

Welcome to Feature Engineering!

In this assignment you'll learn about one of the most important steps on the way to building a great machine learning model: feature engineering.

What is a feature and why we need to engineer it?

Basically, all machine learning algorithms use some input data to create outputs. This input data comprise features, which are usually in the form of structured columns. Algorithms require features with some specific characteristic to work properly.

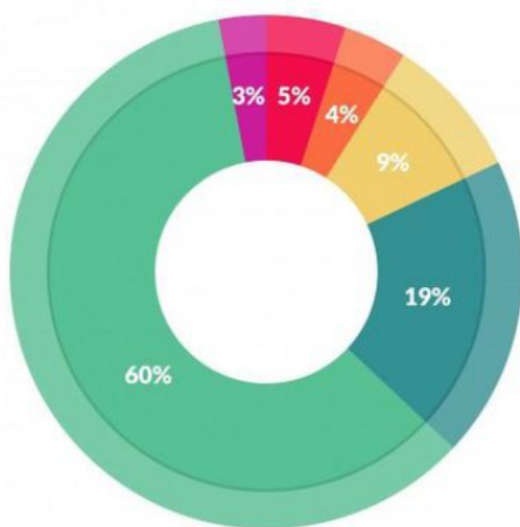
Here, the need for feature engineering arises. With feature engineering, you'll learn how to:

- determine which features are the most important with mutual information
- invent new features in several real-world problem domains
- encode high-cardinality categoricals with a target encoding
- prepare the proper input dataset, compatible with the machine learning algorithm requirements
- improve the performance of machine learning models

Significance of Feature Engineering

According to a survey in Forbes, data scientists spend 80% of their time on data preparation. That's why we need feature engineering for extracting useful features from raw data using maths, statistics and domain knowledge.

The best way to achieve expertise is practicing different techniques on various datasets and observing their effect on model performances. So fellas, get ready to dive into various techniques of Feature Engineering.



What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets; 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

Throughout the Assignment, we'll be needing use of these 2 libraries, so let's import them first.

In [3]:

```
# importing Pandas  
  
# importing NumPy
```

1. Feature Creation

5 points

Preprocessing is all about changing raw data into something that will help you improve the performance of your model. But you must be thinking how can we create own Feature and will it help increase the performance of my model ?

So, let's see what is actually meant by Feature Creation. You've a bank data and you're building a profit predictor model. You'll have all the values required like interest rate, revenue etc but what actually make sense is having Profit amount out of it as a core feature.

So, here's where you'll require feature creation techniques where you can use calculations on existing features and add another column to your data.

Here's the 500_Person_Gender_Height_Weight_Index dataset, let's see how feature creation is useful here. You can download this dataset from:

<https://www.kaggle.com/yersever/500-person-gender-height-weight-bodymassindex>
(<https://www.kaggle.com/yersever/500-person-gender-height-weight-bodymassindex>)

In [4]:

```
# read data  
  
#data.head()
```

Out[4]:

	Gender	Height	Weight	Index
0	Male	174	96	4
1	Male	189	87	2
2	Female	185	110	4
3	Female	195	104	3
4	Male	149	61	3

Lets create 2 new features in our data. One 'Height_meters' which will have height in meters instead of cm to get another feature named 'calculated_BMI', which will have BMI for each record

Don't worry if you don't remember BMI formula here we provide you

$BMI = \text{weight (kg)} \div \text{height}^2 \text{ (m)}$

Reference video to create new column: <https://www.youtube.com/watch?v=Wg0uiBf8dug>

(<https://www.youtube.com/watch?v=Wg0uiBf8dug>).

In [5]:

```
# We need to calculate BMI which can be done using Height and Weight features
data['Height_meters'] = #calculate
data['calculated_BMI'] = #calculate
data.head()
```

Out[5]:

	Gender	Height	Weight	Index	Height_meters	calculated_BMI
0	Male	174	96	4	1.74	31.708284
1	Male	189	87	2	1.89	24.355421
2	Female	185	110	4	1.85	32.140248
3	Female	195	104	3	1.95	27.350427
4	Male	149	61	3	1.49	27.476240

Now we need to see some deviation from maximum height, minimum height and average calculated BMI. So for this lets create separate 3 columns i.e. 'deviation_from_max_height', 'deviation_from_min_weight' and 'deviation_from_avg_BMI'

Hint: you can use mean, median, max or min like stats to calculate. deviation can be calculated as difference between current value and your stat(whatever your using like min, max etc) value

In [6]:

```
data['deviation_from_max_height'] = #calculate
data['deviation_from_min_weight'] = #calculate
data['deviation_from_avg_BMI'] = #calculate
#See how it affected our data using head
```

Out[6]:

	Gender	Height	Weight	Index	Height_meters	calculated_BMI	deviation_from_max_height
0	Male	174	96	4	1.74	31.708284	-25
1	Male	189	87	2	1.89	24.355421	-10
2	Female	185	110	4	1.85	32.140248	-14
3	Female	195	104	3	1.95	27.350427	-4
4	Male	149	61	3	1.49	27.476240	-50

Alright, as you have understood how important feature engineering is, so now we'll be using Kaggle's "Titanic - Machine learning from Disaster" dataset for performing next some of the feature engineering techniques, you can find its link below:

<https://www.kaggle.com/c/titanic/data> (<https://www.kaggle.com/c/titanic/data>)

In [7]:

read data

data.head()

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

2. Imputation

10 points

The most common problem you will encounter when you try to prepare your data for machine learning is about missing values which do affect the performance of your machine learning model. The reason for the missing values can be human errors, interruptions in the data flow, privacy concerns, and so on.

The most simple solution to the missing values one can think about is to drop the rows or the entire column. There is not an optimum threshold for dropping but you can use 70% as an example value and try to drop the rows and columns which have missing values with higher than this threshold.

Reference video to drop null values:

In [9]:

```
# keeping threshold as 70%
threshold = 0.7
#filter columns with mean missing value higher than threshold
data = data[data.columns[data.isnull().mean() < threshold]]

#filter rows with mean missing value rate higher than threshold
data = data.loc[data.isnull().mean(axis=1) < threshold]

data.head()
```

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	I
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

Fun fact for you geeky!

You can also use dropna with thresh parameter set as 70% of data size to do this

Reference: <https://www.youtube.com/watch?v=jXNkKKR5b80> (<https://www.youtube.com/watch?v=jXNkKKR5b80>)

Can you do it now with our data? try below.

In []:

```
# read the data again with name d1
d1=#

#drop columns with condition that if our data has 70% or more than 70% of nan in it using d
```

Now, as you can observe that 'cabin' column from the data is dropped. But that's not it, there're still cloulmns where there are NaN values which we need to take care of.

Interesting right? Let's see what else we can do ahead.

Note: We are using data further and not d1.

In [10]:

```
# display total null in each column
```

Out[10]:

```
PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age             177
SibSp            0
Parch            0
Ticket           0
Fare             0
Embarked         2
dtype: int64
```

But What is Imputation ?

This answer lies itself in its definition which basically justifies why do we need it as well:

' The assignment of a value to something by inference from the value of the products or processes to which it contributes '

Numerical Imputation

Imputation is a more preferable option rather than dropping because it preserves the data size. However, there is an important selection of what you impute to the missing values. We suggest beginning with considering a possible default value of missing values in the column. For example, if you have a column that only has 1 and NA, then it is likely that the NA rows correspond to 0. For another example, if you have a column that shows the "customer visit count in last month", the missing values might be replaced with 0 as long as you think it is a sensible solution.

Another reason for the missing values is joining tables with different sizes and in this case, imputing 0 might be reasonable as well. Except for the case of having a default value for missing values, I think the best imputation way is to use the medians of the columns. As the averages of the columns are sensitive to the outlier values, while medians are more solid in this respect.

Let's see this with respect to the example of Age column of our data.

Reference video to fill nan with mean: <https://www.youtube.com/watch?v=t5NUbThn6rY>
(<https://www.youtube.com/watch?v=t5NUbThn6rY>)

Note: But you can use median instead of mean

In [11]:

```
# Filling missing values with medians of Age column
data['Age'] = #fill missing

data.head()
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	I
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

In [12]:

```
# Check again total number of nan in each feature

data.isnull().sum()
```

Out[12]:

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       2
dtype: int64
```

Observed something strange ?

Yes, Embarked column still has 2 null values but how do we deal with such categorical imputations ?

Categorical Imputation

Replacing the missing values with the maximum occurred value in a column is a good option as well for handling categorical columns. But if you think the values in the column are distributed uniformly and there is not a dominant value, imputing a category like "Other" might be more sensible, because in such a case, your imputation is likely to converge a random selection.

Reference video: <https://www.youtube.com/watch?v=jJ9gdOezo3U> (<https://www.youtube.com/watch?v=jJ9gdOezo3U>)

In [13]:

```
# Lets impute nan in column Embarked with maximum occurred values in it.  
  
#check total count of nan again in each feature
```

Out[13]:

```
PassengerId    0  
Survived       0  
Pclass         0  
Name           0  
Sex            0  
Age            0  
SibSp          0  
Parch          0  
Ticket         0  
Fare           0  
Embarked       0  
dtype: int64
```

Well done! Now our data does not have any missing values

3.Handling Outliers

30 points

Before mentioning how outliers can be handled, We want you to know that the best way to detect the outliers is to demonstrate the data visually. All other statistical methodologies are open to making mistakes, whereas visualizing the outliers gives a chance to take a decision with high precision.

Statistical methodologies are less precise, but on the other hand, they have a superiority, they are fast. Here you'll learn two different ways of handling outliers. These will detect them using standard deviation, and percentiles.

Outlier Detection with Standard Deviation

If a value has a distance to the average higher than x * standard deviation, it can be assumed as an outlier. Then what x should be? There is no trivial solution for x , but usually, a value between 2 and 4 seems practical.

Reference video: <https://www.youtube.com/watch?v=KFuEAGR3HS4> (<https://www.youtube.com/watch?v=KFuEAGR3HS4>)

In [14]:

```
#We will need same data for other techniques as well so better we leave original undisturbed
new_data = #copy data

# Lets define our factor=3 as our factor to calculate our limit
factor = #

# Calculate upper limit and lower limit of 'Fare' column to decide the threshold to detect
#Hint use: upper_lim=mean+std*factor
#          lower_lim=mean-std*factor

upper_lim = #calculate using above formula
lower_lim = #calculate using above formula
```

In [15]:

```
#Now filter our 'Fare' with data lower than upper_lim and greater than lower_lim
new_data = #filter out with above condition

#print the length of original and new data
```

Length of original data: 891

Length of new data: 871

In addition, z-score can be used instead of the formula above. Z-score (or standard score) standardizes the distance between a value and the mean using the standard deviation.

We hope you have already learned z-score technique in our Data cleaning assignment.

Outlier Detection with Percentiles

Another mathematical method to detect outliers is to use percentiles. You can assume a certain percent of the value from the top or the bottom as an outlier. The key point is here to set the percentage value once again, and this depends on the distribution of your data as mentioned earlier.

Additionally, a common mistake is using the percentiles according to the range of the data. In other words, if your data ranges from 0 to 100, your top 5% is not the values between 96 and 100. Top 5% means here the values that are out of the 95th percentile of data.

Reference video: <https://www.youtube.com/watch?v=2QroSt474lQ> (<https://www.youtube.com/watch?v=2QroSt474lQ>)

In [16]:

```
#copy data again into new_data
new_data = data.copy()

#create upper and lower limits for column 'Fare' based on 95% and 5% respectively.
upper_lim = data['Fare'].quantile(0.95)
lower_lim = data['Fare'].quantile(0.05)

#Dropping the outlier rows in 'Fare' with your created limited.

new_data = data[(data['Fare'] < upper_lim) & (data['Fare'] > lower_lim)]

#print the length of both original and new data after filter
```

Length of original data: 891

Length of new data: 791

An Outlier Dilemma: Drop or Cap

Another option for handling outliers is to cap them instead of dropping. So you can keep your data size and at the end of the day, it might be better for the final model performance. On the other hand, capping can affect the distribution of the data, thus it better not to exaggerate it.

Reference link: <https://www.pluralsight.com/guides/cleaning-up-data-from-outliers>
(<https://www.pluralsight.com/guides/cleaning-up-data-from-outliers>).

Note: check Quantile-based Flooring and Capping section in given link

In [17]:

```
#copy data into new_data
new_data = data.copy()

#create upper and lower limits for column 'Fare' based on 95% and 5% respectively.
upper_lim = data['Fare'].quantile(0.95)
lower_lim = data['Fare'].quantile(0.05)

#Capping the outlier rows with Percentiles

# print length of original and new data after capping.
```

Length of original data: 891

Length of new data: 891

4. Scaling

40 points

In most cases, the numerical features of the dataset do not have a certain range and they differ from each other. In real life, it is nonsense to expect age and Ticket Fare columns to have the same range. But from the machine learning point of view, how these two columns can be compared?

Scaling solves this problem. The continuous features become identical in terms of the range, after a scaling process. This process is not mandatory for many algorithms, but it might be still nice to apply.

Basically, there are two common ways of scaling:

1. Normalization

$$X_{\text{norm}} = (X - X_{\text{min}}) / (X_{\text{max}} - X_{\text{min}})$$

Normalization scales all values in a fixed range between 0 and 1. This transformation does not change the distribution of the feature and due to the decreased standard deviations, the effects of the outliers increases. Therefore, before normalization, it is recommended to handle the outliers.

Note: here reference video is not provided as you need to directly apply formula given.

In [18]:

```
# copy the data into new_data
new_data = #

#Normalizing the Age column using the given normalization formula on column Age
new_data['Age'] = #

#Similarly Normalizing the Fare column using the given formula
new_data['Fare'] = #

new_data.head()
```

Out[18]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	0.271174	1	0	A/5 21171	0.014
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	0.472229	1	0	PC 17599	0.139
2	3	1	3	Heikkinen, Miss. Laina	female	0.321438	0	0	STON/O2. 3101282	0.015
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	0.434531	1	0	113803	0.103
4	5	0	3	Allen, Mr. William Henry	male	0.434531	0	0	373450	0.015

2. Standardization

$Z = (X - X_{\text{mean}}) / \sigma$; where σ = standard deviation

Standardization (or z-score normalization) scales the values while taking into account standard deviation. If the standard deviation of features is different, their range also would differ from each other. This reduces the effect of the outliers in the features.

In [19]:

```
# copy the data into new_data
new_data = data.copy()

#Standardizing the Age column using the Z formula given above
new_data['Age'] = #

#Standardizing the Fare column using the Z formula given above
new_data['Fare'] = #

new_data.head()
```

Out[19]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	-0.565419	1	0	A/5 21171	-0.50
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	0.663488	1	0	PC 17599	0.78
2	3	1	3	Heikkinen, Miss. Laina	female	-0.258192	0	0	STON/O2. 3101282	-0.48
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	0.433068	1	0	113803	0.42
4	5	0	3	Allen, Mr. William Henry	male	0.433068	0	0	373450	-0.48

We have used formula of both the techniques, so that you get logic behind there implimentation.

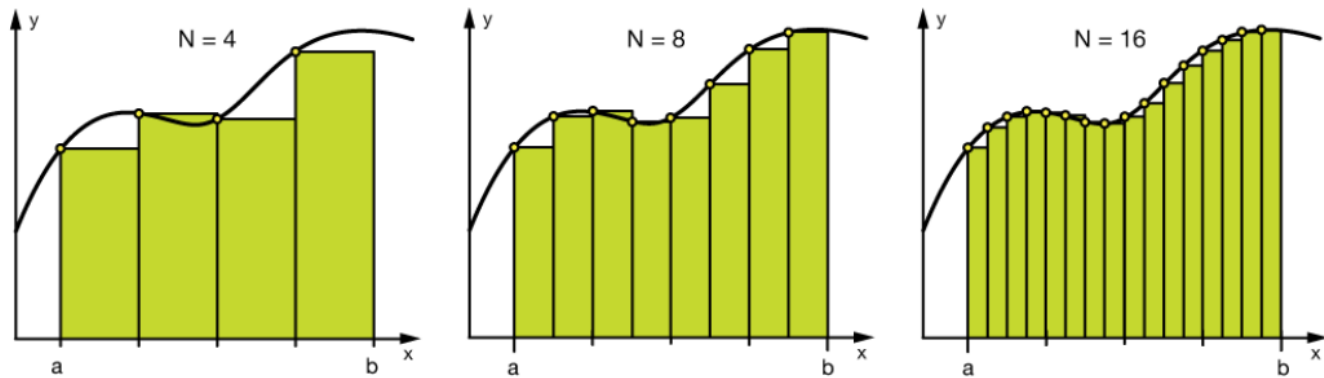
But good thing to know for you is that scikit learn already has library to scale the data with your choice of technique. Nice Right?

