綾里の二酸化炭素濃度の予測

Prediction of carbon dioxide concentration in Ryori

経済学部2年 No.22020372

松江 陽

目次

分析概要

配布されたデータ(1987年1月～2021年6月の綾里における二酸化炭素濃度の月平均値)

から、学習用データと検証用データを作成

↓

学習用データをもとに、

①成分分解、②SARIMAモデル、③LSTMモデル　の3つの手法で分析し、未来予測を行う

↓

検証用データを用いて、3つの予測結果を比較し、考察する

分析の準備

- 速報値でないものを用いる
- 欠損値の確認→2011年4月が欠測

→欠測地点より前を学習用データとしてみる

（分析①、②では .dropna を用いればよいが、
分析③では欠損値があると処理が煩雑になりそう）

→学習用と検証用の比が約 3:1 になり、分割として丁度よいため、この分割方法を採用する

※今回は都合よく、欠損値のない学習用データを得られたが、そうでない場合には、スプライン補間などによる穴埋めを考える

```python
#欠損値の確認
df[df['co2'] == '--']
```

	co2
date	
2011-04-01	--

```python
#分析の対象と検証用に分ける
#今回は欠損値を含まない、2011年3月までを分析の対象にする
train = df[:'2011-03-01']
test = df['2011-04-01':]
```

```python
#分析対象と検証用の割合が約3:1であることが確認できるので、
len(train)/(len(train)+len(test))
```

```
0.7348484848484849
```

分析の準備

●可視化

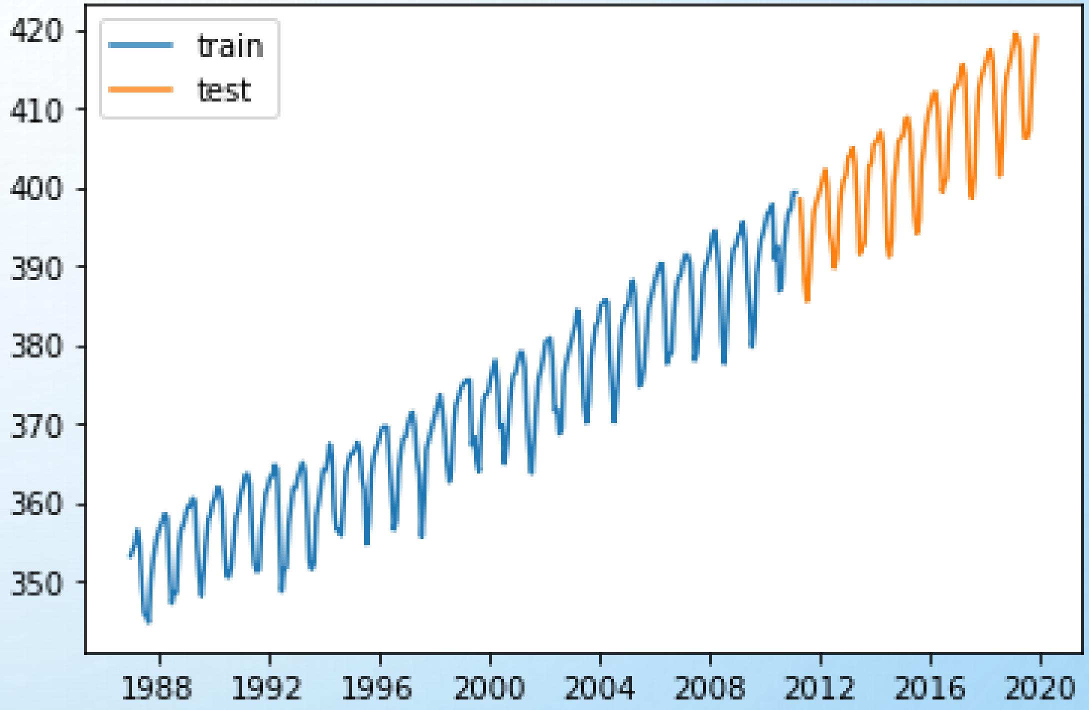
ギザギザ(山は12か月周期)

