In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
```

In [2]:
```
# Fetching dataset
data = pd.read_csv('tripdata.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Duration	Start date	End date	Start station number	Start station	End station number	End station	Bike number	Member type
0	1012	2010-09-20 11:27:04	2010-09-20 11:43:56	31208	M St & New Jersey Ave SE	31108	4th & M St SW	W00742	Member
1	61	2010-09-20 11:41:22	2010-09-20 11:42:23	31209	1st & N St SE	31209	1st & N St SE	W00032	Member
2	2690	2010-09-20 12:05:37	2010-09-20 12:50:27	31600	5th & K St NW	31100	19th St & Pennsylvania Ave NW	W00993	Member
3	1406	2010-09-20 12:06:05	2010-09-20 12:29:32	31600	5th & K St NW	31602	Park Rd & Holmead Pl NW	W00344	Member
4	1413	2010-09-20 12:10:43	2010-09-20 12:34:17	31100	19th St & Pennsylvania Ave NW	31201	15th & P St NW	W00883	Member

In [4]:
```
data.sample(5)
```

Out[4]:

	Duration	Start date	End date	Start station number	Start station	End station number	End station	Bike number	Member type
27701	1842	2010-10-24 14:31:59	2010-10-24 15:02:42	31200	Massachusetts Ave & Dupont Circle NW	31225	C & O Canal & Wisconsin Ave NW	W00789	Casual
23895	312	2010-10-22 01:21:05	2010-10-22 01:26:18	31214	17th & Corcoran St NW	31101	14th & V St NW	W00721	Member
22144	1030	2010-10-20 17:59:35	2010-10-20 18:16:45	31620	5th & F St NW	31601	19th & East Capitol St SE	W00100	Member
35855	910	2010-10-30 08:22:11	2010-10-30 08:37:21	31616	3rd & H St NE	31610	Eastern Market / 7th & North Carolina Ave SE	W00804	Member
35469	907	2010-10-29 19:33:56	2010-10-29 19:49:03	31112	Harvard St & Adams Mill Rd NW	31203	14th & Rhode Island Ave NW	W00354	Member

In [5]:
```
data.describe()
```

Out[5]:

	Duration	Start station number	End station number
count	115597.000000	115597.000000	115597.000000
mean	1254.649956	31266.213431	31268.042250
std	2914.317998	187.645048	186.194316
min	60.000000	31000.000000	31000.000000
25%	403.000000	31110.000000	31111.000000
50%	665.000000	31213.000000	31214.000000
75%	1120.000000	31301.000000	31238.000000
max	85644.000000	31805.000000	31805.000000

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115597 entries, 0 to 115596
Data columns (total 9 columns):
 #   Column                Non-Null Count   Dtype 
---  ------                --------------   ----- 
 0   Duration              115597 non-null  int64 
 1   Start date            115597 non-null  object
 2   End date              115597 non-null  object
 3   Start station number  115597 non-null  int64 
 4   Start station         115597 non-null  object
```

```
 5   End station number    115597 non-null  int64
 6   End station           115597 non-null  object
 7   Bike number           115597 non-null  object
 8   Member type           115597 non-null  object
dtypes: int64(3), object(6)
memory usage: 7.9+ MB
```

In [7]:
```
data.columns
```

Out[7]:
```
Index(['Duration', 'Start date', 'End date', 'Start station number',
       'Start station', 'End station number', 'End station', 'Bike number',
       'Member type'],
      dtype='object')
```

In [8]:
```
# Exploring Our Target Value .
data['Member type'].value_counts()
```

Out[8]:
```
Member     91586
Casual     24001
Unknown       10
Name: Member type, dtype: int64
```

In [9]:
```
data.shape
```

Out[9]:
```
(115597, 9)
```

In [10]:
```
data=data[data['End date'] >=data['Start date']]
```

In [11]:
```
data.shape
```

Out[11]:
```
(115595, 9)
```

In [12]:
```
# Drop The Irrelevant Columns
data=data.drop(['Start date','End date','Start station','End station'],axis=1)
```

In [13]:
```
data.head()
```

Out[13]:

	Duration	Start station number	End station number	Bike number	Member type
0	1012	31208	31108	W00742	Member
1	61	31209	31209	W00032	Member
2	2690	31600	31100	W00993	Member
3	1406	31600	31602	W00344	Member
4	1413	31100	31201	W00883	Member

In [14]:
```
data.dtypes
```

Out[14]:
```
Duration                int64
Start station number    int64
End station number      int64
Bike number            object
Member type            object
dtype: object
```

In [15]:
```
for l in data:
    if(data[l].dtype=='object'):
        data[l]= data[l].astype('category')
```

```
        data[l]= data[l].cat.codes
```

In [16]:
```
data.head()
```

Out[16]:

	Duration	Start station number	End station number	Bike number	Member type
0	1012	31208	31108	614	1
1	61	31209	31209	41	1
2	2690	31600	31100	836	1
3	1406	31600	31602	282	1
4	1413	31100	31201	734	1

In [17]:
```
# Splitting of dataset into independent and dependent features
X=data.drop(columns=['Member type'])
y=data['Member type']
```

In [18]:
```
X
```

Out[18]:

	Duration	Start station number	End station number	Bike number
0	1012	31208	31108	614
1	61	31209	31209	41
2	2690	31600	31100	836
3	1406	31600	31602	282
4	1413	31100	31201	734
...	...	...	...	...
115592	2179	31110	31623	716
115593	953	31106	31401	764
115594	737	31602	31401	819
115595	514	31111	31202	946
115596	51962	31111	31111	636

115595 rows × 4 columns

In [19]:
```
y
```

Out[19]:
```
0         1
1         1
2         1
3         1
4         1
         ..
115592    0
115593    1
115594    1
115595    1
115596    0
Name: Member type, Length: 115595, dtype: int8
```

In [20]:
```
# Training And Testing of Model
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=42)
```

In [21]:
```
# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(random_state=0)
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
# Check Accuracy
score=metrics.accuracy_score(y_test,y_pred)
dt_score=score*100
print("Accuracy Score Achieved Using Decision Tree Classifier Model iScore Achieveds : {:.1f}%".format(dt_score))
```

```
Accuracy Score Achieved Using Decision Tree Classifier Model iScore Achieveds : 79.3%
```

In [22]:
```
# KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier()
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
# Accuracy
score=metrics.accuracy_score(y_test,y_pred)
knn_score=score*100
print("Accuracy Score Achieved Using KNeighbors Classifier Model is : {:.1f}%".format(knn_score))
```

```
Accuracy Score Achieved Using KNeighbors Classifier Model is : 84.7%
```

In [23]:
```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
classifier=RandomForestClassifier()
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
# Check Accuracy of this model
score=metrics.accuracy_score(y_test,y_pred)
rfc_score=score*100
print("Accuracy Score Achieved Using Random Forest Classifier Model is : {:.1f}%".format(rfc_score))
```

```
Accuracy Score Achieved Using Random Forest Classifier Model is : 86.3%
```

In [24]:
```
# SVM Classifier
from sklearn.svm import SVC
classifier=SVC(kernel='linear',random_state=0)
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
# Check Accuracy of this Model
score=metrics.accuracy_score(y_test,y_pred)
svm_score=score*100
print("Accuracy Score Achieved Using SVM Classifier Model is : {:.1f}%".format(svm_score))
```

```
Accuracy Score Achieved Using SVM Classifier Model is : 85.1%
```

In [25]:
```
all_scores=[dt_score,knn_score,rfc_score,svm_score]
algorithms=['Decision Tree','K-Nearset Neighbors','Random Forest','Support Vector']
```

In [26]:
```
for i in range(len(algorithms)):
    print("The accuracy score achieved using "+algorithms[i]+" is: "+str(all_scores[i])+" %")
```

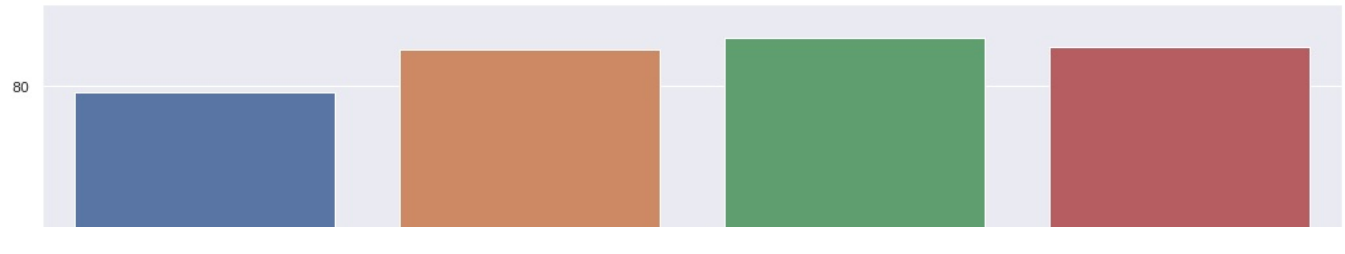
```
The accuracy score achieved using Decision Tree is: 79.26225820962664 %
The accuracy score achieved using K-Nearset Neighbors is: 84.68459116232395 %
The accuracy score achieved using Random Forest is: 86.25904010519395 %
The accuracy score achieved using Support Vector is: 85.10329077130696 %
```

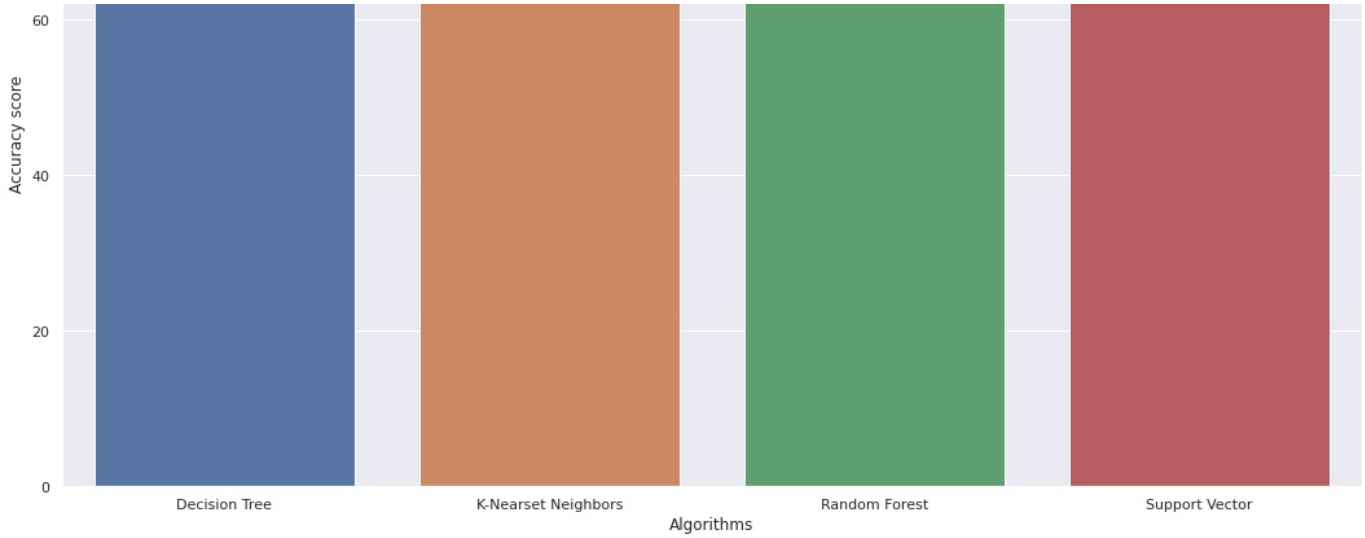
In [27]:
```
sns.set(rc={'figure.figsize':(18,10)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms,all_scores)
```

```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[27]: <AxesSubplot:xlabel='Algorithms', ylabel='Accuracy score'>





In [ ]: