Unit - 3

- **Feature Engineering:**
  - Feature Extraction and Engineering,
  - Feature Engineering on Numeric Data,
  - Feature Engineering on Categorical Data,
  - Feature Engineering on Text Data,
  - Feature Engineering on Temporal Data,
  - Feature Engineering on Image Data,
  - Feature Scaling,
  - Feature Selection.
- **Dimensionality Reduction:**
  - Feature extraction with Principal Component Analysis.

What is feature?

- A feature is an attribute of a data set that is used in a machine learning process.
- The features in a data set are also called its dimensions.
- So a data set having ‘n’ features is called an n-dimensional data set.
- Example:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
6.7	3.3	5.7	2.5	Virginica
4.9	3	1.4	0.2	Setosa
5.5	2.6	4.4	1.2	Versicolor
6.8	3.2	5.9	2.3	Virginica
5.5	2.5	4	1.3	Versicolor
5.1	3.5	1.4	0.2	Setosa
6.1	3	4.6	1.4	versicolor

Class variable or dependent variable

What is Feature Engineering?

- ***Feature engineering refers to the process of translating a data set into features such that these features are able to represent the data set more effectively and result in a better learning performance.***
- feature engineering is an important pre-processing step for machine learning. It has two major elements:
  1. feature transformation
  2. feature subset selection

Why Feature Engineering?

- Better representation of data
- Better performing models
- Essential for model building and evaluation:
- More flexibility on data types

How Do You Engineer Features?

- Feature engineering strategies can be applies on following data types:
  - Numeric data, Categorical data
  - Text data, Temporal data
  - Image data
- Another aspect into feature engineering has recently gained prominence. Here, the machine itself try to detect patterns and extract useful data representations from the raw data, which can be used as features. This process is also known as "**auto feature generation**".
- Deep Learning has proved to be extremely effective in this area and neural network architectures like convolutional neural networks (CNNs), recurrent neural networks (RNNs), and Long Short Term Memory networks (LSTMs) are extensively used for auto feature engineering and extraction

Feature Engineering on Numerical Data

- The Numeric data is an information expressed as numbers, which can represent in measurements, observations, or recordings.
- Numeric data can be single numbers or groups of numbers (vectors) that each represent a specific piece of information.
- The most common types of numeric data are whole numbers (integers) and decimal numbers (floats).
- Even though numeric data can be directly fed into Machine Learning models, you would still need to engineer features that are relevant to the scenario, problem, and domain before building a model. Hence the need for feature engineering remains

1. Raw Measures :

- Raw measures typically indicated using numeric variables directly as features without any form of transformation or engineering. Typically these features can indicate values or counts

2. Values :

- scalar values in its raw form indicate a specific measurement, metric, or observation belonging to a specific variable or field.

```
In [2]: poke_df = pd.read_csv('datasets/Pokemon.csv', encoding='utf-8')
   ...: poke_df.head()
```

Out[2]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

- there are several attributes there which represent numeric raw values which can be used directly.

```
In [3]: poke_df[['HP', 'Attack', 'Defense']].head()
Out[3]:
   HP  Attack  Defense
0  45      49       49
1  60      62       63
2  80      82       83
3  80     100      123
4  39      52       43
```

You can directly use these attributes as features that are depicted in the previous dataframe. These include each Pokémon's HP (Hit Points), Attack, and Defense stats. In fact, we can also compute some basic statistical measures on these fields using the following code.

```
In [4]: poke_df[['HP', 'Attack', 'Defense']].describe()
Out[4]:
                HP      Attack     Defense
count   800.000000  800.000000  800.000000
mean     69.258750   79.001250   73.842500
std      25.534669   32.457366   31.183501
min       1.000000    5.000000    5.000000
25%      50.000000   55.000000   50.000000
50%      65.000000   75.000000   70.000000
75%      80.000000  100.000000   90.000000
max     255.000000  190.000000  230.000000
```

3. Counts: Raw numeric measures can also indicate counts, frequencies and occurrences of specific attributes.

```
In [5]: popsong_df = pd.read_csv('datasets/song_views.csv', encoding='utf-8')
   ...: popsong_df.head(10)
```

Out[5]:

	user_id	song_id	title	listen_count
0	b6b799f34a204bd928ea014c243ddad6d0be4f8f	SOBONKR12A58A7A7E0	You're The One	2
1	b41ead730ac14f6b6717b9cf8859d5579f3f8d4d	SOBONKR12A58A7A7E0	You're The One	0
2	4c84359a164b161496d05282707cecbd50adbfc4	SOBONKR12A58A7A7E0	You're The One	0
3	779b5908593756abb6ff7586177c966022668b06	SOBONKR12A58A7A7E0	You're The One	0
4	dd88ea94f605a63d9fc37a214127e3f00e85e42d	SOBONKR12A58A7A7E0	You're The One	0
5	68f0359a2f1cedb0d15c98d88017281db79f9bc6	SOBONKR12A58A7A7E0	You're The One	0
6	116a4c95d63623a967edf2f3456c90ebbf964e6f	SOBONKR12A58A7A7E0	You're The One	17
7	45544491ccfcdc0b0803c34f201a6287ed4e30f8	SOBONKR12A58A7A7E0	You're The One	0
8	e701a24d9b6c59f5ac37ab28462ca82470e27cfb	SOBONKR12A58A7A7E0	You're The One	68
9	edc8b7b1fd592a3b69c3d823a742e1a064abec95	SOBONKR12A58A7A7E0	You're The One	0

Figure 4-4. *Song listen counts as a numeric feature*

4. Binarization:

- Often raw numeric frequencies or counts are not necessary in building models especially with regard to methods applied in building recommender engines.
- For example if I want to know if a person is interested or has listened to a particular song, I am more concerned about the various songs he/she has listened to.
- In this case, a binary feature is preferred as opposed to a count based feature. We can binarize our listen_count field from our earlier dataset in the following way

```
In [6]: watched = np.array(popsong_df['listen_count'])
   ...: watched[watched >= 1] = 1
   ...: popsong_df['watched'] = watched
```

- You can also use scikit-learn's Binarizer class here from its preprocessing module to perform the same task instead of numpy arrays

```
from sklearn.preprocessing import Binarizer

bn = Binarizer()
pd_watched = bn.transform([popsong_df['listen_count']])[0]
popsong_df['pd_watched'] = pd_watched
popsong_df.head(11)
```

	user_id	song_id	title	listen_count	watched	pd_watched
0	b6b799f34a204bd928ea014c243ddad6d0be4f8f	SOBONKR12A58A7A7E0	You're The One	2	1	1
1	b41ead730ac14f6b6717b9cf8859d5579f3f8d4d	SOBONKR12A58A7A7E0	You're The One	0	0	0
2	4c84359a164b161496d05282707cecbd50adbfc4	SOBONKR12A58A7A7E0	You're The One	0	0	0
3	779b5908593756abb6ff7586177c966022668b06	SOBONKR12A58A7A7E0	You're The One	0	0	0
4	dd88ea94f605a63d9fc37a214127e3f00e85e42d	SOBONKR12A58A7A7E0	You're The One	0	0	0
5	68f0359a2f1cedb0d15c98d88017281db79f9bc6	SOBONKR12A58A7A7E0	You're The One	0	0	0
6	116a4c95d63623a967edf2f3456c90ebbf964e6f	SOBONKR12A58A7A7E0	You're The One	17	1	1
7	45544491ccfcdc0b0803c34f201a6287ed4e30f8	SOBONKR12A58A7A7E0	You're The One	0	0	0
8	e701a24d9b6c59f5ac37ab28462ca82470e27cfb	SOBONKR12A58A7A7E0	You're The One	68	1	1
9	edc8b7b1fd592a3b69c3d823a742e1a064abec95	SOBONKR12A58A7A7E0	You're The One	0	0	0

5. Rounding:

- Often when dealing with numeric attributes like proportions or percentages, we may not need values with a high amount of precision. Hence it makes sense to round off these high precision percentages into numeric integers.

```
In [8]: items_popularity = pd.read_csv('datasets/item_popularity.csv', encoding='utf-8')
   ...: # rounding off percentages
   ...: items_popularity['popularity_scale_10'] =
                    np.array(np.round((items_popularity['pop_percent'] * 10)), dtype='int')
   ...: items_popularity['popularity_scale_100'] =
                    np.array(np.round((items_popularity['pop_percent'] * 100)), dtype='int')
   ...: items_popularity
Out[8]:
    item_id  pop_percent  popularity_scale_10  popularity_scale_100
0  it_01345      0.98324                   10                    98
1  it_03431      0.56123                    6                    56
2  it_04572      0.12098                    1                    12
3  it_98021      0.35476                    4                    35
4  it_01298      0.92101                    9                    92
5  it_90120      0.81212                    8                    81
6  it_10123      0.56502                    6                    57
```

6. Interactions:

Imagine you have data where you want to predict something (like a price) using various factors (like size and weight). A model can be like a math equation that uses these factors with certain weights (like how much each factor matters). This helps predict the outcome.

However, often in several real-world datasets and scenarios, it makes sense to also try to capture the interactions between these feature variables as a part of the input feature set. A simple depiction of the extension of the above linear regression formulation with interaction features would be $y = c_1x_1 + c_2x_2 + \ldots + c_nx_n + c_{11}x_1^2 + c_{22}x_2^2 + c_{12}x_1x_2 + \ldots$ where features like $\{x_1x_2, x_1^2, \ldots\}$ denote the interaction features. Let's try engineering some interaction features on our Pokémon dataset now.

Degree of polynomial 2 [y = c1 * x1 + c2 * x2 + c3 * x1^2 + c4 * x2^2 + c5 * x1 * x2 + c0]

Example

But sometimes, things aren't that simple. Sometimes, the way factors interact matters too. Like, size and weight might affect the outcome together, not just separately. So, we extend the model to include these interactions, which makes it more accurate for real-life situations. It's like adding extra parts to the equation to show how factors work together.

```
In [9]: atk_def = poke_df[['Attack', 'Defense']]
   ...: atk_def.head()
Out[9]:
   Attack  Defense
0      49       49
1      62       63
2      82       83
3     100      123
4      52       43
```

```
In [10]: from sklearn.preprocessing import PolynomialFeatures
    ...:
    ...: pf = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
    ...: res = pf.fit_transform(atk_def)
    ...: res
Out[10]:
array([[    49.,     49.,   2401.,   2401.,   2401.],
       [    62.,     63.,   3844.,   3906.,   3969.],
       [    82.,     83.,   6724.,   6806.,   6889.],
       ...,
       [   110.,     60.,  12100.,   6600.,   3600.],
       [   160.,     60.,  25600.,   9600.,   3600.],
       [   110.,    120.,  12100.,  13200.,  14400.]])


  In [12]: intr_features = pd.DataFrame(res,
      ...:                              columns=['Attack', 'Defense',
      ...:                                       'Attack^2', 'Attack x Defense', 'Defense^2'])
      ...: intr_features.head(5)
  Out[12]:
     Attack  Defense  Attack^2  Attack x Defense  Defense^2
  0    49.0     49.0    2401.0            2401.0     2401.0
  1    62.0     63.0    3844.0            3906.0     3969.0
  2    82.0     83.0    6724.0            6806.0     6889.0
  3   100.0    123.0   10000.0           12300.0    15129.0
  4    52.0     43.0    2704.0            2236.0     1849.0
```

7. Binning:

- Binning which is also known as quantization.
- The operation of binning is used for transforming continuous numeric values into discrete ones.
- These discrete numbers can be thought of as bins into which the raw values or numbers are binned or grouped into.
- Each bin represents a specific degree of intensity and has a specific range of values which must fall into that bin.
- There are various ways of binning data which include **fixed-width and adaptive binning.**

Fixed – width binning:

- In fixed-width binning, as the name indicates, we have specific fixed widths for each of the bins, which are usually pre-defined by the user analyzing the data.
- Each bin has a pre-fixed range of values which should be assigned to that bin on the basis of some business or custom logic, rules, or necessary transformations.

```
Age Range: Bin
---------------
 0 -  9   : 0
10 - 19   : 1
20 - 29   : 2
30 - 39   : 3
```

```
In [17]: fcc_survey_df['Age_bin_round'] = np.array(np.floor(np.array(fcc_survey_df['Age']) /
                                                                                        10.))
    ...: fcc_survey_df[['ID.x', 'Age', 'Age_bin_round']].iloc[1071:1076]
Out[17]:
                                ID.x   Age  Age_bin_round
1071  6a02aa4618c99fdb3e24de522a099431  17.0            1.0
1072  f0e5e47278c5f248fe861c5f7214c07a  38.0            3.0
1073  6e14f6d0779b7e424fa3fdd9e4bd3bf9  21.0            2.0
1074  c2654c07dc929cdf3dad4d1aec4ffbb3  53.0            5.0
1075  f07449fc9339b2e57703ec7886232523  35.0            3.0
```

Adaptive Binning :

- Quantile based binning is a good strategy to use for adaptive binning.
- These are specific values or cut-points which help in partitioning the continuous valued distribution of a specific numeric field into discrete contiguous bins or intervals.
- Thus, q-Quantiles help in partitioning a numeric attribute into q equal partitions.

```
In [23]: quantile_labels = ['0-25Q', '25-50Q', '50-75Q', '75-100Q']
    ...: fcc_survey_df['Income_quantile_range'] = pd.qcut(fcc_survey_df['Income'],
    ...:                                                  q=quantile_list)
    ...: fcc_survey_df['Income_quantile_label'] = pd.qcut(fcc_survey_df['Income'],
    ...:                                                  q=quantile_list,
    ...:                                                  labels=quantile_labels)
    ...: fcc_survey_df[['ID.x', 'Age', 'Income',
                        'Income_quantile_range', 'Income_quantile_label']].iloc[4:9]
```

Out[23]:

	ID.x	Age	Income	Income_quantile_range	Income_quantile_label
4	9368291c93d5d5f5c8cdb1a575e18bec	20.0	6000.0	(5999.999, 20000.0]	0-25Q
5	dd0e77eab9270e4b67c19b0d6bbf621b	34.0	40000.0	(37000.0, 60000.0]	50-75Q
6	7599c0aa0419b59fd11ffede98a3665d	23.0	32000.0	(20000.0, 37000.0]	25-50Q
7	6dff182db452487f07a47596f314bddc	35.0	40000.0	(37000.0, 60000.0]	50-75Q
8	9dc233f8ed1c6eb2432672ab4bb39249	33.0	80000.0	(60000.0, 200000.0]	75-100Q

Feature Engineering on Categorical Data

- Any attribute or feature that is categorical in nature represents discrete values that belong to a specific finite set of categories or classes.
- Category or class labels can be text or numeric in nature. Usually there are two types of categorical variables—nominal and ordinal.
- Nominal Categorical feature - doesn't have any order

  Example: video game genres, weather seasons, country names
- Ordinal Categorical feature – have specified order

  Example: Clothing size, education levels

Engineer features from categorical data

- Transforming Nominal features
- Transforming Ordinal features
- Encoding Categorical features
  - One hot encoding
  - Dummy coding scheme
  - Effect coding scheme
  - Bin – counting scheme
  - Feature Hashing scheme

Transforming Nominal Features

- Nominal features or attributes are categorical variables that usually have a finite set of distinct discrete values.
- Often these values are in string or text format and Machine Learning algorithms cannot understand them directly.

```
In [2]: vg_df = pd.read_csv('datasets/vgsales.csv', encoding='utf-8')
   ...: vg_df[['Name', 'Platform', 'Year', 'Genre', 'Publisher']].iloc[1:7]
Out[2]:
                      Name Platform    Year         Genre Publisher
1         Super Mario Bros.      NES  1985.0      Platform  Nintendo
2            Mario Kart Wii      Wii  2008.0        Racing  Nintendo
3         Wii Sports Resort      Wii  2009.0        Sports  Nintendo
4  Pokemon Red/Pokemon Blue       GB  1996.0  Role-Playing  Nintendo
5                    Tetris       GB  1989.0        Puzzle  Nintendo
6     New Super Mario Bros.       DS  2006.0      Platform  Nintendo
```

