MET CS 688 Feature Engineering III

Leila Ghaedi – Fall 2024

Creator: cemagraphics | Credit: Getty Images/iStockphoto

State of the Art Sentiment Classifiers

- BERT (https://github.com/google-research/bert)
- GPT models
- RoBERTa https://arxiv.org/abs/1907.11692
- InstructGPT (what is used in ChatGPT) and other RLHF based models

Qualitative Analysis

- **Qualitative analysis** method used in many different discipline, such as HCI (Human Computer Interaction), Psychology and Ethnography.

- There are lots of scientific disciplines that favor qualitative analysis (e.g. interview subjects or survey) over quantitative analysis (e.g. statistical and ML methods).

- Some scientist believe a human behavioral studies can not be acceptable without qualitative analysis.

Thematic Analysis

- It is a popular **qualitative analysis** method used in many different discipline.

- **Theme analysis** is the process of identifying patterns across data objects, collected under the umbrella of a central concept, that is important to the understanding of a phenomenon.

- In layman term, themes are **summaries of information** related to a particular **topic**.

Theme Analysis Step by Step

Step 1: Become familiar with the data

Step 2: Generate initial codes

Step 3: Search for themes Step

4: Review themes

Step 5: Define (refine) themes

Step 6: Write-up

Theme Analysis Challenges

- We can not use interview or survey questions as theme. Themes should be extracted solely from answers that participants provide.

keyBERT, RAKE Algorithms (Automatic Theme Extraction)

- There are algorithm that automatically extract keywords from the text, such as Rapid Automatic Keyword Extraction (Rake), keyBERT, etc.

- Rake:

  - Python implementation: https://pypi.org/project/rake-nltk/#:~:text=RAKE%20short%20for%20Rapid%20Automatic,other%20words%20in%20the%20text.

- keyBERT

  - Python implementation: https://pypi.org/project/keybert/

keyBERT, RAKE Algorithms (Automatic Theme Extraction)

- RAKE short for Rapid Automatic Keyword Extraction algorithm, is a domain independent keyword extraction algorithm which tries to determine key phrases in a body of text by analyzing the frequency of word appearance and its co-occurance with other words in the text.
- KeyBERT is a minimal and easy-to-use keyword extraction technique that leverages BERT embeddings to create keywords and keyphrases that are most similar to a document.

Deriving Features from Dates and Time Variables

- Date and time variables are those that contain information about dates, times, or date and time.
- In programming, we refer to these variables as datetime variables. Examples of the datetime variables are date of birth, time of the accident, and date of last payment. The datetime variables usually contain a multitude of different labels corresponding to a specific combination of date and time.
- We do not utilize the datetime variables in their raw format when building machine learning models. Instead, we enrich the dataset by deriving multiple features from these variables.

Some Features Extracted from Dates and Time Variables

- Extracting date and time parts from a datetime variable
- Deriving representations of the year and month
- Creating representations of day and week
- Extracting time parts from a time variable
- Capturing the elapsed time between datetime variables
- Working with time in different time zones

Create a data frame of date time values

- Let's create 100 datetime values, with values beginning from 2021-01-01 at 6:20:00 AM followed by increments of 1 hour.
- Then, let’s capture the value range in a data frame and display the top five rows:

In [1]:
```
import datetime
import pandas as pd
import numpy as np
```

In [2]:
```
# 100 hours starting with 6:20:00 AM Jan 1st, 2021
rng = pd.date_range('1/1/2021 06:20:00',  periods=100, freq='H')
df = pd.DataFrame({'date': rng})
df.head()
```

Out[2]:

	date
0	2021-01-01 06:20:00
1	2021-01-01 07:20:00
2	2021-01-01 08:20:00
3	2021-01-01 09:20:00
4	2021-01-01 10:20:00

datetime type

- The variable is cast as datetime, the default output of pandas' date_range(), as we can see in the following output:

```
In [3]: df.dtypes

Out[3]: date    datetime64[ns]
        dtype: object
```

Capture the Date and Time Part of datetime

- Let's capture the date and time part of the date variable in new features using pandas' dt and then display the top five rows:

```
In [4]: df['date_part'] = df['date'].dt.date
        df['time_part'] = df['date'].dt.time
        df.head()
```

Out[4]:

	date	date_part	time_part
0	2021-01-01 06:20:00	2021-01-01	06:20:00
1	2021-01-01 07:20:00	2021-01-01	07:20:00
2	2021-01-01 08:20:00	2021-01-01	08:20:00
3	2021-01-01 09:20:00	2021-01-01	09:20:00
4	2021-01-01 10:20:00	2021-01-01	10:20:00

Change the data type into datetime

- Let's learn how to change the data type of a variable into a datetime variable and get month, year and quarter of the datetime.

```
In [5]: df_2 = pd.DataFrame({'date_var':['Jan-03-2015', 'Apr-08-2013',
                                         'Jun-09-2014', 'Jan-21-2016',
                                         'Dec-27-2014']})
```

```
In [6]: df_2["datetime_var"] = pd.to_datetime(df_2['date_var'])
        df_2['month'] = df_2['datetime_var'].dt.month
        df_2['year'] = df_2['datetime_var'].dt.year
        df_2['quarter'] = df_2['datetime_var'].dt.quarter
        df_2
```

Out[6]:

	date_var	datetime_var	month	year	quarter
0	Jan-03-2015	2015-01-03	1	2015	1
1	Apr-08-2013	2013-04-08	4	2013	2
2	Jun-09-2014	2014-06-09	6	2014	2
3	Jan-21-2016	2016-01-21	1	2016	1
4	Dec-27-2014	2014-12-27	12	2014	4

Extract week of year, day of week

In [7]:
```
df_2['week'] = df_2['datetime_var'].dt.isocalendar().week
df_2['day_week'] = df_2['datetime_var'].dt.weekday
df_2['is_weekend'] = np.where(df_2['day_week'].isin([5, 6]), 1, 0)
df_2
```

Out[7]:

	date_var	datetime_var	month	year	quarter	week	day_week	is_weekend
0	Jan-03-2015	2015-01-03	1	2015	1	1	5	1
1	Apr-08-2013	2013-04-08	4	2013	2	15	0	0
2	Jun-09-2014	2014-06-09	6	2014	2	24	0	0
3	Jan-21-2016	2016-01-21	1	2016	1	3	3	0
4	Dec-27-2014	2014-12-27	12	2014	4	52	5	1