You can check its library implementation here: <https://youtu.be/MHCONU3RYsA>
(<https://youtu.be/MHCONU3RYsA>)

5. Binning

Data binning is a data pre-processing method used to minimize the effects of small observation errors. Binning is done to create bins for continuous variables where they are converted to categorical variables.

The original data values are divided into small intervals known as bins and then they are replaced by a general value calculated for that bin. This has a smoothing effect on the input data and may also reduce the chances of overfitting in case of small datasets.



Binning illustration of numerical data

We performed Binning for numerical age data so that we can prevent overfitting of model and rather deal this column as a categorical feature. It was observed that grouping into age groups improved the model performance, so sometimes in such cases grouping make more sense instead of using continuous numerical distribution for your model

Reference video : <https://www.youtube.com/watch?v=mDgL3Pp1IMw> (<https://www.youtube.com/watch?v=mDgL3Pp1IMw>)

In [20]:

```
# copy the data into new_data
new_data = #

# Impliment binning on Age and make another column names Age_group with bins=[0,18,40,100]
new_data['Age_group'] = #

#count total values of each category
```

Out[20]:

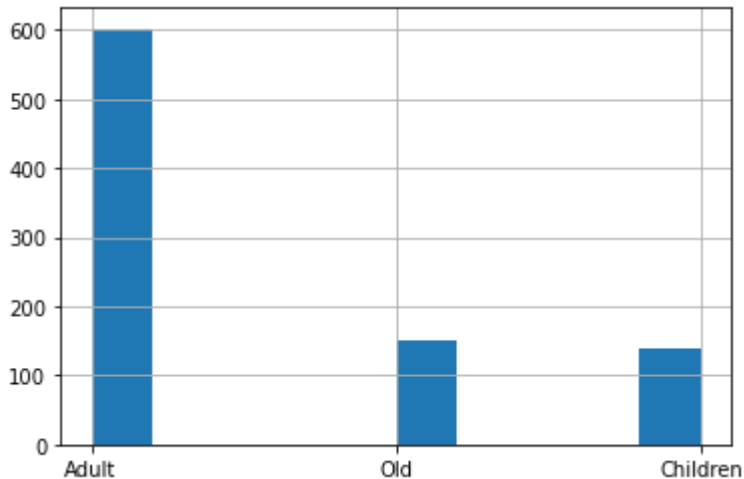
```
Adult      602
Old        150
Children   139
Name: Age_group, dtype: int64
```

In [21]:

```
#Checking 'Age_group' distribution
```

Out[21]:

<AxesSubplot:>



6. Encoding Categorical Variables

40 points

Reference video: <https://www.youtube.com/watch?v=J2gz0mbvg78> (<https://www.youtube.com/watch?v=J2gz0mbvg78>)

Some algorithms can work with categorical data directly.

For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation).

Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.

This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

In [22]:

```
# Count unique value counts for Sex  
# Count unique value counts for Embarked
```

Out[22]:

```
S      646  
C      168  
Q       77  
Name: Embarked, dtype: int64
```

You can see that Sex are having 2 categories and Embarked has 3 categories.

Let's use encoding on both the columns

Feature Encoding can be done using two steps:

1. Integer Encoding
2. One-Hot Encoding

Let's Implement both the techniques.

1. Integer encoding

Integer encoding (also known as label encoding) includes replacing the categories with digits from 1 to n (or 0 to n-1, depending on the implementation), where n is the number of the variable's distinct categories (the cardinality), and these numbers are assigned arbitrarily.

For example, "red" is 1, "green" is 2, and "blue" is 3.

Using sci-kit learn library we will import LabelEncoder to do this on Sex column

In [23]:

```
# Import Label encode
from sklearn.preprocessing import LabelEncoder

# creating instance of Labelencoder
labelencoder = #instance

# Assigning numerical values in column Sex and storing in another column named Sex_encode
data['Sex_encode'] = #

#print data.head()
data.head()
```

Out[23]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	I
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

Hey buddy. Did you see? male is encoded as 1 and female as 0

Lets check One-Hot Encoding technique

2. One-Hot Encoding

Though label encoding is straight but it has the disadvantage that the numeric values can be misinterpreted by algorithms as having some sort of hierarchy/order in them. This ordering issue is addressed in another common alternative approach called 'One-Hot Encoding'. In this strategy, each category value is converted into a new column and assigned a 1 or 0 (notation for true/false) value to the column.

Hey Intellipath! A fun fact!

The binary variables are often called “dummy variables” in other fields, such as statistics. Even we will be using pandas `get_dummies` function to do one hot encoding

Let's consider column Embarked to do this

In [24]:

```
#Using get_dummies function encode and concat encoded features in data
data=#

#data.head()
data.head()
```

Out[24]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	I
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

Columns C, Q and S are one hot encoded features

Wohoo! great job smarty! Did you see your last 4 columns are binary encoded out of Sex and Embarked features.

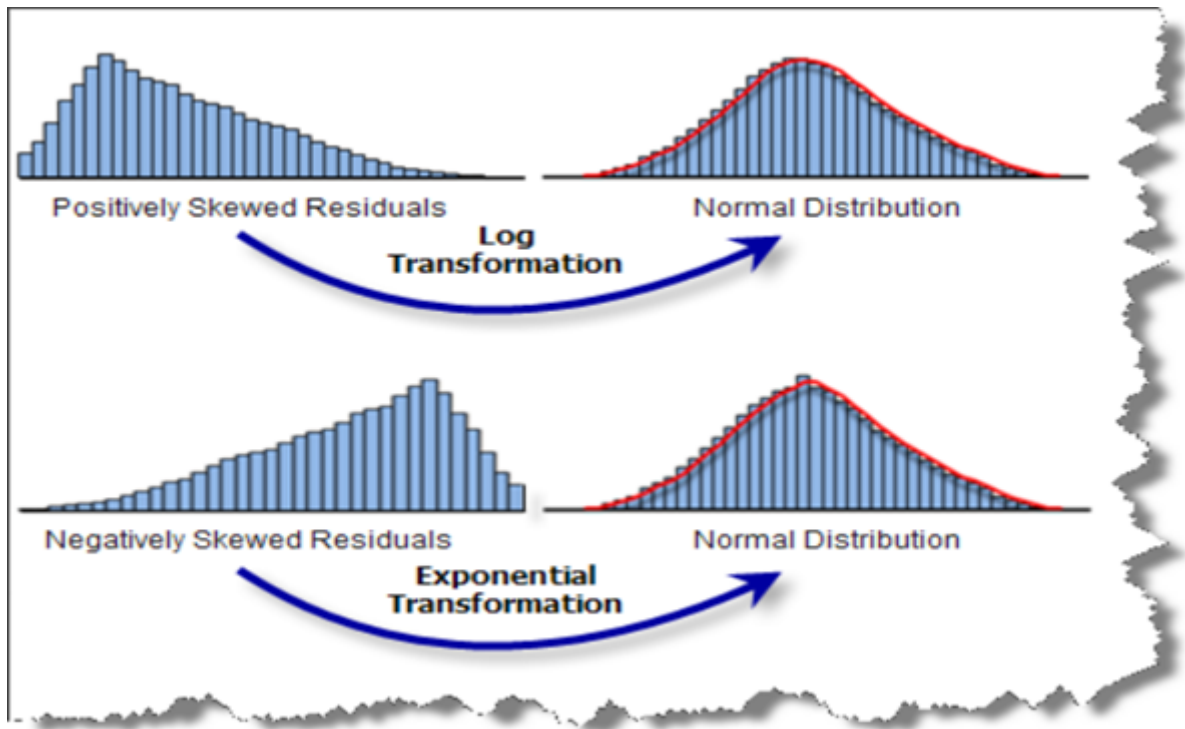
7. Variable Transformation

Reference video: https://youtu.be/U_wKdCBC-w0 (https://youtu.be/U_wKdCBC-w0).

15+15+15+15 = 60 points

Some machine learning models, like linear and logistic regression, assume that the variables follow a normal distribution. More likely, variables in real datasets will follow more a skewed distribution.

By applying a number of transformations to these variables, and mapping their skewed distribution to a normal distribution, we can increase the performance of our models.



How can we transform variables?

The most commonly-used methods to transform variables are the following:

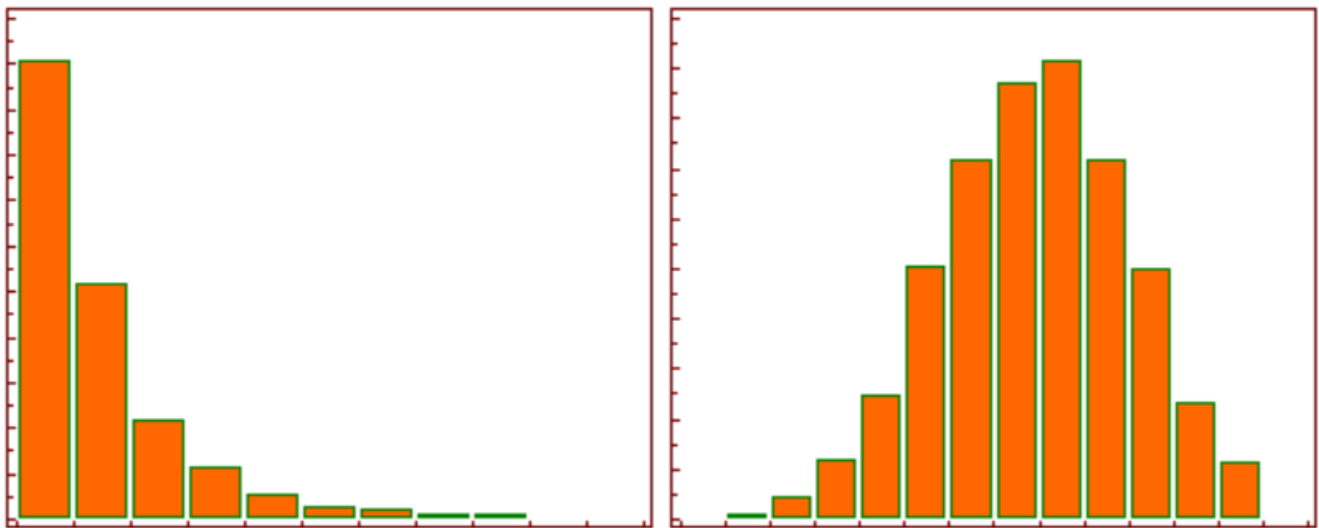
- Logarithmic transformation
- Square root transformation
- Reciprocal transformation
- Exponential or power transformation

1. Logarithmic transformation [$f(x) = \ln(x)$]

This is the simplest and most popular among the different types of transformations and involves a substantial transformation that significantly affects distribution shape.

We can use it (natural logarithmic \ln or log base 10) to make extremely skewed distributions less skewed, especially for right-skewed distributions.

You should note that this function is defined only for strictly positive numbers.



For understanding Logarithmic transformation, let's us see the example of kc-House-data whose link you can find here.

<https://www.kaggle.com/shivachandel/kc-house-data> (<https://www.kaggle.com/shivachandel/kc-house-data>)

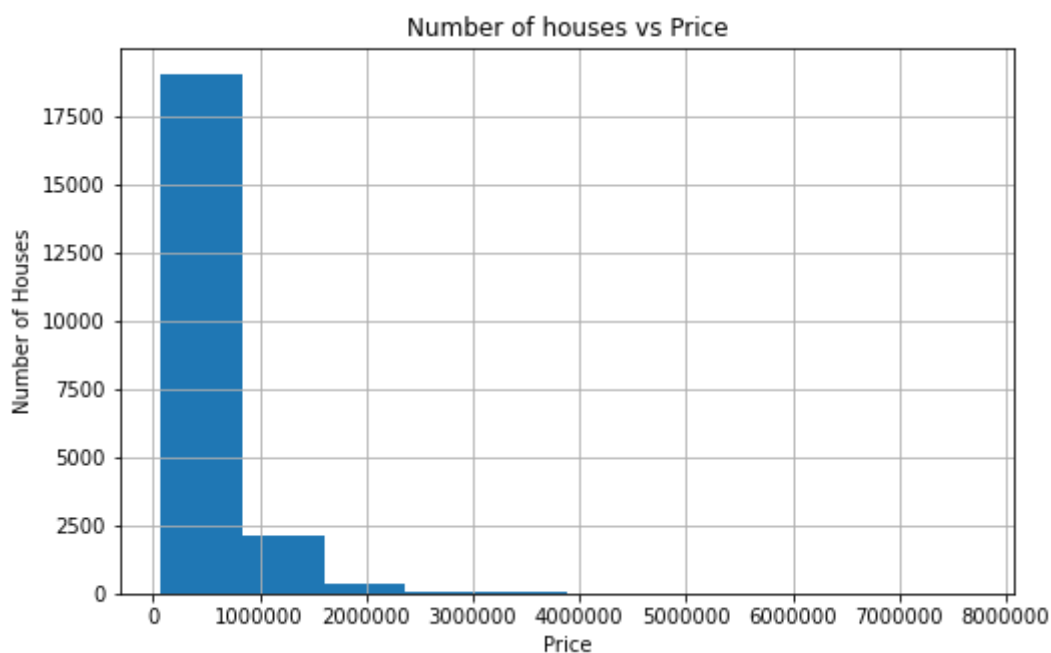
In [26]:

```
import matplotlib.pyplot as plt

#reading the data into variable new_data
new_data = #read data

#plot price histogram

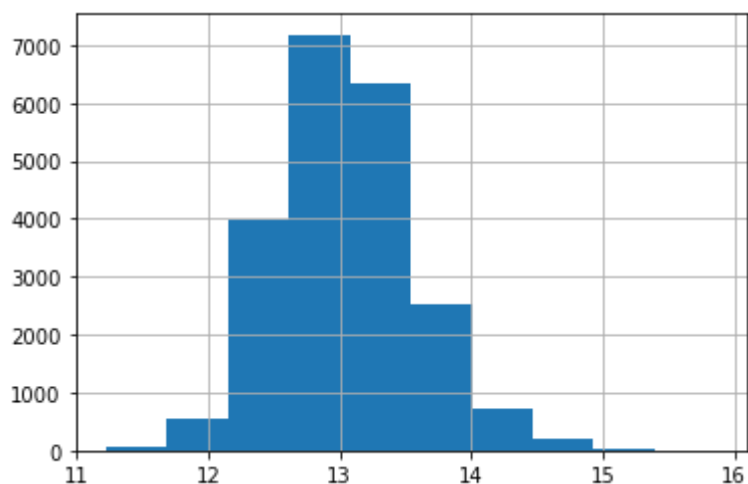
plt.title('Number of houses vs Price')
plt.ylabel('Number of Houses')
plt.xlabel("Price")
plt.ticklabel_format(style='plain')
```



Lets apply log transform on price column and plot its distribution

In [54]:

```
#Log transform
```



Interesting plot right? You have seen the difference between non transformed and transformed Price data.

2. Square root transformation [$f(x) = \sqrt{x}$]

Another simple transformation, this one has an average effect on distribution shape: it's weaker than logarithmic transformation, and it's also used for reducing right-skewed distributions.

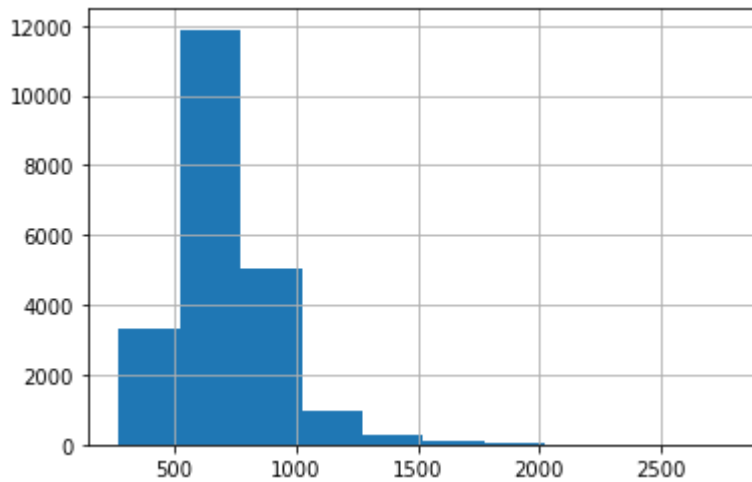
One particular advantage of square root transformation is that you can apply it to zero values.

You should note that this function is defined only for positive numbers.



In [60]:

```
#square root tranform and its distribution
```

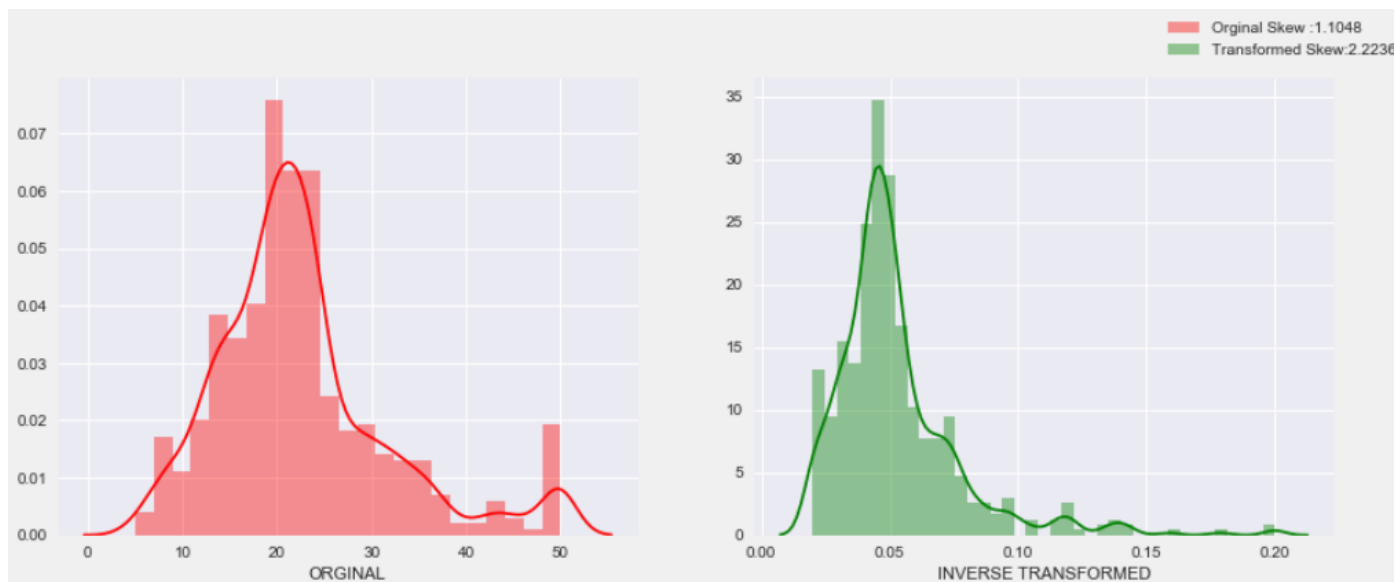


Yes buddy did you noticed from the above plot? that Square root transformation is actually weaker than log transformation

3. Reciprocal transformation [$f(x) = 1/x$]

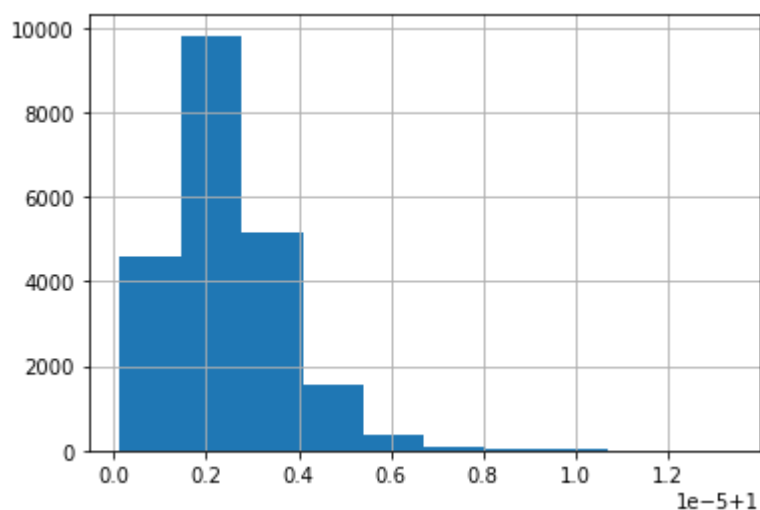
The reciprocal transformation is a powerful transformation with a radical effect. The reciprocal reverses the order among values of the same sign, so large values become smaller. The negative reciprocal preserves the order among values of the same sign.

