종목 선정 모델 개발

Contents


```
import FinanceDataReader as fdr
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import warnings
import pickle
warnings.filterwarnings('ignore')
pd.options.display.float_format = '{:,.3f}'.format
```

선형모델에 대한 중요한 가정과 설명은 다음 절에서 추가로 설명드리겠으나, 수익률에 따라 단조 증가나 감소의 형태를 보이지 않는 피쳐(설명변수)는 변형을 해야 선형모형에서 더 유의미하게 사용될 수 있습니다. 주로 Binning (오름차순으로 정렬 후, 여러개 구간으로 분리) 을 통하여 이런 비선형적인 관계를 선형적으로 변경합니다. 2차 함수나 로그함수 등을 이용해 선형적으로 변경할 수 도 있습니다. 우리는 앞서 수익율과 피쳐사이에 선형적인 관계를 가지지 않는 가설(예: 섹터의 평균 수익률 대비 종목 수익률)들 이 있었습니다. 이런 피처들에 대하여 Binning 없이 적합할 수 있는 모델이 일반화가법모형(Generalized Additive Model) 입니다. 또한 가설 검정에서는 5 영업일 동안의 최대 수익률을 예측변수로 이용했으나, 모델의 overfitting (과대적합) 문제를 최소화하기 위하여, 예측값을 이진값(0/1)으로 치환한 후, 로지스틱 일반화가법모형(Logistic Generalized Additive Model) 을 구현합니다. 로지스틱 회귀모형은 $log(odds) = a0 + a1 * x1 + a2 * x2 \ldots$ 으로 표현할 수 있는데요. 여기서 X 를 여러개의 spline 로 함수로 만든 후, 다시 합하여 X 와 $log(odds)$ 의 비선형적관계를 표현할 수 있도록 한 것이 Logistic GAM 입니다. 이 모델의 구현은 Statsmodels 에서 가능합니다만, pyGAM 패키지는 자동으로 하이퍼파라미터를 찾는 기능이 있어 편리합니다. GAM 을 선택한 다른 이유는 피처사이에 상호작용이 크지 않을 것이라는 가정이 있습니다. 무엇보다도 좋은 점은 모델이 왜 이 종목을 선택했는지에 대한 해석이 가능합니다. 향후, 모델의 예측력이 저하되는 경우 어떤 피처가 원인인지도 파악이 가능합니다.

단순히, 스코어가 높은 모든 종목을 매수하는 것이 아니라, 오늘의 종가 수익률과 주가를 고려하여 기본적인 필터링을 합니다. 분석결과 종가 수익률은 높고, 최근 20일 대비 가격이 낮은 종목을 매수하면 리스크가 적은 것으로 판단됩니다.

이번절에서는 책에서 종목선정을 위해 사용할 GAM 모델을 개발하겠습니다. 아나콘다 프롬프트에서 conda install -c conda-forge pygam 로 설치를 해 줍니다. 관련 링크 https://anaconda.org/conda-forge/pygam

모델링을 위해 준비한 데이터를 읽습니다. 그리고 모델의 오버피팅을 최소화하기 위하여 타겟변수를 0 과 1 로 치환합니다. 5% 익절은 다음과 같이 데이터로 표현할 수 있습니다. - 'max_close' 가 5% 이상일 때 1, 아니면 0. 파이썬 코드는 아래와 같습니다.

```
np.where(feature_all['max_close']>= 1.05, 1, 0)
```

타겟 변수 - 'target' 값이 1 인 비율을 보니, 약 24% 입니다. 타겟변수의 비율이 너무 적으면 모델 트레이닝이 어렵습니다.

```
feature_all = pd.read_pickle('feature_all.pkl')
feature_all['target'] = np.where(feature_all['max_close']>= 1.05, 1, 0)
target = feature_all['target'].mean()
print(f'% of target:{target: 5.1%}')
```

```
% of target: 24.3%
```

날짜와 종목은 모델의 입력피처가 아닙니다. 편의를 위해 제거하거나 인덱스로 처리합니다. 모델 트레이닝 용도로 10,000 개 샘플을 뽑아 예측모델을 만들고, 나머지 데이터는 테스트(혹은 백테스팅)를 하겠습니다.

```
mdl_all = feature_all.set_index([feature_all.index,'code'])

train = mdl_all.sample(10000, random_state=124)
test = mdl_all.loc[~mdl_all.index.isin(train.index)]
print(len(train), len(test))
```

```
10000 319307
```

입력 피처의 갯수와 데이터타입을 확인합니다.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 10000 entries, ('2021-10-22', '312610') to ('2021-04-28', '011320')
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   sector              10000 non-null  object
 1   return              10000 non-null  float64
 2   kosdaq_return       10000 non-null  float64
 3   price_z             10000 non-null  float64
 4   volume_z            10000 non-null  float64
 5   num_high/close      10000 non-null  float64
 6   num_long            10000 non-null  float64
 7   num_z>1.96          10000 non-null  float64
 8   num_win_market      10000 non-null  float64
 9   pct_win_market      10000 non-null  float64
 10  return over sector  10000 non-null  float64
 11  max_close           10000 non-null  float64
 12  mean_close          10000 non-null  float64
 13  min_close           10000 non-null  float64
 14  target              10000 non-null  int32
dtypes: float64(13), int32(1), object(1)
memory usage: 1.2+ MB
```

각 변수별로 다른 'lambda' (Wiggliness Penalty Weight) 을 적용해서 grid Search 를 합니다. spline 수는 20 이 default 값입니다. spline 수는 고정하고 lambda의 최적 조합을 찾거나, lambda 를 고정하고, spline 수의 최적 조합을 찾는 것이 현실적이고, 두 하이퍼파라미터를 동시에 조합하여 grid Search 하는 것은 시간이 많이 걸립니다. 다양한 시도를 통하여 더 좋은 모델을 구현할 수 있겠으나, 이 책에서는 grid search 로 변수별 최적의 lambda 를 찾는 것으로 모델을 완성합니다. P value 가 크게 나타나는 입력변수는 제거하는 것이 좋겠습니다.

```
from pygam import LogisticGAM, s, f, te, l
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss

feature_list =
['price_z','volume_z','num_high/close','num_win_market','pct_win_market','return over
sector']
X = train[feature_list]
y = train['target']
X_test = test[feature_list]
y_test = test['target']

# 하이퍼파라미터 설정 N 개의 변수면 (M x N) 개의 리스트로 생성함으로써 변수별로 다른 하이퍼
파라미터 테스트 가능.
# M 개만 1D 리스트를 만들면 동일한 lambda 른 모든 변수에 적용함.
lam_list = [np.logspace(0, 3, 2)]*8

gam = LogisticGAM(te(0, 1, n_splines=5) + s(1) + s(2) + s(3) + s(4) + te(4, 5,
n_splines=5)).gridsearch(X.to_numpy(), y.to_numpy(), lam=lam_list)

print(gam.summary())
print(gam.accuracy(X_test, y_test))
```

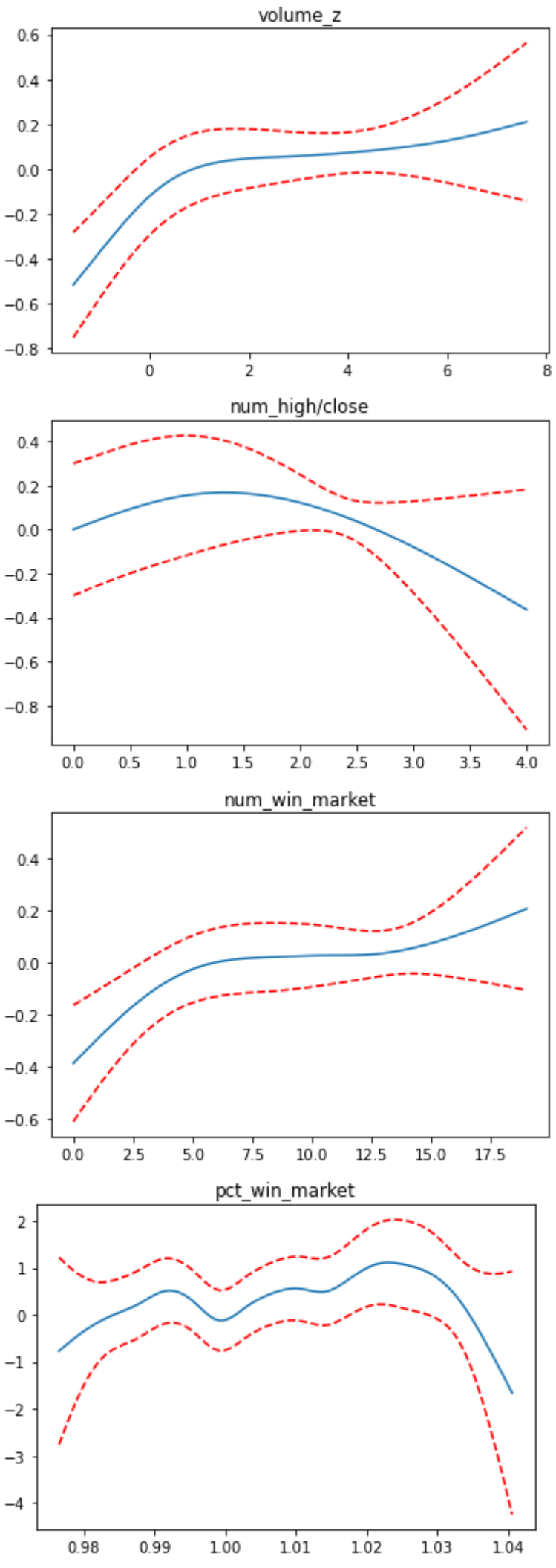
```
100% (256 of 256) |######################| Elapsed Time: 0:03:32 Time:  0:03:32
```

