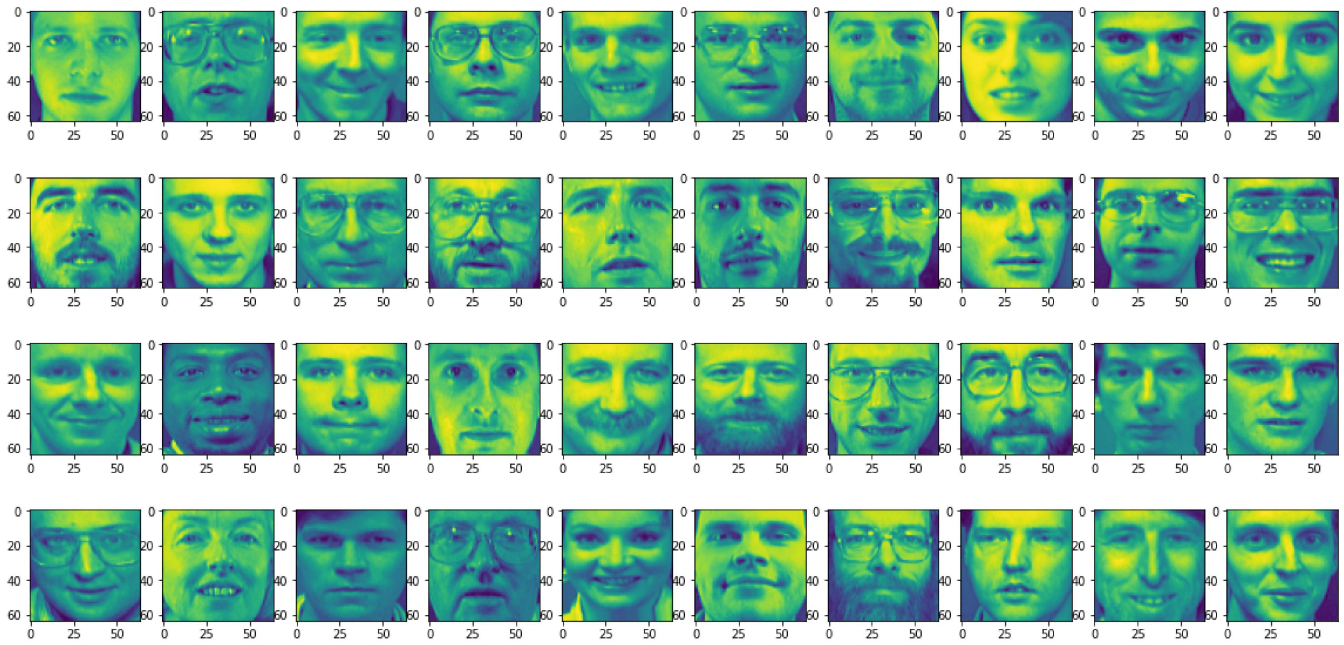
```
labels=np.load("faces_target.npy")
labels.shape
```

```
(400,)
```

```
labels
```

