LEHIGH UNIVERSITY

Predicting Recessions using Unlikely Indicators

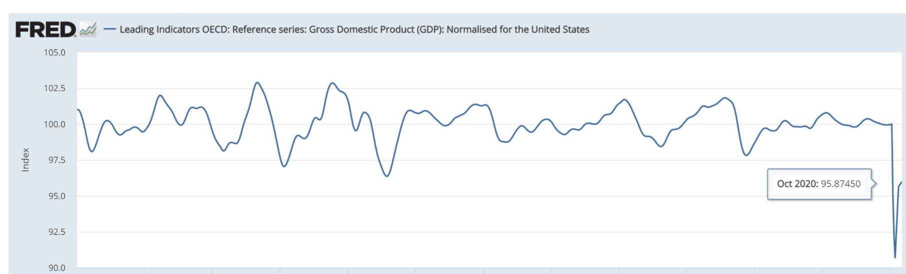
Anders Seline, Mollie Maggiacomo, and Andrew Zhang

Research Question

Our group intends to create a means of predicting if the economy as a whole shows signs of approaching a recessionary period by observing the values of a chosen set of factors. These include well-known recession indicators widely used by economists, as well as other data points near and dear to almost every college student. For starters, the usual metrics of an oncoming downturn we will consider are as follows:

- Consumer Confidence Index (CCI)
- 10-Federal Funds Rate Spread
- Normalized GDP
- Total Manufacturing Employment
- Inflation

However, as college students, we wanted to use some unusual recession indicators we are more interested in in our regression model



Unlikely Indicators

Alcohol

- During periods of recession, consumption of alcohol tends to go up as people feel the need to forget about their daily struggles

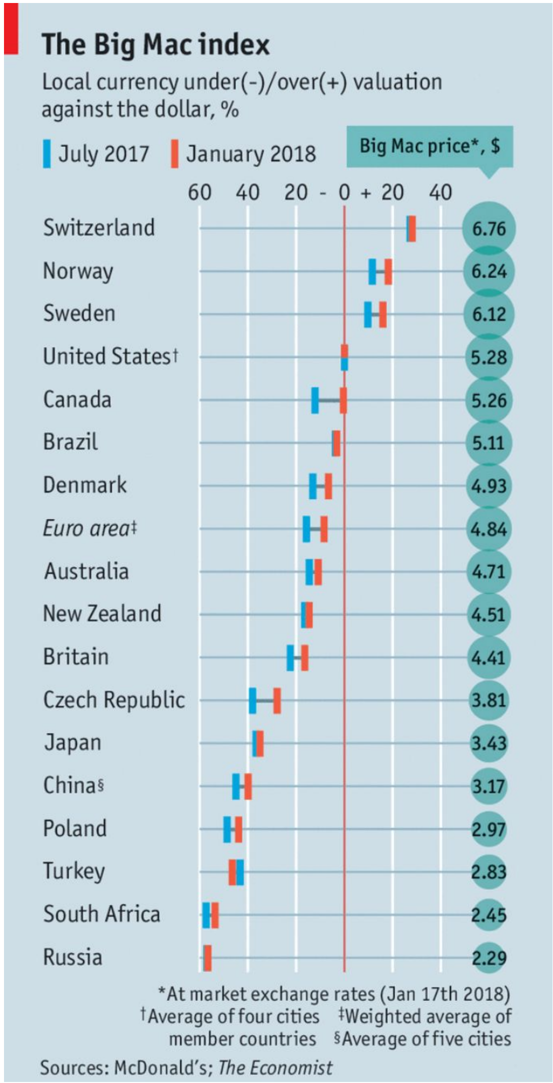
Copper

- Copper, a commodity material that is used heavily in construction, tends to lose demand during recession periods as construction projects get put on hold or pushed back

Unlikely Indicators

Fast Food

- We also wanted to use *The Economist's* Big Mac index as another indicator, but were ultimately unable to do so due to its short lifespan
- This would have served as a quasi-exchange rate, as it normalizes the price of goods in each country and compares the relative difference back to the monetary exchange rate
- As the dollar tends to weaken during recessions, we could have compared the US Big Mac Index with multiple others, and have our model search for trends



Step 1: Data Exploration

- Gathered our data from the FRED Economic Database
- Time Frame: 1967-2021
- CSV Files:
  - CCI Index
  - Copper Pricing
  - GDP
  - Inflation
  - Manufacturing Employment
  - 10-FF Yield Spread
  - Alcohol Sales Volume
  - Recession in Next Month

Name	Last Modified
CCI.csv	11 days ago
CPRICING.csv	11 days ago
full_time_indicators.csv	19 hours ago
GDP.csv	11 days ago
INFLATION.csv	19 hours ago
MANEMP.csv	19 hours ago
PCALC.csv	11 days ago
T10YFF.csv	11 days ago
USREC.csv	11 days ago

		date	cci	copper	copper_pct	gdp	alc	alc_pct	spread	emp	emp_pct	inflation	rec_next_month
1	0	1967-01-01	100.91371711802	55.2	0.0	100.868342478118	8.853	0.0	-0.36	18033	0.0	2.77278562259307	0.0
2	1	1967-02-01	101.063222562007	55.2	0.0	100.7207424096	8.93733333333333	0.00952596106781134	0.52	17978	-0.0030499639549714797	2.8977031667840922	0.0
3	2	1967-03-01	101.165525611314	54.0	-0.021739130434782705	100.553109723236	9.02166666666667	0.009436073399970102	0.91	17940	-0.0021136945155190157	3.022620710975115	0.0
4	3	1967-04-01	101.242167398696	53.1	-0.016666666666666607	100.379757610786	9.106	0.00934786624792161	0.0	17878	-0.003455964325529526	3.1475382551661375	0.0
5	4	1967-05-01	101.297344700464	52.4	-0.013182674199623379	100.219858524299	9.19033333333333	0.009261292920418773	0.77	17832	-0.0025729947421412325	3.27245579935716	0.0
6	5	1967-06-01	101.321804466293	52.1	-0.005725190839694583	100.08987011546	9.27466666666667	0.009176308440027592	0.81	17812	-0.0011215791834903177	3.3973733435481823	0.0
7	6	1967-07-01	101.286883156174	52.1	0.0	99.995384636833	9.359	0.00909286946520993	1.72	17784	-0.0015719739501459662	3.5222908877392047	0.0
8	7	1967-08-01	101.16926361197	52.1	0.0	99.9385770623946	9.44333333333333	0.009010934216618693	1.43	17905	0.006803868645973932	3.6472084319302276	0.0
9	8	1967-09-01	101.035113006269	52.7	0.011516314779270731	99.9232325195438	9.52766666666667	0.008930462407342077	1.25	17794	-0.006199385646467492	3.77212597612125	0.0
10	9	1967-10-01	100.974258552544	53.8	0.020872865275142205	99.9572481164241	9.612	0.008851415176853283	1.56	17800	0.00033719231201523314	3.8970435203122724	0.0
11	10	1967-11-01	101.050894278172	56.5	0.05018587360594795	100.048762304548	9.69633333333333	0.00877375502843658	1.57	17985	0.010393258426966367	4.021961064503295	0.0
12	11	1967-12-01	101.180991644122	57.6	0.019469026548672552	100.198938071591	9.78066666666667	0.008697445769878742	1.28	18025	0.0022240756185709643	4.146878608694317	0.0
13	12	1968-01-01	101.248056132149	59.0	0.02430555555555558	100.39070213727	9.865	0.00862245245722848	1.055	18040	0.0008321775312065682	4.27179615288534	0.0
14	13	1968-02-01	101.165422193022	62.3	0.055932203389830404	100.598771991729	9.96075	0.009706031424227213	0.83	18054	0.0007760532150775656	4.371011990168854	0.0
15	14	1968-03-01	101.00158107131	63.7	0.022471910112359605	100.796775426688	10.0565	0.009612729965112976	0.81	18067	0.0007200620361138554	4.470227827452367	0.0
16	15	1968-04-01	100.848051232027	61.8	-0.02982731554160134	100.963361666027	10.15225	0.009521205190672788	0.14	18131	0.0035423700669729996	4.56944366473588	0.0
17	16	1968-05-01	100.768455969851	56.2	-0.09061488673139151	101.082664880418	10.248	0.009431406830997968	-0.65	18190	0.0032540951960731146	4.668659502019393	0.0
18	17	1968-06-01	100.737289735917	55.2	-0.017793594306049876	101.147130914283	10.34375	0.009343286494925884	-0.27	18228	0.002089059923034542	4.767875339302907	0.0
19	18	1968-07-01	100.719513399992	54.0	-0.021739130434782705	101.163793936173	10.439499999999999	0.00925679758308151	-0.51	18265	0.0020298441957427027	4.86709117658642	0.0
20	19	1968-08-01	100.70172048441	53.6	-0.007407407407407418	101.147155073388	10.53525	0.009171895205709157	-0.62	18254	-0.0006022447303586365	4.966307013869933	0.0
21	20	1968-09-01	100.709913499998	53.5	-0.001865671641791078	101.115610095348	10.631	0.00908853610498106	-0.33	18252	0.00010956502684345004	5.065522851153447	0.0
22	21	1968-10-01	100.795977183197	53.8	0.005607476635513864	101.090455953083	10.72675	0.009006678581506833	-0.51	18293	0.002246329169406147	5.16473868843696	0.0
23	22	1968-11-01	100.998165800676	54.0	0.003717472118959231	101.088685764161	10.8225	0.00892628242477933	-0.39	18346	0.0028972831137592525	5.263954525720473	0.0
24	23	1968-12-01	101.223719787677	54.9	0.016666666666666607	101.115572282148	10.91825	0.008847308847308888	-0.22	18410	0.0034884988553363083	5.363170363003986	0.0
25	24	1969-01-01	101.337502720813	57.3	0.04371584699453557	101.155747531586	11.014	0.008769720422228788	-0.20500000000000002	18432	0.0011950027159153276	5.4623862002875	0.0
26	25	1969-02-01	101.227272920004	57.6	0.00523560209424101	101.185767357679	11.12341666666666	0.009934326009321426	-0.19	18502	0.0037977430555555802	5.493708628470419	0.0
27	26	1969-03-01	100.980726105618	58.6	0.017361111111111116	101.185796203767	11.232833333333332	0.009836605958900568	-0.74	18558	0.0030266998162360448	5.525031056653338	0.0
28	27	1969-04-01	100.695951130211	60.2	0.027303754266211566	101.152107445208	11.34225	0.009740789649391068	-0.38	18554	0.00021554046772276347	5.556353484836258	0.0
[illegible]	[illegible]	1969-05-01	100.4282027225	61.4	0.019933554817275656	101.08703929378	11.451666666666666	0.009646821985643594	-2.0	18588	0.0018324889511696085	5.587675913019177	0.0

Step 1: Data Exploration

Merging Datasets and Formatting Data to Time-Frame

```
[68]: r = pd.DataFrame(pd.date_range(start=rspread.DATE.min(), end=rspread.DATE.max()),columns=["DATE"])
      rspread = r.merge(rspread.assign(DATE=pd.to_datetime(rspread['DATE'])), how='left').fillna(method='ffill')
```

```
[69]: # merge all datasets together
      cci_copper = pd.merge(left=cci, right=copper, how='inner', on='DATE')
      cc_gdp = pd.merge(left=cci_copper, right=gdp, how='inner', on='DATE')
      cc_gdp_alc = pd.merge(left=cc_gdp, right=alc, how='left', on='DATE')
      cc_gdp_alc = cc_gdp_alc.interpolate(method='linear')
      cc_gdp_alc['DATE'] = pd.to_datetime(cc_gdp_alc['DATE'])
      cc_gdp_alc_r = pd.merge(left=cc_gdp_alc, right=rspread, how='left', on='DATE')
      cc_gdp_alc_r['DATE'] = pd.to_datetime(cc_gdp_alc_r['DATE'])
      # get rid of bad data in the column before adding to set
      spread = cc_gdp_alc_r['T10YFF']
      for i in range(len(spread)):
          # linearlly interpolate
          if spread[i] == '.':
              spread[i] = (
                  float(
                  (float(spread[i - 1]) + float(spread[i + 1]))
                  / 2
                  )
              )
      spread_np = spread.astype(np.float64)
      del cc_gdp_alc_r['T10YFF']
      cc_gdp_alc_r['T10YFF'] = spread_np
      # sloppy, but works
      man_emp['DATE'] = pd.to_datetime(man_emp['DATE'])
      cc_gdp_alc_r_emp = pd.merge(left=cc_gdp_alc_r, right=man_emp, how='left', on='DATE')
      inflation['DATE'] = pd.to_datetime(inflation['DATE'])
      indicators_w_rec = pd.merge(left=cc_gdp_alc_r_emp, right=inflation, how='left', on='DATE')
      rec['DATE'] = pd.to_datetime(rec['DATE'])
      indicators = pd.merge(left=indicators_w_rec, right=rec, how='left', on='DATE')
      indicators['FPCPITOTLZGUSA'] = indicators['FPCPITOTLZGUSA'].interpolate(method='linear')
      warnings.filterwarnings('ignore')
```

```
[70]: # rename all columns
      indicators = indicators.rename(columns={
          'DATE' : 'date', 'CSCICP03USM665S' : 'cci', 'WPUSI019011' : 'copper',
          'USALORSGPNOSTSAM' : 'gdp', 'DAOPRC1A027NBEA' : 'alc',
          'T10YFF' : 'spread', 'MANEMP' : 'emp', 'FPCPITOTLZGUSA' : 'inflation',
           'USREC' : 'rec'
      })
      # remove nans in alc column --> have all dates defined already
      indicators['alc'] = indicators['alc'].fillna(method='ffill')
      # shift all recession values
      indicators['rec_next_month'] = indicators['rec'].shift(-1)
      indicators = indicators.drop('rec', axis=1)
      # and drop the last row to remove nan value from shift
      indicators = indicators.drop(labels=646, axis=0)
```

Step 1: Data Exploration

Data Usage and Normalization

- We had to normalize the alcohol sales and price of copper over time due to the increasing values of raw data over time by creating a column to calculate the % change for each time interval

```
[74]: indicators['copper_pct'] = indicators['copper'].pct_change()
      indicators['alc_pct'] = indicators['alc'].pct_change()
      indicators['emp_pct'] = indicators['emp'].pct_change()
      inf_list = list(indicators['inflation'])
      pct_change = [0]
      # drop
      for i in range(1, len(inf_list)):
          b = inf_list[i]
          a = inf_list[i - 1]
          chng = (b - a) / np.absolute(a)
          pct_change.append(chng)

[76]: indicators['copper_pct'] = indicators['copper_pct'].fillna(0)
      indicators['alc_pct'] = indicators['alc_pct'].fillna(0)
      indicators['emp_pct'] = indicators['emp_pct'].fillna(0)
      indicators['inflation_pct'] = pct_change

      indicators = indicators[[
          'date', 'cci', 'copper', 'copper_pct', 'gdp', 'alc', 'alc_pct',
          'spread', 'emp', 'emp_pct', 'inflation', 'inflation_pct',
          'rec_next_month'
      ]]
      indicators = indicators.drop('inflation_pct', axis=1)
      indicators.to_csv('data/full_time_indicators.csv')
      indicators.describe()
```

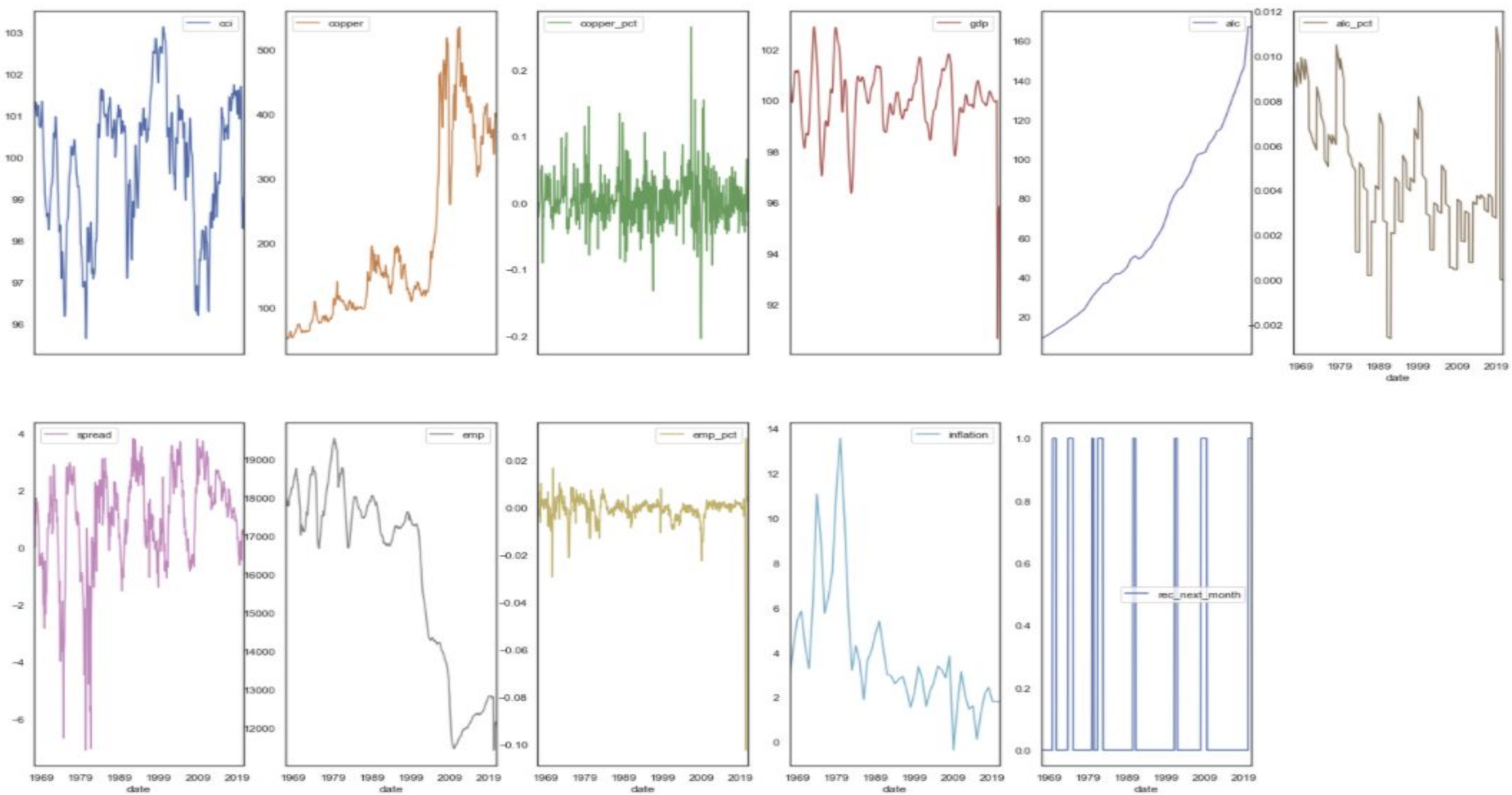
Step 1: Data Exploration

Data Visualization

	date	cci	copper	gdp	alc	spread	emp	inflation	rec_next_month
0	1967-01-01	100.913717	55.2	100.868342	8.853000	-0.36	18033	2.772786	0.0
1	1967-02-01	101.063223	55.2	100.720742	8.937333	0.52	17978	2.897703	0.0
2	1967-03-01	101.165526	54.0	100.553110	9.021667	0.91	17940	3.022621	0.0
3	1967-04-01	101.242167	53.1	100.379758	9.106000	0.00	17878	3.147538	0.0
4	1967-05-01	101.297345	52.4	100.219859	9.190333	0.77	17832	3.272456	0.0
...	...	...	...	...	...	...	...	...	...
641	2020-06-01	98.281638	359.8	92.322976	167.134000	0.61	11999	1.812210	1.0
642	2020-07-01	98.301124	383.5	93.984370	167.134000	0.61	12037	1.812210	1.0
643	2020-08-01	98.526634	394.3	95.633067	167.134000	0.45	12068	1.812210	1.0
644	2020-09-01	98.875920	401.6	95.741927	167.134000	0.59	12123	1.812210	1.0
645	2020-10-01	99.042669	396.3	95.845674	167.134000	0.59	12155	1.812210	1.0

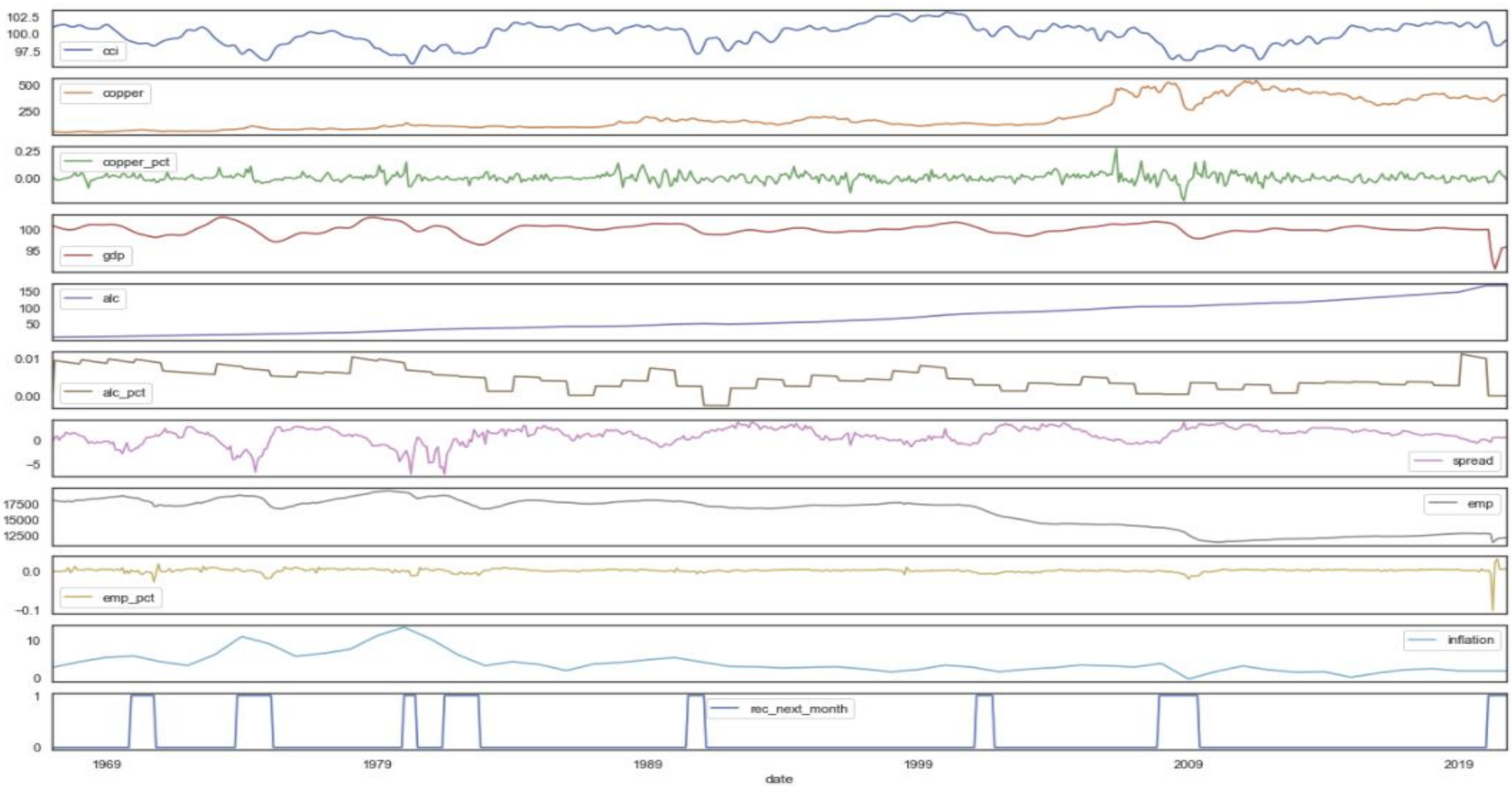
Step 1: Data Exploration

Data Visualization



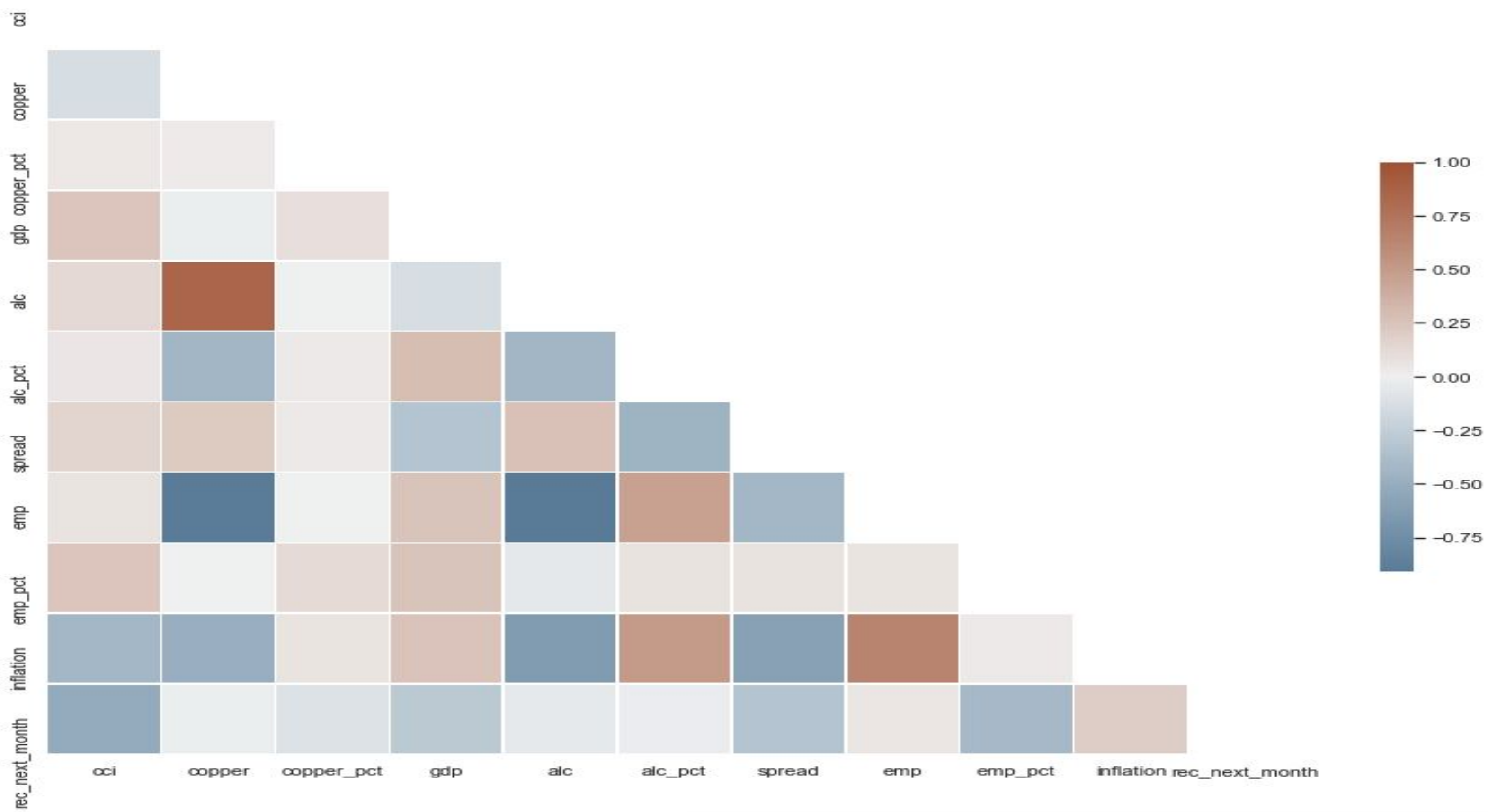
Step 1: Data Exploration

Data Visualization



Step 1: Data Exploration

Data Visualization



Step 2: Model Manipulation

Models Used

1. Logistic Regression
2. Random Forest Model
3. Gradient Boosted Trees

Model Fitting

1. Initial fitting: Years 1967 (start of time) to 1980
2. Time Series fitting: 1980 to 2010
3. Testing Data: 2010 to present

Step 2: Model Manipulation

Model Tuning

Logistic Regression:

- *class weight : balanced*, ensures that the weights of each class are adjusted inversely proportional to class frequencies. This was a conscious decision made after comparing different weighting methods

Random Forest & Gradient Boosted Trees:

- These models were tuned using cross validation to determine optimal parameters
- Note the difference between *n_estimators* - since gradient boosting is less inclined to overfitting, we can use slightly more estimators here to boost accuracy

Step 2: Model Manipulation

```
params = {
    'learning_rate' : 0.5,
    'n_estimators' : 500,
    'verbosity' : 0,
    'use_label_encoder' : False
}
gb = XGBClassifier(**params)
```

```
params = {
    'max_iter':1000,
    'class_weight':'balanced'
}
logit = LogisticRegression(**params)
```

```
params = {
    'bootstrap' : 'True',
    'class_weight' : 'balanced',
    'min_samples_leaf' : 3,
    'min_samples_split' : 10,
    'n_estimators' : 400
}
rand_forest = RandomForestClassifier(**params)
```

Step 2: Model Manipulation

Model 1: Logistic Regression

1. Set parameters: ‘max_iter’ and ‘class_weight’
2. Fit to training data before 1980
3. Incorporating more data, one month at a time in a series , from 1980 to 2010
4. Get the weights of each indicator, and the prediction, at each time interval
5. Merge factor weights into combined dataframe = logi_results
6. Create dataframe for the classification report using predicted and actual recession values = logi_class_rep
7. Save both df’s to CSV files in the outputs folder for further analysis

Step 2: Model Manipulation

Model 2: Random Forest

1. Input parameters: ‘bootstrap’, ‘class_weight’, ‘min_samples_leaf’, ‘min_samples_split’, ‘n_estimators’
2. Fit each model on period 1967-1980, and perform time series analysis from 1980-2010
3. Create Random Forest model with *RandomForestClassifier* and fit x and y_train data
4. Create coefficient lists for each variable
5. Get all the predictions
6. Add dates and combine into one df = rf_results
7. Create dataframe for the classification report = rf_class_rep
8. Save both df’s to CSV files in the outputs folder for further analysis

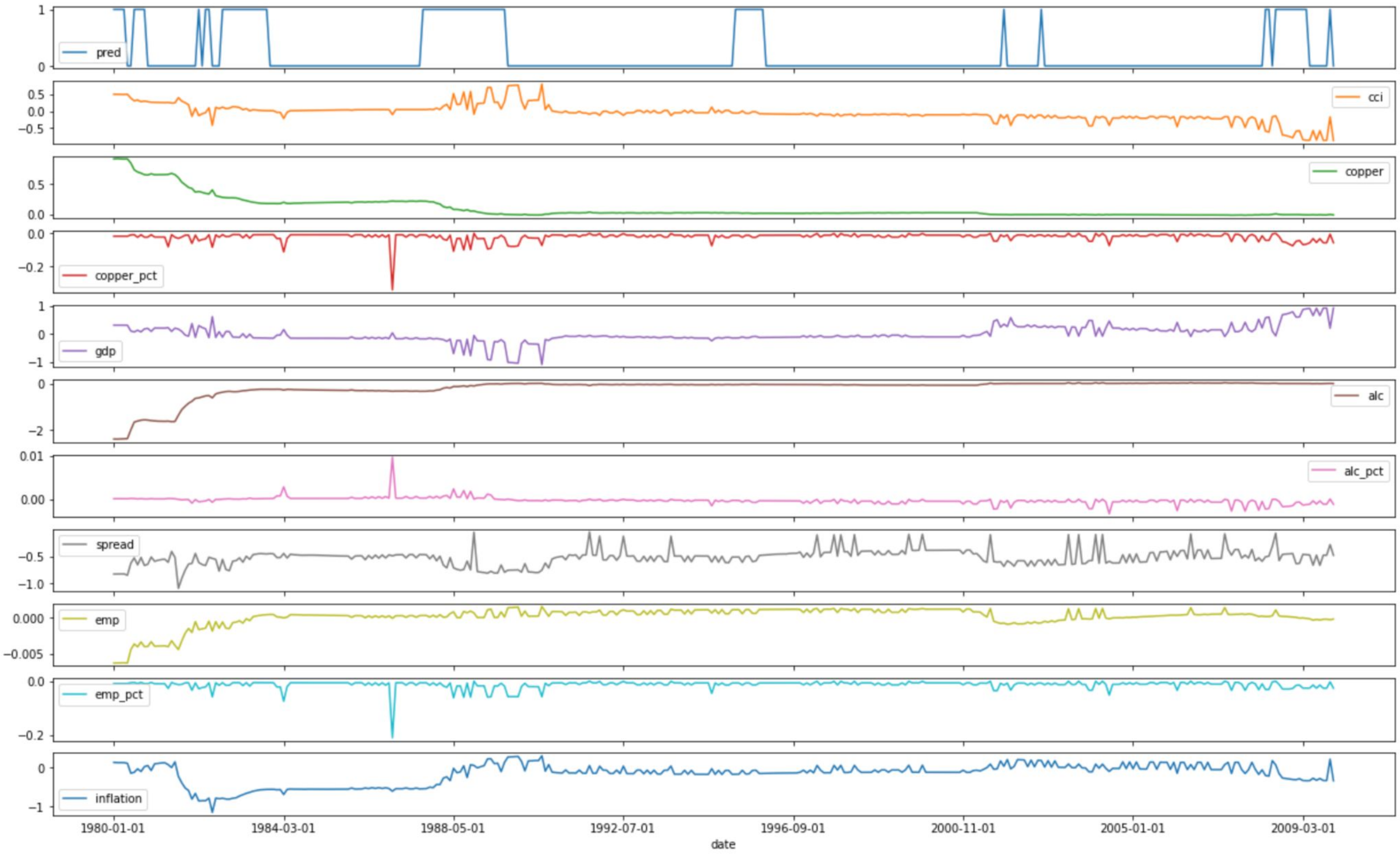
Step 2: Model Manipulation

Model 3: Gradient Boosted Trees

1. Input parameters: ‘learning_rate’, ‘n_estimators’, ‘verbosity’, ‘use_label_encoder’
2. Fit each model on period 1967-1980, and perform time series analysis from 1980-2010
3. Create Gradient Boosted Trees model with *XGBClassifier* and fit data
4. Create coefficient lists for each variable
5. Get all the predictions
6. Add dates and combine into one df = gb_results
7. Create dataframe for the classification report = gb_class_rep
8. Save both df’s to CSV files in the outputs folder for further analysis

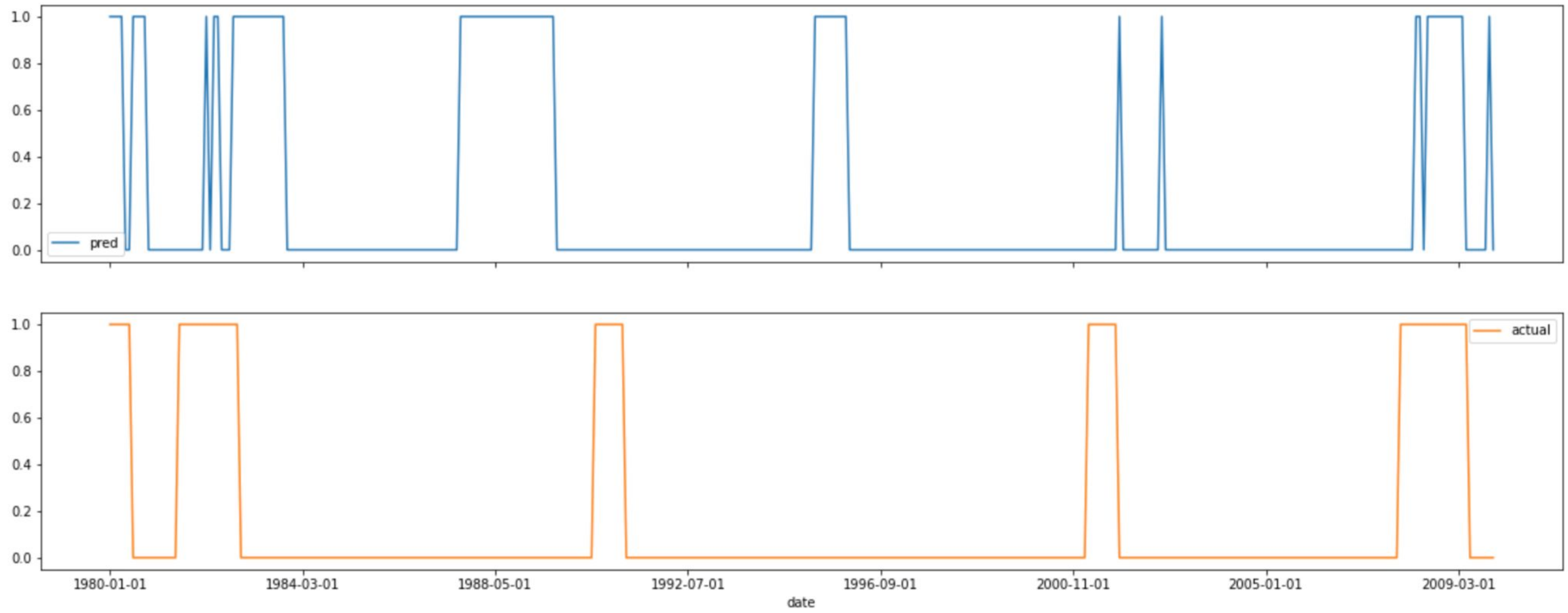
Step 3: Analyze Models

Model 1: Logistic Regression Results



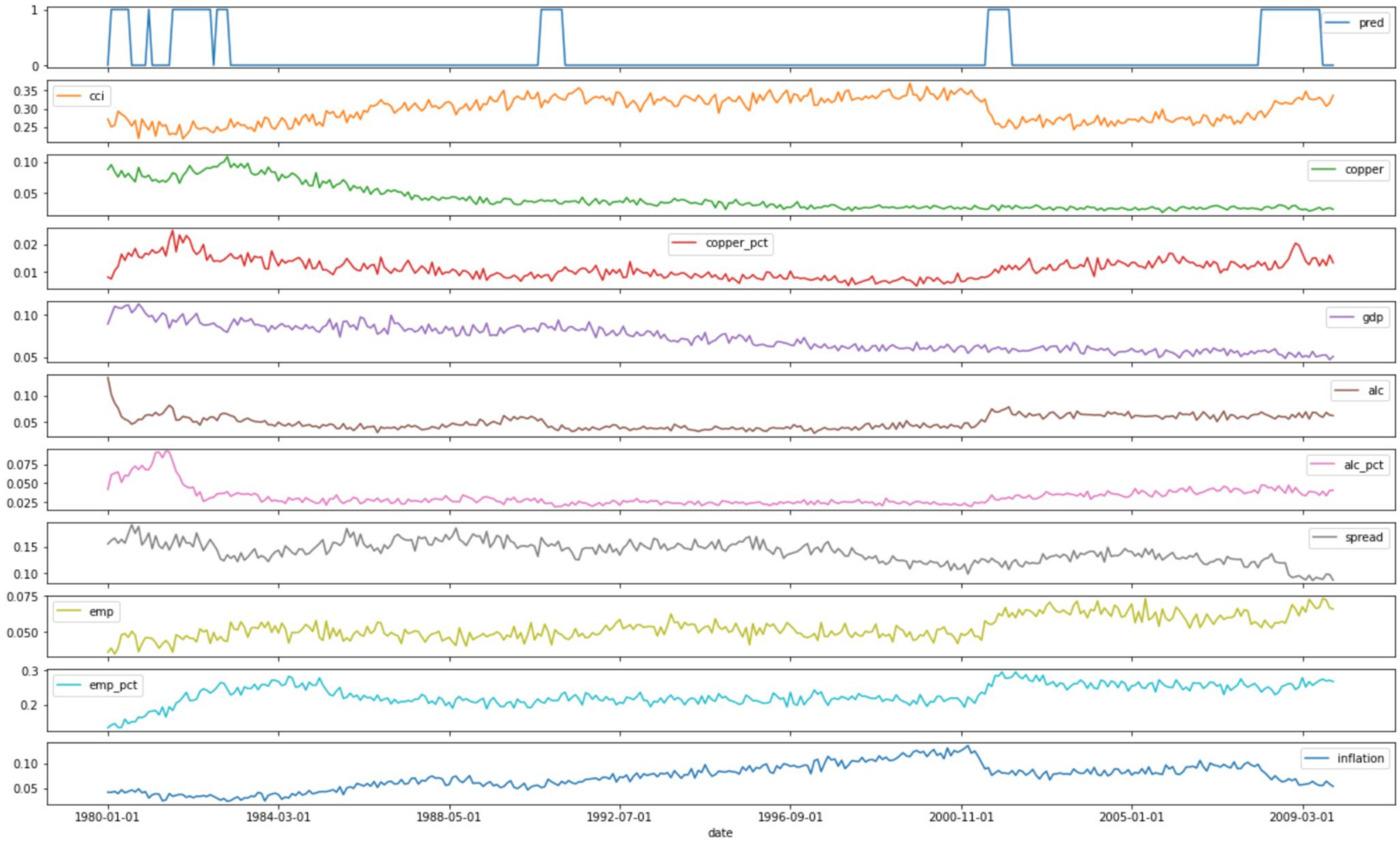
Step 3: Analyze Models

Model 1: Logistic Regression Results



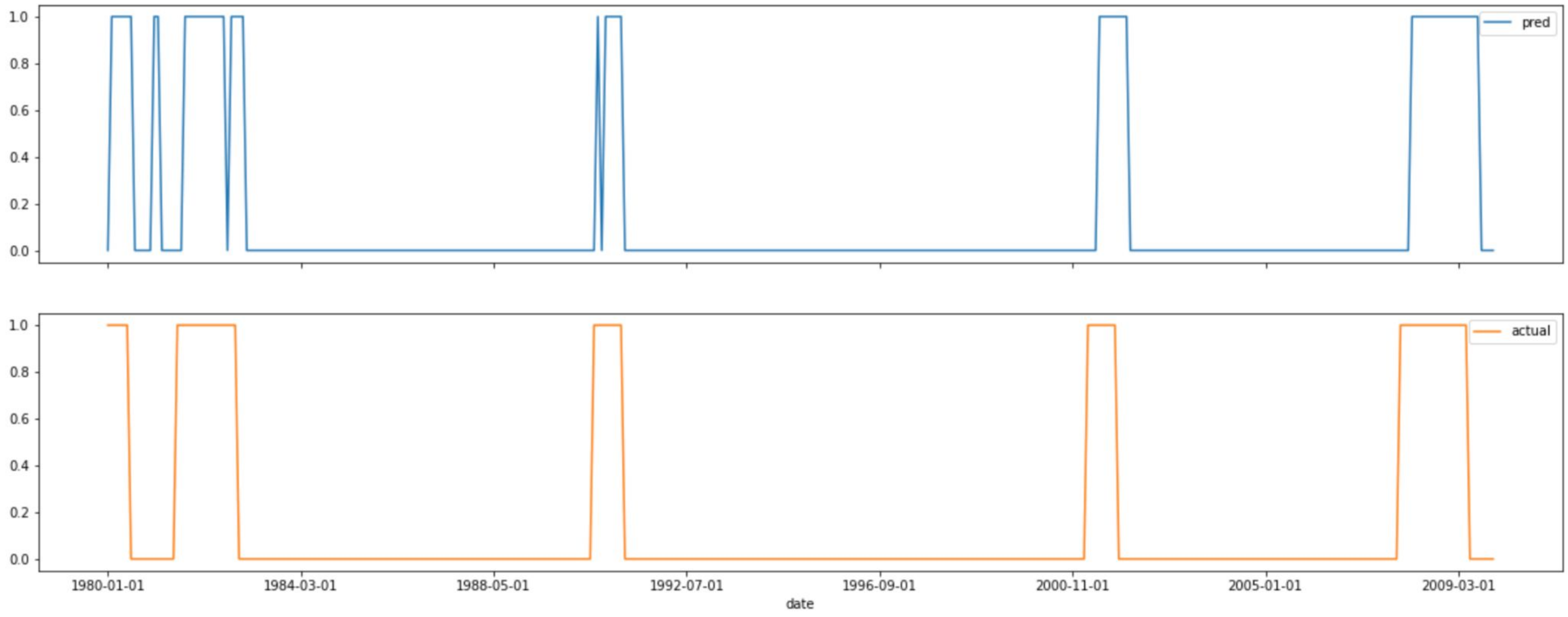
Step 3: Analyze Models

Model 2: Random Forest Results



Step 3: Analyze Models

Model 2: Random Forest Results

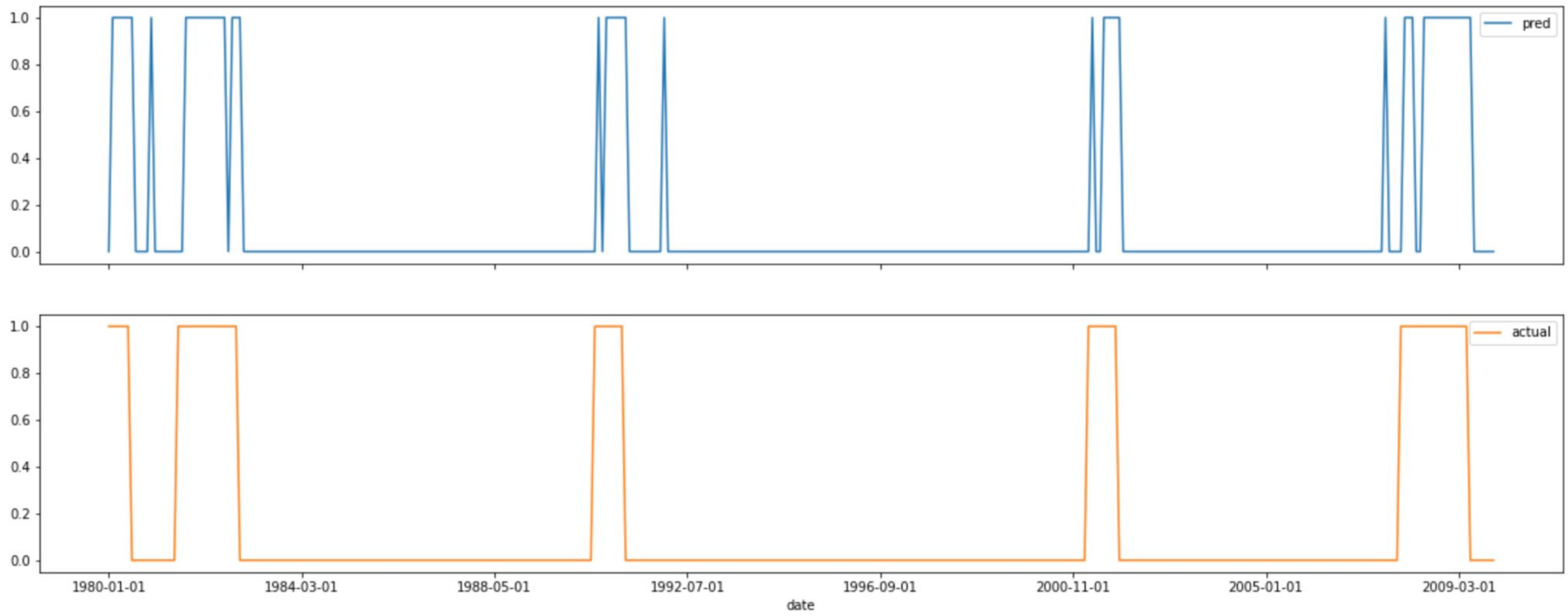


Step 3: Analyze Models

Model 3: Gradient Boosted Trees Results

Step 3: Analyze Models

Model 3: Gradient Boosted Trees Results



Step 3: Analyze Models

Classification Report Results on Full Dataset

Logistical Regression Report

	precision	recall	f1-score	support
no recession	0.88	0.83	0.85	304
recession	0.28	0.38	0.32	56
accuracy			0.76	360
macro avg	0.58	0.60	0.59	360
weighted avg	0.79	0.76	0.77	360

Gradient Boosted Trees Report

	precision	recall	f1-score	support
no recession	0.96	0.97	0.97	304
recession	0.85	0.79	0.81	56
accuracy			0.94	360
macro avg	0.90	0.88	0.89	360
weighted avg	0.94	0.94	0.94	360

Gradient Boosted Trees Report

	precision	recall	f1-score	support
no recession	0.96	0.97	0.97	304
recession	0.85	0.79	0.81	56
accuracy			0.94	360
macro avg	0.90	0.88	0.89	360
weighted avg	0.94	0.94	0.94	360

Step 3: Analyze Models

Classification Report Results on Test Data

Logistic Regression

	precision	recall	f1-score	support
no recession	1.00	0.99	1.00	121
recession	0.90	1.00	0.95	9
accuracy			0.99	130
macro avg	0.95	1.00	0.97	130
weighted avg	0.99	0.99	0.99	130

Random Forest

	precision	recall	f1-score	support
no recession	1.00	1.00	1.00	121
recession	1.00	1.00	1.00	9
accuracy			1.00	130
macro avg	1.00	1.00	1.00	130
weighted avg	1.00	1.00	1.00	130

Gradient Boosted Trees

	precision	recall	f1-score	support
no recession	1.00	1.00	1.00	121
recession	1.00	1.00	1.00	9
accuracy			1.00	130
macro avg	1.00	1.00	1.00	130
weighted avg	1.00	1.00	1.00	130

Analysis of Results

- Our models were very successful in predicting recessions as shown by the graphical representations and the classification reports
- Depending on the conditions of the recession, different indicators responded differently during different recession periods

Questions?