빅데이터 팀 프로젝트

팀 1조
컴퓨터전공 박진서
컴퓨터전공 심지섭

목차

목표

주식파일(stock_history.csv)을 읽어들여 독립변수로 고려되어질 데이터 값을 만들어 stock_history_add.csv 파일에 저장 후 데이터 값들과 외부 데이터 값을 이용한 다중회귀분석 모델 식을 구하여, 목표에 해당하는 cvNd_diff_rate 근사한 값을 도출한다. 또한 다중회귀모델을 통한 alpha, beta와 r-squared값을 찾는다.

1.stock_history.add 파일을 만들기 위한 코드 설명.

cv_diff_value(종가 일간 변화량)

```
def make_cv_diff_value(Dataframe):

    i=Dataframe.index[-1]
    k = Dataframe.index[0]
    while i > k:
        Dataframe.loc[i-1, "cv_diff_value"] = (Dataframe.loc[i-1, "close_value"] - Dataframe.loc[i, "close_value"])
        i -= 1

    return Dataframe["cv_diff_value"]
```

cv_diff_value (전일비) parameter값인 Dataframe을 읽어서, 각 종목마다
현재날짜에서 전일 날짜를 뺀 값을 저장 후 반환 한다.
주어진 파일의 날짜가 역순으로 저장되어 있기에 index또한 역순으로 저장함.

1.stock_history.add 파일을 만들기 위한 코드 설명.

cv_diff_rate(종가 일간 변화율)

```
def make_cv_diff_rate(Dataframe):

    close_value_Series = Dataframe["close_value"]
    cv_diff_rate_Series = Dataframe["cv_diff_rate"]
    last_index = close_value_Series.index[-1]

    for index in Dataframe.index:
        if index == last_index: break
        cv_diff_rate_Series.at[index] = close_value_Series.at[index] / close_value_Series.at[index + 1] * 100 - 100
    return cv_diff_rate_Series
```

전일 대비 종가의 변화율를 계산하는 함수. Dataframe을 읽어서 cv_diff_rate 컬럼값에 저장 후 Series형으로 반환한다.

1.stock_history.add 파일을 만들기 위한 코드 설명.

cv_maN_value(종가의 N일 이동평균)

```
def make_cv_maN_value(Dataframe,n=5): #parameter filename=".csv file", n=number,default=5

    if len(Dataframe) < n: return
    #주가이동평균선은 시장의 전반적인 주가 흐름을 판단하고 향후 주가의 추이를 전망하는 데 자주 사용되는 주식시장의 대표적인 기술지표
    g_close=Dataframe["close_value"]
    Dataframe["cv_ma" + str(n) +"_value"] = g_close.rolling(window=n).mean().shift(-(n - 1))  # Series type
    return Dataframe["cv_ma" + str(n) +"_value"]
```

N일동안의 종가들의 합을 평균을 낸 값. 컬럼값이 n보다 작을 시 return하고, 받아진 n값에 따라 rolling함수를 이용하여 n일동안의 종가 평균을 계산 후 반환한다.

1.stock_history.add 파일을 만들기 위한 코드 설명.

cv_maN_rate(종가의 N일 이동평균의 일간 변화율)

```
def make_cv_maN_rate(Dataframe,n=5):
    if len(Dataframe) < n: return
    g_close = Dataframe["close_value"]
    cv_ma_value = g_close.rolling(window=n).mean().shift(-(n - 1))  # Series type

    i=Dataframe.index[-n]
    k = Dataframe.index[0]

    while i > k:
            Dataframe.loc[i-1,"cv_ma"+ str(n) +"_rate"] = (cv_ma_value[i-1]-cv_ma_value[i])/cv_ma_value[i]*100 #rolling 함수 쓰기.
            i -= 1

    return Dataframe["cv_ma"+ str(n) +"_rate"]
```

종가의 n일 이동평균의 전일 대비율을 계산 하는 함수. 종가의 n일 이동평균을 rolling함수로 저장 후, 전일 대비율을 계산하여 반환한다.

1.stock_history.add 파일을 만들기 위한 코드 설명.

ud_Nd(종가 상승률의 대한 지표)

```
def make_ud_Nd(Dataframe,N=5):
    if len(Dataframe) < N: return
    close_value_Series = Dataframe["close_value"]
    ud_Nd_Series = Dataframe["ud_" + str(N) + "d"]
    last_index = close_value_Series.index[-N]

    for index in Dataframe.index:
        if index == last_index: break
        val = 0
        for i in range(0, N, 1):
            if close_value_Series.at[index + i] - close_value_Series.at[index + i + 1] > 0:
                val += 1
            else:
                val -= 1

        if val % N == 0:
            val = val / N
        else:
            val = 0
        ud_Nd_Series.at[index + 1] = val
    return ud_Nd_Series
```

1.stock_history.add 파일을 만들기 위한 코드 설명.

ud_Nd(종가 상승률의 대한 지표)

N일동안의 종가 비율을 계산 하여 N일동안 연속 오를시 1을 return, N일동안 연속 하락 시 -1을 return, 그렇지 않으면 0을 return한다. N일동안 종가가 증가했는지 아닌지를 파악 한 후, 기준에 맞는 값을 ud_Nd Series로 만들어 반환한다.

1.stock_history.add 파일을 만들기 위한 코드 설명.

cv_Nd_diff_rate(N일간의 종가 상승률 설정)

```
def make_cvNd_diff_rate(Dataframe, N=5):
    if len(Dataframe) < N: return
    close_value_Series = Dataframe["close_value"]
    cvNd_diff_rate_Series = Dataframe["cv" + str(N) + "d_diff_rate"]
    last_index = close_value_Series.index[-N]

    for index in Dataframe.index:
        if index == last_index: break
        cvNd_diff_rate_Series.at[index + 1] = (close_value_Series.at[index] - close_value_Series.at[index + N]) / close_value_Series.at[index + N]
    return cvNd_diff_rate_Series
```

N일전과 현 종가의 차이를 비율로 나타내어 N-1일의 컬럼값에 지정하여 반환하는 함수. 후에 다중회귀분석 모델링을 통해 비교할 y값이기도 하다.

1.stock_history.add 파일을 만들기 위한 코드 설명.

main문

```
if __name__ == "__main__":
    # 추가할 컬럼스를 리스트화 시킴
    add_columns=["basic_date", "stockname", "stock_code", "open_value","high_value", "low_value", "close_value", "volume_value",
                 "cv_diff_value", "cv_diff_rate", "cv_ma3_value","cv_ma4_value", "cv_ma5_value", "cv_ma3_rate", "cv_ma4_rate", "cv_ma5_rate",
                 "ud_3d", "ud_4d", "ud_5d", "cv3d_diff_rate", "cv4d_diff_rate", "cv5d_diff_rate"]

    df = pd.read_csv("stock_history.csv", skiprows=[0], names=add_columns, encoding='euc-kr') #추가할 컬럼을 넣어 read file.

    # csv파일을 읽어 컬럼 리스트 수정.

    # 함수가 돌아갈때마다 생기는 각 그룹의 결과값을 저장하기 위한 리스트 모음
    lst_cv_diff_value=[]
    lst_cv_diff_rate=[]
    lst_cv_ma3_value, lst_cv_ma4_value, lst_cv_ma5_value=[], [], []
    lst_cv_ma3_rate, lst_cv_ma4_rate, lst_cv_ma5_rate=[], [], []
    lst_ud_3d, lst_ud_4d, lst_ud_5d=[], [], []
    lst_cv3d_diff_rate, lst_cv4d_diff_rate, lst_cv5d_diff_rate=[], [], []
```

stock_history를 읽고 stock_history_add를 위해 만들 컬럼값들을 리스트화 시켜 읽음과 동시에 0번째 인덱스에 수정을한다. 그 이후 함수들로 인해 반환 될 값들을 받을 리스트를 만든다.

1.stock_history.add 파일을 만들기 위한 코드 설명.

main문

```
for name, group in df.groupby("stockname"):
    group = group.copy() #SettingWithCopyWarning 방지


    #리스트에 각 그룹마다의 결과값을 append함수로 넣기.

    lst_cv_diff_value.append(make_cv_diff_value(group))

    lst_cv_diff_rate.append(make_cv_diff_rate(group))

    lst_cv_ma3_value.append(make_cv_maN_value(group, 3))
    lst_cv_ma4_value.append(make_cv_maN_value(group, 4))
    lst_cv_ma5_value.append(make_cv_maN_value(group))

    lst_cv_ma3_rate.append(make_cv_maN_rate(group, 3))
    lst_cv_ma4_rate.append(make_cv_maN_rate(group, 4))
    lst_cv_ma5_rate.append(make_cv_maN_rate(group))

    lst_ud_3d.append(make_ud_Nd(group, 3))
    lst_ud_4d.append(make_ud_Nd(group, 4))
    lst_ud_5d.append(make_ud_Nd(group))

    lst_cv3d_diff_rate.append(make_cvNd_diff_rate(group,3))
    lst_cv4d_diff_rate.append(make_cvNd_diff_rate(group,4))
    lst_cv5d_diff_rate.append(make_cvNd_diff_rate(group))
```

```
#Dataframe 합치기, concat 이용
g_cv_diff_value = pd.concat(lst_cv_diff_value)

g_cv_diff_rate = pd.concat(lst_cv_diff_rate)

g_cv_ma3_value = pd.concat(lst_cv_ma3_value)
g_cv_ma4_value = pd.concat(lst_cv_ma4_value)
g_cv_ma5_value = pd.concat(lst_cv_ma5_value)

g_cv_ma3_rate=pd.concat(lst_cv_ma3_rate)
g_cv_ma4_rate = pd.concat(lst_cv_ma4_rate)
g_cv_ma5_rate = pd.concat(lst_cv_ma5_rate)

g_ud_3d=pd.concat(lst_ud_3d)
g_ud_4d = pd.concat(lst_ud_4d)
g_ud_5d = pd.concat(lst_ud_5d)

g_cv3d_diff_rate=pd.concat(lst_cv3d_diff_rate)
g_cv4d_diff_rate = pd.concat(lst_cv4d_diff_rate)
g_cv5d_diff_rate = pd.concat(lst_cv5d_diff_rate)
```

1.stock_history.add 파일을 만들기 위한 코드 설명.

main문

```
#새로운 stock history add를 만들기 위해 컬럼 값 넣어주기.
df["cv_diff_value"] = g_cv_diff_value

df["cv_diff_rate"] = g_cv_diff_rate

df["cv_ma3_value"] = g_cv_ma3_value
df["cv_ma4_value"] = g_cv_ma4_value
df["cv_ma5_value"] = g_cv_ma5_value

df["cv_ma3_rate"] = g_cv_ma3_rate
df["cv_ma4_rate"] = g_cv_ma4_rate
df["cv_ma5_rate"] = g_cv_ma5_rate

df["ud_3d"] = g_ud_3d
df["ud_4d"] = g_ud_4d
df["ud_5d"] = g_ud_5d

df["cv3d_diff_rate"] = g_cv3d_diff_rate
df["cv4d_diff_rate"] = g_cv4d_diff_rate
df["cv5d_diff_rate"] = g_cv5d_diff_rate

#새로운 add.csv파일을 만들어 저장.
df.to_csv("stock_history_add.csv", index=False, encoding="ms949")
```

for문을 통해 stockname에 의해 분류 된 주식 종목들에 함수를 적용하여 나온 값들을 초기에 만든 각 리스트에 append시킨 후, 각 리스트를 concat을 통해 Series화 시킨다.

각 Series화 된 값들을 그에 상응하는 데이터프레임 컬럼값에 저장 후, 새롭게 생긴 컬럼값을 추가 후, stock_history_add.csv파일을 만들어서 저장한다.

2.종목 선정

기준 설정:

*주식의 거래날짜 수: 주식의 거래날짜 수가 적을 시 모델링에 필요한train set과 테스트할 test set을 만들기 부적절함. 따라서 230일의 거래를 가진 주식을 선정.

첫번째 기준(수익성):

-1*(종가일간변화율(cv_diff_rate)의 양수값의 총합 / 종가일간변화율(cv_diff_rate)의 음수값의 총합) 이 큰 순서로 TOP5를 선정, 하락보다 상승이 높은 종목을 선택함으로서 수익성이 높은 종목을 선택할 수 있음.

위 TOP5를 종가일간변화율의변화율(cv_diff_rate_rate)의 절대값의 합이 작은 순서대로 재배열 하여 TOP5중 안정성이 높은 종목을 선정. (cv_diff_rate_rate가 작다는것은 변화율의 변동의폭이 작은 (계속 꾸준히 올라가는) 것을 의미한다.)

2.종목 선정

```
def make_cv_diff_rate_rate(Dataframe):
    cv_diff_rate_Series = Dataframe["cv_diff_rate"]
    cv_diff_rate_rate_Series = Dataframe["cv_diff_rate_rate"]
    last_index = cv_diff_rate_Series.index[-1]

    for index in Dataframe.index:
        if index == last_index: break
        cv_diff_rate_rate_Series.at[index] = cv_diff_rate_Series.at[index] - cv_diff_rate_Series.at[index + 1]
    return cv_diff_rate_rate_Series
```

주어진 dataframe을 이용하여 종가의 변화율의변화율(cv_diff_rate_rate)을 계산하여 Series로 리턴해주는 함수

2.종목 선정

```
def sum_cv_diff_rate_rate(Dataframe):
    cv_diff_rate_rate_Series = Dataframe["cv_diff_rate_rate"]
    SUM = abs(cv_diff_rate_rate_Series).sum()
    return SUM
```

주어진 dataframe을 이용하여 종가의 변화율의변화율(cv_diff_rate_rate)의 절대값의 합을 리턴해주는 함수

```
def cv_diff_rate_compare(Dataframe):
    cv_diff_rate_Series = Dataframe["cv_diff_rate"]
    PLUS = cv_diff_rate_Series[cv_diff_rate_Series > 0].sum()
    MINUS = cv_diff_rate_Series[cv_diff_rate_Series < 0].sum()
    return [PLUS/ -MINUS, PLUS, MINUS]
```

주어진 dataframe을 이용하여 종가의 변화율(cv_diff_rate)의 -1*양수/음수, 양수의총합, 음수의 총합을 리턴해주는 함수

2.종목 선정

```
['STX', [13.42, 955.09, -71.17], 1981.24, 230]
['천보', [8.9, 15.93, -1.79], 13.76, 8]
['STX중공업', [6.0, 2080.76, -346.97], 4532.59, 230]
['RFHIC', [5.18, 714.84, -138.01], 1544.02, 230]
['대유', [3.84, 52.71, -13.71], 78.08, 7]
```

왼쪽부터

종목명, 종가의변화율 -1*양수/음수, 양수의 총합, 음수의 총합, 변화율의변화율의 절대값의 총합, 종목의 row개수이다.

종목의 row의 수가 적은 '천보' 와 '대유' 제외
나머지 종목들의 종가의 변화율의변화율(cv_diff_rate_rate)이 너무높아 학습에 적합하지 않은 종목들이라 판단.
수익율이 높은 종목들은 사용하지 않기로 하였다.

2.종목 선정

기준 설정:

*주식의 거래날짜 수: 주식의 거래날짜 수가 적을 시 모델링에 필요한train set과 테스트할 test set을 만들기 부적절함. 따라서 230일의 거래를 가진 주식을 선정.

두번째 기준(안정성): 주식의 종가 일간변화율(cv_diff_rate)의 절대값의 합이 낮은 주식이 예측하기 쉬운 모델이라고 생각. 또한 예측값과 실제값의 차이가 적을것이기 때문에 결정계수값도 높을것이라고 생각. 수익성을 고려하지 않고 안정성만을 고려하여 TOP5를 선정하기로 하였다.

2.종목 선정

기준 설정:

2. 주어진 주식들 중 변화율의 총합이 낮은 모델들을 선정, 그 이후 다음 기준을 선정.

3.특정 주식 제외: 기준2에 따른 종목을 선정 시, 변화율이 존재하지 않는 주식을 파악. 그러한 주식들은 코넥스에 상장. 코넥스 주식 특성 상 기관투자자나 자산몇억이상개인투자만 가능한 점을 고려. 실질적인 다중회귀모델 학습에 좋은 지표가 아니기에 배제함.

2.종목 선정

sum_of_abs_cv_diff_rate

```
def sum_of_abs_cv_diff_rate(Dataframe):

    cv_rate_series=abs(Dataframe["cv_diff_rate"])
    return cv_rate_series.sum()
```

cv_diff_rate컬럼값을 절대값으로 변환 후, cv_diff_rate의 총합을 return해주는 함수.