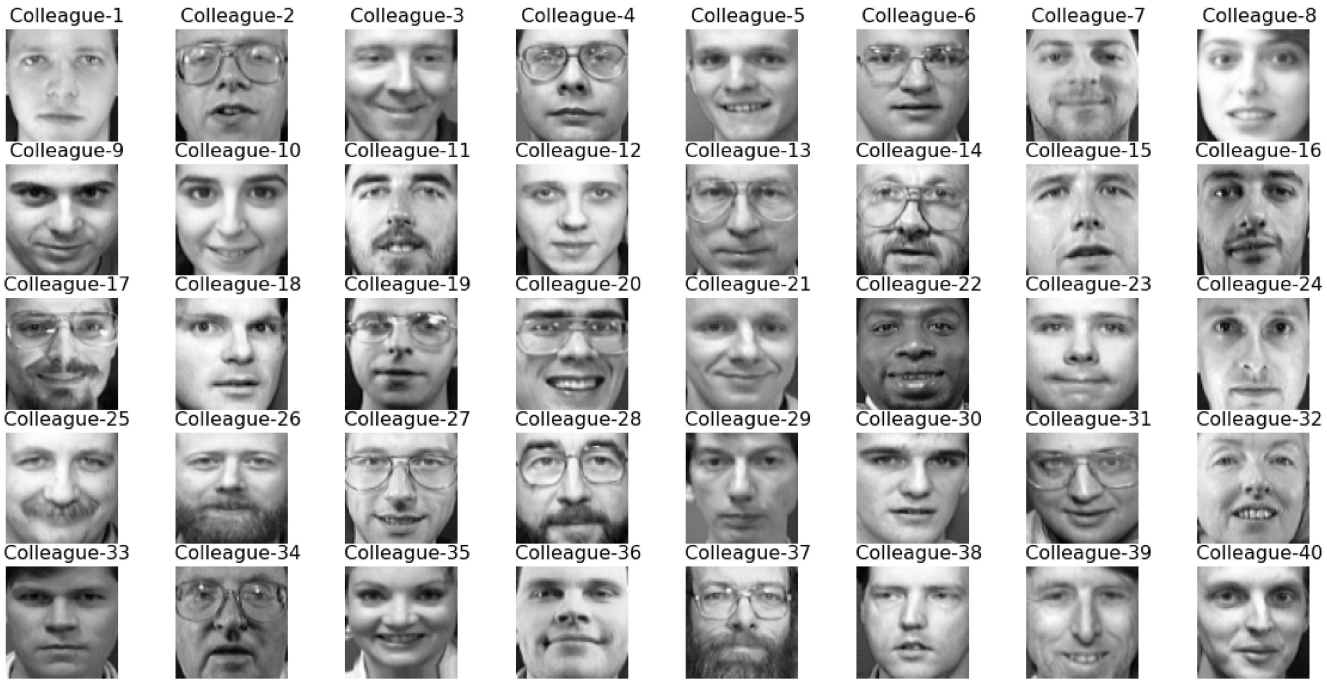
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  5,
        5,  5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 13,
       13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 15, 15, 15,
       15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
       17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18,
       18, 18, 18, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20,
       20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22,
       22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 23, 23, 23, 23,
       23, 23, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 25,
       25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 27, 27,
       27, 27, 27, 27, 27, 27, 27, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28,
       28, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 30, 30, 30, 30, 30, 30,
       30, 30, 30, 30, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32,
       32, 32, 32, 32, 32, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33,
       34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 35, 35, 35, 35, 35, 35, 35,
       35, 35, 35, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 37, 37, 37, 37,
       37, 37, 37, 37, 37, 37, 38, 38, 38, 38, 38, 38, 38, 38, 38, 38, 39,
       39, 39, 39, 39, 39, 39, 39, 39], dtype=int32)
```

```
fig = plt.figure(figsize=(20, 10))
columns = 10
rows = 4
j=1
for i in range(1,400,10):
    img = pics[i-1,:,:]
    fig.add_subplot(rows, columns, j)
    j+=1
    plt.imshow(img, cmap = plt.get_cmap("viridis"))#colormap possible values = default, virid
plt.show()
```



```
fig = plt.figure(figsize=(20, 10))
columns = 8
rows = 5
for i in range(1, columns*rows +1):
    img = pics[10*(i-1),:,:]
    fig.add_subplot(rows, columns, i)
    plt.imshow(img, cmap = plt.get_cmap('gray'))#colormap possible values = default, viridis,
    plt.title("  Colleague-{}".format(i), fontsize=16)
    plt.axis('off')

plt.suptitle("All the 40 people's face data we have plotted here", fontsize=25)
plt.show()
```




```
X= pics # store images in Xdata
Y= labels.reshape(-1,1) # store labels in Ydata


x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state=10)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((320, 64, 64), (80, 64, 64), (320, 1), (80, 1))
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state=10)

print("x_train: ",x_train.shape)
print("x_test: ",x_test.shape)
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)
```

```
x_train:  (320, 64, 64)
x_test:  (80, 64, 64)
y_train:  (320, 1)
y_test:  (80, 1)
```

```
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]*x_train.shape[2])
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]*x_test.shape[2])

print("x_train: ",x_train.shape)
print("x_test: ",x_test.shape)
```

```
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)
```

```
x_train:  (320, 4096)
x_test:  (80, 4096)
y_train:  (320, 1)
```

```
Estimator_list = []
Accuracy_list= []


from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
lr = LogisticRegression(max_iter=1000)
lr.fit(x_train, y_train)
model_accuracy = round(lr.score(x_test, y_test)*100,2)

print("LogReg model accuracy is %", model_accuracy)

Estimator_list.append("Logistic Regression")
Accuracy_list.append(model_accuracy)
```

```
LogReg model accuracy is % 96.25
```

```
from sklearn.svm import SVC
from sklearn.metrics import f1_score
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(x_train, y_train)
model_accuracy = round(classifier.score(x_test, y_test)*100,2)
print("SVM  model accuracy is %", model_accuracy)

Estimator_list.append("Support vector classifier")
Accuracy_list.append(model_accuracy)
```

```
SVM  model accuracy is % 95.0
```

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=1000,multi_class="multinomial",solver='sag')
lr.fit(x_train, y_train)
softmax_accuracy = round(lr.score(x_test, y_test)*100,2)

print("softmax_accuracy is %", softmax_accuracy)

Estimator_list.append("softmax Regression")
Accuracy_list.append(softmax_accuracy)
```

```
softmax_accuracy is % 96.25
```

```
?LogisticRegression
```

```
y_pred=lr.predict(x_test)
ans=f1_score(y_test, y_pred, average='macro')


print("f1 score for softmax regression is:",ans)
```

```
f1 score for softmax regression is: 0.9626984126984125
```

```
from sklearn.neighbors import KNeighborsClassifier

Knn = KNeighborsClassifier(n_neighbors = 1) # n_neighbors=1 gives the best result for this da
Knn.fit(x_train, y_train)
Knn_accuracy = round(Knn.score(x_test, y_test)*100,2)

print("Knn_accuracy is %", Knn_accuracy)

Estimator_list.append("KNN")
Accuracy_list.append(Knn_accuracy)
```

```
Knn_accuracy is % 92.5
```

```
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[1,2,3,4,5,6,7,8,9,10,11]}

knn = KNeighborsClassifier()

model = GridSearchCV(knn, params, cv=5,)
model.fit(x_train,y_train)
model.best_params_
```

```
{'n_neighbors': 1}
```

```
df = pd.DataFrame({'METHOD': Estimator_list, 'ACCURACY (%)': Accuracy_list})
df = df.sort_values(by=['ACCURACY (%)'])
df = df.reset_index(drop=True)
df.head()
```

	METHOD	ACCURACY (%)
0	KNN	92.50
1	Logistic Regression	96.25
2	softmax Regression	96.25

Dataset-2 (Insurance decisioning)

The insurance industry has been variously described as laggard, sloth-like and conservative in its responses to digital disruption that is underway, But not anymore. Datascientist Like you will help them in making the best use of new-age technologies to constantly innovate, ramp up customer satisfaction, stay ahead in the race and to provide personalized products, better service.

Detecting risks early in the process enables insurers to make better use of underwriters' time and gives them a huge competitive advantage.Use appropriate machine learning algorithms to predict future medical expenses of individuals that help your company to make decision on charging the premiums of insurance based on data collected from client drivers.

Step-1:Load the files given below in colab.(Link: https://drive.google.com/file/d/1vnsENE26qOH5y2UBEZcFG-AdF3VsKtZt/view?usp=sharing)

Step-2: Check the shape,info of your dataset.

Step-3: Check if any duplicate sample is there in the dataset.

Step-4: Remove duplicate data and perform one hot encoding for categorical data

Step-5: Use train-test split to split the dataset by keeping random state=10 and test size =0.2.

Step-6: Check the shape of X_train,Y_train,X_test,Y_test.

Step-7: Train your model using KNN regressior estimator.

Step-8: Create a dataframe which shows k value with corresponding error..

Step-9: Plot the error curve and answer which K value will give the best result.

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving medical expnse pred.csv to medical expnse pred (1).csv

```
data=pd.read_csv("/content/medical_expnse_pred.csv")
data.head(5)
```

	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   expenses  1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
data.describe()
```

	age	bmi	children	expenses
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.665471	1.094918	13270.422414
std	14.049960	6.098382	1.205493	12110.011240
min	18.000000	16.000000	0.000000	1121.870000
25%	27.000000	26.300000	0.000000	4740.287500
50%	39.000000	30.400000	1.000000	9382.030000
75%	51.000000	34.700000	2.000000	16639.915000
max	64.000000	53.100000	5.000000	63770.430000

```
data['sex'].unique()
```

```
array(['female', 'male'], dtype=object)
```

```
data['smoker'].value_counts()
```

```
no     1064
yes     274
Name: smoker, dtype: int64
```

```
data[data.duplicated()]
```

	age	sex	bmi	children	smoker	region	expenses
581	19	male	30.6	0	no	northwest	1639.56

```
data2=data.copy()
```

```
data2.drop_duplicates(inplace=True)
```

```
data2.shape
```

```
(1337, 7)
```

```
cat_cols = data2.select_dtypes(exclude = 'number')
cat_cols.columns
```

```
Index(['sex', 'smoker', 'region'], dtype='object')
```

```
cat_cols
```

	sex	smoker	region
0	female	yes	southwest

```
num_cols = data2.select_dtypes(include = 'number')
num_cols
```

	age	bmi	children	expenses
0	19	27.9	0	16884.92
1	18	33.8	1	1725.55
2	28	33.0	3	4449.46
3	33	22.7	0	21984.47
4	32	28.9	0	3866.86
...	...	...	...	...
1333	50	31.0	3	10600.55
1334	18	31.9	0	2205.98
1335	18	36.9	0	1629.83
1336	21	25.8	0	2007.95
1337	61	29.1	0	29141.36

1337 rows × 4 columns

```
onehot_cat_cols = pd.get_dummies(cat_cols)
```

```
onehot_cat_cols.head()
```

	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	reg
0	1	0	0	1	0	0	
1	0	1	1	0	0	0	
2	0	1	1	0	0	0	
3	0	1	1	0	0	1	
4	0	1	1	0	0	1	

```
df_final = pd.concat([num_cols,onehot_cat_cols],sort=True,axis=1)
```

```
df_final.head(2)
```

	age	bmi	children	expenses	sex_female	sex_male	smoker_no	smoker_yes	region_no
0	19	27.9	0	16884.92	1	0	0	1	
1	18	33.8	1	1725.55	0	1	1	0	

```
X = df_final.drop('expenses',axis=1)
```

```
y = df_final['expenses']
```

```
from sklearn.model_selection import train_test_split
X_train , X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25, random_state = 0)
```

```
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
((1002, 11), (1002,), (335, 11), (335,))
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt,ceil
import matplotlib.pyplot as plt
%matplotlib inline
```

```
sqrt(data2.shape[0])
```

```
36.565010597564445
```

```
l = ceil(sqrt(data2.shape[0]))#closest int value
```

```
l
```

```
37
```

```
rmse = []
for k in range(0,l+1):
    k = k+1
    model = KNeighborsRegressor(n_neighbors=k)
    model.fit(X_train,y_train)
    y_test_pred = model.predict(X_test)
    rmse_error = sqrt(mean_squared_error(y_test,y_test_pred))
    rmse.append(rmse_error)
    print('RMSE value for k=' , k , 'is:', rmse_error)
```

```
RMSE value for k= 1 is: 11786.012159320124
RMSE value for k= 2 is: 10607.706606488884
```

```
RMSE value for k= 3 is: 10457.284072294755
RMSE value for k= 4 is: 10231.018956732843
RMSE value for k= 5 is: 10479.92545821663
RMSE value for k= 6 is: 10518.982322122994
RMSE value for k= 7 is: 10606.859351174191
RMSE value for k= 8 is: 10710.863199980842
RMSE value for k= 9 is: 10898.48708940924
RMSE value for k= 10 is: 11046.595393735017
RMSE value for k= 11 is: 11108.998902776999
RMSE value for k= 12 is: 11168.236052821367
RMSE value for k= 13 is: 11176.939159668093
RMSE value for k= 14 is: 11224.03134266892
RMSE value for k= 15 is: 11291.457798448362
RMSE value for k= 16 is: 11341.40626924684
RMSE value for k= 17 is: 11370.68513483235
RMSE value for k= 18 is: 11365.430682027109
RMSE value for k= 19 is: 11359.428434387266
RMSE value for k= 20 is: 11361.252994222368
RMSE value for k= 21 is: 11388.017032555212
RMSE value for k= 22 is: 11339.873958602495
RMSE value for k= 23 is: 11376.842831536294
RMSE value for k= 24 is: 11350.230214339981
RMSE value for k= 25 is: 11275.938799657779
RMSE value for k= 26 is: 11312.448086351798
RMSE value for k= 27 is: 11377.270562081676
RMSE value for k= 28 is: 11347.505028291389
RMSE value for k= 29 is: 11363.571894245391
RMSE value for k= 30 is: 11389.731281846163
RMSE value for k= 31 is: 11402.948421729137
RMSE value for k= 32 is: 11420.615830023153
RMSE value for k= 33 is: 11404.361315347996
RMSE value for k= 34 is: 11430.145968422936
RMSE value for k= 35 is: 11429.494839831841
RMSE value for k= 36 is: 11416.016083575605
RMSE value for k= 37 is: 11435.212212491442
RMSE value for k= 38 is: 11454.323764895762
```

