時系列データの初歩

1.はじめに

時系列データの取扱いの基本をまとめました。機械学習の範疇になります。

機械学習とは「機械が自動で学習する」ということで、データからパターンを見出すとうことが特長です。1 次データを入手した後、

データ観察→データの前処理→モデリング→モデルの評価　と、進めます。

1 次データは、放電加工機のメッセージ履歴のデータですが、データの内容に拘ることはなく、ライブラリ、コードを中心に、まとめます。（モデリングの良否は問わないでください）　参考にした WEB は以下です。

ごちきか　https://gochikika.ntt.com/index.html

2.データ観察

2.1 データ準備

通常、データは csv で入手され、pandas または numpy に変換されます。

Pandas の説明

Python において、データ解析を支援する機能を提供するライブラリです。特に、数表および時系列データを操作するためのデータ構造と演算を提供します。

使い方は、エクセルに似ています。（エクセルより複雑なことが出来そうです）

Pandas のコード紹介

```
data=pd.read_csv("data_d_2022.csv")
```

#data_d_2022.csv は、右下のような、放電加工機の解析に使用している内容です。

```
print(data.head())　#上から 5 番目mでのデータを表示します。
```

```
        date     point
0   2022/4/1  1.210145
1   2022/4/2  1.154762
2   2022/4/3  1.448819
3   2022/4/4  1.415301
4   2022/4/5  1.085106
```

	A	B
1	date	point
2	2022/4/1	1.210145
3	2022/4/2	1.154762
4	2022/4/3	1.448819
5	2022/4/4	1.415301
6	2022/4/5	1.085106
365	2023/3/30	0.888889
366	2023/3/31	1.098361

data=pd.read_csv("data_d_2022.csv",index_col=0,parse_dates=True)

#インデックスを 0 行目(A 列)にして、datetime 型にしています。

print(data.head())　以下の表示となります。

	point
date	
2022-04-01	1.210145
2022-04-02	1.154762
2022-04-03	1.448819
2022-04-04	1.415301
2022-04-05	1.085106

Numpy の説明

Python において数値計算を効率的に行うための拡張モジュールです。多次元配列（例えばベクトルや行列などを表現できる）のサポートを行います。高速な計算を可能にします。NumPy は Python の標準機能であるリスト型変数ではなく、ndarray 型という特別な配列を使用します。特徴として　同じ型を持つ要素しか格納することができません。各次元ごとの（2 次元なら列ごとや行ごと）の要素数は必ず一定となります。

Numpy のコード紹介

data = np.loadtxt("data_d_2022fornumpy.csv",encoding="utf-8_sig")

#csv のデータを Numpy で取込みます。パラメータは色々あります。

Utf-8_sig は、MS のコードは CP932 ですので、変換してます。

np.set_printoptions(threshold=0) #表示する個数を減らしています。

print(data) #以下のように、表示します。

[1.21014493 1.1547619 1.4488189 ... 1.18181818 0.88888889 1.09836066]

元となる c SV データ（data_d_2022fornumpy.csv）は以下の一次元データです。

	A	B
1	1.210145	
2	1.154762	
3	1.448819	
4	1.415301	
5	1.085106	
	⋮	
364	0.888889	
365	1.098361	

2.2　データの可視化　（グラフで確認しましょう）

seaborn というデータ可視化システムもあり、洗練された図を描けて、コードも簡単です。ただ、ユーザーは、Matplotlib の方が多いようです。

ここでは、ライブラリの　Matplotlib　を紹介します。二通りの方法があります。

・pyplot インターフェース　1 枚のグラフを手早く作成するとき有効です。

以下のコードで作成します。

```
plt.figure()
plt.plot(x,y)
plt.show()
```

・オブジェクト指向インターフェース

Plt.subplots()　や　.add_subplot()　を用いて作成します。

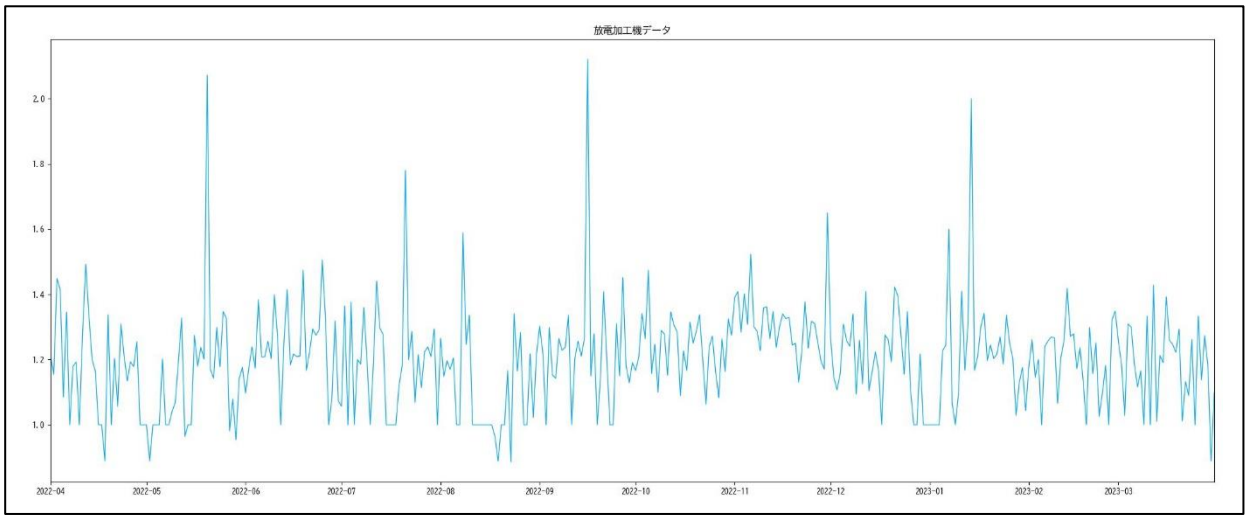
複数グラフ、2 軸グラフの作成、細かいレイアウトの調整が必要なとき有効です。

1.1 で準備したデータを用いて、作成します。コードは以下です。

```
x = data.index　　# x 軸: 時刻　y 軸: ポイントと明示します。
y = data.iloc[:, 0]　# データの一列目. 「:」で行すべて選択、「0」で 1 列目を選択
#グラフの作成コードです。
from matplotlib.dates import DateFormatter
　# matplotlib で日付のフォーマットを取り扱うモジュールを追加で読み込む
plt.rcParams['font.family'] = 'BIZ UDGothic' # Windows
　#日本語対応フォントを読み込み
fig = plt.figure(figsize=(7, 4), dpi=100)
　# figure の引数; figsize で画像のサイズ、dpi で解像度を設定できます。
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y, color='#02A8F3', linewidth=1.0)
```

```
    # 線の色や、太さを指定
ax.set_xlim(x[0], x[-1])
    # プロットをグラフの限界まで広げる（余計な空白を作らない）
ax.set_title('放電加工機データ')
        # タイトルをつける
plt.show()
```

作成したグラフは以下です。



2.3 基本的な要約統計量の計算 （データを数値で評価しましょう）

データの性質を観察するため、要約統計量を計算します。コードを紹介します。

print(data.describe()) #カウント数、平均、標準偏差(不偏分散の平方根)、
最小値第一四分位点、中央値、第三四分位点、最大値 を表示します。

	point
count	365.000000
mean	1.200761
std	0.159735
min	0.886364
25%	1.095745
50%	1.203125
75%	1.283951
max	2.121212