Features like Platform, Genre, and Publisher are nominal categorical variables. Let's now try to transform the video game Genre feature into a numeric representation

```
In [3]: genres = np.unique(vg_df['Genre'])
   ...: genres
Out[3]:
array(['Action', 'Adventure', 'Fighting', 'Misc', 'Platform', 'Puzzle',
       'Racing', 'Role-Playing', 'Shooter', 'Simulation', 'Sports',
       'Strategy'], dtype=object)
```

```
In [4]: from sklearn.preprocessing import LabelEncoder
   ...:
   ...: gle = LabelEncoder()
   ...: genre_labels = gle.fit_transform(vg_df['Genre'])
   ...: genre_mappings = {index: label for index, label in enumerate(gle.classes_)}
   ...: genre_mappings
Out[4]:
{0: 'Action', 1: 'Adventure', 2: 'Fighting', 3: 'Misc',
 4: 'Platform', 5: 'Puzzle', 6: 'Racing', 7: 'Role-Playing',
 8: 'Shooter', 9: 'Simulation', 10: 'Sports', 11: 'Strategy'}
```

From the output, we can see that a mapping scheme has been generated where each genre value is mapped to a number with the help of the `LabelEncoder` object `gle`. The transformed labels are stored in the `genre_labels` value. Let's write it back to the original dataframe and view the results.

```
In [5]: vg_df['GenreLabel'] = genre_labels
   ...: vg_df[['Name', 'Platform', 'Year', 'Genre', 'GenreLabel']].iloc[1:7]
Out[5]:
                      Name Platform    Year         Genre  GenreLabel
1          Super Mario Bros.      NES  1985.0      Platform           4
2             Mario Kart Wii      Wii  2008.0        Racing           6
3          Wii Sports Resort      Wii  2009.0        Sports          10
4  Pokemon Red/Pokemon Blue       GB  1996.0  Role-Playing           7
5                    Tetris       GB  1989.0        Puzzle           5
6      New Super Mario Bros.       DS  2006.0      Platform           4
```

The `GenreLabel` field depicts the mapped numeric labels for each of the `Genre` labels and we can clearly see that this adheres to the mappings that we generated earlier.

Transforming Ordinal Features

- Ordinal features are similar to nominal features except that order matters and is an inherent property with which we can interpret the values of these features

```
In [6]: poke_df = pd.read_csv('datasets/Pokemon.csv', encoding='utf-8')
   ...: poke_df = poke_df.sample(random_state=1, frac=1).reset_index(drop=True)
   ...:
   ...: np.unique(poke_df['Generation'])
Out[6]: array(['Gen 1', 'Gen 2', 'Gen 3', 'Gen 4', 'Gen 5', 'Gen 6'], dtype=object)
```

```
In [7]: gen_ord_map = {'Gen 1': 1, 'Gen 2': 2, 'Gen 3': 3,
   ...:                'Gen 4': 4, 'Gen 5': 5, 'Gen 6': 6}
   ...:
   ...: poke_df['GenerationLabel'] = poke_df['Generation'].map(gen_ord_map)
   ...: poke_df[['Name', 'Generation', 'GenerationLabel']].iloc[4:10]
Out[7]:
                  Name Generation  GenerationLabel
4            Octillery      Gen 2                2
5          Helioptile       Gen 6                6
6              Dialga       Gen 4                4
7  DeoxysDefense Forme      Gen 3                3
8             Rapidash      Gen 1                1
9              Swanna       Gen 5                5
```

Thus, you can see that it is really easy to build your own transformation mapping scheme with the help of Python dictionaries and use the `map(...)` function from `pandas` to transform the ordinal feature.

Encoding Categorical Features

1. One Hot Encoding Scheme:

- Considering we have numeric representation of any categorical feature with **m labels,** the one hot encoding scheme, encodes or transforms the feature into **m binary features**, which can only contain a value of 1 or 0.
- Each observation in the categorical feature is thus converted into a vector of size m with only one of the values as 1 (indicating it as active).

```
In [8]: poke_df[['Name', 'Generation', 'Legendary']].iloc[4:10]
Out[8]:
                    Name Generation  Legendary
4              Octillery      Gen 2      False
5           Helioptile      Gen 6      False
6               Dialga      Gen 4       True
7  DeoxysDefense Forme      Gen 3       True
8              Rapidash      Gen 1      False
9                Swanna      Gen 5      False
```

```
In [9]: from sklearn.preprocessing import OneHotEncoder, LabelEncoder
   ...:
   ...: # transform and map pokemon generations
   ...: gen_le = LabelEncoder()

   ...: gen_labels = gen_le.fit_transform(poke_df['Generation'])
   ...: poke_df['Gen_Label'] = gen_labels
   ...:
   ...: # transform and map pokemon legendary status
   ...: leg_le = LabelEncoder()
   ...: leg_labels = leg_le.fit_transform(poke_df['Legendary'])
   ...: poke_df['Lgnd_Label'] = leg_labels
   ...:
   ...: poke_df_sub = poke_df[['Name', 'Generation', 'Gen_Label', 'Legendary', 'Lgnd_Label']]
   ...: poke_df_sub.iloc[4:10]
Out[9]:
                  Name Generation  Gen_Label  Legendary  Lgnd_Label
4             Octillery      Gen 2          1      False           0
5           Helioptile      Gen 6          5      False           0
6               Dialga      Gen 4          3       True           1
7  DeoxysDefense Forme      Gen 3          2       True           1
8             Rapidash      Gen 1          0      False           0
9               Swanna      Gen 5          4      False           0
```

```
In [10]: # encode generation labels using one-hot encoding scheme
    ...: gen_ohe = OneHotEncoder()
    ...: gen_feature_arr = gen_ohe.fit_transform(poke_df[['Gen_Label']]).toarray()
    ...: gen_feature_labels = list(gen_le.classes_)
    ...: gen_features = pd.DataFrame(gen_feature_arr, columns=gen_feature_labels)
    ...:
    ...: # encode legendary status labels using one-hot encoding scheme
    ...: leg_ohe = OneHotEncoder()
    ...: leg_feature_arr = leg_ohe.fit_transform(poke_df[['Lgnd_Label']]).toarray()
    ...: leg_feature_labels = ['Legendary_'+str(cls_label) for cls_label in leg_le.classes_]
    ...: leg_features = pd.DataFrame(leg_feature_arr, columns=leg_feature_labels)
```

Now, you should remember that you can always encode both the features together using the `fit_transform(...)` function by passing it a two-dimensional array of the two features. But we are depicting this encoding for each feature separately, to make things easier to understand. Besides this, we can also create separate dataframes and label them accordingly. Let's now concatenate these feature frames and see the final result.

```
In [11]: poke_df_ohe = pd.concat([poke_df_sub, gen_features, leg_features], axis=1)
    ...: columns = sum([['Name', 'Generation', 'Gen_Label'],gen_feature_labels,
    ...:                ['Legendary', 'Lgnd_Label'],leg_feature_labels], [])
    ...: poke_df_ohe[columns].iloc[4:10]
```

Out[11]:

	Name	Generation	Gen_Label	Gen 1	Gen 2	Gen 3	Gen 4	Gen 5	Gen 6	Legendary	Lgnd_Label	Legendary_False	Legendary_True
4	Octillery	Gen 2	1	0.0	1.0	0.0	0.0	0.0	0.0	False	0	1.0	0.0
5	Helioptile	Gen 6	5	0.0	0.0	0.0	0.0	0.0	1.0	False	0	1.0	0.0
6	Dialga	Gen 4	3	0.0	0.0	0.0	1.0	0.0	0.0	True	1	0.0	1.0
7	DeoxysDefense Forme	Gen 3	2	0.0	0.0	1.0	0.0	0.0	0.0	True	1	0.0	1.0
8	Rapidash	Gen 1	0	1.0	0.0	0.0	0.0	0.0	0.0	False	0	1.0	0.0
9	Swanna	Gen 5	4	0.0	0.0	0.0	0.0	1.0	0.0	False	0	1.0	0.0

Figure 4-15. *Feature set depicting one hot encoded features for Pokémon generation and legendary status*

Pandas also provides a wonderful function called `to_dummies(...)`, which helps us easily perform one hot encoding. The following code depicts how to achieve this.

```
In [15]: gen_onehot_features = pd.get_dummies(poke_df['Generation'])
    ...: pd.concat([poke_df[['Name', 'Generation']], gen_onehot_features], axis=1).
iloc[4:10]
Out[15]:
                  Name Generation  Gen 1  Gen 2  Gen 3  Gen 4  Gen 5  Gen 6
4            Octillery      Gen 2      0      1      0      0      0      0
5           Helioptile      Gen 6      0      0      0      0      0      1
6               Dialga      Gen 4      0      0      0      1      0      0
7  DeoxysDefense Forme      Gen 3      0      0      1      0      0      0
8             Rapidash      Gen 1      1      0      0      0      0      0
9               Swanna      Gen 5      0      0      0      0      1      0
```

The output depicts the one hot encoding scheme for Pokémon generation values similar to what we depicted in our previous analyses.

2. Dummy Coding Scheme:

- The dummy coding scheme is similar to the one hot encoding scheme, except in the case of dummy coding scheme, when applied on a categorical feature with m distinct labels, we get m-1 binary features.
- Thus each value of the categorical variable gets converted into a vector of size m-1.
- The extra feature is completely disregarded and thus if the category values range from {0, 1, ..., m-1} the 0th or the m-1th feature is usually represented by a vector of all zeros (0).

```
In [16]: gen_dummy_features = pd.get_dummies(poke_df['Generation'], drop_first=True)
    ...: pd.concat([poke_df[['Name', 'Generation']], gen_dummy_features], axis=1).iloc[4:10]
Out[16]:
                 Name Generation  Gen 2  Gen 3  Gen 4  Gen 5  Gen 6
4           Octillery      Gen 2      1      0      0      0      0
5          Helioptile      Gen 6      0      0      0      0      1
6              Dialga      Gen 4      0      0      1      0      0
7  DeoxysDefense Forme      Gen 3      0      1      0      0      0
8            Rapidash      Gen 1      0      0      0      0      0
9              Swanna      Gen 5      0      0      0      1      0
```

If you want, you can also choose to drop the last level binary encoded feature (Gen 6) by using the following code.

```
In [17]: gen_onehot_features = pd.get_dummies(poke_df['Generation'])
    ...: gen_dummy_features = gen_onehot_features.iloc[:,:-1]
    ...: pd.concat([poke_df[['Name', 'Generation']], gen_dummy_features], axis=1).iloc[4:10]
Out[17]:
                  Name Generation  Gen 1  Gen 2  Gen 3  Gen 4  Gen 5
4            Octillery      Gen 2      0      1      0      0      0
5          Helioptile       Gen 6      0      0      0      0      0
6              Dialga       Gen 4      0      0      0      1      0
7  DeoxysDefense Forme      Gen 3      0      0      1      0      0
8             Rapidash      Gen 1      1      0      0      0      0
9              Swanna       Gen 5      0      0      0      0      1
```

Thus from these outputs you can see that based on the encoded level binary feature which we drop, that particular categorical value is represented by a vector/encoded features, which all represent 0. For example in the previous result feature set, Pokémon Heloptile belongs to Gen 6 and is represented by all 0s in the encoded dummy features.

3. Effect Coding Scheme:

- The effect coding scheme is very similar to the dummy coding scheme in most aspects.
- However, the encoded features or feature vector, for the category values that represent all 0s in the dummy coding scheme, is replaced by -1s in the effect coding scheme.

```
In [18]: gen_onehot_features = pd.get_dummies(poke_df['Generation'])
    ...: gen_effect_features = gen_onehot_features.iloc[:,:-1]
    ...: gen_effect_features.loc[np.all(gen_effect_features == 0, axis=1)] = -1.
    ...: pd.concat([poke_df[['Name', 'Generation']], gen_effect_features], axis=1).iloc[4:10]
Out[18]:
                 Name Generation  Gen 1  Gen 2  Gen 3  Gen 4  Gen 5
4           Octillery      Gen 2    0.0    1.0    0.0    0.0    0.0
5          Helioptile      Gen 6   -1.0   -1.0   -1.0   -1.0   -1.0
6              Dialga      Gen 4    0.0    0.0    0.0    1.0    0.0
7  DeoxysDefense Forme      Gen 3    0.0    0.0    1.0    0.0    0.0
8            Rapidash      Gen 1    1.0    0.0    0.0    0.0    0.0
9              Swanna      Gen 5    0.0    0.0    0.0    0.0    1.0
```

We can clearly see from the output feature set that all 0s have been replaced by -1 in case of values which were previously all 0 in the dummy coding scheme.

4. **Bin-Counting Scheme:**

- The bin-counting scheme is useful for dealing with categorical variables with many categories.
- In this scheme, instead of using the actual label values for encoding, we use probability based statistical information about the value and the actual target or response value which we aim to predict in our modeling efforts .

Imagine you're working on a cybersecurity project, and you want to predict whether a certain IP address is likely to be associated with a DDOS (Distributed Denial of Service) attack. In this case, the IP addresses are the categorical features, and the target variable is whether or not a DDOS attack occurred.

IP Address	Associated with DDOS
192.168.1.1	Yes
192.168.1.2	No
192.168.1.1	No
192.168.1.3	Yes
192.168.1.2	No
192.168.1.1	Yes

- For IP address 192.168.1.1: It has been associated with DDOS attacks 2 times out of 3 occurrences. So, the probability of a DDOS attack is 2/3 ≈ 0.67.
- For IP address 192.168.1.2: It has not been associated with any DDOS attacks. So, the probability of a DDOS attack is 0/2 = 0.
- For IP address 192.168.1.3: It has been associated with a DDOS attack 1 time out of 1 occurrence. So, the probability of a DDOS attack is 1/1 = 1.

When you encounter a new IP address in the future, you look up its probability value in the historical data and use that probability as the encoded feature value. For instance, if you encounter IP address 192.168.1.1 again, you would encode it with the probability of 0.67, indicating the likelihood of it being associated with a DDOS attack.

5. Feature Hashing Scheme:

- Hashing schemes work on strings, numbers and other structures like vectors.
- You can think of hashed outputs as a finite set of **h** bins such that when hash function is applied on the same values, they get assigned to the same bin out of the h bins based on the hash value.
- We can assign the value of h, which becomes the final size of the encoded feature vector for each categorical feature we encode using the feature hashing scheme.
- Thus even if we have over 1000 distinct categories in a feature and we set h = 10, the output feature set will still have only 10 features as compared to 1000 features if we used a one hot encoding scheme.

```
In [21]: from sklearn.feature_extraction import FeatureHasher
    ...:
    ...: fh = FeatureHasher(n_features=6, input_type='string')
    ...: hashed_features = fh.fit_transform(vg_df['Genre'])
    ...: hashed_features = hashed_features.toarray()
    ...: pd.concat([vg_df[['Name', 'Genre']], pd.DataFrame(hashed_features)], axis=1).
         iloc[1:7]
Out[21]:
                      Name         Genre    0    1    2    3    4    5
1          Super Mario Bros.      Platform  0.0  2.0  2.0 -1.0  1.0  0.0
2             Mario Kart Wii       Racing -1.0  0.0  0.0  0.0  0.0 -1.0
3          Wii Sports Resort        Sports -2.0  2.0  0.0 -2.0  0.0  0.0
4  Pokemon Red/Pokemon Blue  Role-Playing -1.0  1.0  2.0  0.0  1.0 -1.0
5                    Tetris       Puzzle  0.0  1.0  1.0 -2.0  1.0 -1.0
6      New Super Mario Bros.      Platform  0.0  2.0  2.0 -1.0  1.0  0.0
```

