Question 1

Using a Manhattan distance metric with k nearest neighbor classification with k=3, k=5 and k=7, classify the following universities as public or private and determine the number of errors in classification for each different value of k. Assume that the ordinal data is NOT to be scaled between the minimum and maximum values when computing an overall distance for all attributes.

In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
df_train1 = pd.read_csv('assign2 data1.JPG.csv')
df_train2 = pd.read_csv('assign2 data3.JPG.csv')

df_test1 = pd.read_csv('assign2 data2.csv')
df_test2 = pd.read_csv('assign2 data4.csv')
```

In [ ]:
```
# splitting the target from both the train and test dataset
df_train1['Private/Public']=np.where(df_train1['Private/Public']=='private',1,0)
df_test1['Private/Public']=np.where(df_test1['Private/Public']=='private',1,0)
df_train1x = df_train1.loc[:, df_train1.columns != 'Private/Public']
df_train1y=df_train1['Private/Public']
df_test1x = df_test1.loc[:, df_test1.columns != 'Private/Public']
df_test1y=df_test1['Private/Public']

# encoding the categorical variables
uni_name = list(set(list(df_train1['University Name'].unique()) + list(df_test1['University Name'].unique())))
uni_state = list(set(list(df_train1['University State'].unique()) + list(df_test1['University State'].unique())))

df_train1x[uni_name + uni_state]=0
df_test1x[uni_name + uni_state]=0

for i in range(1,6):
    df_train1x[f'Academics (1 5)_{i}']=0
    df_train1x.loc[df_train1x['Academics (1 5)']==i, [f'Academics (1 5)_{i}']]=1
    df_train1x[f'Social (1 5)_{i}']=0
    df_train1x.loc[df_train1x['Social (1 5)']==i, [f'Social (1 5)_{i}']]=1
    df_train1x[f'Quality of Life_{i}']=0
    df_train1x.loc[df_train1x['Quality of Life (1 5)']==i, [f'Quality of Life (1 5)_{i}']]=1
    df_test1x[f'Academics (1 5)_{i}']=0
    df_test1x.loc[df_test1x['Academics (1 5)']==i, [f'Academics (1 5)_{i}']]=1
    df_test1x[f'Social (1 5)_{i}']=0
    df_test1x.loc[df_test1x['Social (1 5)']==i, [f'Social (1 5)_{i}']]=1
    df_test1x[f'Quality of Life_{i}']=0
    df_test1x.loc[df_test1x['Quality of Life (1 5)']==i, [f'Quality of Life (1 5)_{i}']]=1

for i in uni_name:
    df_train1x.loc[df_train1x['University Name']==i, [i]]=1
    df_test1x.loc[df_test1x['University Name']==i, [i]]=1

for i in uni_state:
    df_train1x.loc[df_train1x['University State']==i, [i]]=1
    df_test1x.loc[df_test1x['University State']==i, [i]]=1

df_train1x.fillna(0, inplace=True)
df_test1x.fillna(0, inplace=True)
```

```
# knowing the SAT score range is between 200-800, I used a min and max scaling for the sat math and verbal scores.
# scaling the numerical variables between 0 and 1
for i in ['SAT verbal', 'SAT math']:
    df_train1x[f'{i}_scaled'] = (df_train1x[i] - 200)/(800-200)
    df_test1x[f'{i}_scaled'] = (df_test1x[i] - 200)/(800-200)

# dropping the old categories after encoding
df_train1x.drop(['University Name', 'University State', 'Academics (1 5)','Social (1 5)','Quality of Life (1 5)','SAT verbal', 'SAT math'], axis=1, inplace=True)
df_test1x.drop(['University Name', 'University State', 'Academics (1 5)','Social (1 5)','Quality of Life (1 5)','SAT verbal', 'SAT math'], axis=1, inplace=True)
```

In [ ]:
```
# KNN algorithm
k_values = [3,5,7]
dist = np.zeros(len(df_test1x))
errors = np.zeros(len(df_test1x))
for k in k_values:
    for i in range(len(df_test1x)):
        # manhattan distance
        dist = np.sum(abs(df_train1x - df_test1x.iloc[i,:]), axis=1)
        sortIndex = np.argsort(dist)
        bestLabels = df_train1y.loc[sortIndex[0:k]]
        prediction = (sum(bestLabels) > k/2.0)*1.0
        errors[i] = (df_test1y[i] != prediction)*1.0

    print(f"With k = {k}, the total errors = ", np.sum(errors))
```

```
With k = 3, the total errors =  2.0
With k = 5, the total errors =  1.0
With k = 7, the total errors =  3.0
```

In [ ]:
```
# compared to the KNN used in the Sklearn package (this was done as a validation check).
# from sklearn.neighbors import KNeighborsClassifier
# for k in [3,5,7]:
#     knn = KNeighborsClassifier(n_neighbors=k, p=1)
#     knn.fit(df_train1x.to_numpy(), df_train1y.to_numpy())
#     y_pred_test = knn.predict(df_test1x.to_numpy())
#     errors = sum(df_test1y != y_pred_test)
#     print(f"With k = {k}, the total errors = ", errors)
```

```
With k = 3, the total errors =  2
With k = 5, the total errors =  1
With k = 7, the total errors =  3
```

From the above KNN algorithm we can see that optimal number neighbours is 5; as it had the lowest misclassification amount.

Question 2

Assuming that all of the binary data can be treated as symmetric, use k nearest neighbor classification with k = 3, 5 and 7, classify the following data for two different attributes – bladder inflammation (yes, no) and Nephritis (yes, no). Additionally, determine how many errors we would obtain for the two classification results for each value of k.

In [ ]:
```
# binary encoding
df_train2.replace({'yes':1, 'no':0},inplace=True)
df_test2.replace({'yes':1, 'no':0}, inplace=True)