Extract time parts from datetime

In [8]:
```
rng_ = pd.date_range('2022-03-05', periods=20, freq='3h15min10s')
df_3 = pd.DataFrame({'date': rng_})
```

In [9]:
```
df_3['hour'] = df_3['date'].dt.hour
df_3['min'] = df_3['date'].dt.minute
df_3['sec'] = df_3['date'].dt.second
df_3['is_morning'] = np.where( (df_3['hour'] < 12) & (df_3['hour'] >= 6), 1, 0 )
df_3.head()
```

Out[9]:

	date	hour	min	sec	is_morning
0	2022-03-05 00:00:00	0	0	0	0
1	2022-03-05 03:15:10	3	15	10	0
2	2022-03-05 06:30:20	6	30	20	1
3	2022-03-05 09:45:30	9	45	30	1
4	2022-03-05 13:00:40	13	0	40	0

Elapsed days between two datetimes

In [10]:
```
rng_week = pd.date_range('2023-03-05', periods=20, freq='W')
rng_month = pd.date_range('2023-05-05', periods=20, freq='M')
df_4 = pd.DataFrame({'date1': rng_week, 'date2': rng_month})

df_4['elapsed_days'] = (df_4['date2'] - df_4['date1']).dt.days
df_4.head()
```

Out[10]:

	date1	date2	elapsed_days
0	2023-03-05	2023-05-31	87
1	2023-03-12	2023-06-30	110
2	2023-03-19	2023-07-31	134
3	2023-03-26	2023-08-31	158
4	2023-04-02	2023-09-30	181

Feature Scaling

- Many machine learning algorithms are sensitive to the scale and magnitude of the features.
- In particular, the coefficients of the linear models depend on the scale of the feature, that is, changing the feature scale will change the coefficients' value.
- In linear models, as well as algorithms that depend on distance calculations, such as clustering and principal component analysis, features with bigger value ranges tend to dominate over features with smaller ranges.
- So having features within a similar scale allows us to compare feature importance, also helps algorithms converge faster, improving performance and training times.

Machine learning models that are sensitive to scaling

1. Gradient descent-based algorithms: These include linear regression, logistic regression, and neural networks.
2. Distance-based algorithms: These include K Nearest Neighbors (KNN), K-Means clustering, and Support Vector Machine (SVM).
3. Tree-based algorithms: These include decision trees and tree-based ensemble methods.

Feature Scaling Methods

- Standardizing the features
- Performing mean normalization
- Scaling to the maximum and minimum values
- Implementing maximum absolute scaling
- Scaling with the median and quantiles Scaling to vector unit length

Standardizing the Features

- Standardization is the process of centering the variable at zero and standardizing the variance to 1.
- The transformation is called the z-score and represents how many standard deviations a given observation deviates from the mean.

$$z = \frac{x - mean(x)}{std(x)}$$

Standardizing the Features

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
wine = load_wine()
data = pd.DataFrame(wine.data, columns=wine.feature_names)
data['target'] = wine.target
data.head()
```

Out[2]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86

Standardizing the Features

```
In [3]: data.describe()
```

Out[3]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000

Standardizing the Features

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(
    data.drop('target', axis=1), data['target'], test_size=0.3, random_state=0)
```

In [5]:
```
(pd.DataFrame(X_train,  columns=wine.feature_names)).describe()
```

Out[5]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols
count	124.000000	124.000000	124.000000	124.000000	124.000000	124.000000	124.000000	124.000000
mean	12.983065	2.383710	2.363145	19.525806	100.088710	2.258387	1.969516	0.364274
std	0.801340	1.136696	0.276377	3.579569	14.659281	0.600198	0.977046	0.124235
min	11.030000	0.890000	1.360000	10.600000	70.000000	1.100000	0.470000	0.140000
25%	12.362500	1.607500	2.225000	17.075000	89.000000	1.735000	1.072500	0.270000
50%	13.040000	1.885000	2.360000	19.450000	98.000000	2.200000	2.065000	0.340000
75%	13.640000	3.247500	2.565000	21.700000	106.250000	2.705000	2.767500	0.450000

Standardizing the Features

In [6]:
```
# we'll set up a standard scaler transformer using StandardScaler()
# from scikit-learn and fit it to the train set so that it learns
# each variable's mean and standard deviation:

scaler = StandardScaler()
scaler.fit(X_train)
```

Out[6]:
```
StandardScaler()
```

Standardizing the Features

In [7]:
```
# Now, let's standardize the train and test sets with the trained scaler
# that is, we'll remove each variable's mean and divide the result by the standard deviation:

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

(pd.DataFrame(X_train_scaled,  columns=wine.feature_names)).head()
```

Out[7]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins
0	0.910831	-0.462599	-0.011426	-0.820679	0.062417	0.588204	0.935654	-0.761914	0.130072
1	-0.956099	-0.966087	-1.537254	-0.147487	-0.554019	0.169986	0.072432	0.207913	0.784626
2	0.359522	1.675016	-0.374718	0.133010	1.363782	-1.118128	-1.314889	0.531189	-0.440566
3	0.221695	1.047864	-0.774340	0.413506	0.130910	-1.268687	-1.458759	0.531189	-0.524483
4	1.098777	-0.771758	1.114780	1.535493	-0.964977	1.156982	0.915101	-1.246827	0.432174

Standardizing the Features

In [8]:
```
# StandardScaler() stores the mean and standard deviation that were learned from the train set

scaler.mean_
```

Out[8]:
```
array([1.29830645e+01, 2.38370968e+00, 2.36314516e+00, 1.95258065e+01,
       1.00088710e+02, 2.25838710e+00, 1.96951613e+00, 3.64274194e-01,
       1.61250000e+00, 4.99991935e+00, 9.55854839e-01, 2.60193548e+00,
       7.46766129e+02])
```

In [9]:
```
scaler.scale_
```

Out[9]:
```
array([7.98101757e-01, 1.13210300e+00, 2.75260483e-01, 3.56510578e+00,
       1.46000509e+01, 5.97773270e-01, 9.73098522e-01, 1.23733411e-01,
       5.95825024e-01, 2.34182051e+00, 2.34596176e-01, 7.19337371e-01,
       3.07610251e+02])
```

Mean Normalization

- In mean normalization, we center the variable at zero and rescale the distribution to the value range. This procedure involves subtracting the mean from each observation and then dividing the result by the difference between the minimum and maximum values.
- This transformation results in a distribution centered at 0, with its minimum and maximum values within the range of -1 to 1.

$$xscaled = \frac{x - mean(x)}{max(x) - min(x)}$$

Mean Normalization

In [10]:
```
# Mean Normalization
# let's get the mean values per variable in the train set

means = X_train.mean(axis=0)
means
```

Out[10]:
```
alcohol                         12.983065
malic_acid                       2.383710
ash                              2.363145
alcalinity_of_ash               19.525806
magnesium                      100.088710
total_phenols                    2.258387
flavanoids                       1.969516
nonflavanoid_phenols             0.364274
proanthocyanins                  1.612500
color_intensity                  4.999919
hue                              0.955855
od280/od315_of_diluted_wines     2.601935
proline                        746.766129
dtype: float64
```

Mean Normalization

In [11]:
```
# Mean Normalization
# let's capture the difference between the maximum and
# minimum values per variable in the train set
ranges = X_train.max(axis=0)-X_train.min(axis=0)
ranges
```

Out[11]:
```
alcohol                         3.72
malic_acid                      4.76
ash                             1.86
alcalinity_of_ash              19.40
magnesium                      92.00
total_phenols                   2.78
flavanoids                      3.27
nonflavanoid_phenols            0.52
proanthocyanins                 3.16
color_intensity                10.47
hue                             1.17
od280/od315_of_diluted_wines    2.73
proline                      1235.00
dtype: float64
```

Mean Normalization

In [12]:
```
# Now, we'll implement the mean normalization of the
# train and test sets by utilizing the learned parameters:

X_train_scaled = (X_train - means) / ranges
X_test_scaled = (X_test - means) / ranges
X_train_scaled
```

Out[12]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids
22	0.195413	-0.110023	-0.001691	-0.150815	0.009905	0.126479	0.278435
108	-0.205125	-0.229771	-0.227497	-0.027103	-0.087921	0.036551	0.021555
175	0.077133	0.398380	-0.055454	0.024443	0.216427	-0.240427	-0.391289
145	0.047563	0.249221	-0.114594	0.075989	0.020775	-0.272801	-0.434103
71	0.235735	-0.183552	0.164976	0.282175	-0.153138	0.248782	0.272319
...	...	...	...	...	...	...	...
103	-0.312652	-0.139435	-0.259755	-0.001330	-0.153138	0.086911	-0.100769
67	-0.164802	-0.254981	-0.238250	0.003824	-0.240095	-0.053377	0.009322
117	-0.151361	-0.162544	-0.093089	0.153309	0.085992	-0.092945	0.036845
47	0.246488	-0.147838	-0.130723	-0.181743	0.009905	0.302738	0.434399

Other Scaling Methods

- Scaling to the maximum and minimum values

$$xscaled = \frac{x - min(x)}{max(x) - min(x)}$$

- Maximum absolute scaling

$$xscaled = \frac{x}{max(x)}$$

- Scaling with median and quartiles

$$xscaled = \frac{x - median(x)}{75thQuantile(x) - 25thQuantile(x)}$$

Feature Engineering for Image Data

- Image processing refers to the automatic processing, manipulation, analysis, and interpretation of images using algorithms and codes on a computer.
- Features in image data is a region that we can label, like face of an individual in a picture, cars in the street, legs of chickens, etc.
- The image processing algorithms transform an image into a matrix or tensor and then process it as a numerical data.

Feature Engineering for Image Data

- Image feature engineering is the process of taking raw data and extracting features that are useful for modeling, like color, texture, and shape.

Necessary Python Libraries For Image Processing

- numpy,
- Scipy
- scikit-image
- **opencv-python (**pip install opencv-python)
- Matplotlib
- pip install numpy==1.23.0

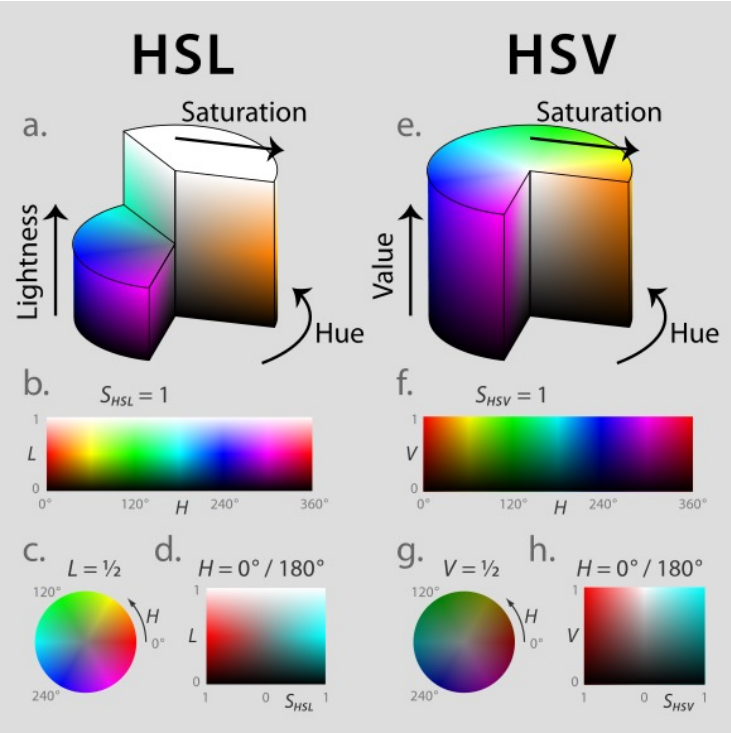
opencv-python

- https://pypi.org/project/opencv-python/
- OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly for real-time computer vision.
- Originally developed by Intel, it was later supported by Willow Garage, then Itseez (which was later acquired by Intel).
- The library is cross-platform and licensed as free and open-source software under Apache License 2.

RGB, BGR, HSV and HSV Color Models

- RGB (Red, Green, Blue) image is simply a composite of three independent grayscale images that correspond to the intensity of red, green, and blue light.
- HSL (for hue, saturation, lightness)
- HSV (for hue, saturation, value)
- HSV and HSL are alternative representations of the RGB color model, designed in the 1970s by computer graphics researchers. In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top.

HSL, HSV Color Models



https://en.wikipedia.org/wiki/HSL_and_HSV

Advantage of HSV and HSL

- HSV and HSL are more robust towards external lighting changes. This means that in cases of minor changes in external lighting (such as pale shadows) Hue values vary relatively lesser than RGB values.
- That is why, HSV and HSL are better color detection/thresholding over RGB/BGR . This means that in cases of minor changes in external lighting (such as pale shadows,etc.) Hue values vary relatively lesser than RGB values.

cv2.cvtColor() method is used to convert an image from one color space to another. There are more than 150 color-space conversion methods available in OpenCV.

In [1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from skimage.util import random_noise

# read the image in BGR format
image = cv2.imread('Baboon.png')
# When the image file is read with the OpenCV function imread(),
# the order of colors is BGR (blue, green, red)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.show()
```

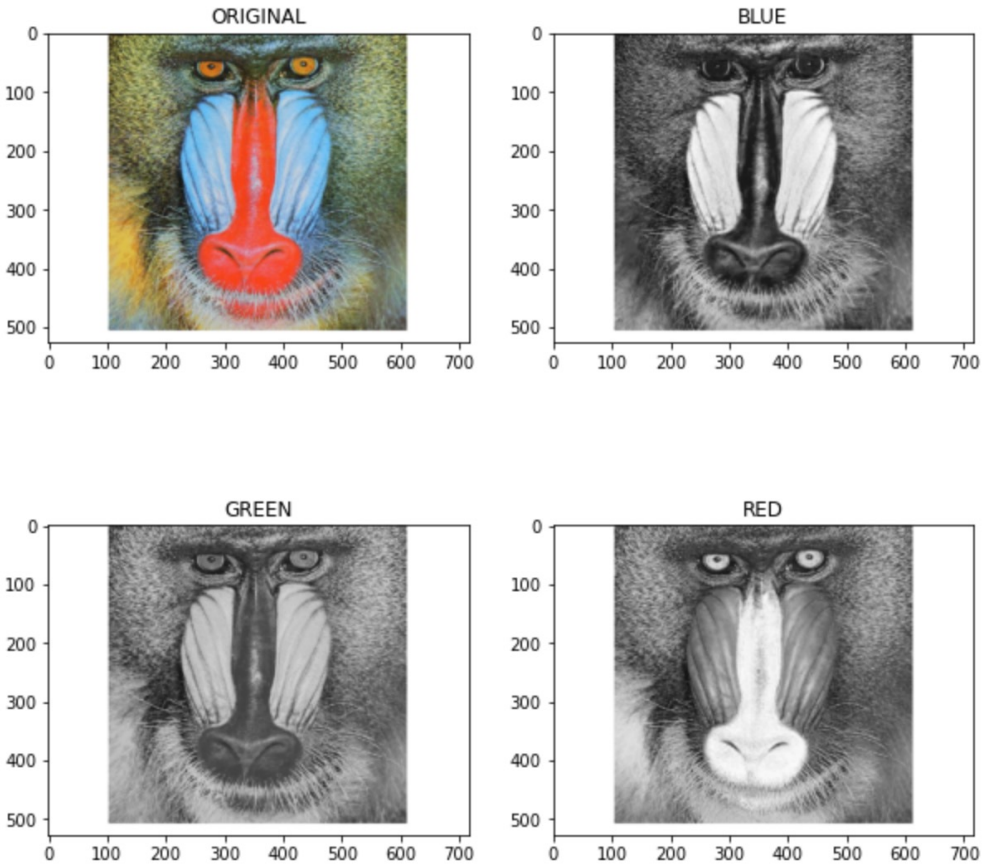
In [2]:
```
print(type(image))
image.shape
```

```
<class 'numpy.ndarray'>
```

Out[2]:
```
(527, 717, 3)
```

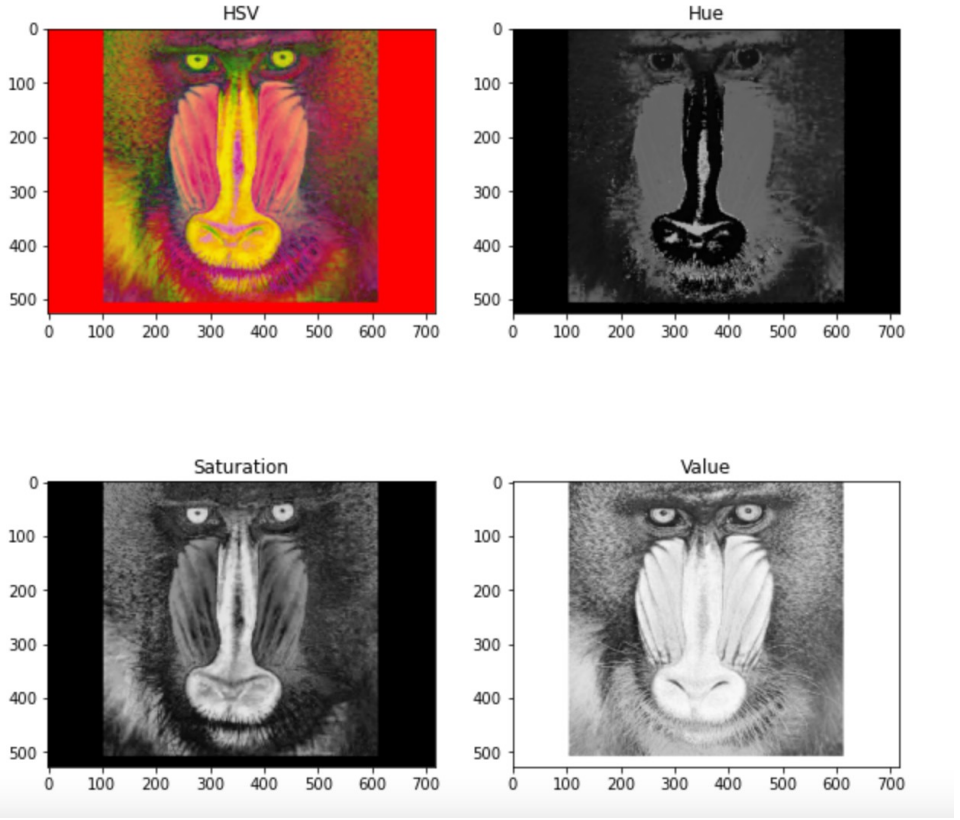
In [3]:

```
fig = plt.figure(figsize=(10, 10))
# setting values to rows and column variables
rows = 2
columns = 2

# Adds a subplot at the 1st position
# Original image
fig.add_subplot(rows, columns, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("ORIGINAL")

# Adds a subplot at the 2nd position
# RED
fig.add_subplot(rows, columns, 2)
plt.imshow(cv2.cvtColor(image[:, :, 0], cv2.COLOR_BGR2RGB))
plt.title("BLUE")

# Adds a subplot at the 3rd position
# GREEN
fig.add_subplot(rows, columns, 3)
plt.imshow(cv2.cvtColor(image[:, :, 1], cv2.COLOR_BGR2RGB))
plt.title("GREEN")

# Adds a subplot at the 4th position
# BLUE
fig.add_subplot(rows, columns, 4)
plt.imshow(cv2.cvtColor(image[:, :, 2], cv2.COLOR_BGR2RGB))
plt.title("RED")
```


ORIGINAL
BLUE
GREEN
RED
0
100
200
300
400
500
600
700

In [4]:

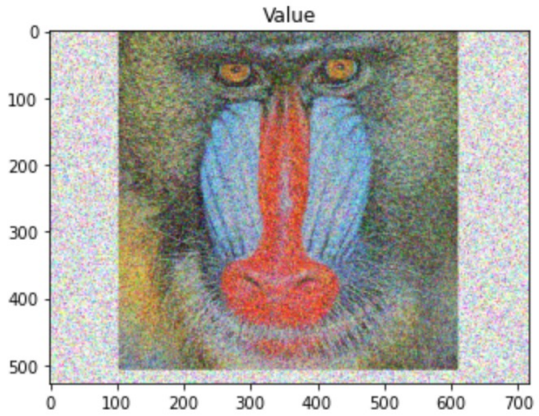
```
# Convert BGR to HSV (Hue, Saturation, Value)
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

fig = plt.figure(figsize=(10, 10))
# setting values to rows and column variables
rows = 2
columns = 2

# Original HSV Image
fig.add_subplot(rows, columns, 1)
plt.imshow(cv2.cvtColor(hsv_image, cv2.COLOR_BGR2RGB))
plt.title("HSV")

# Hue
fig.add_subplot(rows, columns, 2)
plt.imshow(cv2.cvtColor(hsv_image[:, :, 0], cv2.COLOR_BGR2RGB))
plt.title("Hue")

# Saturation
fig.add_subplot(rows, columns, 3)
plt.imshow(cv2.cvtColor(hsv_image[:, :, 1], cv2.COLOR_BGR2RGB))
plt.title("Saturation")

# Lightness
fig.add_subplot(rows, columns, 4)
plt.imshow(cv2.cvtColor(hsv_image[:, :, 2], cv2.COLOR_BGR2RGB))
plt.title("Value")
```


HSV
Hue
Saturation
Value
0
100
200
300
400
500
600
700

Add Salt and Pepper Noise

In [5]:
```
# Add salt and pepper noise to the image.
noise_image = random_noise(image, mode='s&p',amount=0.3)

# The above function returns a floating-point image
# on the range [0, 1], thus we changed it to 'uint8'
# and from [0,255]
noise_image = np.array(255*noise_image, dtype = 'uint8')

