ANIMALS

In [100]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [101]:
```
import os
%matplotlib inline
```

In [102]:
```
animal = pd.read_csv("C:\\Users\\nishi\\Desktop\\Assignments\\KNN\\Zoo.csv")
```

In [103]:
```
animal
```

Out[103]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes
0	aardvark	1	0	0	1	0	0	1	1	1	1
1	antelope	1	0	0	1	0	0	0	1	1	1
2	bass	0	0	1	0	0	1	1	1	1	0
3	bear	1	0	0	1	0	0	1	1	1	1
4	boar	1	0	0	1	0	0	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...
96	wallaby	1	0	0	1	0	0	0	1	1	1
97	wasp	1	0	1	0	1	0	0	0	0	1
98	wolf	1	0	0	1	0	0	1	1	1	1
99	worm	0	0	1	0	0	0	0	0	0	1
100	wren	0	1	1	0	1	0	0	0	1	1

101 rows × 18 columns

In [104]:
```
animal.head(10)
```

Out[104]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes
0	aardvark	1	0	0	1	0	0	1	1	1	1
1	antelope	1	0	0	1	0	0	0	1	1	1
2	bass	0	0	1	0	0	1	1	1	1	0
3	bear	1	0	0	1	0	0	1	1	1	1
4	boar	1	0	0	1	0	0	1	1	1	1
5	buffalo	1	0	0	1	0	0	0	1	1	1
6	calf	1	0	0	1	0	0	0	1	1	1
7	carp	0	0	1	0	0	1	0	1	1	0
8	catfish	0	0	1	0	0	1	1	1	1	0
9	cavy	1	0	0	1	0	0	0	1	1	1

In [105]:
```
animal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   animal name  101 non-null    object
 1   hair         101 non-null    int64 
 2   feathers     101 non-null    int64 
 3   eggs         101 non-null    int64 
 4   milk         101 non-null    int64 
 5   airborne     101 non-null    int64 
 6   aquatic      101 non-null    int64 
 7   predator     101 non-null    int64 
 8   toothed      101 non-null    int64 
 9   backbone     101 non-null    int64 
 10  breathes     101 non-null    int64 
 11  venomous     101 non-null    int64 
 12  fins         101 non-null    int64 
 13  legs         101 non-null    int64 
 14  tail         101 non-null    int64 
 15  domestic     101 non-null    int64 
 16  catsize      101 non-null    int64 
 17  type         101 non-null    int64 
dtypes: int64(17), object(1)
memory usage: 14.3+ KB
```

In [106]:
```
animal=animal.rename({'animal name':'animal_name'},axis=1)
```

In [107]:
```
animal.isnull().sum()
```

Out[107]:
```
animal_name    0
hair           0
feathers       0
eggs           0
milk           0
airborne       0
aquatic        0
predator       0
toothed        0
backbone       0
breathes       0
venomous       0
fins           0
legs           0
tail           0
domestic       0
catsize        0
type           0
dtype: int64
```

In [108]:
```
duplicates = animal.animal_name.value_counts()
duplicates[duplicates > 1]
```

Out[108]:
```
frog    2
Name: animal_name, dtype: int64
```

In [109]:
```
frog = animal.loc[animal['animal_name'] == 'frog']
frog
```

Out[109]:

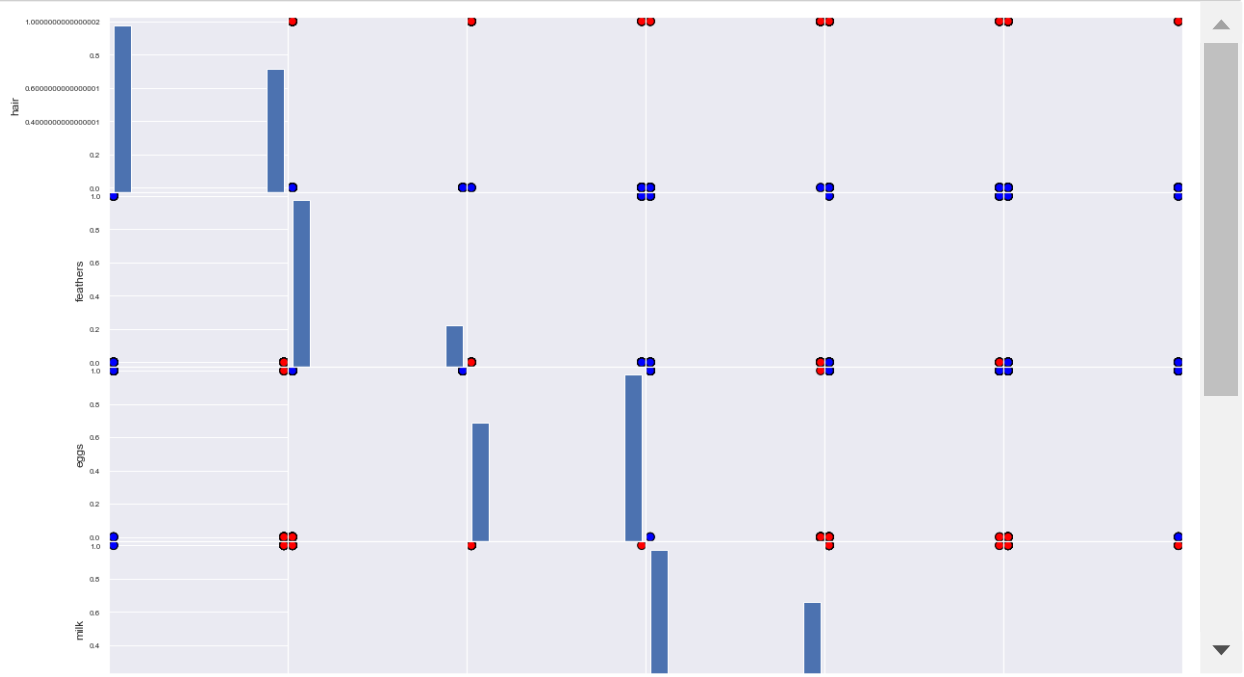
	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	brea
25	frog	0	0	1	0	0	1	1	1	1	
26	frog	0	0	1	0	0	1	1	1	1	

In [110]:
```
animal['animal_name'][(animal.venomous == 1 )& (animal.animal_name == 'frog')] =
```

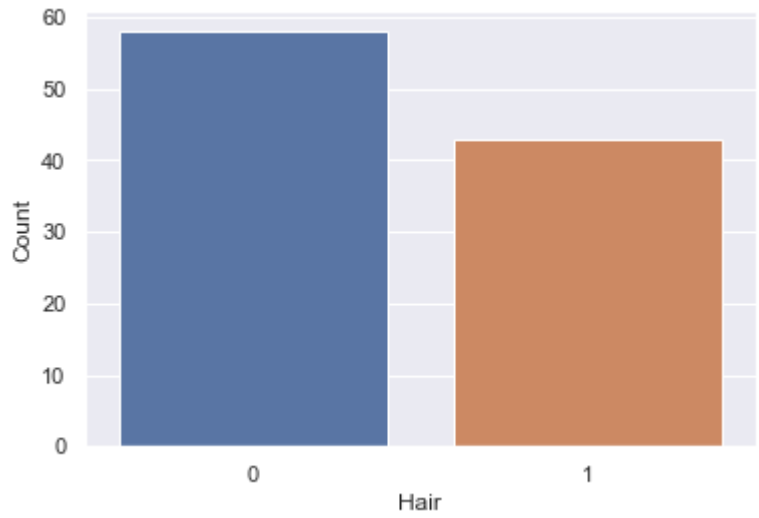
In [111]:
```
color_list = [("red" if i == 1 else "blue" if i == 0 else "yellow" ) for i in ani
unique_color = list(set(color_list))
unique_color
```

Out[111]:
```
['red', 'blue']
```

In [112]:
```
pd.plotting.scatter_matrix(animal.iloc[:,:7],
                                        c=color_list,
                                        figsize= [20,20],
                                        diagonal='hist',
                                        alpha=1,
                                        s = 300,
                                        marker = '.',
                                        edgecolor= "black")
plt.show()
```



In [113]:
```
sns.countplot(x="hair", data=animal)
plt.xlabel("Hair")
plt.ylabel("Count")
plt.show()
animal.loc[:,'hair'].value_counts()
```



Out[113]:
```
0    58
1    43
Name: hair, dtype: int64
```

In [114]:
```
ani_class = pd.read_csv("C:\\Users\\nishi\\Desktop\\Assignments\\KNN\\class.csv")
ani_class
```

Out[114]:

	Class_Number	Number_Of_Animal_Species_In_Class	Class_Type	Animal_Names
0	1	41	Mammal	aardvark, antelope, bear, boar, buffalo, calf,...
1	2	20	Bird	chicken, crow, dove, duck, flamingo, gull, haw...
2	3	5	Reptile	pitviper, seasnake, slowworm, tortoise, tuatara
3	4	13	Fish	bass, carp, catfish, chub, dogfish, haddock, h...
4	5	4	Amphibian	frog, frog, newt, toad
5	6	8	Bug	flea, gnat, honeybee, housefly, ladybird, moth...
6	7	10	Invertebrate	clam, crab, crayfish, lobster, octopus, scorpi...

In [115]:
```
df = pd.merge(animal,ani_class,how='left',left_on='type',right_on='Class_Number')
df.head()
```

Out[115]:

	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	...	fi
0	aardvark	1	0	0	1	0	0	1	1	1	...	
1	antelope	1	0	0	1	0	0	0	1	1	...	
2	bass	0	0	1	0	0	1	1	1	1	...	
3	bear	1	0	0	1	0	0	1	1	1	...	
4	boar	1	0	0	1	0	0	1	1	1	...	

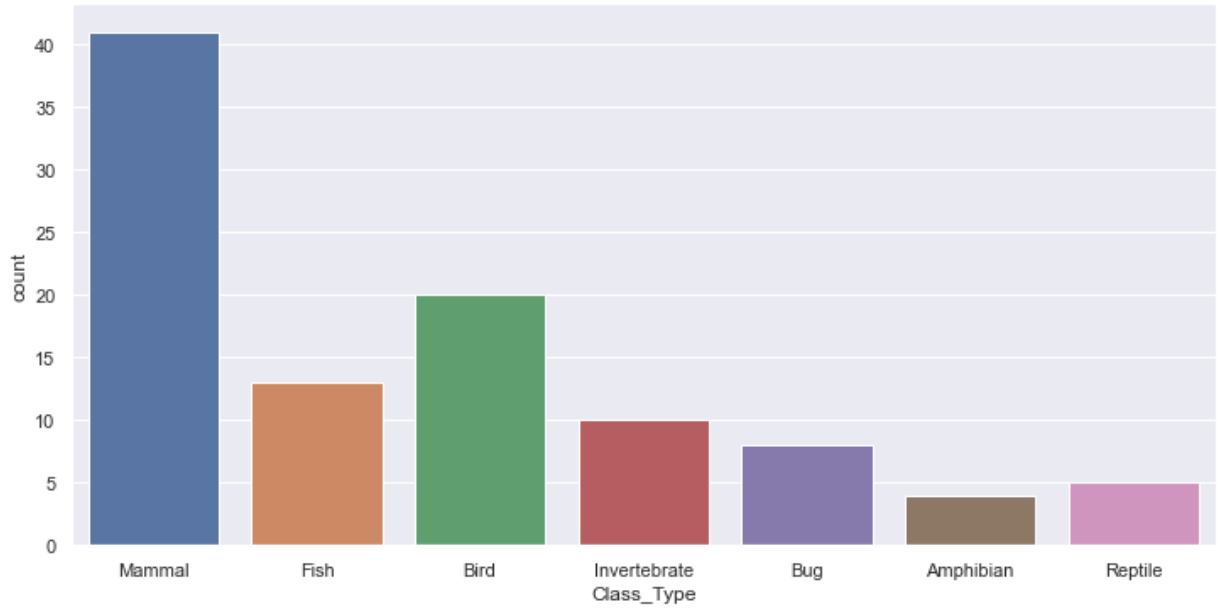
5 rows × 22 columns

In [116]:
```
type_list = [i for i in df.type]
unique_type = list(set(type_list))
unique_type
```

Out[116]:
```
[1, 2, 3, 4, 5, 6, 7]
```

In [117]:
```
sns.factorplot('Class_Type', data=df, kind="count",size = 5,aspect = 2)
```

Out[117]:
```
<seaborn.axisgrid.FacetGrid at 0x1bb6325a670>
```



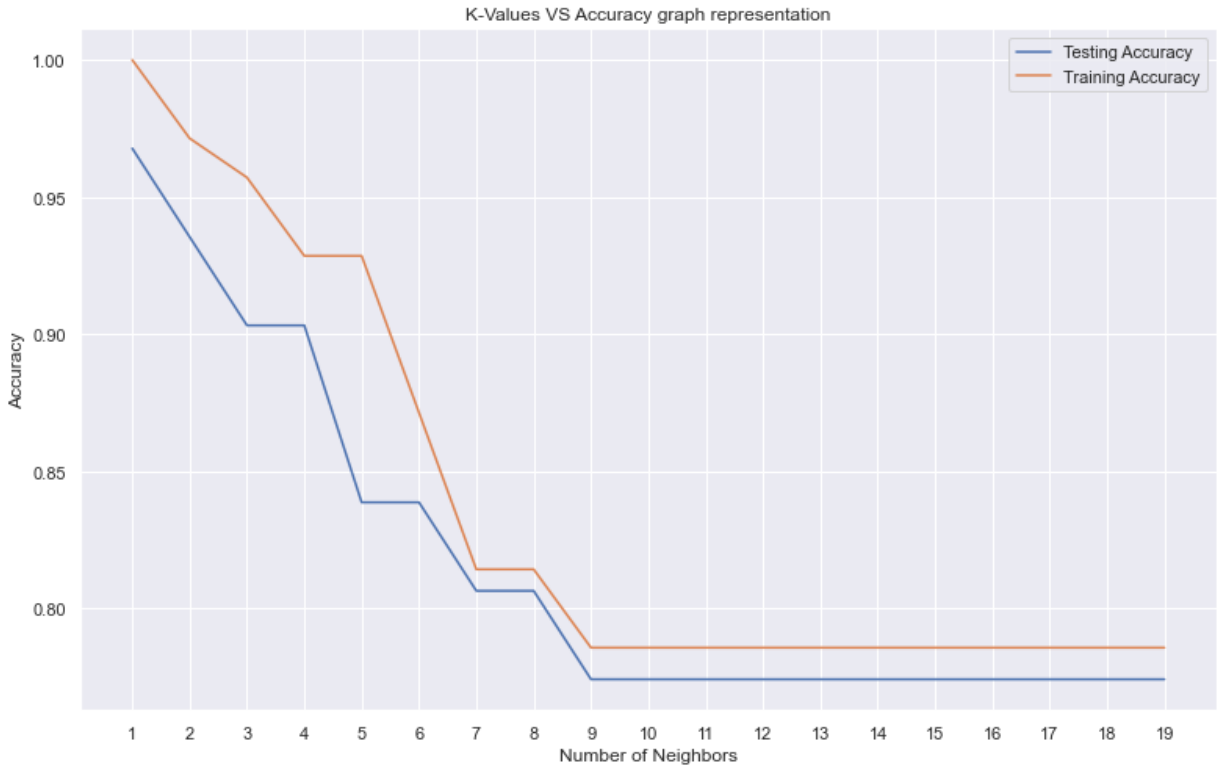
In [118]:
```
from sklearn.model_selection import train_test_split
X = animal.iloc[:,1:17]
y = animal.iloc[:,17]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
```

In [119]:
```
from sklearn.neighbors import KNeighborsClassifier
# Declare the model
clf = KNeighborsClassifier(n_neighbors=3)

# Train the model
clf.fit(X_train, y_train)
y_pred_KNeighborsClassifier = clf.predict(X_test)

scrs = []
from sklearn.metrics import accuracy_score
#Get Accuracy Score
score = accuracy_score(y_pred_KNeighborsClassifier,y_test)
scrs.append(score)
```

In [120]:
```
from sklearn.model_selection import cross_val_score


cv_scores = [] # store cross vadidation score of all the algorithms.

score_knn=cross_val_score(clf, X,y, cv=10)

print("K-Nearest Neighbors Accuracy: %0.2f (+/- %0.2f) with k value equals to 3"
```

```
K-Nearest Neighbors Accuracy: 0.95 (+/- 0.10) with k value equals to 3
```

In [121]:
```
k_values = np.arange(1,20)
train_accuracy = []
test_accuracy = []

for i, k in enumerate(k_values):
    # k from 1 to 20(exclude)
    knn = KNeighborsClassifier(n_neighbors=k)
    # Fit with knn
    knn.fit(X_train,y_train)
    #train accuracy
    train_accuracy.append(knn.score(X_train, y_train))
    # test accuracy
    test_accuracy.append(knn.score(X_test, y_test))


plt.figure(figsize=[13,8])
plt.plot(k_values, test_accuracy, label = 'Testing Accuracy')
plt.plot(k_values, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.title('K-Values VS Accuracy graph representation')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.xticks(k_values)

plt.show()
print("Best accuracy is {} with K = {}".format(np.max(test_accuracy),1+test_accur

cv_scores.append(np.max(test_accuracy))
```




```
Best accuracy is 0.967741935483871 with K = 1
```

In [122]:

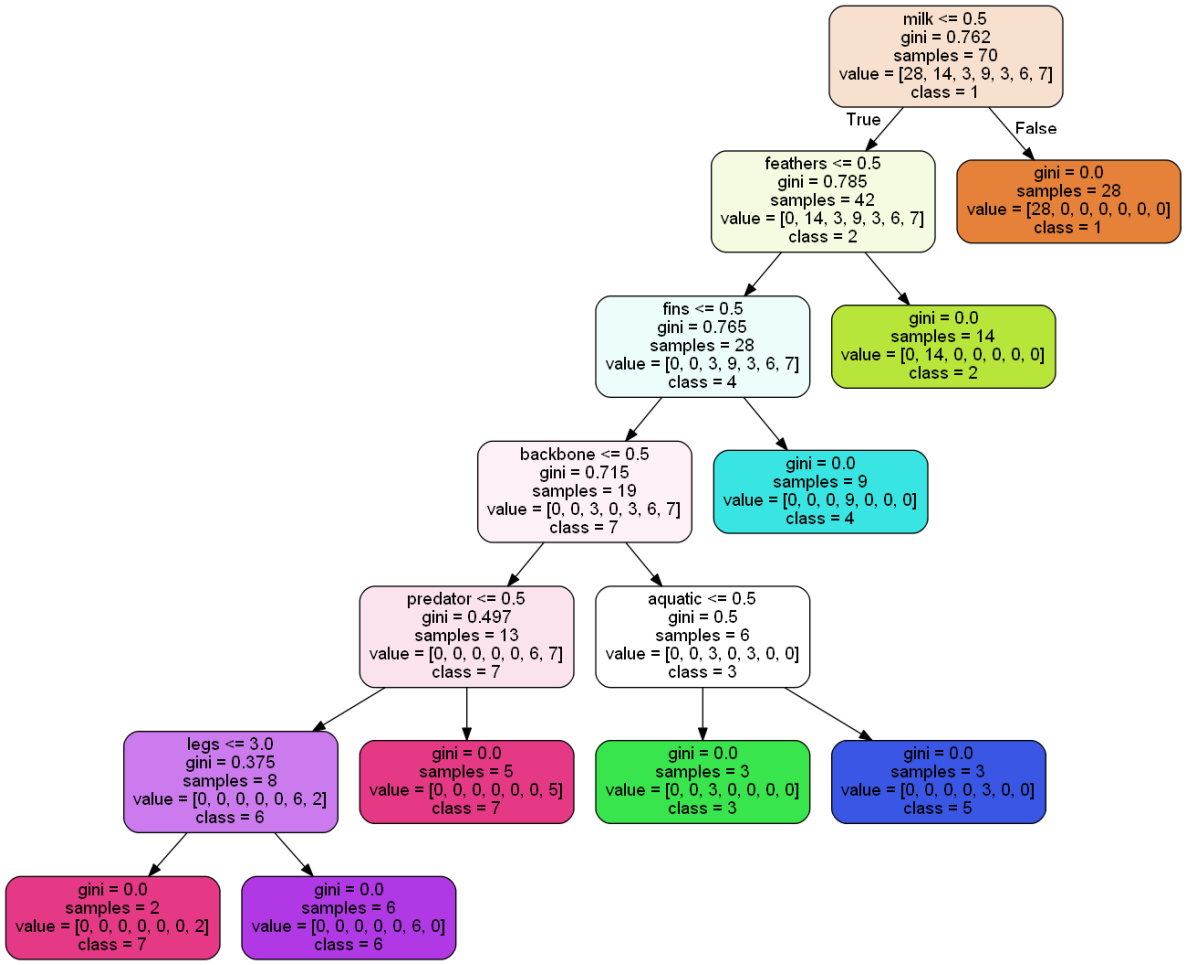
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Declare the model
svm = SVC(kernel='linear', C=0.2, random_state=0)

# Train the model
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

#Get Accuracy Score
score = accuracy_score(y_pred_svm,y_test)
scrs.append(score)
score_svm=cross_val_score(svm, X,y, cv=10)
print("Support Vector Machine Accuracy: %0.2f (+/- %0.2f)" % (score_svm.mean(), s
cv_score = score_svm.mean()
cv_scores.append(cv_score)
```

```
Support Vector Machine Accuracy: 0.96 (+/- 0.10)
```

In [123]:

```
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
from IPython.display import Image
from sklearn.tree import export_graphviz
import graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
# Declare the model
cf = DecisionTreeClassifier(random_state = 0,criterion='gini')
# train the model
cf.fit(X_train, y_train)

y_pred_DecisionTreeClassifier = cf.predict(X_test)
from graphviz import Digraph

scr = accuracy_score(y_pred_DecisionTreeClassifier,y_test)
scrs.append(scr)

global tree
tree = []
tree = cf

dot_data = export_graphviz(tree,
                           filled=True,
                           rounded=True,
                           class_names=["1","2","3","4","5","6","7" ],
                           feature_names=X.columns,
                           out_file=None)

graph = graph_from_dot_data(dot_data)

Image(graph.create_png())
```

Out[123]:



In [124]:
```
from sklearn.model_selection import cross_val_score
score_tree=cross_val_score(cf, X,y, cv=10)
score_tree
```

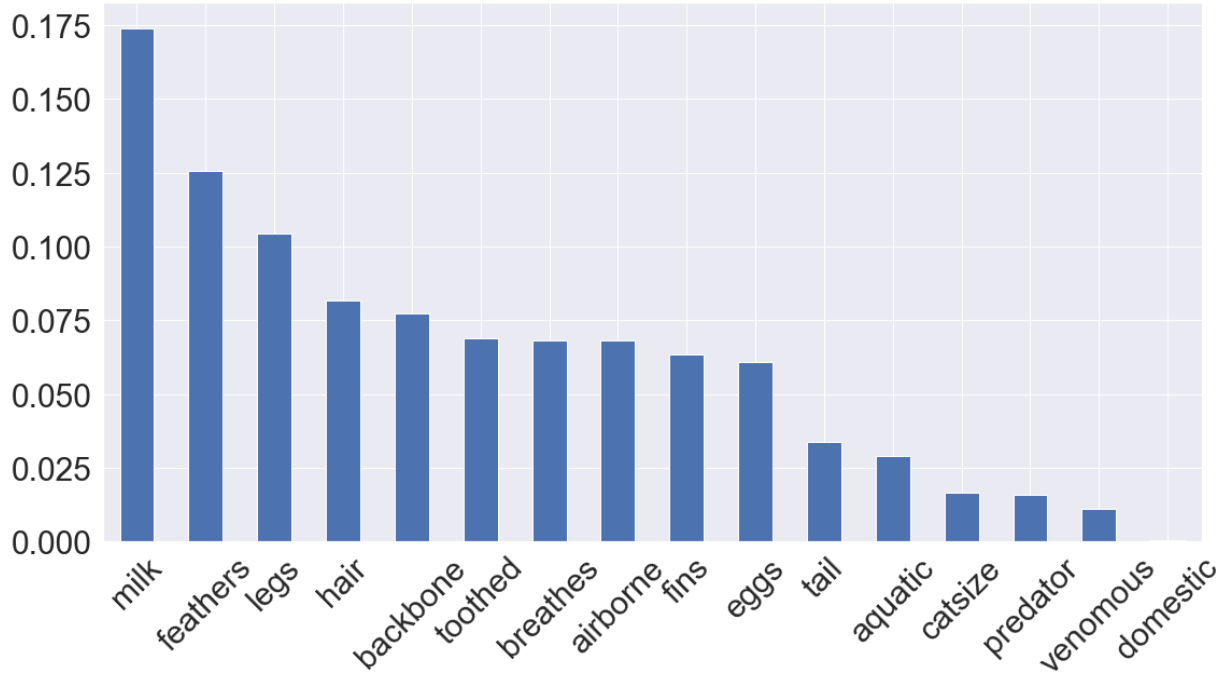
Out[124]:
```
array([1. , 1. , 1. , 1. , 0.9, 0.9, 0.8, 1. , 1. , 1. ])
```

In [125]:
```
print("Decision Tree Accuracy: %0.2f (+/- %0.2f)" % (score_tree.mean(), score_tre
cv_score = score_tree.mean()
cv_scores.append(cv_score)
```

```
Decision Tree Accuracy: 0.96 (+/- 0.13)
```

In [126]:

```
from sklearn.ensemble import RandomForestClassifier


# Declare and train the model
clf = RandomForestClassifier(random_state = 0,n_estimators=25, n_jobs = 2)
clf.fit(X_train, y_train)
y_pred_RandomForestClassifier = clf.predict(X_test)
#Get Accuracy Score
score = accuracy_score(y_pred_RandomForestClassifier,y_test)
scrs.append(score)

global importances
# Get the feature importances
importances = []
importances = clf.feature_importances_


# Convert the importances into one-dimensional 1darray with corresponding df colu
f_importances = pd.Series(importances, X.columns)

# Sort the array in descending order of the importances
f_importances.sort_values(ascending=False, inplace=True)

# Make the bar Plot from f_importances
f_importances.plot(x='Features', y='Importance', kind='bar', figsize=(16,9), rot=

# Show the plot
plt.tight_layout()
plt.show()
```



In [127]:
```
score_forest=cross_val_score(clf, X,y, cv=10)
score_forest
print("Random Forest Accuracy: %0.2f (+/- %0.2f)" % (score_forest.mean(), score_f
cv_score = score_forest.mean()
cv_scores.append(cv_score)
```

```
Random Forest Accuracy: 0.97 (+/- 0.09)
```

In [128]:
```
from sklearn.linear_model import Perceptron

# Declare the model
clf = Perceptron(eta0=0.1, random_state=0)

# Train the model
clf.fit(X_train, y_train)
y_pred_Perceptron = clf.predict(X_test)
#Get Accuracy Score
score = accuracy_score(y_pred_Perceptron,y_test)
scrs.append(score)
```

In [129]:
```
score_perceptron=cross_val_score(clf, X,y, cv=10)
score_perceptron
print("Perceptron Accuracy: %0.2f (+/- %0.2f)" % (score_perceptron.mean(), score_
cv_score_mean = score_perceptron.mean()
cv_score.append(cv_score)
```

```
Perceptron Accuracy: 0.93 (+/- 0.09)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-129-1570423c4be1> in <module>
      3 print("Perceptron Accuracy: %0.2f (+/- %0.2f)" % (score_perceptron.mean
(), score_perceptron.std() * 2))
      4 cv_score_mean = score_perceptron.mean()
----> 5 cv_score.append(cv_score)

AttributeError: 'numpy.float64' object has no attribute 'append'
```

In [ ]:
```
a={'KNeighborsClassifier': 0.95,'Support Vector Machine':0.96,'Decision tree':0.9
print(scrs)
Acc_scores = pd.Series(a,['KNeighborsClassifier','Support Vector Machine','Decisi

current_palette = sns.color_palette("muted", n_colors=5)
cmap = ListedColormap(sns.color_palette(current_palette).as_hex())
#colors = np.random.randint(0,5,5)

# Make the bar Plot from f_importances
Acc_scores.plot(x='Classifiers', y='Accuracy scores',kind = 'bar',figsize=(16,9),
#plt.bar(fscores,clfs)
plt.xlabel('', fontsize=30)
plt.ylabel('Accuracy Score', fontsize=30)
plt.ylim([0.75,1])
# Show the plot
plt.tight_layout()
plt.show()
```

In [99]:
```
print(cv_score)
```

```
0.6127705627705629
```

GLASS

In [40]:
```
#Get the libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
sns.set()
```

In [31]:
```
glass =pd.read_csv("C:\\Users\\nishi\\Desktop\\Assignments\\KNN\\glass.csv")
glass.head()
```

Out[31]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

In [41]:
```
glass1=glass.copy()
```

In [42]:
```
glass1.loc[glass['Type'] == 1, 'Type'] = 'building_windows_float_processed'
glass1.loc[glass['Type'] == 2, 'Type'] = 'building_windows_non_float_processed'
glass1.loc[glass['Type'] == 3, 'Type'] = 'vehicle_windows_float_processed'
glass1.loc[glass['Type'] == 4, 'Type'] = 'vehicle_windows_non_float_processed'
glass1.loc[glass['Type'] == 5, 'Type'] = 'containers'
glass1.loc[glass['Type'] == 6, 'Type'] = 'tableware'
glass1.loc[glass['Type'] == 7, 'Type'] = 'headlamps'
```

In [43]:
```
glass1.head()
```

Out[43]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	building_windows_float_processed
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	building_windows_float_processed
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	building_windows_float_processed
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	building_windows_float_processed
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	building_windows_float_processed

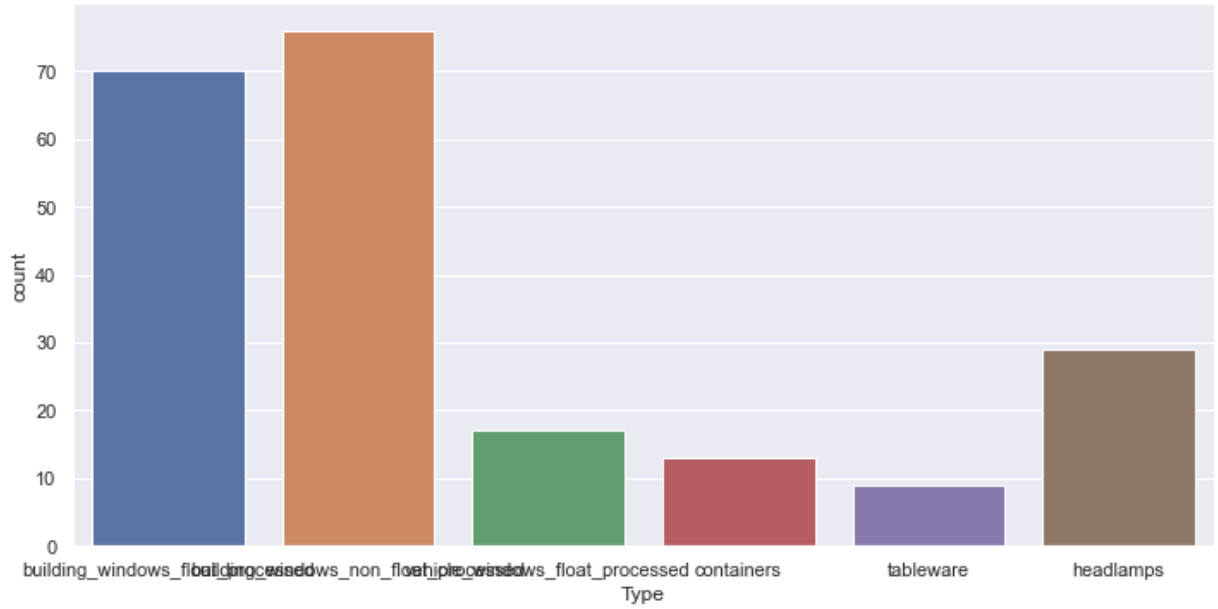
In [44]:
```
glass1.describe()
```

Out[44]:

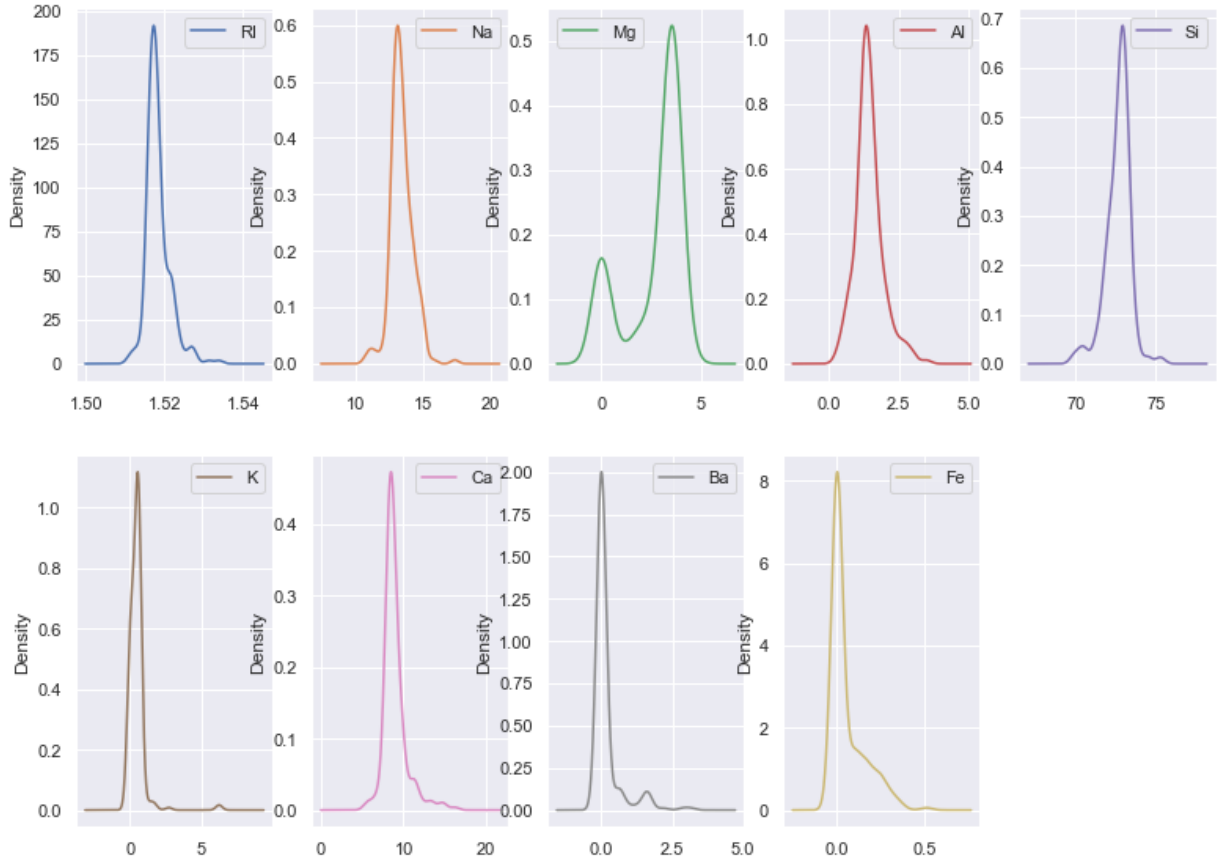
	RI	Na	Mg	Al	Si	K	Ca	
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.00
mean	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.17
std	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.49
min	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.00
25%	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.00
50%	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.00
75%	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.00
max	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.15

In [46]:
```
sns.factorplot('Type', data=glass1, kind="count",size=5,aspect = 2)
```

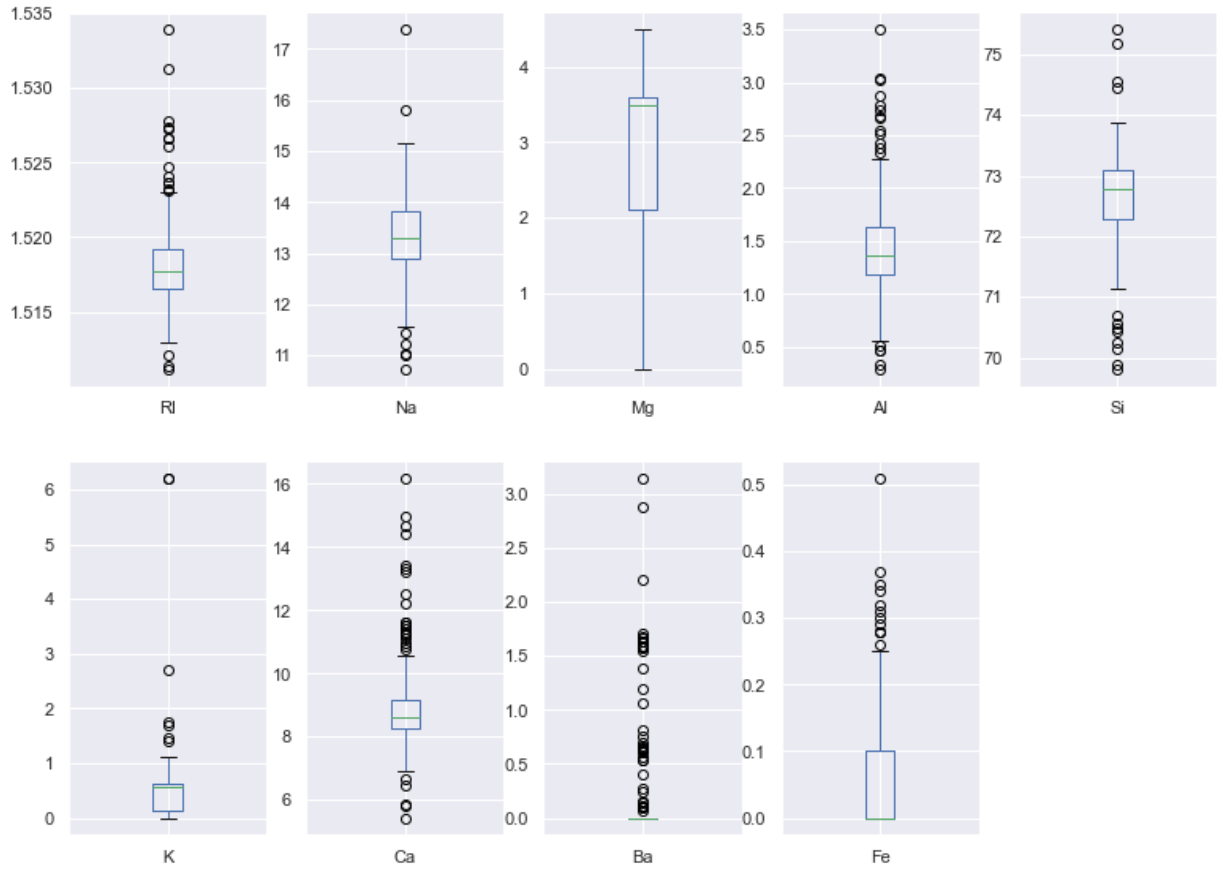
Out[46]:
```
<seaborn.axisgrid.FacetGrid at 0x1bb6312f610>
```



In [48]:
```
glass1.plot(kind='density',subplots=True, layout=(4,5), figsize=(13,20),sharex=Fa
plt.show()
```



In [49]:
```
glass1.plot(kind='box',subplots=True, layout=(4,5), figsize=(13,20),sharex=False,
plt.show()
```



In [50]:
```
# finding the correlation between the data
cor = glass1.corr(method='pearson')
```

In [51]:
```
cor.style.background_gradient(cmap='coolwarm')
```

Out[51]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	F
RI	1.000000	-0.191885	-0.122274	-0.407326	-0.542052	-0.289833	0.810403	-0.000386	0.14301
Na	-0.191885	1.000000	-0.273732	0.156794	-0.069809	-0.266087	-0.275442	0.326603	-0.24134
Mg	-0.122274	-0.273732	1.000000	-0.481799	-0.165927	0.005396	-0.443750	-0.492262	0.08306
Al	-0.407326	0.156794	-0.481799	1.000000	-0.005524	0.325958	-0.259592	0.479404	-0.07440
Si	-0.542052	-0.069809	-0.165927	-0.005524	1.000000	-0.193331	-0.208732	-0.102151	-0.09420
K	-0.289833	-0.266087	0.005396	0.325958	-0.193331	1.000000	-0.317836	-0.042618	-0.00771
Ca	0.810403	-0.275442	-0.443750	-0.259592	-0.208732	-0.317836	1.000000	-0.112841	0.12496
Ba	-0.000386	0.326603	-0.492262	0.479404	-0.102151	-0.042618	-0.112841	1.000000	-0.05869
Fe	0.143010	-0.241346	0.083060	-0.074402	-0.094201	-0.007719	0.124968	-0.058692	1.00000

In [79]:
```
# Finding the optimal number K
X=np.array(glass1.iloc[:,3:5])
y=np.array(glass1['Type'])
```

In [71]:
```
x
```

```
       [ 2.68, 73.39],

       [ 2.54, 73.23],
       [ 2.34, 73.28],
       [ 2.66, 73.1 ],
       [ 2.51, 73.05],
       [ 2.25, 73.5 ],
       [ 1.19, 75.18],
       [ 2.42, 73.72],
       [ 1.99, 73.11],
       [ 2.27, 73.3 ],
       [ 1.8 , 72.99],
       [ 1.87, 73.11],
       [ 1.82, 72.86],
       [ 2.74, 72.85],
       [ 2.88, 72.61],
       [ 1.99, 73.06],
       [ 2.02, 73.42],
       [ 1.94, 73.61],
       [ 2.08, 73.36]])
```

In [69]:
```
y
```

```
       'vehicle_windows_float_processed',
       'vehicle_windows_float_processed',
       'vehicle_windows_float_processed',
       'vehicle_windows_float_processed',
       'vehicle_windows_float_processed',
       'vehicle_windows_float_processed',
       'vehicle_windows_float_processed',
       'vehicle_windows_float_processed', 'containers', 'containers',
       'containers', 'containers', 'containers', 'containers',
       'containers', 'containers', 'containers', 'containers',
       'containers', 'containers', 'containers', 'tableware', 'tableware',
       'tableware', 'tableware', 'tableware', 'tableware', 'tableware',
       'tableware', 'tableware', 'headlamps', 'headlamps', 'headlamps',

       'headlamps', 'headlamps', 'headlamps', 'headlamps', 'headlamps',
       'headlamps', 'headlamps', 'headlamps', 'headlamps', 'headlamps',
       'headlamps', 'headlamps', 'headlamps', 'headlamps', 'headlamps',
       'headlamps', 'headlamps', 'headlamps', 'headlamps', 'headlamps',
       'headlamps', 'headlamps', 'headlamps', 'headlamps', 'headlamps',
       'headlamps'], dtype=object)
```

In [80]:
```
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.2,random_sta
```

In [81]:
```
test_accuracy.describe()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-81-7d3cfe6e148f> in <module>
----> 1 test_accuracy.describe()

AttributeError: 'list' object has no attribute 'describe'
```

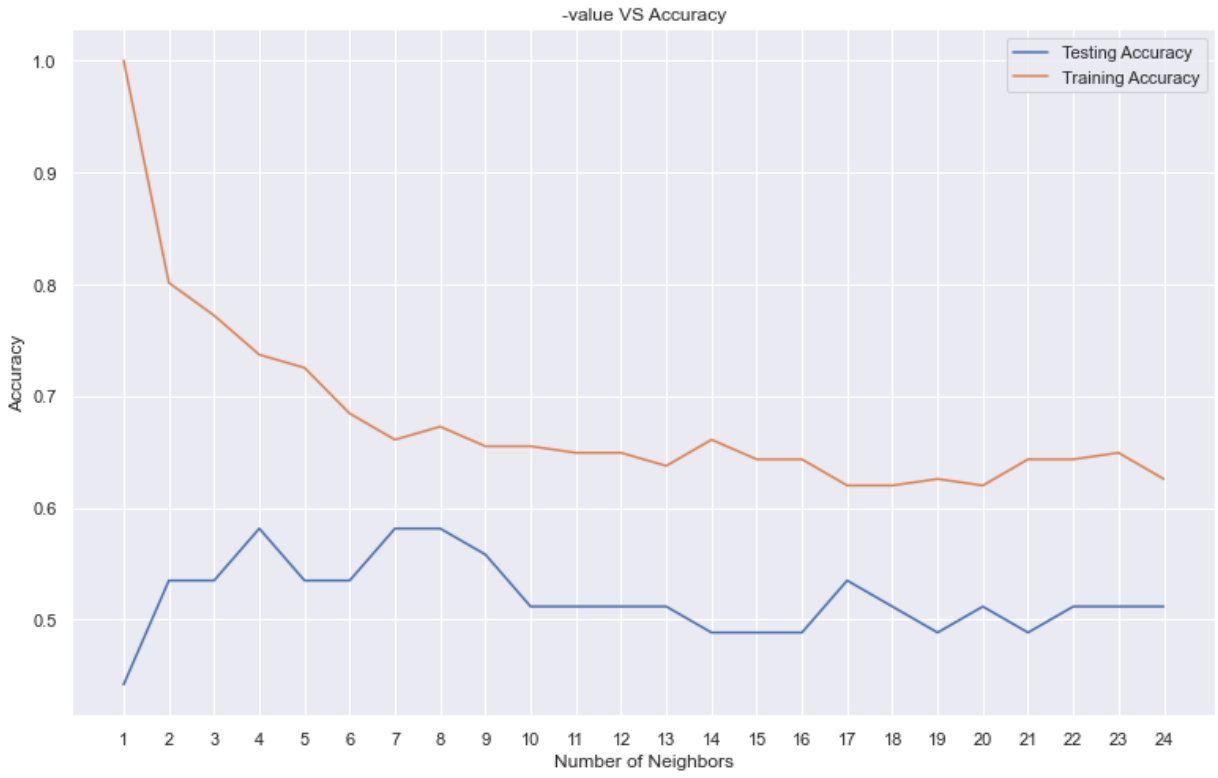
In [75]:
```
train_accuracy
```

Out[75]:
```
[1.0,
 0.8011695906432749,
 0.7719298245614035,
 0.7368421052631579,
 0.7251461988304093,
 0.6842105263157895,
 0.6608187134502924,
 0.672514619883041,
 0.6549707602339181,
 0.6549707602339181,
 0.6491228070175439,
 0.6491228070175439,
 0.6374269005847953,
 0.6608187134502924,
 0.6432748538011696,
 0.6432748538011696,
 0.6198830409356725,
 0.6198830409356725,
 0.6257309941520468,
 0.6198830409356725,
 0.6432748538011696,
 0.6432748538011696,
 0.6491228070175439,
 0.6257309941520468,
 1.0,
 0.8011695906432749,
 0.7719298245614035,
 0.7368421052631579,
 0.7251461988304093,
 0.6842105263157895,
 0.6608187134502924,
 0.672514619883041,
 0.6549707602339181,
 0.6549707602339181,
 0.6491228070175439,
 0.6491228070175439,
 0.6374269005847953,
 0.6608187134502924,
 0.6432748538011696,
 0.6432748538011696,
 0.6198830409356725,
 0.6198830409356725,
 0.6257309941520468,
 0.6198830409356725,
 0.6432748538011696,
 0.6432748538011696,
 0.6491228070175439,
 0.6257309941520468]
```

In [82]:
```
k_values=np.arange(1,25)
train_accuracy=[]
test_accuracy=[]
```

In [83]:
```
for i,k in enumerate(k_values):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test,y_test))
```

In [84]:
```
plt.figure(figsize=[13,8])
plt.plot(k_values, test_accuracy, label = 'Testing Accuracy')
plt.plot(k_values, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.title('-value VS Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.show()
```



In [85]:
```
# Applying the Algorithm
knn = KNeighborsClassifier(n_neighbors=4)
```

In [86]:
```
knn.fit(X_train, y_train)
y_pred_KNeighborsClassifier=knn.predict(X_test)
```

In [91]:
```
scores = []
cv_scores = []
```

In [92]:
```
score = accuracy_score(y_pred_KNeighborsClassifier,y_test)
scores.append(score)
```

In [93]:
```
score_knn=cross_val_score(knn, X,y, cv=10)
```

In [94]:
```
score_knn.mean()
```

Out[94]:
```
0.6127705627705629
```

In [95]:
```
score_knn.std()*2
```

Out[95]:
```
0.23547117559816877
```

In [96]:
```
cv_score=score_knn.mean()
```

In [97]:
```
cv_scores.append(cv_score)
```

In [98]:
```
cv_scores
```

Out[98]:
```
[0.6127705627705629]
```

In [ ]:
```
# The accuracy is 0.61 (+/-0.23)
```