→季節変化の影響

→夏は植物の光合成が盛んになり、大気中の二酸化炭素濃度は減少

右肩上がり

→二酸化炭素排出量の増加



分析の準備

- 自己相関・偏自己相関の可視化

自己相関関数（Autocorrelation function）

過去の値が現在の値にどれくらい影響しているか、その関係性を示す

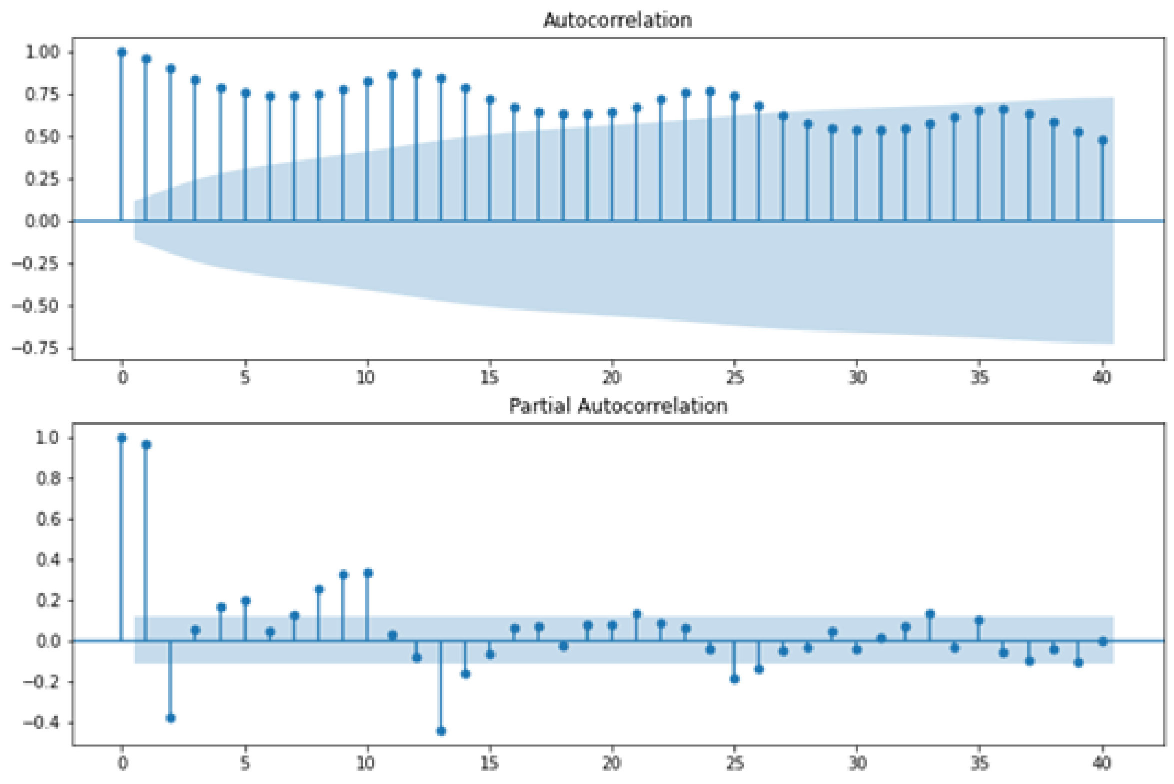
偏自己相関関数（Partial autocorrelation）

推移律を考慮した自己相関関数

自己相関のラグ（ずらす時点の度合い）は12、24、36で高くなっている

→12か月の周期をもっている

```
#自己相関のグラフ
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(train, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(train, lags=40, ax=ax2)
```



水色になっている部分は95%信頼区間を表している

分析① - 成分分解による予測

時系列データの観測値を、以下のように分解する。

観測値 ＝ トレンド成分 + 季節成分 + 残差成分

トレンド成分 : 一定の増加傾向、あるいは減少傾向のような、長期に渡るデータの傾向

季節成分 : 四季、雨季・乾季、月・週などによって、一定の間隔で繰り返される波動要因

残差成分 : 上記2つのパターン要因では説明できない不規則変動

分析① - 成分分解による予測

手順

1. 元データの成分分解
2. トレンドを回帰モデルに当てはめる
3. 求めた回帰式からトレンドを予測
4. 予測したトレンドに季節成分を加える
5. 予測結果の可視化

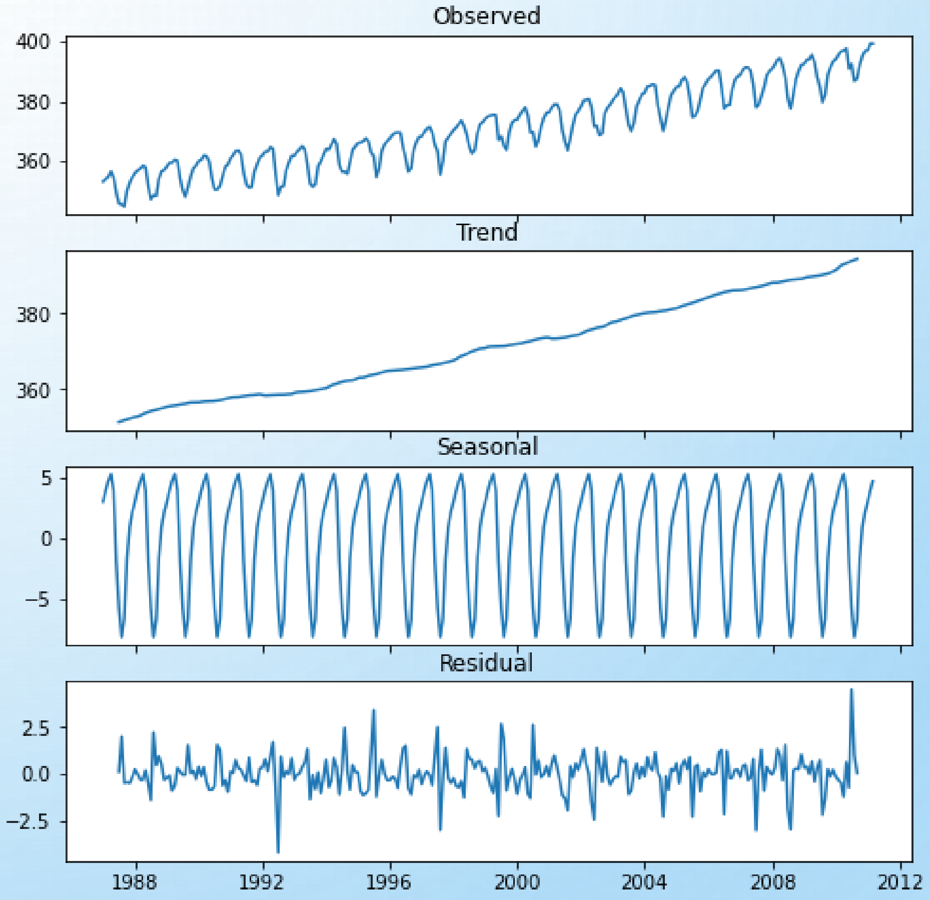
分析① - 成分分解による予測

手順

1. **元データの成分分解**
2. トレンドを回帰モデルに当てはめる
3. 求めた回帰式からトレンドを予測
4. 予測したトレンドに季節成分を加える
5. 予測結果の可視化

```
#必要ライブラリのインポート
import statsmodels.api as sm
```

```
#トレンド成分、季節成分、残差成分に分解
#加法モデル(デフォルト)
decompose_result = sm.tsa.seasonal_decompose(train, period = 12)
```



- 季節変動の周期は12
- 傾向変動の大きさに関係なく一定の季節変動と見なせるので加法モデルを用いる。

（傾向変動が大きくなれば、それに比例して季節変動も大きくなる場合は乗法モデルを用いる。）

分析① - 成分分解による予測

手順

1. 元データの成分分解
2. **トレンドを回帰モデルに当てはめる**
3. 求めた回帰式からトレンドを予測
4. 予測したトレンドに季節成分を加える
5. 予測結果の可視化

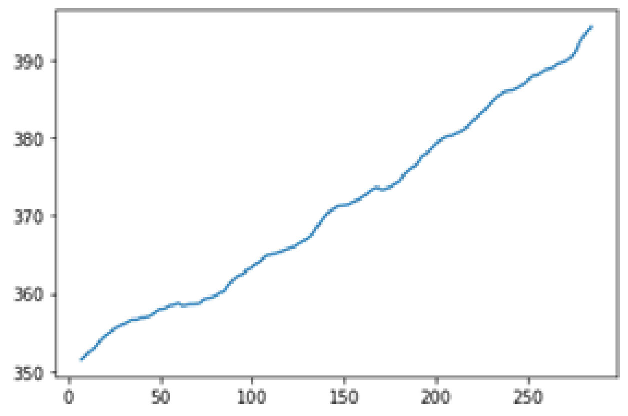
- 計測開始時点の1987年1月を x =1、1987年2月を $x = 2$ … とおく

- トレンドは直線とみなし、1次回帰式にあてはめる

結果より、予測したトレンドは、

$$y = 349.5014 + 0.1478x$$

で表される



OLS Regression Results

Dep. Variable:	trend	**R-squared:**	0.991
Model:	OLS	**Adj. R-squared:**	0.991
Method:	Least Squares	**F-statistic:**	2.919e+04
Date:	Sun, 13 Mar 2022	**Prob (F-statistic):**	9.20e-283
Time:	13:02:21	**Log-Likelihood:**	-437.17
No. Observations:	279	**AIC:**	878.3
Df Residuals:	277	**BIC:**	885.6
Df Model:	1		
Covariance Type:	nonrobust		

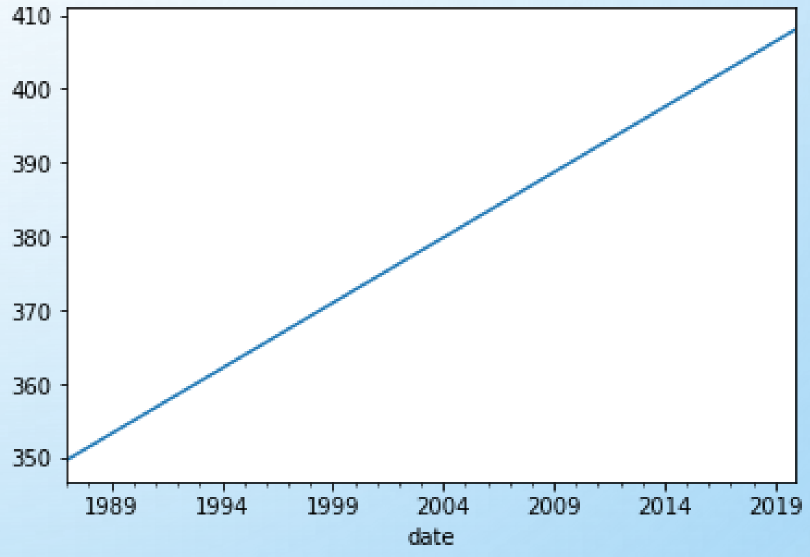
	coef	std err	t	P>|t|	[0.025	0.975]
const	349.5014	0.144	2423.242	0.000	349.217	349.785
x1	0.1478	0.001	170.850	0.000	0.146	0.149

分析① - 成分分解による予測

手順

1. 元データの成分分解
2. トレンドを回帰モデルに当てはめる
3. **求めた回帰式からトレンドを予測**
4. 予測したトレンドに季節成分を加える
5. 予測結果の可視化

```
#回帰式からトレンドを予測
result_1 = df.reset_index()
result_1.index = result_1.index+1
result_1['prtrend'] =  w0 + w1*result_1.index
```



分析① - 成分分解による予測

手順

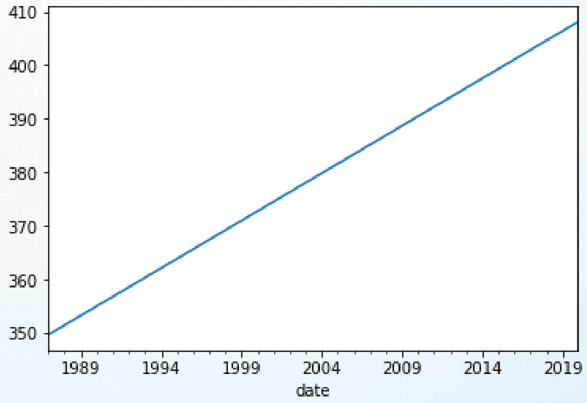
1. 元データの成分分解
2. トレンドを回帰モデルに当てはめる
3. 求めた回帰式からトレンドを予測
4. **予測したトレンドに季節成分を加える**
5. 予測結果の可視化

prtrend：予測したトレンド

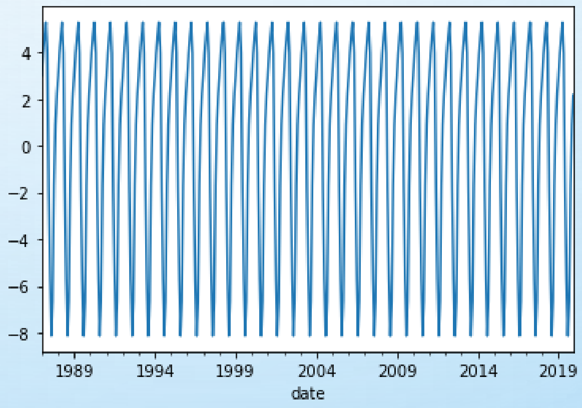
seasonal：分解した季節成分

pred_1：予測結果（prtrend + seasonal）

学習用データから予測したトレンド



学習用データから分解した季節成分



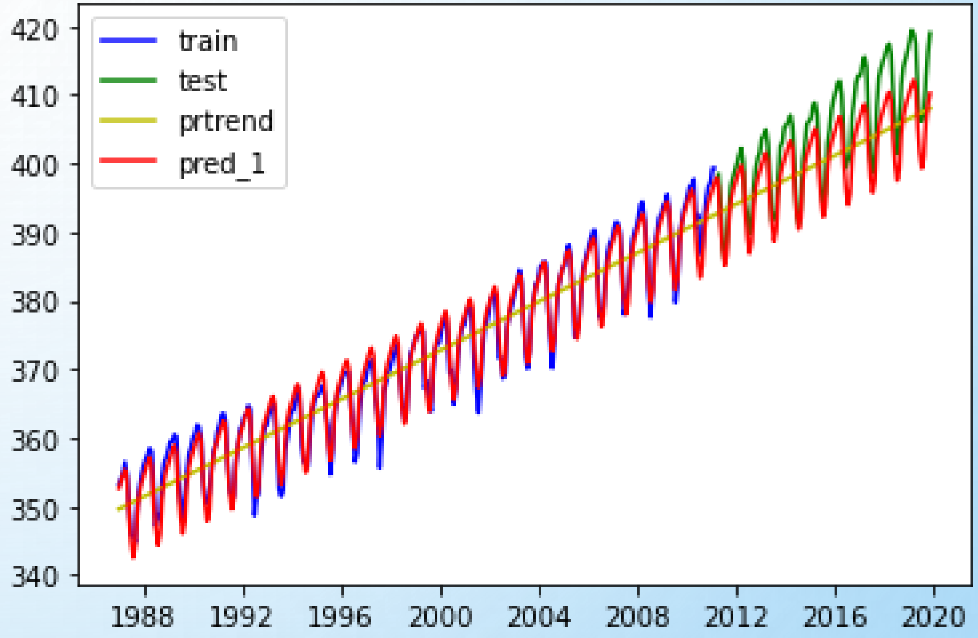
	prtrend	seasonal	pred_1
date			
1987-01-01	349.649161	3.016650	352.665811
1987-02-01	349.796943	3.996722	353.793666
1987-03-01	349.944726	4.705418	354.650144
1987-04-01	350.092508	5.290744	355.383252
1987-05-01	350.240291	3.963932	354.204223
...	...	...	...
2019-08-01	407.432091	-8.126655	399.305436
2019-09-01	407.579874	-6.661724	400.918149
2019-10-01	407.727656	-1.711974	406.015683
2019-11-01	407.875439	0.923896	408.799335
2019-12-01	408.023221	2.175708	410.198929

396 rows × 3 columns

分析① - 成分分解による予測

手順

1. 元データの成分分解
2. トレンドを回帰モデルに当てはめる
3. 求めた回帰式からトレンドを予測
4. 予測したトレンドに季節成分を加える
5. **予測結果の可視化**



参考のため、予測したトレンドprtrendも可視化した。

分析② - SARIMAモデルによる予測

● **ARIMA…Autoregressive Integrated Moving Average : 自己回帰和分移動平均モデル**

自己回帰モデル(ARモデル)、移動平均モデル(MAモデル)、和分モデル(Iモデル)の
3モデルを組み合わせたモデル

ARIMA(p, d, q)

● **SARIMA…Seasonal ARIMA**

ARIMAモデルに周期成分を取り入れたモデル

時系列方向の説明にARIMA(p, d, q) モデルを使い、周期方向の説明にも
ARIMA(P, D, Q)モデルを使うため、周期 s を合わせて7つの次数を決める

SARIMA $(p, d, q)(P, D, Q)[s]$ （今回は12か月周期なので、s =12）

分析② - SARIMAモデルによる予測

~ SARIMAモデル推定の前に~ ARIMAモデルによる予測

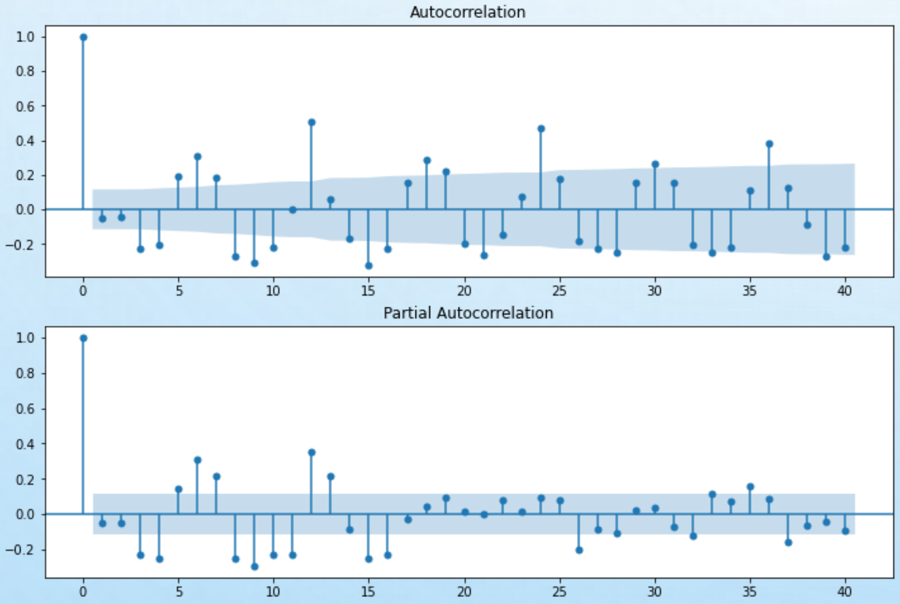
SARIMAモデルでの予測の前にARIMAモデルによる予測をする

（決め打ちで1階差分として推定）

↓

12か月周期で残差に高い自己相関（周期性が残ってしまう）

→周期成分を取り入れたSARIMAモデルで推定




```
#和分過程と推定して差分をとる
diff = train - train.shift()
diff = diff.dropna()

#差分系列への自動ARMA推定関数の実行
resDiff = sm.tsa.arma_order_select_ic(diff, ic='aic', trend='nc')
resDiff
```

```
{'aic':              0            1            2
 0          NaN  1422.668983  1415.722280
 1  1419.770370  1418.153423  1400.924313
 2  1414.754920  1349.156697  1260.857155
 3  1398.873952  1338.290834  1259.967138
 4  1383.885409  1336.066584  1254.218918,
 'aic_min_order': (4, 2)}
```

```
#ARIMAモデルの推定
#P=4, q=2 が最善となったので、それをモデル化
#真ん中の1は1階差分
from statsmodels.tsa.arima_model import ARIMA
ARIMA_4_1_2 = ARIMA(train, order=(4, 1, 2)).fit(dist=False)
ARIMA_4_1_2.params
```

```
const           0.158106
ar.L1.D.co2     1.726728
ar.L2.D.co2    -1.229680
ar.L3.D.co2     0.392536
ar.L4.D.co2    -0.203234
ma.L1.D.co2    -1.834706
ma.L2.D.co2     0.923346
dtype: float64
```

分析② - SARIMAモデルによる予測

手順

1. **モデル選択**
2. モデルの構築
3. モデルを用いて予測
4. 残差の自己相関の可視化
5. 予測結果の可視化

AIC（Akaike Information Criterion：赤池情報量基準）

が最小となるSARIMAの次数を探す

→ order=(3,1,3)、seasonal_order=(1,1,1,12)

でAICが最小

```python
max_p = 3
max_q = 3
max_d = 2
max_sp = 1
max_sq = 1
max_sd = 1

pattern = max_p*(max_d + 1)*(max_q + 1)*(max_sp + 1)*(max_sq + 1)*(max_sd + 1)

modelSelection = pd.DataFrame(index=range(pattern), columns=["model", "aic"])

# 自動SARIMA選択
num = 0

for p in range(1, max_p + 1):
    for d in range(0, max_d + 1):
        for q in range(0, max_q + 1):
            for sp in range(0, max_sp + 1):
                for sd in range(0, max_sd + 1):
                    for sq in range(0, max_sq + 1):
                        sarima = sm.tsa.SARIMAX(
                            train, order=(p,d,q),
                            seasonal_order=(sp,sd,sq,12),
                            enforce_stationarity = False,
                            enforce_invertibility = False
                        ).fit()
                        modelSelection.iloc[num]["model"] = "order=(" + str(p) + ","+
                        modelSelection.iloc[num]["aic"] = sarima.aic
                        num = num + 1
```

```python
# aicが小さい順に並べかえ
modelSelection.sort_values(by='aic').head()
```

