TP8 - KNN - Ex.1

In [1]:
```
# Load the libraries
import numpy as np
import pandas as pd
import seaborn as sns
sns.set(style="ticks", color_codes=True)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

a. Loading the dataset

In [2]:
```
#Loading dataset
abalone_data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-data
```

In [3]:
```
abalone_data.columns = ['Sex','Length','Diameter','Height','Whole weight','Shuck
abalone_data.info()
```

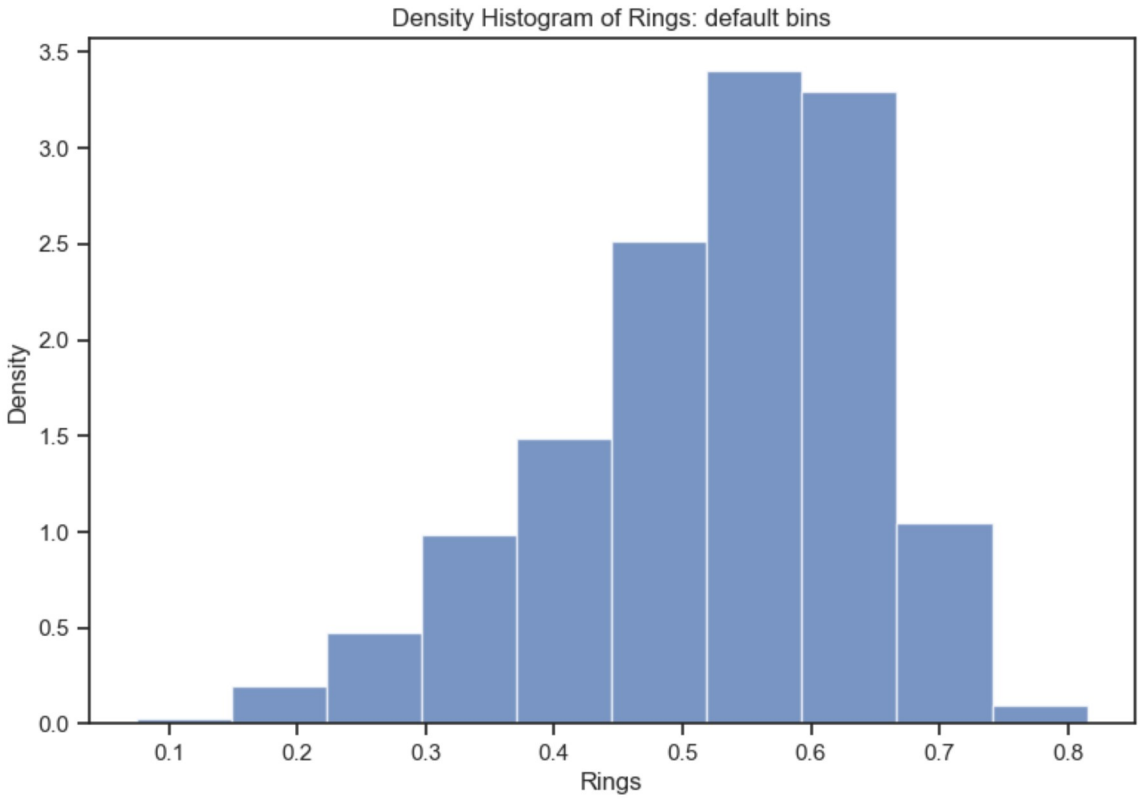
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4176 entries, 0 to 4175
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4176 non-null   object
 1   Length          4176 non-null   float64
 2   Diameter        4176 non-null   float64
 3   Height          4176 non-null   float64
 4   Whole weight    4176 non-null   float64
 5   Shucked weight  4176 non-null   float64
 6   Viscera weight  4176 non-null   float64
 7   Shell weight    4176 non-null   float64
 8   Rings           4176 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

In [4]:
```
abalone_data.describe()
```

Out[4]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	She weigl
count	4176.000000	4176.000000	4176.000000	4176.000000	4176.00000	4176.000000	4176.0000(
mean	0.524009	0.407892	0.139527	0.828818	0.35940	0.180613	0.2388[
std	0.120103	0.099250	0.041826	0.490424	0.22198	0.109620	0.13921
min	0.075000	0.055000	0.000000	0.002000	0.00100	0.000500	0.0015(
25%	0.450000	0.350000	0.115000	0.441500	0.18600	0.093375	0.1300(
50%	0.545000	0.425000	0.140000	0.799750	0.33600	0.171000	0.2340(
75%	0.615000	0.480000	0.165000	1.153250	0.50200	0.253000	0.3290(
max	0.815000	0.650000	1.130000	2.825500	1.48800	0.760000	1.0050(

In [5]:
```
figure = plt.figure(figsize=(20, 6))

