Feature Scaling

Rules to work with SCi-kit learn

1. Your data must be strictly a Numpy array
2. Sklearn doesnot support handling missing values for Non-numeric columns (so use pandas)
3. Ensure your feature set is completely NUMERIC

1		`data = pd.read_csv('datasample.csv')`
		
		`data.info()`
		
		`<class 'pandas.core.frame.DataFrame'>`
		`RangeIndex: 10 entries, 0 to 9`
		`Data columns (total 4 columns):`
		` #  Column  Non-Null Count  Dtype`
		`---  ------  --------------  -----`
		` 0  Country  9 non-null  object`
		` 1  Age  9 non-null  float64`
		` 2  Salary  9 non-null  float64`
		` 3  Purchased  10 non-null  object`
		`dtypes: float64(2), object(2)`
		`memory usage: 448.0+ bytes`

2		Data **Country** **Age** **Salary** **Purchased** 0 France 44.0 72000.0 No 1 Spain 27.0 48000.0 Yes 2 Germany 30.0 54000.0 No 3 Spain 38.0 61000.0 No 4 Germany 40.0 NaN Yes 5 France 35.0 58000.0 Yes 6 Spain NaN 52000.0 No 7 France 48.0 79000.0 Yes 8 NaN 50.0 83000.0 No 9 France 37.0 67000.0 Yes
3		`data.describe()` **Age** **Salary** Count 9.000000 9.000000 mean 38.777778 63777.777778 std 7.693793 12265.579662 min 27.000000 48000.000000

		25% 35.000000 54000.000000 50% 38.000000 61000.000000 75% 44.000000 72000.000000 max 50.000000 83000.000000

4	Replace NaN with mode of Country	`data.Country.fillna(data.Country.mode()[0], inplace=True)`
		`data`
		
		Country Age Salary Purchased
		0 France 44.0 72000.0 No
		1 Spain 27.0 48000.0 Yes
		2 Germany 30.0 54000.0 No
		3 Spain 38.0 61000.0 No
		4 Germany 40.0 NaN Yes
		5 France 35.0 58000.0 Yes
		6 Spain NaN 52000.0 No
		7 France 48.0 79000.0 Yes
		8 France 50.0 83000.0 No
		9 France 37.0 67000.0 Yes

5	Get the stats summary	`data.describe()` **Age Salary** count 9.000000 9.000000 mean 38.777778 63777.777778 std 7.693793 12265.579662 min 27.000000 48000.000000 25% 35.000000 54000.000000 50% 38.000000 61000.000000 75% 44.000000 72000.000000 max 50.000000 83000.000000

6	Replace NaN with mode of Country	`data.Country.fillna(data.Country.mode()[0], inplace=True)`
		`data`
		
		Country Age Salary Purchased
		0 France 44.0 72000.0 No
		1 Spain 27.0 48000.0 Yes
		2 Germany 30.0 54000.0 No
		3 Spain 38.0 61000.0 No
		4 Germany 40.0 NaN Yes
		5 France 35.0 58000.0 Yes
		6 Spain NaN 52000.0 No
		7 France 48.0 79000.0 Yes
		8 France 50.0 83000.0 No

		9 France 37.0 67000.0 Yes
7		features = data.iloc[:,[0,1,2]].values label = data.iloc[:,[3]].values
8		type(features) features array([['France', 44.0, 72000.0], ['Spain', 27.0, 48000.0], ['Germany', 30.0, 54000.0], ['Spain', 38.0, 61000.0], ['Germany', 40.0, nan], ['France', 35.0, 58000.0], ['Spain', nan, 52000.0], ['France', 48.0, 79000.0], ['France', 50.0, 83000.0], ['France', 37.0, 67000.0]], dtype=object)
9		type(label) label array([['No'], ['Yes'], ['No'], ['No'], ['Yes'], ['Yes'], ['No'], ['Yes'], ['No'], ['Yes']], dtype=object)
10	Dealing With Missing Values Step1: Import the relevant package Step2: Create object and instantiate the object Step3: Fit the object with the data (Calculation) Step4: Transform your dataset with fitted values calc in Step3	from sklearn.impute import SimpleImputer si = SimpleImputer(strategy='mean', missing_values=np.nan) si.fit(features[:,[1,2]]) features[:,[1,2]] = si.transform(features[:,[1,2]]) features array([['France', 44.0, 72000.0], ['Spain', 27.0, 48000.0], ['Germany', 30.0, 54000.0], ['Spain', 38.0, 61000.0], ['Germany', 40.0, 63777.77777777778], ['France', 35.0, 58000.0], ['Spain', 38.77777777777778, 52000.0], ['France', 48.0, 79000.0], ['France', 50.0, 83000.0], ['France', 37.0, 67000.0]], dtype=object)
11	Handling categorical columns	features[:,0].reshape(-1,1) array([['France'], ['Spain'], ['Germany'], ['Spain'], ['Germany'], ['France'], ['Spain'],

		['France'], ['France'], ['France']], dtype=object)
12	One Hot Encoding ---> Creates Dummy Variables	from sklearn.preprocessing import OneHotEncoder ohe = OneHotEncoder(sparse=False) fCountry = ohe.fit_transform(features[:,0].reshape(-1,1)) fCountry array([[1., 0., 0.], [0., 0., 1.], [0., 1., 0.], [0., 0., 1.], [0., 1., 0.], [1., 0., 0.], [0., 0., 1.], [1., 0., 0.], [1., 0., 0.], [1., 0., 0.]])
13		features[:,[1,2]] array([[44.0, 72000.0], [27.0, 48000.0], [30.0, 54000.0], [38.0, 61000.0], [40.0, 63777.77777777778], [35.0, 58000.0], [38.77777777777778, 52000.0], [48.0, 79000.0], [50.0, 83000.0], [37.0, 67000.0]], dtype=object)
14		finalFeatures = np.concatenate((fCountry,features[:,[1,2]]) , axis = 1) finalFeatures array([[1.0, 0.0, 0.0, 44.0, 72000.0], [0.0, 0.0, 1.0, 27.0, 48000.0], [0.0, 1.0, 0.0, 30.0, 54000.0], [0.0, 0.0, 1.0, 38.0, 61000.0], [0.0, 1.0, 0.0, 40.0, 63777.77777777778], [1.0, 0.0, 0.0, 35.0, 58000.0], [0.0, 0.0, 1.0, 38.77777777777778, 52000.0], [1.0, 0.0, 0.0, 48.0, 79000.0], [1.0, 0.0, 0.0, 50.0, 83000.0], [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
15	**Feature Scaling** StandardScaler: Bring your data such that the mean is 0 and stddev is 1	from sklearn.preprocessing import StandardScaler **StandardScaler** sc = StandardScaler() sc.fit(finalFeatures) featureStandardScaler = sc.transform(finalFeatures)

MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler

mm = MinMaxScaler(feature_range=(0,1))
mm.fit(finalFeatures)
featureMinMaxScaler = mm.transform(finalFeatures)
```

```
featureMinMaxScaler
```

```
array([[1.        , 0.        , 0.        , 0.73913043,
0.68571429],
       [0.        , 0.        , 1.        , 0.        , 0.
],
       [0.        , 1.        , 0.        , 0.13043478,
0.17142857],
       [0.        , 0.        , 1.        , 0.47826087,
0.37142857],
       [0.        , 1.        , 0.        , 0.56521739,
0.45079365],
       [1.        , 0.        , 0.        , 0.34782609,
0.28571429],
       [0.        , 0.        , 1.        , 0.51207729,
0.11428571],
       [1.        , 0.        , 0.        , 0.91304348,
0.88571429],
       [1.        , 0.        , 0.        , 1.        , 1.
],
       [1.        , 0.        , 0.        , 0.43478261,
0.54285714]])
```