Importing Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import plotly.express as px
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
```

Downloading the training and testing dataset

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/satimage/sat.trn
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/satimage/sat.tst
```

```
--2021-04-03 08:48:59--  https://archive.ics.uci.edu/ml/machine-learning-databases/statl
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connectec
HTTP request sent, awaiting response... 200 OK
Length: 525830 (514K) [application/x-httpd-php]
Saving to: ‘sat.trn.2’

sat.trn.2           100%[===================>] 513.51K  --.-KB/s    in 0.1s

2021-04-03 08:48:59 (3.61 MB/s) - ‘sat.trn.2’ saved [525830/525830]

--2021-04-03 08:48:59--  https://archive.ics.uci.edu/ml/machine-learning-databases/statl
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connectec
HTTP request sent, awaiting response... 200 OK
Length: 236745 (231K) [application/x-httpd-php]
Saving to: ‘sat.tst.2’

sat.tst.2           100%[===================>] 231.20K  --.-KB/s    in 0.1s

2021-04-03 08:48:59 (1.95 MB/s) - ‘sat.tst.2’ saved [236745/236745]
```

Reading and Storing training and testing dataframe

```
train = pd.DataFrame(np.genfromtxt('sat.trn'))
train.rename(columns={train.columns[-1]: "Class" }, inplace=True)
```

```
print(train.head(20))
```

```
        0      1      2     3     4  ...    32     33     34    35  Class
0   92.0  115.0  120.0  94.0  84.0  ...  84.0  107.0  113.0  87.0    3.0
```

```
1   84.0  102.0  106.0  79.0  84.0  ...  84.0   99.0  104.0  79.0    3.0
2   84.0  102.0  102.0  83.0  80.0  ...  84.0   99.0  104.0  79.0    3.0
3   80.0  102.0  102.0  79.0  84.0  ...  84.0  103.0  104.0  79.0    3.0
4   84.0   94.0  102.0  79.0  80.0  ...  79.0  107.0  109.0  87.0    3.0
5   80.0   94.0   98.0  76.0  80.0  ...  79.0  107.0  109.0  87.0    3.0
6   76.0  102.0  106.0  83.0  76.0  ...  79.0  103.0  104.0  79.0    3.0
7   76.0  102.0  106.0  87.0  80.0  ...  79.0   95.0  100.0  79.0    3.0
8   76.0   89.0   98.0  76.0  76.0  ...  79.0   87.0   93.0  71.0    4.0
9   76.0   94.0   98.0  76.0  76.0  ...  79.0   87.0   93.0  67.0    4.0
10  76.0   98.0  102.0  72.0  76.0  ...  75.0   87.0   96.0  71.0    4.0
11  72.0   94.0   90.0  72.0  72.0  ...  71.0   87.0   89.0  67.0    4.0
12  72.0   89.0   94.0  76.0  72.0  ...  71.0   79.0   81.0  62.0    4.0
13  76.0   94.0   98.0  76.0  72.0  ...  67.0   75.0   85.0  62.0    4.0
14  68.0   85.0   86.0  68.0  68.0  ...  71.0   75.0   81.0  62.0    4.0
15  68.0   89.0   86.0  72.0  68.0  ...  67.0   75.0   85.0  71.0    4.0
16  68.0   85.0   90.0  76.0  68.0  ...  67.0   75.0   96.0  79.0    4.0
17  68.0   94.0   94.0  79.0  76.0  ...  75.0   83.0   96.0  83.0    4.0
18  80.0   94.0  102.0  83.0  80.0  ...  84.0   99.0  109.0  87.0    3.0
19  88.0  106.0  115.0  87.0  88.0  ...  88.0  107.0  109.0  83.0    3.0

[20 rows x 37 columns]
```

```
test = pd.DataFrame(np.genfromtxt('sat.tst'))
test.rename(columns={test.columns[-1]: "Class" }, inplace=True)
```

```
print(test.sample(20))
```

```
         0      1      2      3     4  ...    32     33     34     35  Class
672   68.0   77.0   90.0   72.0  68.0  ...  67.0   68.0   89.0   79.0    5.0
715   43.0   29.0  113.0  114.0  43.0  ...  40.0   32.0  100.0  107.0    2.0
838   64.0   73.0   74.0   57.0  64.0  ...  67.0   72.0   74.0   58.0    7.0
925   89.0  106.0  110.0   83.0  82.0  ...  76.0   87.0   91.0   67.0    4.0
1158  66.0  109.0  122.0  100.0  66.0  ...  63.0  111.0  124.0  101.0    1.0
1916  63.0   66.0   90.0   79.0  63.0  ...  88.0  111.0  120.0   94.0    1.0
1651  72.0   85.0   98.0   79.0  64.0  ...  51.0   51.0   81.0   79.0    5.0
1971  60.0   85.0   94.0   79.0  60.0  ...  63.0   79.0  100.0   87.0    1.0
1605  67.0   95.0  105.0   86.0  67.0  ...  76.0   89.0  115.0   94.0    5.0
1897  52.0   56.0   80.0   74.0  59.0  ...  57.0   67.0   85.0   76.0    5.0
28    84.0   98.0  102.0   79.0  80.0  ...  71.0   75.0   85.0   67.0    7.0
1584  66.0  113.0  127.0  100.0  66.0  ...  63.0  106.0  114.0   90.0    1.0
1848  63.0   91.0  100.0   83.0  67.0  ...  63.0   91.0  101.0   86.0    1.0
1967  76.0  106.0  115.0   94.0  76.0  ...  79.0  103.0  123.0  100.0    1.0
200   88.0  103.0  113.0   85.0  88.0  ...  86.0  100.0  108.0   81.0    3.0
467   97.0  115.0  119.0   94.0  93.0  ...  76.0   91.0   96.0   74.0    4.0
1777  44.0   31.0  114.0  140.0  44.0  ...  46.0   39.0  108.0  114.0    2.0
559   63.0   64.0   85.0   67.0  67.0  ...  67.0   66.0   72.0   53.0    7.0
1387  76.0   87.0   91.0   63.0  80.0  ...  66.0   79.0   76.0   59.0    4.0
1631  88.0  106.0  106.0   87.0  88.0  ...  93.0  107.0  109.0   87.0    3.0

[20 rows x 37 columns]
```

```
x_test = test[test.columns[:-1]].to_numpy()
y_test = test[test.columns[-1]].to_numpy()
print(type(x_test))
```

```
<class 'numpy.ndarray'>
```

```
print(train.columns[-1])
```

```
Class
```

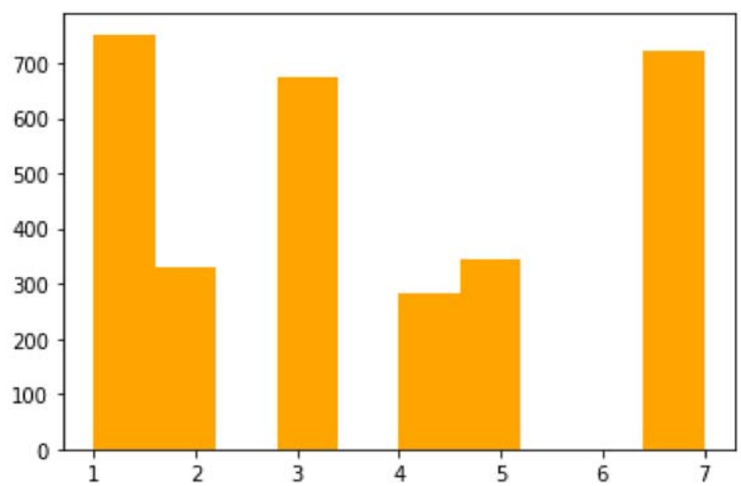
Training and Validation Split

Training and validation set split in 70:30 ratio.

```
x_train, x_val, y_train, y_val = train_test_split(train[train.columns[:-1]].to_numpy(), train
```

```
plt.hist(y_train,color='orange')
```

```
(array([752., 329.,   0., 673.,   0., 283., 345.,   0.,   0., 722.]),
 array([1. , 1.6, 2.2, 2.8, 3.4, 4. , 4.6, 5.2, 5.8, 6.4, 7. ]),
 <a list of 10 Patch objects>)
```




```
print(len(x_train),len(x_val),len(x_test))
```

```
3104 1331 2000
```

TSNE Plot for data visualisaiton

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2,perplexity=50, random_state=6, n_jobs=-1).fit_transform(train)
```

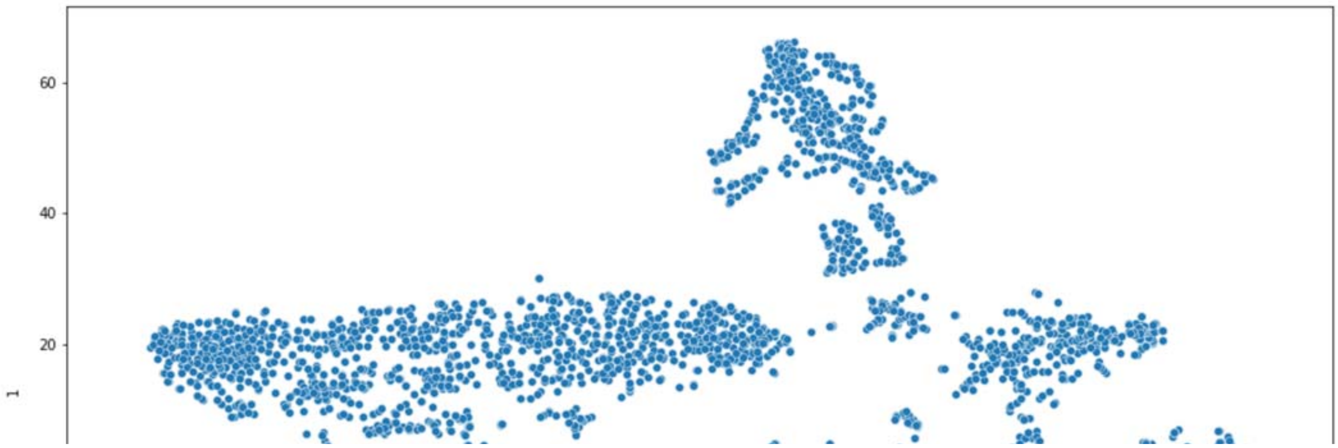
```
tsne_df = pd.DataFrame(tsne)
tsne_df.sample(10)
```

```
tsne_df.sample(10)
```

	0	1
1485	58.583157	-1.797761
258	6.324732	-22.913366
1518	21.538527	45.873734
3715	-15.998539	-30.814167
2325	22.539925	23.036531
2466	34.453903	15.857704
1015	-54.343067	-31.753153
3028	-46.861969	20.278341
1977	40.363178	-7.454031
232	-13.745310	-24.735798

```
plt.figure(figsize=(16,10))
# colorsIdx = {'A': 'rgb(215,48,39)', 'B': 'rgb(215,148,39)'}
# cols      = ['orange','blue']
sns.scatterplot(
    x=0, y=1,
    palette=sns.color_palette("hls", 10),
    data=tsne_df,
    legend="full",
    alpha=1
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe1639d1f90>
```




```
test_x = np.array([[1,2,3] , [-4,5,-6]])
test_xt = np.array([[0,0,0], [1,2,3], [4,5,6], [-4, 4, -6]])
test_yt = np.array([[1, 1, 2, 3]]).T
```

Double-click (or enter) to edit

kNN algorithm implementation from scratch

```
from scipy.spatial.distance import cdist


def get_p_y_x_using_knn(y, k, classes):
    first_k_neighbors = y[:, :k]

