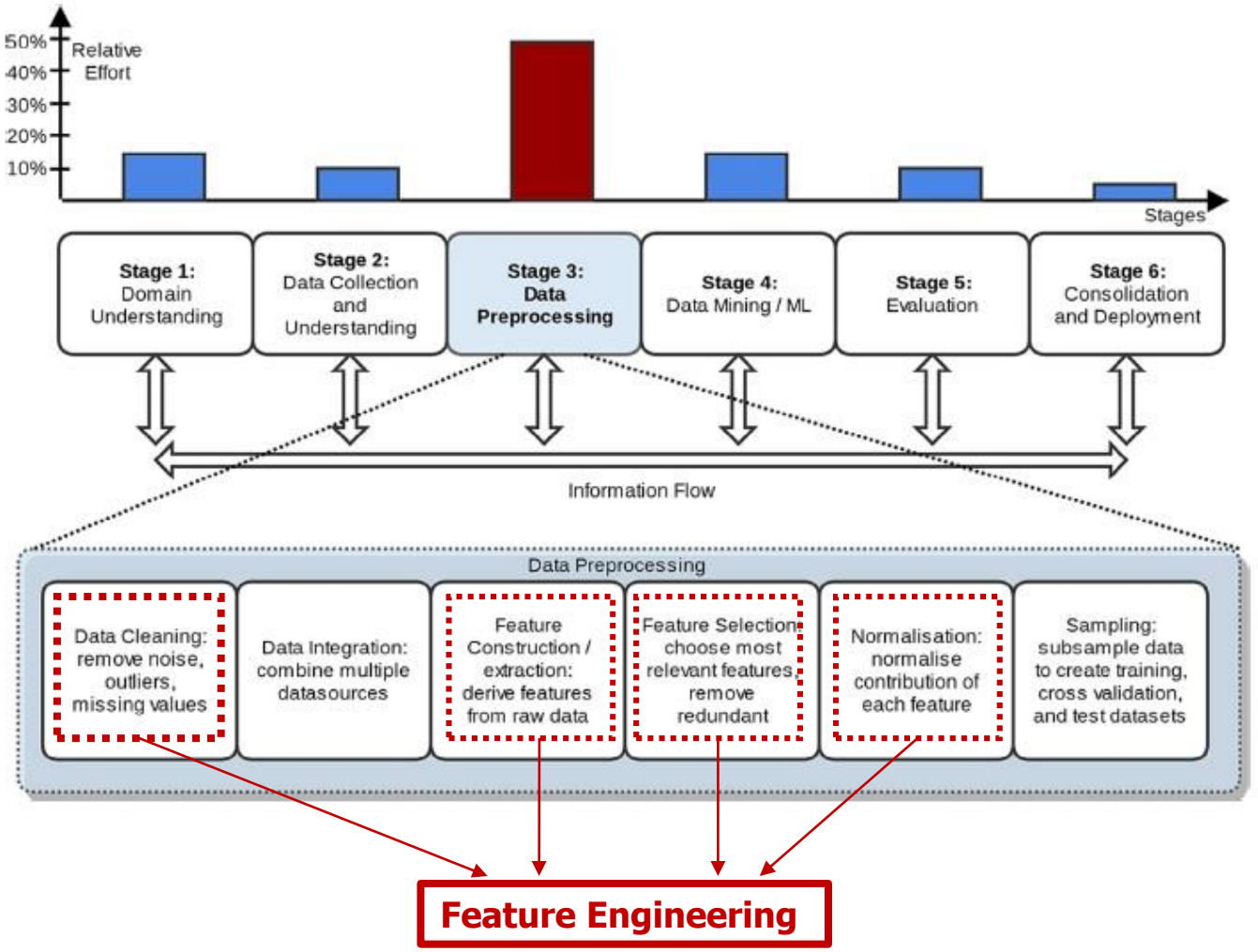
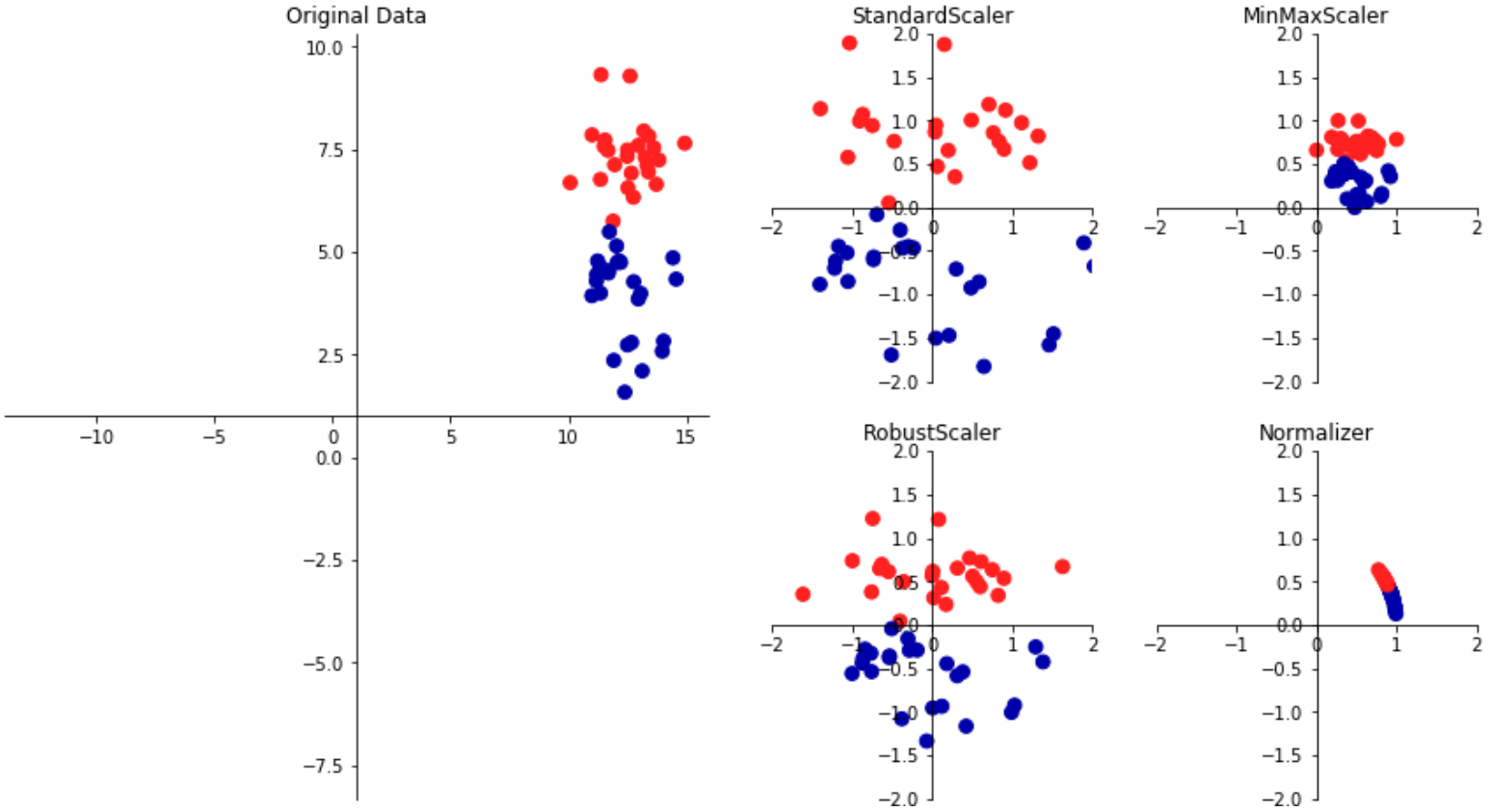
Feature Engineering

Coming up with features is difficult, time-consuming, requires expert knowledge."Applied machine learning" is basically feature engineering. *- Andrew Ng, Machine Learning and AI via brain simulations*



Scale Transformations

- Neural network와 SVM 같은 알고리즘은 feature의 scale에 매우 민감하기 때문에 모델링 전에 scale을 조정하는 전처리 작업이 필요
- scikit-learn은 아래와 같은 다양한 scale 변환 방법을 제공



Scikit-Learn Preprocessor Interface

① Import the preprocessor

② Instantiate the preprocessor

③ Fit the data to the preprocessor: .fit()

④ Generate the preprocessed data: .transform()

The effect of preprocessing on supervised learning

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

```
from sklearn.svm import SVC
svm = SVC(C=100)
svm.fit(X_train, y_train).score(X_test, y_test)
```

```
0.6293706293706294
```

```
# preprocessing using 0-1 scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit(X_train).transform(X_train)
# Scaling training and test data the same way
X_test_scaled = scaler.fit(X_test).transform(X_test)
svm.fit(X_train_scaled, y_train).score(X_test_scaled, y_test)
```

변환 후 0과 1사이의 값을 가짐
outlier에 민감

```
0.8601398601398601
```

```
# preprocessing using zero mean and unit variance scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
svm.fit(X_train_scaled, y_train).score(X_test_scaled, y_test)
```

변환 후 특정 range 안에 반드시 속하지 않음
Outlier에 상대적으로 덜 민감

```
0.951048951048951
```

Feature Selection

- Model based feature selection
  - 지도학습 알고리즘을 사용하여 feature의 중요도를 평가한 후 가장 중요한 feature만 선택
  - Fitting 후 feature_importances_ 혹은 coef_ attribute가 있는 모델 사용 가능
- Iterative feature selection
  - feature를 하나씩 추가(또는 제거)하면서 최적의 feature를 찾는 방법
- Univariate feature selection
  - 개별 feature와 class 간의 유의한 통계적 관계가 있는지를 계산하여 feature 선택
  - 계산이 매우 빠름

Model based feature selection

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel(RandomForestClassifier(), threshold=None)
```

```
X_train_fs = select.fit(X_train, y_train).transform(X_train)
print("X_train.shape: {}, X_train_fs.shape: {}".format(
    X_train.shape, X_train_fs.shape))
```

```
X_train.shape: (426, 30), X_train_fs.shape: (426, 8)
```

```
mask = select.get_support()
plt.matshow(mask.reshape(1,-1), cmap="gray_r")
```

```
<matplotlib.image.AxesImage at 0x27c14668e80>
```

In **numpy.reshape()**, one shape dimension can be **-1**. In this case, the value is inferred from the length of the array and remaining dimensions.

All built-in colormaps can be reversed by appending **_r**: For instance, **gray_r** is the reverse of **gray**.
See color map.

```
X_test_fs = select.transform(X_test)
svm.fit(X_train_fs, y_train).score(X_test_fs, y_test)
```

```
0.6153846153846154
```

Other Tasks of Feature Engineering

- Handling Missing Values
  - 데이터가 NaN을 포함하고 있을 경우 어떻게 대체할 것인가?
- Handling Categorical Variables
  - Character로 입력된 변수를 숫치형으로 변환시키는 작업 필요
- Feature Transformation
  - 그룹별 summary
  - 기존 feature 간의 결합
  - 개별 feature 의 함수적 변환
  - 상호작용과 다항식 추가

Handling Missing Values

- 데이터가 결측치를 포함하고 있는가?
- 데이터가 NaN을 포함하고 있을 경우 어떻게 대체할 것인가?
  - 해당 관측치의 삭제
  - mean/median/mode 등의 값으로 대체

```
dataP.isnull().sum()
```

customer_ID	0
shopping_pt	0
record_type	0
day	0
time	0
state	0
location	0
group_size	0
homeowner	0
car_age	0
car_value	0
risk_factor	34346
age_oldest	0
age_youngest	0
married_couple	0
C_previous	836
duration_previous	836
A	0
B	0
C	0

결측치 개수

Handling Missing Values: Drop

- 관측치가 존재하는 feature들을 subset에 입력하여 삭제

```
dataP_drop=dataP.dropna(subset=['risk_factor','C_previous','duration_previous'])
dataP.shape
```

```
(97009, 60)
```

```
dataP_drop.isnull().any()
```

```
customer_ID          False
shopping_pt          False
record_type          False
day                  False
time                 False
location             False
group_size           False
homeowner            False
car_age              False
car_value            False
risk_factor          False
age_oldest           False
age_youngest         False
married_couple       False
C_previous           False
duration_previous    False
A                    False
B                    False
C                    False
D                    False
E                    False
F                    False
```

결측치가 존재하는
feature만 선택

Handling Missing Values: Impute

- 결측치를 다른 값으로 대체하는 방법
  - 중위수 대체: strategy="median"
  - 평균 대체: strategy="mean"
  - 최빈값 대체: strategy="most_frequent"
- Categorical 변수는 평균과 중위수의 의미가 없음
  - 최빈값 대체가 적당
- sklearn의 imputer 클래스는 numeric type만 처리 가능
  - object type으로 입력된 categorical 변수는 전처리 필요

- Continuous variable의 결측치 대체
  - Imputer 클래스를 가진 object를 생성
  - 생성된 object를 train

```
from sklearn.preprocessing import Imputer
imputer_con=Imputer(strategy="median")
imputer_con.fit(dataP[con])
```

  - imputer를 사용하여 변수를 변환

```
X=imputer_con.transform(dataP[con])
```

  - 변환 후 numpy array로 변환되기 때문에 DataFrame으로 변환시키는 과정 필요

```
dataP_imp=dataP
dataP_imp[con]=pd.DataFrame(X, columns=dataP[con].columns,index=dataP.index)
```

Categorical variable의 결측값 대체

- Character로 코딩된 feature만 추출해서 numeric 으로 인코딩

```
dataP[cat].dtypes
```

```
homeowner          int64
car_value         object
risk_factor      float64
married_couple     int64
C_previous       float64
state             object
location           int64
shopping_pt        int64
dtype: object
```

```
obj=['car_value','state']
```

category type으로 변경

숫자로 encoding

```
dataP[obj]=pd.DataFrame(dataP[obj].apply(lambda x: x.astype('category').cat.codes), index=dataP.index)
```

dataframe으로 변경

- Imputer 학습 후 변환
  - 최빈값으로 대체
  - DataFrame으로 변환

```
imputer_cat=Imputer(strategy="most_frequent")
imputer_cat.fit(dataP[cat])
X=imputer_cat.transform(dataP[cat])
dataP_imp[cat]=pd.DataFrame(X, columns=dataP[cat].columns,index=dataP.index)
```

```
Imputer(axis=0, copy=True, missing_values='NaN', strategy='most_frequent',
    verbose=0)
```

Handling Categorical Variables: One-hot encoding

- Nominal variable에 적용
  - 카테고리가 순서로서의 의미가 없을 경우
  - pd.get_dummies() 사용
- 각 카테고리 별로 dummy 변수 생성

```
dataP_imp=pd.get_dummies(dataP_imp,columns=['day'])
```

```
dataP_imp.filter(like='day').head()
```

column name에 'state'가 들어간 column만 선택

	day_0	day_1	day_2	day_3	day_4	day_5	day_6
8	1	0	0	0	0	0	0
14	0	0	0	1	0	0	0
22	0	0	0	0	1	0	0
26	0	0	0	0	1	0	0
32	0	1	0	0	0	0	0

Handling Categorical Variables: Label encoding

- Ordinal variable에 사용
  - 카테고리가 순서의 의미를 내재하고 있을 때 사용
- 예를 들어, car_value라는 변수에
  - 신차일 때의 가치가 a, b, …, i로 입력되어 있고
  - a<b<…<i 순으로 높은 가치라고 하자.

```
dataP['car_value'].value_counts()
```

```
e    32161
f    25943
d    16402
g    14387
h     4158
c     3072
i      502
b      210
a      174
Name: car_value, dtype: int64
```

- cat.code를 사용하여 변환

```
dataP['car_value']=dataP['car_value'].astype('category')
```

object 타입을 category 타입으로 변환

```
dataP['car_value']=dataP['car_value'].cat.codes
```

alphabet 순서로 숫자 encoding

```
dataP['car_value'].value_counts()
```

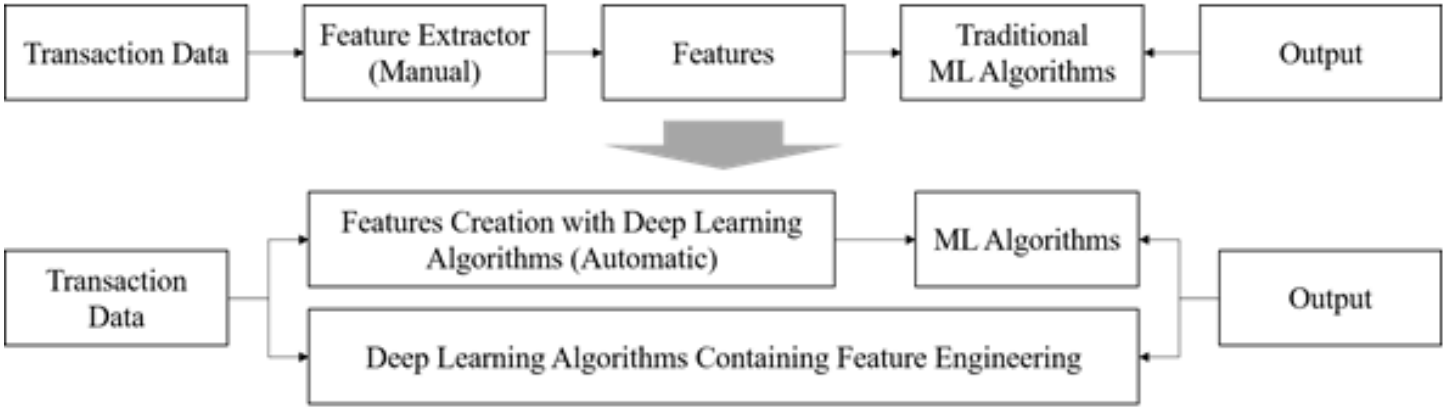
```
4    32161
5    25943
3    16402
6    14387
7     4158
2     3072
8      502
1      210
0      174
Name: car_value, dtype: int64
```

Feature Transformation

- 그룹별 summary
  - Ex) state는 고객이 위치한 36개 주를 나타냄 => 주별로 평균 cost를 계산하여 새로운 feature 생성
- 기존 feature 간의 결합
  - Ex) 1인당 견적을 계산
- 개별 feature 의 함수적 변환
  - np.log, np.sqrt, np.square 등을 사용
- 상호작용과 다항식 추가
  - Ex) 속성 x에 대해 x^2, x^3, x^4 추가

Automated Feature Engineering

- Feature Engineering의 자동화는 머신러닝 분야에서 급부상하고 있는 연구주제



- MIT는 Deep Feature Synthesis라는 알고리즘을 발표 (2015)
  - 온라인 데이터 과학 대회에서 906개 참가팀 중 615개 팀을 이김으로써 그 효과를 입증
  - Featuretools라는 오픈 소스 라이브러리로 제공