2.종목 선정

find_stable_rate_stocks

```
def find_stable_rate_stocks(stocks, n=15):

    n_stocks = dict()
    while n > 0:
        stock_name = min(stocks, key=stocks.get)
        n_stocks[stock_name]=stocks[stock_name]
        del stocks[stock_name]
        n -= 1
    return n_stocks
```

각 주식이름이 key이고, 변화율의 합이 value인 stocks(type=dict())를 받아 n만큼 주식들의 minimum값 부터 차례대로 저장하여 dict을 반환 한다.

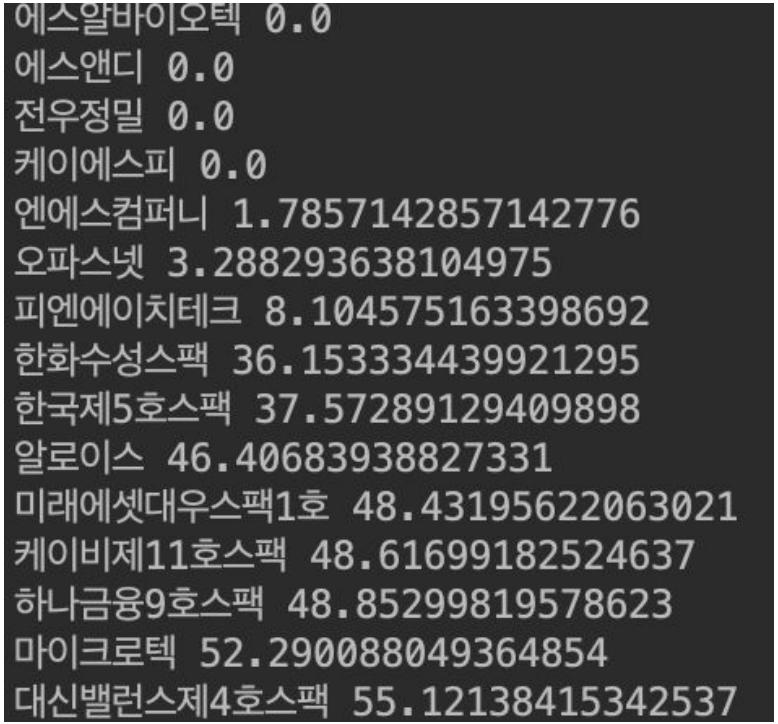
2.종목 선정

main문

```
if __name__ == "__main__":
    df = pd.read_csv("stock_history_add.csv", encoding="euc-kr")

    stock_dict=dict() #key:주식이름, value=sum(abs())

    for name, group in df.groupby("stockname"):

        group = group.copy() #SettingWithCopyWarning 방지
        if len(group.index) > 100:
            stock_dict[name]=sum_of_abs_cv_diff_rate(group)

    stocks=find_stable_rate_stocks(stock_dict)
    for key in stocks:
        print(key, stocks[key])
```

Stock_history_add 파일을 읽어 stockname의해 그룹화를 하여 각 그룹마다 변화율의 합이 가장 작은 순으로 저장하는 stocks 딕셔너리를 생성. 그 이후 print문으로 확인하여 종목 선정에 기준이 됨.

2.종목 선정



실행 화면.

이후 종목선정3에 의거하여 코넥스 종목을 제외.

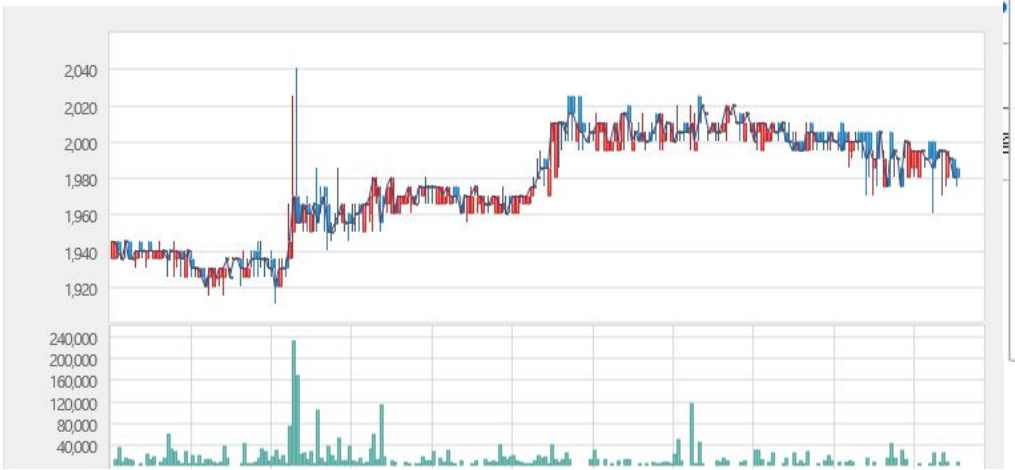
3.데이터 모델 설명

1. 선정한 주식 : 마이크로텍.

