DACON

상 점 매 출 예 측 스 터 디

INDEX

DACON PROEJCT

RED PAGE

진행 방향

- 자기소개
- 시계열, 회귀
- 측정지표
- 데이터 파이프라인

RED PAGE

자기 소개

자기소개

회귀와 시계열

시계열

크게, 회귀 안에 있고, 시간을 고려한다는 점이 큰 차이점이다.

회귀

관찰된 연속형 변수들에 대해 두 변수 사이의 모형을 구한 뒤 적합도를 측정해 내는 분석 방법

가정

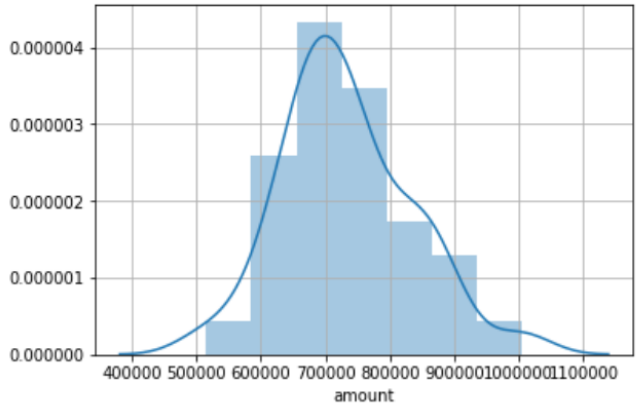
- 선형성 – 모든 독립변수에 한 해 동일한 분산을 갖는다.
- 정규성 – 기댓값은 0이다.
- 독립성 – 독립변수들 간에 상호관계가 없다.

즉, 정규분포를 이룬다.

target 변수의 대칭성(정규성) 확인

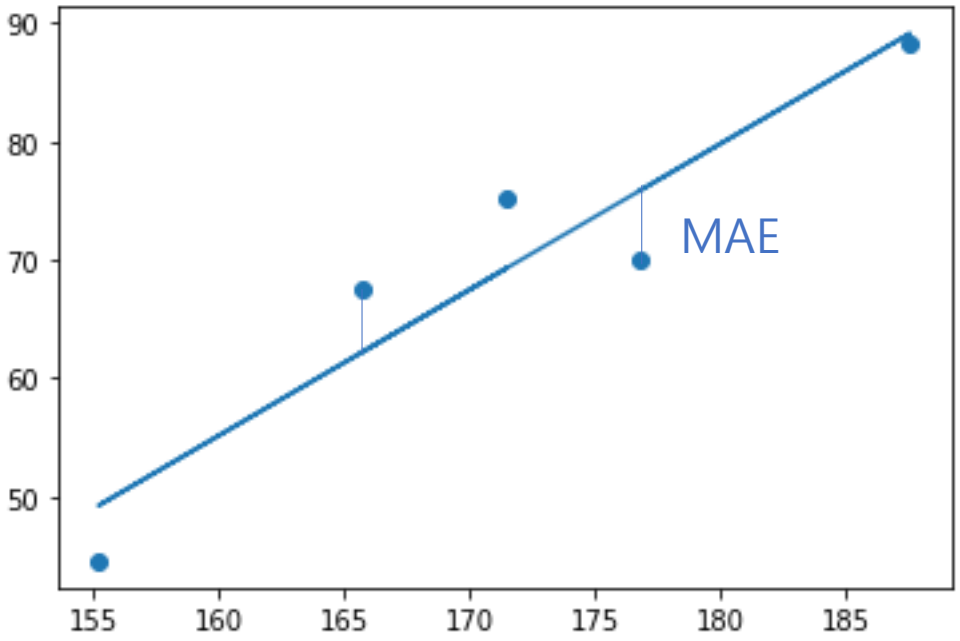
```
In [10]: for i in df_month.store_id.unique()[:3]:
             print("Skewness :", df_month[df_month.store_id == i].amount.skew())
             sns.distplot(df_month[df_month.store_id == i].amount)
             plt.grid()
             plt.show()
```

```
Skewness : 0.48245510908400707
```



MAE 측정지표의 이해

MAE(Mean Absolute Error)



$$MAE=\frac{1}{n}\sum|y-\dot{y}|$$

- 예측값과 실제값 차이를 모두 더하는 개념
- 모델의 under/over perfomance 구분 불가

Under performance : 모델이 실제보다 낮은 값으로 예측.
Over perfromance : 모델이 실제보다 높은 값으로 예측.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.DataFrame({'height':[165.7,171.5,155.2,187.6,176.8],'weight':[67.5,75.3,44.6,88.2,70.1]})

plt.scatter(x=data['height'],y=data['weight'])
line=LinearRegression()
line.fit(data['height'].values.reshape(-1,1), data['weight'])

plt.scatter(data['height'],data['weight'])
plt.plot(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

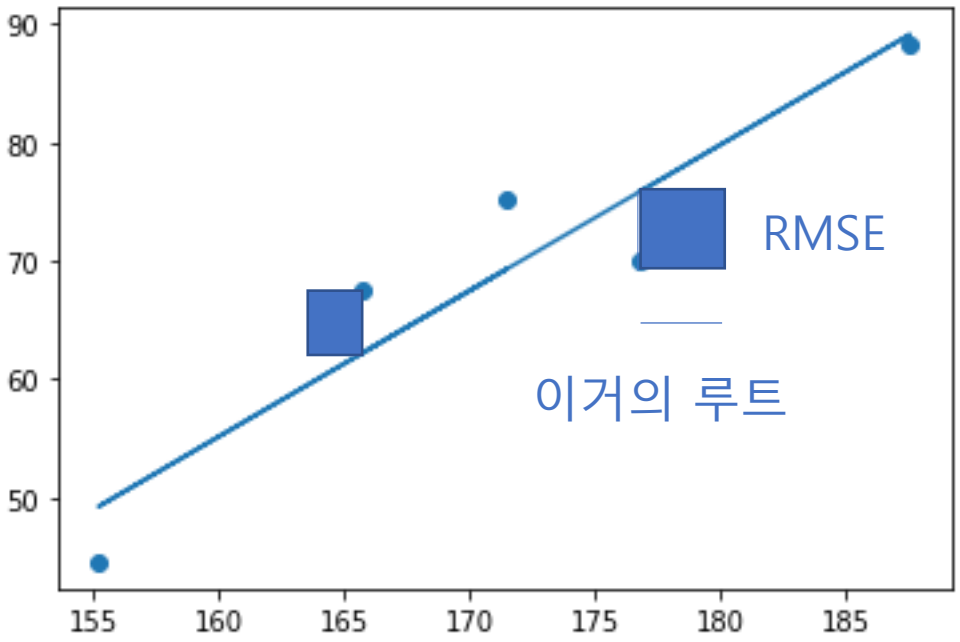
```
In [15]: from sklearn.metrics import mean_absolute_error

         mean_absolute_error(data['height'],line.predict(data['height'].values.reshape(-1,1)))

Out[15]: 102.21999999999997
```

```
In [16]: def MAE(x,y):
             return np.mean(np.abs((x-y)))

         MAE(data['height'],line.predict(data['height'].values.reshape(-1,1)))

Out[16]: 102.21999999999997
```

RED PAGE

RMSE 측정지표의 이해

RMSE(Root Mean Squared Error)



$$RMSE=\sqrt{\frac{1}{n}\sum|y-\dot{y}|^2}$$

- 실제값과 유사한 단위이지만
  에러에 제곱을 하기 때문에 크면 클수록 그에 따른 가중치가 부과됨.
- 에러에 따른 손실이 기하급수적으로 올라가는 상황에 적절함.