print(data.mean()) #平均

point	1.200761

print(data.var()) #分散

point	0.025515

print(data.std()) #標準偏差

point	0.159735

print(data.skew()) #歪度（平均を中心とした分布の非対称性の程度、
正の値：平均より大きい向きに偏る）

point	1.375758

print(data.kurt()) #尖度（分布の尖り具合、 3 より大きいと正規分布より尖る）

point	6.269576

print(data.min()) #最小値

point	0.886364

print(data.max()) #最大値

point	2.121212

print(data.median()) #中央値

point	1.203125

2.4　自己相関関数と相互相関関数（時系列データに特有の指標となります）

時間方向に対する類似性を評価します。モデル選択、特徴量設計に有用です。

2.4.1 自己相関関数と偏自己相関関数

自己相関関数は、

$$\frac{E[(x(t)-\mu)(x(t-k)-\mu)]}{\sigma 2}$$　の式で表します。

k はラグで　x(t-k) とは　k だけ時間をずらしたデータです。

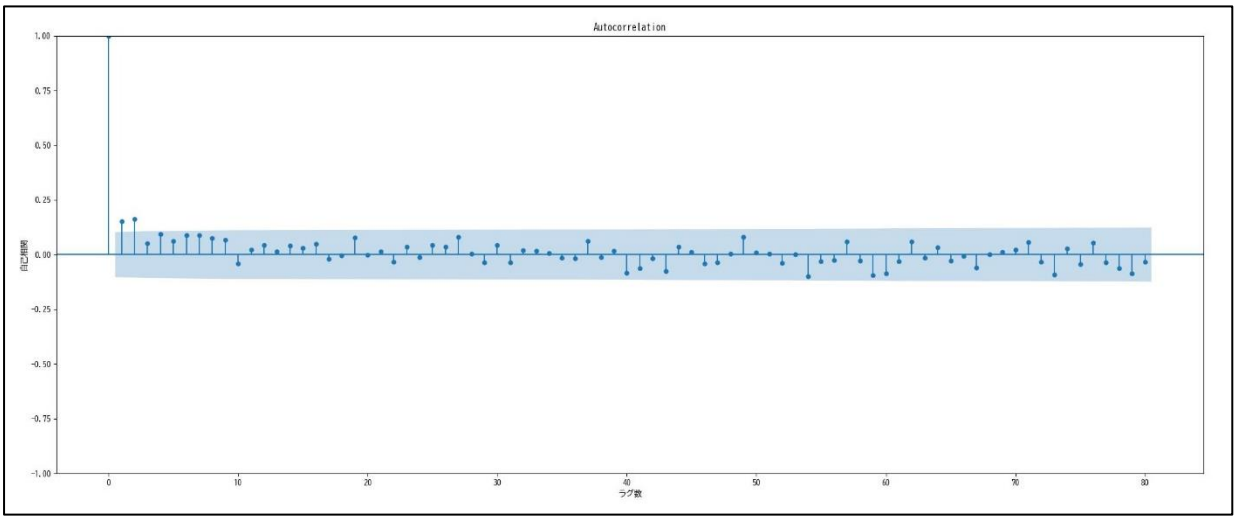
μ、σ2　は、x(t)　の　平均と分散です。x(t)　は定常であるとします。

分子は自己共分散　であり、分母は、関数が[-1,1]になるよう、正規化します。

E は期待値で、加重平均をとります。

コードを紹介します。ライブリ　statsmodels で自動計算します。

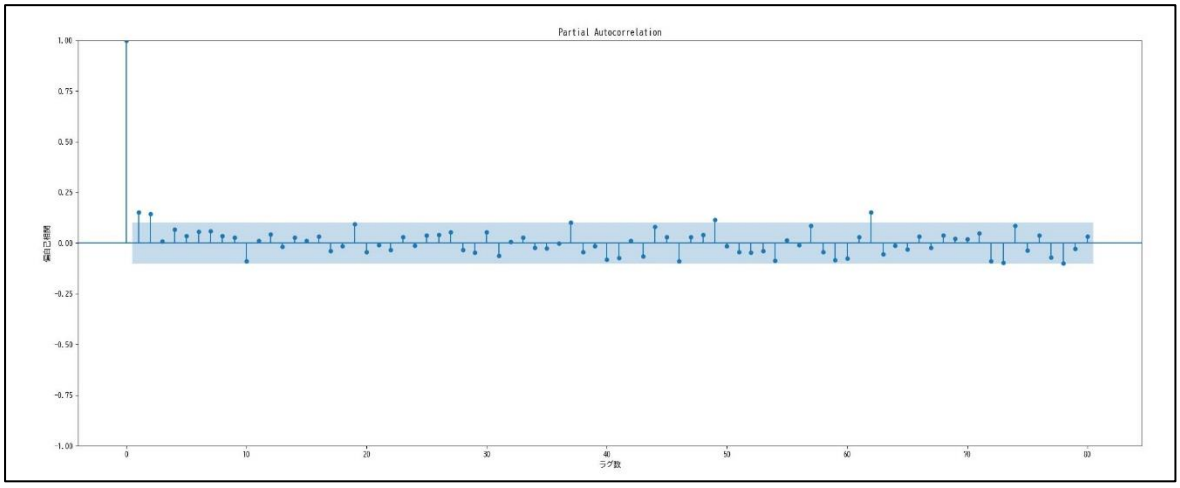
```
import statsmodels.api as sm　#ライブラリのインポート
fig = plt.figure(figsize=(8, 5))
ax1 = fig.add_subplot(111)
sm.graphics.tsa.plot_acf(y, lags=80, ax=ax1)　#グラフを自動作成
ax1.set_xlabel('ラグ数')
ax1.set_ylabel('自己相関')
plt.show()
```



偏自己相関関数は、ある時刻間だけの相関を計算します。

コードを紹介します。ライブリ　statsmodels で自動計算します。

```
fig = plt.figure(figsize=(8, 5))
ax1 = fig.add_subplot(111)
sm.graphics.tsa.plot_pacf(y, lags=80, ax=ax1) #グラフを自動作成
ax1.set_xlabel('ラグ数')
ax1.set_ylabel('偏自己相関')
plt.show()
```



2.4.2 相互相関関数

2 種類の時系列データ間の相関を見る時に利用します。

今回は、1 種類のデータですので、検証はできません。

statsmodels に実装されており、ccf() という関数があります。

2.5　定常性の確認　（データが定常性であることの確認を行います）

2.5.1 定常性とは（時系列データの解析で、時に依存してるか否かの確認は重要です）

定常性とは、データの従う確率過程に関する性質であり、時間不変性（=時間に依存せずどの時点でも性質が同じ）の度合いを示します。強定常性と弱定常性があります。

強定常性： 確率過程の有限次元分布が時間シフトに関して不変であること。

弱定常性: 確率過程の平均・自己共分散が有限で、時間シフトに関して不変であること。

つまり、任意の時において、期待値 E と自己共分散 Cov が以下を満たすときです。

$$\mathrm{E}[x_t] = \mu$$

$$\mathrm{Cov}[x_t, x_{t+k}] = \mathrm{E}[(x_t - \mu)(x_{t+k} - \mu)] = \gamma_k$$

