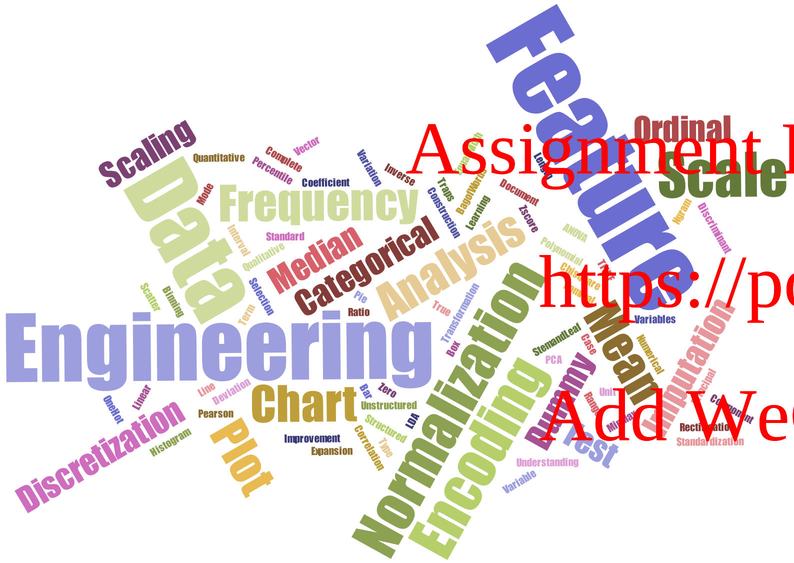
Feature Engineering (Concepts – Part 2)

December 2020



Contents

- Feature Improvement
- Feature Construction

Feature Improvement

Data Probability Distribution

Variable Transformations

- Linear and logistic regression assume that the variables are normally distributed
- If they are not, a mathematical transformation can be applied to change them into normal distribution, and sometimes even unmask linear relationships between variables and their targets
- Transforming variables may improve the performance of linear ML models
- Commonly used mathematical transformations include
  - Logarithm, Reciprocal, Square Root, Cube Root, Power, Box-Cox and Yeo-Johnson

Variable Distribution

- A probability distribution is a function that describes the likelihood of obtaining the possible values of a variable
- There are many well-described variable distributions
  - Normal distribution for continuous variables
  - Binomial distribution for discrete variables
  - Poisson distribution for discrete variables
- A better spread of values may improve model performance

The Boston Housing Dataset

Index	Variable	Definition
0	AGE	proportion of owner-occupied units built prior to 1940
1	B	1000*(Bk-0.63)^2, Bk is the proportion of blacks by town
2	CHAS	Charles River dummy variable (1 if tract bounds river; 0 otherwise)
3	CRIM	per capita crime rate by town
4	DIS	weighted distances to five Boston employment centres
5	INDUS	proportion of non-retail business acres per town
6	LSTAT	% lower status of the population
7	NOX	nitric oxides concentration (parts per 10 million)
8	PTRATIO	pupil-teacher ratio by town
9	RAD	index of accessibility to radial highways
10	RM	average number of rooms per dwelling
11	TAX	full-value property-tax rate per US$10,000
12	ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
13		

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA.

Source: https://www.kaggle.com/prasadperera/the-boston-housing-dataset

Python: Examining Variable Distribution (1)

```
# load the relevant packages
import pandas as pd
import matplotlib.pyplot as plt
```

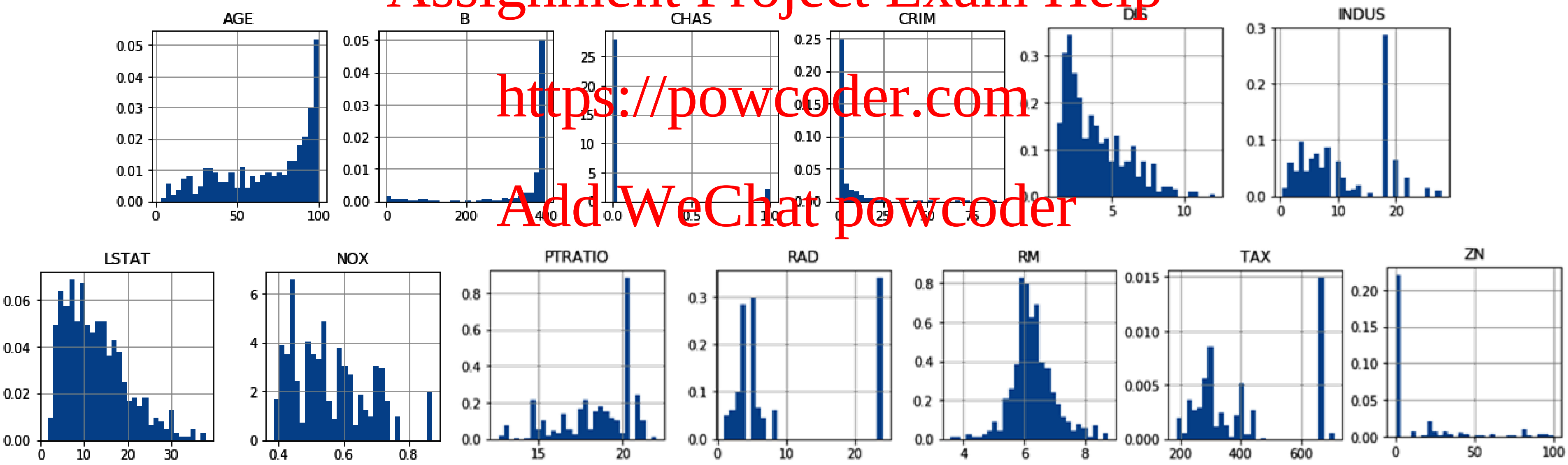
```
# load the Boston House Prices dataset from scikit-learn
from sklearn.datasets import load_boston

data = load_boston()
data = pd.DataFrame(data.data, columns=data.feature_names)
```

Python: Examining Variable Distribution (2)

visualize the variable distribution with histograms

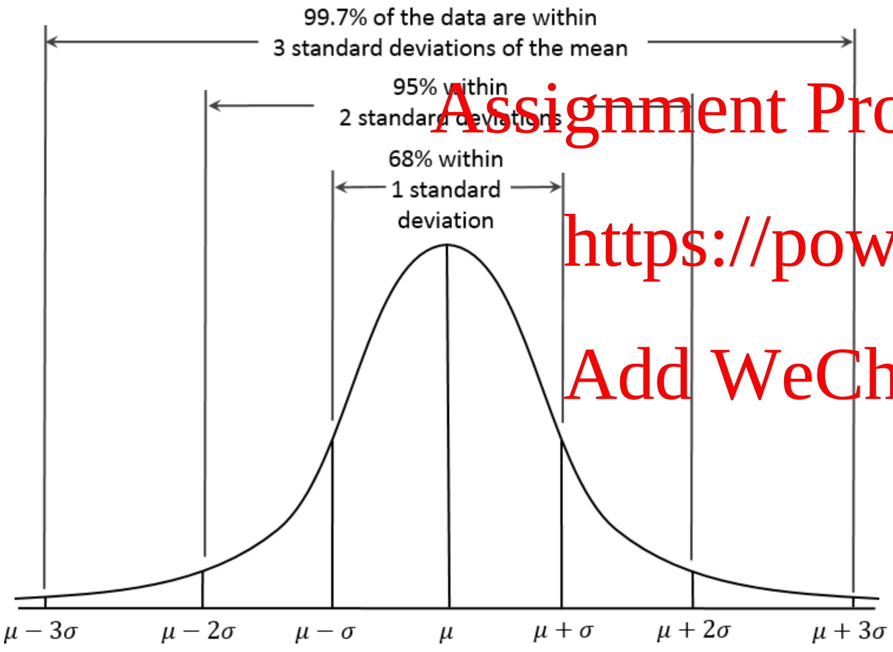
```
data.hist(bins = 30, figsize = (12,12), density = True)
plt.show()
```

