```
Programmed By : Rithik Tripathi
```

[Connect with me on Linkedin (https://www.linkedin.com/in/rithik-tripathi-data-scientist/)](https://www.linkedin.com/in/rithik-tripathi-data-scientist/)

In this notebook, We will see 3 basic methods for Dimensionality Reduction.

Dimensionality Reduction - Part 1 ¶

In [1]:

```python
import pandas as pd
```

Missing Value Ratio

```
if a feature is having more than 70% of records missing, there is no point in still using it, even with imputation as the variance of the feature will be low and not much of information will be available with it.
```

In [2]:

```python
df = pd.read_csv("Dataset/Dimensionality_Reduction/missing_value_ratio.csv")
df.head()
```

Out[2]:

	ID	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count
0	AB101	1.0	0.0	0.0	1.0	9.84	14.395	81.0	NaN	16
1	AB102	1.0	NaN	0.0	NaN	9.02	13.635	80.0	NaN	40
2	AB103	1.0	0.0	NaN	1.0	9.02	13.635	80.0	NaN	32
3	AB104	NaN	0.0	NaN	1.0	9.84	14.395	75.0	NaN	13
4	AB105	1.0	NaN	0.0	NaN	9.84	14.395	NaN	16.9979	1

In [7]:

```python
# percentage of missing values in each feature
null_stats = round((df.isnull().sum() / df.shape[0] )*100,2)
null_stats
```

Out[7]:

```
ID             0.00
season         0.07
holiday       48.50
workingday     0.07
weather        0.03
temp           0.00
atemp          0.00
humidity       0.04
windspeed     41.02
count          0.00
dtype: float64
```

> 40 % missing values are still not a huge number, we could handle them, but for demonstration purpose, we will proceed ahead with this dataset

In [16]:

```python
# list of variables with nulls below threshold
variables_below_threshold_list = []
null_threshold = 40
for var in range(df.shape[1]):
    if null_stats[var] <= null_threshold:
        variables_below_threshold_list.append(df.columns[var])
```

In [17]:

```python
variables_below_threshold_list
```

Out[17]:

```
['ID', 'season', 'workingday', 'weather', 'temp', 'atemp', 'humidity', 'count']
```

In [19]:

```
# Generating new dataframe with values below threshold
new_df = df[variables_below_threshold_list]
new_df.head()
```

Out[19]:

	ID	season	workingday	weather	temp	atemp	humidity	count
0	AB101	1.0	0.0	1.0	9.84	14.395	81.0	16
1	AB102	1.0	0.0	NaN	9.02	13.635	80.0	40
2	AB103	1.0	NaN	1.0	9.02	13.635	80.0	32
3	AB104	NaN	NaN	1.0	9.84	14.395	75.0	13
4	AB105	1.0	0.0	NaN	9.84	14.395	NaN	1

In [20]:

```
# percentage of missing values in each feature
new_null_stats = round((new_df.isnull().sum() / new_df.shape[0] )*100,2)
new_null_stats
```

Out[20]:

```
ID            0.00
season        0.07
workingday    0.07
weather       0.03
temp          0.00
atemp         0.00
humidity      0.04
count         0.00
dtype: float64
```

In [21]:

```
df.shape, new_df.shape
```

Out[21]:

```
((12980, 10), (12980, 8))
```

In [ ]:

```

```

Low Variance Filter

```
as known, low variance implies not much of data points are available in it and hence not much of information is available.
```

In [34]:

```
import pandas as pd
from sklearn.preprocessing import normalize
```

In [35]:

```
df = pd.read_csv("Dataset/Dimensionality_Reduction/low_variance_filter.csv")
df.head()
```

Out[35]:

	ID	temp	atemp	humidity	windspeed	count
0	AB101	9.84	14.395	81	0.0	16
1	AB102	9.02	13.635	80	0.0	40
2	AB103	9.02	13.635	80	0.0	32
3	AB104	9.84	14.395	75	0.0	13
4	AB105	9.84	14.395	75	0.0	1

In [36]:

```
# percentage of missing values in each feature
null_stats = round((df.isnull().sum() / df.shape[0] )*100,2)
null_stats

# nulls have been already handles in this data to demonstrate the variance concept
```

Out[36]:

```
ID           0.0
temp         0.0
atemp        0.0
humidity     0.0
windspeed    0.0
count        0.0
dtype: float64
```

In [37]:

```
# as since ID is a index variable, and of no use for model, we are dropping it
df.drop(['ID'], axis=1, inplace=True)
```

```
before proceeding, we will normalize the data because we are going to calculate variance in next steps and scaling is required for that

How Normalize() Works
---------------------

It considers each Row as an idividual Unit Vector. (whose magnitude is 1, having n dimensions, in the same direction as original vector)
So it divides each value by Square root of magnitude of each vector to obtain the new value.

Its an easy way of implemention Normalisation, we can go ahead and use Min_Max_Scalar as well.
```

In [38]:

```
normalize = normalize(df)
df_normalized = pd.DataFrame(normalize, columns=df.columns)
df_normalized.head()
```

Out[38]:

	temp	atemp	humidity	windspeed	count
0	0.116607	0.170585	0.959872	0.0	0.189604
1	0.099203	0.149960	0.879850	0.0	0.439925
2	0.102851	0.155473	0.912202	0.0	0.364881
3	0.126009	0.184339	0.960431	0.0	0.166475
4	0.127781	0.186932	0.973940	0.0	0.012986

In [40]:

```
df_normalized.describe()
```

Out[40]:

	temp	atemp	humidity	windspeed	count
count	12980.000000	12980.000000	12980.000000	12980.000000	12980.000000
mean	0.133424	0.157703	0.484986	0.098251	0.695990
std	0.076665	0.089314	0.305763	0.093572	0.334630
min	0.002577	0.000000	0.000000	0.000000	0.009497
25%	0.075355	0.089242	0.205452	0.036109	0.424225
50%	0.116010	0.138768	0.417778	0.073078	0.872426
75%	0.174950	0.207530	0.805784	0.132569	0.964887
max	0.558100	0.658868	0.991642	0.751237	0.998711

In [43]:

```
# calculating Variance
df_variance = df_normalized.var()
df_variance
```

Out[43]:

```
temp         0.005877
atemp        0.007977
humidity     0.093491
windspeed    0.008756
count        0.111977
dtype: float64
```

In [46]:

```
# list of variables with nulls below threshold
variables_above_threshold_list = []
variance_threshold = 0.006
for var in range(df.shape[1]):
    if df_variance[var] >= variance_threshold:
        variables_above_threshold_list.append(df.columns[var])
```

In [47]:

```
variables_above_threshold_list
```

Out[47]:

```
['atemp', 'humidity', 'windspeed', 'count']
```

In [49]:

```
new_df = df[variables_above_threshold_list]
new_df.head()
```

Out[49]:

	atemp	humidity	windspeed	count
0	14.395	81	0.0	16
1	13.635	80	0.0	40
2	13.635	80	0.0	32
3	14.395	75	0.0	13
4	14.395	75	0.0	1

In [50]:

```
df.shape, new_df.shape
```

Out[50]:

```
((12980, 5), (12980, 4))
```

In [ ]:

```

```

High Correlation Filter

```
if two or more features are highly correlated, we can select the one which is having highest correlation with target and drop others.
```

In [51]:

```
import numpy as np
import pandas as pd
```

In [53]:

```
df = pd.read_csv("Dataset/Dimensionality_Reduction/high_correlation_fllter.csv")
df.head()
```

Out[53]:

	ID	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count
0	AB101	1	0	0	1	9.84	14.395	81	0.0	16
1	AB102	1	0	0	1	9.02	13.635	80	0.0	40
2	AB103	1	0	0	1	9.02	13.635	80	0.0	32
3	AB104	1	0	0	1	9.84	14.395	75	0.0	13
4	AB105	1	0	0	1	9.84	14.395	75	0.0	1

In [54]:

```
# percentage of missing values in each feature
null_stats = round((df.isnull().sum() / df.shape[0] )*100,2)
null_stats

# nulls have been already handles in this data to demonstrate the correlation concept
```

Out[54]:

```
ID            0.0
season        0.0
holiday       0.0
workingday    0.0
weather       0.0
temp          0.0
atemp         0.0
humidity      0.0
windspeed     0.0
count         0.0
dtype: float64
```

In [63]:

```
# as we are calculating the correlation between the features first, we are dropping target feature
x_df = df.drop('count', axis=1)

# calculating correlation b/w each pair of features
corr_matrix = x_df.corr()
corr_matrix = corr_matrix.reset_index()
corr_matrix
```

Out[63]:

	index	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	season	1.000000	-0.010959	0.014343	-0.013005	0.394560	0.397765	0.181712	-0.135762
1	holiday	-0.010959	1.000000	-0.248558	-0.018406	-0.025104	-0.032903	-0.029520	0.021646
2	workingday	0.014343	-0.248558	1.000000	0.052788	0.060589	0.064840	0.028026	0.001986
3	weather	-0.013005	-0.018406	0.052788	1.000000	-0.093655	-0.094877	0.432497	0.011120
4	temp	0.394560	-0.025104	0.060589	-0.093655	1.000000	0.991839	-0.048478	-0.008669
5	atemp	0.397765	-0.032903	0.064840	-0.094877	0.991839	1.000000	-0.031606	-0.049997
6	humidity	0.181712	-0.029520	0.028026	0.432497	-0.048478	-0.031606	1.000000	-0.296975
7	windspeed	-0.135762	0.021646	0.001986	0.011120	-0.008669	-0.049997	-0.296975	1.000000

```
as this feature is hard to sample out the required results, there is a function melt() which will create individual rows od each combination and represent the same and hence would be easier to fetch details out
```

In [67]:

```
corr_matrix.reset_index()
corr_matrix.melt(id_vars='index', value_vars = corr_matrix.columns.tolist(), value_name = 'correlation')
```

Out[67]:

	index	variable	correlation
0	season	season	1.000000
1	holiday	season	-0.010959
2	workingday	season	0.014343
3	weather	season	-0.013005
4	temp	season	0.394560
...	...	...	...
59	weather	windspeed	0.011120
60	temp	windspeed	-0.008669
61	atemp	windspeed	-0.049997
62	humidity	windspeed	-0.296975
63	windspeed	windspeed	1.000000

64 rows × 3 columns

In [61]:

```
corr_matrix.reset_index()

```

Out[61]:

	index	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	season	1.000000	-0.010959	0.014343	-0.013005	0.394560	0.397765	0.181712	-0.135762
1	holiday	-0.010959	1.000000	-0.248558	-0.018406	-0.025104	-0.032903	-0.029520	0.021646
2	workingday	0.014343	-0.248558	1.000000	0.052788	0.060589	0.064840	0.028026	0.001986
3	weather	-0.013005	-0.018406	0.052788	1.000000	-0.093655	-0.094877	0.432497	0.011120
4	temp	0.394560	-0.025104	0.060589	-0.093655	1.000000	0.991839	-0.048478	-0.008669
5	atemp	0.397765	-0.032903	0.064840	-0.094877	0.991839	1.000000	-0.031606	-0.049997
6	humidity	0.181712	-0.029520	0.028026	0.432497	-0.048478	-0.031606	1.000000	-0.296975
7	windspeed	-0.135762	0.021646	0.001986	0.011120	-0.008669	-0.049997	-0.296975	1.000000

In [ ]:

```

```