Machine Learning HW3

January 26, 2024

[1]:
```
import pandas as pd
```

1 1.) Clean the Apple Data to get a quarterly series of EPS.

[4]:
```
y = pd.read_csv("AAPL_quarterly_financials.csv")
```

[4]:

	name	ttm	09/30/2023	06/30/2023	\
0	TotalRevenue	383,285,000,000	89,498,000,000	81,797,000,000	
1	\tOperatingRevenue	383,285,000,000	89,498,000,000	81,797,000,000	
2	CostOfRevenue	214,137,000,000	49,071,000,000	45,384,000,000	
3	GrossProfit	169,148,000,000	40,427,000,000	36,413,000,000	
4	OperatingExpense	54,847,000,000	13,458,000,000	13,415,000,000	

	03/31/2023	12/31/2022	09/30/2022	06/30/2022	\
0	94,836,000,000	117,154,000,000	90,146,000,000	82,959,000,000	
1	94,836,000,000	117,154,000,000	90,146,000,000	82,959,000,000	
2	52,860,000,000	66,822,000,000	52,051,000,000	47,074,000,000	
3	41,976,000,000	50,332,000,000	38,095,000,000	35,885,000,000	
4	13,658,000,000	14,316,000,000	13,201,000,000	12,809,000,000	

	03/31/2022	12/31/2021	…	12/31/1987	09/30/1987	\
0	97,278,000,000	123,945,000,000	…	1,042,400,000	786,500,000	
1	97,278,000,000	123,945,000,000	…	1,042,400,000	786,500,000	
2	54,719,000,000	69,702,000,000	…	NaN	NaN	
3	42,559,000,000	54,243,000,000	…	NaN	NaN	
4	12,580,000,000	12,755,000,000	…	NaN	NaN	

	06/30/1987	03/31/1987	12/31/1986	09/30/1986	06/30/1986	\
0	637,100,000	575,300,000	662,300,000	510,800,000	448,300,000	
1	637,100,000	575,300,000	662,300,000	510,800,000	448,300,000	
2	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	

	03/31/1986	12/31/1985	09/30/1985
0	408,900,000	533,900,000	409,700,000
1	408,900,000	533,900,000	409,700,000

```
2          NaN          NaN          NaN
3          NaN          NaN          NaN
4          NaN          NaN          NaN

[5 rows x 155 columns]
```

[5]:
```
y.index = y.name
```

[6]:
```
y = pd.DataFrame(y.loc["BasicEPS", :]).iloc[2:,:]
```

[7]:
```
y.index = pd.to_datetime(y.index)
```

[8]:
```
# CHECK IF NAS ARE NO DIVIDEND PERIOD
y = y.sort_index().fillna(0.)
```

2 2.) Come up with 6 search terms you think could nowcast earnings. (Different than the ones I used) Add in 3 terms that that you think will not Nowcast earnings. Pull in the gtrends data

[9]:
```
from pytrends.request import TrendReq
```

[10]:
```
# Create pytrends object
pytrends = TrendReq(hl='en-US', tz=360)

# Set up the keywords and the timeframe
keywords = ['new phone technology', 'phone with screen','cell phone','sell␣
 ↪phone','new company','Apple IPO','global warming','lamborghini','paparazzi']␣
 ↪ # Add your keywords here
start_date = '2004-01-01'
end_date = '2024-01-01'

# Create an empty DataFrame to store the results
df = pd.DataFrame()

# Iterate through keywords and fetch data
for keyword in keywords:
    pytrends.build_payload([keyword], cat=0, timeframe=f'{start_date}␣
 ↪{end_date}', geo='', gprop='')
    interest_over_time_df = pytrends.interest_over_time()
    df[keyword] = interest_over_time_df[keyword]
```