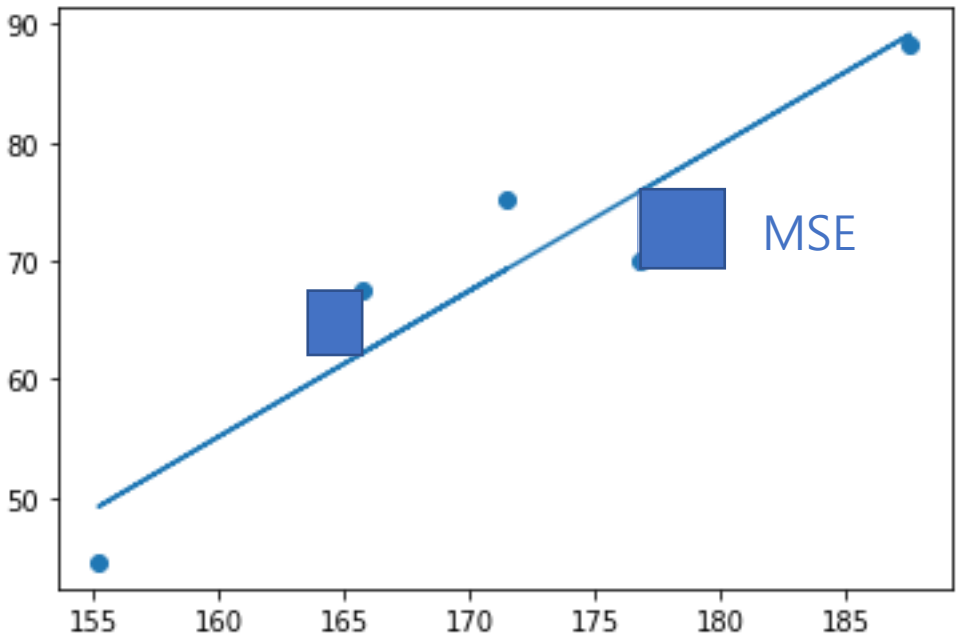
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.DataFrame({'height':[165.7,171.5,155.2,187.6,176.8],'weight':[67.5,75.3,44.6,88.2,70.1]})

plt.scatter(x=data['height'],y=data['weight'])
line=LinearRegression()
line.fit(data['height'].values.reshape(-1,1), data['weight'])

plt.scatter(data['height'],data['weight'])
plt.plot(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

```
In [20]: from sklearn.metrics import mean_squared_error

         mean_absolute_error(data['height'],line.predict(data['height'].values.reshape(-1,1)))**(0.5)

Out[20]: 10.110390694725895
```

```
In [22]: def RMSE(x,y):
             return np.mean(np.abs((x-y)))**(0.5)

         RMSE(data['height'],line.predict(data['height'].values.reshape(-1,1)))

Out[22]: 10.110390694725895
```

MSE 측정지표의 이해

MSE(Mean Squared Error)



$$MSE=\frac{1}{n}\sum|y-\hat{y}|^2$$

- 예측값과 실제값 차이의 면적의 합이다.
- 이상치가 존재하면 수치가 크게 늘어난다. 즉, 이상치에 민감하다.

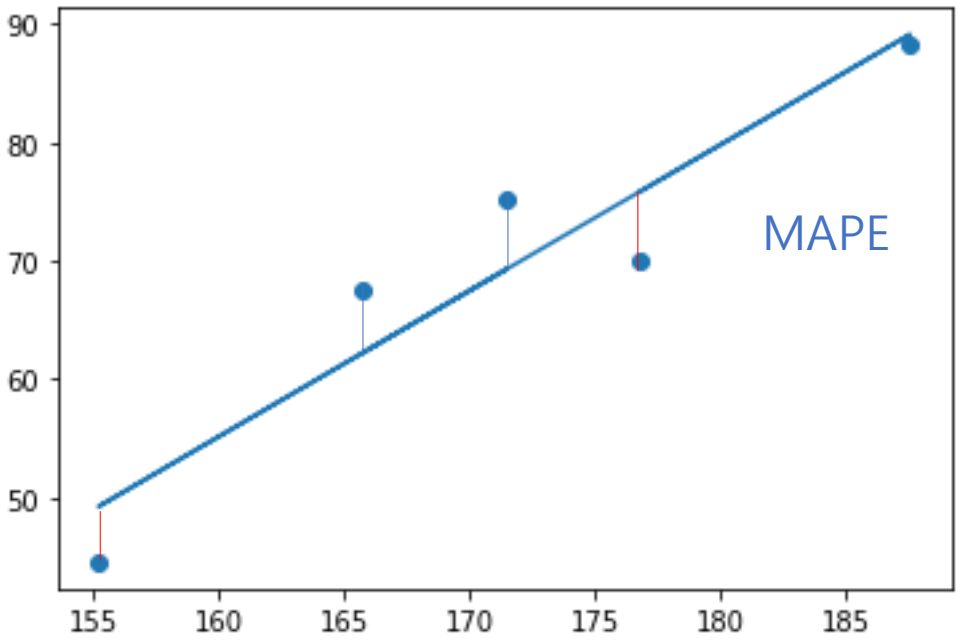
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.DataFrame({'height':[165.7,171.5,155.2,187.6,176.8],'weight':[67.5,75.3,44.6,88.2,70.1]})

plt.scatter(x=data['height'],y=data['weight'])
line=LinearRegression()
line.fit(data['height'].values.reshape(-1,1), data['weight'])

plt.scatter(data['height'],data['weight'])
plt.plot(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

```
In [5]: from sklearn.metrics import mean_squared_error

        mean_squared_error(data['height'],line.predict(data['height'].values.reshape(-1,1)))
Out[5]: 10455.137003825872

In [6]: def MSE(x,y):
            return np.mean(np.square((x - y)))

        MSE(data['height'],line.predict(data['height'].values.reshape(-1,1)))
Out[6]: 10455.137003825872
```

RED PAGE

MAPE 측정지표의 이해

MAPE(Mean Absolute Percentage Error)



$$MAPE = \frac{1}{n}\sum\left|\frac{y-\dot{y}}{y}\right| * 100$$

- 실제값과 예측값의 차이를 실제값으로 나눈 후 절댓값을 씌워 100을 곱해서 구한다.
- 이상치에 대해서 MSE보다 영향을 받지 않는다.
- 하지만 실제값이 0에 가까울수록 무한대가 되므로 사용하기 어렵고, 0이라면 사용할 수 없다.

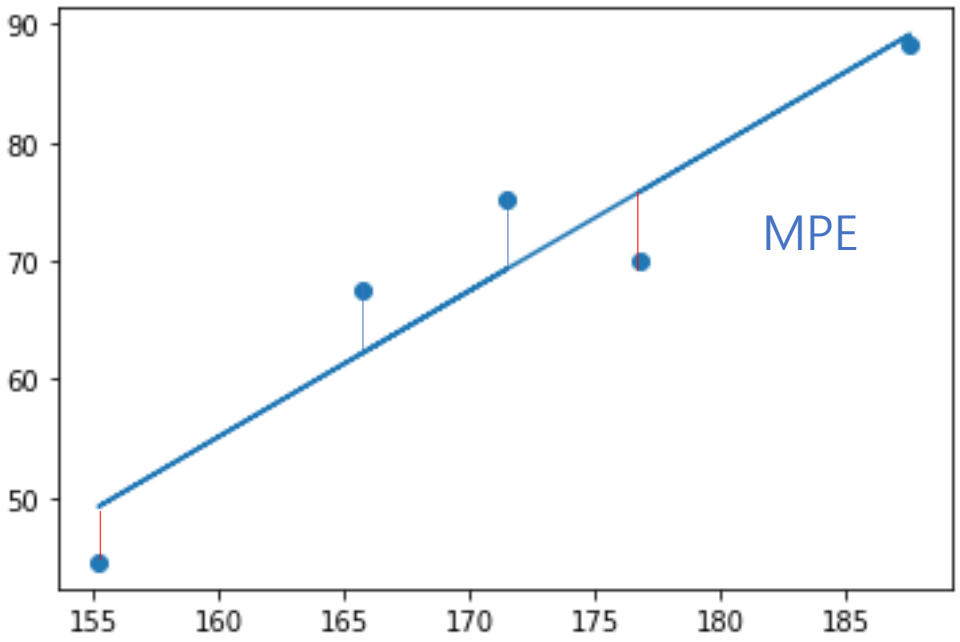
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.DataFrame({'height':[165.7,171.5,155.2,187.6,176.8],'weight':[67.5,75.3,44.6,88.2,70.1]})

plt.scatter(x=data['height'],y=data['weight'])
line=LinearRegression()
line.fit(data['height'].values.reshape(-1,1), data['weight'])

plt.scatter(data['height'],data['weight'])
plt.plot(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

```
In [7]: def MAPE(x,y):
            return np.mean(np.abs((x-y)/x)) * 100

        MAPE(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

```
Out[7]: 59.98496027196878
```

MPE 측정지표의 이해

MPE(Mean Percentage Error)



$$MPE = \frac{1}{n}\sum \frac{y-\dot{y}}{y} * 100$$

- MAPE에서 절댓값을 없앤 방법이다.
- 이 모델의 큰 장점은 예측이 over됐는지, under됐는지 파악할 수 있다는 점이다.
- 즉, 이 모델은 낮게 나왔다는 뜻이다. Under performance 다.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.DataFrame({'height':[165.7,171.5,155.2,187.6,176.8],'weight':[67.5,75.3,44.6,88.2,70.1]})

plt.scatter(x=data['height'],y=data['weight'])
line=LinearRegression()
line.fit(data['height'].values.reshape(-1,1), data['weight'])

plt.scatter(data['height'],data['weight'])
plt.plot(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

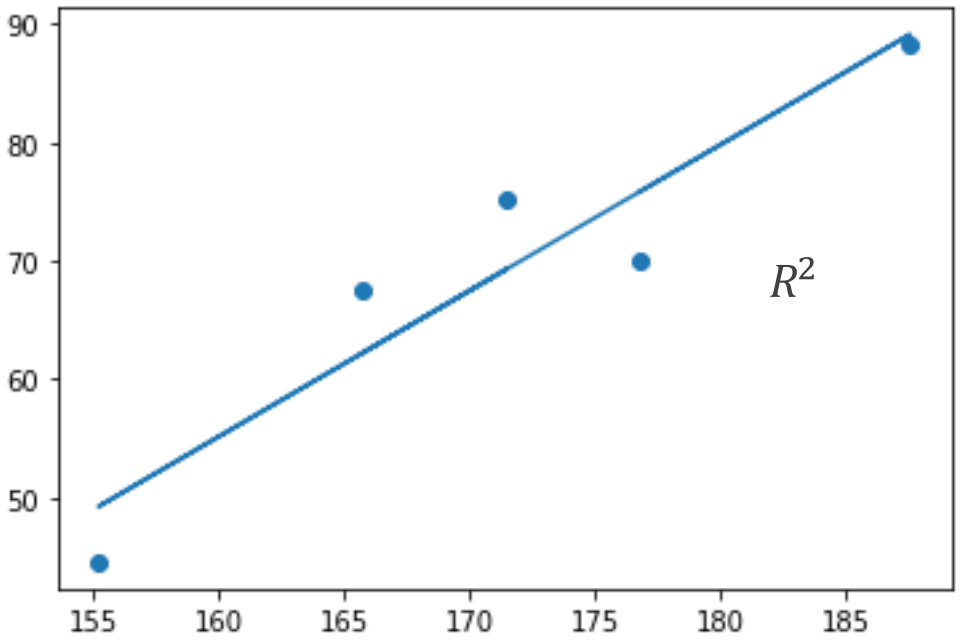
```
In [8]: def MPE(x,y):
            return np.mean((x-y)/x) * 100

        MPE(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

```
Out[8]: 59.98496027196878
```

RED PAGE

R^2 측정지표의 이해

R^2 (R Squared)



$$R^2 = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$$

- 잔차(실제값-예측값)의 제곱을 전체오차(실제값-평균값)의 제곱으로 나눠준 값을 나타낸다.
- 모델을 설명하는 결정계수다.
- 실제값과 예측값의 차이가 작을수록, 실제값과 평균갑의 차이가 클수록 R^2 를 높일 수 있다.
- 독립변수가 얼마나 적절한지 판단할때 사용할 수 있다.

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.DataFrame({'height':[165.7,171.5,155.2,187.6,176.8],'weight':[67.5,75.3,44.6,88.2,70.1]})

plt.scatter(x=data['height'],y=data['weight'])
line=LinearRegression()
line.fit(data['height'].values.reshape(-1,1), data['weight'])

plt.scatter(data['height'],data['weight'])
plt.plot(data['height'],line.predict(data['height'].values.reshape(-1,1)))
```

```
In [27]: from sklearn.metrics import r2_score

         r2_score(data['weight'],line.predict(data['height'].values.reshape(-1,1)))

Out[27]: 0.8812325670664384
```

```
In [26]: def r2(x,y):
             return 1-np.sum((x-y)**2)/np.sum((x-np.mean(x))**2)

         r2(data['weight'],line.predict(data['height'].values.reshape(-1,1)))

Out[26]: 0.8812325670664384
```

DACON 답변

이유

NN dacon_base_line_초급 2019.07.12 00:00 댓글 달기

안녕하세요 [redacted]님,

저희는 대회 기획 단계에서 평가산식을 RMSE로 할지 MAE로 할 지 고민을 했습니다.

잘 알다시피, MAE는 에러에 절댓값을 취하기 때문에, 에러의 크기 그대로 반영됩니다.

그러므로 예측 결과물의 에러가 10이 나온 것이 5로 나온 것보다, 정확히 2배가 나쁜 도메인에서 쓰기 적합한 산식이죠.

즉, 에러에 따른 손실이 선형적으로 올라가는 상황에서 쓰기 적합합니다.

그러나, RMSE는 에러에 제곱을 하기 때문에 에러가 크면 클수록 그에 따른 가중치가 높이 반영됩니다(물론 최종 결과물에 루트를 씌우긴 하지만 여전히 높게 반영됩니다).

그러므로 예측 결과물의 에러가 10이 나온 것이 5로 나온 것보다, 정확히 2^2(4)배가 나쁜 도메인에서 쓰기 적합한 산식입니다.

즉, 에러에 따른 손실이 기하 급수적으로 올라가는 상황에서 쓰기 적합합니다.

저희는 이 점에 착안해, 현재 저희의 스폰서인 펀다가 왜 이 모델을 원하는지를 생각해보았습니다.

펀다는 각 상점별로 얼만큼의 금액을 대출해줄지를 결정하기 위해 매출 예측 모델을 원합니다.

만약 어떤 상점 A와 B가 펀다에게 대출을 요청한 상황에서, 펀다가 미래 3개월간 A 상점은 2000만원의 매출을, B 상점은 3000만원의 매출을 올릴 것이라고 예측했다고 가정합시다.

그런데, 실제로 상점 A는 1500만원, B는 1800만원의 매출을 냈습니다.

이 상황에서 펀다가 손해보는 금액은 ((2000-1500)+(3000-1800))이지 (((2000-1500)**2+(3000-1800)**2)**(1/2))이 아닙니다

RED PAGE

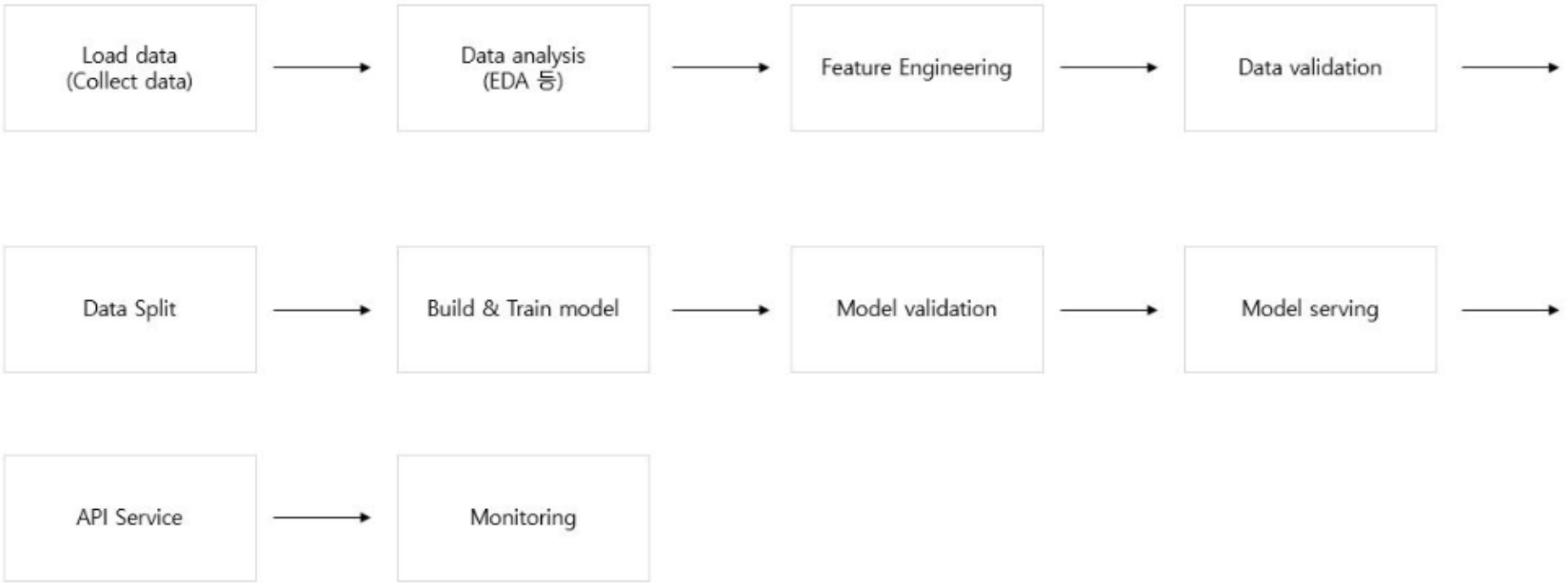
결론

우리는 왜 MAE 측정지표를 사용하는가?

에러의 크기를 그대로 알기 위해서이다.

데이터 파이프라인

데이터 파이프라인



머신러닝 파이프라인 예시

피처 엔지니어링

원-핫 인코딩(edited)

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
393	27.0	4	140.0	86.0	2790.0	15.6	82	1
394	44.0	4	97.0	52.0	2130.0	24.6	82	2
395	32.0	4					82	1
396	28.0	4					82	1
397	31.0	4	119.0	82.0	2720.0	19.4	82	1

여기서 origin은 국가를 나타내는 '범주형' 타입으로 써, 원-핫 인코딩으로 다시 나타낼 수 있다.
즉, 수치화를 하는것이다.

```
origin = dataset.pop('Origin')
# 원 핫 인코딩 범주형 데이터를 이렇게 푼다.
dataset['USA'] = (origin == 1)*1.0
dataset['Europe'] = (origin == 2)*1.0
dataset['Japan'] = (origin == 3)*1.0
dataset.tail()
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	USA	Europe	Japan
393	27.0	4	140.0	86.0	2790.0	15.6	82	1.0	0.0	0.0
394	44.0	4	97.0	52.0	2130.0	24.6	82	0.0	1.0	0.0
395	32.0	4	135.0	84.0	2295.0	11.6	82	1.0	0.0	0.0
396	28.0	4	120.0	79.0	2625.0	18.6	82	1.0	0.0	0.0
397	31.0	4	119.0	82.0	2720.0	19.4	82	1.0	0.0	0.0

문제점은 데이터의 양이 늘어난다는 단점이 있습니다.

피처 엔지니어링

라벨 인코딩 (edited)

이번 코드도 같다. 여기서 dayofweek는 월~일을 0부터 6으로 나타낸것이다. * weekday 와 같은 기능을 한다.

dayofweek	The number of the day of the week with Monday=0, Sunday=6
weekday	The number of the day of the week with Monday=0, Sunday=6

문제점은 일괄적인 숫자로 반환되면서 예측성능이 떨어질 수 있다.
숫자의 크고 작음이 특성으로 작용할 수 있기 때문이다.
또, 카테고리 성질을 갖고있으므로 scikit-learn 에서는 사용할 수 없다. 그래서 바꿔주야한다.

피처 엔지니어링

새로운 피처 생성

```
# 일 단위
data_day = pd.DataFrame()
# 똑같이 store_id 별로 나눈다.
for i in tqdm(data.store_id.unique()):
    data_num = data[data.store_id == i]
    #rule 'day'
    count_cols = data_num['card_id'].resample(rule='d').count().rename('num_of_pay')
    # 재방문 인덱스 확인
    #여기서 reset_index는 인덱스를 없애는게 아니라 새로운 인덱스를 만든다.
    revisit_idx = data_num.card_id.value_counts().reset_index().query("card_id > 2")["index"].values
    revisit_ct = data_num[data_num.card_id.isin(revisit_idx)].card_id.resample(rule='d').count().rename('num_of_revisit')
    # 할부 개월수와 매출액은 일 단위로 합
    sum_cols = data_num[['installment_term', 'amount']].resample(rule='d').sum()
    #
    data_num_day = pd.concat([count_cols, revisit_ct, sum_cols], axis=1)
    #
    data_num_day.insert(0, 'store_id', i)
    data_num_day.insert(4, 'region', data_num[data_num.store_id == i].region.unique()[0])
    data_num_day.insert(5, 'type_of_business', data_num[data_num.store_id == i].type_of_business.unique()[0])
    #
    data_day = pd.concat([data_day, data_num_day], axis=0)
#요일 및 일하는 날,
data_day.insert(1, 'day_of_week', data_day.index.dayofweek)
data_day.insert(2, 'business_day', data_day.day_of_week.replace({0:1, 2:1, 3:1, 4:1, 5:0, 6:0}).values)
data_day.num_of_revisit.fillna(0, inplace=True)

data_day
#data_day.to_csv('./data09/funda_day.csv')
```

```
data_day
#data_day.to_csv('./data09/funda_day.csv')
```

```
100%|██████████| 1967/1967 [06:25<00:00,  5.10it/s]
```

Out[44]:

date_slice	store_id	num_of_pay	num_of_revisit	installment_term	region	type_of_business	amount
2016-06-01	0	4	4.0	0	NaN	기타 미용업	12571.428571
2016-06-02	0	7	3.0	0	NaN	기타 미용업	40571.428571
2016-06-03	0	3	2.0	0	NaN	기타 미용업	18142.857143
2016-06-04	0	7	3.0	0	NaN	기타 미용업	31714.285714
2016-06-05	0	3	3.0	0	NaN	기타 미용업	10428.571429
...	...	...	...	...	...	...	...
2019-02-24	2136	13	1.0	0	제주 제주시	기타 주점업	85357.142857
2019-02-25	2136	7	2.0	0	제주 제주시	기타 주점업	37214.285714
2019-02-26	2136	6	1.0	0	제주 제주시	기타 주점업	47142.857143
2019-02-27	2136	10	1.0	0	제주 제주시	기타 주점업	65071.428571
2019-02-28	2136	13	4.0	0	제주 제주시	기타 주점업	65857.142857

1837047 rows × 7 columns

데이터 resampling

데이터 resampling

시계열 모델링을 위해서
특정 시간 단위 구간별 시계열 데이터를 집계, 요약 하는 방법으로 resample 함수가 있다.

비슷한 함수로 groupby 가 있지만, 시간별 데이터를 다룰땐 resample 함수가 용이하다.

아래 주소는 도움말이다.
https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#time-date-components

```
# 일 단위
data_day = pd.DataFrame()
# 똑같이 store_id 별로 나눈다.
for i in tqdm(data.store_id.unique()):
    data_num = data[data.store_id == i]
    #rule 'day'
    count_cols = data_num['card_id'].resample(rule='d').count().rename('num_of_pay')
    # 재방문 인덱스 확인
    #여기서 reset_index는 인덱스를 없애는게 아니라 새로운 인덱스를 만든다.
    revisit_idx = data_num.card_id.value_counts().reset_index().query("card_id > 2")["index"].values
    revisit_ct = data_num[data_num.card_id.isin(revisit_idx)].card_id.resample(rule='d').count().rename('num_of_revisit')
    # 할부 개월수와 매출액은 일 단위로 합
    sum_cols = data_num[['installment_term', 'amount']].resample(rule='d').sum()
    #
    data_num_day = pd.concat([count_cols, revisit_ct, sum_cols], axis=1)
    #
    data_num_day.insert(0, 'store_id', i)
    data_num_day.insert(4, 'region', data_num[data_num.store_id == i].region.unique()[0])
    data_num_day.insert(5, 'type_of_business', data_num[data_num.store_id == i].type_of_business.unique()[0])
    #
    data_day = pd.concat([data_day, data_num_day], axis=0)
#요일 및 일하는 날,
data_day.insert(1, 'day_of_week', data_day.index.dayofweek)
data_day.insert(2, 'business_day', data_day.day_of_week.replace({0:1, 2:1, 3:1, 4:1, 5:0, 6:0}).values)
data_day.num_of_revisit.fillna(0, inplace=True)

data_day
#data_day.to_csv('./data09/funda_day.csv')
```

피처 엔지니어링

피처 엔지니어링의 의미

피처 엔지니어링은 예측 모델의 정밀도를 향상시키기 위해 원시 데이터의 피처를 변환, 추출, 새로운 피처를 생성하는 작업이다.

구체적인 예시로,

- 명목(normal) 변수의 인코딩(수치화) – 라벨 인코딩,원 핫인코딩(edited)
- 정규화, 그룹화 등 변환 (resampling)
- 결측값 처리 (region 과 type_of_business 처리)
- Data Resampling – 날짜 resampling (월별)

YELLOW PAGE

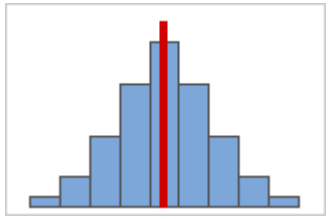
2주차

- 왜도, 변동계수
- 피처 스케일링
- 탐색적 데이터 분석
- 상관계수, 상관 관계

비대칭도

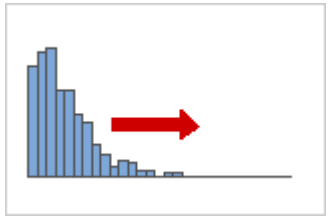
왜도

왜도는 비대칭도를 알려준다. 왜도 값(0, 음수 또는 양수)이 데이터 형상에 대한 정보를 알려준다.

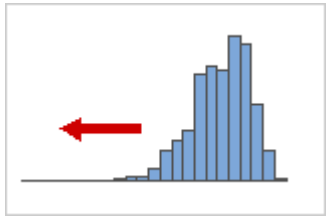


옆 그림은 데이터가 대칭을 이루는데, 왜도 값은 0 이다. 즉, 데이터가 대칭에 가까울 수록 0에 가까워진다.

https://hong-yp-ml-records.tistory.com/28



옆 그림은 데이터가 왼쪽으로 기울었는데 , 왜도 값은 양수이다.
즉, ' 꼬리'가 오른쪽에 있고, 왼쪽으로 기울 수록 왜도가 양수로 간다.



옆 그림은 데이터가 오른쪽으로 기울었는데 , 왜도 값은 음수이다.
즉, ' 꼬리'가 왼쪽에 있고, 오른쪽으로 기울 수록 왜도가 음수로 간다.

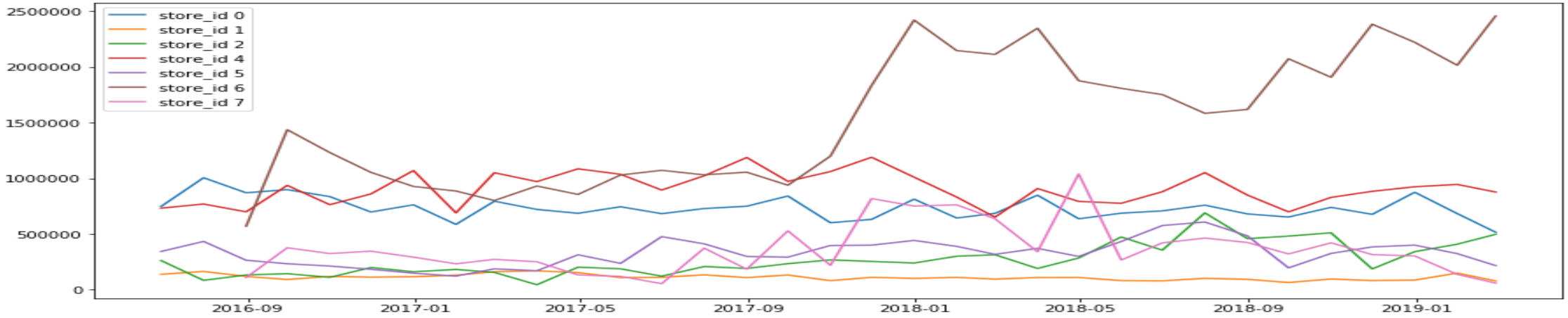
비대칭도

변동계수

변동계수(cv)는 표준편차를 평균으로 나눠줌으로써 계산할 수 있다. 즉, 어느정도 퍼져있나를 볼 수 있다.

변동계수는 서로 다른 두 집단의 자료 분포를 평균의 관점에서 어느 정도 퍼져있나를 대략적으로 확인할 수 있는 값이다.
특히, 두 집단 자료의 측정 단위가 서로 달라 평균이 크게 차이 날 경우에는 두자료의 분포를 직접적으로 비교할 수 없을 때 변동계수를많이 이용한다. 한편,자료의 평균이 0이거나 0에 가까울때는 변동계수가 무한히 커질 수 있기 때문에 사용시 주의해야 한다.

현재, 우리가 따르고 있는 baseline 코드를 보면 각 store_id의 amount 마다 다른 특성을 가지고 있다.
즉, 각 store_id의 amount가 저렇게 된것에는 측정단위가 다르다.



피처 스케일링

피처 스케일링

표준화

각 값들이 평균으로 얼마나 떨어져있는지를 확인하려고 할때 사용.
또, 데이터의 간극을 줄이는데 사용한다.

정규화

각 값들이 같은 중요도로 반영되도록 해주는 것이 정규화의 목적이다.
즉, 전체 구간이 설정되어 데이터 내에서 특정 데이터가 가지는 위치를 볼때 사용한다.

피처 스케일링

피처 스케일링

Standards Scaler(표준화) 기본 스케일,평균과 표준편차 사용.

그러나, 이상치가 있다면, 평균과 표준편차에 영향을 끼쳐 변환된 데이터는 매우 달라져 균형잡힌 척도를 보장할 수 없다.

Min Max Scaler (정규화) 최대 최소가 각각 1,0 이 되게 한다.

이상치가 있다면, 변환된 값이 매우 좁은 범위를 가질 수 있다. 얘도 이상치에 민감하다.

Maxabs Scaler (정규화) 절댓값이 0~1에 있게, 즉 모든값이 -1~1에 있게 한다.

양수 부분은 minmax scaler와 비슷하게 작용하며, 이상치에 민감하다.

Log (정규화) 정규성을 높이고 데이터간 편차를 줄여 왜도를 줄일 수 있기 때문에 사용.

단위수가 너무 큰 값을 작은 변수들과 함께 분석할때 결과 왜곡을 막기위해 사용.

피처 스케일링

결론

특성에 따라 원본데이터의 형태를 유지하는 것이 좋을 때도 있다.
그래서 표준화는 건너띄고 지성민님의 코드는 정규화를 생략하지만
1등 코드에서 부분 정규화를 하므로 정규화에 대해서 공부할 것이다. (생각)

3주차

- box-cox 변환
- 단순이동평균
  가중이동평균
  지수이동평균
- AR,MA,ARMA,ARIMA
- 단위근검정

Boxcox 변환

box-cot 변환

x 에 대해, $g(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda}, \lambda \neq 1 \\ \log(x), \lambda = 0 \end{cases}$

박스-칵스 변환이라고 한다.

박스-칵스 변환의 주된 용도는 데이터를 정규분포에 가깝게 만들거나, 데이터의 분산을 안정화하는 것이다.
그래서, 정규성을 가정하는 분석이나, 정상성을 요구하는 분석을 사용하기 앞서, 데이터 전처리에 사용한다.
그래서, 주로 λ 0, ½, 1 이 있다. $=(\lambda$ 일때, log 함수로, 1일 때는 항등함수, ½ 일때는 제곱근함수이다.
그래서 λ 1일때, 정상성을 갖는다고 한다. 여기서 정상성은 뚜렷한 추세가 없이 시계열의 평균이 시간축과 평행한 상태를 얘기한다. 아까 위에서 말했듯이 데이터의 분산을 안정화하는 용도로 사용된다. 분산이 크다는 것은 평균에서 멀리떨어져있고, 편차가 크다는 말을 의미한다. 그러니까 뒤로 갈수록 크기가 커지는 우리가 다루는 몇몇 'amount'가 보여주는 것이 그렇다.

```
from scipy.stats import boxcox

data_month.amount=data_month.amount.apply(lambda x:x+2 if x==0.0 else x)
data_box=pd.DataFrame(columns=['optimal_lambda'])

for i in tqdm(data_month.store_id.unique()):
    sub_=data_month[data_month.store_id==i].amount
    _, l = boxcox(sub_)
    data_box.loc[i,'optimal_lambda']=l
```

```
data_an=pd.concat([data_cv,data_box],axis=1)
data_an.head(10)
```

	cv	optimal_lambda
0	0.136249	-0.0557794
1	0.237716	-0.00245897
2	0.538133	0.2769
4	0.155397	0.51382
5	0.359291	0.552884
6	0.371041	0.481652
7	0.630949	0.306357
8	0.385799	0.915679
9	0.484134	0.411813

단순 이동 평균

단순 이동 평균(Simple Moving Average)

'평균' 앞에 '이동'이 붙은 것은 새로운 데이터가 생성 될때 마다 사용하는 데이터 범위가 이동하기 때문이다.

최근 n개의 관측값을 이용하여 계산한 이동평균 이다.

관측값은 동일한 가중치를 부여한다.

$$sma_t = \frac{Z_t + Z_{t-1} + Z_{t-2} + \dots + Z_{t-(n-1)}}{n}$$

	amount	amount_2ma	amount_3ma
0	7.470000e+05	NaN	NaN
1	1.005000e+06	876000.000000	NaN
2	8.697143e+05	937357.142857	873904.761905
3	8.978571e+05	883785.714286	924190.476190
4	8.354286e+05	866642.857143	867666.666667

가중 이동 평균

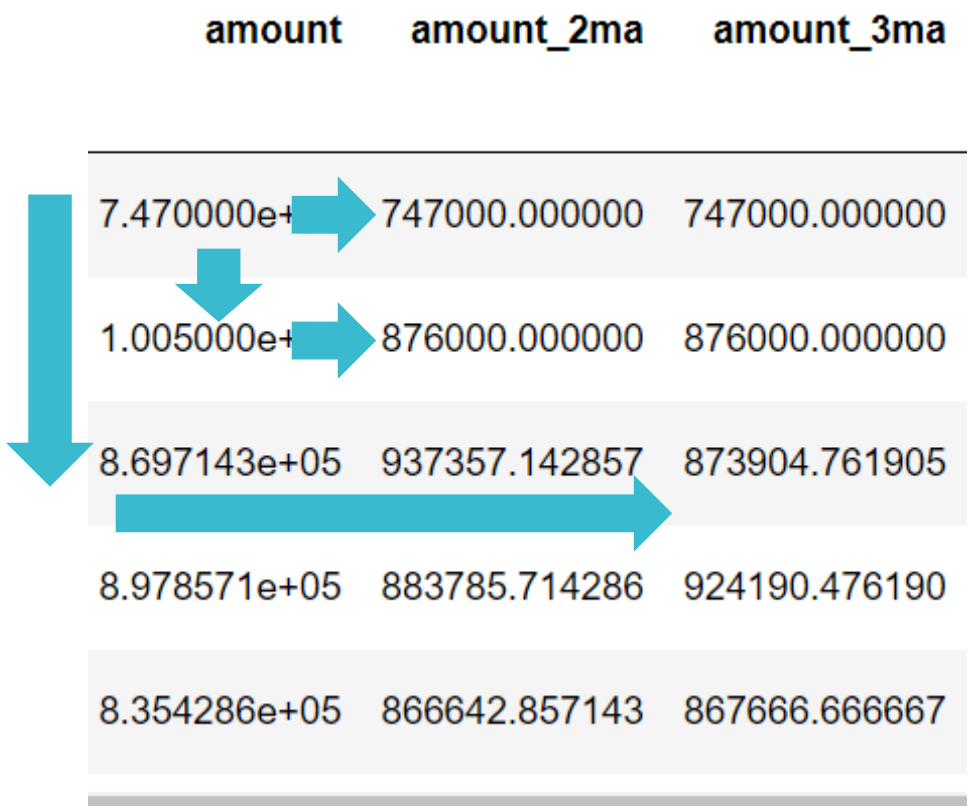
가중 이동 평균(Weighted Moving Average)

가중치 w_i를 각 데이터에 곱한 후 가중치의 합으로 나눠 계산한다.

기간에 따라 데이터의 가중치를 조정함으로써 단순이동평균의 단점을 보완한다.

하지만, 특정 기간에 나타나는 데이터의 변화만 반영하는 문제점이있다.

$$WMA_t = \frac{Z_t W_t + Z_{t-1} W_{t-1} + \dots + Z_{t-(n-1)} W_{t-(n-1)}}{W_t + W_{t-1} + \dots + W_{t-(n-1)}}$$



amount	amount_2ma	amount_3ma
7.470000e+	747000.000000	747000.000000
1.005000e+	876000.000000	876000.000000
8.697143e+05	937357.142857	873904.761905
8.978571e+05	883785.714286	924190.476190
8.354286e+05	866642.857143	867666.666667

지수 가중 이동 평균

지수 이동 평균(Exponential Moving Average)

가중 이동 평균중에 가장 추세를 잘 따르는 가중 이동 평균이다.

보면, 가중을 두는 것이 최근데이터 그리고 전 모든 데이터이다.

즉, 과거의 데이터를 버리지 않고 최근의 데이터를 신경쓰기 때문에 과거의 데이터는 점차 작아진다.

*여기서 N은 지수이동평균 일 수 이다.

$$EMA_t = EMA_{t-1} * \left(1 - \frac{2}{N+1}\right) + Z_t * \frac{2}{N+1}$$

amount	amount_3ema	amount_6ema
7.470000e+05	7.470000e+05	7.470000e+05
1.005000e+06	9.190000e+05	8.975000e+05
8.697143e+05	8.908367e+05	8.850092e+05
8.978571e+05	8.945810e+05	8.899718e+05
8.354286e+05	8.640507e+05	8.708287e+05
...	...	...
2.012214e+06	2.094633e+06	2.086559e+06
2.127643e+06	2.111138e+06	2.098317e+06
2.427429e+06	2.269283e+06	2.192461e+06
1.867786e+06	2.068534e+06	2.099618e+06
2.227429e+06	2.147982e+06	2.136157e+06

AR 과 MA 모형

AR 모형(Auto Regressive model)

보통 표현을 AR(p) 라고 표현하며, p 시점 이전의 데이터가 영향을 준다.

$$Z_t = Z_{t-1}\Phi_1 + Z_{t-2}\Phi_2 + \dots + Z_{t-p}\Phi_p + a_t$$

Φ는 자기회귀계수이고, a_t는 색 잡음이다.

AR 모형에서 어느시점에서 자기상관함수(ACF, Auto correlation function)는 빠르게 감소하고, 부분자기함수(PACF, Partial Auto correlation function)은 절단점을 가진다.

MA 모형(Moving Average model)

MA(q)라고 표현을 하며, q 시점이 전의 백색잡음이 영향을 준다.

$$Z_t = a_t + \theta_1 a_{t-1} + \theta_2 a_{t-2} + \dots + \theta_q a_{t-q}$$

θ는 이동평균계수이다.

MA 모형에서 어느시점에서 자기상관함수(ACF, Auto correlation function)는 절단점을 가지고, 부분자기함수(PACF, Partial Auto correlation function)은 빠르게 감소한다.

ARMA 모형

ARMA(p,q)로 표현한다.

AR과 MA의 모형을 모두 가지고 있다고 생각하면 된다.

ARIMA 모형(Autoregressive Integrated Moving Average)

자기회귀(AR)와 이동평균(MA)를 모두 고려하는 모형인데 ARMA와 ARIMA의 차이점은 비정상성을 설명하기위해 데이터의 차분을 이용한다는 점이다. ARIMA(p,d,q) 라고 표현하며, 실제 시계열 데이터는 하나의 경향을 강하게 띄기 때문에 $p+q<2$, $p*q=0$ 인 조건을 통상 사용한다고 한다. p,q 는 ACF와 PACF를 통해 추정할 수 있다. 한편 차분은 실제값과 연이은 실제값의 차이다. 이 차분을 객관적으로 구할 필요가 있을 때 사용하는것이 단위근 검정이다.