```
LogisticGAM
=============================================
============================================================
Distribution:                      BinomialDist Effective DoF:
21.7127
Link Function:                        LogitLink Log Likelihood:
-5473.5193
Number of Samples:                        10000 AIC:
10990.4639
                                                AICc:
10990.5719
                                                UBRE:
3.1008
                                                Scale:
1.0
                                                Pseudo R-Squared:
0.0197
================================================================================
=====================
Feature Function                  Lambda               Rank         EDoF         P >
x        Sig. Code
================================= ==================== ============ ============
============ ============
te(0, 1)                          [1. 1.]              25           6.2
1.11e-15     ***
s(1)                              [1000.]              20           0.1
1.39e-04     ***
s(2)                              [1000.]              20           1.8
1.65e-01
s(3)                              [1000.]              20           2.7
4.35e-02     *
s(4)                              [1.]                 20           8.9
1.11e-11     ***
te(4, 5)                          [1. 1.]              25           2.0
1.80e-01
intercept                                              1            0.0
2.07e-01
================================================================================
=====================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model
identifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models
or models with
         known smoothing parameters, but when smoothing parameters have been
estimated, the p-values
         are typically lower than they should be, meaning that the tests reject the
null too readily.
None
0.7571130530379218
```

```
for i, term in enumerate(gam.terms):
    print(i, term)
```

```
0 tensor_term
1 spline_term
2 spline_term
3 spline_term
4 spline_term
5 tensor_term
6 intercept_term
```

```
feature_list =
['price_z','volume_z','num_high/close','num_win_market','pct_win_market','return over
sector']
for i, term in enumerate(gam.terms):

    if i>=1 and i<=4:

        XX = gam.generate_X_grid(term=i)
        pdep, confi = gam.partial_dependence(term=i, X=XX, width=0.95)

        plt.figure()
        plt.plot(XX[:, term.feature], pdep)
        plt.plot(XX[:, term.feature], confi, c='r', ls='--')
        plt.title(feature_list[i])
        plt.show()
```



완성된 모델을 pickle 로 binary 파일로 저장합니다.

```
import pickle
with open("gam.pkl", "wb") as file:
    pickle.dump(gam, file)
```

```
with open("gam.pkl", "rb") as file:
    gam = pickle.load(file)
```

```
print(gam.get_params())
print(gam.coef_.shape)
```

```
{'max_iter': 100, 'tol': 0.0001, 'callbacks': [Deviance(), Diffs(), Accuracy()],
'verbose': False, 'terms': te(0, 1) + s(1) + s(2) + s(3) + s(4) + te(4, 5) +
intercept, 'fit_intercept': True}
(131,)
```

```
for i in range(6):
    print(f'{i}: {gam._compute_p_value(i): 5.3f}
{gam.generate_X_grid(term=i).shape}')
```

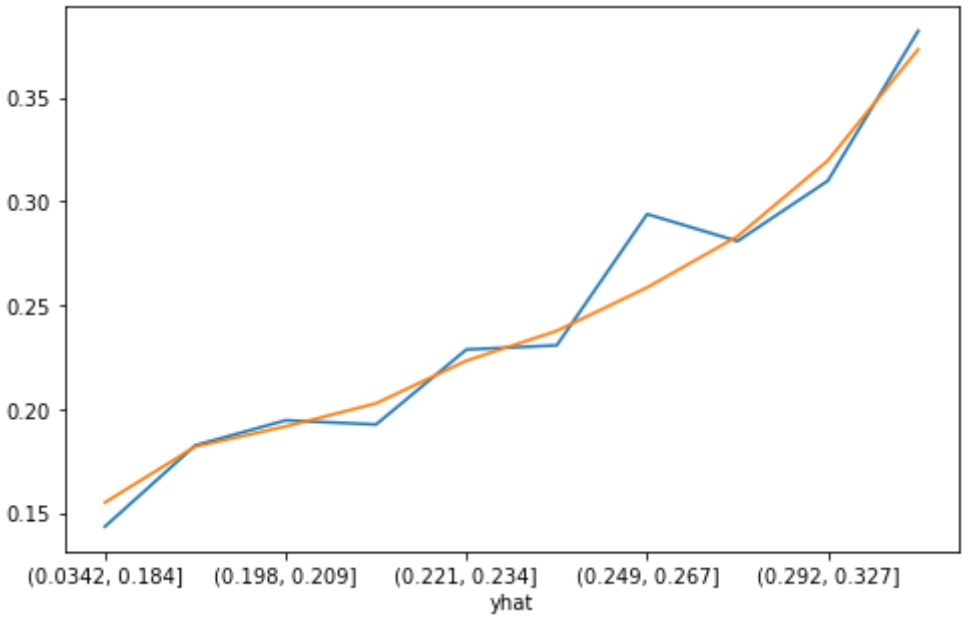
```
0:  0.000 (10000, 6)
1:  0.000 (100, 6)
2:  0.165 (100, 6)
3:  0.043 (100, 6)
4:  0.000 (100, 6)
5:  0.180 (10000, 6)
```

간단하게 십분위수 분석을 하고, 성능을 평가합니다. 안정적인 모델을 만들었습니다. 이론적으로는 마지막 Decile(제 10 십분위 수)에서 랜덤하게 종목을 골라 동일한 금액으로 매수를 한다면, 5 영업일이내 5% 익절할 확률이 37% 가 됩니다. 100% 만족스럽지는 않지만, 생성된 GAM 모델을 이용하여 종목 추천을 받도록 하겠습니다.

```
feature_list =
['price_z','volume_z','num_high/close','num_win_market','pct_win_market','return over
sector']
X = train[feature_list]
y = train['target']
X_test = test[feature_list]
y_test = test['target']

yhat = gam.predict_proba(X.to_numpy())
yhat = pd.Series(yhat, name='yhat', index=y.index)

yhat_test = gam.predict_proba(X_test.to_numpy())
yhat_test = pd.Series(yhat_test, name='yhat', index=y_test.index)
```

```
def perf(y, yhat): # Decile 분석 함수
    combined = pd.concat([y, yhat], axis=1)
    ranks = pd.qcut(combined['yhat'], q=10)
    print(combined.groupby(ranks)['target'].agg(['count','mean']))
    combined.groupby(ranks)['target'].mean().plot(figsize=(8,5))

perf(y, yhat)
perf(y_test, yhat_test)
```

```
               count  mean
yhat
(0.121, 0.184]   1000 0.144
(0.184, 0.198]   1000 0.183
(0.198, 0.21]    1000 0.195
(0.21, 0.222]    1000 0.193
(0.222, 0.235]   1000 0.229
(0.235, 0.251]   1000 0.231
(0.251, 0.268]   1000 0.294
(0.268, 0.291]   1000 0.281
(0.291, 0.326]   1000 0.310
(0.326, 0.688]   1000 0.382
                count  mean
yhat
(0.0342, 0.184]  31931 0.156
(0.184, 0.198]   31931 0.182
(0.198, 0.209]   31930 0.192
(0.209, 0.221]   31931 0.203
(0.221, 0.234]   31931 0.224
(0.234, 0.249]   31930 0.238
(0.249, 0.267]   31931 0.259
(0.267, 0.292]   31930 0.283
(0.292, 0.327]   31931 0.320
(0.327, 0.783]   31931 0.373
```



단순히 스코어가 높다고 무조건 매수했다가 큰 낙폭으로 손해를 볼 수도 있기 때문에 기본적인 필터링이 필요합니다. 오늘 종가 수익률과 가격 변동성으로 기본적인 필터를 만들어 보겠습니다.

```
test['yhat'] = yhat_test
test['yhat_rank'] = pd.qcut(test['yhat'], q=10)
test.groupby('yhat_rank')['target'].mean()
```

```
yhat_rank
(0.0342, 0.184]   0.156
(0.184, 0.198]    0.182
(0.198, 0.209]    0.192
(0.209, 0.221]    0.203
(0.221, 0.234]    0.224
(0.234, 0.249]    0.238
(0.249, 0.267]    0.259
(0.267, 0.292]    0.283
(0.292, 0.327]    0.320
(0.327, 0.783]    0.373
Name: target, dtype: float64
```

종목선정은 상위 스코어 구간에서 할 것이므로 상위 구간에서 대하여 수익률 및 표준화 가격 구간으로 분리해서 미래 수익률을 보겠습니다. 표준화된 가격이 낮고 당일 수익율이 높은 경우 미래 수익률이 높을 것으로 예상됩니다.

```
tops = test[test['yhat'] > 0.3].copy()

tops['return_rank']  = pd.qcut(tops['return'], q=5) # 종가 수익률
tops['price_rank']  = pd.qcut(tops['price_z'], q=5) # 가격 변동성
tops.groupby(['return_rank','price_rank'])
['target'].mean().unstack().style.set_table_attributes('style="font-size: 12px"')
```

price_rank return_rank	(-4.36, -1.836]	(-1.836, -0.962]	(-0.962, 0.529]	(0.529, 1.659]	(1.659, 4.359]
(0.325, 0.958]	0.467728	0.401386	0.365480	0.402558	0.352185
(0.958, 0.98]	0.323312	0.322218	0.312813	0.371080	0.380762
(0.98, 1.0]	0.247881	0.301568	0.303585	0.335506	0.329555
(1.0, 1.031]	0.385084	0.341678	0.318606	0.339809	0.330736
(1.031, 1.3]	0.670000	0.471311	0.335314	0.379326	0.368984

참고로 groupby 로 데이터를 요약하는 방법은 직관적이나, 각 행과 열의 총계는 보여주지 않는다는 단점이 있습니다. 총계가 보고 싶을 때는 pivot_table 에서 'margins=True' 를 인수로 넣어주면 총계를 볼 수 있습니다.

```
pd.pivot_table(data = tops, index = 'return_rank', columns = 'price_rank', values = 'target', aggfunc='mean', margins=True).style.set_table_attributes('style="font-size: 12px"')
```

price_rank return_rank	(-4.36, -1.836]	(-1.836, -0.962]	(-0.962, 0.529]	(0.529, 1.659]	(1.659, 4.359]	All
(0.325, 0.958]	0.467728	0.401386	0.365480	0.402558	0.352185	0.419958
(0.958, 0.98]	0.323312	0.322218	0.312813	0.371080	0.380762	0.331053
(0.98, 1.0]	0.247881	0.301568	0.303585	0.335506	0.329555	0.302817
(1.0, 1.031]	0.385084	0.341678	0.318606	0.339809	0.330736	0.335920
(1.031, 1.3]	0.670000	0.471311	0.335314	0.379326	0.368984	0.376038
All	0.376790	0.343522	0.325730	0.363139	0.355839	0.353005

최저 수익률(리스크)도 조사합니다. 당일 수익률 높고, 표준화 된 주가가 낮은 좌하단 부분의 리스크가 낮습니다.

```
pd.pivot_table(data = tops, index = 'return_rank', columns = 'price_rank', values = 'min_close', aggfunc='mean', margins=True).style.set_table_attributes('style="font-size: 12px"')
```

price_rank return_rank	(-4.36, -1.836]	(-1.836, -0.962]	(-0.962, 0.529]	(0.529, 1.659]	(1.659, 4.359]	All
(0.325, 0.958]	0.971249	0.955960	0.950540	0.949295	0.936891	0.959307
(0.958, 0.98]	0.968130	0.961449	0.955620	0.955920	0.948468	0.961134
(0.98, 1.0]	0.963653	0.965476	0.958850	0.955340	0.945676	0.959550
(1.0, 1.031]	0.971163	0.965972	0.958342	0.953141	0.952475	0.958020
(1.031, 1.3]	0.992942	0.970208	0.950858	0.947691	0.948263	0.950377
All	0.969054	0.963253	0.955302	0.952232	0.948584	0.957685

위 결과를 종합하면 당일 종가 수익률은 높고, 최근 20일 대비 가격이 낮은 종목을 매수하면 리스크가 적을 것으로 판단됩니다. 'return' 은 1.03 보다 크고, 'price_z' 는 0 보다 작은 종목만을 고르겠습니다.

```
tops[ (tops['return'] > 1.03) & (tops['price_z'] < 0)][['return','price_z']].head().style.set_table_attributes('style="font-size: 12px"')
```

	code	return	price_z
2021-05-26	**037440**	1.038084	-0.747394
2021-06-24	**037440**	1.050481	-0.605273
2022-03-10	**037440**	1.147170	-0.495290
2021-04-19	**189980**	1.031782	-1.108919
2021-10-07	**189980**	1.073741	-1.231395

By KHS