時系列解析では、単に定常性と言った場合には弱定常性を指すことが多いです。

2.5.2 定常性の確認方法

プロットしてグラフ確認

横一線で、ある程度の幅で振動するのノイズのように見えます。トレンドが含まれると非定常過程となります。

ヒストグラムによる確認

適当な時で、データを分割して、ヒストグラムに差があれば、非定常過程となります。

統計的検定による確認

拡張 Dickey-Fuller 検定（ADF 検定）を利用します。

コードを紹介します。ライブリ　statsmodels で自動計算します。

```
import statsmodels.tsa.api as tsa
adf_rlt_pv = tsa.adfuller(y.values)
print(f'ADF statistics: {adf_rlt_pv[0]}')
print(f'# of lags used: {adf_rlt_pv[2]}')
print(f'Critical values: {adf_rlt_pv[4]}')
```

以下が出力です。

```
ADF statistics: -10.629988477502625
# of lags used: 1
Critical values:
{'1%': -3.448493650810824,
 '5%': -2.8695352280356556,
 '10%': -2.5710293341377715}
```

検定推定量は、-10.629988477502625 で、5%-棄却域（-2.8695352280356556）に、入ります。（数字が小さい）　単位根が存在しない、定常と判定できます。

3 データの前処理　（データの不具合を是正する方法です）

3.1 欠損値処理

収集したデータから欠損値（missing value） が生じる場合があります。線形回帰モデル、決定木、ニューラルネットワークなど、多くのモデルは欠損値があるデータを直接扱えません。時系列モデリングにおいても、サンプリング間隔が一定であることを仮定している場合が多く、欠損値のある時刻のデータを除外することもできません。したがって、データの前処理の段階で欠損値を確認し、何らかの方法で補完する必要があります。

欠損値を探して、直前の値を入れるという関数があるので、コードを紹介します。

```
def my_typecheck(x)　#数値に変換できるかできないかを判定する関数
  try:
      x = float(x)　#判定する x を float 型（小数点表示）表示にします
  except:
      x = x　　# x と同じなら数字なので、例外となります。
  return type(x)　#同じ出ない時、型を戻り値にします。str になる
def drop_str(df, method='ffill'):　#str を落として補完する関数
  df = df.mask(df.applymap(my_typecheck)==str, np.nan)
  # mask は　pandas の関数で、条件設定（str の時）で　欠損値(np.nan)に入替ます。#applymap は　関数(my_typecheck)を指示しています。
  df = df.fillna(method=method).astype(float)
    # fillna は pandas の欠損値補間の関数で、
    #method=’ffill’ は直前の値を np.nan に補完します。
 return df
df = drop_str(df) #定義した関数で処理を実行
```

3.2 正規化

正規化とは、数値の取り得る範囲を揃える変換処理のことです。

複数の列の値を利用する時、列ごとの値のスケールが大きく異なると、問題が発生するので、正規化という手段を講じます。二つの方法を紹介します。

平均 0, 分散 1 に変換する正規化

$\text{StandardScaler(x)} = \frac{x-mean(x)}{std(x)}$　という数式になります。

コードを紹介します。　ライブリ sklearn　で計算します。

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()  # 正規化を行うオブジェクトを生成
data_normalized_ss=pd.DataFrame(ss.fit_transform(data),
columns=data.column)
    # fit_transform 関数は、fit 関数（正規化するための前準備の計算）と
    # transform 関数（準備された情報から正規化の変換処理を行う）の両方を行う
print(data_normalized_ss.head())
```

以下が出力です。

	point
0	0.058830
1	-0.288363
2	1.555063
3	1.344939
4	-0.725030

最小値 0,最大値 1 に変換する正規化

$\text{MinMaxScaler(x)} = \frac{x-mean(x)}{std(x)}$　という数式になります。

コードを紹介します。　ライブリ sklearn　で計算します。

```
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler() # 正規化を行うオブジェクトを生成
data_normalized_mm=pd.DataFrame(mm.fit_transform(data),
columns=data.columns)
```