```
min(rmse)
```

```
10231.018956732843
```

```
#from sklearn.model_selection import GridSearchCV
#params = {'n_neighbors':[2,3,4,5,6,7,8,9]}

#knn = KNeighborsRegressor()

#model = GridSearchCV(knn, params, cv=5,scoring="neg_mean_squared_error")
#model.fit(X_train,y_train)
#model.best_params_


error_curve = pd.DataFrame(rmse,columns=['error'])
```

```
error_curve.head()
```

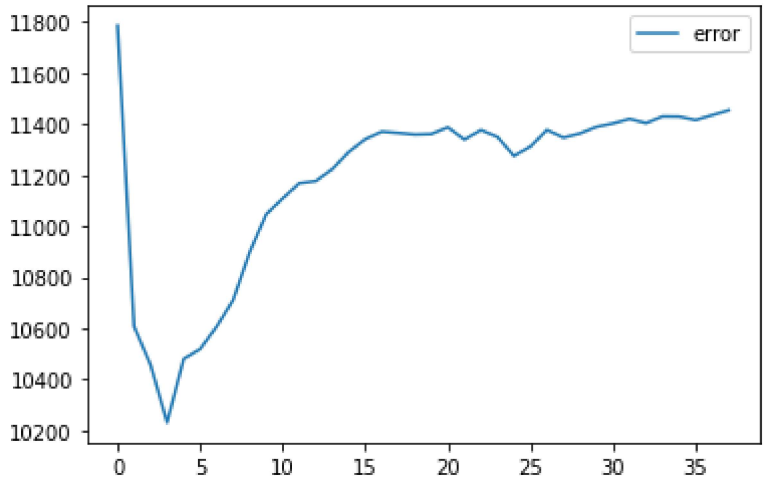
	error
0	11786.012159
1	10607.706606
2	10457.284072
3	10231.018957
4	10479.925458

```
error_curve.describe()
```

	error
count	38.000000
mean	11184.898671
std	354.798192
min	10231.018957
25%	11123.808190
50%	11348.867621
75%	11389.302720
max	11786.012159

```
error_curve.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1de5b54950>
```



Conclusion:

RMSE value is minimum at k = 4

Regression for Large Scale data :

1) Create a large dataset having 100000 samples and 50 features.

2)Split it using train_test_split.

3)Reshape it in such a manner that your each chunk contain 100 samples.

4) Import stdscalar and transform features using partial_fit method.

5) Import SGDRegressor estimator and train your model using it.

6) Check accuracy of your model on train and test set.

7) Print the intercept and cofficient value coressponding to each features.

8) Check the value of cofficient and intercept after 4th iteration.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split

X, Y = datasets.make_regression(n_samples=100000,n_features=50, noise=10, random_state=10)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.9, random_state=10)

X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

```
((90000, 50), (10000, 50), (90000,), (10000,))
```

```
X_train, X_test = X_train.reshape(-1,100,50), X_test.reshape(-1,100,50)
Y_train, Y_test = Y_train.reshape(-1,100), Y_test.reshape(-1,100)

X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

```
((900, 100, 50), (100, 100, 50), (900, 100), (100, 100))
```

No of training samples=90,000 What we want is make chunk of 100 data

```
X_train.shape[2]
```

```
50
```

```
import numpy as np
train_data = np.concatenate((X, Y[:, np.newaxis]), axis=1)
a = np.asarray(train_data)
np.savetxt("data_for_large_scale_swi.csv", a, delimiter=",")
```

```
from sklearn.preprocessing import StandardScaler

### Scaling Data

scaler = StandardScaler()

for i in range(X_train.shape[0]):
    X_batch, Y_batch = X_train[i], Y_train[i]
    scaler.partial_fit(X_batch, Y_batch) ## Partially fitting data in batches
```

```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error,r2_score

regressor = SGDRegressor()

for i in range(X_train.shape[0]): ## Looping through batches
    X_batch, Y_batch = X_train[i], Y_train[i]
    new_X_batch = scaler.transform(X_batch)
    regressor.partial_fit(new_X_batch, Y_batch) ## Partially fitting data in batches

Y_test_preds = []
for j in range(X_test.shape[0]): ## Looping through test batches for making predictions
    Y_preds = regressor.predict(X_test[j])
    Y_test_preds.extend(Y_preds.tolist())

print("Test MSE      : {}".format(mean_squared_error(Y_test.reshape(-1), Y_test_preds)))
print("Test R2 Score : {}".format(r2_score(Y_test.reshape(-1), Y_test_preds)))
```

```
Test MSE      : 101.89979220276909
Test R2 Score : 0.9942576140935281
```

```
from sklearn.metrics import mean_squared_error, r2_score

Y_train_preds = []
for j in range(X_train.shape[0]): ## Looping through train batches for making predictions
    Y_preds = regressor.predict(X_train[j])
    Y_train_preds.extend(Y_preds.tolist())

print("Train MSE      : {}".format(mean_squared_error(Y_train.reshape(-1), Y_train_preds)))
print("Train R2 Score : {}".format(r2_score(Y_train.reshape(-1), Y_train_preds)))
```

```
Train MSE      : 100.70360644200434
```

```
Train R2 Score : 0.994344394140802
```

```
regressor.intercept_
```

```
array([0.00910036])
```

```
regressor.coef_
```

```
array([ 1.22858911e-01,  3.56265990e-02, -1.84264088e-02,  4.62869891e-02,
        3.94421827e-02, -4.51722564e-02,  4.24056098e+01,  1.12709367e-02,
        2.10916472e+01,  4.45498221e-01,  7.39023248e-02,  3.24007458e+01,
       -9.06563689e-02, -3.23803985e-03, -3.15794270e-02, -3.56630856e-01,
       -3.62863372e-01,  3.54598664e+01, -2.15556009e-01, -1.11394956e-01,
        1.40201036e-01,  2.75438264e-01,  3.83329721e+01, -2.88137637e-02,
       -1.61753769e-01, -5.65846932e-02,  7.39882945e+01, -2.28578541e-02,
        1.73627799e-02,  1.29399874e-01, -3.73811969e-02,  1.28012517e-01,
       -1.44659379e-01, -1.06260589e-02, -3.80728739e-02,  9.05423238e+00,
        4.84635306e-01, -5.22314373e-02,  2.17063876e-01, -3.10198595e-02,
        5.52013486e+01,  4.28792266e-01, -1.68351107e-01, -6.36379165e-04,
        5.61569552e+01, -2.58578145e-01, -3.24459239e-01, -2.50540093e-01,
       -1.25112000e-01,  9.27576042e-02])
```

Cofficient after nth iteration

```
X_train.shape[0]
```

```
900
```

```
from sklearn.linear_model import SGDRegressor

regressor = SGDRegressor()

for i in range(X_train.shape[0]): ## Looping through batches
    X_batch, Y_batch = X_train[i], Y_train[i]
    scaled_X= scaler.transform(X_batch)
    regressor.partial_fit(scaled_X, Y_batch) ## Partially fitting data in batches
    print(i,regressor.intercept_)
    if i==3:
      break
```

```
0 [-2.38014267]
1 [-1.08887003]
2 [-1.61727297]
3 [0.70578256]
```

```
regressor.intercept_
```

```
array([0.70578256])
```

```
regressor.coef_
```

```
array([-1.8204058 , -0.5596251 ,  0.31086477, -3.11024724,  1.74667944,
       -4.23118478, 31.27091646,  0.48956723, 18.12701049, -2.0464112 ,
       -2.10087935, 18.18226694,  1.33801961,  0.38225181, -2.89185451,
        2.2581106 , -1.41999896, 23.99040883, -3.33768337,  0.87297804,
        0.62959545,  2.86221251, 27.44717267, -4.09704071, -1.38198557,
       -3.31895449, 55.23059448, -2.04706595,  1.02094844, -0.95738499,
       -2.12533548,  2.7425588 ,  2.86860907, -3.33238789,  1.58978959,
        5.02711918,  0.40521869, -1.29407923,  5.33438916,  2.05774743,
       39.50438811,  0.21721716, -0.45920908,  0.59296577, 39.73803301,
       -6.30482578,  2.09222435,  0.94741819,  0.27090429, -2.95363339])
```