axes = figure.add_subplot(1, 2, 2)
axes.hist(abalone_data.Length, density=True, alpha=0.75)
axes.set_title("Density Histogram of Rings: default bins")
axes.set_ylabel("Density")
axes.set_xlabel("Rings")
axes.xaxis.grid(False)
plt.show()
plt.close()
```



b.i.

In [6]:
```
#One-hot encoding (dummy encoding)
abalone_data = pd.get_dummies(abalone_data, dtype=float)
abalone_data.head()
```

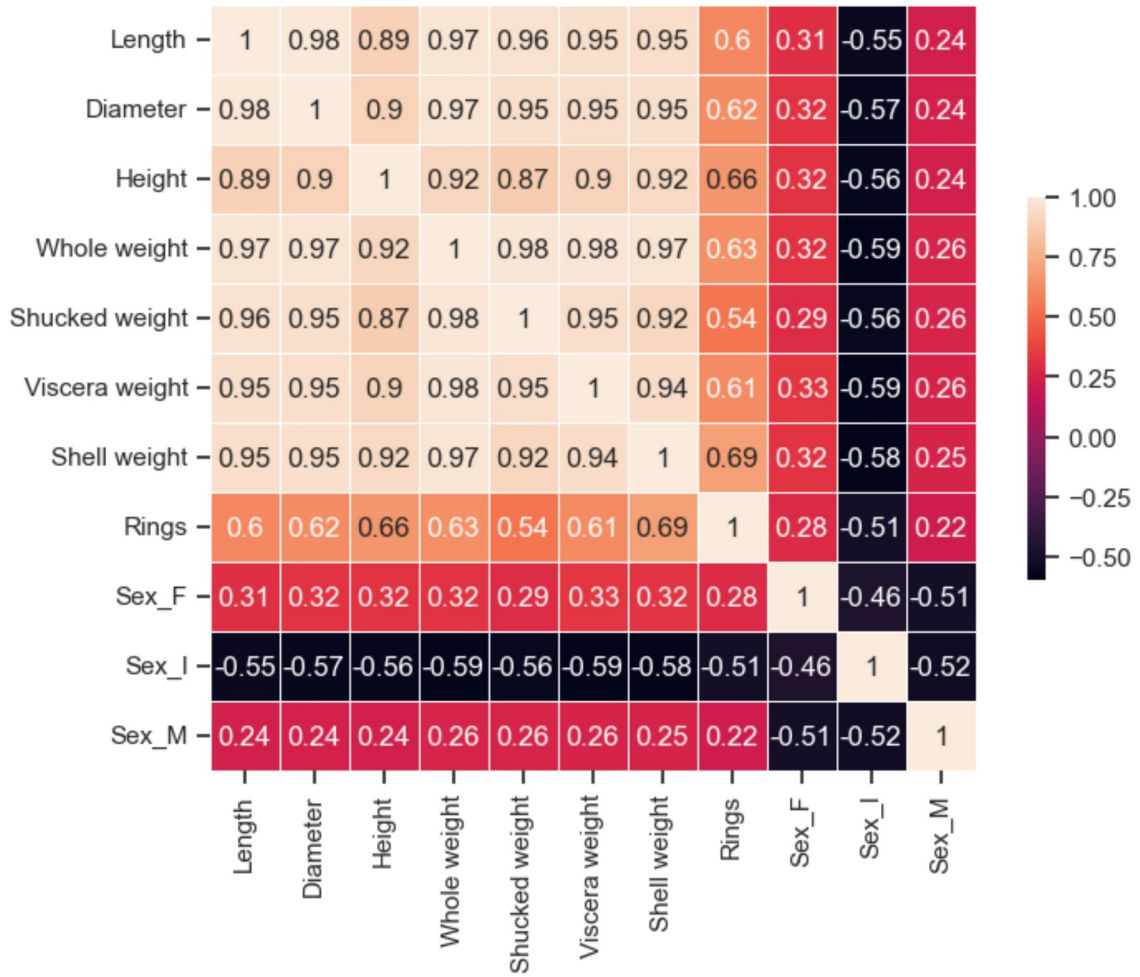
Out[6]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Sex_F	Sex_I	Sex_M
0	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	0.0	0.0	1.0
1	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	1.0	0.0	0.0
2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	0.0	0.0	1.0
3	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7	0.0	1.0	0.0
4	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	8	0.0	1.0	0.0

In [7]:

```
abalone_data.info()

# Compute the correlation matrix
# http://seaborn.pydata.org/examples/many_pairwise_correlations.html

corr = abalone_data.corr('spearman')
f, ax = plt.subplots(figsize=(8, 6))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, vmax=1,square=True,
            linewidths=.5, cbar_kws={"shrink": .5}, ax=ax, annot=True)
plt.show()
plt.close()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4176 entries, 0 to 4175
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Length          4176 non-null   float64
 1   Diameter        4176 non-null   float64
 2   Height          4176 non-null   float64
 3   Whole weight    4176 non-null   float64
 4   Shucked weight  4176 non-null   float64
 5   Viscera weight  4176 non-null   float64
 6   Shell weight    4176 non-null   float64
 7   Rings           4176 non-null   int64  
 8   Sex_F           4176 non-null   float64
 9   Sex_I           4176 non-null   float64
 10  Sex_M           4176 non-null   float64
dtypes: float64(10), int64(1)
memory usage: 359.0 KB
```



b.i.i. Normalization

- Define a min_maxnormalize() function
- Normalize the data using MinMaxScaler to get the range of values in the original data to a scale between 0 to 1, thus simplifying the calculations for the model further.

In [8]:
```
# Normalization of the inputs using MinMax Scaler
scaled_data = MinMaxScaler().fit_transform(abalone_data)
scaled_dataframe = pd.DataFrame(data=scaled_data,columns=abalone_data.columns)
scaled_dataframe.head(5)
```

Out[8]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Sex_F	Sex_I
0	0.371622	0.352941	0.079646	0.079157	0.066241	0.063199	0.068261	0.214286	0.0	0.0
1	0.614865	0.613445	0.119469	0.239065	0.171822	0.185648	0.207773	0.285714	1.0	0.0
2	0.493243	0.521008	0.110619	0.182044	0.144250	0.149440	0.152965	0.321429	0.0	0.0
3	0.344595	0.336134	0.070796	0.071897	0.059516	0.051350	0.053313	0.214286	0.0	1.0
4	0.472973	0.411765	0.084071	0.123783	0.094149	0.101382	0.118087	0.250000	0.0	1.0

c. Divide the data into training and test set (70%, 30%) - Holdout

In [9]:
```
rand_state = 27
testsize = 0.3

X = scaled_dataframe.drop('Rings',axis=1)
y = scaled_dataframe['Rings']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = testsize,
```

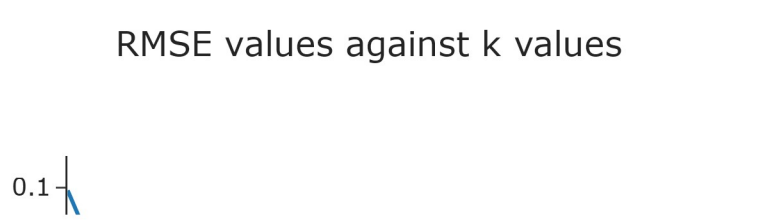
d. Run the model using its default configuration on the test data. Loop over values of K - com MINMAX.

In [10]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

rmse_val = [] # rmse para os k
k_list = []
kmax = 50
kstep = 2

for K in range(1, kmax, kstep):
    model = KNeighborsRegressor(n_neighbors = K)
    model.fit(X_train, y_train)  #fit the model
    y_pred = model.predict(X_test) # make prediction on test set
    error = np.sqrt(mean_squared_error(y_test,y_pred)) # calculate rmse
    rmse_val.append(error) #store rmse values
    k_list.append(K)

    print('RMSE para k= ' , K , ':', error)

# Menor valor de RMSE e respetivo K
min_rmse = min(rmse_val)

# Finding the best K value
best_k = k_list[rmse_val.index(min_rmse)]
print('Melhor K:', best_k)
print('RMSE:', min_rmse)
```

```
RMSE para k=  1 : 0.09989040690769702
RMSE para k=  3 : 0.08481787603713996
RMSE para k=  5 : 0.08073936195532437
RMSE para k=  7 : 0.07828462160990483
RMSE para k=  9 : 0.07745487469765103
RMSE para k=  11 : 0.07730558259484115
RMSE para k=  13 : 0.07709589769297068
RMSE para k=  15 : 0.07730244154210587
RMSE para k=  17 : 0.0770387709125356
RMSE para k=  19 : 0.07707301101747062
RMSE para k=  21 : 0.07693020989409227
RMSE para k=  23 : 0.07703984947859768
RMSE para k=  25 : 0.07735798723151242
RMSE para k=  27 : 0.07730359509576412
RMSE para k=  29 : 0.0773766677713153
RMSE para k=  31 : 0.07776768589025075
RMSE para k=  33 : 0.07796580387578429
RMSE para k=  35 : 0.07804348060650688
RMSE para k=  37 : 0.07818513797474431
RMSE para k=  39 : 0.07823813137884801
RMSE para k=  41 : 0.07836582976557582
RMSE para k=  43 : 0.07844813681680807
RMSE para k=  45 : 0.07860022062469757
RMSE para k=  47 : 0.07872758113783528
RMSE para k=  49 : 0.07876281863546537
Melhor K: 21
RMSE: 0.07693020989409227
```

In [11]:
```
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns

curve = pd.DataFrame(rmse_val) #elbow curve

fig = px.line(curve, x=k_list, y=curve[0], title="RMSE values against k values",
fig.show()
```



e) Classification: KNN - 2 classes Rings: Young, Adult

In [12]:
```
abalone_data.head(5)
```

Out[12]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Sex_F	Sex_I	Sex_M
0	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	0.0	0.0	1.0
1	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	1.0	0.0	0.0
2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	0.0	0.0	1.0
3	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7	0.0	1.0	0.0
4	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	8	0.0	1.0	0.0

In [13]:
```
abalone_data['Rings'] = abalone_data['Rings'].astype(float) + 1.5
```

In [14]:
```
mean_rings_15 = abalone_data['Rings'].mean()
```

In [15]:
```
# Creating a new column based on a condition
abalone_data['Age'] = ['Young' if x < mean_rings_15 else "Adult" for x in abalon
abalone_data.head(5)
```

Out[15]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Sex_F	Sex_I	Sex_M
0	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	8.5	0.0	0.0	1.0
1	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	10.5	1.0	0.0	0.0
2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	11.5	0.0	0.0	1.0
3	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	8.5	0.0	1.0	0.0
4	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	9.5	0.0	1.0	0.0

In [16]:
```
#drop Rings - false predictor - 1x
abalone_data.drop(['Rings'], axis='columns', inplace=True)
abalone_data.head(5)
```

Out[16]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Sex_F	Sex_I	Sex_M	Age
0	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	0.0	0.0	1.0	Young
1	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	1.0	0.0	0.0	Young
2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	0.0	0.0	1.0	Adult
3	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	0.0	1.0	0.0	Young
4	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	0.0	1.0	0.0	Young

