머신러닝을 활용한 여론조사 무응답 예측

김무관

analyze79@naver.com

Contents

프로젝트 소개

✓여론조사 관련 기관에서는 여론조사의 결과 및 선거예측의 정확성 제고를 위해 지속적으로 노력중이다.

✓ 유권자들의 선택 및 정당이나 후보의 적절한 선거전략 수립을 위해서 여론조사를 통한 객관적이고 신뢰성 있는 정보는 매우 중요하다.

✓ 이번 연구에서는 예측 정확도를 높이기 위해 머신러닝을 활용 지지후보가 "없다/무응답"이라고 응답한 무응답층을 분류해보자.

대통령 지지 후보(여론조사 결과)	빈도	퍼센트
1 더불어민주당 문재인 후보	***	**.**
2 자유한국당 홍준표 후보	***	**.**
3 국민의당 안철수 후보	***	**.**
4 바른정당 유승민 후보	***	**.**
5 정의당 심상정 후보	***	**.**
6 기타 후보	***	**.**
8 지지하는 후보가 없다	148	4.94
9 모르겠다	162	5.39
총계	3000	100.00

대통령 지지 후보(여론조사 무응답 예측)	빈도	퍼센트
1 더불어민주당 문재인 후보		
2 자유한국당 홍준표 후보		
3 국민의당 안철수 후보		
4 바른정당 유승민 후보		
5 정의당 심상정 후보		
6 기타 후보		
총계	3000	100.00

분석 프로그램

✓ **Python, Orange 프로그램으로 머신러닝 적용**

데이터 설명

✓ **19대 대통령선거 2~3일전에 시행된 전화여론조사(3000명)**

※ 회사 내부 데이터라서 외부공유 제한됨(연구목적 사용)

< 변수구성 >

ID	응답자 구별코드
SIDO	시도 구분
AREA	권역별
SEX	성별
AGE1	연령(숫자)
AGE	연령대
Q2	지지 후보 결정 시기
Q3	지지 후보 선택 이유
Q4	지지 정당(재질문 통합)
DQ1	정치적 이념성향
DQ2	과거 대선 지지
DQ3	과거 총선 정당 지지
DQ4	직업
DQ5	소득

독립변수

Q1	대통령 지지 후보(재질문 통합)

종속변수(target)

데이터 확인 및 전처리

✓ 데이터 확인

```
## 결측치 확인
df.isna().sum()
```

```
ID          0
SIDO        0
AREA        0
SEX         0
AGE1        0
AGE         0
Q1          0
Q2        329
Q3        329
Q4          0
DQ1         0
DQ2         0
DQ3         0
DQ4         0
DQ5         0
WT          0
TYPE        0
dtype: int64
```

<지지후보 응답데이터>

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2757 entries, 0 to 3085
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ID      2757 non-null   int64
 1   SIDO    2757 non-null   int64
 2   AREA    2757 non-null   int64
 3   SEX     2757 non-null   int64
 4   AGE1    2757 non-null   int64
 5   AGE     2757 non-null   int64
 6   Q1      2757 non-null   int64
 7   Q2      2757 non-null   float64
 8   Q3      2757 non-null   float64
 9   Q4      2757 non-null   int64
 10  DQ1     2757 non-null   int64
 11  DQ2     2757 non-null   int64
 12  DQ3     2757 non-null   int64
 13  DQ4     2757 non-null   int64
 14  DQ5     2757 non-null   int64
 15  WT      2757 non-null   float64
 16  TYPE    2757 non-null   int64
dtypes: float64(3), int64(14)
memory usage: 387.7 KB
```

<지지후보 없음/무응답데이터>

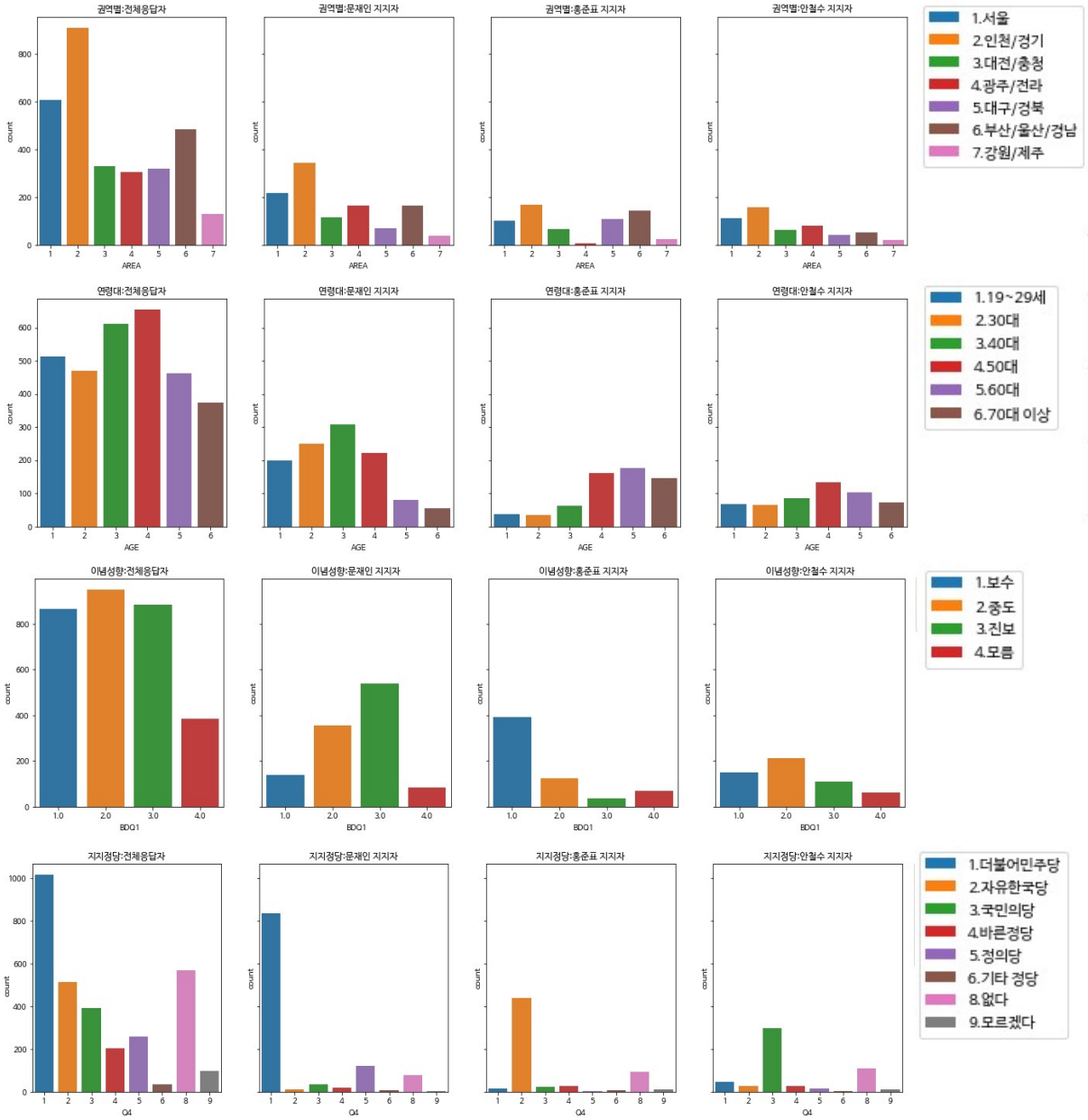
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 329 entries, 3 to 3077
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ID      329 non-null    int64
 1   SIDO    329 non-null    int64
 2   AREA    329 non-null    int64
 3   SEX     329 non-null    int64
 4   AGE1    329 non-null    int64
 5   AGE     329 non-null    int64
 6   Q1      329 non-null    int64
 7   Q2      0 non-null      float64
 8   Q3      0 non-null      float64
 9   Q4      329 non-null    int64
 10  DQ1     329 non-null    int64
 11  DQ2     329 non-null    int64
 12  DQ3     329 non-null    int64
 13  DQ4     329 non-null    int64
 14  DQ5     329 non-null    int64
 15  WT      329 non-null    float64
 16  TYPE    329 non-null    int64
dtypes: float64(3), int64(14)
memory usage: 46.3 KB
```

- Q2, Q3 변수에 결측치가 있는데, Q1(종속변수) 지지후보 "없음/무응답" 사례수에 해당함

- 독립변수에서 Q2, Q3 삭제 필요함(train 데이터셋, test 데이터셋 동일조건 맞추기 위해서)

데이터 확인 및 전처리

✓ 데이터 확인

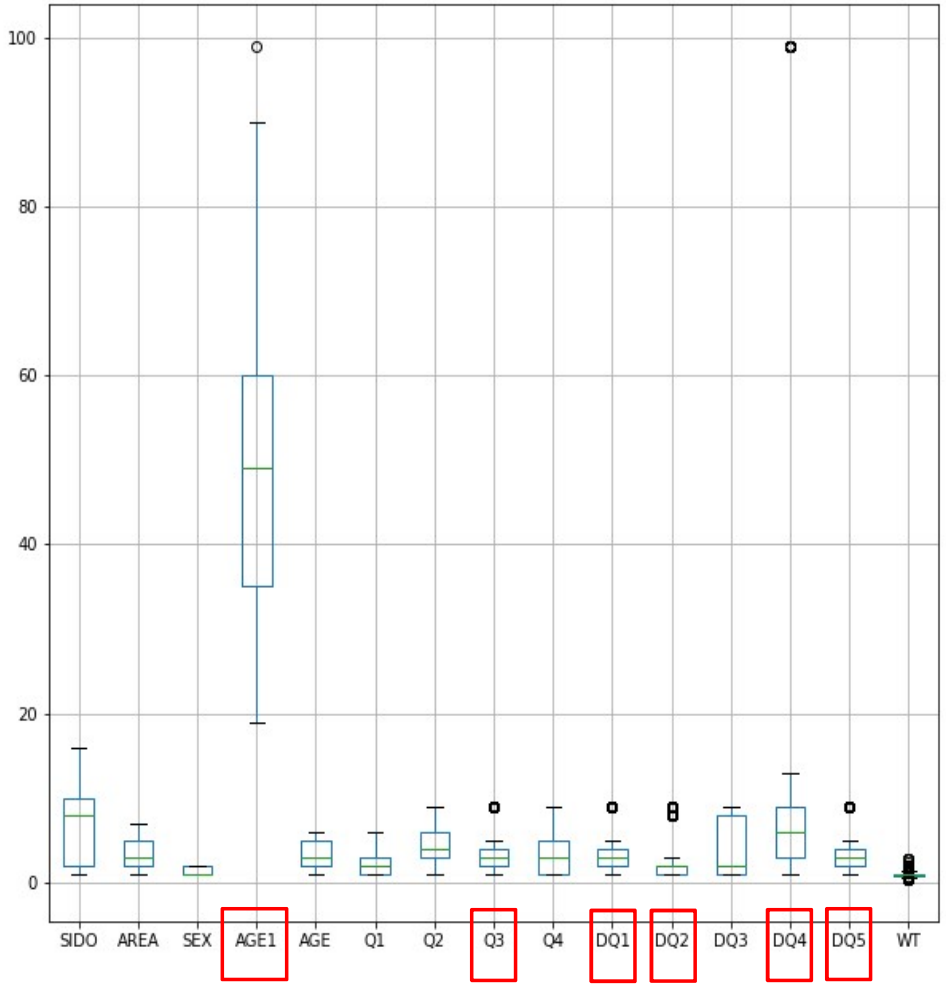


	지역	연령대	이념성향	지지정당
문재인	광주/전라	20~40대	진보	더불어민주당
홍준표	대구/경북	50대 이상	보수	자유한국당
안철수	골고루 (특히 광주/전라)	골고루 (특히 50대)	골고루 (특히 중도)	국민의당

=>지지 후보별로 독립변수 확인결과 상대적으로 아래내용에서 높은 응답을 확인할 수 있었음

데이터 확인 및 전처리

✓ 데이터 전처리(변수값 수정)



=> 독립변수 일부 변수들 극단치 확인됨

```
df['DQ4'].value_counts()  # 99값 모르겠다 (사례수 14개)
```

```
9     661
6     652
2     544
10    235
3     189
12    181
8     155
4     131
5     107
1      84
11     64
7      45
13     24
99     14
Name: DQ4, dtype: int64
```

```
df.loc[(df.DQ4==99),'DQ4']=13
df['DQ4'].value_counts()
```

```
9     661
6     652
2     544
10    235
3     189
12    181
8     155
4     131
5     107
1      84
11     64
7      45
13     38
Name: DQ4, dtype: int64
```

=> DQ4 99 '모르겠다' 값이 14개 있는데, 사례수도 적고 13 '기타'가 있어서 99값을 13으로 수정함

데이터 확인 및 전처리

✓ 데이터 전처리(3개 변수 생성)

```
## 성(SEX) & 연령(AGE) 으로 성연령(BSEXAGE) 변수 생성
df.loc[(df.SEX==1) & (df.AGE==1),'BSEXAGE']=1     #1.남-20대
df.loc[(df.SEX==1) & (df.AGE==2),'BSEXAGE']=2     #2.남-30대
df.loc[(df.SEX==1) & (df.AGE==3),'BSEXAGE']=3     #3.남-40대
df.loc[(df.SEX==1) & (df.AGE==4),'BSEXAGE']=4     #4.남-50대
df.loc[(df.SEX==1) & (df.AGE==5),'BSEXAGE']=5     #5.남-60대
df.loc[(df.SEX==1) & (df.AGE==6),'BSEXAGE']=6     #6.남-70대 이상

df.loc[(df.SEX==2) & (df.AGE==1),'BSEXAGE']=7     #7.여-20대
df.loc[(df.SEX==2) & (df.AGE==2),'BSEXAGE']=8     #8.여-30대
df.loc[(df.SEX==2) & (df.AGE==3),'BSEXAGE']=9     #9.여-40대
df.loc[(df.SEX==2) & (df.AGE==4),'BSEXAGE']=10    #10.여-50대
df.loc[(df.SEX==2) & (df.AGE==5),'BSEXAGE']=11    #11.여-60대
df.loc[(df.SEX==2) & (df.AGE==6),'BSEXAGE']=12    #12.여-70대 이상
```

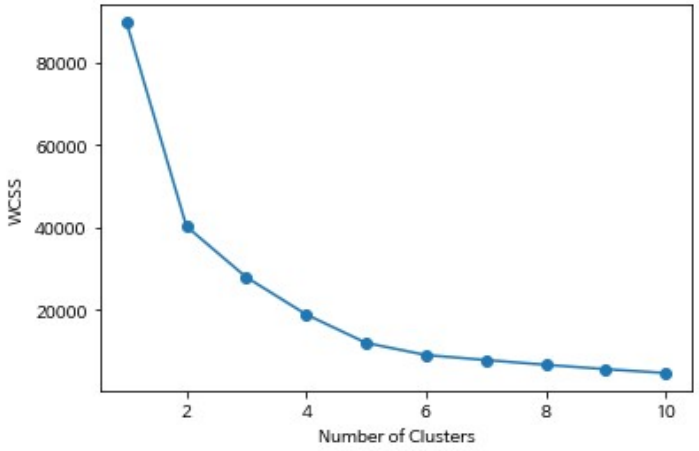
```
## DQ1(정치이념성향) 세부적인 분류를 묶어서 변수 생성
df.loc[(df.DQ1==1) | (df.DQ1==2),'BDQ1']=1   #1.보수(매우 보수/보수)
df.loc[(df.DQ1==3),'BDQ1']=2                 #2.중도
df.loc[(df.DQ1==4) | (df.DQ1==5),'BDQ1']=3   #3.진보(매우 진보/진보)
df.loc[(df.DQ1==9),'BDQ1']=4                 #4.모름
```

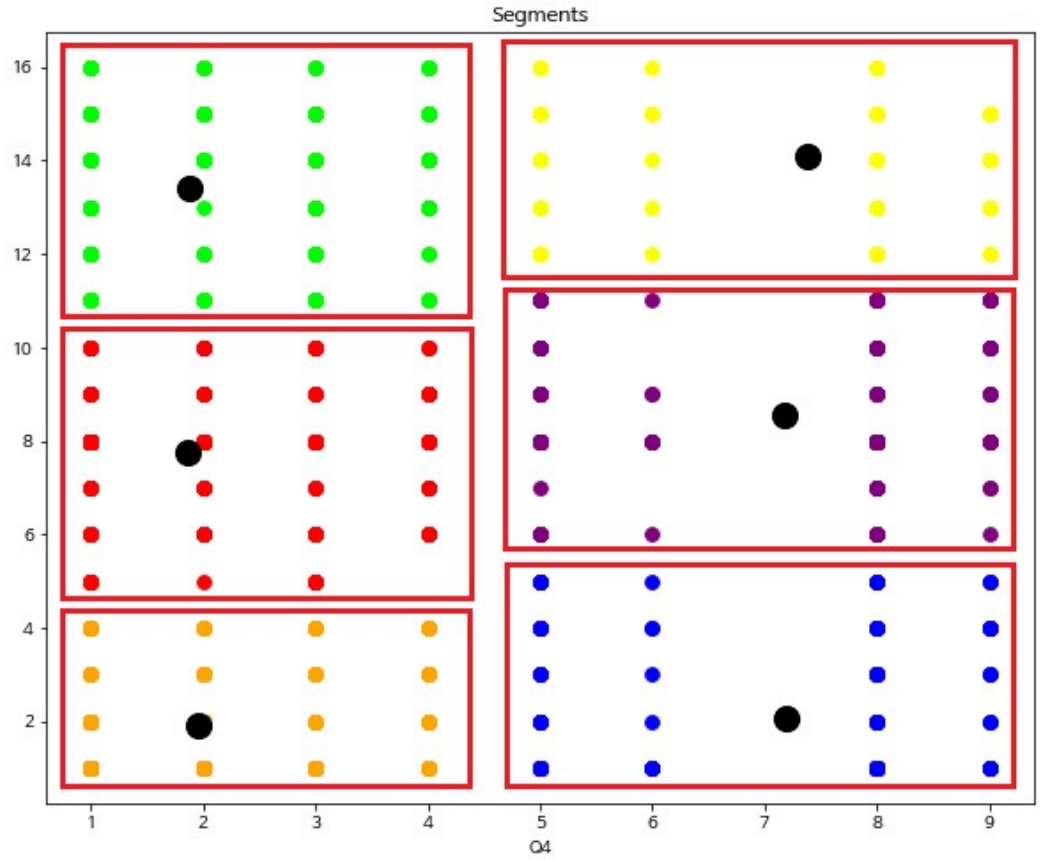
```
## DQ4(직업) 세부적인 분류를 묶어서 변수 생성
df.loc[(df.DQ4==1),'BDQ4']=1                                  #1.농/임/어업
df.loc[(df.DQ4==2),'BDQ4']=2                                  #2.자영업
df.loc[(df.DQ4==3) | (df.DQ4==4) | (df.DQ4==5),'BDQ4']=3      #3.블루칼라
df.loc[(df.DQ4==6) | (df.DQ4==7) | (df.DQ4==8),'BDQ4']=4      #4.화이트칼라
df.loc[(df.DQ4==9),'BDQ4']=5                                  #5.가정주부
df.loc[(df.DQ4==10),'BDQ4']=6                                 #6.학생
df.loc[(df.DQ4==11) | (df.DQ4==12) | (df.DQ4==13),'BDQ4']=7   #7.무직/은퇴/기타
df.loc[(df.DQ4==99),'BDQ4']=8                                 #8.모름
```

데이터 확인 및 전처리

✓ 데이터 전처리(SEG 변수생성)

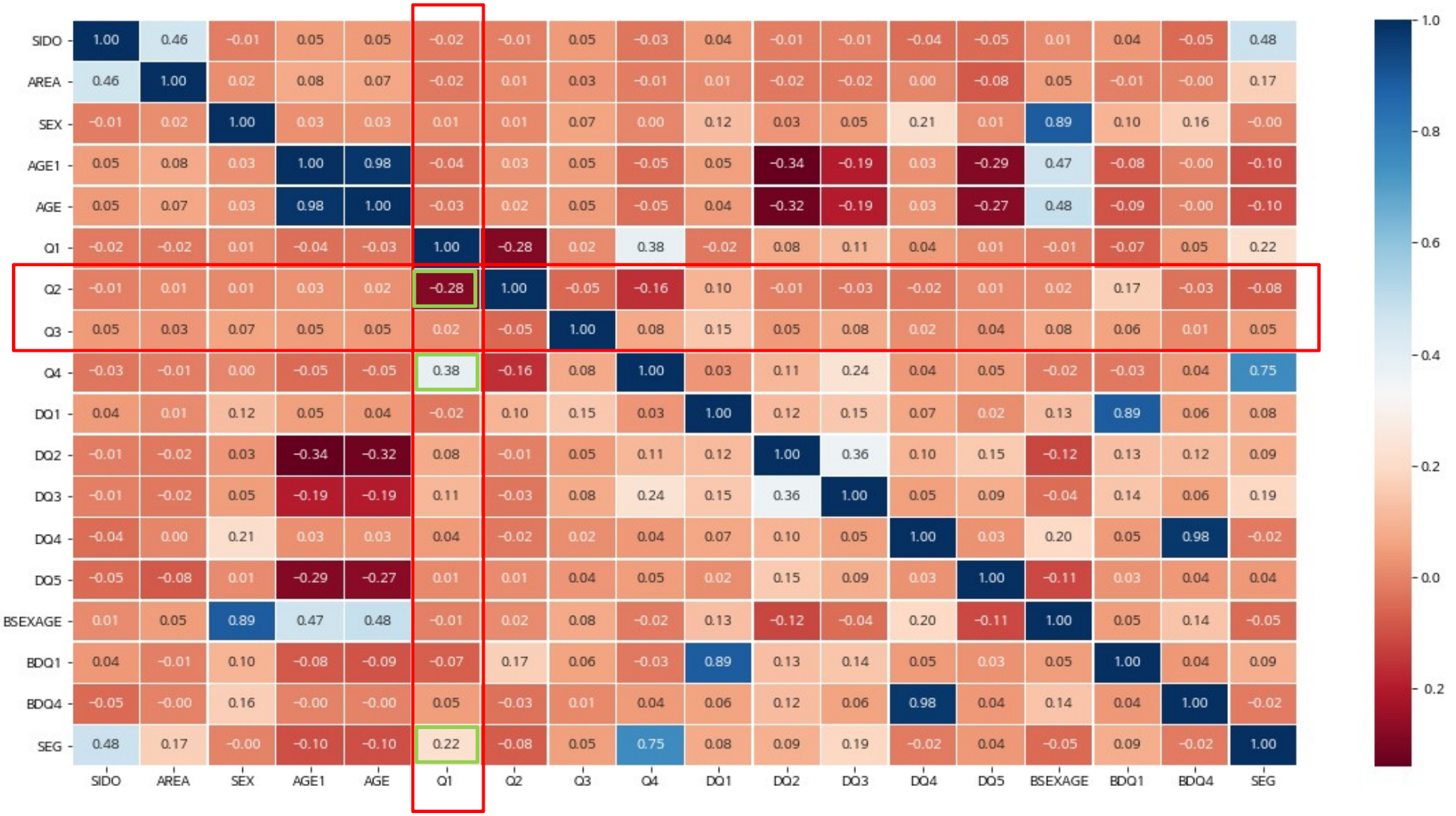
=> K-means Clustering (모델 변수중요도에서 높게 나온 'Q4', 'SIDO' 변수로 시도함)





데이터 확인 및 전처리

✓ 데이터 전처리(상관분석 확인)



=>종속변수 Q1과 상관관계가 높은 것은 Q2, Q4, SEG 변수임

데이터 확인 및 전처리

✓ 데이터 전처리(다중공선성 확인)

```
# 지지후보 응답자 기준
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['Features'] = df33.columns
vif['vif'] = [variance_inflation_factor(
              df33.values, i) for i in range(df33.shape[1])]
vif.sort_values(by='vif',ascending=False)
```

	Features	vif
4	AGE	223.224944
2	SEX	164.560021
12	BSEXAGE	106.254262
3	AGE1	25.574487
10	DQ4	24.076133
14	BDQ4	23.710797
15	SEG	6.092196
13	BDQ1	5.636196
7	DQ1	5.551146
6	Q4	5.113313
0	SIDO	2.949593
8	DQ2	1.310438
1	AREA	1.297882
9	DQ3	1.236737
5	Q1	1.206890
11	DQ5	1.106214

	Features	vif
10	BDQ4	6.845002
2	AGE	6.723444
8	BSEXAGE	5.753967
9	BDQ1	5.487408
1	AREA	4.784120
3	Q1	4.242030
0	SIDO	4.043213
4	Q4	3.229976
6	DQ3	3.135512
7	DQ5	2.945329
5	DQ2	2.693677

다중공선성 : 독립 변수들이 서로 독립이 아니라 상호상관관계가 강한 경우에 발생함

=> vif가 10이상이면 다중공선성이 존재해서 확인 후, 삭제함.

* [SEX, AGE1, AGE, BSEXAGE]
  -> [SEX, AGE1] 독립변수에서 삭제

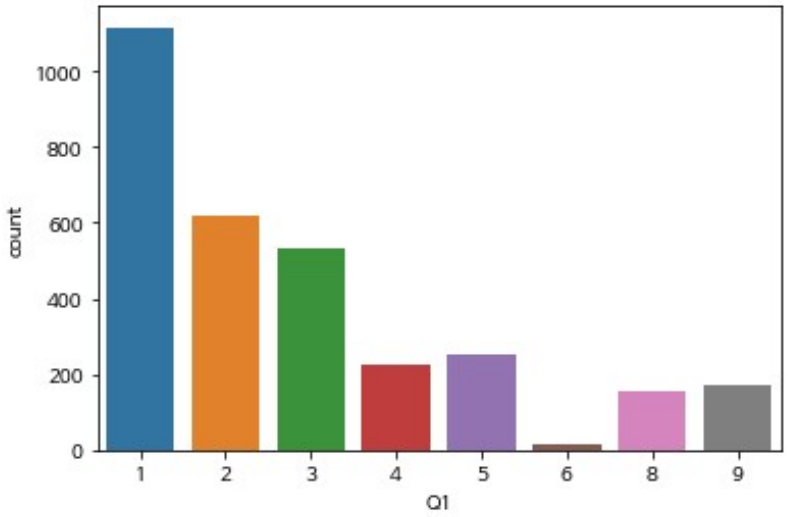
* [DQ1, BDQ1] -> [DQ1] 독립변수에서 삭제

* [DQ4, BDQ4] -> [DQ4] 독립변수에서 삭제

머신러닝 모델링

✓ 종속변수 확인과 다중분류 모델별 accuracy 확인

종속변수 확인결과 범주형변수(명목변수)로 다중분류 모델 적용이 필요함!




```
--------------------------------------------------------------------------------
LogisticRegression - accuracy score :  0.493
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
KNeighborsClassifier - accuracy score :  0.517
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
DecisionTreeClassifier - accuracy score :  0.5673
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
RandomForestClassifier - accuracy score :  0.6739
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
AdaBoostClassifier - accuracy score :  0.6109
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
XGBClassifier - accuracy score :  0.702
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LGBMClassifier - accuracy score :  0.6812
--------------------------------------------------------------------------------
```

=> XGBClassifier 모델이 가장 정확도가 높음

✓ XGBBoost (XGBClassifier, XGBRegressor) 모델 특징

XGBoost(eXtra Gradient Boost)

트리 기반의 알고리즘의 앙상블 학습에서 각광받는 알고리즘 중 하나
GBM에 기반하고 있지만, GBM의 단점인 느린 수행시간, 과적합 규제 등을 해결한 알고리즘

XGBoost의 주요장점

(1) 뛰어난 예측 성능
(2) GBM 대비 빠른 수행 시간
(3) 과적합 규제(Overfitting Regularization)
(4) Tree pruning(트리 가지치기) : 긍정 이득이 없는 분할을 가지치기해서 분할 수를 줄임
(5) 자체 내장된 교차 검증

- 반복 수행시마다 내부적으로 교차검증을 수행해 최적회된 반복 수행횟수를 가질 수 있음
  지정된 반복횟수가 아니라 교차검증을 통해 평가 데이트세트의 평가 값이 최적화되면 반복을 중간에 멈출 수 있는 기능이 있음

(6) 결손값 자체 처리

XGBoost는 독자적인 XGBoost 모듈과 사이킷런 프레임워크 기반의 모듈이 존재합니다.
독자적인 모듈은 고유의 API와 하이퍼파라미터를 사용하지만, 사이킷런 기반 모듈에서는 다른 Estimator와 동일한 사용법을 가지고 있음

* 자료 출처 : https://injo.tistory.com/44

머신러닝 모델링

✓ XGBClassifier 모델 최적화

```
## XGBClassifier : 하이퍼 파라미터 튜닝

from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

xgb = XGBClassifier()

# 파라미터를 딕셔너리 형태로 설정
parameters = {'max_depth' : [2, 3, 4, 5, 6],
              'subsample' : [0.6, 0.7, 0.8, 0.9, 1.0]}

grid_xgb = GridSearchCV(xgb, param_grid=parameters, cv=3, refit=True)

# 하이퍼 파라미터를 순차적으로 학습, 평가
grid_xgb.fit(X_train, y_train)

# GridSearchCV 결과를 추출해 데이터 프레임으로 반환
scores_df = pd.DataFrame(grid_xgb.cv_results_)

print('GridSearch - XGB 최적 파라미터: ', grid_xgb.best_params_)
print('GridSearch - XGB 최고 점수: ', grid_xgb.best_score_)

# GridSearchCV의 refit으로 학습된 estimator 반환
estimator = grid_xgb.best_estimator_

# GridSearchCV의 best_estmator_ 는 이미 최적 학습이 됐으므로 별도 학습이 필요 없음
pred = estimator.predict(X_test)
print('XGB 테스트 데이터세트 정확도: {0: .4f}'.format(accuracy_score(y_test, pred)))
```

```
GridSearch - XGB 최적 파라미터:  {'max_depth': 2, 'subsample': 1.0}
GridSearch - XGB 최고 점수:  0.7092970521541795
XGB 테스트 데이터세트 정확도:  0.7355
```

```
from xgboost import XGBClassifier
xgb_model = XGBClassifier(base_score=0.5,
                          booster='gbtree',
                          colsample_bylevel=1,
                          colsample_bynode=1,
                          colsample_bytree=1,          # 범위 0~1(크면 과적합 발생)
                          gamma=0,                     # =>값을 높이면 과적합 제어
                          learning_rate=0.1,
                          max_delta_step=0,
                          max_depth=2,                 # 통상 3~10 적용(크면 과적합 발생)
                          min_child_weight=1,          # =>값을 높이면 과적합 제어
                          missing=None,
                          n_estimators=100,
                          n_jobs=1,
                          nthread=None,
                          objective='multi:softmax',   # multi:softmax(다중 분류)
                          random_state=42,
                          reg_alpha=0,
                          reg_lambda=1,
                          scale_pos_weight=1,
                          silent=None,
                          subsample=1.0,               # 통상 0.5~1 적용(과적합 제어)
                          verbosity=1)

xgb_pred = xgb_model.fit(X_train, y_train).predict(X_test)
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
1	0.80	0.88	0.84	247
2	0.84	0.86	0.85	120
3	0.62	0.63	0.62	101
4	0.46	0.36	0.41	36
5	0.35	0.18	0.24	45
6	0.00	0.00	0.00	3
accuracy			0.74	552
macro avg	0.51	0.49	0.49	552
weighted avg	0.71	0.74	0.72	552

머신러닝 모델링

✓ XGBClassifier 모델로 예측분석