마이크로텍[A227950] 코스닥증권시장 / 특수 목적용 기계 제조업

2019/05/27 PM 04:38:05 (20분 지연 정보)

현재가	전일비	등락률(%)	시가	고가	저가	거래량(주)	거래대금(원)
1,575	▲ 10	0.64	1,620	1,620	1,545	55,433	86,690,815



기업개요

반도체, FPD(Flat Panel Display, LCD/OLED) 공정 장비 부품인 진공 Chamber와 해당 Chamber에 사용되는 특수 진공 밸브 제조를 주요사업으로 영위.

주요 매출처는 삼성전자, SK하이닉스 등 반도체 및 디스플레이 산업을 선도하고 있는 국내외 기업들로 동종업계 최대 규모의 파트너쉽 관계를 보유.

연구개발을 통해 20건 이상의 국내외 특허권 및 실용실안을 확보한 상태, 동종업계 1위의 진공기술을 보유.

출처 : 에프앤가이드

3. 데이터 모델 설명

1. 선정한 주식의 cvNd_diff_rate(y)를 계산 하기위한 독립변수들:

   기존 stock_history_add.csv의 변수들:

   -close_value, cv_diff_rate, cv_maN_value(N=3,4,5), cv_maN_rate(N=3,4,5), ud_Nd(N=3,4,5)

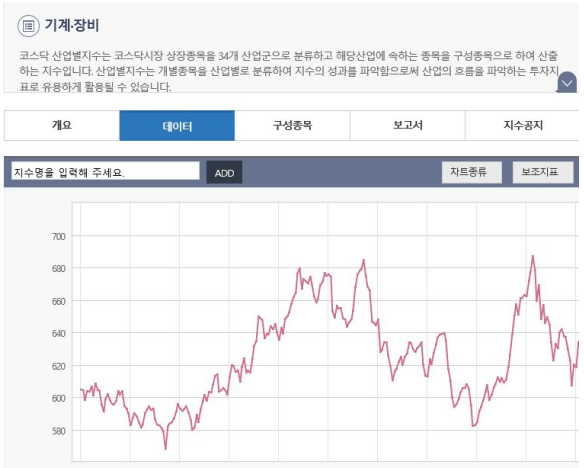
   외부 변수들:

   -cv_diff_rate_of_rate(등락률의 가속도), skhynix_cv_diff_value(sk하이닉스 종가변화량), samsung_cv_diff_value(삼성의 종가변화량), kosdaq_diff_rate(코스닥의 종가변화율)

3.데이터 모델 설명

외부변수 선택기준.

기업개요의 주요매출처인 삼성전자와 sk하이닉스 종목의 정보와 마이크로텍이 속해있는 기계, 장비 산업의 코스닥지수를 외부 독립변수로 선정



3.데이터 모델 설명

1. 선정한 주식의 cvNd_diff_rate(y)를 계산하기위한 독립변수들 중의 선택 변수:

   cv_diff_rate, cv_diff_rate_of_rate, ud_3d, skhynix_cv_diff_value

   cv_diff_rate : 앞선 종목 선정의 큰 기준점이으로 변수 설정

   cv_diff_rate_of_rate : 변화율의 전일대비 변화율을 계산하여 변화율의 가속을 고려.

   ud_3d: N일간의 상승,하강 연속성을 알 수 있는 지표로 모델링에 적합하다 생각.N이 작을때 예측 가능성이 높기에 ud_3d로 선택

   Skhynix_cv_diff_value: 마이크로텍의 주요 매출처인 sk하이닉스의 외부변수 종가변화량을 변수값으로 고려.

4.프로그램 설명(다중회귀)

```
df = pd.read_csv("microtech.csv", encoding="euc-kr")  # 기존 파일의 cvs파일을 읽어드리기.

result = df["cv3d_diff_rate"].tolist()

close_value = df["close_value"].tolist()
cv_diff_value = df["cv_diff_value"].tolist()
cv_diff_rate = df["cv_diff_rate"].tolist()
cv_ma3_value = df["cv_ma3_value"].tolist()
cv_ma4_value = df["cv_ma4_value"].tolist()
cv_ma5_value = df["cv_ma5_value"].tolist()
cv_ma3_rate = df["cv_ma3_rate"].tolist()
cv_ma4_rate = df["cv_ma4_rate"].tolist()
cv_ma5_rate = df["cv_ma5_rate"].tolist()
cv_diff_rate_rate = df["cv_diff_rate_rate"].tolist()
ud_3d = df["ud_3d"].tolist()
ud_4d = df["ud_4d"].tolist()
ud_5d = df["ud_5d"].tolist()
Skhynix_cv_diff_value = df["Skhynix_cv_diff_value"].tolist()
samsung_cv_diff_value = df["samsung_cv_diff_value"].tolist()
kosdaq_diff_rate = df["kosdaq_diff_rate"].tolist()
```

stock_history_add.csv파일의 컬럼값과 외부에서 가져온 외부주가종목, 코스닥의 비율을 리스트로 변환하여 저장.

4.프로그램 설명(다중회귀)

```
x = np.array([np.ones(230), cv_diff_rate, cv_diff_rate_rate, ud_3d, scale(Skhynix_cv_diff_value)])
x = x.T
x = x.astype(np.single)

training_x = x[:140]
test_x = x[140:]

training_lst_cv3d_diff_rate = result[:140]
test_lst_cv3d_diff_rate = result[140:]

training_y = training_lst_cv3d_diff_rate
test_y = test_lst_cv3d_diff_rate
```

기준에 선정 된 독립변수들의 각 리스트를 알파(초기값=1)과 함께 리스트에 넣어 transpose하여 데이터 셋 준비.

그 리스트들은 train set, test set(6:4)로 나누어 저장함.

4.프로그램 설명(다중회귀)

```
print("modeling")
for alpha in [0.0, 0.01, 0.1, 0.15, 0.5, 1]:
    ridge_reg = Ridge(alpha, fit_intercept=False, solver="cholesky")
    # ridge_reg.fit(x, daily_minutes_good)
    ridge_reg.fit(training_x, training_y)
    beta = ridge_reg.coef_
    print("alpha", alpha)
    print("beta", beta)
    print("r-squared", multiple_r_squared(test_x, test_y, beta))
```

Train set으로 인한 Alpha값과 Beta 값을 찾고 생성된 모델링으로 test_set에 적용.

각 Alpha값과 Beta값, 그리고 생성된 모델링의 test_set을 적용하였을 때에 나오는 r-squared값을 print()구문으로 확인.

5.프로그램 실행결과 캡쳐화면

```
modeling
alpha 0.0
beta [ 0.0008371   0.01205555 -0.00632259  0.00277873 -0.0001394 ]
r-squared 0.519379792566398
alpha 0.01
beta [ 0.00083788  0.01202672 -0.00630652  0.00278119 -0.0001395 ]
r-squared 0.5194354335339935
alpha 0.1
beta [ 0.00084463  0.0117737  -0.00616553  0.00280245 -0.00014044]
r-squared 0.5197534835078773
alpha 0.15
beta [ 0.00084818  0.01163788 -0.00608985  0.00281361 -0.00014096]
r-squared 0.519797896942899
alpha 0.5
beta [ 0.00086951  0.01077196 -0.00560754  0.00288015 -0.00014435]
r-squared 0.5180013395302889
alpha 1
beta [ 0.00089138  0.00974552 -0.00503627  0.00294692 -0.0001487 ]
r-squared 0.5111909858649889
```

r-squared의 결과 값이 Alpha값 0.15일 때 최대치인 0.519797896942899 값으로 kmo적합도에 따라 바람직하지 못한 편에 속하나, 주식이란 종목 특성 상 무수한 독립변수들을 고려해야 하기에 예측하기가 쉽지 않으며 kmo<0.5(unacceptable)에는 속하지 않으므로 최선의 선택이라고 판단