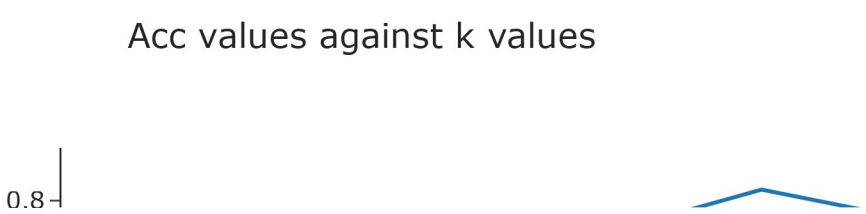
f)

In [17]:
```
rand_state = 27
testsize = 0.3

Xcols = ['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Visc
# Output column
Ycols = ['Age']

X = abalone_data[Xcols]
y = abalone_data[Ycols]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = testsize,
```

In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

accuracy_list = []
k_list = []
kmax = 50
kstep = 2

for K in range(1, kmax, kstep):
    model = KNeighborsClassifier(n_neighbors=K)
    model.fit(X_train, y_train);
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracy_list.append(acc)
    k_list.append(K)

    print('acc para k= ' , K , ':', acc)

# Finding best accuracy and corresponding K value
best_acc = max(accuracy_list)
best_k = k_list[accuracy_list.index(best_acc)]

print('Melhor K:', best_k)
print('Accuracy:', best_acc)
```

```
acc para k=  1 : 0.7310454908220272
acc para k=  3 : 0.7653631284916201
acc para k=  5 : 0.7901037509976058
acc para k=  7 : 0.7956903431763767
acc para k=  9 : 0.8012769353551477
acc para k=  11 : 0.7972865123703112
acc para k=  13 : 0.7980845969672785
acc para k=  15 : 0.7972865123703112
acc para k=  17 : 0.8020750199521149
acc para k=  19 : 0.8012769353551477
acc para k=  21 : 0.7996807661612131
acc para k=  23 : 0.8012769353551477
acc para k=  25 : 0.7988826815642458
acc para k=  27 : 0.7988826815642458
acc para k=  29 : 0.7916999201915403
acc para k=  31 : 0.7916999201915403
acc para k=  33 : 0.7916999201915403
acc para k=  35 : 0.790901835594573
acc para k=  37 : 0.7861133280127693
acc para k=  39 : 0.7805267358339985
acc para k=  41 : 0.7837190742218675
acc para k=  43 : 0.7869114126097366
acc para k=  45 : 0.7877094972067039
acc para k=  47 : 0.7869114126097366
acc para k=  49 : 0.7845171588188348
Melhor K: 17
Accuracy: 0.8020750199521149
```

In [19]:
```
#Plot the above accuracies against the K values for all the three configurations

curve = pd.DataFrame(accuracy_list) #elbow curve

fig = px.line(curve, x=k_list, y=curve[0], title="Acc values against k values",t
fig.show()
```



g) k-fold cross validation

In [20]:
```
# Cross-validation
#------------------------
# KNN model for the best k

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

knn_cv = KNeighborsClassifier(n_neighbors=best_k)
#train model with cv of 5
cv_scores_knn = cross_val_score(knn_cv, X, y, cv=5)

#print each cv score (accuracy) and average them
print(cv_scores_knn)
print('cv_scores_knn mean:{}'.format(np.mean(cv_scores_knn)))

dct = DecisionTreeClassifier(random_state=0)
cv_scores_dct = cross_val_score(dct, X, y, cv=5)

#print each cv score (accuracy) and average them
print(cv_scores_dct)
print('cv_scores_dct mean:{}'.format(np.mean(cv_scores_dct)))
```

```
[0.77033493 0.79640719 0.76646707 0.80598802 0.76886228]
cv_scores_knn mean:0.7816118958255737
[0.72009569 0.71856287 0.70658683 0.72934132 0.70898204]
cv_scores_dct mean:0.716713749534424
```

h) statistical test - ttest

In [21]:
```
#Using the internal function from SciPy Package
from scipy import stats
t_stat, p_val = stats.ttest_ind(cv_scores_knn,cv_scores_dct)
print("t-statistic = " + str(t_stat))
print("p-value = " + str(p_val))
```

```
t-statistic = 7.104623851920019
p-value = 0.00010152838076705696
```