[11]:
```
df = df.resample("Q").mean()
df
```

[11]:
```
            new phone technology  phone with screen  cell phone  sell phone  \
date
```

```
2004-03-31             59.666667           1.666667   79.666667   21.333333
2004-06-30             24.666667          13.000000   78.666667   24.666667
2004-09-30             54.666667           5.666667   86.333333   27.000000
2004-12-31             41.666667           6.000000   85.000000   24.666667
2005-03-31             73.000000           5.666667   86.333333   24.666667
…                            …                  …          …          …
2023-03-31             28.666667          81.666667   13.333333   57.666667
2023-06-30             25.333333          86.333333   13.000000   56.000000
2023-09-30             21.333333          93.333333   13.000000   61.333333
2023-12-31             27.666667          88.000000   13.000000   65.000000
2024-03-31             24.000000          85.000000   12.000000   67.000000

            new company  Apple IPO  global warming  lamborghini  paparazzi
date
2004-03-31    96.666667  38.333333       29.333333    41.000000  25.333333
2004-06-30    90.000000  13.666667       26.666667    41.000000  25.000000
2004-09-30    95.000000   0.000000       16.666667    33.333333  31.000000
2004-12-31    88.000000  28.666667       29.000000    32.666667  31.333333
2005-03-31    86.333333  16.333333       30.000000    35.666667  32.000000
…                   …          …              …            …          …
2023-03-31    61.333333  13.333333       13.333333    64.666667   9.333333
2023-06-30    58.666667  13.666667       13.000000    76.000000   9.666667
2023-09-30    54.000000  25.666667        9.333333    72.333333   9.333333
2023-12-31    60.666667  20.666667       10.333333    72.666667   8.333333
2024-03-31    58.000000  23.000000       11.000000    76.000000   8.000000

[81 rows x 9 columns]
```

[12]:
```
# ALIGN DATA
temp = pd.concat([y, X],axis = 1).dropna()
y = temp[["BasicEPS"]].copy()
X = temp.iloc[:,1:].copy()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[12], line 2
      1 # ALIGN DATA
----> 2 temp = pd.concat([y, X],axis = 1).dropna()
      3 y = temp[["BasicEPS"]].copy()
      4 X = temp.iloc[:,1:].copy()

NameError: name 'X' is not defined
```

3 3.) Normalize all the X data

[13]:
```
from sklearn.preprocessing import StandardScaler
```

[14]:
```
scaler = StandardScaler()
```

[15]:
```
X_scaled = scaler.fit_transform(df)
```

[ ]:

[ ]:

[ ]:

4 4.) Run a Lasso with lambda of .5. Plot a bar chart.

[16]:
```
from sklearn.linear_model import Lasso
```

[17]:
```
lasso = Lasso(alpha = .5)
```

[18]:
```
lasso.fit(X_scaled,y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[18], line 1
----> 1 lasso.fit(X_scaled,y)

File ~\anaconda3\Lib\site-packages\sklearn\base.py:1151, in _fit_context.
 ↪<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File ~\anaconda3\Lib\site-packages\sklearn\linear_model\_coordinate_descent.py:
 ↪905, in ElasticNet.fit(self, X, y, sample_weight, check_input)
    903 if check_input:
    904     X_copied = self.copy_X and self.fit_intercept
--> 905     X, y = self._validate_data(
    906         X,
    907         y,
    908         accept_sparse="csc",
    909         order="F",
    910         dtype=[np.float64, np.float32],
```

```
    911         copy=X_copied,
    912         multi_output=True,
    913         y_numeric=True,
    914     )
    915     y = check_array(
    916         y, order="F", copy=False, dtype=X.dtype.type, ensure_2d=False
    917     )
    919 n_samples, n_features = X.shape

File ~\anaconda3\Lib\site-packages\sklearn\base.py:621, in BaseEstimator.
 ↪_validate_data(self, X, y, reset, validate_separately, cast_to_ndarray,␣
 ↪**check_params)
    619         y = check_array(y, input_name="y", **check_y_params)
    620     else:
--> 621         X, y = check_X_y(X, y, **check_params)
    622     out = X, y
    624 if not no_val_X and check_params.get("ensure_2d", True):

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1165, in␣
 ↪check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy,␣
 ↪force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples,␣
 ↪ensure_min_features, y_numeric, estimator)
   1147 X = check_array(
   1148     X,
   1149     accept_sparse=accept_sparse,
   (…)
   1160     input_name="X",
   1161 )
   1163 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric,␣
 ↪estimator=estimator)
-> 1165 check_consistent_length(X, y)
   1167 return X, y

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:409, in␣
 ↪check_consistent_length(*arrays)
    407 uniques = np.unique(lengths)
    408 if len(uniques) > 1:
--> 409     raise ValueError(
    410         "Found input variables with inconsistent numbers of samples: %r
    411         % [int(l) for l in lengths]
    412     )

ValueError: Found input variables with inconsistent numbers of samples: [81, 15 ]
```

[35]:
```
coefficients = lasso.coefficients
```

```
---------------------------------------------------------------------------
```

```
AttributeError                            Traceback (most recent call last)
Cell In[35], line 1
----> 1 coefficients = lasso.coefficients

AttributeError: 'Lasso' object has no attribute 'coefficients'
```

[ ]:
```
plt.figure(figsize = (12,5)
plt.bar(range(len(coefficients)), coefficients, X.columns)
plt.axhline(0, color = "red")
plt.show()
```

5 5.) Do these coefficient magnitudes make sense?

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: