```
from sklearn.datasets import load_iris
data = load_iris()
data.target[[10, 25, 50]]
```

```
array([0, 0, 1])
```

```
list(data.target_names)
```

```
['setosa', 'versicolor', 'virginica']
```

```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_palette('husl')
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

# save load_iris() sklearn dataset to iris
# if you'd like to check dataset type use: type(load_iris())
# if you'd like to view list of attributes use: dir(load_iris())
iris = load_iris()

# np.c_ is the numpy concatenate function
# which is used to concat iris['data'] and iris['target'] arrays
# for pandas column argument: concat iris['feature_names'] list
# and string list (in this case one string); you can make this anything you'd like..
# the original dataset would probably call this ['Species']
data1 = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
```

```
data1.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
```

```
data1.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667	1.000000
std	0.828066	0.433594	1.764420	0.763161	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
data1['target'].value_counts()
```

```
2.0    50
1.0    50
0.0    50
Name: target, dtype: int64
```

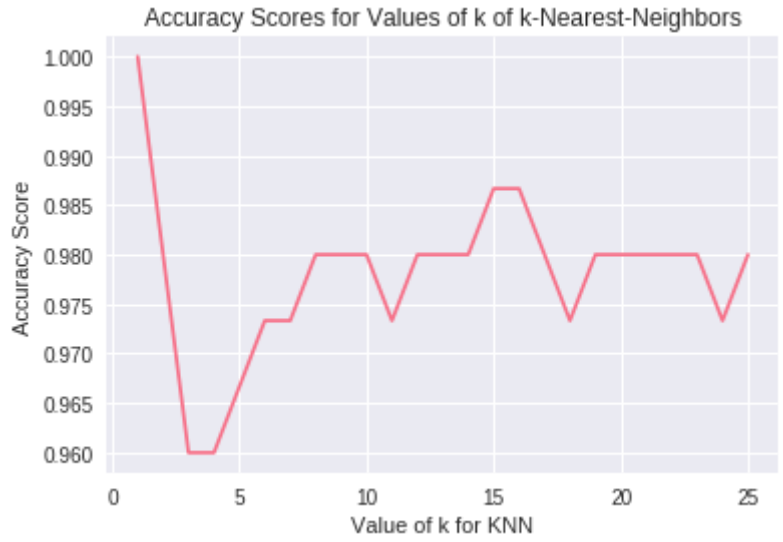
```
data = data1
```

```
X = data.drop(['target'], axis=1)
y = data['target']
# print(X.head())
print(X.shape)
# print(y.head())
print(y.shape)
```

```
(150, 4)
(150,)
```

```
# experimenting with different n values
k_range = list(range(1,26))
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X, y)
    y_pred = knn.predict(X)
    scores.append(metrics.accuracy_score(y, y_pred))

plt.plot(k_range, scores)
plt.xlabel('Value of k for KNN')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors')
plt.show()
```




```
logreg = LogisticRegression()
logreg.fit(X, y)
y_pred = logreg.predict(X)
print(metrics.accuracy_score(y, y_pred))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=5)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
# experimenting with different n values
k_range = list(range(1,26))
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

plt.plot(k_range, scores)
plt.xlabel('Value of k for KNN')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors')
plt.show()
```

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

```
knn = KNeighborsClassifier(n_neighbors=12)
knn.fit(X, y)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
logreg.predict([[6, 3, 4, 2]])
```