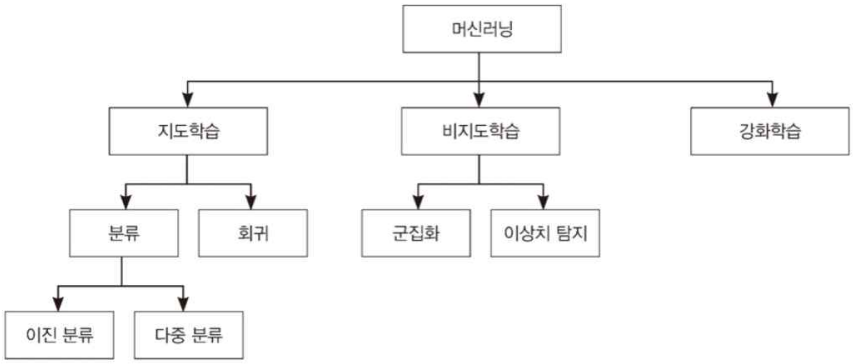
머신러닝

1. 지도학습(Supervised Learning) : 정답이 있는 데이터셋이 있을 때 사용
 · 분류 : 데이터에 따른 정답이 종류를 나타내는 범주일 때 사용
  - k-NN(k-Nearest Neighbors) : 거리를 이용
  - 의사결정 트리 : 트리를 이용한 의사결정
 · 회귀
  - 단순 선형회귀
  - 로지스틱 회귀
 · 정형데이터
 · 머신러닝 프로세스

2. 비지도학습(Unsupervised Learning) : 정답을 모를 때 사용, 데이터의 숨겨진 구조나 패턴 발견
k-means 알고리즘

3. 강화학습(Reinforcement Learning) : 환경과 보상을 기반으로 보상을 최대화하는 최적의 행동을 학습



k-NN : 새로운 데이터가 들어오면 거리를 측정하여 기존 데이터들과 가장 가까운 k개의 이웃 데이터를 찾아 새로운 데이터를 분류하는 방법
거리 측정 방법 : 유클리드거리, 맨해튼 거리
스케일링하는 방법 : 최소 최대 정규화, 표준화

- 데이터 시각화 : countplot(x='카테고리 데이터 속성명', data=데이터 프레임)
- 속성별 개수 : 데이터속성.value_count()
- 상관관계
  → sns.pairplot(data=데이터 프레임, hue=카테고리 데이터 속성명)
  → sns.jointplot(data=데이터 프레임, hue=카테고리 데이터 속성명) -〉 3차원 이상의 데이터(속성 3개 이상)
- 원-핫 인코딩(범주형 데이터) : pd.get_dummies(데이터프레임객체, columns=['인코딩할 속성명'])
- 훈련용_독립변수_객체, 테스트용_독립변수_객체, 훈련용 종속변수 객체, 테스트용 종속변수 객체 = train_test_split(독립변수, 종속변수, test_size=테스트데이터 비율, random_state=숫자)
  → 테스트데이터 비율은 0 ~ 1사이 실수값
  → random_state 지정하지 않으면 랜덤하게 훈련데이터와 테스트 데이터가 분할되므로 지정 필요
- k-NN 모델 : KNeighborsClassifier()
- 결과 확인 : knn모델 객체.score(독립변수, 종속변수) -〉 0 ~ 1 사이 실수 값(정확도)

Q. 펭귄 데이터셋(penguin_size.csv)파일을 이용하여 어떤 종류의 펭귄인지?
1. 데이터

```
import pandas as pd
df = pd.read_csv('penguins_size.csv') # 데이터 불러오기
df
```

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	FEMALE
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	MALE
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	FEMALE
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	MALE

344 rows × 7 columns

```
df.info() # 데이터 기초정보 확인
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                334 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

```
df.describe(include='all').T # 데이터 통계량, T(행렬전치 -> 보기 편하게)
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
species	344	3	Adelie	152	NaN	NaN	NaN	NaN	NaN	NaN	NaN
island	344	3	Biscoe	168	NaN	NaN	NaN	NaN	NaN	NaN	NaN
culmen_length_mm	342.0	NaN	NaN	NaN	43.92193	5.459584	32.1	39.225	44.45	48.5	59.6
culmen_depth_mm	342.0	NaN	NaN	NaN	17.15117	1.974793	13.1	15.6	17.3	18.7	21.5
flipper_length_mm	342.0	NaN	NaN	NaN	200.915205	14.061714	172.0	190.0	197.0	213.0	231.0
body_mass_g	342.0	NaN	NaN	NaN	4201.754386	801.954536	2700.0	3550.0	4050.0	4750.0	6300.0
sex	334	3	MALE	168	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df['sex'].unique() # 성별 속성의 고유한 값 확인
```
```
array(['MALE', 'FEMALE', nan, '.'], dtype=object)
```

```
df[df['sex'] == '.'] # 마침표가 있는 행(줄) 확인
```

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
336	Gentoo	Biscoe	44.5	15.7	217.0	4875.0	.

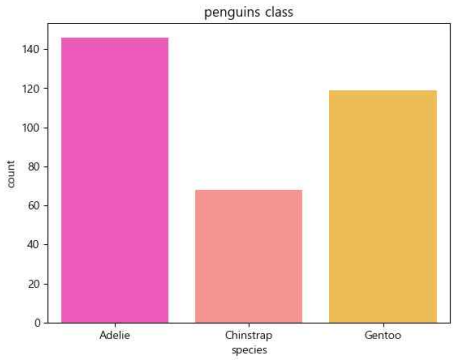
```
df = df.drop(336) # 성별에 마침표가 있는 336번째 행 삭제

df.isna().sum() # 속성별 결측치의 개수
```
```
species               0
island                0
culmen_length_mm      2
culmen_depth_mm       2
flipper_length_mm     2
body_mass_g           2
sex                  10
dtype: int64
```

```
# 결측치 처리
# 수치형 데이터는 평균으로 -> fillna(), 범주형 데이터인 성별은 삭제 -> dropna()
# drop() : 행 또는 열을 직접적으로 삭제 / dropna() : 결측치 값을 가지는 행 또는 열 삭제
df['culmen_length_mm'] = df['culmen_length_mm'].fillna(df['culmen_length_mm'].mean())
df['culmen_depth_mm'] = df['culmen_depth_mm'].fillna(df['culmen_depth_mm'].mean())
df['flipper_length_mm'] = df['flipper_length_mm'].fillna(df['flipper_length_mm'].mean())
df['body_mass_g'] = df['body_mass_g'].fillna(df['body_mass_g'].mean())
df = df.dropna() # 성별의 결측치 10개를 삭제
df.isna().sum()
```
```
species              0
island               0
culmen_length_mm     0
culmen_depth_mm      0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

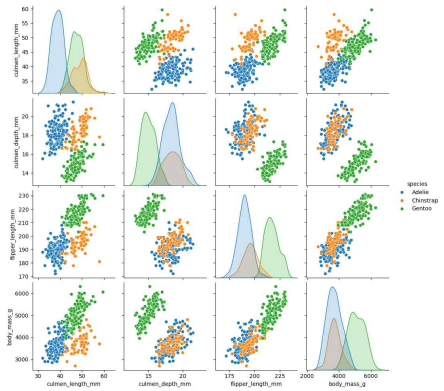
2. 데이터 시각화

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.countplot(x='species', data=df, palette='spring', hue='species')
plt.title('penguins class')
plt.show()
```



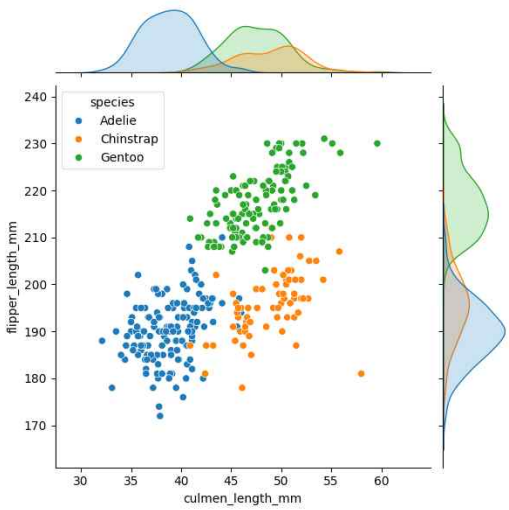

```
df['species'].value_counts() # 펭귄 종의 클래스 개수 파악하기
```

```
species
Adelie       146
Gentoo       119
Chinstrap     68
Name: count, dtype: int64
```



```
# 속성별 상관관계
sns.pairplot(data=df, hue = 'species')
```

```
# x, y축명과 그래프 형태로 산점도를 형성
sns.jointplot(x='culmen_length_mm', y='flipper_length_mm', data=df, kind='scatter', hue='species')
```



3. 모델 학습을 위한 전처리

```
# 범주형 데이터(island, sex) --> 원-핫 인코딩
# island(Biscoe, Dream, Torgersen) / sex(FEMALE, MALE)
penguins_encoding = pd.get_dummies(df, columns=['island', 'sex'])
penguins_encoding.head()
```

	species	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	island_Biscoe	island_Dream	island_Torgersen	sex_FEMALE	sex_MALE
0	Adelie	39.1	18.7	181.0	3750.0	False	False	True	False	True
1	Adelie	39.5	17.4	186.0	3800.0	False	False	True	True	False
2	Adelie	40.3	18.0	195.0	3250.0	False	False	True	True	False
4	Adelie	36.7	19.3	193.0	3450.0	False	False	True	True	False
5	Adelie	39.3	20.6	190.0	3650.0	False	False	True	False	True

4. 독립변수와 종속변수 구분

```
X = penguins_encoding.drop(['species'], axis=1) # species를 제외한 나머지 penguins_encoding
y = penguins_encoding['species'] # species만 추출
```

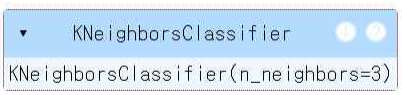
5. K-NN모델 학습 -〉 훈련데이터로 학습한 모델 확인 -〉 테스트 -〉 평가