```
print(data_normalized_mm.head())
```

以下が出力です。

```
      point
0  0.262203
1  0.217353
2  0.455485
3  0.428342
4  0.160945
```

3.3 外れ値

外れ値とは、他の値と比べて、大きく外れた値のことです。

どれくらいの大きさだと外れ値とみなすのかというと、正規分布に従うような変数であれば、残差が標準偏差の 2 倍から 3 倍以上とすることが多いです。

確率的には、残差が標準偏差の 2 倍以上とするなら 4.6%程度、3 倍以上とするなら 0.3%程度の出現率ということになるようです。

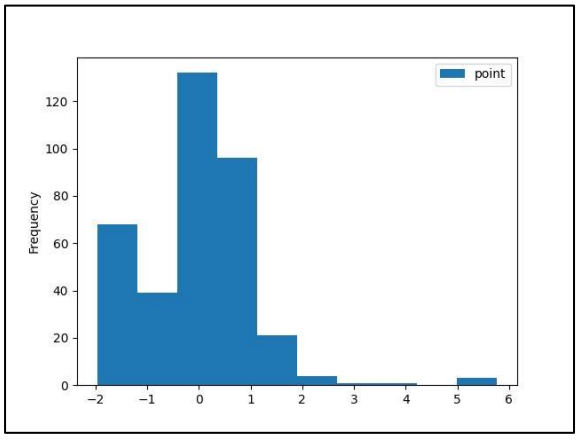
基本統計量に基づいた外れ値処理

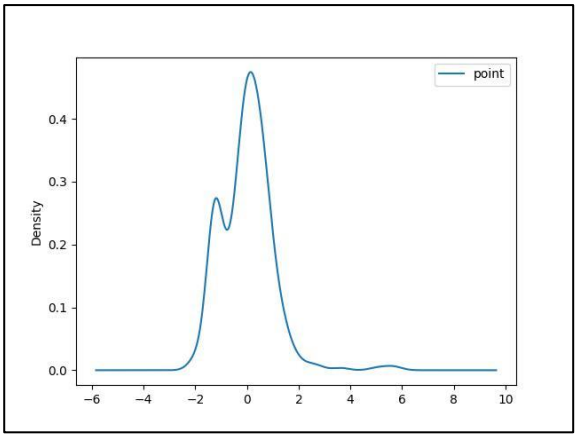
分布の可視化に、ヒストグラムとカーネル密度推定（ノンパラメトリック推定であり、正規分布とかの型を規定せず、データに沿う滑らかな線を作成します）で確認します。

コードを紹介します。　データは平均０，分散１　で正規化してます。

```
data_normalized_ss.plot.hist()      #ヒストグラム
data_normalized_ss.plot.kde()       #カーネル密度推定
plt.show()
```

出力は以下です。





3 .4 観測が時間的に等間隔になっていない場合の処理

時系列データの解析は、均等な時間間隔である必要があります。非等間隔なデータを、等間隔にする方法を、提示します。

データの抜けだけが存在する場合（間隔は分かる）

インデックを等間隔に作成して、抜けの部分に欠損値 NaN を入れます。

関数 pd.date_range()　で作成します。

```
idx_new = pd.date_range(start=df.index[0],end=df.index[-1],freq='S')
  #freq='S' は毎秒単位の意味です。毎時、毎分、毎日、毎月等　指定できます。
  #start は開始日（欠損あるデータ df の 0 番目）
  #end は最後日（欠損あるデータ df の最終番目）
df_new = pd.DataFrame(index = idx_new, columns=df.columns)
  #index は idx_new で、columns は　df と同じ内容のデータを作成します。
df_new.loc[df.index, :] = df
  #df のデータを、df_new に入れます。これで作成完です。
```

データの抜けだけが存在する場合（間隔は不明）

時間間隔の統計量を算出して、間隔を特定します。

```
intervals = pd.Series(df.index).diff().dropna().astype('category')
  #pd.Series（一次元配列（ベクトル））に変換します。diff() を使うためです。
    （pd.DataFrame は 2 次元配列（行列）を扱います）
# diff()　1 行前との差分　dropna()　欠損値を除外します。
  #astype('category') カテゴリ変数化します。
print(intervals.unique(), "¥n")      #時間間隔のユニーク数を表示します。
print(intervals.value_counts())  #間隔の度数分布を表示します。
```

2 個の表示から、時間間隔を、推定します。

3.5 次元削減

列の要素に変更は加えず、個々の列（軸）を削減することで低次元化を達成する手法が「次元削減」です。1 次元のデータであれば、削減の必要はありませんが、多次元のデータを扱うとき、削減した方が、より適切な結果が得られることがあります。

変動が少ない列を除外する

変動の少なさの評価として、平均 0、分散 1 で正規化します。16%点（パーセンタイル）84%点（パーセンタイル）を算出します。標準正規分布では、16%点は-1 であり、84%点は+1 であるので、計算した列の値が、-1 より大きく、1 より小さいと、分布の峰は急峻となり、変動はすくないと判断できます。

パーセンタイルの計算は、以下の関数で計算できます。

```
df.qunantile([0.16,0.84])
```

多重共通性の回避

多重共線性が問題となることがあります。多重共線性とは「説明変数のある変数同士が強く相関している状態が複数起きている状態」と定義されますが、重回帰分析を適用することが難しく、目的変数に対して有意に影響を与える変数が増える、あるいは見逃す可能性があるので、データ分析においては対応しなければならない要素です。

対応方法として、二つの「相関関係が高いと考えられる説明変数を外す」方法があります。

1，相関係数の絶対値が 0.9 以上の組合せをまとめて、その中の一つ説明変数にまとめる。

2，分散拡大係数(VIF) を指標とする。

VIF とは最小二乗回帰分析における多重共線性の深刻さを評価する指標で、推定された回帰係数の分散が多重共線性のためにどれだけ増加したかを定量的に表現します。

これにより独立変数間の多重共線性を検出し、値が大きい場合はその変数を分析から除いた方がよいと考えられます。一般的に使用される閾値は 10 であり、この閾値以上の列は分析から除かれることとなります。

コードを紹介します。閾値が 10 以下になるまで、続けます。

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
    #ライブラリから関数を呼びます。
vif = pd.DataFrame()
vif["VIF Factor"] =  ¥              #vif を計算する
        = [variance_inflation_factor(df.values, i) for i in range(df.shape[1])]
vif.index = df.columns
vif
```

４. モデリング　（ここでモデリングができます）

基本となる回帰モデルから、取り組みます。

４.1　自己回帰モデル

一変量の時系列　$y = (y_0 , y_1 , \cdots) = (y_t)_{t=0,1\ldots}$ において、

t-1 までの値　$(y_0 ,\ldots, y_{t-1})$ を用いて 1 ステップ先の値、y_t を推測します。

統計的な枠組みでこの課題に取り組みます。即ち、時系列の従う統計的モデルを仮定し、データを用いてその下での統計的推測を行い y_tの値を一般の統計的な意味で予測する（predict）か、時系列解析的な意味で予測する（forecast）方法について考えます。

数式として表します。直前 p ステップの時刻での値に線形で依存すると仮定します。

$$y_t = c + \sum_{\gamma}^{p} \Phi_\gamma \, y_{t-\gamma} + \varepsilon_t$$

ε_t　は、ホワイトノイズであり、p 次の自己回帰モデル　AR(p)　と呼びます。

４.2 係数の決定

次数 p は、既知とします。（時間間隔を考慮して決めます。 1 日間隔であり、1 週間単位での変化が予測されるなら　p=7 とかです）

(c, $\Phi_1 \ldots \Phi_p$) がパラメータであり、最小二乗法(OLS)で,損失 L(c, $\Phi_1 \cdots . \Phi_p$)を最小化するパラメータを求めます。

L(c, $\Phi_1 \cdots . \Phi_p$) = $\sum_t (y_t$ -c - $\sum_{\gamma=1}^{p} \Phi_\gamma \, y_{t-\gamma}$ $)^2$

係数の決定と作成できたグラフを、コードと共に紹介します。

ライブライは、Scikit-learn　と　statsmodels のふたつで計算してます。

P= 7 とします。

Scikit-learn で計算

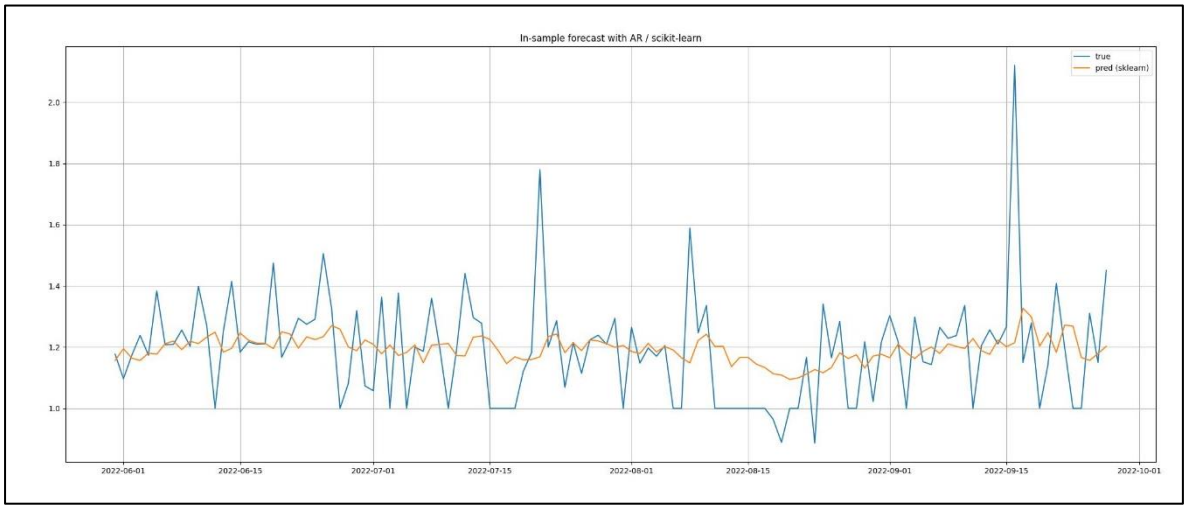
```
data=pd.read_csv("data_d_2022.csv")
data=pd.read_csv("data_d_2022.csv",index_col=0,parse_dates=True)
x = data.index
y = data.iloc[:, 0]
y_lagged_index_ar = np.arange(p)[::-1].reshape(1, -1) + np.arange(y.shape[0]＼
                                                    - p).reshape(-1, 1)
y_lagged_ar = y.values[y_lagged_index_ar]  # 説明変数(観察値)
y_target_ar = y.values[p:].reshape(-1, 1)  # 目的変数(予測値)
skl_ar = LinearRegression()
skl_ar.fit(y_lagged_ar, y_target_ar)   #OLS で係数を決定するコード
print(f'OLS fitting of the AR({p}) model with scikit-learn::')
print(f'* c: {skl_ar.intercept_}.')      #切片の値です。
print(f'* phi: {skl_ar.coef_}.')         #係数の値です。
#出力は以下です。
```

```
OLS fitting of the AR(7) model with scikit-learn::
* c: [0.6805107].
*phi: [[ 0.12681822  0.12608896 -0.00815636  0.06258685  0.01727127
0.04753413 0.06046577]].
```

```
#グラフ作成します。
y_target_skl_ar_pred = skl_ar.predict(y_lagged_ar)
_fig, _ax = plt.subplots(1, 1, figsize=(10, 5))
_begin = 60      #インデックス 60 から
_window = 120   #120 の間で計算します。
```

```
_lag= p      #7 にしました。
_ax.set_title('In-sample forecast with AR / scikit-learn')
_ax.plot(
    y.index[_begin:(_begin + _window)],
    y_target_ar[(_begin - _lag):(_begin - _lag + _window)],
    label='true',
)
_ax.plot(
    y.index[_begin:(_begin + _window)],
    y_target_skl_ar_pred[(_begin - _lag):(_begin - _lag + _window)],
    label='pred (sklearn)',
)
_ax.grid()
_ax.legend()
plt.show()
```

＃グラフは以下です。



statamodels で計算

```
sm_ar = tsa.AutoReg(endog=y.values, lags=p)
sm_ar_rlt = sm_ar.fit()  #OLS で係数を決定するコード
print(f'Conditional MLE fitting of the AR({p}) model with statsmodels::')
print(f'* c: {sm_ar_rlt.params[0]}.')
print(f'* phi: {sm_ar_rlt.params[1:]}.'
```

#出力は以下です。

```
Conditional MLE fitting of the AR(7) model with statsmodels::
* c: 0.6805107043850123.
* phi: [ 0.12681822  0.12608896 -0.00815636  0.06258685  0.01727127
0.04753413 0.06046577].
```

```
y_sm_ar_pred = sm_ar_rlt.predict()  # 引数なしで呼ぶと in-sample 予測を実行
_fig, _ax = plt.subplots(1, 1, figsize=(10, 5))
_begin = 60
_window = 120
_ax.set_title('In-sample forecast with AR / statsmodels')
_ax.plot(
    y.index[_begin:(_begin + _window)],
    sm_ar.endog.flatten()[_begin:(_begin + _window)],
    label='true',
)
_ax.plot(
    y.index[_begin:(_begin + _window)],
    y_sm_ar_pred[(_begin - p):(_begin + _window - p)],
    label='pred (statsmodels)',
```

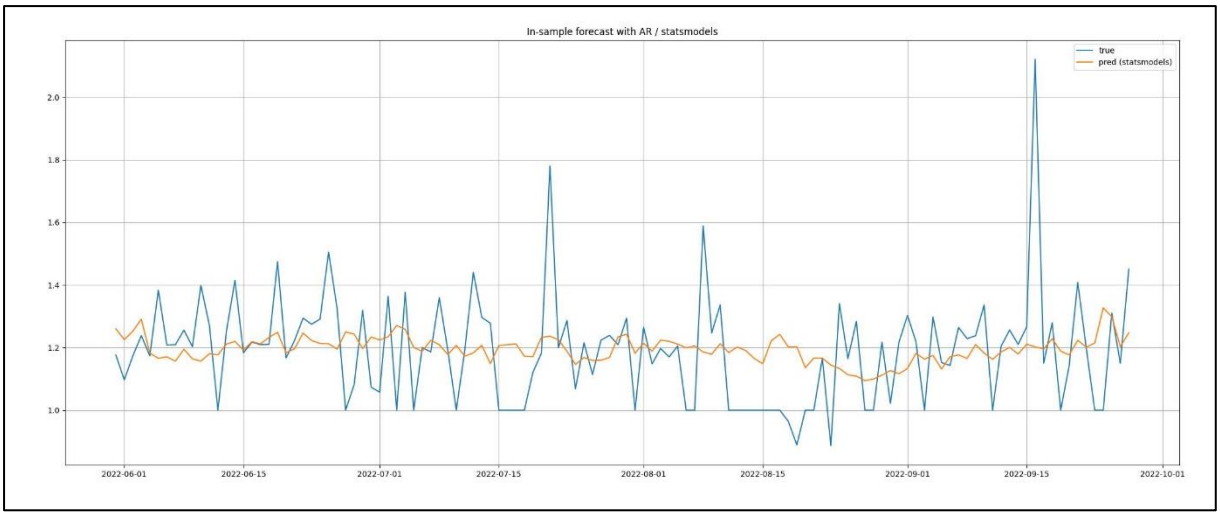
)

_ax.grid()

_ax.legend()

plt.show()

#グラフ作成します。



今まで放電加工機のデータをモデル化して検討してますが、自己回帰モデルにおいては、学習データの推定時において、実データと予測データの乖離があります。実データは、ある確率分布に従う真のデータの確率変数の実現値ですので、推定データは、確率分布の期待値（平均値）と考えれば、当然かもしれません。

状態空間モデルのカルマンフィルタは逐次推定で、推定値は学習データから外れなかったです。

５.モデルの評価

学習したモデルが学習に用いたデータにどれほど当てはまっているか、比較するための数値的な指標が必要になります。こうした指標は複数のモデルやハイパーパラメータの比較に利用できるのみならず、採用したモデルと学習の方法によっては、そのモデルが未知のデータに対してどれほどの不確かさで以て推論できるかを考察するときの参考にすることもできます。これら評価指標の選択に絶対的な基準はなく、分析の目的と理論上の要請に合わせて適切なものを選択して利用する必要があります。

指標を紹介します。

MSE

平均二乗誤差（mean squared error; MSE）とは

$$\mathrm{MSE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

で与えられる値のことです。二乗平均誤差とも呼ばれます。MSE の最小値は 0 で、値が 0 に近いほどモデルとデータがよく当てはまっていることを示します。

RMSE

二乗平均平方根誤差（root mean squared error; RMSE）とは

$$\mathrm{RMSE}(y,\hat{y}) = \sqrt{MSE(y,\hat{y})}$$

で与えられる値のことで、MSE の平方根をとったものです。従って RMSE の最小値も 0 で、値が 0 に近いほどモデルとデータがよく当てはまっていることを示します。

MAE

平均絶対誤差（mean absolute error; MAE）とは

$$\mathrm{MAE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

で与えられる値のことです。MAE の最小値は 0 で、値が 0 に近いほどモデルとデータがよく当てはまっていることを示します。

MAPE

平均絶対百分率誤差（mean absolute percentage error; MAPE）とは

$$\mathrm{MAPE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

で与えられる値のことです。MAPE の最小値は 0 で、値が 0 に近いほどモデルとデータがよく当てはまっていることを示します。

MAPE は「予測した値が真の値から平均して何パーセントずれているか」を表す値として自然に解釈できます。即ち、今回の例ではモデルの予測した値が真の値から平均して約 9.24%ずれていることになります。

R2

決定係数（coefficient of determination）にはいくつかの定義がありますが、一般的には

$$R^2 = 1 - \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \ / \ \sum_{i=1}^{n}(y_i - \bar{y})^2 \ , \ \bar{y} = \sum_{i=1}^{n} y_i$$

で与えられる値を指します。

これまでに紹介した誤差の指標と異なり、R2 に限っては値が大きいほど（最大値が 1 なので、1 に近いほど）モデルとデータがよく当てはまっていることを示します。

コードを以下に紹介します。

```
from sklearn.metrics import (
    mean_squared_error,  # MSE
    mean_absolute_error,  # MAE
    r2_score, # R2
)
mse_skl = mean_squared_error(y_target_ar, y_target_skl_ar_pred)
rmse_skl = np.sqrt(mse_skl)
mae_skl = mean_absolute_error(y_target_ar, y_target_skl_ar_pred)
mape_skl = 100. * mean_absolute_error(np.ones_like(y_target_ar),
```

```
y_target_skl_ar_pred / y_target_ar)
r2_skl = r2_score(y_target_ar, y_target_skl_ar_pred)
print(f'MSE (sklearn): {mse_skl}.')
print(f'RMSE (sklearn): {rmse_skl}.')
print(f'MAE (sklearn): {mae_skl}.')
print(f'MAPE (sklearn): {mape_skl}.')
print(f'R2 (sklearn): {r2_skl}.')
```

出力は以下です。

```
MSE (sklearn): 0.02396434803197567.
RMSE (sklearn): 0.15480422485182913.
MAE (sklearn): 0.11054141257318728.
MAPE (sklearn): 9.24819331056724.
R2 (sklearn): 0.05754256303655203.
```

６．終わりに、

データの扱いについて、まとめました。モデルリングの項で、回帰モデルしか紹介してません。実は、放電加工機のメッセージデータについては、予測がうまくいかなく、以下の色々なモデルに取組みました。

・回帰モデル

・状態空間モデル

・非線形非ガウスモデル（粒子フィルター）

・回帰モデル（AR モデル、ARIMA モデル、SARIMA モデル）

予測は向上していますし、他にも、各種のモデルは色々あります。

扱うデータによって最適なモデルがあるので、モデリングは、一番理解しずらいところかと思います。