Normal Distribution

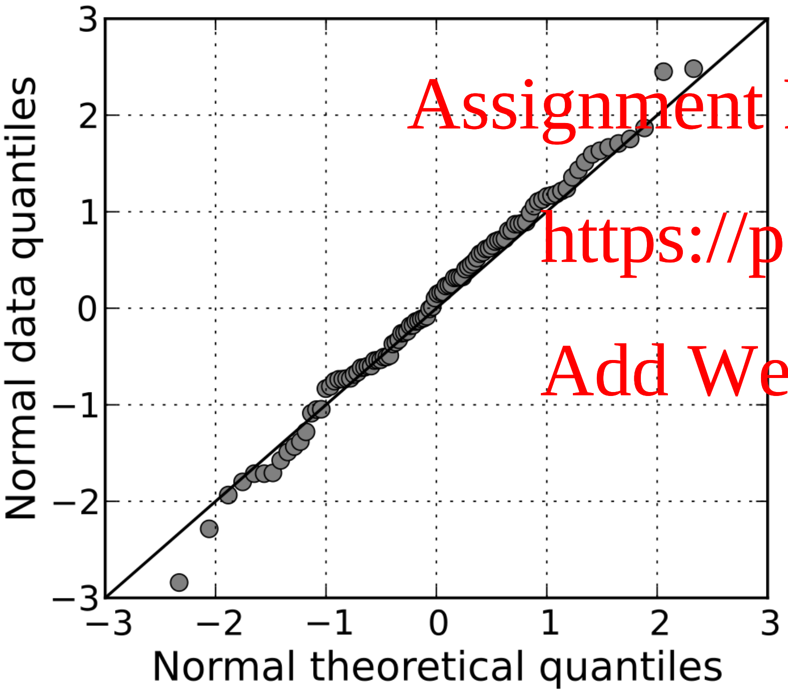
- Linear models assume that the independent variables are normally distributed
- Failure to meet this assumption may produce algorithms that perform poorly
- To check for normal distribution, use histograms and Q-Q plots
  - In a Q-Q plot, the quantiles of the independent variable are plotted against the expected quantiles of the normal distribution
  - If the variable is normally distributed, the dots in the Q-Q plot should fall along a 45 degree diagonal

Most raw data as a whole are not normally distributed normal



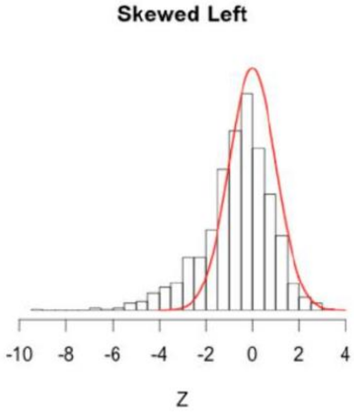
- Normal / Gaussian distribution is a probability distribution that is symmetric about the mean
  - Data near the mean are more frequent in occurrence than data far from the mean - a bell curve
  - The mean, median & mode are all equal
- A common misconception that most data follows a normal distribution (i.e. it is the normal thing)
  - Many statistics are normally distributed in their sampling distribution
  - But errors, averages, and totals often are
- Assumptions of normality are generally a last resort
  - Used when empirical probability distributions are not available

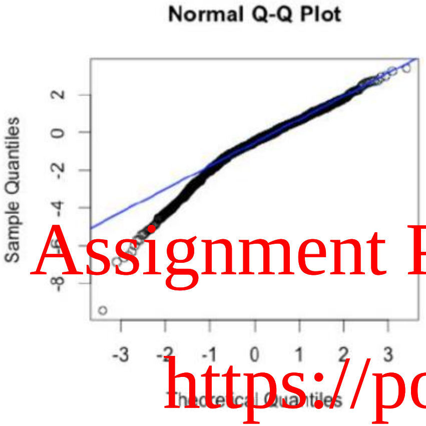
Q-Q plots help to find the type of distribution for a random variable, typically if it is a normal distribution

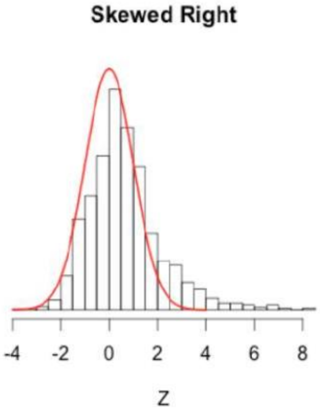


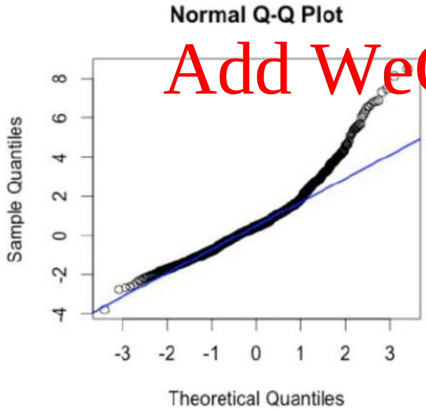
- A Q-Q (Quantile-Quantile) Plot plots the quantiles of two probability distributions against each other
  - Quantiles are cut points dividing the range of a probability distribution into continuous intervals with equal probabilities
- QQ Plots are used to graphically analyze and compare two probability distributions to see if they are exactly equal
  - If the two distributions are exactly equal, the points on the Q-Q Plot will perfectly lie on the straight line y = x