```
%pip install scikit-learn # 사이킷런 설치

# 훈련 데이터와 테스트 데이터 분리
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=11)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(233, 9) (100, 9) (233,) (100,)
```

```
# k-NN 모델 학습하기
from sklearn.neighbors import KNeighborsClassifier
k = 3 # 하이퍼 파라미터
knn = KNeighborsClassifier(n_neighbors=k) # 모델 생성
knn.fit(X_train, y_train) # 훈련용 데이터를 fit() 함수로 학습 -> 모델 학습
```




```
# 훈련데이터로 학습한 결과 확인 : knn.score(x, y) -> 0.0 ~ 1.0
print(knn.score(X_train, y_train)) # 정확도
```

```
0.9055793991416309
```

```
# 테스트하기 -> 훈련에 사용되지 않은 테스트용 독립변수(X_test)를 이용
y_pred = knn.predict(X_test)
```

```
y_pred[:20] # 20개까지만
```

```
array(['Adelie', 'Gentoo', 'Adelie', 'Chinstrap', 'Gentoo', 'Adelie',
       'Gentoo', 'Gentoo', 'Adelie', 'Gentoo', 'Adelie', 'Gentoo',
       'Gentoo', 'Gentoo', 'Adelie', 'Adelie', 'Adelie', 'Gentoo',
       'Chinstrap', 'Adelie'], dtype=object)
```

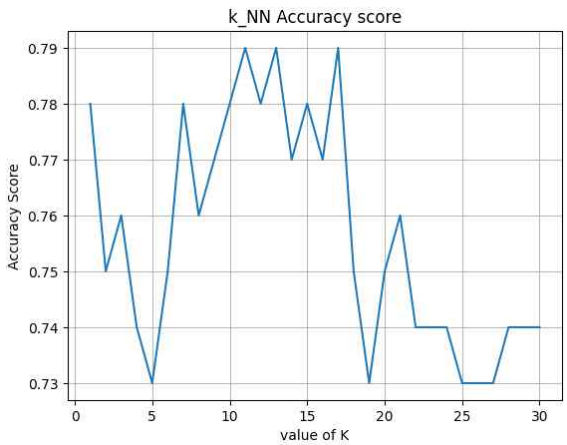
```
# 평가하기
print(f'knn accuracy : {knn.score(X_test, y_test)}')
```

```
knn accuracy : 0.76
```

6. k값에 따른 성능 비교

```
import numpy as np
import matplotlib.pyplot as plt
k_range = range(1, 31) # k값을 1 ~ 30 으로 설정
scores = [] # k값에 따른 정확도를 저장할 리스트 생성
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k) # k=1, 2, ..., 30
    knn.fit(X_train, y_train) # 학습 : fit -> train
    scores.append(knn.score(X_test, y_test))
    # score리스트에 테스트 데이터로 정확도를 산출한 결과 추가

# x축은 k_range, y축은 scores값으로 그래프 작성
plt.plot(k_range, scores) # 꺾은선 그래프
plt.xlabel('value of K')
plt.ylabel('Accuracy Score')
plt.xticks(np.arange(0, 31, 5)) # x축에 5단위로 눈금 표시
plt.grid()
plt.title('k_NN Accuracy score')
plt.show()
```



7. 모델 성능 개선

의사결정트리

범례 레이블 생성

객체.fig.legend(title='범례 제목', handles=범례의 value, labels=범례의 레이블, loc=범례 위치, ncol=범례 표시할 열의 개수)

기존 범례삭제 : 객체.legend.remove()

상관관계 : pd.get_dummies(데이터프레임)

상관관계 분석 : 객체.corr().round()

히트맵으로 상관관계분석 : sns.heatmap(데이터프레임, annot=True/False, square=True/False, cmap='색상명')

독립변수와 종속변수 추출

훈련데이터와 테스트데이터로 분리 : train_test_split(독립변수, 종속변수, test_size=테스트데이터_비율, random_state=숫자)

모델 생성 : DecisionTreeClassifier()

학습 : 객체.fit(독립변수, 종속변수)

훈련데이터로 학습한 결과 확인 : .score(독립변수, 종속변수)

테스트 :

의사결정트리 확인 : plot_tree(, filled=True/False, feature_names=속성명 리스트)

분류 기준을 정하는 방법 - 속성 중요도 : 의사결정트리객체.feature_importances_

사전 가지치기 방식 적용 : 의사결정트리객체=DecisionTreeClassifier(max_Depth=최대깊이)

회귀

선형회귀(Linear Regression)

- 단순선형회귀($y = wx + b$) : 독립변수 1개
- 다중선형회귀($y = w_1x_1 + w_2x_2 + ... + w_nx_n + b$) : 독립변수 2개 이상

Q. advertising.csv에서 광고 플랫폼에 따른 판매량 예측

1. 데이터셋

```
import pandas as pd
filename = r'input\advertising.csv'
ad = pd.read_csv(filename)
ad.sample(3) # 랜덤하게 데이터 3개 불러옴
```

	TV	Radio	Newspaper	Sales
194	149.7	35.6	6.0	17.3
113	209.6	20.6	10.7	20.9
57	136.2	19.2	16.6	13.2

```
ad.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

총 200개의 데이터로 구성
속성 4개
결측치 없음 확인

```
ad.describe().T
```

	count	mean	std	min	25%	50%	75%	max
TV	200.0	147.0425	85.854236	0.7	74.375	149.75	218.825	296.4
Radio	200.0	23.2640	14.846809	0.0	9.975	22.90	36.525	49.6
Newspaper	200.0	30.5540	21.778621	0.3	12.750	25.75	45.100	114.0
Sales	200.0	15.1305	5.283892	1.6	11.000	16.00	19.050	27.0

2. 속성별 상관관계

sns.heatmap(데이터프레임, annot=True/False, cmap='색상명')

- annot : 히트맵에 값 포함여부, 기본값을 False
- cmap : matplotlib colormap 이름 또는 객체

sns.pairplot(data=데이터프레임객체, x_vars=['속성명'], y_vars=['속성명'], height=크기, kind='그래프종류')

- 그래프 종류 : scatter(산점도), kde(커널 밀도 추정), hist(히스토그램), reg(추세선 포함)

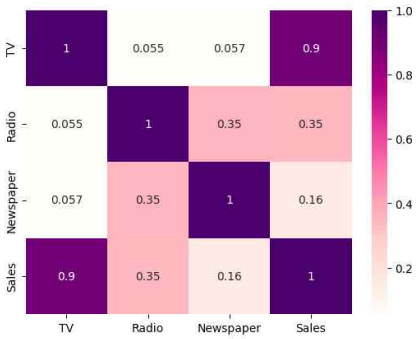
```
# 참고 – cmap 다양한 색상 종류
from matplotlib import cm
cmaps = plt.colormaps()
print(len(cmaps)) # 색상종류 -> 192개
print(cmaps)
```

```
['magma', 'inferno', 'plasma', 'viridis', 'cividis', 'twilight', 'twilight_shifted', 'turbo', 'berlin', 'managua', 'vanimo',
```

```
corrMatrix = ad.corr()
corrMatrix
```

	TV	Radio	Newspaper	Sales
TV	1.000000	0.054809	0.056648	0.901208
Radio	0.054809	1.000000	0.354104	0.349631
Newspaper	0.056648	0.354104	1.000000	0.157960
Sales	0.901208	0.349631	0.157960	1.000000

```
# 상관관계 시각화 -> 히트맵
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(corrMatrix, annot=True, cmap='RdPu')
plt.show()
```

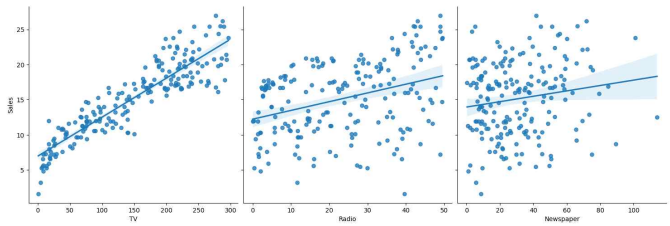



```
# 상관관계 기준으로 -> Sales(판매량) 높은 순서대로 정렬
corr_sort = corrMatrix[['Sales']].sort_values('Sales', ascending=False)
corr_sort
```

	Sales
Sales	1.000000
TV	0.901208
Radio	0.349631
Newspaper	0.157960

```
heatmap = sns.heatmap(corr_sort, annot=True, cmap='twilight_shifted')
heatmap.set_title('Features Correlating with Sales')
plt.show()
```

```
# x축 : tv, 라디오, 신문
# y축 : 판매량
sns.pairplot(data=ad, x_vars=['TV', 'Radio', 'Newspaper'], y_vars=['Sales'], height=5, kind='reg')
```



3. 단순선형 회귀 데이터 나누기

독립변수 객체 = 데이터프레임 객체[[‘속성명1’, ‘속성명2’, ...]]

종속변수 객체 = 데이터프레임 객체[‘속성명’]

```
X_data1 = ad[['TV']] # 독립변수
y_data1 = ad['Sales'] # 종속변수
display(X_data1) # [[]] -> 데이터프레임
display(y_data1) # [] -> 시리즈
```

	TV
0	230.1
1	44.5
2	17.2
3	151.5
4	180.8
...	...
195	38.2
196	94.2
197	177.0
198	283.6
199	232.1

200 rows × 1 columns

```
0      22.1
1      10.4
2      12.0
3      16.5
4      17.9
      ...
195     7.6
196    14.0
197    14.8
198    25.5
199    18.4
Name: Sales, Length: 200, dtype: float64
```

```
# 속성 값의 범위를 일정하게 맞춰주는 방법으로 표준화를 적용
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled1 = scaler.fit_transform(X_data1) # 독립변수를 표준화시켜서 X_scaled1에 새로 저장
X_scaled1
```

```
array([[ 0.96985227],
       [-1.19737623],
       [-1.51615499],
 ...
       [-1.27094056],
       [-0.61703541],
       [ 0.34981006],
       [ 1.59456522],
       [ 0.99320602]])
```

4. 훈련 데이터와 테스트 데이터 나누기

```
from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_scaled1, y_data1, test_size=0.3, random_state=10)
print(X_train1.shape, X_test1.shape, y_train1.shape, y_test1.shape)
```

```
(140, 1) (60, 1) (140,) (60,)
```

5. 모델 생성 및 학습, 평가

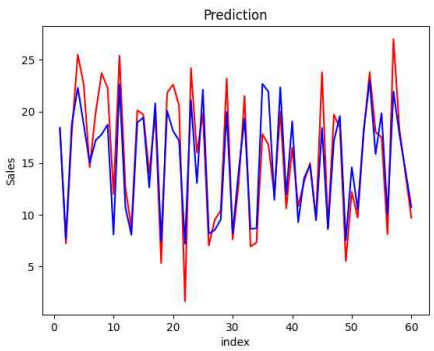
모델 평가 : 평균제곱오차(MSE : Mean Squared Error)와 결정계수(R^2) 사용

- MSE가 0에 가까울수록 좋은 모델(예측값이 실제값과 얼마나 차이가 있는지 평가함)
- R^2는 0 ~ 1사이의 값을 가지며, 1에 가까울수록 모델의 적합도가 좋음을 의미

```
from sklearn.linear_model import LinearRegression
lr_model1 = LinearRegression() # 선형회귀 모델 생성
lr_model1.fit(X_train1, y_train1) # 훈련 데이터로 학습
```

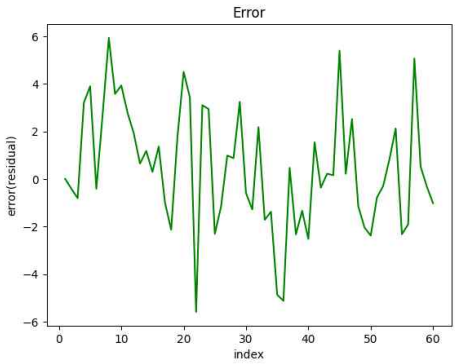
```
▾ LinearRegression
LinearRegression()
```

```
# 실제값과 예측값 시각화 – 테스트 데이터로 성능 평가, 예측
y_pred1 = lr_model1.predict(X_test1) # 예측
c = [i for i in range(1, 61)]
plt.plot(c, y_test1, color='r') # 실제값 꺾은선 그래프
plt.plot(c, y_pred1, color='b') # 예측값 꺾은선 그래프
plt.xlabel('index') # x축 제목
plt.ylabel('Sales') # y축 제목
plt.title('Prediction')
plt.show()
```




```
# 오차 시각화
error = y_test1 - y_pred1
plt.plot(c, error, color='g')
plt.xlabel('index')
```

```
plt.ylabel('error(residual)')
plt.title('Error')
plt.show()
```

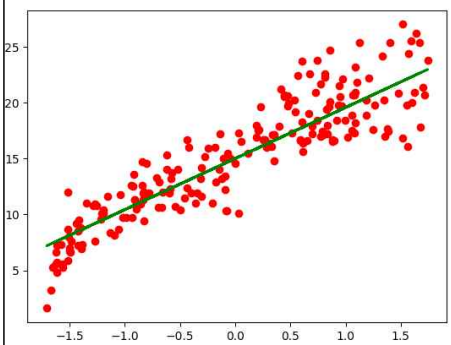



```
# 성능평가 -> MSE(평균 제곱 오차), R^2(결정계수)
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
print(f'MSE : {mean_squared_error(y_test1, y_pred1):.2f}')
print(f'r2_score : {r2_score(y_test1, y_pred1):.2f}')
```

```
MSE : 6.46
r2_score : 0.83
```

6. 판매량 예측 - 회귀선을 그리고 회귀식 구하여 TV광고비와 판매량 사이의 관계 표현

```
plt.scatter(X_scaled1, y_data1, color='r', label='scatter plot') # 산점도
plt.plot(X_test1, y_pred1, color='g', linewidth=2, label='Regression Line') # 직선 -> 회귀선
plt.show()
```



```
# 선형 회귀식 구하기
w1 = lr_model1.coef_ # Slope Coefficients(기울기계수)
print(w1) # 리스트로 출력됨
```

```
[4.57220359]
```

```
print(f'Slope of TV : {w1[0]:.2f}')
```

```
Slope of TV : 4.57
```

```
b1 = lr_model1.intercept_ # Intercept(절편)
print(f'Intercept : {b1.round(2)}')
```

```
Intercept : 15.0
```

7. 다중 선형 회귀

```
X_data2 = ad.drop(['Sales'], axis=1) # 독립변수 -> 3개
y_data2 = ad['Sales']

from sklearn.preprocessing import StandardScaler # 데이터 표준화
scaler = StandardScaler()
X_scaled2 = scaler.fit_transform(X_data2)

from sklearn.model_selection import train_test_split # 훈련, 테스트 데이터 나누기
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_scaled2, y_data2, test_size=0.3,
random_state=10)
print(X_train2.shape, X_test2.shape, y_train2.shape, y_test2.shape)
```

```
(140, 3) (60, 3) (140,) (60,)
```

```
from sklearn.linear_model import LinearRegression # 모델 생성 및 학습
lr_model2 = LinearRegression()
lr_model2.fit(X_train2, y_train2) # 훈련데이터로
```

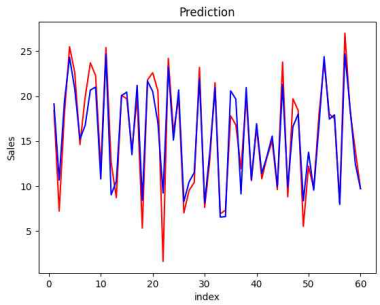
```
▾ LinearRegression
LinearRegression()
```

```
mlr = pd.DataFrame({
    'Actual_value' : y_test2, # 실제값
    'Model prediction' : lr_model2.predict(X_test2) # 예측값
})
mlr.head()
```

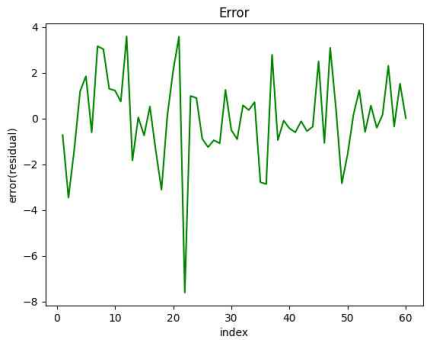
	Actual_value	Model prediction
59	18.4	19.127479
5	7.2	10.658525
20	18.0	19.356496
198	25.5	24.315643
52	22.6	20.751037

```
# 실제값과 예측값 시각화
y_pred2 = lr_model2.predict(X_test2)
c = [i for i in range(1, 61)]
plt.plot(c, y_test2, color='r')
```

```
plt.plot(c, y_pred2, color='b')
plt.xlabel('index')
plt.ylabel('Sales')
plt.title('Prediction')
plt.show()
```




```
# 오차 시각화
error2 = y_test2 - y_pred2
plt.plot(c, error2, color='g')
plt.xlabel('index')
plt.ylabel('error(residual)')
plt.title('Error')
plt.show()
```




```
# 성능 평가 -> MSE, R^2으로
print(f'MSE : {mean_squared_error(y_test2, y_pred2):.2f}')  MSE : 3.66
print(f'r2_score : {r2_score(y_test2, y_pred2):.2f}')  r2_score : 0.90

# 판매량 예측 -> 기울기계수, 절편
w2 = lr_model2.coef_
print(w2)  [4.48690288 1.5923683  0.0094239 ]
print(f'TV : {w2[0]:.2f}')   TV : 4.49
print(f'Radio : {w2[1]:.2f}') Radio : 1.59
print(f'newspaper : {w2[2]:.2f}')  newspaper : 0.01
```

```
b2 = lr_model2.intercept_
print(f'Intercept : {b2.round(2)}') Intercept : 15.13
```

회귀모델 성능 평가지표

· 평균 제곱 오차(MSE) = $\frac{1}{n}\sum_{i=1}^{n}(\text{실제값} - \text{예측값})^2$ (오차 제곱합(RSS)을 데이터개수로 나눠 평균값을 구한 것)

· 평균 제곱근 오차(RMSE) = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{실제값} - \text{예측값})^2}$

· 평균 절대 오차(MAE) = $\frac{1}{n}\sum_{i=1}^{n}|\text{실제값} - \text{예측값}|$

· 결정계수(R^2Score) = $\frac{SSR}{SST} = 1 - \frac{SSE}{SST} = 1 - \frac{\sum(y_i - \hat{y_i})^2}{\sum(y_i - \bar{y})^2}$ (단, SST = SSR + SSE)

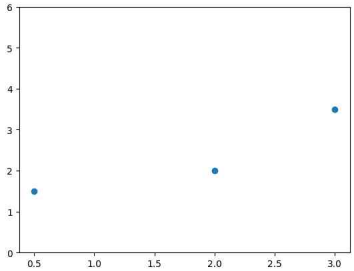
오차를 최소로 만들기 위해 -〉 경사하강법(Grandient descent) : 오차를 최소화하는 수학적 최적화

인공지능이 예측하는 모델 $y = wx$에서 w*라는 이상적인 값(오차를 최소화 할 수 있는 w)을 찾는 방법

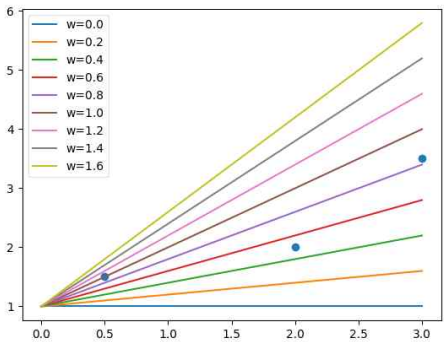
$w_0 \rightarrow w_1 \rightarrow w_2 \rightarrow \cdots \rightarrow w_n \rightarrow w^*$ (w^* 는 오차가 작은 값이 존재할 것이라는 가정에서 시작)

Q. 경사하강법 순서 : (x, y) 3개의 데이터를 좌표평면에 표시, 예측모델 그래프로 나타내기 -〉 모든 데이터의 오차 구하기 -〉 평균제곱오차(MSE)값의 변화 그래프로 나타내기

```
import matplotlib.pyplot as plt
x = [0.5, 2, 3]
y = [1.5, 2, 3.5]
plt.scatter(x, y)
plt.ylim(0, 6) # y축 범위 설정
plt.show()
```



```
# w값의 변화에 따른 직선 그래프 그려보기
import numpy as np
plt.scatter(x, y)
x = np.arange(4)
w_range = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6]
for w in w_range:
    y = w * x + 1
    plt.plot(x, y, label='w=' + str(w))
    plt.legend(loc='upper left')
plt.show()
```

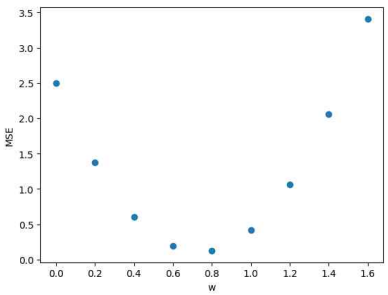



```
# w값의 변화에 따른 MSE 값 구하기
data = [(0.5, 1.5), (2, 2), (3, 3.5)]
w_range = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6]
```

```
result = [] # w=0일 때부터 0.2간격으로 1.6까지 리스트로
b = 1
for w in w_range:
    sum = 0
    n = 0
    for (x, y) in data:
        y_hat = w*x + b # 모델의 예측값
        n += 1
        sum += (y - y_hat)**2 # RSS
    sum /= n # MSE
    result.append(round(sum, 3)) # 소수셋째자리까지 반올림 -> 리스트에 추가
    print(f'w값이 {w}일 때 MSE 값 : {sum:.3f}')
print(result)
```

```
w값이 0.0일 때 MSE 값 : 2.500
w값이 0.2일 때 MSE 값 : 1.377
w값이 0.4일 때 MSE 값 : 0.607
w값이 0.6일 때 MSE 값 : 0.190
w값이 0.8일 때 MSE 값 : 0.127
w값이 1.0일 때 MSE 값 : 0.417
w값이 1.2일 때 MSE 값 : 1.060
w값이 1.4일 때 MSE 값 : 2.057
w값이 1.6일 때 MSE 값 : 3.407
[2.5, 1.377, 0.607, 0.19, 0.127, 0.417, 1.06, 2.057, 3.407]
```

```
# w값에 따른 MSE값의 변화를 그래프로
plt.scatter(w_range, result)
plt.xlabel('w')
plt.ylabel('MSE')
plt.show()
```



로지스틱회귀(Logistic Regression)

- 선형회귀의 변형으로 연속적인 값인 독립변수를 입력받아 종속변수를 이산적인 값으로 분류
- 예측해야하는 값이 연속적인 값이 아니라 성공/실패, 합격/불합격, 양성/음성처럼 이산적인 값의 경우 사용
- S자 모양의 곡선은 직선보다 오차가 작고 데이터의 분포를 잘 파악하는 곡선 -〉 직선을 S자 모양의 곡선으로 변환하는 것을 로짓변환이라고 함(로짓변환을 통해 이진 분류 문제 해결)
- 로지스틱함수 출력값 : $f(x) = \frac{1}{1+e^{-x}}$ (0과 1사이 값)

Q. spam.csv로 스팸문자 구분해내는 방법

1. 데이터셋

불필요한 열 삭제 : 데이터프레임 객체.drop(데이터프레임객체.columns[[열의 번호]], axis=1)

컬럼명 변경 : 데이터프레임객체.rename(columns={'현재컬럼명' : '새로운 컬럼명', ...})

```
import pandas as pd
filename = r'input\spam.csv'
sms = pd.read_csv(filename, encoding='latin-1')
# encoding='latin-1' : csv파일에 사용되는 문자인코딩방식, 텍스트 깨지지 않게 해줌
sms.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
sms = sms.drop(sms.columns[[2, 3, 4]], axis=1) # 불필요한 열 삭제
sms = sms.rename(columns = {'v1' : 'target', 'v2' : 'message'}) # 컬럼이름 변경
sms.head()
```

	target	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
sms.info() # 기본적인 정보 확인
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   target   5572 non-null   object
 1   message  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

스팸데이터는 5572개의 데이터
최종속성 2개
결측치 X
데이터 유형 object(문자형)

2. 워드 클라우드 표현 : 중요도는 단어의 빈도수를 이용(빈도수 많으면 글자 크기 커짐)

워드클라우드 설치 : python -m pip install wordcloud

WordCloud(colormap, width, height, max_words=).generate(생성대상)

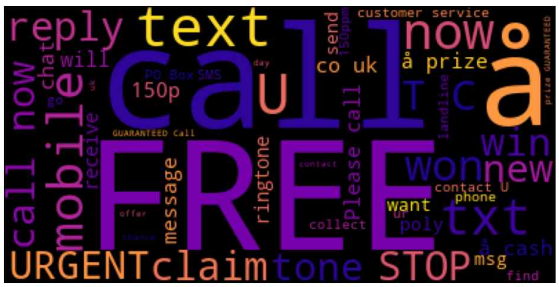
- man_font_size, min_font_size 같은 속성 사용가능

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# target컬럼에서 spam 찾아서 message컬럼의 텍스트만 추출해서 문자열 합치기(공백포함) : ''.join()
spam_words = ' '.join(sms.loc[sms['target'] == 'spam']['message'])
spam_words
```

```
'Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)'
```

```
spam_wc = WordCloud(colormap='plasma', max_words=50).generate(spam_words)
# max_words=50 : 빈도수 상위 50개만 출력
plt.figure(figsize=(24, 6))
plt.axis('off') # x, y축의 눈금 제거
plt.imshow(spam_wc) # 워드 클라우드 출력
plt.show()
```



3. 데이터 전처리 - 텍스트를 숫자로 변환

CountVectorizer(max_features=값) : 고유단어의 빈도수를 기준으로 주어진 문장을 벡터로 변환

단어카운트객체.fit_transform(독립변수).toarray() : 독립변수 속성으로부터 각 단어의 빈도수를 배열형태로 변환

```
from sklearn.feature_extraction.text import CountVectorizer
X = sms['message'] # 독립변수 X
y = sms['target'] # 종속변수 y
cv = CountVectorizer(max_features=2500) # 최대단어의 개수가 2500으로 제한
X = cv.fit_transform(X).toarray() # 텍스트(단어) 수치화(배열형태)
X
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], shape=(5572, 2500))
```

4. 훈련데이터와 테스트데이터 나누기

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(5014, 2500) (558, 2500) (5014,) (558,)
```

5. 모델 생성, 테스트 및 평가

LogisticRegression(solver='liblinear')

- solver : 모델의 최적화에 사용할 알고리즘 지정
- solver='liblinear' : 작은 데이터셋에 적합한 경사하강법 기반의 최적화 알고리즘

accuracy_score() 함수 : 테스트 데이터의 독립변수 값을 넣어 실제 테스트 데이터의 종속변수값과 예측값이 얼마나 일치하는지 정확도 확인

```
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression(solver='liblinear')
LR_model.fit(X_train, y_train) # 훈련데이터로 학습
```

```
LogisticRegression(solver='liblinear')
```

```
y_pred = LR_model.predict(X_test) # 테스트 데이터 예측값
from sklearn.metrics import accuracy_score
print(f'정확도 : {accuracy_score(y_test, y_pred):.3f}')
```

```
정확도 : 0.978
```

6. 혼동행렬 : confusion_matrix()

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[464,   1],
       [ 11,  82]])
```

혼동행렬		예측	
		True	False
실제	True	TP	FN
	False	FP	TN

$$\text{정확도} = \frac{TP+TN}{TP+FP+FN+TN} = \frac{464+82}{464+1+11+82} \fallingdotseq 0.98$$

$$\text{정밀도} = \frac{TP}{TP+FP} = \frac{464}{464+11} \fallingdotseq 0.98$$

$$\text{재현율} = \frac{TP}{TP+FN} = \frac{464}{464+1} \fallingdotseq 0.998$$

정밀도와 재현율은 서로 상호 보완적으로 사용할 수 있으며 두 지표가 모두 높을수록 좋은 모델

+

오즈(odds) = $\frac{P}{1-P} = \frac{\text{성공 확률}}{\text{실패 확률}}$ (실패확률에 대해 성공확률을 비율로 보는 것)

로짓변환 : 오즈에 로그함수 취하면 $-\infty$ ~ ∞로 발산, 0을 기준으로 대칭성 가지게 됨

로짓변환 결과 -> 로지스틱함수 or 시그모이드 함수 : $y = \frac{1}{1+e^{-(wx+b)}}$ (0과 1사이)

- w값이 작아질수록 완만한 S자모양, 오차는 증가
- b값이 너무 크거나 작으면 오차 증가

```
import numpy as np
import matplotlib.pyplot as plt

def logistic_function(x, w, b=0): # 로지스틱 함수 정의
    return 1/(1 + np.exp(-(w*x + b)))

x = np.linspace(-10, 10, 1000) # -10 ~ 10까지 균일하게 1000개로 나눈 배열 생성
w_values = [0.5, 1, 1.5] # w(가중치)
colors = ['red', 'blue', 'green'] # 각 w값에 대한 그래프 색상
plt.figure(figsize=(10, 6))

for w, color in zip(w_values, colors):
    y = logistic_function(x, w) # 로지스틱 함수 호출
    plt.plot(x, y, color = color, label=f'w : {w}')

plt.title('Logistic function for different w values')
plt.legend()
plt.show()
```