# 01.

December 21, 2021

## 1

1.

[ ]:
```
from sklearn.preprocessing import LabelEncoder
```

[ ]:
```
items = ["TV", "  ", "   ", "  ", "  ", "  ", " ", " "]
```

[ ]:
```
#
le = LabelEncoder()
```

[ ]:
```
#
le.fit(items)
```

[ ]:
```
LabelEncoder()
```

[ ]:
```
#
labels = le.transform(items)
labels
```

[ ]:
```
array([0, 1, 4, 5, 3, 3, 2, 2])
```

[ ]:
```
#
le2 = LabelEncoder()
labels = le2.fit_transform(items)
labels
```

[ ]:
```
array([0, 1, 4, 5, 3, 3, 2, 2])
```

[ ]:
```
#
labels = LabelEncoder().fit_transform(items)
labels
```

[ ]:
```
array([0, 1, 4, 5, 3, 3, 2, 2])
```

[ ]:
```
# method chaining
s = "A quick browm fox"
s.lower().lstrip().rstrip().replace('a', "the").replace("fox", "wolf").split()
```

[ ]: ['the', 'quick', 'browm', 'wolf']

[ ]:
```
le2.inverse_transform([4, 5, 2, 3, 0, 1])
```

[ ]: array(['   ', '  ', ' ', '  ', 'TV', '  '], dtype='<U5')

## 2 2. One - hot encoding

[ ]:
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
```

[ ]:
```
oh_labels = ohe.fit_transform(labels.reshape(-1, 1))
oh_labels
```

[ ]:
```
<8x6 sparse matrix of type '<class 'numpy.float64'>'
        with 8 stored elements in Compressed Sparse Row format>
```

[ ]:
```
oh_labels.toarray()
```

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

(Standardization) -

[ ]:
```
from sklearn.datasets import load_iris
iris = load_iris()
```

[ ]:
```
import pandas as pd
df = pd.DataFrame(iris.data, columns = iris.feature_names)
df.describe()
```

[ ]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | \ |
|---|---|---|---|---|
| count | 150.000000 | 150.000000 | 150.000000 | |
| mean | 5.843333 | 3.057333 | 3.758000 | |
| std | 0.828066 | 0.435866 | 1.765298 | |
| min | 4.300000 | 2.000000 | 1.000000 | |
| 25% | 5.100000 | 2.800000 | 1.600000 | |
| 50% | 5.800000 | 3.000000 | 4.350000 | |
| 75% | 6.400000 | 3.300000 | 5.100000 | |
| max | 7.900000 | 4.400000 | 6.900000 | |

```
       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

[ ]:
```
from sklearn.preprocessing import StandardScaler
iris_std = StandardScaler().fit_transform(iris.data)
```

[ ]:
```
df2 = pd.DataFrame(iris_std, columns = iris.feature_names)
df2.describe()
```

[ ]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count       1.500000e+02      1.500000e+02       1.500000e+02
mean       -1.690315e-15     -1.842970e-15      -1.698641e-15
std         1.003350e+00      1.003350e+00       1.003350e+00
min        -1.870024e+00     -2.433947e+00      -1.567576e+00
25%        -9.006812e-01     -5.923730e-01      -1.226552e+00
50%        -5.250608e-02     -1.319795e-01       3.364776e-01
75%         6.745011e-01      5.586108e-01       7.627583e-01
max         2.492019e+00      3.090775e+00       1.785832e+00

       petal width (cm)
count      1.500000e+02
mean      -1.409243e-15
std        1.003350e+00
min       -1.447076e+00
25%       -1.183812e+00
50%        1.325097e-01
75%        7.906707e-01
max        1.712096e+00
```

- Logistic REgression

[ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =  train_test_split(
    iris.data, iris.target, stratify = iris.target,test_size = 0.2,␣
 ↪random_state = 2021
)
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
lrc = LogisticRegression()
lrc.fit(X_train, y_train)
```

```
/Volumes/Coding/opt/miniconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:
```
LogisticRegression()
```

[ ]:
```
#
X_train, X_test, y_train, y_test =  train_test_split(
    iris_std, iris.target, stratify = iris.target,test_size = 0.2, random_state␣
↪= 2021
)
lrc = LogisticRegression()
lrc.fit(X_train, y_train)
```

[ ]:
```
LogisticRegression()
```

[ ]:
```
lrc.score(X_test,y_test)
```

[ ]:
```
0.9
```

### 2.0.1 4. (Normalization) - 0 ~ 1

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
iris_mn = MinMaxScaler().fit_transform(iris.data)
```

[ ]:
```
df3 = pd.DataFrame(iris_mn, columns = iris.feature_names)
df3.describe()
```

[ ]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            0.428704          0.440556           0.467458
std             0.230018          0.181611           0.299203
min             0.000000          0.000000           0.000000
25%             0.222222          0.333333           0.101695
50%             0.416667          0.416667           0.567797
75%             0.583333          0.541667           0.694915
max             1.000000          1.000000           1.000000

       petal width (cm)
count        150.000000
```

```
mean            0.458056
std             0.317599
min             0.000000
25%             0.083333
50%             0.500000
75%             0.708333
max             1.000000
```

[ ]:
```
X_train, X_test, y_train, y_test =  train_test_split(
    iris_std, iris.target, stratify = iris.target,test_size = 0.2, random_state␣
↪= 2021
)
lrc = LogisticRegression()
lrc.fit(X_train, y_train)
lrc.score(X_test,y_test)
```

[ ]:
```
0.9
```