Skewed Q-Q Plots



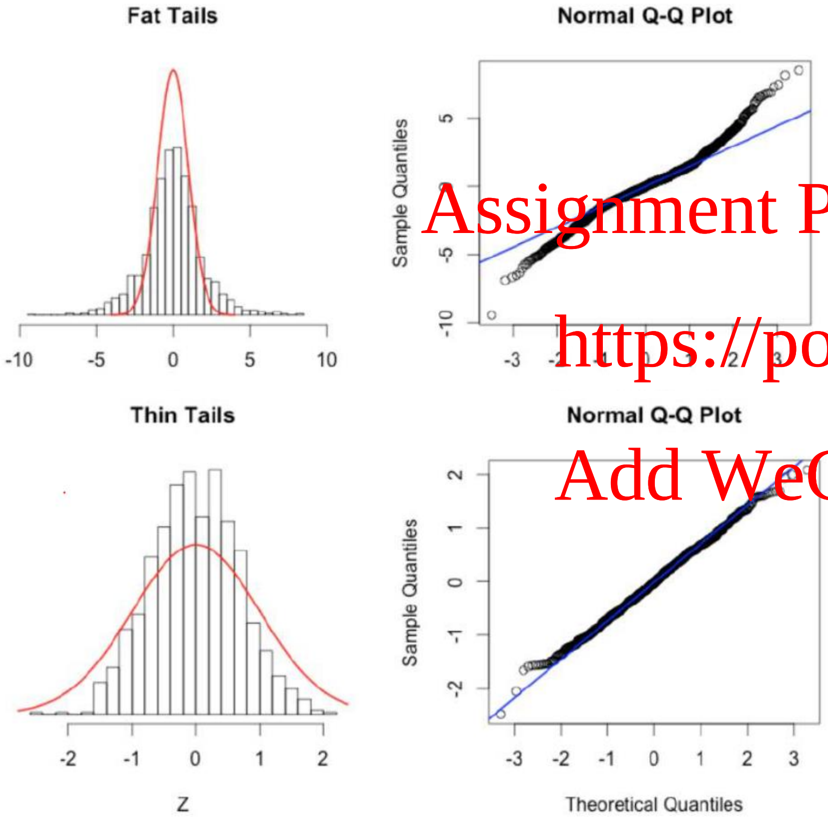




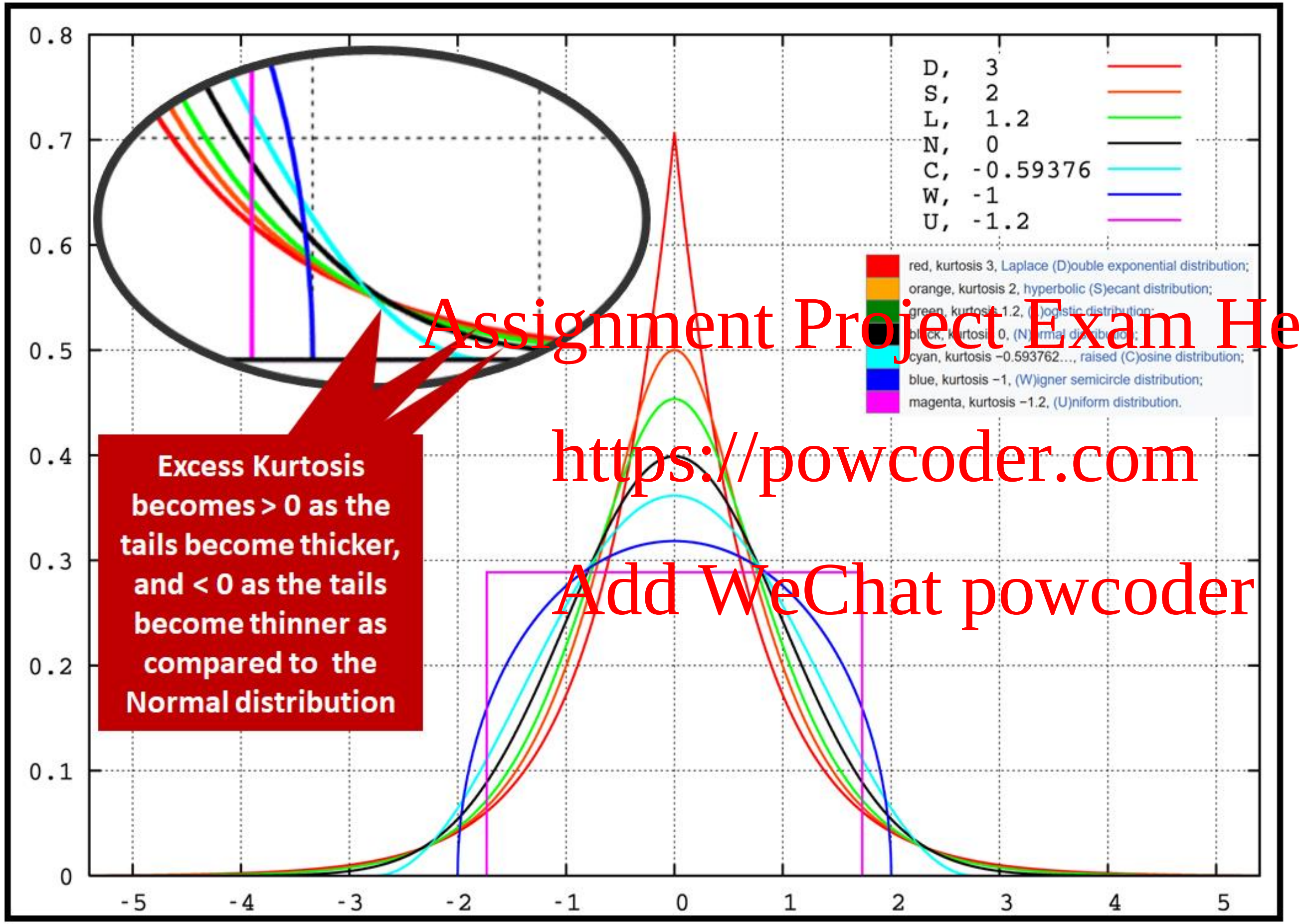


- Q-Q plots can find the skewness (a measure of asymmetry) of a distribution
- If the bottom end deviates from the straight line but the upper end does not, the distribution has a longer tail to its left
  - left-skewed or negatively skewed
- If the upper end deviates from the straight line and the lower end follows the straight line, the distribution has a longer tail to its right
  - right-skewed or positively skewed

Tailed Q-Q Plots



- Q-Q plots can find the Kurtosis (a measure of tailedness) of a distribution
- A distribution with a fat tail will have both ends of the plot deviating from the straight line and its centre following the straight line
- A thin-tailed distribution will form a Q-Q plot with a less or negligible deviation at both ends of the plot
  - a perfect fit for the Normal Distribution



- Kurtosis measures how heavily the tails differ from a normal distribution
  - It identifies whether the tails of a distribution contain extreme values
- In finance, it is used as a measure of financial risk
  - A large kurtosis is associated with a high level of risk
  - A small kurtosis signals a moderate level of risk because the probabilities of extreme returns are relatively low

Python: Identifying Normal Distribution (1)

```
# load the relevant packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```



```
# generate an array containing 200 observations that are normally distributed
np.random.seed(29)
x = np.random.randn(200)
```

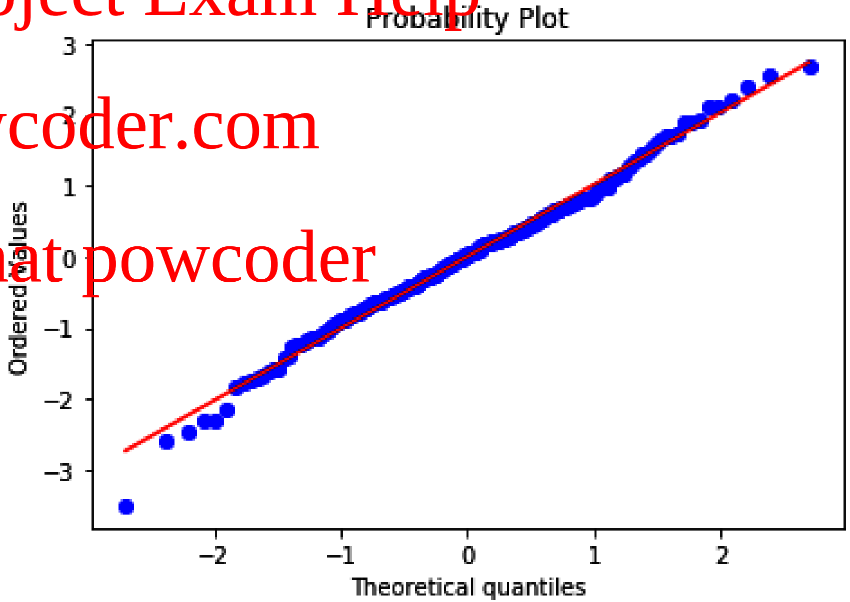


```
# create a dataframe after transposing the generated array
data = pd.DataFrame([x]).T
data.columns = [‘x’]
```

Python: Identifying Normal Distribution (2)

display a Q-Q plot to assess a normal distribution

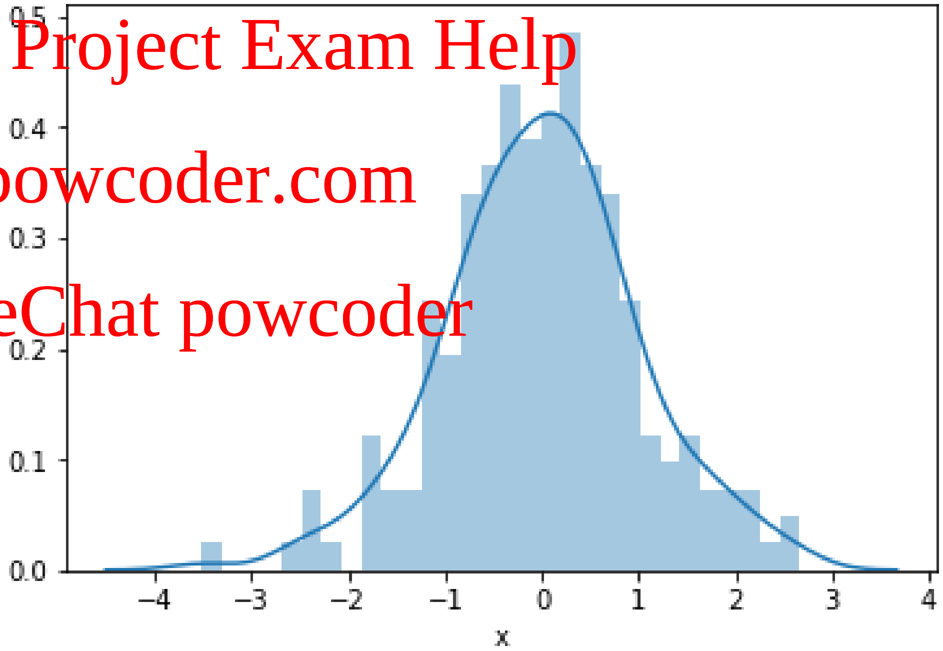
```
stats.probplot(data['x'], dist = "norm", plot = plt)
plt.show()
```

Python: Identifying Normal Distribution (3)

make a histogram and a density plot of the variable distribution

```
sns.distplot(data['x'], bins = 30)
```



Data Normalization

Normalization ensures that all rows and columns are treated equally under the eyes of machine learning

- Many ML algorithms are sensitive to the scale and magnitude of the features
  - linear models (e.g., clustering, principal component analysis) involving distance calculation are particularly sensitive to these
  - features with bigger value ranges tend to dominate over features with smaller ranges
- Normalization is applicable to numerical variables and will align/transform both columns and rows so as to satisfy a consistent set of rules
  - e.g., to transform all quantitative columns to a value range between 0 and 1
  - e.g., to make all columns having the same mean and standard deviation so that all variable values appear nicely on the same histogram
- Normalization is meant to level the playing field of data by ensuring that all rows and columns are treated equally under the eyes of machine learning

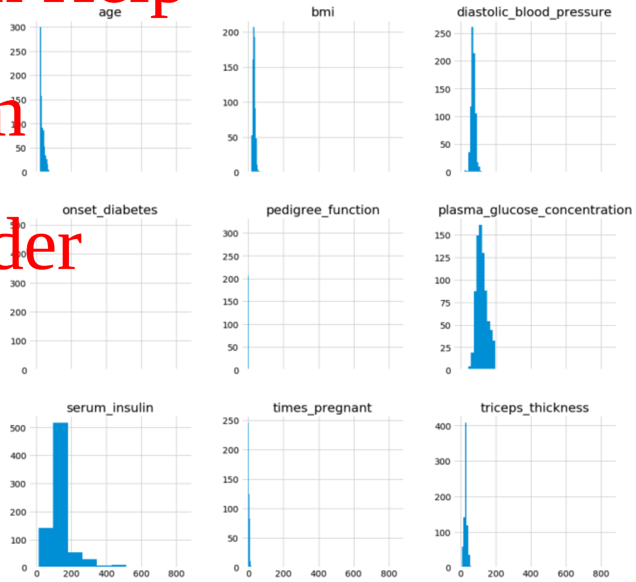
Some ML algorithms are affected greatly by data scales and diversity of scales might result in suboptimal learning