BLUE PAGE

단위근 검정

p_value https://adnoctum.tistory.com/332

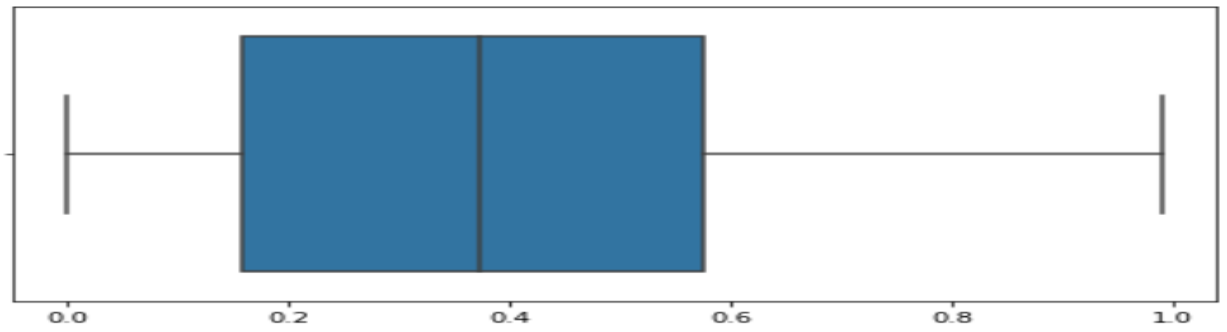
단위근 https://chukycheese.github.io/translation/statistics/augmented-dickey-fuller-test/

단위근 검정

단위근은 AR이 포함된 모형에 관련이 있다.
쉽게, AR(1)인 모형을 생각하면 $Z_t = Z_{t-1}\Phi_1 + a_t$
이다. AR(2)는 $Z_t = Z_{t-1}\Phi_1 + Z_{t-2}\Phi_2 + a_t$
것이다. 여기서 Z_t대해 정리하면 와 Z_t는 Z_{t-1}
L차이가 있을 것이다. 만 L차의 비율을 갖고 둘의
차이가 있다. AR(1) 모형은 현재와 과거의 데이터는
관련이 있기 때문이다. 즉 좌변에 정리하면
$Z_t(1-\Phi_1 L) = a_t$인데 $L_{(\Phi)} = 1 - \Phi L$
을 만족하는 $L = \frac{1}{\Phi}$ 의 근이라고 한다. 여기서
$|L| > 1$만족해야 하므로, 즉 $|\Phi| < 1$이다.
하지만, 단위근이 존재하면 $\Phi = 1$
여기서, p-value가 나오는데 p_value에 의해 $\Phi = 1$
이냐, $|\Phi| < 1$를 인지 가리는 것이 ADF 검정이다.
p_value는 귀무가설이 맞다는 전제하에, 관측된 통계값
혹은 그 값보다 큰 값이 나올 확률이다.
귀무가설은 단위근이 있다. $\Phi = 1$으로 표현하고,
대립가설은 정상성을 만족한다. $|\Phi| < 1$으로 표현
한다.

```
from pmdarima.arima.stationarity import ADFTest

def time_series(data,i):
    ts=pd.Series(data[data.store_id==i].amount.values)
    return ts

def adf_test(y):
    adf_test = ADFTest(alpha=0.05)
    try :
        p_val, should_diff = adf_test.should_diff(y)
    except:
        # 막히는 경우 자료가 2개 밖에 없어서 p_val=0 으로 했다.
        # p_val=0 이라는 뜻은 통계값보다 더 큰 값이 나올 확률이 0이라는 것이다.
        # 즉, 그대로 가겠다는 뜻이다.
        p_val=0
        pass
    return p_val

adf_p=[]
for i in tqdm(data_month['store_id'].unique()):
    ts=time_series(data_month,i)
    adf_p.append(adf_test(ts))

plt.figure(figsize=(8, 4))
sns.boxplot(adf_p)
```

```
100%|| 1967/1967 [00:07<00:00, 280.60it/s]
<matplotlib.axes._subplots.AxesSubplot at 0x25959541288>
```



AIC

— AIC(아카이케 정보 기준)

원래 AIC, BIC는 회귀모형의 성능을 비교할때 선택 기준이다. 두 선택 기준 모두 통계 모델의 상대적인 품질을 평가하는 데 값이 낮을 수록 좋다. 먼저 AIC의 공식은 다음과 같다.
여기서 는 $AIC = 2k - 2\ln(L^*)$ 적합도를 나타낸다. 여기서 별이 붙은 이유는 이 $-2\ln(L^*)$ 어느 상수라는 뜻이다. AIC는 낮을 수록 좋다고 했다. 그 말은 적합도가 높아야 한다는 뜻이다. 적합도를 높이려면 여러 불필요한 독립변수들(parameters)을 사용할 필요가 있다. 이러한 경우를 방지하기 위해 penalty를 준것이 파라미터 개수가 주어진다.

ARIMA Model Results

Dep. Variable:	D2.y	**No. Observations:**	31
Model:	ARIMA(0, 2, 2)	**Log Likelihood**	-359.941
Method:	css-mle	**S.D. of innovations**	23989.092
Date:	Wed, 01 Jul 2020	**AIC**	727.882
Time:	14:35:59	**BIC**	733.618
Sample:	2	**HQIC**	729.752

	coef	std err	z	P>|z|	[0.025	0.975]
const	50.2345	174.111	0.289	0.773	-291.017	391.486
ma.L1.D2.y	-1.7254	0.290	-5.943	0.000	-2.294	-1.156
ma.L2.D2.y	0.7255	0.277	2.620	0.009	0.183	1.268

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	1.0001	+0.0000j	1.0001	0.0000
MA.2	1.3783	+0.0000j	1.3783	0.0000

BIC

— BIC(베이시안 정보 기준)

BIC도 AIC와 마찬가지로 회귀모형에서 선택기준으로 사용되고 공식으로는 이와 같다. $BIC = k\log(n) - 2\ln(L^*)$ AIC와 다른점은 $k\log(n)$전통 2k보다 크므로 BIC가 AIC보다 penalty를 더 준 것으로 알 수 있다. 여기서 n은 데이터의 개수이다. 그리고 앞의 AIC,BIC의 점수가 달라진 것을 알 수 있는데 P>|z|가 학습의 적정성을 위해 확인 되는 t-test값이다. 즉, p-value 유의수준 95%에서 보면 MA(1)의 계수는 유효합니다. 100%에서 봐도 유효하다고 할 수 있습니다. 값만 보면. MA(2)의 계수는 p-value 유의수준 약 90%에서 유효하다고 볼 수 있다. 앞에서 결과를 보면 constant 에서는 유효하지 않다. 상수 측면에서 봤을 때, p-value가 0.05보다 훨씬 큰 0.7이므로 안정적이지 못하다는 뜻이다. 당연하다 store_id 마다 특성이 다른데 어느 기준으로 정규화를 못해줬기 때문이다. 그래서 model.fit() 에서 파라미터를 trend='nc' 로 놓고 했다. 알 수 있는점은 모델의 적합한 정도는 AIC가 BIC가 아닐 수 있다는 것을 알아야 한다.

ARIMA Model Results

Dep. Variable:	D2.y	**No. Observations:**	31
Model:	ARIMA(0, 2, 2)	**Log Likelihood**	-359.987
Method:	css-mle	**S.D. of innovations**	23973.412
Date:	Wed, 01 Jul 2020	**AIC**	725.973
Time:	14:44:12	**BIC**	730.275
Sample:	2	**HQIC**	727.375

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1.D2.y	-1.7391	0.303	-5.743	0.000	-2.333	-1.146
ma.L2.D2.y	0.7391	0.289	2.555	0.011	0.172	1.306

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	1.0000	+0.0000j	1.0000	0.0000
MA.2	1.3529	+0.0000j	1.3529	0.0000

THANK YOU

감 사 합 니 다

PPT PROEJCT