    N1, N2 = y.shape
    number_of_classes = classes.shape[0]

    prob_matrix = np.zeros(shape=(N1, number_of_classes))

    for i, row in enumerate(first_k_neighbors):
        for j, value in enumerate(classes):
            prob_matrix[i][j] = list(row).count(value) / k

    return prob_matrix


def predict(X_test, X_train, y_train, k):
    order = cdist(X_test, X_train, metric="euclidean").argsort(kind='mergesort')

    sorted_labels = np.squeeze(y_train[order])
    classes = np.unique(sorted_labels)
    # print(classes)
    p_y_x = get_p_y_x_using_knn(sorted_labels, k, classes)
    number_of_classes = p_y_x.shape[1]
    reversed_rows = np.fliplr(p_y_x)
```

```
    reversed_rows = np.flipr(p_y_x)

    prediction = classes[number_of_classes - (np.argmax(reversed_rows, axis=1) + 1)]
    return prediction
```

```
from sklearn.metrics import classification_report, accuracy_score
y_predict = predict(x_val, x_train, y_train,10)
```

```
print(classification_report(y_val, y_predict))
```

```
              precision    recall  f1-score   support

         1.0       0.98      0.97      0.97       320
         2.0       0.96      0.94      0.95       150
         3.0       0.88      0.93      0.91       288
         4.0       0.68      0.59      0.63       132
         5.0       0.88      0.82      0.85       125
         7.0       0.87      0.90      0.88       316

    accuracy                           0.89      1331
   macro avg       0.87      0.86      0.87      1331
weighted avg       0.89      0.89      0.89      1331
```

Finding Optimal value of k using grid search

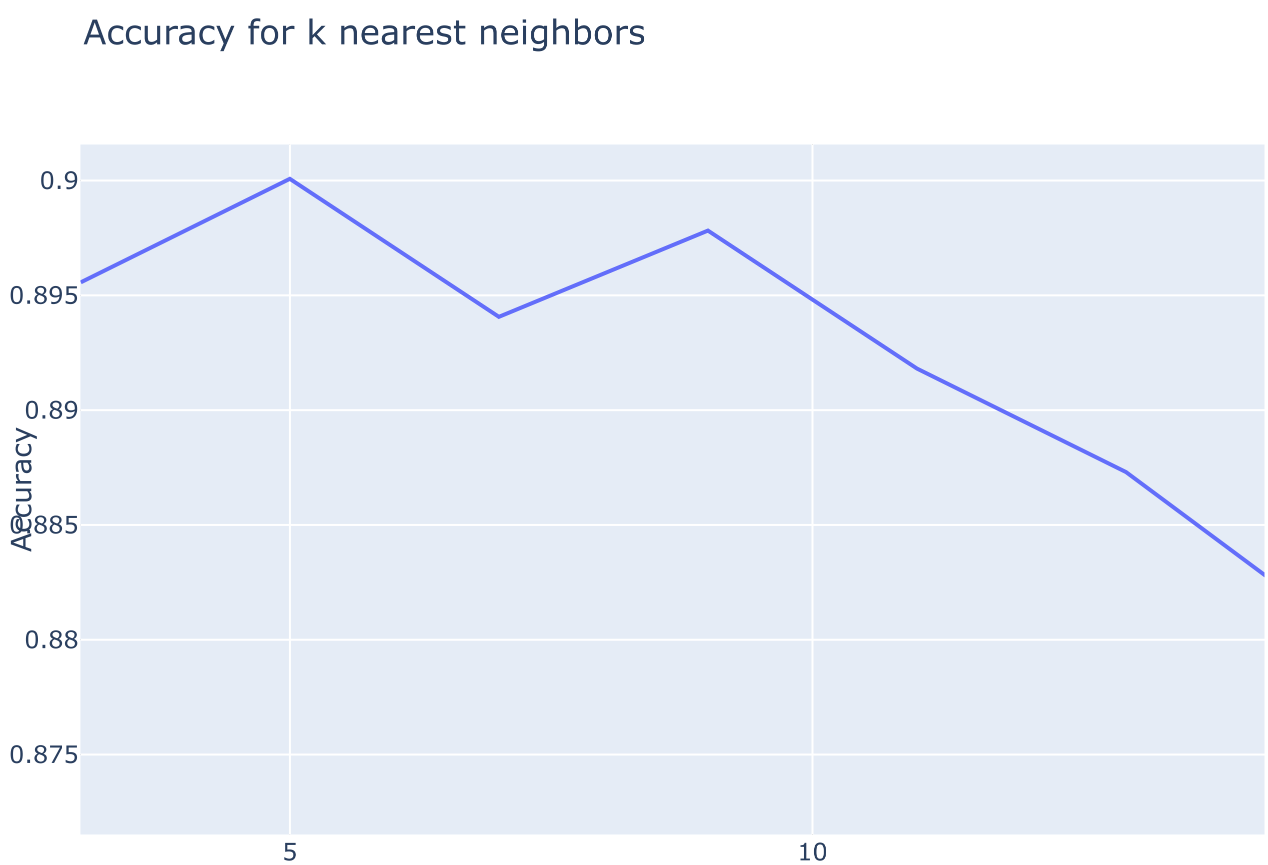
```
# return accuracy for a particular k value when prediction is true by calulating the mean val
def get_acc(prediction, y_true):
    N1 = prediction.shape[0]
    accuracy = np.sum(prediction == y_true) / N1
    return accuracy
```

```
# returns optimal value of k from a given range of values
def kselector(x_val, y_val, x_train, y_train, k_values):
    accuracies = []

    for k in tqdm(k_values):
        prediction = predict(x_val, x_train, y_train, k)

        acc = get_acc(prediction, y_val)
        accuracies.append(acc)

    k_optimal = k_values[accuracies.index(max(accuracies))]

    return k_optimal, accuracies
```

```
k_range = np.arange(3,33,2)
best_k, accuracies = kselector(x_val, y_val, x_train, y_train, k_range)
```

```
100%|██████████| 15/15 [00:08<00:00,  1.84it/s]
```

```
px.line(y=accuracies, x= k_range, title="Accuracy for k nearest neighbors",
        labels=dict(x="K Param", y="Accuracy"))
```




```
print("Optimal  value of k is:",best_k)
```

```
Optimal  value of k is: 5
```

```
y_pred = predict(x_test, x_train, y_train, best_k)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       0.98      0.99      0.99       461
         2.0       0.97      0.96      0.97       224
         3.0       0.92      0.92      0.92       397
         4.0       0.72      0.69      0.70       211
         5.0       0.90      0.87      0.89       237
```

```
         7.0       0.85      0.88      0.87       470

    accuracy                           0.90      2000
   macro avg       0.89      0.89      0.89      2000
weighted avg       0.90      0.90      0.90      2000
```

Comparing our model with sklearn

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
neigh = KNeighborsClassifier(5)
neigh.fit(x_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
y_pred_skl = neigh.predict(x_test)
```

```
print(classification_report(y_test, y_pred_skl))
```

```
              precision    recall  f1-score   support

         1.0       0.98      0.99      0.99       461
         2.0       0.97      0.96      0.96       224
         3.0       0.89      0.92      0.91       397
         4.0       0.70      0.65      0.67       211
         5.0       0.91      0.87      0.89       237
         7.0       0.85      0.86      0.86       470

    accuracy                           0.89      2000
   macro avg       0.88      0.88      0.88      2000
weighted avg       0.89      0.89      0.89      2000
```

```
print("Accuracy for the model from sklearn is",accuracy_score(y_test, y_pred_skl))
print("Accuracy for the model from scratch is",accuracy_score(y_test, y_pred))
```

```
Accuracy for the model from sklearn is 0.894
Accuracy for the model from scratch is 0.9015
```

1m 19s completed at 2:29 PM