# creating a joint target to capture the two different attributes of interest
df_train2['target']=0
df_test2['target']=0
df_train2.loc[(df_train2['Nephritis']==1)&(df_train2['Bladder Inflamation']==1),['target']]=3
df_train2.loc[(df_train2['Nephritis']==0)&(df_train2['Bladder Inflamation']==1),['target']]=2
```

```
df_train2.loc[(df_train2['Nephritis']==1)&(df_train2['Bladder Inflamation']==0),['target']]=1
df_test2.loc[(df_test2['Nephritis']==1)&(df_test2['Bladder Inflamation']==1),['target']]=3
df_test2.loc[(df_test2['Nephritis']==0)&(df_test2['Bladder Inflamation']==1),['target']]=2
df_test2.loc[(df_test2['Nephritis']==1)&(df_test2['Bladder Inflamation']==0),['target']]=1

# scaling the patient temperature value to be between 0 and 1
max_temp = max(set(list(df_train2['Temperature of Patient (°C)'].unique())+list(df_test2['Temperature of Patient (°C)'].unique())))
min_temp = min(set(list(df_train2['Temperature of Patient (°C)'].unique())+list(df_test2['Temperature of Patient (°C)'].unique())))

df_train2['temp_scaled'] = (df_train2['Temperature of Patient (°C)'] - min_temp)/(max_temp-min_temp)
df_test2['temp_scaled'] = (df_test2['Temperature of Patient (°C)'] - min_temp)/(max_temp-min_temp)

# splitting the predictor from the independent variables
df_train2x= df_train2.drop(['Temperature of Patient (°C)','Patient Number','Nephritis','Bladder Inflamation','target'], axis=1)
df_train2y=df_train2['target']
df_test2x= df_test2.drop(['Temperature of Patient (°C)','Patient Number','Nephritis','Bladder Inflamation','target'], axis=1)
df_test2y=df_test2['target']
```

In [ ]:

```
# KNN algorithm
k_values = [3,5,7]
dist = np.zeros(len(df_test2x))
errors = np.zeros(len(df_test2x))
for k in k_values:
    for i in range(len(df_test2x)):
        # manhattan distance
        dist = np.sum(abs(df_train2x - df_test2x.iloc[i,:]), axis=1)
        sortIndex = np.argsort(dist)
        bestLabels = df_train2y.loc[sortIndex[0:k]]
        prediction = (sum(bestLabels) > k/2.0)*1.0
        errors[i] = (df_test2y[i] != prediction)*1.0

    print(f"With k = {k}, the total errors = ", np.sum(errors))
```

```
With k = 3, the total errors =  1.0
With k = 5, the total errors =  1.0
With k = 7, the total errors =  3.0
```

Based on the above results the optimal value of k is 3 as it provides the least amount of misclassification and the least complicated model. The above results is based on the fact that the two attributes of interest were jointly classified; meaning there is one less feature used to train the model. This approach will be called method 1.

In [ ]:

```
df_train2 = pd.read_csv('assign2 data3.JPG.csv')

df_test2 = pd.read_csv('assign2 data4.csv')

# binary encoding
df_train2.replace({'yes':1, 'no':0},inplace=True)
df_test2.replace({'yes':1, 'no':0}, inplace=True)


# scaling the patient temperature value to be between 0 and 1
max_temp = max(set(list(df_train2['Temperature of Patient (°C)'].unique())+list(df_test2['Temperature of Patient (°C)'].unique())))
min_temp = min(set(list(df_train2['Temperature of Patient (°C)'].unique())+list(df_test2['Temperature of Patient (°C)'].unique())))

df_train2['temp_scaled'] = (df_train2['Temperature of Patient (°C)'] - min_temp)/(max_temp-min_temp)
df_test2['temp_scaled'] = (df_test2['Temperature of Patient (°C)'] - min_temp)/(max_temp-min_temp)

# splitting the predictor from the independent variables
for t in ['Nephritis','Bladder Inflamation']:
    df_train2x= df_train2.drop(['Temperature of Patient (°C)','Patient Number',t], axis=1)
```

```
    df_train2y=df_train2[t]
    df_test2x= df_test2.drop(['Temperature of Patient (°C)','Patient Number',t], axis=1)
    df_test2y=df_test2[t]

    # KNN algorithm
    k_values = [3,5,7]
    dist = np.zeros(len(df_test2x))
    errors = np.zeros(len(df_test2x))
    for k in k_values:
        for i in range(len(df_test2x)):
            # manhattan distance
            dist = np.sum(abs(df_train2x - df_test2x.iloc[i,:]), axis=1)
            sortIndex = np.argsort(dist)
            bestLabels = df_train2y.loc[sortIndex[0:k]]
            prediction = (sum(bestLabels) > k/2.0)*1.0
            errors[i] = (df_test2y[i] != prediction)*1.0

        print(f"With k = {k} and the attribute of interest is {t}, the total errors = ", np.sum(errors))
```

```
With k = 3 and the attribute of interest is Nephritis, the total errors =  0.0
With k = 5 and the attribute of interest is Nephritis, the total errors =  0.0
With k = 7 and the attribute of interest is Nephritis, the total errors =  0.0
With k = 3 and the attribute of interest is Bladder Inflamation, the total errors =  1.0
With k = 5 and the attribute of interest is Bladder Inflamation, the total errors =  1.0
With k = 7 and the attribute of interest is Bladder Inflamation, the total errors =  1.0
```

When we classify the each attribute one at a time we see the classification accuracy increases. The addition of either Nephritis or Bladder Inflamation has helped the model performance. Particularly, when we include Bladder Inflamation as a feature when classifying Nephritis; in this scenario there is no misclassification error at any value of k.