Feature Engineering on Text Data

- In case of unstructured data like text documents, the first challenge is dealing with the unpredictable nature of the syntax, format, and content of the documents, which make it a challenge to extract useful information for building models.
- The second challenge is transforming these textual representations into numeric representations that can be understood by Machine Learning algorithms.
- There exist various feature engineering techniques employed by data scientists daily to extract numeric feature vectors from unstructured text

load some sample text documents, do some basic pre-processing

```
In [2]: corpus = ['The sky is blue and beautiful.',
   ...:           'Love this blue and beautiful sky!',
   ...:           'The quick brown fox jumps over the lazy dog.',
   ...:           'The brown fox is quick and the blue dog is lazy!',
   ...:           'The sky is very blue and the sky is very beautiful today',
   ...:           'The dog is lazy but the brown fox is quick!'
   ...: ]
   ...: labels = ['weather', 'weather', 'animals', 'animals', 'weather', 'animals']
   ...: corpus = np.array(corpus)
   ...: corpus_df = pd.DataFrame({'Document': corpus,
   ...:                           'Category': labels})
   ...: corpus_df = corpus_df[['Document', 'Category']]
   ...: corpus_df
Out[2]:
                                            Document Category
0                    The sky is blue and beautiful.  weather
1                  Love this blue and beautiful sky!  weather
2       The quick brown fox jumps over the lazy dog.  animals
3   The brown fox is quick and the blue dog is lazy!  animals
4  The sky is very blue and the sky is very beaut...  weather
5    The dog is lazy but the brown fox is quick!      animals
```

Text pre processing

- Text tokenization and lower casing
- Removing special characters
- Removing stop words
- Correcting spellings
- Stemming
- Lemmatization

Tokenization

Corpus: collection of text documents.

Tokens: Tokens are a basic meaningful unit of a sentence or a document

N-Grams: N-grams is a combination of N words or characters together. for example, if we have a sentence “I Love My Phone”.

Sentence : I love my phone

Unigrams (n =1) : I, Love, my, phone
Bigrams (n=2) : I Love, Love my, my phone
Trigrams (n=3) : I love my, love my phone

Tokenization: process of splitting a text object into smaller units known as tokens. Examples of tokens can be words, characters, numbers, symbols, or n-grams.

White space tokenizer / Unigram tokenizer

Sentence : "I went to New-York to play football"
Tokens : "I", "went", "to", "New-York", "to", "play", "football"

Regular expression tokenization: In which a regular expression pattern is used to get the tokens. For example, consider the following string containing multiple delimiters. We can split the sentence by passing a splitting pattern.

Regular expression tokenizer

Sentence : "Football,Cricket;Golf Tennis"

re.split(r'[;,\s]', line)

Tokens : "Football", "Cricket", "Golf", "Tennis"

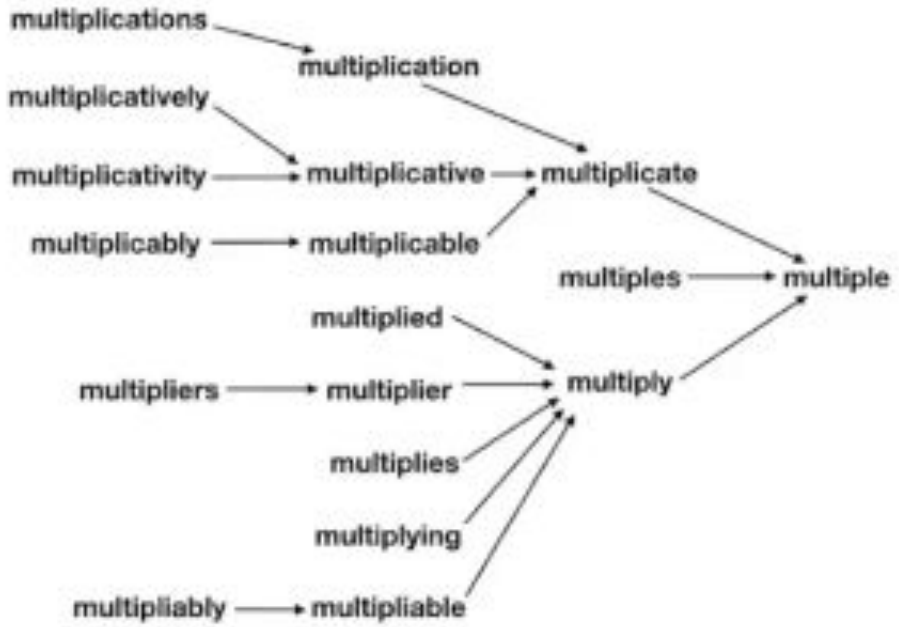
Normalization: Normalization is the process of converting a token into its base form. Two popular methods used for normalization are stemming and lemmatization.

- **Stemming:** removing inflationary forms from a given token. For example laughing, laughed, laughs, laugh all will become laugh after the stemming process.

Form	Suffix	Stem
studies	-es	studi
studying	-ing	study
niñas	-as	niñ
niñez	-ez	niñ

Stemming is not a good process for normalization. since sometimes it can produce non-meaningful words which are not present in the dictionary. Consider the sentence ” His teams are not winning”. After stemming we get “Hi team are not winn ” . Notice that the keyword winn is not a regular word. Also, “hi” has changed the context of the entire sentence.

Lemmatization: Lemmatization is a systematic process of removing the inflectional form of a token and transform it into a lemma. It makes use of word structure, vocabulary, part of speech tags, and grammar relations.



- The output of lemmatization is a root word called a lemma. for example “am”, “are”, “is” will be converted to “be”. Similarly, running runs, ‘ran’ will be replaced by ‘run’.

Removing Stop Words: useless words (data), are referred to as stop words.

- A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Bag of Words model

- The core principle of this model is to convert text documents into numeric vectors.
- The dimension or size of each vector is N where N indicates all possible distinct words across the corpus of documents.
- Each document once transformed is a numeric vector of size N where the values or weights in the vector indicate the frequency of each word in that specific document.

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)

    ...: cv_matrix = cv.fit_transform(norm_corpus)
    ...: cv_matrix = cv_matrix.toarray()
    ...: cv_matrix
Out[5]:
array([[1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1],
       [0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]], dtype=int64)
```

The output represents a numeric term frequency based feature vector for each document like we mentioned before. To understand it better, we can represent it using the feature names and view it as a data frame.

```
In [6]: vocab = cv.get_feature_names()
   ...: pd.DataFrame(cv_matrix, columns=vocab)
Out[6]:
   beautiful  blue  brown  dog  fox  jumps  lazy  love  quick  sky  today
0          1     1      0    0    0      0     0     0      0    1      0
1          1     1      0    0    0      0     0     1      0    1      0
2          0     0      1    1    1      1     1     0      1    0      0
3          0     1      1    1    1      0     1     0      1    0      0
4          1     1      0    0    0      0     0     0      0    2      1
5          0     0      1    1    1      0     1     0      1    0      0
```

Bag of N-grams model

- An n-gram is basically a collection of word tokens from a text document such that these tokens are contiguous and occur in a sequence.
- Bi-grams indicate n-grams of order 2 (two words), Tri-grams indicate n-grams of order 3 (three words), and so on.
- We can easily extend the bag of words model to use a bag of n-grams model to give us n-gram based feature vectors.

```
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(norm_corpus)
bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)
```

	beautiful sky	beautiful today	blue beautiful	blue dog	blue sky	brown fox	dog lazy	fox jumps	fox quick	jumps lazy	lazy brown	lazy dog	love blue	quick blue	quick brown	sky beautiful	sky blue
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0
3	0	0	0	1	0	1	1	0	1	0	0	0	0	1	0	0	0
4	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
5	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0

TF-IDF

- TF-IDF stands for Term Frequency-Inverse Document Frequency, which uses a combination of two metrics in its computation, namely: term frequency (TF) and inverse document frequency (IDF).

- Term Frequency (TF)-
  - It gives us the recurrence of the word in each report in the corpus.
  - It is the proportion of the number of times the word shows up in a report contrasted with the all-out the number of words in that record.
  - It increments as the quantity of events of that word inside the record increments.

TF-IDF

- Inverse Data Frequency (IDF)-
  - It is used to figure out the heaviness of uncommon words over all reports in the corpus.
  - The words that occur seldom in the corpus have a high IDF score.

$$idf_i = \log(\frac{n}{df_i})$$

- Joining these two, we think of the TF-IDF score (w) for a word in a record in the corpus.

$$w_{i,j} = tf_{i,j} \times idf_i$$

The cycle is ridden on the track.

The bus is driven on the road.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
cycle	1/7	0	log(2/1) = 0.3	0.043	0
bus	0	1/7	log(2/1) = 0.3	0	0.043
is	1/7	1/7	log(2/2) = 0	0	0
ridden	1/7	0	log(2/1) = 0.3	0.043	0
driven	0	1/7	log(2/1) = 0.3	0	0.043
on	1/7	1/7	log(2/2) = 0	0	0
the	1/7	1/7	log(2/2) = 0	0	0
track	1/7	0	log(2/1) = 0.3	0.043	0
road	0	1/7	log(2/1) = 0.3	0	0.043

Suppose we have 3 texts and we need to vectorize these texts using TF-IDF

Text 1	i love natural language processing but i hate python
Text 2	i like image processing
Text 3	i like signal processing and image processing

Create a term frequency matrix where rows are documents and columns are distinct terms throughout all documents. Count word occurrences in every text.

	and	*but*	*hate*	*i*	*image*	*language*	*like*	*love*	*natural*	*processing*	*python*	*signal*
Text 1	0	1	1	2	0	1	0	1	1	1	1	0
Text 2	0	0	0	1	1	0	1	0	0	1	0	0
Text 3	1	0	0	1	1	0	1	0	0	2	0	1

Compute inverse document frequency (IDF) using the previously explained formula.

Term	and	but	hate	i	image	language	like	love	natural	processing	python	signal
IDF	0.47712	0.47712	0.4771	0	0.1760913	0.477121	0.1760913	0.477121	0.47712125	0	0.477121	0.477121

Multiply TF matrix with IDF respectively

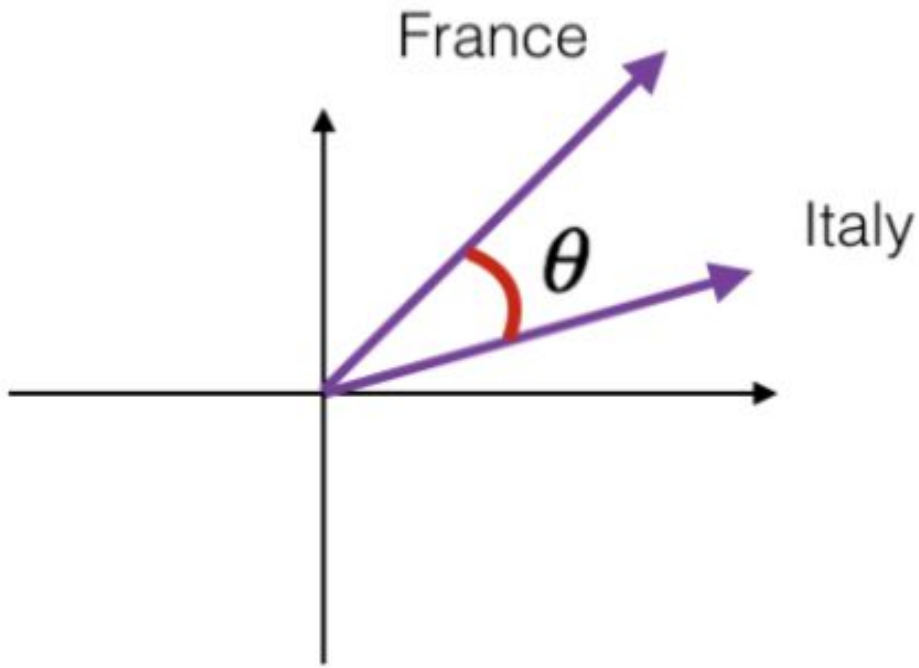
	and	but	hate	i	image	language	like	love	natural	processing	python	signal
Text 1	0	0.47712	0.4771	0	0	0.477121	0	0.477121	0.47712125	0	0.477121	0
Text 2	0	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0
Text 3	0.47712	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0.477121

Document Similarity:

- This is very useful in domains like search engines, document clustering, and information retrieval.
- Document similarity is the process of using a distance or similarity based metric that can be used to identify how similar a text document is with another document based on features extracted from the documents like bag of words or tf-idf.
- Pairwise document similarity in a corpus involves computing document similarity for each pair of documents in a corpus.

- Thus if you have C documents in a corpus, you would end up with a C x C matrix such that each row and column represents the similarity score for a pair of documents, which represent the indices at the row and column, respectively.
- There are several similarity and distance metrics that are used to compute document similarity. These include cosine similarity, Euclidean distance, Jaccard distance, and so on.

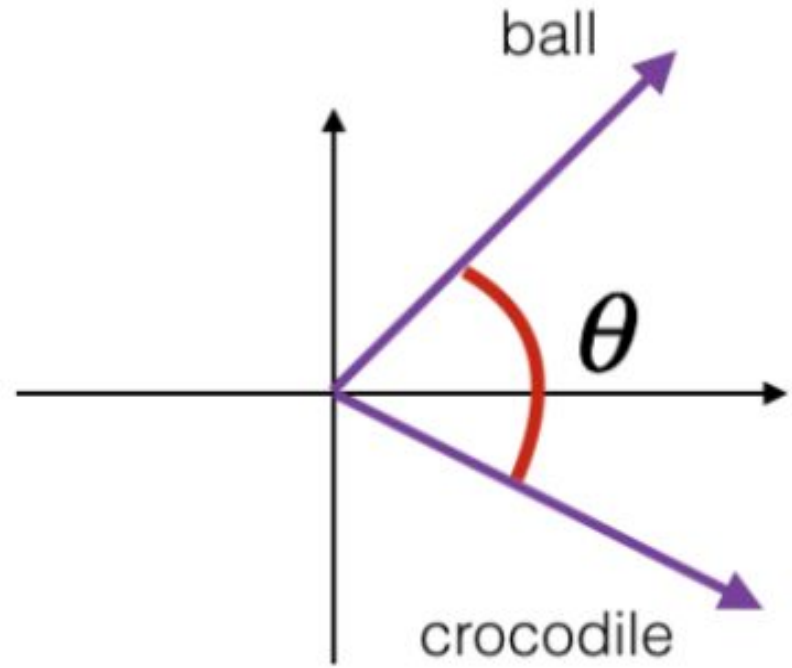
$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$



France and Italy are quite similar

θ is close to 0°

$\cos(\theta) \approx 1$



ball and crocodile are not similar

θ is close to 90°

$\cos(\theta) \approx 0$

Euclidean Distance:

- Is probably one of the most known formulas for computing the distance between two points applying the Pythagorean theorem.
- To get it you just need to subtract the points from the vectors, raise them to squares, add them up and take the square root of them

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