	model	aic
255	order=(3,1,3), season=(1,1,1)	853.764576

分析② - SARIMAモデルによる予測

手順

1. モデル選択
2. **モデルの構築**
3. **モデルを用いて予測**
4. 残差の自己相関の可視化
5. 予測結果の可視化

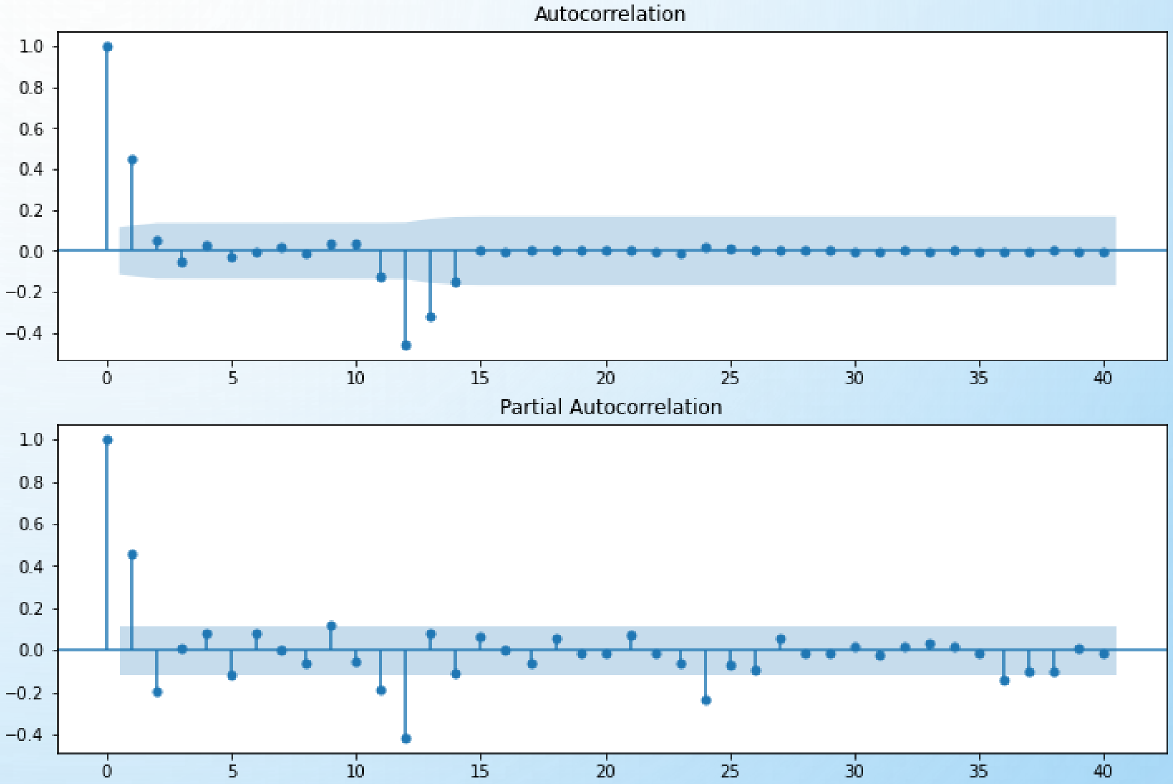
```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                co2   No. Observations:                  291
Model:             SARIMAX(3, 1, 3)x(1, 1, [1], 12)   Log Likelihood                -417.882
Date:                            Sat, 12 Mar 2022   AIC                            853.765
Time:                                    21:07:16   BIC                            885.880
Sample:                                01-01-1987   HQIC                           866.672
                                     - 03-01-2011
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.3797      0.089    -15.484      0.000      -1.554      -1.205
ar.L2         -0.8977      0.116     -7.760      0.000      -1.124      -0.671
ar.L3          0.0054      0.087      0.062      0.951      -0.165       0.176
ma.L1          0.7077      0.077      9.225      0.000       0.557       0.858
ma.L2         -0.1553      0.093     -1.670      0.095      -0.338       0.027
ma.L3         -0.7328      0.080     -9.165      0.000      -0.890      -0.576
ar.S.L12       0.2496      0.065      3.818      0.000       0.121       0.378
ma.S.L12      -1.0000    731.423     -0.001      0.999   -1434.562    1432.562
sigma2         1.2300    899.692      0.001      0.999   -1762.134    1764.594
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                98.22
Prob(Q):                              0.95   Prob(JB):                         0.00
Heteroskedasticity (H):               1.21   Skew:                            -0.04
Prob(H) (two-sided):                  0.37   Kurtosis:                         6.00
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
pred_2 = SARIMA_3_1_3_111.predict('1989-03-01', '2019-12-01')
```

分析② - SARIMAモデルによる予測

手順

1. モデル選択
2. モデルの構築
3. モデルを用いて予測
4. **残差の自己相関の可視化**
5. 予測結果の可視化

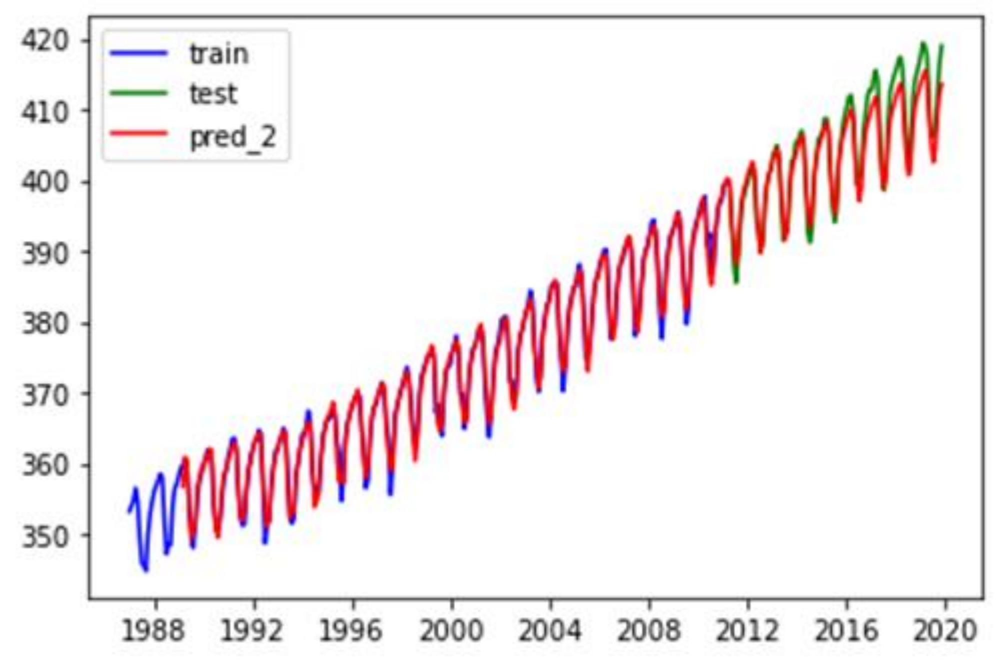


完全ではないが、改善されている

分析② - SARIMAモデルによる予測

手順

1. モデル選択
2. モデルの構築
3. モデルを用いて予測
4. 残差の自己相関の可視化
5. **予測結果の可視化**



分析③ - LSTMモデルによる予測

● **LSTM（Long short-term memory : 長・短期記憶）**

RNNの一種

時系列を考慮する層を改良したもの

RNNが抱えていた勾配消失問題を解消

● **RNN（Recurrent neural network : 回帰型ニューラルネットワーク）**

ニューラルネットワークを拡張

時系列データを扱えるようにしたもの

分析③ - LSTMモデルによる予測

手順

1. 正規化
2. 入力データと教師データの作成
3. モデルの構築
4. 学習
5. モデルを用いて予測
6. 予測結果の可視化

分析③ - LSTMモデルによる予測

手順

1. 正規化
2. 入力データと教師データの作成
3. モデルの構築
4. 学習
5. モデルを用いて予測
6. 予測結果の可視化

機械学習の前処理として、正規化を行う

教師データ：ある月の二酸化炭素濃度

入力データ：教師データの月の、以前3年（36か月）の各月の二酸化炭素濃度

```
#正規化
from sklearn import preprocessing
mm = preprocessing.MinMaxScaler()
train_norm = mm.fit_transform(train)
```

```
# 入力データと教師データの作成
def make_dataset(low_data, maxlen):

    data, target = [], []

    for i in range(len(low_data)-maxlen):
        data.append(low_data[i:i + maxlen])
        target.append(low_data[i + maxlen])

    re_data = np.array(data).reshape(len(data), maxlen, 1)
    re_target = np.array(target).reshape(len(data), 1)

    return re_data, re_target
```

```
# RNNへの入力データ数
window_size = 12*3
data, label = make_dataset(train_norm, window_size)
```

分析③ - LSTMモデルによる予測

手順

1. 正規化
2. 入力データと教師データの作成
3. **モデルの構築**
4. 学習
5. モデルを用いて予測
6. 予測結果の可視化

```
#LSTMの入力の長さ
length_of_sequence = data.shape[1]
#時系列データの単位時間における特徴量の次元数
#今回は単位時間あたり1つのスカラー量なので 1
in_out_neurons = 1
#隠れ層の次元数
n_hidden = 300
```

```
# ネットワークの構築
model = Sequential()
model.add(LSTM(n_hidden, batch_input_shape=(None, length_of_sequence, in_out_neurons), return_sequences=False))
model.add(Dense(in_out_neurons))
model.add(Activation('linear'))
optimizer = Adam(learning_rate=1e-3)

#コンパイル
model.compile(loss="mean_squared_error", optimizer=optimizer)
```

分析③ - LSTMモデルによる予測

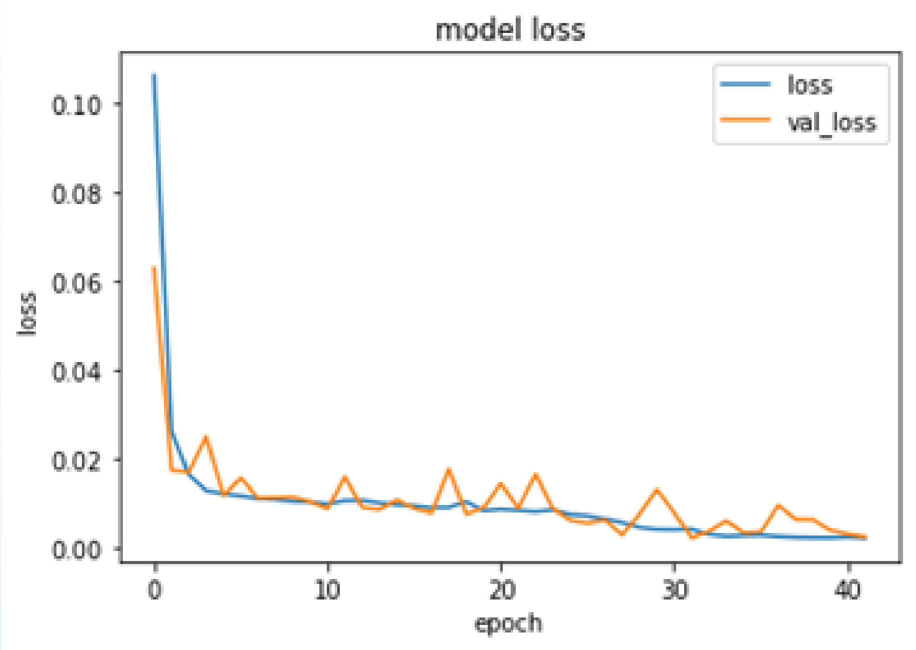
手順

1. 正規化
2. 入力データと教師データの作成
3. モデルの構築
4. **学習**
5. モデルを用いて予測
6. 予測結果の可視化

```
#過学習を防ぐ
#10エポック数の間に val_loss (テストデータに対して計算された損失値) に改善がないと, 学習が停止
early_stopping = EarlyStopping(monitor='val_loss', mode='auto', patience=10)
```

```
#バッチ数は32or64とする
#学習データのうち、10%を検証データ (validation) として使用
hist = model.fit(data, label,
                 batch_size=32, epochs=200,
                 callbacks=[early_stopping],
                 validation_split=0.1)
```

損失値の遷移



分析③ - LSTMモデルによる予測

手順

1. 正規化
2. 入力データと教師データの作成
3. モデルの構築
4. 学習
5. **モデルを用いて予測**
6. 予測結果の可視化

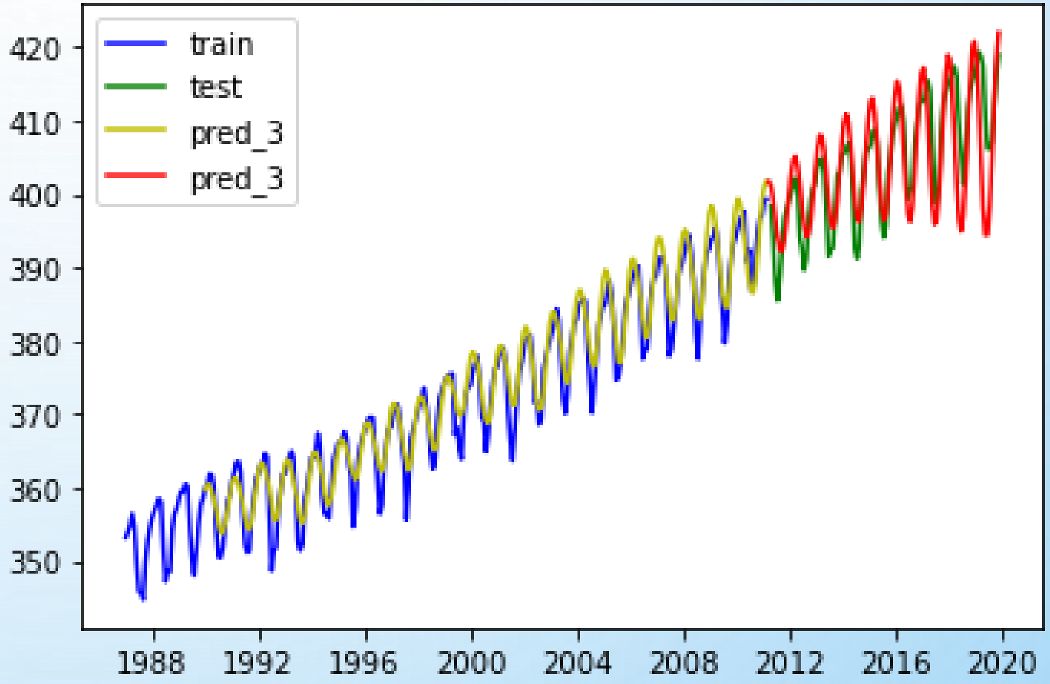
```
#trainの期間の予測し、正規化したので元に戻し、リスト化
train_pred = mm.inverse_transform(model.predict(data)).reshape(1,-1)[0].tolist()
```

```
#未来予測（testの期間の予測）

#dataの最後尾のデータセットを取得
future_test = data[-1].T

#予測結果を保存する配列
future_result = np.empty((0))

#予測期間はtestデータの長さ分
pred_time_length = len(test)

#予測結果をfuture_resultに格納
for step in range(pred_time_length):
    test_data = np.reshape(future_test, (1, window_size, 1))

    #モデルによる予測
    batch_predict = model.predict(test_data)

    #future_testの最初の数値を削除
    future_test = np.delete(future_test, 0)
    #future_testの最後に予測結果を加える
    future_test = np.append(future_test, batch_predict)

    #future_resultに予測結果を加える
    future_result = np.append(future_result, batch_predict)

#正規化したので元に戻し、リスト化
test_pred = mm.inverse_transform(future_result.reshape(1,-1))[0].tolist()
```

分析③ - LSTMモデルによる予測

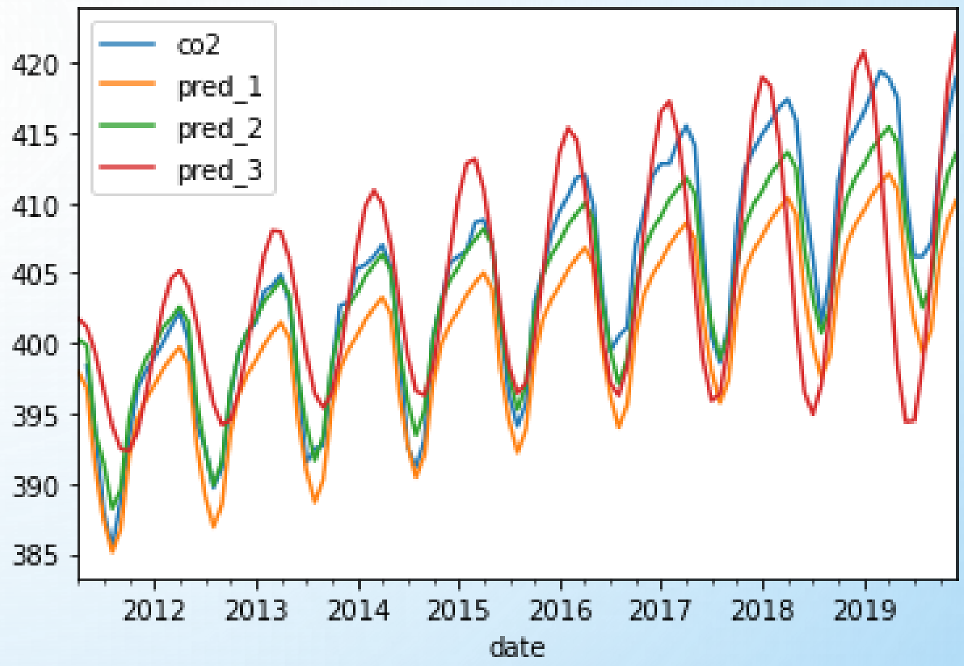
手順

1. 正規化
2. 入力データと教師データの作成
3. モデルの構築
4. 学習
5. モデルを用いて予測
6. **予測結果の可視化**



予測結果の比較

1. **可視化による比較**



2. **MAEによる比較**

MAE（Mean Absolute Error :平均絶対誤差）

$$\frac{1}{N}\sum_{i=1}^{N}|\hat{y_i} - y_i|$$

MAEが小さいほど精度が高い

	分析① 成分分解	分析② SARIMA	分析③ LSTM
MAE	4.27163	1.80376	4.06851
順位	3	1	2

考察

分析① 成分分解	分析② SARIMA	分析③ LSTM
全体的に予測値が実測値より低く、その差は徐々に大きくなっている →近年、二酸化炭素の排出のペースが増加している（予測したトレンドよりも実際は傾きが急になっている）と考えられる。	3つのモデルの中で最も精度の高い結果を得られた →次数を決め打ちせず、総当たりでモデルを選択したことが一因と考えられる。	予測値の周期的な変動の幅が徐々に大きくなっている →ハイパーパラメータの設定を改善すれば、より高精度の予測が可能だと考えられる。