수요예측
피식회

머신러닝과 딥러닝을 이용한

구내식당 수요예측

MENU

Appetizer

Panel
Topic
Plan

Main

Data Processing
EDA
Machine learning
Deep learning
Result

Dessert

Insight
Review

Panel


수요
미식회
20대 입맛 대표
맛 ★★★★
분위기 ★★★★½
재방문 의사 ★★★★½
50대 입맛 대표
맛 ★★★½
분위기 ★★½
재방문 의사 ★★★½
40대 입맛 대표
맛 ★★★½
분위기 ★★
재방문 의사 ★★½

Topic

DACON

구내식당 식수 인원 예측 AI 경진대회

정형 | 한국토지주택공사 | 식수예측 | MAE

- ☑ 머신러닝, 딥러닝 모델링에 적합한 주제
- ☑ 생활과 밀접한 관련이 있는 주제
- ☑ 팀원들이 흥미를 가지는 주제

Plan

프로젝트 일정~

9/8(수): 주제 선정 및 기획안 작성

9/9(목) ~ 9/10(금): 데이터 전처리

9/13(월): 시각화, 모델 학습 계획

9/14(화) ~ 9/15(수): 모델 학습 및 인사이트 도출

9/16(목): 발표자료 마무리 및 연습


수요예측
미식회


RECIPE

Preprocessing

Tool

RANDOM FOREST

dmlc XGBoost

LightGBM

CatBoost

Process

1 Preprocessing
중복제거, NA값 대치, 모든 컬럼 수치화

2 Feature engineering
파생변수 생성 또는 변수 변환

3 EDA
시각화 및 데이터 정보 획득

4 Anomaly Detection
이상치 탐지 및 이상치 처리

5 Feature Selection
중요 변수 선택 및 생략, train / validation 분리 (8:2)

6 Modeling
머신러닝 및 딥러닝 학습 후 예측

7 Prediction
학습된 모델로 validation set 예측, 성능 평가

Preprocessing

일자	요일	본사정원수	본사휴가자수	본사출장자수	본사시간외근무명령서승인건수	현본사소속재택근무자수	조식메뉴	중식메뉴	석식메뉴	중식계	석식계
2016-02-01	월	2601	50	150	238	0	모닝롤/찐빵	쌀밥/잡곡	쌀밥/잡곡	1039	331
2016-02-02	화	2601	50	173	319	0	모닝롤/단호	쌀밥/잡곡	콩나물밥	867	560
2016-02-03	수	2601	56	180	111	0	모닝롤/베이	카레덮밥	쌀밥/잡곡	1017	573
2016-02-04	목	2601	104	220	355	0	모닝롤/토마	쌀밥/잡곡	미니김밥	978	525
2016-02-05	금	2601	278	181	34	0	모닝롤/와플	쌀밥/잡곡	쌀밥/잡곡	925	330
2016-02-11	목	2601	383	143	417	0	팬케익/찐빵	쌀밥/잡곡	찰현미밥	1045	550

Data : column

일자
요일
본사정원수
본사휴가자수
본사출장자수
본사시간외근무명령서승인건수
현본사소속재택근무자수
조식메뉴
중식메뉴
석식메뉴
중식계
석식계

일자 관련 컬럼 추가

1 년, 월, 일, 주, 요일 컬럼 추가

```
train_x['년'] = train_x['일자'].dt.year
train_x['월'] = train_x['일자'].dt.month
train_x['일'] = train_x['일자'].dt.day
train_x['주'] = train_x['일자'].dt.isocalendar().week
train_x['요일'] = train_x['일자'].dt.weekday
```

2 계절 컬럼 추가

```
season = []
for i in train_x['월']:
  if i in [3,4,5]:
    season.append(0)
  elif i in [6,7,8]:
    season.append(1)
  elif i in [9,10,11]:
    season.append(2)
  else:
    season.append(3)
train_x['계절'] = season
```

3 2020년 전후 여부 컬럼 추가 (코로나19)

```
train_x["before_2020"] = [1 if i < 2020 else 0
for i in train_x['년']]
```

Preprocessing

	A	B	C	D	E	F	G	H	I	J	K	L
1	일자	요일	본사정원수	본사휴가자수	본사출장자수	본사시간외근무명령서승인건수	현본사소속재택근무자수	조식메뉴	중식메뉴	석식메뉴	중식계	석식계
2	2016-02-01	월	2601	50	150	238	0	모닝롤/찐빵	쌀밥/잡곡(	쌀밥/잡곡(	1039	331
3	2016-02-02	화	2601	50	173	319	0	모닝롤/단호	쌀밥/잡곡(	콩나물밥	867	560
4	2016-02-03	수	2601	56	180	111	0	모닝롤/베이	카레덮밥 (	쌀밥/잡곡(	1017	573
5	2016-02-04	목	2601	104	220	355	0	모닝롤/토마	쌀밥/잡곡(	미니김밥	978	525
6	2016-02-05	금	2601	278	181	34	0	모닝롤/와플	쌀밥/잡곡(	쌀밥/잡곡(	925	330
7	2016-02-11	목	2601	383	143	417	0	팬케익/찐빵	쌀밥/잡곡(	참치회덮밥	1045	550

Data : column

일자
요일
본사정원수
본사휴가자수
본사출장자수
본사시간외근무명령서승인건수
현본사소속재택근무자수
조식메뉴
중식메뉴
석식메뉴
중식계
석식계

일자 관련 컬럼 추가

4 공휴일 전일 여부 컬럼 추가

```
a = []
for n in range(len(train_x)):
  if n == 0:
    if (train_x.iloc[n, 1]) == (train_x.iloc[n + 1, 1] - 1):
      a.append(0)
    else:
      a.append(1)
        ⋮
train_x["공휴일여부"] = a
```

5 sin, cos 주기성 변환 파생변수 컬럼 추가

일자 컬럼의 주기성 데이터를 추출 하기 위해
연중 시간에 Sin, Cos 연산을 적용.

▶시간을 명확한 "하루 중 시간" 및 "연중 시간" 신호로 변환

산식 : (np.sin(timestamp_s * (2 * np.pi / day)))

Preprocessing

	A	B	C	D	E	F	G	H	I	J	K	L
1	일자	요일	본사정원수	본사휴가자수	본사출장자수	본사시간외근무명령서승인건수	현본사소속재택근무자수	조식메뉴	중식메뉴	석식메뉴	중식계	석식계
2	2016-02-01	월	2601	50	150	236	0	모닝롤/찐빵	쌀밥/잡곡밥/짬뽕/잡곡		1039	331
3	2016-02-02	화	2601	50	173	319	0	모닝롤/단호박	쌀밥/잡곡밥/콩나물밥/		867	560
4	2016-02-03	수	2601	56	180	111	0	모닝롤/베이직	쌀밥/잡곡밥/짬뽕/잡곡		1017	573
5	2016-02-04	목	2601	104	220	355	0	모닝롤/토마	쌀밥/잡곡밥/까나리감자		978	525
6	2016-02-05	금	2601	278	181	34	0	모닝롤/마늘	쌀밥/잡곡밥/짬뽕/잡곡		925	330
7	2016-02-11	목	2601	383	143	417	0	팬케익/찐빵	쌀밥/잡곡밥/청직희덮밥		1045	550

Data : column

- 일자
- 요일
- 본사정원수
- 본사휴가자수
- 본사출장자수
- 본사시간외근무명령서승인건수
- 현본사소속재택근무자수
- 조식메뉴
- 중식메뉴
- 석식메뉴
- 중식계
- 석식계

- 년, 월, 일, 주, 요일
- 계절
- 2020년 전후 여부
- 공휴일 전후 여부
- sin, cos 주기성 변환 파생 변수

인원수 관련 컬럼 추가

1 식사 가용 인원 컬럼 추가

```
train_x['식사가용인원'] =
train_x['본사정원수']-(train_x['본사휴가자수
']+train_x['본사출장자수']+train_x['현본사소속재택근
무자수']),astype(int)
```

2 휴가 비율 컬럼 추가

```
train_x['휴가비율'] = round(train_x['본사휴가자수'] /
train_x['본사정원수'],3),astype(float)
```

3 출장 비율 컬럼 추가

```
train_x['출장비율'] = round(train_x['본사출장자수'] /
train_x['본사정원수'],3),astype(float)
```

4 재택 비율 컬럼 추가

```
train_x['재택비율'] = round(train_x['현본사소속재택근
무자수'] / train_x['본사정원수'],3),astype(float)
```

Preprocessing

	A	B	C	D	E	F	G	H	I	J	K	L
1	일자	요일	본사정원수	본사휴가자수	본사출장자수	본사시간외근무명령서승인건수	현본사소속재택근무자수	조식메뉴	중식메뉴	석식메뉴	중식계	석식계
2	2016-02-01	월	2601	50	150	236	0	모닝롤/찐빵	찰잡/칠리찰잡/칠리		1039	331
3	2016-02-02	화	2601	50	171	319	0	모닝롤/단호박찰잡/칠리콘나물밥*			867	560
4	2016-02-03	수	2601	56	160	111	0	모닝롤/베이직카레밥업	찰잡/칠리		1017	573
5	2016-02-04	목	2601	104	220	355	0	모닝롤/토마토찰잡/칠리까나리김밥*			978	525
6	2016-02-05	금	2601	278	181	34	0	모닝롤/마늘	찰잡/칠리찰잡/칠리		925	330
7	2016-02-11	목	2601	383	143	417	0	팬케익/찐빵	찰잡/칠리정치회덮밥		1045	550

Data : column

일자	년, 월, 일, 주, 요일
요일	계절
본사정원수	2020년 전후 여부
본사휴가자수	공휴일 전후 여부
본사출장자수	sin, cos 주기성 변환 파생 변수
본사시간외근무명령서승인건수	식사가용인원
현본사소속재택근무자수	휴가비율, 출장비율, 재택비율
조식메뉴	
중식메뉴	
석식메뉴	
중식계	
석식계	

메뉴 관련 컬럼 추가

1 메뉴 컬럼을 공백 기준으로 분리

```
lunch_train = [ ]
for day in range(len(train_x)):
  tmp = train_x.loc[day, "중식메뉴"].split(' ') # 공백으로 문자열 구분
  tmp = ' '.join(tmp).split() # 빈 원소 삭제
```

2 분리한 메뉴 컬럼 생성

```
menu_len_list = []
bob = []; gook = []; jm = []; side1 = []; side2 = []; kimchi = []; dessert = [];
for i, day_menu in enumerate(lunch_train):
  bob_tmp = day_menu[0]; bob.append(bob_tmp)
  gook_tmp = day_menu[1];
gook.append(gook_tmp)
  jm_tmp = day_menu[2]; jm.append(jm_tmp)
  side1_tmp = day_menu[3];
side1.append(side1_tmp)
  side2_tmp = day_menu[4];
side2.append(side2_tmp)
```

Preprocessing

	A	B	C	D	E	F	G	H	I	J	K	L
1	일자	요일	본사정원수	본사휴가자수	본사출장자수	본사시간외근무명령서승인건수	현본사소속재택근무자수	조식메뉴	중식메뉴	석식메뉴	중식계	석식계
2	2016-02-01	월	2601	50	150	238	0	모닝롤/콘푸	쌀밥/잡곡밥	쌀밥/	1039	331
3	2016-02-02	화	2601	50	171	319	0	모닝롤/단호	쌀밥/잡곡밥	콩나물밥/	867	560
4	2016-02-03	수	2601	56	180	111	0	모닝롤/베이	카레덮밥	쌀밥/	1017	573
5	2016-02-04	목	2601	104	220	355	0	모닝롤/토마	쌀밥/잡곡밥	미니김밥/	978	525
6	2016-02-05	금	2601	278	181	34	0	모닝롤/마늘	쌀밥/잡곡밥	쌀밥/	925	330
7	2016-02-11	목	2601	383	143	417	0	팬케익/콘	쌀밥/잡곡밥	참치회덮	1045	550

Data : column

일자
요일
본사정원수
본사휴가자수
본사출장자수
본사시간외근무명령서승인건수
현본사소속재택근무자수
조식메뉴
중식메뉴
석식메뉴
중식계
석식계

년, 월, 일, 주, 요일
계절
2020년 전후 여부
공휴일 전후 여부
sin, cos 주기성 변환 파생 변수
식사가용인원
휴가비율, 출장비율, 재택비율

메뉴 관련 컬럼 추가

중식메뉴
쌀밥/잡곡밥 (쌀,현미흑미:국내산) 오징어찌개 쇠불고기 (쇠고기:호주산) 계란찜 청포묵무침 요구르트 포기김치 (배추,고추가루:국내산)
쌀밥/잡곡밥 (쌀,현미흑미:국내산) 김치찌개 가자미튀김 모둠소세지구이 마늘쫑무침 요구르트 배추겉절이 (배추,고추가루:국내산)
카레덮밥 (쌀,현미흑미:국내산) 팽이장국 치킨핑거 (닭고기:국내산) 쫄면야채무침 견과류조림 요구르트 포기김치 (배추,고추가루:국내산)

밥	국	메인메뉴	반찬1	반찬2	김치	사이드
쌀밥/잡곡밥	오징어찌개	쇠불고기	계란찜	청포묵무침	포기김치	요구르트
쌀밥/잡곡밥	김치찌개	가자미튀김	모듬소시지구이	마늘쫑무침	배추겉절이	요구르트
카레덮밥	팽이장국	치킨핑거	쫄면야채무침	견과류조림	포기김치	요구르트

Preprocessing

일자	요일	본사정원수	본사휴가자수	본사출장자수	본사시간외근무명령서승인건수	현본사소속재택근무자수	조식메뉴	중식메뉴	석식메뉴	중식계	석식계
2016-02-01	월	2601	50	150	238	0	모닝롤/찐빵	쌀밥/잡곡	쌀밥/잡곡	1039	331
2016-02-02	화	2601	50	173	319	0	모닝롤/단호	쌀밥/잡곡	콩나물밥	867	560
2016-02-03	수	2601	56	180	111	0	모닝롤/베이	카레덮밥 (	쌀밥/잡곡	1017	573
2016-02-04	목	2601	104	220	355	0	모닝롤/토마	쌀밥/잡곡	미니김밥	978	525
2016-02-05	금	2601	278	181	34	0	모닝롤/와플	쌀밥/잡곡	쌀밥/잡곡	925	330
2016-02-11	목	2601	383	143	417	0	팬케익/찐빵	쌀밥/잡곡	참치회덮밥	1045	550

Data : column

일자
요일
본사정원수
본사휴가자수
본사출장자수
본사시간외근무명령서승인건수
현본사소속재택근무자수
조식메뉴
중식메뉴
석식메뉴
중식계
석식계

년, 월, 일, 주, 요일
계절
2020년 전후 여부
공휴일 전후 여부
sin, cos 주기성 변환 파생 변수
식사가용인원
휴가비율, 출장비율, 재택비율
메뉴, 특식 여부, 신메뉴 여부

메뉴 관련 컬럼 추가

3 특식 컬럼 생성

```
mt = []; # 특식 컬럼 변수명 지정

for i in train_x['밥']:
  if '쌀밥' not in i:
    jmt.append(1) # '쌀밥' 이라는 단어가 '밥' 컬럼에 있는 경우 1을 반환
  else:
    jmt.append(0) # 있는 경우 0을 반환
```

4 신메뉴 컬럼 생성

```
new = [];

for i in range(len(train_x)):
    if 'New' in train_x.loc[i, '중식메뉴']:
# 중식메뉴 컬럼에 'New' 라는 문자열이 있으면
      new.append(1) # 1을 반환
  else:
      new.append(0) # 아닌 경우 0을 반환
```

5 영향력 없는 밥, 김치 컬럼 분석대상에서 제외

```
train_x.drop(["중식메뉴", "밥", "김치"], axis=1,
inplace=True)
```

Preprocessing

	A	B	C
1	일시	기온	강수량
13	2016-01-01 11:00	3.2	
14	2016-01-01 12:00	4.8	
15	2016-01-01 13:00	6.4	
16	2016-01-01 14:00	7.8	

	A	B	C
1	일시	기온	강수량
3133	2016-05-10 11:00	15.2	5.6
3134	2016-05-10 12:00	15.3	1.7
3135	2016-05-10 13:00	15.3	4.7
3136	2016-05-10 14:00	15.7	1.4

Data : column

일시
기온
강수량

Data : source

기상자료개방포털 - 진주시 종관기상관측(ASOS) 자료
https://data.kma.go.kr/cmmn/main.do

날씨 관련 컬럼 추가

1 기온/강수량 컬럼 특정 시간대(11-13시) 지정

```
tmp_list = [ ]
for i in forecast["일시"]:
  if ("11:00" in i) or ("12:00" in i) or ("13:00" in i):
    tmp_list.append(True)
  else:
    tmp_list.append(False)
```

2 기온/강수량 결측치 처리 (전날 값으로 대체)

```
train_x["기온"][findIdx(train_x["기온"].isna(),
[True])[0]] = train_x["기온"][findIdx(train_x["기온
"].isna(), [True])[0]-1]
train_x["강수량"][findIdx(train_x["강수량"].isna(),
[True])[0]] = 0
train_x.isna().sum().sum()
```

3 기존 데이터에 기온/강수량 컬럼 병합

```
train_x["강수여부"] = [1 if i>0 else 0 for i in
train_x["강수량"]]
                    ⋮
train_x = pd.merge(train_x, forecast_new,
how="left", on="일자")
```

Data : column

일자	년, 월, 일, 주, 요일
요일	계절
본사정원수	2020년 전후 여부
본사휴가자수	공휴일 전후 여부
본사출장자수	sin, cos 주기성 변환 파생 변수
본사시간외근무명령서승인건수	식사가용인원
현본사소속재택근무자수	휴가비율, 출장비율, 재택비율
조식메뉴	메뉴, 특식 여부, 신메뉴 여부
중식메뉴	기온
석식메뉴	강수량
중식계	
석식계	

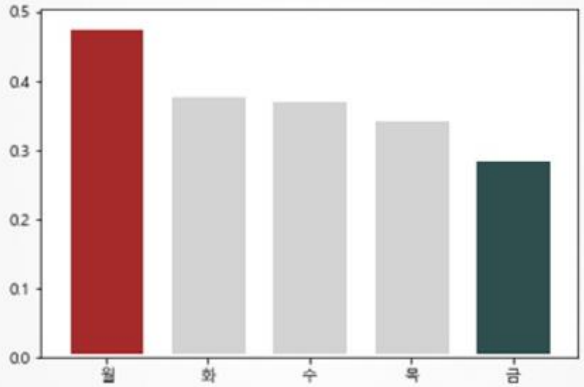
EDA

1. 시간 관련 중식계 시각화
   요일별 중식계, 월별 중식계
2. 인원 관련 중식계 시각화
   식사가용인원별 중식계
3. 메뉴 관련 중식계 시각화
   특식 여부별 중식계, 신메뉴 여부별 중식계
4. 날씨 관련 중식계 시각화
   기온별 중식계, 강수량별 중식계

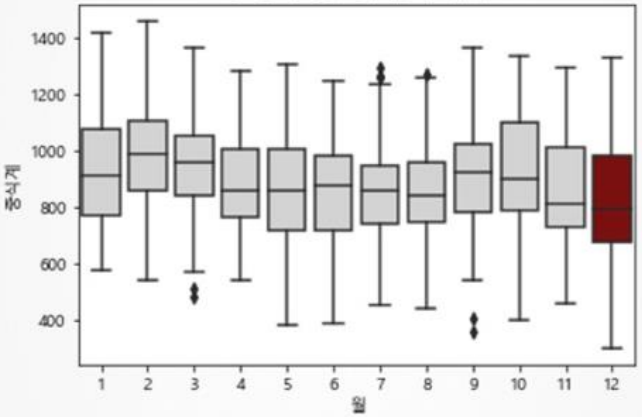
1 시간 관련 중식계 시각화

요일별 중식계, 월별 중식계

각 요일별 직원 중식이용비율

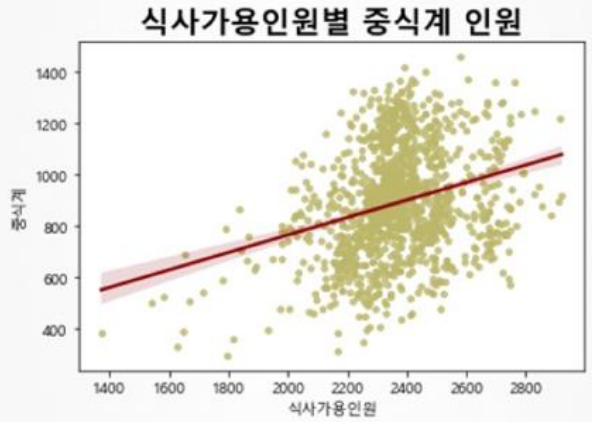


월별 중식계 인원



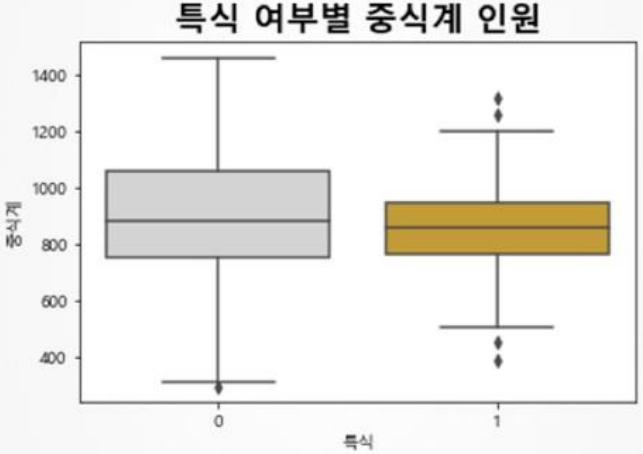
2 인원 관련 중식계 시각화

식사가용인원별 중식계

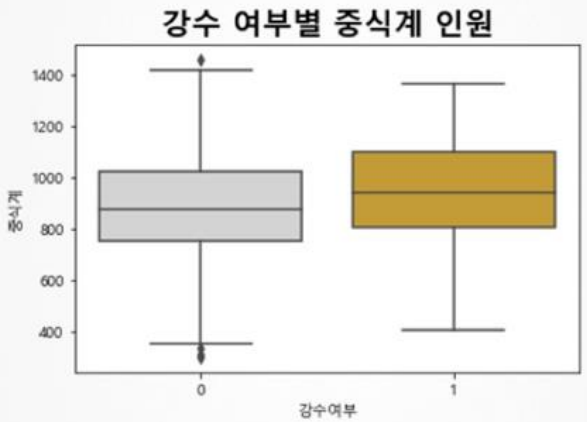


3 메뉴 관련 중식계 시각화

특식 여부별 중식계, 신메뉴 여부별 중식계



날씨 관련 중식계 시각화
강수 여부별 중식계



시각화 요약.

- 시간 관련 중식계
1. 직원들이 월요일에 중식 이용률이 높고, 금요일에는 중식 이용률이 낮음
2. 12월(연말) 중식계 인원이 다른 월에 비해 낮음
(직원들의 남은 휴가 사용 및 종무식 등...의 영향일 것으로 판단)

- 인원 관련 중식계
1. 가용인원이 많을 수록 중식계도 늘어나는 선형관계 확인. (당연할 수 있는 결과이지만 시각화를 통해 확실하게 확인하는 절차를 진행)

- 메뉴 관련 중식계
1. 특식의 경우 중식계에 큰 영향이 없음을 확인
2. 신메뉴가 나오는 날은 중식계 인원이 더 많고 바이어스는 더 적음

- 날씨 데이터
1. 기온은 중식계에 큰 영향을 주지 않았음
2. 강수가 있는 날 중식계가 늘어난다는 선형관계를 파악함.

Modeling

Tool

RANDOM FOREST

dmlc XGBoost

LightGBM

CatBoost

Panel

김영준

김남이

이예주

이지예

Modeling
RANDOM FOREST

1 Fixed parameter

```
"learning_rate": 0.01
"n_estimators": 500
```

2 Parameter Grid

```
"num_leaves": [pow(2, i) - 1 for i in [2, 4, 6, 8]],
"subsample": [0.4, 0.6, 0.8],
"colsample_bytree": [0.6, 0.8, 1],
"reg_lambda": list(np.linspace(0.1, 10, 10).round(3))
```

3 Best Parameter

```
best_iteration: 79
'colsample_bytree':0.6
'num_leaves': 63
'reg_lambda': 0.1
'subsample': 0.8
```

4 Performance

```
RMSE: 95.69
R2: 0.768
```

Modeling
XGBoost

1 Fixed parameter

```
ntrees = 5000 , booster="gbtree",
n_estimators=int(ntrees*0.3),
objective="reg:squarederror"
```

2 Parameter Grid

```
xgb_param_grid = {
  'learning_rate': [0.01,0.05],
  'max_depth': [2,4,6],
  'reg_lambda' : [0.5, 1, 5, 10],
  'subsample' : [0.5, 0.6, 0.8]}
```

3 Best Parameter

```
n_estimators=5000, learning_rate=0.01,
max_depth=4, reg_lambda= 5, subsample=0.5,
objective="reg:squarederror",colsample_bytree=0.
8,seed=343 , #best iteration : 80.79243
```

4 Performance

```
Mean squared error: 80.1128598973427
R2 score: 0.8394807725314319
```

```
# 1. 일자 관련 컬럼
# 년월일주, 요일, 계절 추가
train_x['년'] = train_x['일자'].dt.year
train_x['월'] = train_x['일자'].dt.month
train_x['일'] = train_x['일자'].dt.day
train_x['주'] = train_x['일자'].dt.isocalendar().week
train_x['요일'] = train_x['일자'].dt.weekday
# 계절
season = []
for i in train_x['월']:
    if i in [3,4,5]:
        season.append(0)
    elif i in [6,7,8]:
        season.append(1)
    elif i in [9,10,11]:
        season.append(2)
    else:
        season.append(3)
train_x['계절'] = season

weekofmonth = []
for i in train_x["주"]:
    weekofmonth.append((i-1) % 4)
train_x['주'] = weekofmonth
```

<- 먼가 다 보여주긴해야 할 거 같긴 한데.. 쫌 더 있어보이는 계절만 ex) 로 해서 일부만 보여주듯이 들어가면 좋을듯해서?

일자, 요일, 주, 계절성 연관이 있는지 알아보기 위해 컬럼 추가

봄(3,4,5월) = 0
여름(6,7,8월) = 1
가을(9,10,11월) = 2
겨울(12,1,2월) =3 으로 값 설정함.

2. 2020년 전후 여부 컬럼추가 - 코로나 관련

```
train_x["before_2020"] = [1 if i < 2020 else 0 for i in train_x['년']]
```

코로나 후로 재택인원수가 많이 증가했음.
그래서 코로나 전후 식수 영향을 확인하기 위해 해당 컬럼 추가
(2020년 전 = 1, 2020년 이후 = 0)

```
# 3. 공휴일전후여부 컬럼추가 (공휴일전후:1 / 공휴일전후아님:0)
a = []
for n in range(len(train_x)):
    if n == 0:  # 첫행은 앞의 행이 없으므로, '뒷행-1'값이랑만 일치하면 평일
        if (train_x.iloc[n, 1]) == (train_x.iloc[n + 1, 1] - 1):
            a.append(0)
        else:
            a.append(1)     <- 여기까지만 대충 보여주고 뒤에 .....(후략) 이라고 쓰는건 어때?

    elif n == len(train_x) - 1:  # 마지막행은 뒷 행 없음, '앞행+1' 값만 일치하면 평일
        if (train_x.iloc[n, 1] == (train_x.iloc[n - 1, 1] + 1)):
            a.append(0)
        else:
            a.append(1)

    elif train_x.iloc[n, 1] == 0:  # 월요일(0)의 경우, '뒷행-1'값은 0, '앞행+1'값은 5여야 평일
        if (train_x.iloc[n + 1, 1] - 1 == 0) and (train_x.iloc[n - 1, 1] + 1 == 5):
            a.append(0)
        else:
            a.append(1)

    elif train_x.iloc[n, 1] == 4:  # 금요일(4)의 경우, '앞행+1'값은 4, '뒷행-1' 값은 -1이어야 평일
        if (train_x.iloc[n + 1, 1] - 1 == -1) and (train_x.iloc[n - 1, 1] + 1 == 4):
            a.append(0)
        else:
            a.append(1)

    elif ((train_x.iloc[n, 1] == (train_x.iloc[n - 1, 1] + 1)) and (train_x.iloc[n, 1]) == (
            train_x.iloc[n + 1, 1] - 1)):  # 월,금 아니면(1,2,3) 앞행+1값과 뒷행-1값은 모두 일치해야 평일.
        a.append(0)
    else:
        a.append(1)

train_x["공휴일여부"] = a
```

공휴일 전후 일자가 식수계에 영향을 주는지를 확인하기 위해 별도의 컬럼 추가

공휴일 전후일 = 1
공휴일 전후일 아님 = 0

```
# sin, cos 주기성 변환 파생변수 컬럼추가
time_zero = datetime(1970, 1, 1, 0, 0, 0)
day_to_sec = 24*60*60
year_to_sec = (365.2425)*day_to_sec
frequency_sin_year = []
frequency_cos_year = []
for i in train_x["일자"]:
    time_to_sec = i.to_pydatetime()
    time_interval = (time_to_sec -
time_zero).total_seconds()

frequency_sin_year.append(np.sin((time_inte
rval / year_to_sec) * 2 * np.pi))

frequency_cos_year.append(np.cos((time_int
erval / year_to_sec) * 2 * np.pi))

train_x["frequency_sin_year"] =
frequency_sin_year
train_x["frequency_cos_year"] =
frequency_cos_year
# train_x["frequency_sin_year"].plot()
# train_x["frequency_cos_year"].plot()
```

여긴 <- 여기 나와있는 코드 말고 아랫 문장부터만 언급해도 될듯.

sin, cos 주기성 변환 파생변수 컬럼추가

일자 컬럼의 주기성 데이터를 추출 하기 위해 연중 시간에 Sin, Cos 연산을 적용.
= 시간을 명확한 "하루 중 시간" 및 "연중 시간" 신호로 변환

산식 : (np.sin(timestamp_s * (2 * np.pi / day)))

2. 인원수 관련 컬럼 - 파생변수 추가시킴 (식사가용인원, 휴가비율, 출장비율, 재택비율)

```
train_x['식사가용인원'] = train_x['본사정원수']-(train_x['본사휴가자수']+train_x['본사출장자수']+train_x['현본사소속재택근무자수']).astype(int)
train_x['휴가비율'] = round(train_x['본사휴가자수'] / train_x['본사정원수'],3).astype(float)
train_x['출장비율'] = round(train_x['본사출장자수'] / train_x['본사정원수'],3).astype(float)
train_x['재택비율'] = round(train_x['현본사소속재택근무자수'] / train_x['본사정원수'],3).astype(float)
```

```
# 3. 메뉴 관련 컬럼
# 메뉴를 밥, 국, 메인메뉴, 반찬1, 반찬2, 김치, 사이드 변수로 분리
# 일별 점심메뉴를 작은 리스트로 갖고 있는 2중 리스트 (lunch_train) 만들기
lunch_train = []
for day in range(len(train_x)):
    tmp = train_x.loc[day, "중식메뉴"].split(' ')  # 공백으로 문자열 구분
    tmp = ' '.join(tmp).split()  # 빈 원소 삭제

    search = '(' # 원산지 정보는 삭제
    # (원산지는 항상 맨 뒤 space로 구분 되므로 괄호하나만 해도 처리 가능)
    for menu in tmp:
        if search in menu:
            tmp.remove(menu)
    lunch_train.append(tmp)

# lunch_train

# lunch train data에 메뉴명별 칼럼 만들기 (밥, 국, 반찬1-3)
menu_len_list = []
bob = []; gook = []; jm = []; side1 = []; side2 = []; kimchi = []; dessert = [];
for i, day_menu in enumerate(lunch_train):
    bob_tmp = day_menu[0]; bob.append(bob_tmp)
    gook_tmp = day_menu[1]; gook.append(gook_tmp)
    jm_tmp = day_menu[2]; jm.append(jm_tmp)
    side1_tmp = day_menu[3]; side1.append(side1_tmp)
    side2_tmp = day_menu[4]; side2.append(side2_tmp)

    if i < 1067:
        kimchi_tmp = day_menu[-1]; kimchi.append(kimchi_tmp)
        dessert_tmp = day_menu[-2]; dessert.append(dessert_tmp)
    else:
```

=> 메뉴를 밥, 국, 메인메뉴, 반찬1, 반찬2, 김치, 사이드 변수로 분리

```
# 특식 컬럼 추가 (쌀밥이 아닌 비빔밥 등의 특식 메뉴 제공)
jmt = [];  # 특식 컬럼 변수명 지정

for i in train_x['밥']:
    if '쌀밥' not in i:
        jmt.append(1)  # '쌀밥'이라는 단어가 '밥' 컬럼에 있는 경우 1을 반환
    else:
        jmt.append(0)  # 있는 경우 0을 반환

# print(jmt)
train_x['특식'] = jmt
# 따라서 특식에 쌀밥이 아닌 다른 밥메뉴들이 들어갈 경우 1을 반환

# 신메뉴 컬럼 추가 (신메뉴가 수요에 미치는 영향을 파악) (중식)
new = [];

for i in range(len(train_x)):  # 행 길이 만큼 반복
    if 'New' in train_x.loc[i, '중식메뉴']:  # 중식메뉴 컬럼에 'New'라는 문자열이 있으면
        new.append(1)  # 1을 반환
    else:
        new.append(0)  # 아님 0을 반환

train_x['신메뉴'] = new

train_x.drop(["중식메뉴", "밥", "김치"], axis=1, inplace=True)
```

특식과 신메뉴 컬럼 추가,

이후

영향력이 없다고 판단한
밥과 김치는 분석 대상에서 제외시킴

메뉴 분리전 분리후

중식메뉴
쌀밥/잡곡밥 (쌀,현미흑미:국내산) 오징어찌개 쇠불고기 (쇠고기:호주산) 계란찜 ...
쌀밥/잡곡밥 (쌀,현미흑미:국내산) 김치찌개 가자미튀김 모둠소세지구이 마늘쫑무...
카레덮밥 (쌀,현미흑미:국내산) 팽이장국 치킨핑거 (닭고기:국내산) 쫄면야채무침 ...
쌀밥/잡곡밥 (쌀,현미흑미:국내산) 쇠고기무국 주꾸미볶음 부추전 시금치나물 ...
쌀밥/잡곡밥 (쌀,현미흑미:국내산) 떡국 돈육씨앗강정 (돼지고기:국내산) 우엉잡채...

---------->

밥	국	메인메뉴	반찬1	반찬2	김치	사이드
쌀밥/잡곡밥	오징어찌개	쇠불고기	계란찜	청포묵무침	포기김치	요구르트
쌀밥/잡곡밥	김치찌개	가자미튀김	모둠소세지구이	마늘쫑무침	배추겉절이	요구르트
카레덮밥	팽이장국	치킨핑거	쫄면야채무침	견과류조림	포기김치	요구르트
쌀밥/잡곡밥	쇠고기무국	주꾸미볶음	부추전	시금치나물	포기김치	요구르트
쌀밥/잡곡밥	떡국	돈육씨앗강정	우엉잡채	청경채무침	포기김치	요구르트

```
# Label Encoding 수행
label_encoder = MyLabelEncoder()
train_x = label_encoder.fit_transform(train_x, col_names=["국", "메인메뉴", "반찬1", "반찬2", "사이
드"])

# 메뉴관련컬럼(5개)을 tensorflow embedding layer를 통해 dense vector로 변환하기
# OneHotEncoding 시 몇 백개 이상의 컬럼이 생성됨
rnd.random(33)
tf.random.set_seed(56)
menu_vec = ["국", "메인메뉴", "반찬1", "반찬2", "사이드"]
# 레이어 만들기
def createEmbeddingModel(nCols):
    # ----------- Embedding layers ----------------------
    B0_input = layers.Input(shape=(nCols), name="B0_input")
    B0_embedding = layers.Embedding(input_dim=512,
                                    output_dim=64,
                                    name="B0_embedding")(B0_input)
    # ----------- Convolution layers ----------------------
    B1_conv1d = layers.Conv1D(4, 1, activation='relu', name="B1_conv1d")(B0_embedding)
    B1_flatten = layers.Flatten(name="extract")(B1_conv1d)
    B1_dropout = layers.Dropout(0.25, name="B1_dropout")(B1_flatten)
    # ----------- Regressor ----------------------
    last_regressor = layers.Dense(1)(B1_dropout)
    model_mlp = Model(B0_input, last_regressor)
    model_mlp.compile(loss="mse", optimizer=optimizers.Adam(3e-3),
                      metrics=tf_metrics.RootMeanSquaredError(name="rmse"))
    return model_mlp
```

- (마지막)기상청 외부데이터 결측치 처리 언급

	A	B	C
	일시	기온	강수량
	2016-01-01 0:00	-3.5	
	2016-01-01 1:00	-4	

 # 기상청 외부데이터 추가 (날씨)

**** 이거 바로 뒷페이지에**
필요한 코드랑 내용 정리해놨어!!!!!!!!

```
# 기상청 외부데이터 추가 (날씨)
# 파일 경로 유의
# 점심 11,12,13 저녁 17,18,19
forecast = read_csv("./2차프로젝트/2016_2021_진주_기온강수
.csv", encoding="euc-kr")
forecast["강수량"].fillna(0, inplace=True)
forecast.isna().sum()

findIdx(forecast["기온"].isna(), [True])
forecast["기온"][40765:40768]
forecast["기온"][40766] = forecast["기온"][40765:40768].mean()
forecast["기온"][40768:40771]
forecast["기온"][40769] = forecast["기온"][40768:40771].mean()
forecast.isna().sum().sum()
train_x.isna().sum().sum()

# forecast
tmp_list = []
for i in forecast["일시"]:
    if ("11:00" in i) or ("12:00" in i) or ("13:00" in i):
        tmp_list.append(True)
    else:
        tmp_list.append(False)

forecast = forecast[tmp_list]
```

기상청 외부데이터 추가한부분

 # 기상청 외부데이터 추가 (날씨)

11시, 12시, 13시 기준 강수량이 증식수계에 영향을 주었을까?를 확인하기 위해.

-> 일일히 결측치 확인 등등 모든걸 설명하기엔 오래 걸리니까
바로 옆에 볼드처리 해둔 코드만 보여주면 될 거 같아.
1) 11시 , 12시, 13시 의 강수량 데이터만 뽑아낸부분,
2) 결측치 처리한 부분,

```
tmp_list = []
for i in forecast["일시"]:
    if ("11:00" in i) or ("12:00" in i) or ("13:00" in i):
        tmp_list.append(True)
    else:
        tmp_list.append(False)
```

```
# 기온 및 강수량의 11~13 시 결측치는 전날 기온 및 강수량으로 대체하였다
train_x["기온"][findIdx(train_x["기온"].isna(), [True])[0]] = train_x["기온"][findIdx(train_x["기온"].isna(), [True])[0]-1]
train_x["강수량"][findIdx(train_x["강수량"].isna(), [True])[0]] = 0
train_x.isna().sum().sum()

train_x["강수여부"] = [1 if i>0 else 0 for i in train_x["강수량"]]
# train_x.drop(["일자"], axis=1, inplace=True)
```

프로젝트 수행과정

시각화

1. 시간 관련 중식계 시각화
2. 인원 관련 중식계 시각화
3. 메뉴 관련 중식계 시각화
4. 날씨 관련 중식계 시각화

프로젝트 수행과정

머신러닝

각자 한개의 모델링을 맡았다고하고
다음페이지부터 4개 모델링 한개씩 세부
설명

김영준 RandomForest
--- Fixed parameter ---
“learning_rate”: 0.01
“n_estimators”: 500

--- Parameter Grid ---
"num_leaves": [pow(2, i) - 1 for i in [2, 4, 6, 8]],
"subsample": [0.4, 0.6, 0.8],
"colsample_bytree": [0.6, 0.8, 1],
"reg_lambda": list(np.linspace(0.1, 10, 10).round(3))

--- best parameter ---
best_iteration: 79
'colsample_bytree':0.6
'num_leaves': 63
'reg_lambda': 0.1
'subsample': 0.8

categorical_feature 옵션을 활용 (OneHot Encoding 이 아닌 Label Encoding 된 데이터 사용)
Early stopping rounds: n_estimators * 0.2
model_tuner.fit(train_x, train_y, categorical_feature=findIdx(train_x, cat_vars), verbose=False)

--- Performance ---
RMSE: 95.69
R2: 0.768

남이 모델링
쓰세요
XGBoost

#3차GridSearchCV (이제그만….) 최적의 하이퍼 파라미터 찾기

ntrees = 5000

```
model_xgb3 = XGBRegressor(booster="gbtree", n_estimators=int(ntrees*0.3), objective="reg:squarederror", seed=343)

objective="reg:squarederror", seed=343)
xgb_param_grid = {
    'learning_rate': [0.01,0.05],
    'max_depth': [2,4,6],
    'reg_lambda' : [0.5, 1, 5, 10],
    'subsample' : [0.5, 0.6, 0.8]
}

xgb_grid3 = GridTuner(model_xgb3, param_grid=xgb_param_grid,
scoring='neg_root_mean_squared_error',
                cv=10, n_jobs=-1, refit=False, verbose=1)
xgb_grid3.fit(train_x_oh, train_y)
```

Xgb boost 이어서..

#최적 파라미터 검색 후 아래 파라미터 값으로 조정하여 학습

n_estimators=5000, learning_rate=0.01, max_depth=4, reg_lambda= 5, subsample=0.5,
objective="reg:squarederror",colsample_bytree=0.8,seed=343

#test 데이터셋으로 학습

```
xgb4_pred = xgb4.predict(val_x_oh)
xgb4_rmse = mean_squared_error(val_y, xgb4_pred)
xgb4_r2 = r2_score(val_y, xgb4_pred)
print('Mean squared error: ', np.sqrt(xgb4_rmse))
print('R2 score: ', xgb4_r2)
```

#best iteration

[2478] validation_0-rmse:80.79243

#Performance

Mean squared error: 80.1128598973427

R2 score: 0.8394807725314319

예주 모델링 쓰세요
LightGBM

```
#최적 파라미터 찾기(5000)
ntrees = 5000
model_lgb = LGBMRegressor(boosting="goss", n_estimators=int(ntrees*0.2),
objective="regression", seed=525)
lgb_param_grid = {
    'learning_rate': [0.01,0.05,0.1],
    'num_leaves': [3,7,15,31],
    'reg_lambda' : [0.1, 1, 10],
    'subsample' : [0.5,0.6,0.7]
}
lgb_grid = GridSearchCV(model_lgb, param_grid=lgb_param_grid,
                        scoring='neg_root_mean_squared_error',
                    cv=10, n_jobs=-1, refit=False, verbose=1)
lgb_grid.fit(train_x_oh, train_y)

print("최적 하이퍼 파라미터:" , lgb_grid.best_params_)
```

```
Fitting 10 folds for each of 108 candidates, totalling 1080 fits
최적 하이퍼 파라미터: {'learning_rate': 0.05, 'num_leaves': 3, 'reg_lambda': 1, 'subsample':
0.5}
```

```
#학습
start = time.time()
lgb = LGBMRegressor(boosting="goss", n_estimators = ntrees,
                    objective="regression", seed=525,
                    learning_rate = 0.05, num_leaves = 3,
                   reg_lambda= 1,subsample= 0.5)
evals = [(X_test, y_test)]
lgb.fit(X_train, y_train, early_stopping_rounds = 100, eval_metric='rmse', eval_set=evals,
verbose=True)
lgb_pred = lgb.predict(X_test)

from math import sqrt

lgb_rmse = sqrt(mean_squared_error(y_test, lgb_pred))
lgb_r2 = r2_score(y_test, lgb_pred)
print('Mean squared error: ', lgb_rmse)
print('R2 score: ', lgb_r2)
```

```
Early stopping, best iteration is:
[329]           valid_0's rmse: 90.5191         valid_0's l2: 8193.71
RMSE:  90.51911534354515
R2 score:  0.8052377660686656
```

지예 모델링 쓰세요
CatBoost

1. CatBoost 최적 하이퍼 파라미터 찾기

ntrees = 3000

```
cb = cb.CatBoostRegressor(random_state=11, n_estimators=int(ntrees*0.2), loss_function = 'RMSE' )


from sklearn.model_selection import GridSearchCV
param = {
    'learning_rate' : [0.05, 0.06, 0.1],
    'max_depth' : [2,5,8],
    'l2_leaf_reg' : [0,3,5,10]
}
grid_cv = GridSearchCV(cb, param_grid=param,  scoring='neg_root_mean_squared_error', cv=10, verbose=1, n_jobs=-1)
grid_cv.fit(train_x_oh, train_y)
print('최적 하이퍼 파라미터: \n', grid_cv.best_params_)
print('최고 예측 정확도(RMSE의 -값): {0:.4f}'.format(grid_cv.best_score_))
```

##================> [결과값]
최적 하이퍼 파라미터:
{'l2_leaf_reg': 3, 'learning_rate': 0.06, 'max_depth': 5}
최고 예측 정확도(RMSE의 -값): -79.7086

2. 최적 하이퍼파라미터에 적용시키기

“코드 수행 시간이 다른 모델링 기법 보다 오래 소요
되므로

CatBoost는 Tree수를 5000이 아닌 3000으로 설정”

```
ntrees = 3000

cb1 = cb.CatBoostRegressor(l2_leaf_reg = 3,learning_rate = 0.06, n_estimators = ntrees, max_depth=5, boosting_type='Plain', early_stopping_rounds=500, use_best_model=True, loss_function = 'RMSE') # 최적 하이퍼파라미터에 적용한 후 다시 학습시키기
cb1_model = cb1.fit(train_x_oh, train_y, eval_set=[(val_x_oh, val_y)])

# GridSearchCV를 이용해 최적으로 학습된 estimators로 예측 수행
cb1_model_predict = cb1_model.predict(val_x_oh)


start = time.time()

print("Time:%.1f" % (time.time() - start), "seconds") # 코드 실행 시간 계산


print("RMSE: {:.3f}".format(sqrt(mean_squared_error(val_y,cb1_model_predict))))
print("R2: {:.3f}".format(r2_score(val_y,cb1_model_predict)))

##===============> [결과값]
bestTest = 78.74562259
bestIteration = 535

Shrink model to first 536 iterations.
Time:0.0 seconds
RMSE: 78.746
R2: 0.843
```

```
#---------------- Stacked Ensemble----------------
# RandomForest 제외 (성능 미달)

rnd.seed(1234)
stacking_base_models = [
    ("XGBoost", xgb.XGBRegressor(booster="gbtree", objective="reg:squarederror",
                                 n_estimators=2478, max_depth=4,
                                 subsample=0.6, colsample_bytree=0.8,
                                 reg_lambda=5, learning_rate=0.01,
                                 verbosity=0, random_state=777)),
    ("LightGBM", lgb.LGBMRegressor(boosting_type="goss", objective="regression",
                                   n_estimators=912, num_leaves=2**3-1,
                                   subsample=0.5, colsample_bytree=0.8,
                                   reg_lambda=11, learning_rate=0.01,
                                   silent=True, random_state=777)),
    ("CatBoost", cat.CatBoostRegressor(boosting_type='Plain', loss_function='RMSE',
                                       n_estimators=535, max_depth=5,
                                       rsm=0.8, # rsm = colsample_bytree
                                       l2_leaf_reg=3, learning_rate=0.06,
                                       silent=True, random_seed=777))
]
```

Stacking Ensemble

Meta Leaner Definition (2가지 모델 시도)

1. Linear : 각 예측치에 대해 선형결합하여 다른 모델이 비해 직관적인 결과를 산출
Performance
{'RMSE': 76.8885220689633, '**R2': 0.8723713829212315**}

2. ElasticNet : 각 모델의 예측치에 정규화를 수행하여 성능이 높은 모델의 가중치를 더 높게 가져감

> Hyper-Parameter details
ElasticNetCV(l1_ratio=[.1, .3, .5, .6, .7, .75, .8, .85, .9, .95, .99, 1], n_alphas=1000)

Performance
{'RMSE': 77.21209404644407, '**R2': 0.8712949169257178**}

4가지 모델
결과 비교

딥러닝 여기부터~

딥러닝 모델 공통 파라미터 정의

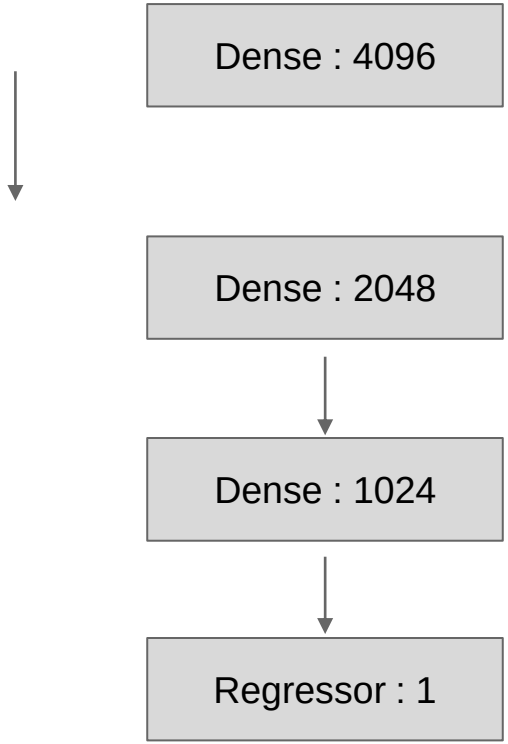
```
tf.random.set_seed(54321)

epochs = 200
batch_size = 8
patientRate = 0.2

cb_earlystopping = tf_callbacks.EarlyStopping(patience=int(np.floor(epochs * patientRate)),
restore_best_weights=True)
cb_reduceLR = tf_callbacks.ReduceLROnPlateau(patience=int(np.floor(epochs * (patientRate ** 2))),
factor=0.8, min_lr=1e-4)
cb_lists = [cb_earlystopping, cb_reduceLR, TqdmCallback(verbose=0)]
```

이지예

```
dropoutRate=0.2
B1 = layers.Dense(2 ** 12, activation="relu")(B0)
B1_dropout = layers.Dropout(rate=dropoutRate,name="B1_dropout")(B1)
B2 = layers.Dense(2 ** 11, activation="relu")(B1_dropout)
B2_dropout = layers.Dropout(rate=dropoutRate,name="B2_dropout")(B2)
B3 = layers.Dense(2 ** 10, activation="relu")(B2_dropout)
```

RMSE: 92.212

R2: 0.785



#남이 딥러닝

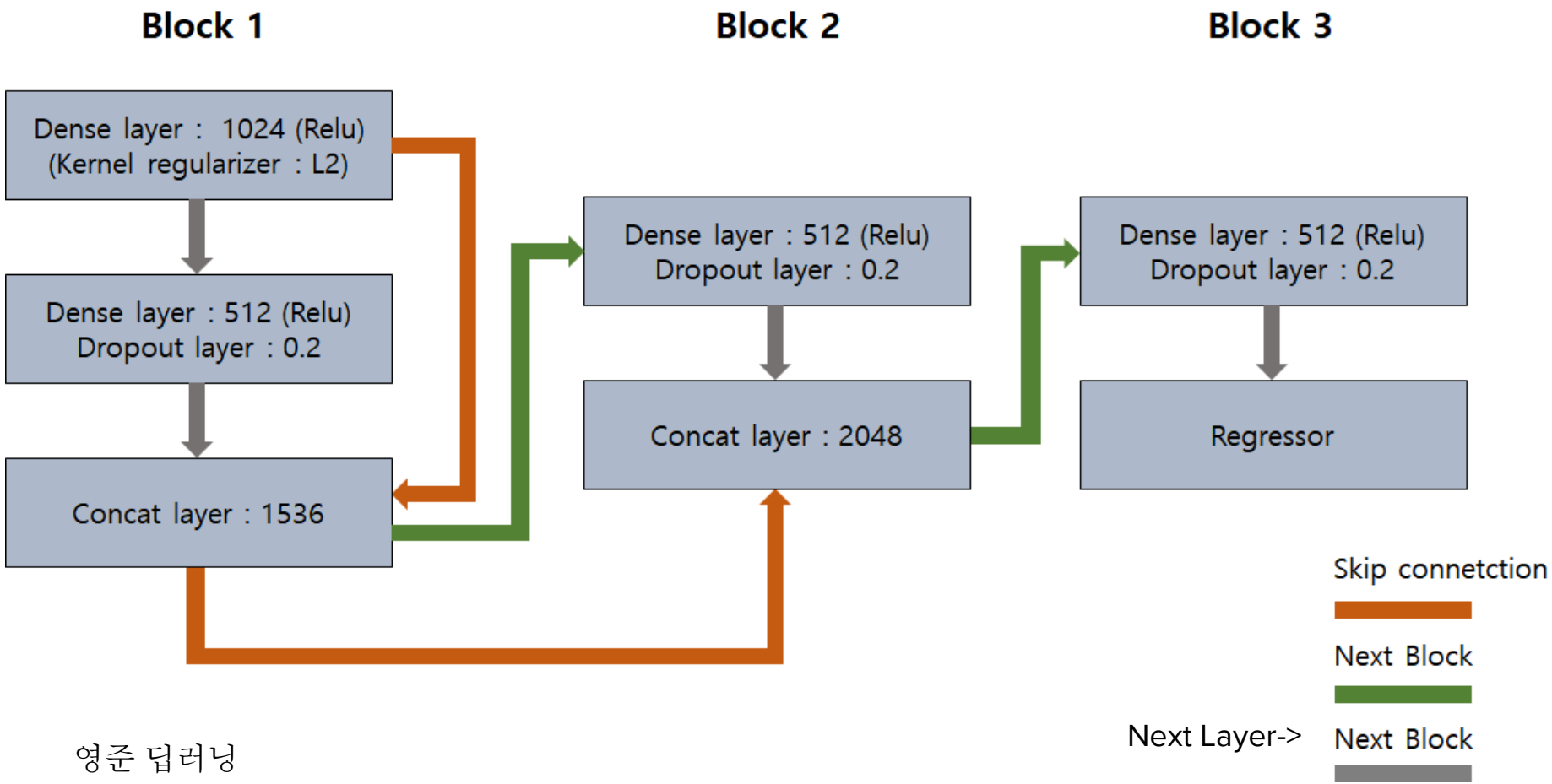
B1 = layers.Dense(512, activation="relu")(B0)

B2 = layers.Dense(512, activation="relu")(B1)

B3 = layers.Dense(256, activation="relu")(B2)

B4 = layers.Dense(128, activation="relu")(B3)

Mean squared error: 8522.406784431943
R2 score: 0.7868502392769979


Block 1
Block 2
Block 3
Dense layer : 1024 (Relu)
(Kernel regularizer : L2)
Dense layer : 512 (Relu)
Dropout layer : 0.2
Concat layer : 1536
Dense layer : 512 (Relu)
Dropout layer : 0.2
Concat layer : 2048
Dense layer : 512 (Relu)
Dropout layer : 0.2
Regressor
Skip connetction
Next Block
Next Layer->
Next Block

영준 딥러닝

#==========<∑◎ ㅐ ㈜Zon∈>==========

```
B1= layers.Dense(2**12,activation="relu")(B0)

B2= layers.Dense(2**11, activation="relu")(B1)

B3= layers.Dense(2**10, activation="relu")(B2)

B4= layers.Dense(2**9, activation="relu")(B3)

B5= layers.Dense(2**8, activation="relu")(B4)

B6= layers.Dense(2**7, activation="relu")(B5)

B7= layers.Dense(2**6, activation="relu")(B6)
```

```
Mean squared error:  116.52502828173681
R2 score:  0.6496400179456532
```

```
def createMLP_DenseNet():
    hiddenLayers = 512
    dropoutRate = 0.2

    B0_input = layers.Input(shape=train_x_oh.shape[1], name="B0_input")
    B0_embedding = layers.Dense(hiddenLayers * 2, activation="relu",
                                kernel_regularizer="l2", name="B0_embedding")(B0_input)

    B1_dense = layers.Dense(hiddenLayers, activation="relu", name="B1_dense")(B0_embedding)
    B1_dropout = layers.Dropout(rate=dropoutRate, name="B1_dropout")(B1_dense)
    B1_concat1 = layers.concatenate([B0_embedding, B1_dropout], name="B1_concat1")

    B2_dense = layers.Dense(hiddenLayers, activation="relu", name="B2_dense")(B1_concat1)
    B2_dropout = layers.Dropout(dropoutRate, name="B2_dropout")(B2_dense)
    B2_concat2 = layers.concatenate([B0_embedding, B1_dropout, B2_dropout], name="B2_concat2")

    B3_dense = layers.Dense(hiddenLayers, activation="relu", name="B3_dense")(B2_concat2)
    B3_dropout = layers.Dropout(dropoutRate, name="B3_dropout")(B3_dense)

    layer_final = layers.Dense(int(hiddenLayers/2), activation="relu", name="layer_final")(B3_dropout)

    layer_regressor = layers.Dense(1, name="Regressor")(layer_final)
    model_mlp = Model(B0_input, layer_regressor)
    model_mlp.compile(loss="mse", optimizer=optimizers.Adam(3e-3),
                      metrics=tf_metrics.RootMeanSquaredError(name="rmse"))
    # model_mlp.summary()
    return model_mlp
```

```
def createMLP_LP():
    hiddenLayers = 256
    dropoutRate = 0.2

    # Source : Kaggle - Laurent Pourchot
    # URL : https://www.kaggle.com/pourchot/simple-neural-network

    # === input layers ===
    B0_input = layers.Input(shape=train_x_oh.shape[1], dtype="float32", name="B0_input")
    B0_embedding = layers.Dense(units=hiddenLayers * 2, activation="relu",
                                kernel_regularizer="l2", name="B0_dense")(B0_input)

    # === learning layers ===
    B1_dense = tfa.layers.WeightNormalization(
        layers.Dense(
            units=hiddenLayers, activation="selu"), name="B1_dense"
    )(B0_embedding)
    B1_dropout = layers.Dropout(rate=dropoutRate, name="B1_dropout")(B1_dense)
    B1_concat = layers.Concatenate(name="B1_concat")([B0_embedding, B1_dropout])

    B2_dense = tfa.layers.WeightNormalization(
        layers.Dense(
            units=hiddenLayers, activation="relu"), name="B2_dense"
    )(B1_concat)
    B2_dropout = layers.Dropout(rate=dropoutRate, name="B2_dropout")(B2_dense)
    B2_concat = layers.Concatenate(name="B2_concat")([B0_embedding, B1_dropout, B2_dropout])

    B3_dense = tfa.layers.WeightNormalization(
        layers.Dense(
            units=hiddenLayers, activation="elu"), name="B3_dense"
    )(B2_concat)
    B3_dropout = layers.Dropout(rate=dropoutRate, name="B3_dropout")(B3_dense)

    # === top layers ===
    layer_final = layers.Dense(units=int(hiddenLayers / 2), activation="relu", name="layer_final")(B3_dropout)

    layer_regressor = layers.Dense(1, name="Regressor")(layer_final)
    model_mlp = Model(B0_input, layer_regressor)
    model_mlp.compile(loss="mse", optimizer=optimizers.Adam(3e-3),
                      metrics=tf_metrics.RootMeanSquaredError(name="rmse"))
    # model_mlp.summary()
    return model_mlp
```

{'RMSE': 77.43464976380788, 'R2': 0.8686687143576897}

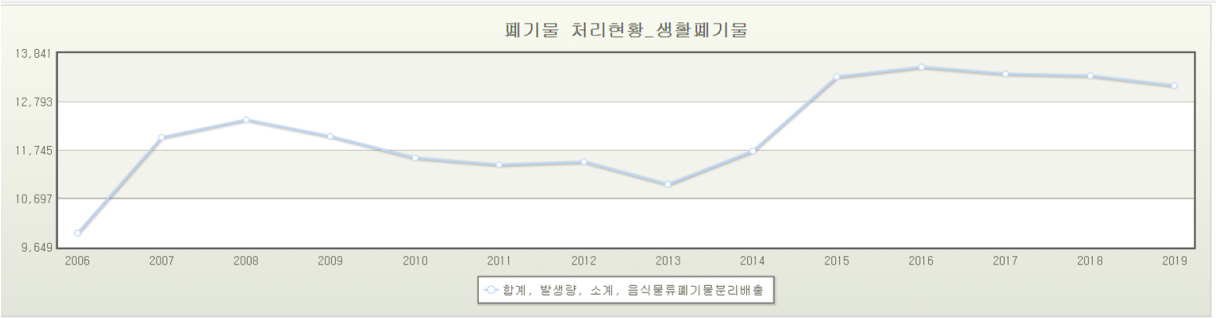
영준 오류 결과값.
아침에 결과값 수정예정.

인사이트

1.더이상 단순한 시계열 추세와 담당자의 직관적 경험이 아닌, 분석을 통해 얻어진 정확도 높은 예측을 통해 잔반 발생량을 획기적으로 줄일 수 있을 것으로 기대함.

2. 금번 프로젝트는 한국토지주택공사의 구내식당 식수를 기반으로 예측하였으나,

해당 모델을 국내 기업 구내식당 식수 인원 예측에도 적용시켜 동일한 효과를 충분히 기대해 볼 수 있을 거라고 예상함.



출처 KOSIS

참고 기사

https://futurechosun.com/archives/57579

‘ 연구에 따르면 음식물쓰레기는 전체 온실가스 배출량이 10%를 차지’

http://www.dailypop.kr/news/articleView.html?idxno=53395

유엔 환경계획(UN Environment Programme)의 2021년 폐기물 지수 보고서(Food Waste Index Report)에 따르면, 전 세계에서 버려진 음식물 쓰레기는 연 평균 9억3100만 톤

http://www.newstomato.com/ReadNews.aspx?no=1073042&inflow=N

‘ 환경부의 '전국 폐기물 발생 및 처리현황'에 따르면 2019년 기준 일평균 음식물 쓰레기 배출량은 1만4314톤’

보완사항/느낀점

이지예 : (이제 오피셜용 느낀점으로 적는다.)
[느낀점]: 프로젝트를 하면서 머릿속에 섞여있던 머신러닝과 딥러닝에 대한 개념이 이전보다 훨씬 정리가 된 것 같습니다. 매일 10시까지 집에도 못가고 야근했지만 파이팅 넘치는 팀원들 덕분에 전~~혀 힘들지 않았습니다. 영준샘... 스승의 은혜 감사합니다.... 덕분에 일주일간 정말 많이 배웠습니다~^^
[아쉬웠던점]: 개인적으로 개념 정립이 완전히 되지 않은 상태에서 프로젝트가 시작되어서, 시간 내에 혼자 힘으로 코드를 문제 없이 진행시키는 건 솔직히 힘들었을 것 같습니다. 능숙하게 직접 이것 저것 실행해보지 못한 점이 무척 아쉽습니다. 다음 프로젝트 전까지 공부를 많이 해둬야겠다고 일주일 내내 생각했습니다.

이예주: 김영준의 코드교실 추천합니다. 아무것도 모르던 우리 아이가 성장했어요^^! (여기 영준이한테 보내는 영상편지인가? 진짜 동영상 만든다 영상편지 처럼 아련한 자막 나오고 우리 가족 사진 나오고 영준이가 기겁하고 싫어할 것 같음) <그때 감사의의미로 꽃다발전달. 완전 감동~ 그때 울지마! 울지마 ! 외쳐주면 영준이가 울면됨 완벽! 나 지금r2 값 60대 나와서 미칠것같음~~~~ -> 이건 진짜 수정했으면 좋겠다.... 부끄럽다 => 영상 브금 스승의은혜 카네이션 옷에 달아드려야지.. - 카네이션 사옴 콜.. 영주니 내일 집에 카네이션 달고 가야돼
R2값 60 나오는게 부끄러워? 열심히 고쳐볼게
[알림] 점심시간에 영준이에게 보내는 영상편지 찍는 시간 있습니다. (확인했습니다. 영상편지 bgm으로는 이루마의kisstherain 어떠신지요?)

김남이:
[느낀점]: 좋은 팀원들을 만나서 팀플이 왜 이렇게 재밌어? 하고 계속 얘기할 수 있었던 시간이었습니다. 덕분에 머신러닝에 대해 잘 이해할 수 있게 되었고, 제가 공부를 해야할 것이 아직도 너무 많다는 것을 느꼈습니다. 팀원들 다들 의견 충돌도 없이 얘기를 잘 들어주셔서 감사한 시간이었습니다.
[아쉬웠던 점]: 뇌내 알고리즘을 코드로 구현하는 것이 빨리 됐다면 더 많은 것을 시도해볼 수 있었을걸 하는 아쉬움이 있습니다. 물리적 시간을 갈아넣는 연구가 지속적으로 필요할 것 같습니다.

김영준
[느낀점] : 데이터 분석의 전체적인 프로세스를 진행해 볼 수 있었던 점이 좋았습니다. 또한 feature engineering 과정에서 팀원들과 브레인스토밍을 하며 아이디어를 공유하는 과정이 매우 유익했습니다. 전처리 과정에 오류가 있어 해당 부분부터 코드를 다시 실행시켜야하는 경우가 있었는데, 팀원들이 지치지 않고 따라와주어서 고마웠습니다.
[보완사항] : feature engineering 과정 및 시각화 해석 부분에서는 팀원 각자의 의견이 서로 공유가 잘 되었으나, 코드 이해에 대한 부분에서는 공유가 덜 된 부분이 있어 조금 아쉬웠습니다. 또한 시간이 더 있었으면 ElasticNet, SVM, KNN 모델들도 시도해 볼 수 있었으나 그럴 수 없었던 부분도 아쉬웠습니다.

임베딩 레이어 설명
-> Dense layer 와 연산은 같으나
return 을 input 에 연결된 가중치를 벡터라이즈하여 리턴

input_dim : 임베딩을할 총 vocabrary 사이즈를 지정합니다.
output_dim : 실수로 변환될 차원의 수를 지정합니다.
input_length : Input tensor의 shape를 지정합니다.

exceptinon control
input 타입에 대해 강제 int 형변환을 수행 합니다. -> 내림
input_dim 을 넘는 값이 들어오면 error 를 발생시킵니다.

Convolution layer로 feature를 한 번더 뽑은 후 flatten 합니다.
dropout을 수행합니다.

학습 완료 후 flatten 층만 추출하여 predict 합니다.
각 메뉴에 따른 4차원의 Dense Vector를 리턴합니다.

데이터 출처

데이콘 - 구내식당 식수 인원 예측 AI 경진대회

https://dacon.io/competitions/official/235743/data

기상청외부데이터
https://data.kma.go.kr/cmmn/main.do

<일자 관련 컬럼>

1 일자 - 년, 계절, 월, 주, 일, 요일 정보 추출

2. 2020년 전후 여부 컬럼 - 코로나 관련
(2020전:1 / 2020후:0)

3. 공휴일 전수 컬럼
다음날이 공휴일 이거나 공휴일 다음날 인 경우 1
아니면 0

4. 일자 컬럼의 주기성 데이터를 추출 하기 위해 연중 시간에 Sin, Cos 연산을 적용하였습니다. 시간을 명확한 "하루 중 시간" 및 "연중 시간" 신호로 변환하는 것입니다.

산식 : (np.sin(timestamp_s * (2 * np.pi / day)))

<인원수 관련 컬럼>

5. '식사가용인원' = '본사정원수'-('본사휴가자수'+'본사출장자수'+'현본사소속재택근무자수')
6. '야근비율' = round('본사시간외근무명령서승인건수' / '식사가용인원', 3)
7. '휴가비율' = round('본사휴가자수' / '본사정원수', 3)
8. '출장비율' = round('본사출장자수' / '본사정원수', 3)
9. '재택비율' = round('현본사소속재택근무자수' / '본사정원수', 3)

메뉴 관련 컬럼

10. ‘중식메뉴’ 컬럼을 밥, 국, 메인메뉴, 반찬1, 반찬2, 김치, 사이드 컬럼으로 분리 (이후 밥, 김치 컬럼은 Drop)

11. 특식 컬럼 추가 (if ‘밥’ in x[‘밥’])
특식일 경ㅇ우1, 특식이 아닐 경우 0

12. 신메뉴 컬럼 추가 (if ‘new’ in x[‘중식메뉴’])
New가 중식메뉴 문자열 안에 있으면 1, 없으면 0

13. 메뉴관련컬럼(5개)을 tensorflow embedding layer를 통해 dense vector로 변환하기

```
tf.random.set_seed(56)
menu_vec = ["국", "메인메뉴", "반찬1", "반찬2", "사이드"]
    # 모델 구조
    # ----- Embedding layers -----
    B0_input = layers.Input(shape=(#), name="B0_input")
    B0_embedding = layers.Embedding(input_dim=512,
                                    output_dim=64,
                                    name="B0_embedding")(B0_input)
    # ----- Convolution layers -----
    B1_conv1d = layers.Conv1D(4, 1, activation='relu', name="B1_conv1d")(B0_embedding)
    B1_flatten = layers.Flatten(name="extract")(B1_conv1d)
    B1_dropout = layers.Dropout(0.25, name="B1_dropout")(B1_flatten)
    # ----- Regressor -----
```

메뉴 관련 컬럼

10. ‘중식메뉴’ 컬럼을 밥, 국, 메인메뉴, 반찬1, 반찬2, 김치, 사이드 컬럼으로 분리 (이후 밥, 김치 컬럼은 Drop)

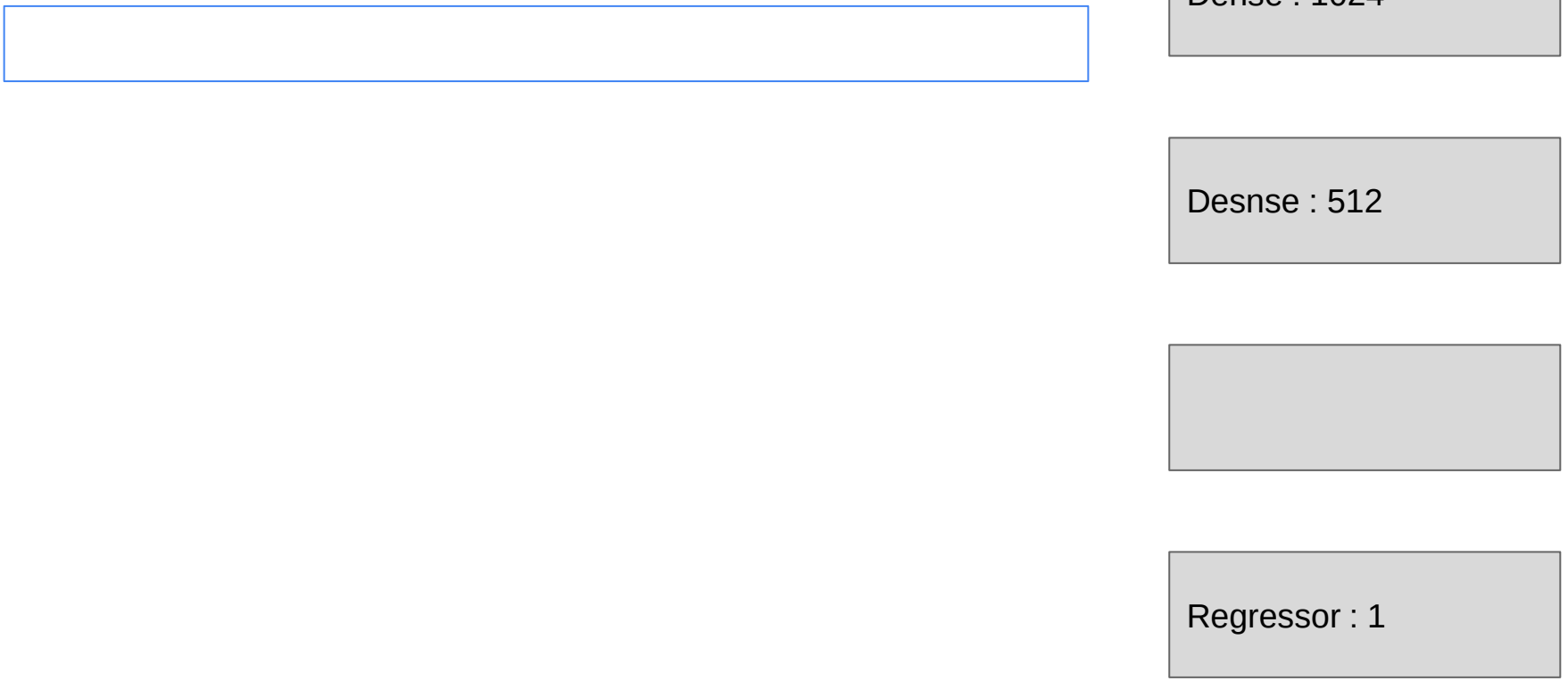
11. 특식 컬럼 추가 (if ‘밥’ in x[‘밥’])
특식일 경우 0, 특식이 아닐 경우 1

12. 신메뉴 컬럼 추가 (if ‘new’ in x[‘중식메뉴’])
New가 중식메뉴 문자열 안에 있으면 1, 없으면 0

13. 메뉴관련컬럼(5개)을 tensorflow embedding layer를 통해 dense vector로 변환하기
tf.random.set_seed(56)
menu_vec = ["국", "메인메뉴", "반찬1", "반찬2", "사이드"]

14. 기상청 데이터 추가 (기온 및날씨)
점심식수요에 영향을 주는 날씨는 11,12,13 으로 간주
기온은 해당일 11,12,13 시 평균으로 계산 (결측치는 전날 기온으로 대치)
강수량은 해당일 11,12,13 시 최대값으로 계산 (결측치는 전날 강수량으로 대치)

15. 강수여부 컬럼 추가
강수량 > 0 이면 1, 아니면 0


Dense : 1024
Desnse : 512
Regressor : 1