```
X_train = df1.drop('Q1', axis = 1) # Q1 응답데이터
X_test = df1['Q1']

y_train = df2.drop('Q1', axis = 1) # Q1 무응답데이터
y_test = df2['Q1']

xgb_pred = xgb_model.fit(X_train, X_test).predict(y_train)
print(xgb_pred)
```

```
[4 2 2 2 3 3 1 5 2 1 1 5 1 3 3 3 2 2 1 2 3 5 2 2 2 2 3 2 4 2 3 1 3 3 4 3 2
 2 2 3 2 1 2 3 4 1 4 3 3 3 5 2 2 2 2 1 3 1 2 3 3 1 3 1 2 1 2 2 5 2 2 5 3 2
 1 1 2 5 3 3 5 3 1 4 2 1 2 3 5 2 2 3 1 3 2 3 1 2 2 1 3 5 1 3 3 1 3 1 1 2 5
 2 1 2 2 3 2 2 1 2 2 5 2 3 1 3 3 1 1 3 2 2 5 1 2 4 2 2 2 1 2 3 5 2 2 1 1 3
 3 2 2 2 3 5 3 3 4 3 2 2 1 1 1 3 3 5 5 2 4 3 3 5 5 3 2 3 3 1 3 2 3 2 2 2 1
 1 3 1 2 2 1 2 2 3 1 2 2 1 3 2 1 2 5 2 1 2 1 2 5 5 2 1 1 2 2 3 1 4 2 1 2 2
 2 4 2 4 1 5 4 3 3 5 2 3 2 1 2 3 1 5 1 1 3 5 2 5 2 5 2 1 2 1 2 2 2 3 2 1 5
 2 1 3 2 5 3 5 3 3 3 3 5 1 1 2 1 1 5 3 2 1 5 5 3 5 2 2 2 2 3 3 2 3 2 3 3 2
 3 1 2 4 3 2 5 3 3 1 2 1 5 1 2 5 3 1 4 3 2 3 2 2 3 5 2 1 2 2 5 3 2]
```

ID와 Q1 예측값 결합해서 데이터 생성

	ID	Q1
3	4	4
14	15	3
15	16	3
16	17	2
29	30	3
41	42	3
48	49	1
66	67	5
67	68	2
71	72	1
82	83	1
84	85	1
96	97	1
98	99	3
105	106	3
120	121	3

보팅/스태킹 적용

✓ 보팅(Voting)

- 동일 데이터셋으로 여러 개의 모델로 학습을 진행해서 투표

```
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 개별 모델
xgb_clf = XGBClassifier(objective='multi:softmax', max_depth=2, subsample=1.0, random_state =42)
lgbm_clf = LGBMClassifier(objective='multi:softmax', max_depth=3, subsample=0.6, random_state =42)

# 개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기
vo_clf = VotingClassifier(estimators=[('XGB', xgb_clf), ('LGBM', lgbm_clf)], voting='soft')

vo_clf.fit(X_train, y_train)
pred = vo_clf.predict(X_test)
print('Voting 분류기 정확도: {0:.4f}'.format(accuracy_score(y_test, pred)))

classifiers = [xgb_clf, lgbm_clf]
for classifier in classifiers:
    classifier.fit(X_train, y_train)
    pred = classifier.predict(X_test)
    class_name= classifier.__class__.__name__
    print('{0} 정확도: {1:.4f}'.format(class_name, accuracy_score(y_test , pred)))
```

```
Voting 분류기 정확도: 0.7428
XGBClassifier 정확도: 0.7355
LGBMClassifier 정확도: 0.7446
```

✓ 스태킹(Stacking)

- 모델 예측값으로 실제값을 다시 예측하는 기법

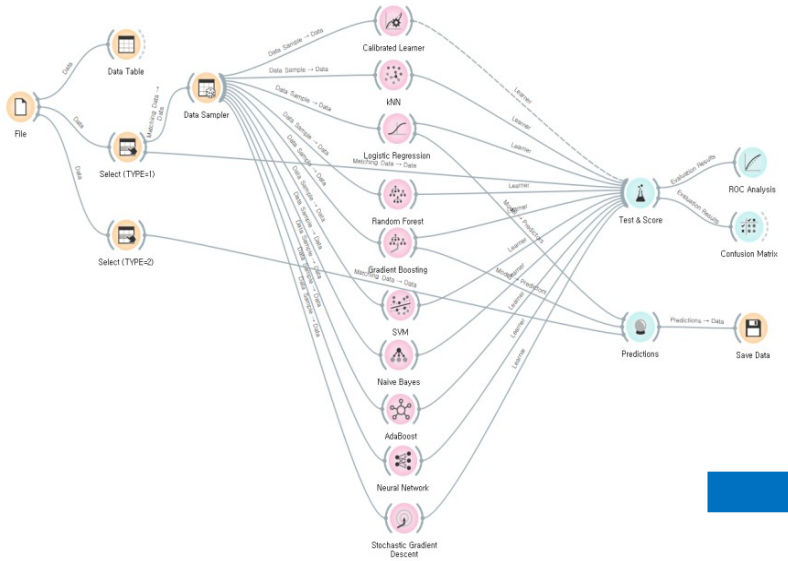
```
from vecstack import stacking
models = [XGBClassifier(objective='multi:softmax', max_depth=2, subsample=1.0, random_state =42),
          LGBMClassifier(objective='multi:softmax', max_depth=3, subsample=0.6, random_state =42),
          RandomForestClassifier(random_state = 42)]

S_train, S_test = stacking(models, X_train, y_train, X_test, regression = False, metric = ['acc'], n_folds = 5)
rfg = RandomForestClassifier()
rfg.fit(S_train, y_train)
stack_y_pred = rfg.predict(S_test)
np.mean(stack_y_pred == y_test)
```

```
0.7409420289855072
```

예측결과 검증

✓ 검증시도1 : Orange 프로그램과 결과비교



Evaluation Results

Model	AUC	CA	F1	Precision	Recall
kNN	0.793	0.624	0.578	0.572	0.624
SVM	0.847	0.650	0.635	0.627	0.650
SGD	0.786	0.701	0.690	0.685	0.701
Random Forest	0.865	0.696	0.679	0.675	0.696
Neural Network	0.835	0.639	0.632	0.626	0.639
Naive Bayes	0.870	0.629	0.640	0.657	0.629
Logistic Regression	0.884	0.708	0.693	0.686	0.708
Gradient Boosting	0.880	0.712	0.699	0.694	0.712
AdaBoost	0.847	0.661	0.645	0.641	0.661

				7월30일
F1	0.694	0.686	F1	0.74
ID	Gradient Boosting	Logistic Regression	ID	파이썬-xgboost
4	4	4	4	4
15	2	3	15	3
16	2	2	16	3
17	2	2	17	2
30	3	3	30	3
42	3	3	42	3
49	2	2	49	1
67	5	3	67	5
68	2	2	68	2
72	1	1	72	1
83	5	1	83	1
85	5	3	85	1
97	1	1	97	1
99	3	3	99	3
106	3	3	106	3
121	2	3	121	3
126	2	2	126	2
127	2	2	127	2
136	1	1	136	1
142	2	2	142	2
158	3	3	158	3
160	1	1	160	5
164	3	2	164	3

7월30일

*파이썬-xgboost VS Gradient Boosting

	사례수	비율
TRUE(같음)	212	64.44
FALSE(다름)	117	35.56
	329	100.00

*파이썬-xgboost VS Logistic Regression

	사례수	비율
TRUE(같음)	223	67.78
FALSE(다름)	106	32.22
	329	100.00

=> 오렌지, 파이썬 프로그램으로 예측한 데이터 값이 서로 어느 정도 동일한지 확인함 (65% 내외로 동일)

예측결과 검증

✓ **검증시도2 : 제19대 대선 개표결과와 여론조사결과(무응답보정 포함) 비교**

(SPSS 프로그램으로 보정데이터 적용한 테이블 결과임)

	개표결과	여론조사	오렌지 (무응답보정)	파이썬 (무응답보정)	
문재인	41.08	37.**	39.81	39.93	<--보정으로 개표결과에 가까워짐
홍준표	24.03	18.**	22.27	21.33	
안철수	21.41	17.**	19.70	20.05	
유승민	6.76	7.**	7.73	7.72	
심상정	6.17	8.**	9.97	10.46	<--과잉 보정됨
기타		0.**	.52	.52	<--보정값에 6이 없음
없음/모름		10.33			

결론 및 제언

✓선거여론조사에서 지지후보의 득표율 예측정확도를 개선하기 위해서 무응답(없다/무응답)층을 머신러닝으로 예측을 시도해 보았다.

✓ 예측(무응답보정)한 결과를 개표결과/기존 여론조사와 비교해보니, 전체적으로 후보별 득표율이 높아지는 것을 볼 수 있었다.

✓ 하지만 기존에 득표율이 낮은 후보에서는 과잉 보정되는 경향을 보였다. 이점은 다른 사례를 통해서 추가검증이 필요해 보인다.

(SPSS 프로그램으로 보정데이터 적용한 테이블 결과임)

	개표결과	여론조사	오렌지 (무응답보정)	파이썬 (무응답보정)
문재인	41.08	37.**	39.81	39.93
홍준표	24.03	18.**	22.27	21.33
안철수	21.41	17.**	19.70	20.05
유승민	6.76	7.**	7.73	7.72
심상정	6.17	8.**	9.97	10.46
기타		0.**	.52	.52
없음/모름		10.33		

Thank you!

감사합니다.