You should note that this function is not defined for zero.



In [65]:

```
#reciprocal transformation and its plot
```



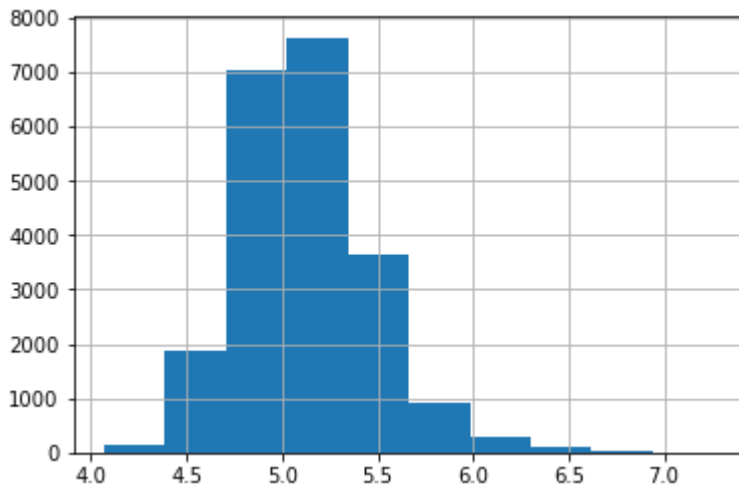
4. Exponential or Power transformation [$f(x) = x^n$ or $f(x) = e^x$]

Power transformation has a reasonable effect on distribution shape; generally, we apply power transformation (power of two usually) to reduce left skewness.

You can use any exponent in this transformation; you can even use the `exp()` function. Try and see which one gives you better results.

In [64]:

#hint: the below output is when we took exponent value as 1/8. You can try and check your



8. Date and Time Engineering

Reference video: <https://youtu.be/RjMbCUpvlgw> (<https://youtu.be/RjMbCUpvlgw>)

Reference link for date formate: <https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes> (<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes>)

Reference link for datetime related

objects: <https://pandas.pydata.org/docs/reference/api/pandas.DatetimeIndex.weekday.html> (<https://pandas.pydata.org/docs/reference/api/pandas.DatetimeIndex.weekday.html>)

10 points

Though date columns usually provide valuable information about the model target, they are neglected as an input or used nonsensically for the machine learning algorithms. It might be the reason for this, that dates can be present in numerous formats, which make it hard to understand by algorithms, even they are simplified to a format like "01-01-2021".

Building an ordinal relationship between the values is very challenging for a machine learning algorithm if you leave the date columns without manipulation. Here, we'll do three types of preprocessing for dates:

- Extracting the parts of the date into different columns: Year, month, day, etc.
- Extracting the time period between the current date and columns in terms of years, months, days, etc.
- Extracting some specific features from the date: Name of the weekday, Weekend or not, holiday or not, etc.

If you transform the date column into the extracted columns like above, the information of them become disclosed and machine learning algorithms can easily understand them.

Now let's understand this using real example of "International football results from 1872 to 2020" dataset :

<https://www.kaggle.com/martj42/international-football-results-from-1872-to-2017>

(<https://www.kaggle.com/martj42/international-football-results-from-1872-to-2017>)

In [35]:

```
# importing datetime
from datetime import date
```

In [36]:

```
#read results.csv
data = #
```

In [37]:

```
#Transform string to date of format "%Y-%m-%d"
data['date'] = #
```

In [38]:

```
#Extracting Year from date
data['year'] = #
```

In [39]:

```
#Extracting Month from date
data['month'] = #
```

In [40]:

```
#Extracting passed years since the current date
#hint: difference of today's year and the year given in date column
data['passed_years'] = #
```

In [41]:

```
#Extracting passed months since the current date
#hint: difference of today's year & month and the year & month given in date column
data['passed_months'] = #
```

In [42]:

```
#Extracting the weekday name from date
data['day_name'] = #
```


In [43]:

```
# Lets check data head for your engineered features
data[['date', 'year', 'month', 'passed_years', 'passed_months', 'day_name']].head()
```

Out[43]:

	date	year	month	passed_years	passed_months	day_name
0	1872-11-30	1872	11	149	1782	Saturday
1	1873-03-08	1873	3	148	1778	Saturday
2	1874-03-07	1874	3	147	1766	Saturday
3	1875-03-06	1875	3	146	1754	Saturday
4	1876-03-04	1876	3	145	1742	Saturday

In [44]:

```
#checking count of each day in dataset
data[['day_name']].value_counts()
```

Out[44]:

```
day_name
Sunday      10243
Wednesday   10021
Saturday     7231
Tuesday      4633
Friday       3970
Thursday     3505
Monday       2481
dtype: int64
```

From above you can see that most of the International football tournament were on Sunday. Such small analysis becomes useful for our machine learning problems to solve.

All these feature engineering techniques become very important to bring good insights and results for any machine learning problems, hence we hope that you understand the importance of feature engineering as part of data science.

Well Done Buddy ! Congrats..you have completed your 8th milestone challenge

FeedBack

We hope you've enjoyed this course so far. We're committed to help you use "AI for All" course to its full potential, so that you have a great learning experience. And that's why we need your help in form of a feedback here.

Please fill this feedback form <https://zfrmz.in/MtRG5oWXBdesm6rmSM7N>
(<https://zfrmz.in/MtRG5oWXBdesm6rmSM7N>).