```
# use the Boston Housing dataset
# make a histogram for each variable
```

```
data.hist(figsize=(15,15))
```


```
# redraw the histograms
# use one and the same scale for the X-axis
```

```
data.hist(figsize=(15,15), sharex=True)
```



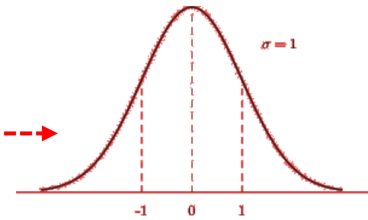
Column values can be normalized so that different columns will have similar data value distribution

Name	Amount	Date	Issued In	Used In	Age	Education	Fraud?
Daniel	$2,600.45	1-Jul-2020	HK	HK	22	Secondary	No
Alex	$2,294.58	1-Oct-2020	HK	RUS	None	Postgraduate	Yes
Adrian	$1,003.30	3-Oct-2020	HK		25	Graduate	Yes
Vicky	$8,488.32	4-Oct-2020	JAPAN	HK	64	Graduate	No
Adams	¥20000	7-Oct-2020	AUS	JAP	58	Primary	No
...	...	...	...	...	...	...	...
Jones	₱3,250.11	Nov 1, 2020	HK	RUS	43	Graduate	No
Mary	₱8,156.20	Nov 1, 2020	HK	N/A	27	Graduate	Yes
Max	€7475,11	Nov 8, 2020	UK	GER	32	Primary	No
Peter	₽500.00	Nov 9, 2020	Hong Kong	RUS	0	Postgraduate	No
Anson	₽7,475.11	Nov 9, 2020	Hong Kong	RUS	20	Postgraduate	Yes

Observations

Feature

Target



Standardization / Z-score Normalization

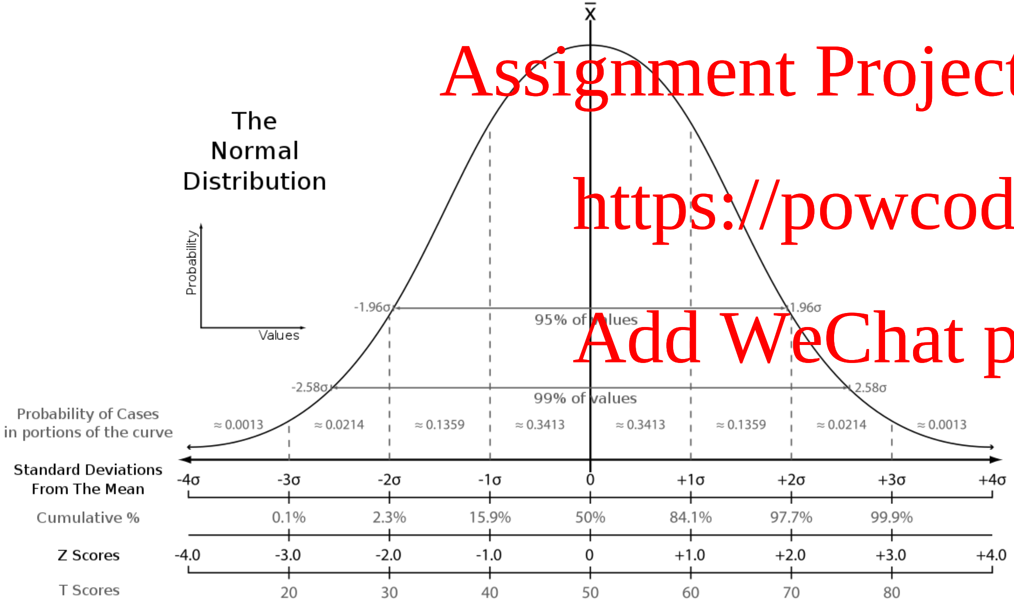
- Standardization is the process of centering the variable at 0 and standardizing the variance (square of standard deviation) to 1
- To standardize features, we subtract the mean from each observation and then divide the result by the standard deviation

$$z = \frac{x - mean(X)}{standard\_deviation(X)}$$

- The z-score represents how many standard deviations a given observation deviates from the mean

Z-score provides a standard scale to compare data having different means & standard deviations

- The standard score or *Z*-score is the number of standard deviations by which a data point is above or below the mean of the population
  - Scores above the mean have positive standard scores, while those below the mean have negative standard scores



$$Z - Score = \frac{data\ point - population\ mean}{population\ standard\ deviation}$$

- This process of converting a data point into a standard score is called standardizing or normalizing

Mean Normalization

- Center the variable mean 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 maximum and minimum values

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

- This transformation results in a distribution centered at 0, with its minimum and maximum values within the range of -1 to 1

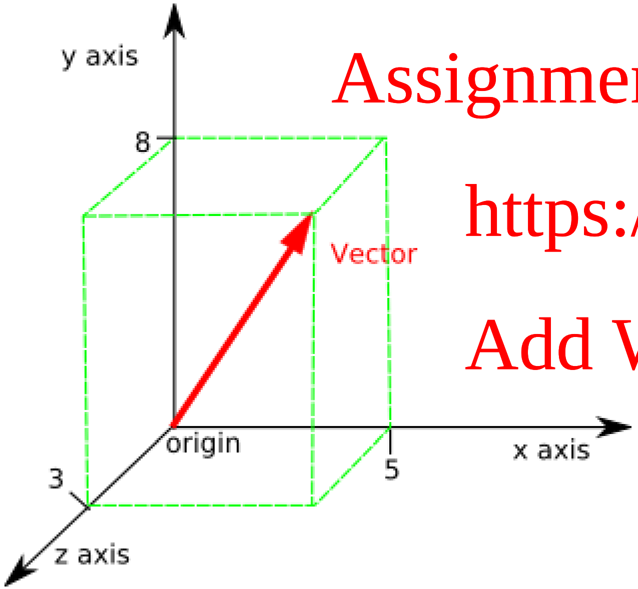
Min-Max Normalization

- Scaling to the minimum and maximum values squeezes the values of the variables between 0 and 1

- To implement this scaling technique, we need to subtract the minimum value from all the observations and divide the result by the value range, that is, the difference between the maximum and minimum values

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

An observation can be represented as a vector in a multi-dimensional vector space



- Each column value can be considered a scalar value that can be captured using one dimension in a multi-dimensional space
- An observation can therefore be captured as a feature vector
- The direction and magnitude of the feature vector is dictated by the value along each dimension, i.e. the feature values
- The angle between the vectors indicates similarity between them (e.g., cosine similarity)

Scaling Feature Vector to Unit Vector

- Scales the feature vector, as opposed to each individual variable
  - A feature vector contains the values of several variables for a single observation

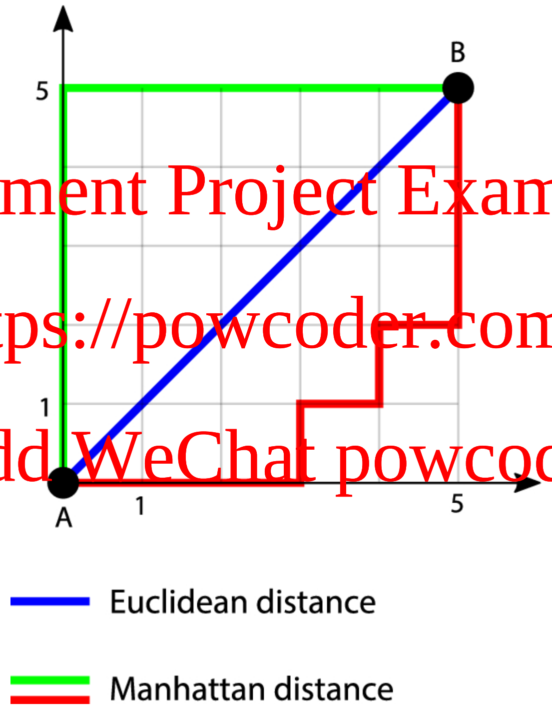
- Dividing each feature vector by its norm
  - The Manhattan distance (l_1 norm): the sum of the absolute variables of the vector

  - $l_1(X) = |x_1| + |x_2| + \cdots + |x_n|$

  - The Euclidean distance (l_2 norm): square root of the sum of the square of the variables of the vector
  - $l_2(X) = \sqrt{x_1^2 + x_2^2 + \cdots + X_n^2}$

Manhattan Distance (l_1 norm)

- Also referred to as Taxicab or City Block Distance
  - The distance between two points is measured along axes at right angle
  - The sum of differences across dimensions
- More appropriate if columns are not similar in type
- Less sensitive to outliers



Euclidean Distance (l_2 norm)

- Most commonly used distance
  - Corresponds to the geometric distance into the multi-dimensional space
- If columns have values with differing scales, it is common to first normalize or standardize the numerical columns

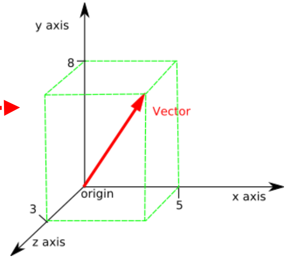
Vector normalization takes a vector of any length and changes its length to 1 while keeping the direction unchanged

Observations

Name	Amount	Date	Issued In	Used In	Age	Education	Fraud?
Daniel	$2,600.45	1-Jul-2020	HK	HK	22	Secondary	No
Alex	$2,294.58	1-Oct-2020	HK	RUS	None	Postgraduate	Yes
Adrian	$1,003.30	3-Oct-2020	HK		25	Graduate	Yes
Vicky	$8,488.32	4-Oct-2020	JAPAN	HK	64	Graduate	No
Adams	¥20000	7-Oct-2020	AUS	JAP	58	Primary	No
…	…	…	…	…	…	…	…
Jones	₽3,250.11	Nov 1, 2020	HK	RUS	43	Graduate	No
Mary	₽8,156.20	Nov 1, 2020	HK	N/A	27	Graduate	Yes
Max	€7475,11	Nov 8, 2020	UK	GER	32	Primary	No
Peter	₽500.00	Nov 9, 2020	Hong Kong	RUS	0	Postgraduate	No
Anson	₽7,475.11	Nov 9, 2020	Hong Kong	RUS	20	Postgraduate	Yes

Feature

Target



The choice of removal, imputation, and normalization is determined by the superiority of model accuracy

	Imputation Technique	# of rows in the training dataset	Accuracy
1	Dropping rows with missing values	392	0.74489
2	Imputing missing values with zero	768	0.7304
3	Imputing missing values with the mean	768	0.7318
4	Imputing missing values with the median	769	0.7357
5	z-Score normalization with median imputation	768	0.7422
6	Min-max normalization with mean imputation	768	0.7461
7	Row normalization with mean imputation	768	0.6823

Feature Construction

Feature construction is a form of data enrichment that adds derived features to data

- Feature construction is a form of data enrichment that adds derived features to data
- Feature construction involves transforming a given set of input features to generate a new set of more powerful features which are then used for prediction
- This may be done either to compress the dataset by reducing the number of features or to improve the prediction performance
- The new features will ideally hold new information and generate new patterns that ML models will be able to exploit and use to increase performance

New features may be constructed based on existing features to enable and enhance machine learning

categorical variable encoding

Name	Amount	Date	Issued In	Used In	Age	Education	Fraud?	Issued In_HK	…
Daniel	$2,600.45	1-Jul-2020	HK	HK	22	Secondary	No	1	…
Alex	$2,294.58	1-Oct-2020	HK	RUS	None	Postgraduate	Yes	1	…
Adrian	$1,003.30	3-Oct-2020	HK		25	Graduate	Yes	1	…
Vicky	$8,488.32	4-Oct-2020	JAPAN	HK	64	Graduate	No	0	…
Adams	¥20000	7-Oct-2020	AUS	JAP	58	Primary	No	0	…
…	…	…	…	…	…	…	…		…
Jones	₽3,250.11	Nov 1, 2020	HK	RUS	43	Graduate	No	1	…
Mary	₽8,156.20	Nov 1, 2020	HK	N/A	27	Graduate	Yes	1	…
Max	€7475,11	Nov 8, 2020	UK	GER	32	Primary	No	0	…
Peter	₽500.00	Nov 9, 2020	Hong Kong	RUS	0	Postgraduate	No	1	…
Anson	₽7,475.11	Nov 9, 2020	Hong Kong	RUS	20	Postgraduate	Yes	1	…

Observations

Feature

Target

Encoding Nominal Qualitative Data

Categorical Encoding

- The values of categorical variables are often encoded as strings

- Scikit-learn does not support strings as values, therefore, we need to transform those strings into numbers

- The act of replacing strings with numbers is called categorical encoding

Dummy Variables

- Dummy variables take the value 0 or 1 to indicate the absence or presence of a category
- They are proxy variables, or numerical stand-ins, for qualitative variables
  - Consider a simple regression analysis for wage determination
    - Say we are given gender, which is qualitative, and years of education, which is quantitative
    - In order to see if gender has an effect on wages, we would dummy code when the person is a female to female = 1, and female = 0 when the person is male.

One-Hot Encoding

- In one-hot encoding, we represent a categorical variable as a group of dummy variables, where each dummy variable represents one category

Gender
Female
Male
Male
Female

Gender_Female	Gender_Male
1	0
0	1
0	1
1	0

- One-hot encoding is applicable to nominal variables
  - for categorical variables not having a natural rank ordering

Dummy Variable Traps

- When working with dummy variables, it is important to avoid the dummy variable trap
- The trap occurs when independent variables are multicollinear or highly correlated
- To avoid the dummy variable trap, simply drop one of the dummy variables

Gender
Female
Male
Male
Female

Gender_Female	Gender_Male
1	0
0	1
0	1
1	0

A categorical variable with k categories can be captured using k-1 dummy variables but sometimes still with k variables

- A categorical variable with k categories can be encoded in k-1 dummy variables
  - For Gender, k is 2 (male and female), therefore, only one dummy variable (k - 1 = 1) is needed to capture all of the information
  - For a color variable that has three categories (red, blue, and green), two (k - 1 = 2) dummy variables are needed
    - red (red = 1, blue = 0), blue (red = 0, blue = 1), green (red = 0, blue = 0)
- There are a few occasions when categorical variables are encoded with k dummy variables
  - When training decision trees, as they do not evaluate the entire feature space at the same time
  - When selecting features recursively
  - When determining the importance of each category within a variable

Python: One-Hot Encoding (1)

load the relevant packages

```
import pandas as pd
```


load the dataset from the current working directory

```
data = pd.read_csv('FIN7790-02-2-feature_construction.csv')
```


show the dataset, which serves purely as a demo dataset

```
data
```


	city	boolean	ordinal_column	quantitative_column
0	tokyo	yes	somewhat like	1.0
1	tokyo	no	like	11.0
2	london	no	somewhat like	-0.5
3	seattle	no	like	10.0
4	san francisco	no	somewhat like	8.3
5	tokyo	yes	dislike	20.0

Python: One-Hot Encoding (2)

```
# list the nominal categorical variables to encode
cols =['city', 'boolean']
```



```
# use pandas get_dummies() to dummify the nominal categorical variables
# drop_first=True avoids the dummy variable trap by removing the first category
encoding = pd.get_dummies(data[cols], drop_first=True)
```



```
# show the one-hot encoding dataset
encoding
```


	city_san francisco	city_seattle	city_tokyo	boolean_yes
0	0	0	1	1
1	0	0	1	0
2	0	0	0	0
3	0	1	0	0
4	1	0	0	0
5	0	0	1	1

Python: One-Hot Encoding (3)

```
# combine the original dataframe with the one-hot encoding dataframe
# drop the ordinal categorical column first to avoid the dummy variable trap
data_enc = pd.concat([data.drop(columns=cols), encoding], axis=1)
```



```
# show the encoded dataset
data_enc
```


	ordinal_column	quantitative_column	city_san francisco	city_seattle	city_tokyo	boolean_yes
0	somewhat like	1.0	0	0	1	1
1	like	11.0	0	0	1	0
2	somewhat like	-0.5	0	0	0	0
3	like	10.0	0	1	0	0
4	somewhat like	8.3	1	0	0	0
5	dislike	20.0	0	0	1	1

get_dummies() will create one binary variable per found category. Hence, if there are more categories in the training dataset than in the testing dataset, get_dummies() will return more columns in the transformed training dataset than in the transformed testing dataset.

Encoding Ordinal Qualitative Data

Ordinal Encoding

- Ordinal encoding consists of
  - replacing the categories with digits from 1 to k (or 0 to k-1, depending on the implementation)
  - k is the number of distinct categories of the variable
- The numbers are assigned arbitrarily
- Ordinal encoding is better suited for non-linear machine learning models
  - ML models can navigate through the arbitrarily assigned digits to try and find patterns that relate to the target

Python: Ordinal Encoding (1)

load the relevant packages

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder
```


load the dataset from the current working directory


```
data = pd.read_csv('FIN7790-02-2-feature_construction.csv')
```

list the columns to encode


```
cols= ['ordinal_column']
data
```

	city	boolean	ordinal_column	quantitative_column
0	tokyo	yes	somewhat like	1.0
1	tokyo	no	like	11.0
2	london	no	somewhat like	-0.5
3	seattle	no	like	10.0
4	san francisco	no	somewhat like	8.3
5	tokyo	yes	dislike	20.0

Python: Ordinal Encoding (2)

```
# capture the encoding as an array of array
# each inner array applies to one column
# list categories in each inner array
# the order of the categories determines the values
mapping = [['dislike', 'like', 'somewhat like']]
```

```
# instantiate the encoder
encoder = OrdinalEncoder(categories=mapping,
                         dtype=np.int32)
```

```
# fit the data to the encoder
encoder.fit(data[cols])
```

```
# list the categories
encoder.categories_
```

```
[array(['dislike', 'like', 'somewhat like'], dtype=object)]
```

```
# build the encoding
encoding = pd.DataFrame(
    encoder.transform(data[cols]),
                      columns=cols)
```

```
# show the encoding
encoding
```

	ordinal_column
0	2
1	1
2	2
3	1
4	2
5	0

Python: Ordinal Encoding (3)

```
# build the encoded dataset
data_enc = pd.concat([data.drop(columns=cols), encoding], axis=1)
```



```
# show the encoded dataset
data_enc
```


	city	boolean	quantitative_column	ordinal_column
0	tokyo	yes	1.0	2
1	tokyo	no	11.0	1
2	london	no	-0.5	2
3	seattle	no	10.0	1
4	san francisco	no	8.3	2
5	tokyo	yes	20.0	0

Encoding Quantitative Data

Discretisation

- Discretization / Binning transforms continuous variables into discrete variables by creating a set of contiguous intervals (bins) spanning the value range
  - Places outliers into the lower or higher intervals together with the remaining inlier values of the distribution
  - Hence, these outliers no longer differ from the rest of the values at the tails of the distribution, as they are now all together in the same interval / bin
- Used to change the distribution of skewed variables, to minimize the influence of outliers, and hence to improve the performance of some ML models
- Binning can be achieved using supervised or unsupervised approaches

Equal-Width Discretization

- The variable values are sorted into intervals of the same width
- The number of intervals is decided arbitrarily

$$Width = \frac{Max(X) - Min(X)}{Bins}$$

- Values in the training dataset range from 0 to 100 and to create 5 bins, bin width = (100 - 0) / 5 = 20

- The bins will be 0-20, 20-40, 40-60, 80-100
- The first bin (0-20) and final bin (80-100) can be expanded to accommodate outliers found in other datasets, i.e., values < 0 or > 100 would be placed in those bins by extending the limits to minus and plus infinity

Python: Equal-Width Discretization (1)

load the relevant packages

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


load the dataset from the current working directory

```
data = pd.read_csv('FIN7790-02-2-feature_construction.csv')
```


list the columns to encode

```
cols= ['quantitative_column']
data
```


	city	boolean	ordinal_column	quantitative_column
0	tokyo	yes	somewhat like	1.0
1	tokyo	no	like	11.0
2	london	no	somewhat like	-0.5
3	seattle	no	like	10.0
4	san francisco	no	somewhat like	8.3
5	tokyo	yes	dislike	20.0

Python: Equal-Width Discretization (2)

initiate an ordinal encoder

```
disc = KBinsDiscretizer(n_bins=10, encode='ordinal',
                        strategy='uniform')
```

fit the data to the discretizer

```
disc.fit(data[cols])
```

list the learnt bins

```
disc.bin_edges_
      array([array([-0.5 ,  1.55,  3.6 ,  5.65,  7.7 ,  9.7
      5, 11.8 , 13.85, 15.9 ,
            17.95, 20.  ])], dtype=object)
```

build the discretization for the quantitative variable

```
discretization = pd.DataFrame(
            disc.transform(data[cols]), columns=cols)
```

show the discretization

```
discretization
```

quantitative_column
1.0
11.0
-0.5
10.0
8.3
20.0

	quantitative_column
0	0.0
1	5.0
2	0.0
3	5.0
4	4.0
5	9.0

[-∞ , 1.55)	0.0
[1.55, 3.6)	1.0
[3.6, 5.65)	2.0
[5.65, 7.7)	3.0
[7.7, 9.75)	4.0

[9.75, 11.8)	5.0
[11.8, 13.85)	6.0
[13.85, 15.9)	7.0
[15.9, 17.95)	8.0
[17.95, ∞)	9.0

Python: Equal-Width Discretization (3)

build the discretized dataset

```
data_disc = pd.concat([data.drop(columns=cols), discretization], axis=1)
```


show the discretized dataset

```
data_disc
```


	city	boolean	ordinal_column	quantitative_column
0	tokyo	yes	somewhat like	0.0
1	tokyo	no	like	5.0
2	london	no	somewhat like	0.0
3	seattle	no	like	5.0
4	san francisco	no	somewhat like	4.0
5	tokyo	yes	dislike	9.0

After one-hot encoding, ordinal encoding, and discretization, the original dataset becomes a purely numerical dataset

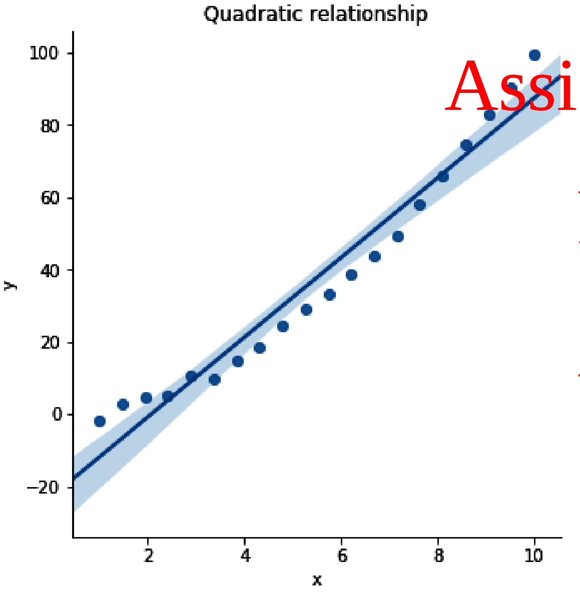
index	ordinal_column	quantitative_column	boolean_yes	city_san francisco	city_seattle	city_tokyo
0	2	0.0	1	0	0	1
1	1	5.0	0	0	0	1
2	2	0.0	0	0	0	0
3	1	5.0	0	0	1	0
4	2	4.0	0	1	0	0
5	0	9.0	1	0	0	1

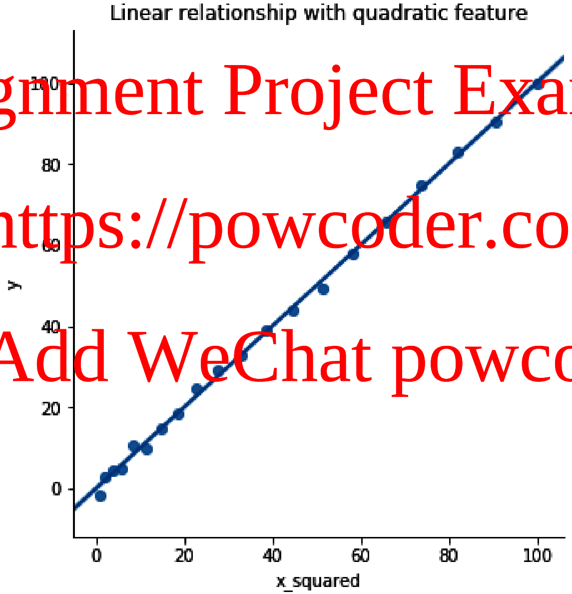
Extending Quantitative Data with Polynomial Features

Polynomial Expansion

- A combination of one feature with itself (i.e. a polynomial combination of the same feature) can also be quite informative or increase the predictive power of the predictive algorithms
  - e.g., the target follows a quadratic relationship with a variable, creating a second degree polynomial of the feature allows us to use it in a linear model
- With similar logic, polynomial combinations of the same or different variables can return new variables that convey additional information and capture feature interaction
- Can be better inputs for our ML algorithms, particularly for linear models

A linear relationship can be created for polynomial features using a polynomial combination





- In the plot on the left, due to the quadratic relationship between the target (y) and the variable (x), there is a poor linear fit
- In the plot on the right, the x^2 variable (a quadratic combination of x) shows a linear relationship with the target (y) and therefore improves the performance of the linear model, which predicts y from x^2

Polynomial features may result in improved modeling performance at the cost of adding thousands of variables

- Often, the input features for a predictive modeling task interact in unexpected and often nonlinear ways
- These interactions can be identified and modeled by a learning algorithm
- Another approach is to engineer new features that expose these interactions and see if they improve model performance
- Transforms like raising input variables to a power can help to better expose the important relationships between input variables and the target variable
- These features are called interaction/polynomial features and allow the use of simpler modeling algorithms as some of the complexity of interpreting the input variables and their relationships is pushed back to the data preparation stage

A set of new polynomial features is created based on the degree of the polynomial combination

- The degree of the polynomial is used to control the number of features added, e.g. a degree of 3 will add two new variables for each input variable

- Typically a small degree, such as 2 or 3, is used
  - 2nd degree polynomial combinations return the following new features

$$[a, b, c]^2 = 1, a, b, c, ab, ac, bc, a^2, b^2, c^2$$

including all possible interactions of degree 1 and degree 2 plus the bias term 1

  - 3rd degree polynomial combinations return the following new features

$$[a, b, c]^3 = 1, a, b, c, ab, ac, bc, abc, a^2b, a^2c, b^2a, b^2c, c^2a, c^2b, a^3, b^3, c^3$$

including all possible interactions of degree 1, degree 2, and degree 3 plus the bias term 1

The Accelerometer Dataset

- The dataset collects data from a wearable accelerometer mounted on the chest intended for activity recognition research
- Data are collected from 15 participants performing 7 activities
- It provides challenges for identification and authentication of people using motion patterns
- Sampling frequency: 52 Hz
- Data calibration: no

- 15 datasets, one for each participant

Index	Variable	Definition	Values	
0	ID	Identifier	Numerical	
1	Xacc	X acceleration	Numerical	
2	Yacc	Y acceleration	Numerical	
3	Zacc	Z acceleration	Numerical	
4	Label	Activity	1	working at computer
			2	standing up, walking and going up/down stairs
			3	standing
			4	walking
			5	Going up/down stairs
			6	walking and talking with someone
			7	talking while standing

Source: https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer

Python: Polynomial Combinations (1)

load relevant packages and dataset with proper feature variables

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

data = pd.read_csv('FIN7790-02-2-accelerometer.csv', header=None)
data.columns = ['ID', 'x', 'y', 'z', 'activity']
data = data.astype({'ID': 'int'})

data.head()
```

	ID	x	y	z	activity
0	0	1502	2215	2153	1
1	1	1667	2072	2047	1
2	2	1611	1957	1906	1
3	3	1601	1939	1831	1
4	4	1643	1965	1879	1

Python: Polynomial Combinations (2)

show information summary

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162501 entries, 0 to 162500
Data columns (total 5 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   ID        162501 non-null  int32
 1   x         162501 non-null  int64
 2   y         162501 non-null  int64
 3   z         162501 non-null  int64
 4   activity  162501 non-null  int64
dtypes: int32(1), int64(4)
memory usage: 5.6 MB
```


show descriptive statistics

```
data.describe()
```

	ID	x	y	z	activity
count	162501.000000	162501.000000	162501.000000	162501.000000	162501.000000
mean	81250.000000	1910.670857	2380.286367	2041.214829	4.899681
std	46910.142472	40.653208	41.925728	59.529406	2.424311
min	0.000000	1455.000000	1697.000000	1644.000000	0.000000
25%	40625.000000	1886.000000	2374.000000	1991.000000	3.000000
50%	81250.000000	1905.000000	2381.000000	2022.000000	7.000000
75%	121880.000000	1935.000000	2386.000000	2101.000000	7.000000
max	162500.000000	2356.000000	2713.000000	2739.000000	7.000000

Python: Polynomial Combinations (3)

```
# split the dataset into features and targets
X = data[['x', 'y', 'z']]
y = data['activity']
```



```
# set up a polynomial expansion transformer of a degree less than or equal to 2
# interaction_only=False retains all of the combinations
# include_bias=False avoids returning the bias term column of all 1's
poly = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
```



```
# fit the transformer to the dataset
# let the transformer learn all of the possible polynomial combinations of the three variables
X_poly = poly.fit_transform(X)
data_X_poly = pd.DataFrame(X_poly, columns=poly.get_feature_names())
```

```
# show combinations covered by the transformer
poly.get_feature_names()
```

```
['x0', 'x1', 'x2', 'x0^2', 'x0 x1', 'x0 x2', 'x1^2', 'x1 x2', 'x2^2']
```

Python: Polynomial Combinations (4)

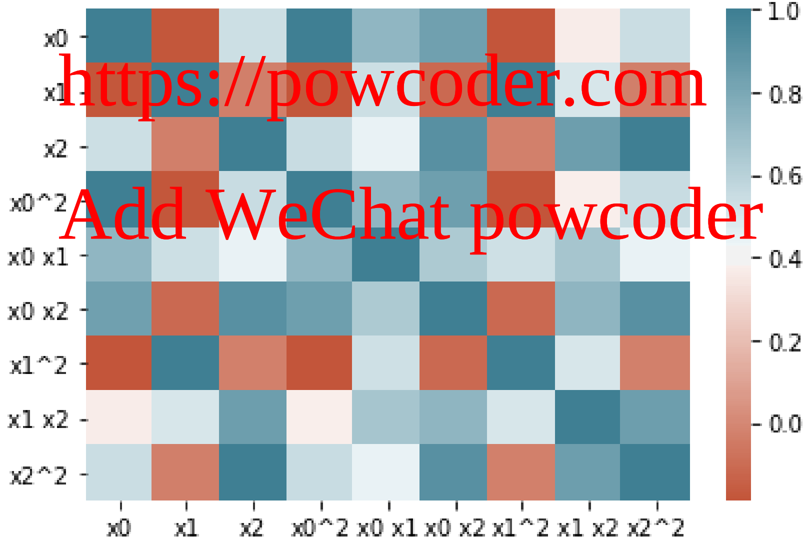
calculate correlation matric between feature pairs

```
data_X_poly.corr()
```


	x0	x1	x2	x0^2	x0 x1	x0 x2	x1^2	x1 x2	x2^2
x0	1.000000	-0.178532	0.542065	0.999710	0.731002	0.836154	-0.186701	0.373188	0.544065
x1	-0.178532	1.000000	-0.027592	-0.178767	0.540347	-0.104801	0.999473	0.502189	-0.027187
x2	0.542065	-0.027592	1.000000	0.549161	0.443159	0.914041	-0.025158	0.850422	0.999881
x0^2	0.999710	-0.178767	0.549161	1.000000	0.730759	0.840733	-0.186603	0.379204	0.551201
x0 x1	0.731002	0.540347	0.443159	0.730759	1.000000	0.641393	0.533193	0.666114	0.445166
x0 x2	0.836154	-0.104801	0.914041	0.840733	0.641393	1.000000	-0.107150	0.734587	0.914968
x1^2	-0.186701	0.999473	-0.025158	-0.186603	0.533193	-0.107150	1.000000	0.504145	-0.024750
x1 x2	0.373188	0.502189	0.850422	0.379204	0.666114	0.734587	0.504145	1.000000	0.850543
x2^2	0.544065	-0.027187	0.999881	0.551201	0.445166	0.914968	-0.024750	0.850543	1.000000

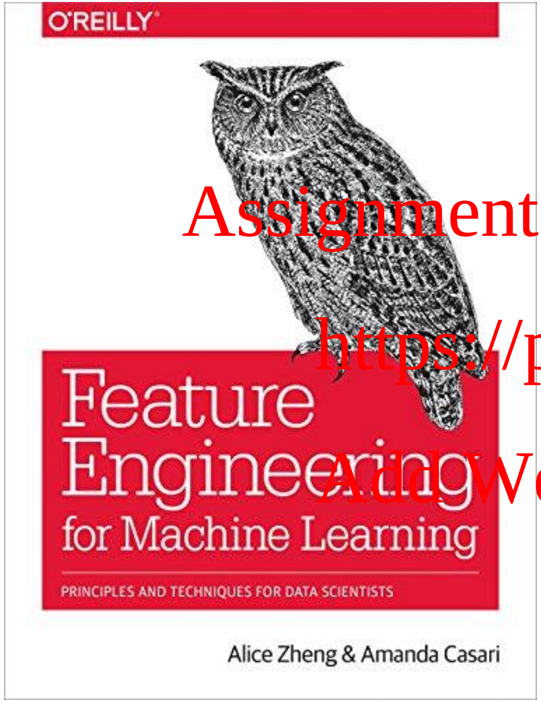
Python: Polynomial Combinations (5)

```
# show correlation matric between feature pairs
# the darker the color, the greater the correlation of the features
sns.heatmap(data_X_poly.corr(), cmap=sns.diverging_palette(20, 220, n=200))
```

References

References



"Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists", Alice Zhang & Amanda Casari, O'Reilly Media, April 2018, ISBN-13: 978-1-491-95324-2

"Feature Engineering Made Simple", Susan Ozdemir & Divya Susarla, Packt Publishing, January 2018, ISBN-13: 978-1-787-28760-0

THANK YOU