```
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!'
]
```

```
In [9]: from sklearn.metrics.pairwise import cosine_similarity
   ...:
   ...: similarity_matrix = cosine_similarity(tv_matrix)
   ...: similarity_df = pd.DataFrame(similarity_matrix)
   ...: similarity_df
Out[9]:
          0         1         2         3         4         5
0  1.000000  0.753128  0.000000  0.185447  0.807539  0.000000
1  0.753128  1.000000  0.000000  0.139665  0.608181  0.000000
2  0.000000  0.000000  1.000000  0.784362  0.000000  0.839987
3  0.185447  0.139665  0.784362  1.000000  0.109653  0.933779
4  0.807539  0.608181  0.000000  0.109653  1.000000  0.000000
5  0.000000  0.000000  0.839987  0.933779  0.000000  1.000000
```

Topic Models

- The idea of topic models revolves around the process of extracting key themes or concepts from a corpus of documents which are represented as topics.
- Each topic can be represented as a bag or collection of words/terms from the document corpus.
- Together, these terms signify a specific topic, theme or a concept and each topic can be easily distinguished from other topics by virtue of the semantic meaning conveyed by these terms.

- Topic models are extremely useful in summarizing large corpus of text documents to extract and depict key concepts. They are also useful in extracting features from text data that capture latent patterns in the data.

	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	0
1	Love this blue and beautiful sky!	weather	0
2	The quick brown fox jumps over the lazy dog.	animals	1
3	The brown fox is quick and the blue dog is lazy!	animals	1
4	The sky is very blue and the sky is very beaut...	weather	0
5	The dog is lazy but the brown fox is quick!	animals	1

Feature Engineering on Image Data

Object Detection:

- The histogram of oriented gradients, also known as HOG, is one of the techniques that’s extensively used in object detection.
- The technique being discussed involves something called the "HOG algorithm," which is used for a process called "feature engineering." Feature engineering is a way to extract useful information from an image that can be used for various tasks, like object detection.

The HOG algorithm works in several steps, similar to how edge detection works. Here's a summary of the steps:

2. **Normalization and Denoising**: The image is adjusted to remove unwanted effects caused by lighting variations. Think of it as making sure the image looks consistent regardless of whether it was taken in bright sunlight or dim indoor lighting. Noise, which refers to random variations in pixel values, is also reduced to make the image cleaner.

2. Gradient Computation: The algorithm calculates what's known as "first order gradients" for the image. Gradients represent how the intensity of the image changes from one pixel to its neighboring pixels. These gradients help capture important information about the edges, textures, and other details in the image.

3. **Gradient Histograms**: The gradients are then organized into histograms. Think of a histogram as a way to count how many gradients are pointing in different directions. This helps capture information about the direction of edges and textures in the image.

4.**Cell-based Gradients**: The image is divided into smaller sections called "cells." These cells could be thought of as small windows. The histograms of gradients are calculated separately for each cell. This allows the algorithm to capture local information about different parts of the image.

5. Normalization of Cells: The histograms within each cell are normalized. This is done to make the algorithm more robust to changes in lighting and contrast. It ensures that the algorithm focuses on the relative relationships between gradients rather than their absolute values.

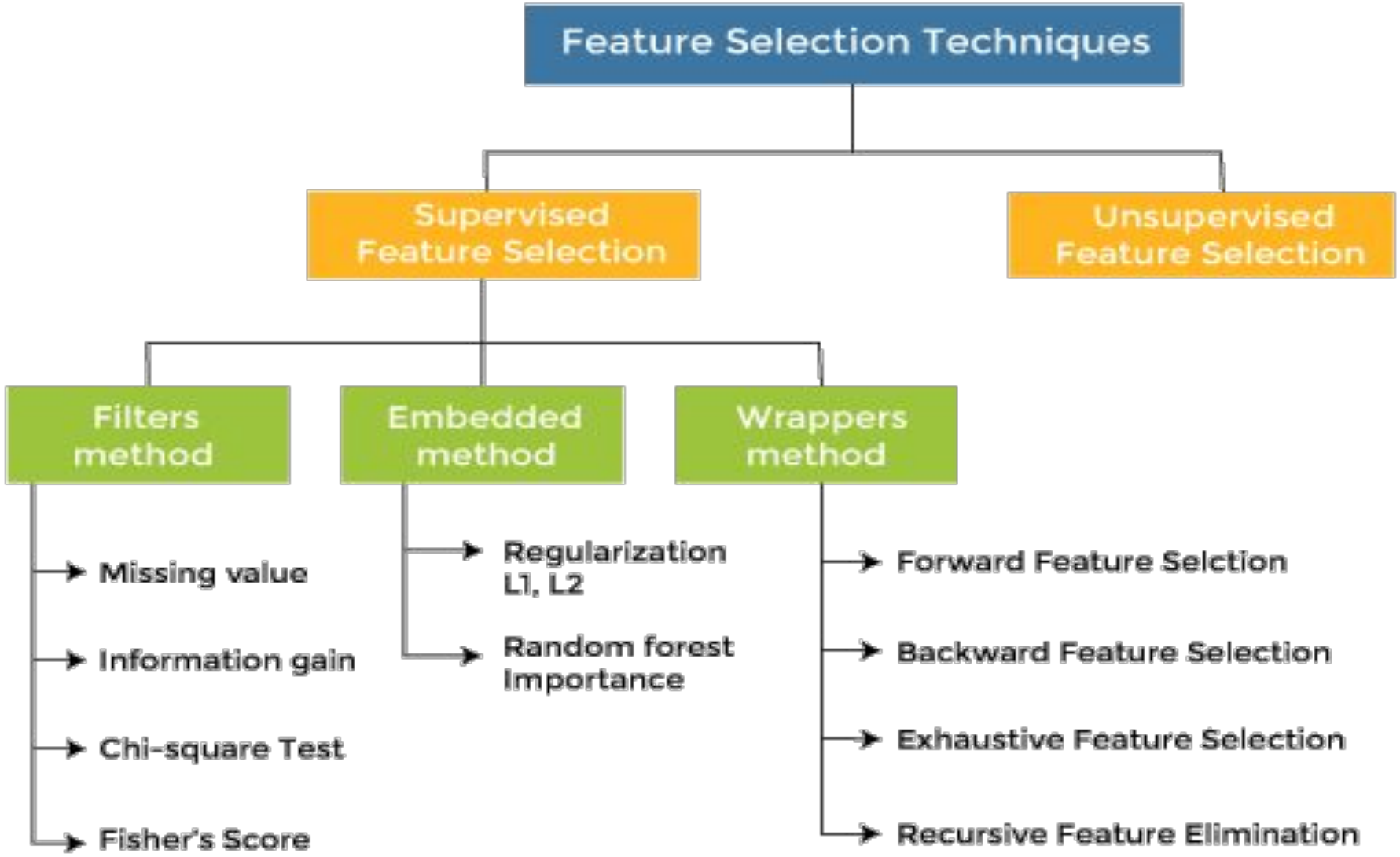
6. Flattened Feature Descriptor: The normalized histograms from all the cells are combined into a single, long list. This list of values is what's referred to as a "feature descriptor." It summarizes the important information about the image's edges, textures, and other details in a way that's suitable for analysis by machine learning models.

7. Object Detection: This feature descriptor can then be used as input to machine learning models for tasks like object detection. Object detection involves finding and identifying objects within an image

Feature selection

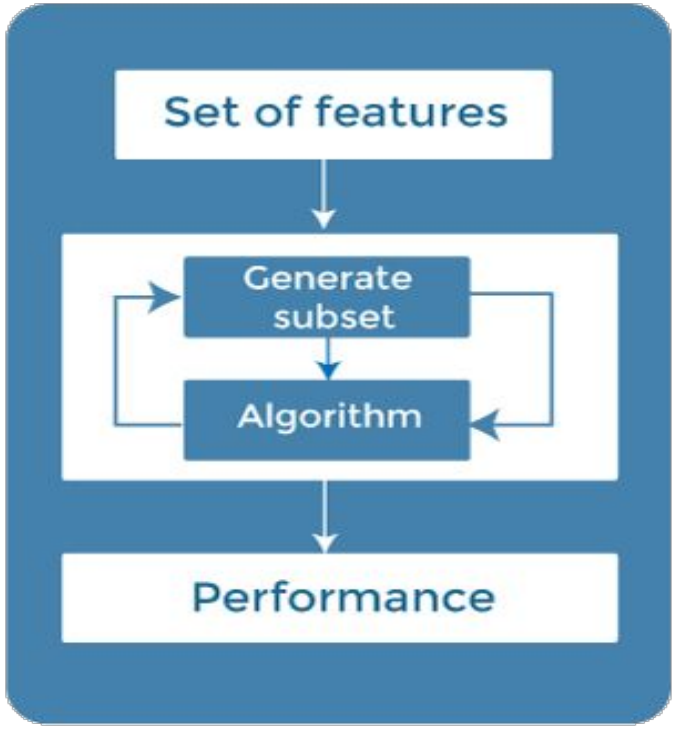
- Feature selection is a way of selecting the subset of the most relevant features from the original features set by removing the redundant, irrelevant, or noisy features.
- Below are some benefits of using feature selection in machine learning:
  - **It helps in avoiding the curse of dimensionality.**
  - **It helps in the simplification of the model so that it can be easily interpreted by the researchers.**
  - **It reduces the training time.**
  - **It reduces overfitting hence enhance the generalization.**

Feature selection techniques



Wrapper methods

- In wrapper methodology, selection of features is done by considering it as a search problem, in which different combinations are made, evaluated, and compared with other combinations.
- It trains the algorithm by using the subset of features iteratively.
- Some techniques of wrapper methods are:
  - Forward selection
  - Backward elimination
  - Exhaustive feature selection
  - Recursive feature elimination



Forward selection:

- Forward selection is an iterative process, which begins with an empty set of features.
- After each iteration, it keeps adding on a feature and evaluates the performance to check whether it is improving the performance or not.
- The process continues until the addition of a new variable/feature does not improve the performance of the model.

Backward elimination:

- It is also an iterative approach, which begins the process by considering all the features and removes the least significant feature.
- This elimination process continues until removing the features does not improve the performance of the model.

Exhaustive feature selection:

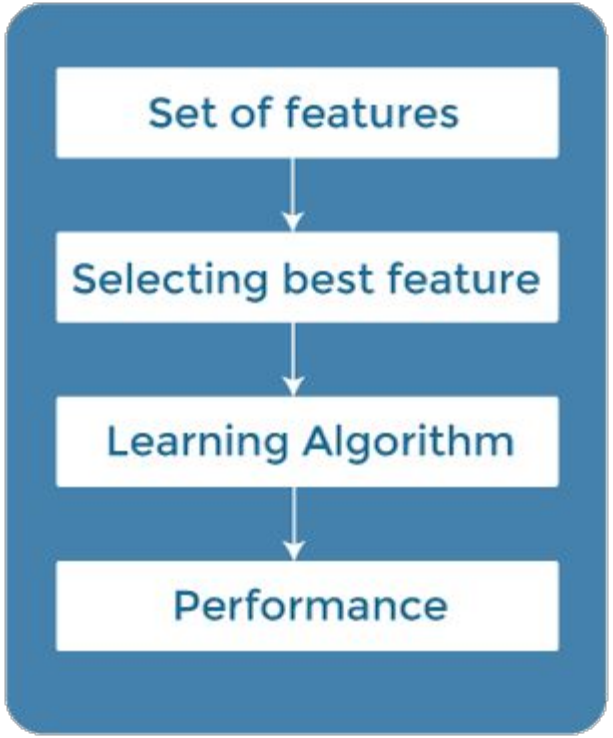
- Exhaustive feature selection is one of the best feature selection methods, which evaluates each feature set as brute-force.
- It means this method tries & make each possible combination of features and return the best performing feature set.

Recursive feature elimination:

- Recursive feature elimination is a recursive greedy optimization approach, where features are selected by recursively taking a smaller and smaller subset of features.
- Now, an estimator is trained with each set of features, and the importance of each feature is determined using *coef_attribute* or through a *feature_importances_attribute.*

Filter methods

- In Filter Method, features are selected on the basis of statistics measures. This method does not depend on the learning algorithm and chooses the features as a pre-processing step.
- The filter method filters out the irrelevant feature and redundant columns from the model by using different metrics through ranking.
- Some techniques of filter methods are:
  - Information Gain
  - Chi-Square Test
  - Fisher's Score
  - Missing Value ratio



Information Gain:

- Information gain determines the reduction in entropy while transforming the dataset.
- It can be used as a feature selection technique by calculating the information gain of each variable with respect to the target variable.

Chi-square Test:

- Chi-square test is a technique to determine the relationship between the categorical variables.
- The chi-square value is calculated between each feature and the target variable, and the desired number of features with the best chi-square value is selected.

Fisher's Score:

- Fisher's score is one of the popular supervised technique of features selection. It returns the rank of the variable on the fisher's criteria in descending order. Then we can select the variables with a large fisher's score.

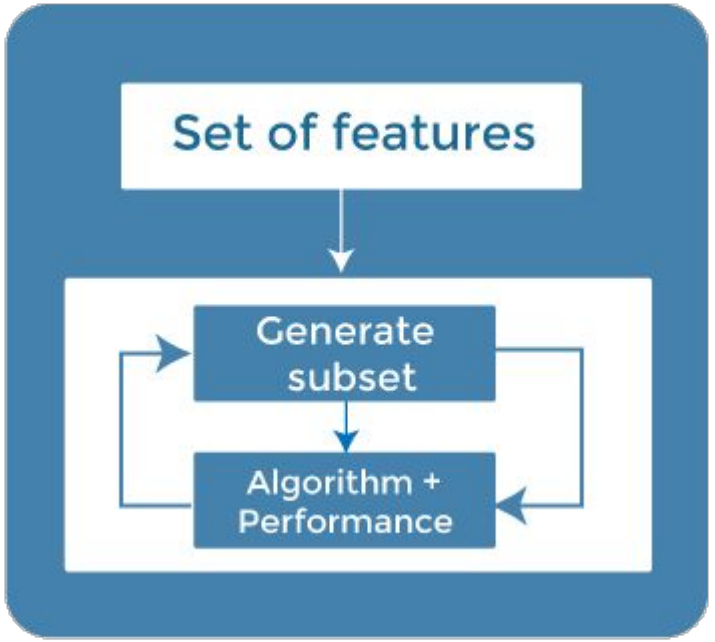
Missing Value Ratio:

- The value of the missing value ratio can be used for evaluating the feature set against the threshold value.. The variable is having more than the threshold value can be dropped.

$$\text{Missing Value Ratio} = \frac{\text{Number of Missing values} * 100}{\text{Total number of observations}}$$

Embedded methods

- Embedded methods combined the advantages of both filter and wrapper methods by considering the interaction of features along with low computational cost.
- These are fast processing methods similar to the filter method but more accurate than the filter method.
- Some of the techniques are:
  - Regularization
  - Random forest importance



Regularization:

- Regularization adds a penalty term to different parameters of the machine learning model for avoiding overfitting in the model.
- This penalty term is added to the coefficients; hence it shrinks some coefficients to zero.
- Those features with zero coefficients can be removed from the dataset.
- The types of regularization techniques are L1 Regularization (Lasso Regularization) or Elastic Nets (L1 and L2 regularization).

- **Random Forest Importance -**
  - Different tree-based methods of feature selection help us with feature importance to provide a way of selecting features.
  - Here, feature importance specifies which feature has more importance in model building or has a great impact on the target variable.
  - Random Forest is such a tree-based method, which is a type of bagging algorithm that aggregates a different number of decision trees.
  - It automatically ranks the nodes by their performance or decrease in the impurity (Gini impurity) over all the trees.
  - Nodes are arranged as per the impurity values, and thus it allows to pruning of trees below a specific node.
  - The remaining nodes create a subset of the most important features.

Filter vs Wrapper vs Embedded methods

Filter methods	Wrapper methods	Embedded methods
Generic set of methods which do not incorporate a **specific machine learning algorithm.**	Evaluates on a **specific machine learning algorithm** to find optimal features.	Embeds (fix) features during **model building process.** Feature selection is done by observing each iteration of model training phase.
Much **faster** compared to Wrapper methods in terms of time complexity	**High computation time** for a dataset with many features	Sits **between Filter methods and Wrapper methods** in terms of time complexity
Less prone to **over-fitting**	High chances of **over-fitting** because it involves training of machine learning models with different combination of features	Generally used to reduce **over-fitting** by **penalizing** the coefficients of a model being too large.
Examples – **Correlation, Chi-Square test, ANOVA, Information gain** etc.	Examples - **Forward Selection, Backward elimination, Stepwise selection** etc.	Examples - **LASSO, Elastic Net, Ridge Regression** etc.