# display the noise image
plt.imshow(cv2.cvtColor(noise_image, cv2.COLOR_BGR2RGB))
plt.title("Value")
```

Out[5]:
```
Text(0.5, 1.0, 'Value')
```

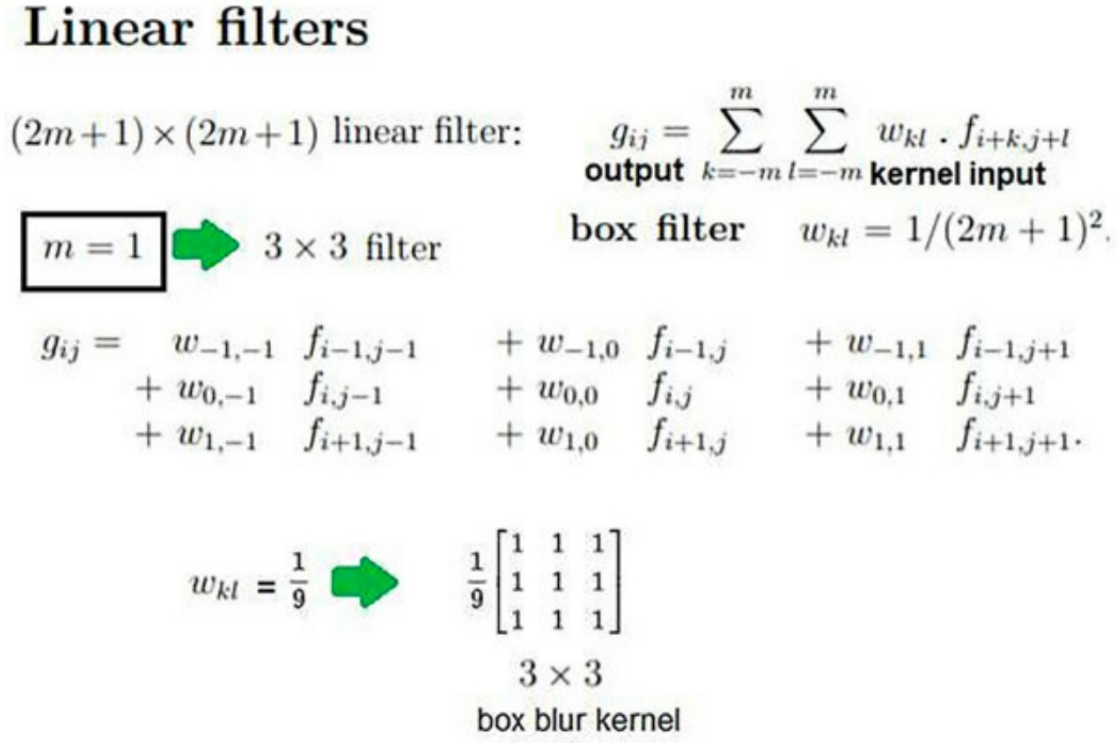


2D Convolution – Image Smoothing (Blurring)

- Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It removes high frequency content (eg: noise, edges) from the image.
- Why? Removing salt and pepper noise
- Methods:
  - Averaging
    - blur = cv.blur(img,(5,5))
  - Median
    - Blur=cv2.medianBlur(img,(5,5))
  - Gaussian Blurring
    - blur = cv.GaussianBlur(img,(5,5),0)

Averaging Blur

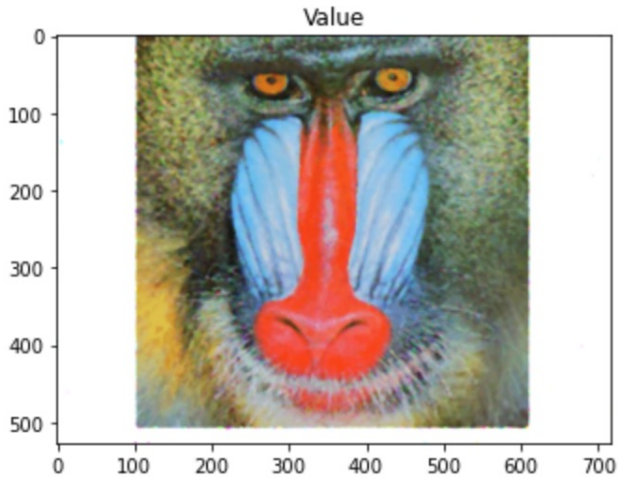
- The moving average or box filter is the simplest of all filters. It replaces each pixel by the average of pixel values in a square centered at that pixel. The following figure defines the linear filters generically and then shows the box filter as a special case:



Remove salt and pepper noise by applying median Blur filter

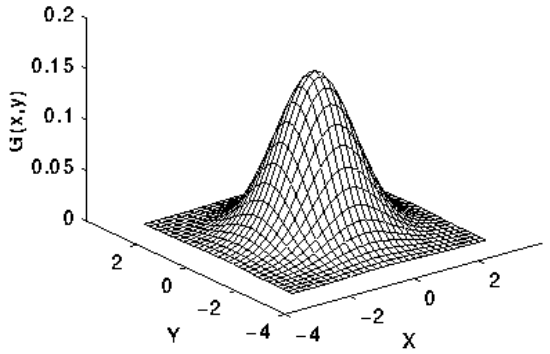
In [6]:
```
# Blur the noisy image to remove salt and
# salt and pepper noise
median_blur= cv2.medianBlur(noise_image, 5)
cv2.imshow('median_blur', median_blur)
# display the noise image
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.title("Value")
```

Out[6]:
```
Text(0.5, 1.0, 'Value')
```



Smoothing with Gaussian Filter

- The averaging filter is simple but has two drawbacks:
  - It's not isotopic (circularly symmetric) which smooths further along diagonals than along rows and columns.
  - Weights have abrupt cut-off which causes discontinuities in the smoothed image.
- Gaussian filter is another smoothing method, which uses a bell shape kernel (window). The weights from the center to the edges of the kernel also decay to zero.



Gaussian Filter Kernel

Gaussian filter $w_{kl} = \frac{1}{2\pi\sigma^2} \exp\left\{\frac{-(k^2+l^2)}{2\sigma^2}\right\}$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

3×3 Gaussian blur kernel

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

5 X 5 Gaussian blur kernel

Gradient in Image Processing

- Gradients are calculated by taking the partial derivatives of an image function in the horizontal and vertical directions. The magnitude of the gradient tells us how quickly the image is changing, while the direction of the gradient tells us the direction in which the image is changing most rapidly.
- The formula for calculating the magnitude and direction of the gradient is:
  G=sqrt(dx^2+dy^2)
  theta=arctan(dy/dx)

Sobel Edge Detection

- Sobel Kernels (Gx, Gy)

-1	0	1
-2	0	2
-1	0	1

Vertical

1	2	1
0	0	0
-1	-2	-1

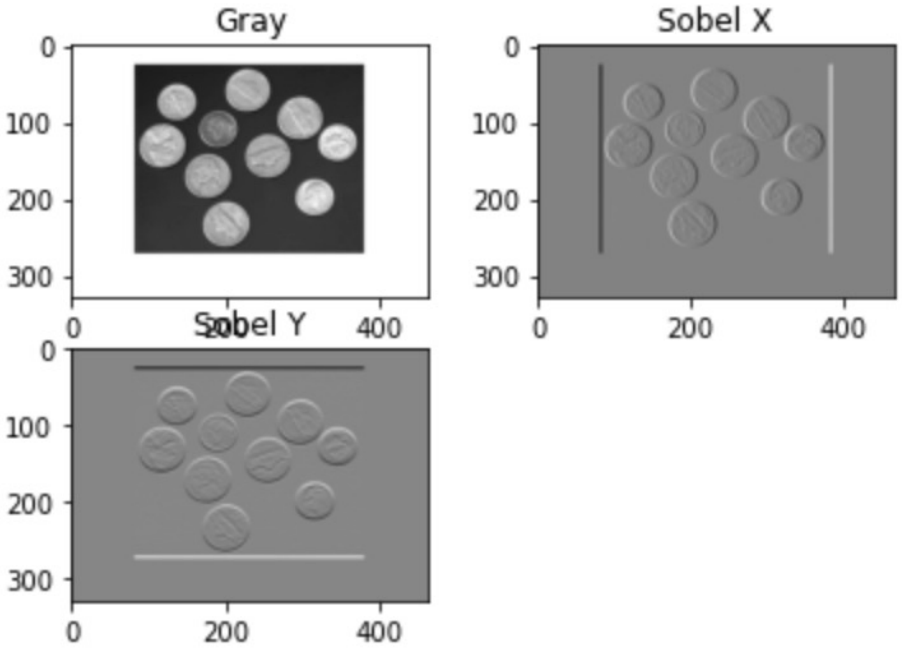
Horizontal

- As you are seeing in the above picture, the edges corresponds to the derivatives. Since images are discrete in nature, we can easily take the derivate of an image using 2D derivative mask.

Sobel Edge Detection

In [7]:
```
# Calculation of Sobelx
coins = cv2.imread('Coins.png')
gray_coins = cv2.cvtColor(coins, cv2.COLOR_RGB2GRAY)

# Calculation of Sobelx
sobelx = cv2.Sobel(gray_coins,cv2.CV_64F,1,0,ksize=5)

# Calculation of Sobely
sobely = cv2.Sobel(gray_coins,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(gray_coins,cmap = 'gray')
plt.title('Gray'),
plt.subplot(2,2,2),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'),
plt.subplot(2,2,3),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'),
```

Sobel Edge Detection



Canny Edge Detection

- Edge detection is one of feature extraction methods.
- The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by Jon F. Canny in 1986.
- Canny(img, threshold1, threshold2, apertureSize = 3, L2gradient = false) Finds edges in an image using the Canny algorithm.

Canny Edge Detection Steps

- Noise reduction using Gaussian filter.
  - Noise (sudden intensity changes) can be assumed as edges, so it's important to remove the noise by applying Gaussian filter before doing the edge detection.
- Gradient calculation along the horizontal and vertical axis.
  - Gradient is a two-dimensional vector that describes how a function changes in multiple dimensions. The gradient is made up of partial derivatives for each variable.
- Double thresholding for segregating strong and weak edges.
  - The gradient magnitudes are compared with two specified threshold values, the first one is lower than the second. The gradients that are smaller than the low threshold value are suppressed, the gradients higher than the high threshold value are marked as strong ones and the corresponding pixels are included in the final edge map. All the rest gradients are marked as weak ones and pixels corresponding to these gradients are considered in the next step.

Canny Edge Detection

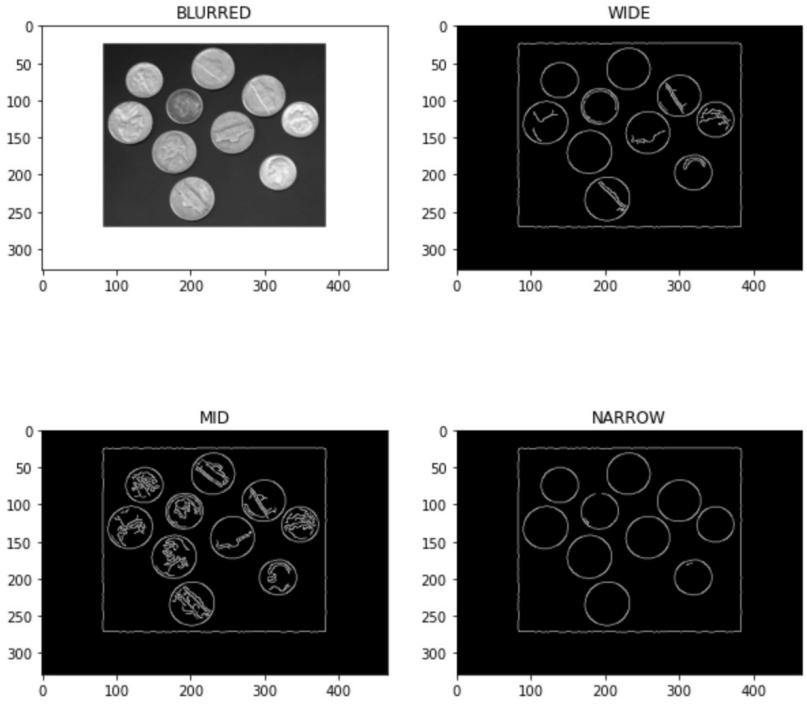
In [8]:
```
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
blurred = cv2.GaussianBlur(gray_coins, (5, 5), 0)
wide= cv2.Canny(blurred, 50, 200)
mid= cv2.Canny(blurred, 30, 150)
narrow= cv2.Canny(blurred, 210, 250)
```

Canny Edge Detection

In [9]:

```
fig = plt.figure(figsize=(10, 10))
# setting values to rows and column variables
rows = 2
columns = 2

#  gray blurred image
fig.add_subplot(rows, columns, 1)
plt.imshow(gray_coins, cmap='gray')
plt.title("BLURRED")

# Wide Canny Filter
fig.add_subplot(rows, columns, 2)
plt.imshow(cv2.cvtColor(wide, cv2.COLOR_BGR2RGB))
plt.title("WIDE")

# Mid Canny Filter
fig.add_subplot(rows, columns, 3)
plt.imshow(cv2.cvtColor(mid, cv2.COLOR_BGR2RGB))
plt.title("MID")

# Narrow Canny Filter
fig.add_subplot(rows, columns, 4)
plt.imshow(cv2.cvtColor(narrow, cv2.COLOR_BGR2RGB))
plt.title("NARROW")
```

Detected Edges Using Canny



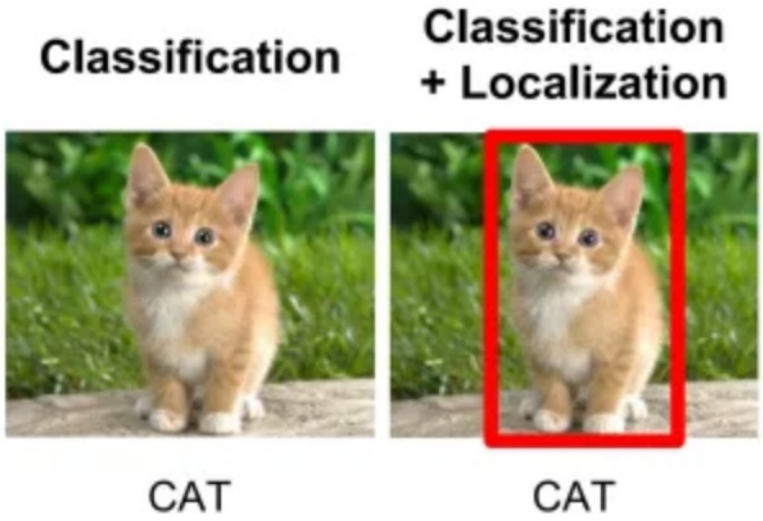
Advance Image Feature Extraction

- Since 2012, deep learning revolutionized image and video processing and their methods for extracting features. There are features are learnt by advanced computer vision algorithms and this section briefly lists those features.

- Objection Detection: The process of identifying an object inside image is referred as object detection. Image classification algorithms are responsible to detect an object in the image.

- Object Localization: Specifying the location of the object inside the image is referred as localization.

Object Localization



Semantic Segmentation vs Instance Segmentation

Semantic segmentation

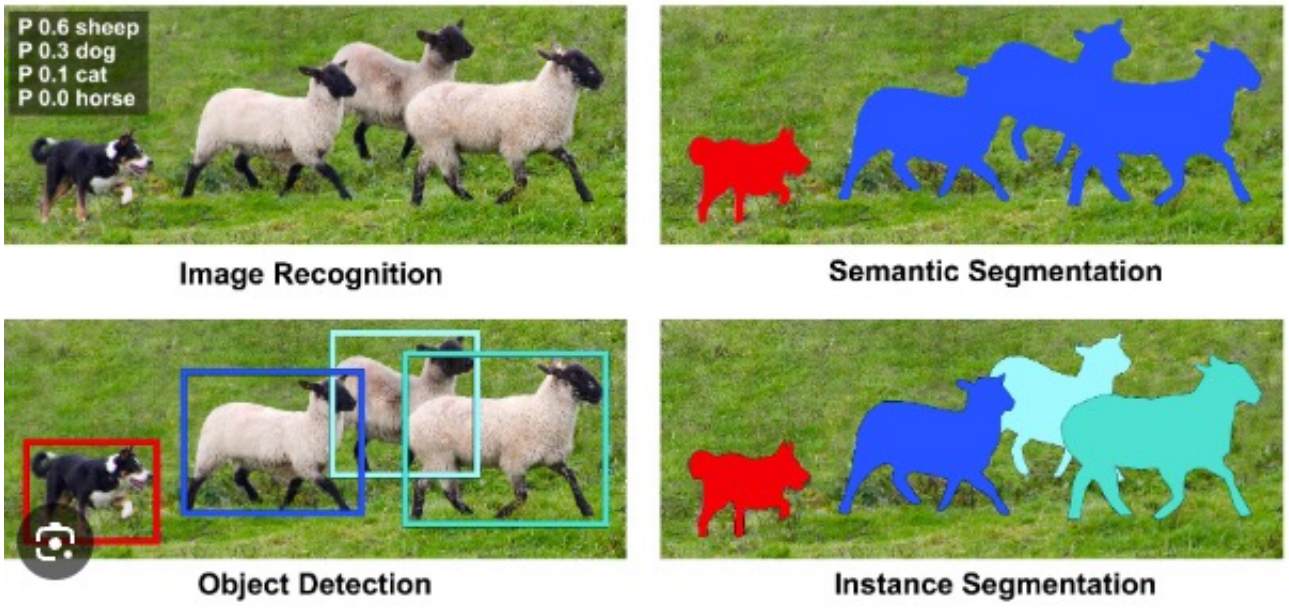
Localization is not accurately specifying the boundaries of an object inside the image. One of the major tasks in computer vision is extracting objects from an image and separating segmenting image in meaningful components. For example, a self-driving car segments a street into different component including pedestrians, signs, street, other cars. Semantic segmentation is the process of separating objects inside an image and identifying the shape of each object. In more technical sense, image segmentation partitions an image on pixel level to accurately identify boundaries for each object inside the image, see the right side of the example.

Instance segmentation

When there are more than one object of the same type existed in the image the instance segmentation enables the semantic segmentation process to recognize every pixel belongs to which segment (of the same type). For example, an image that has three human in it, the instance segmentation recognize each individual person from each other.

Semantic Segmentation vs Instance Segmentation

- Semantic segmentation cannot distinguish between different instances in the same category.



https://manipulation.csail.mit.edu/segmentation.html

Time Series Feature Extraction

- A time series dataset is a sequence of data points or observations gathered at either regular or varying time intervals. Typically, it involves a consecutive set of data points captured at uniform time gaps, which could occur hourly, daily, weekly, monthly, quarterly, or annually.
- Conventional machine learning algorithms might not be the most appropriate choice for analyzing unprocessed time series data since they struggle to grasp the strong interdependence between successive time points. Treating individual time points as features might not yield the best results in such cases.

Time Series Feature Extraction

- For time series data, feature extraction can be done using different time series analysis and decomposition techniques.
- Time Series Feature Extraction Library (TSFEL) is a python package for feature extraction on time series data.

Pip install tsfel

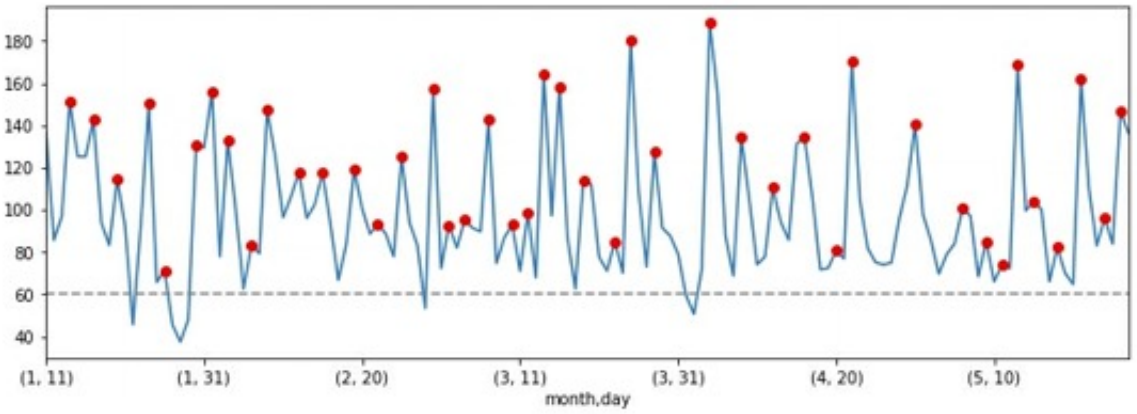
https://tsfel.readthedocs.io/en/latest/descriptions/get_started.html#overview

TSFEL List of Features

- https://tsfel.readthedocs.io/en/latest/descriptions/feature_list.html

Time Series Feature Extraction

1. Aggregation Methods such as maximum, minimum, average, ... That translates his historical data into one single data point.
2. Window Based Aggregation Methods: creates an aggregate measure over a window of time
3. Determining the number of local maxima and minima (Trend)



Time Series Feature Extraction

4- Fourier Transforms: Decomposition of the time series into its constituent frequencies. (Periodic, Seasonality)
5- Wavelet transforms: Analysis of time series data in both time and frequency domains.
6-Autoregression: Modelling the relationship between the time series and lagged versions of itself.
7- Seasonality decomposition: Separation of the time series into trend, seasonal, and residual components.

References

Galli, Soledad. Python Feature Engineering Cookbook: Over 70 recipes for creating, engineering, and transforming features to build machine learning models Packt Publishing. Kindle Edition.

https://www.tensorflow.org/text/guide/word_embeddings

"Thumbs Up? Sentiment Classification using Machine Learning Techniques" by Pang and Lee, published in 2002.