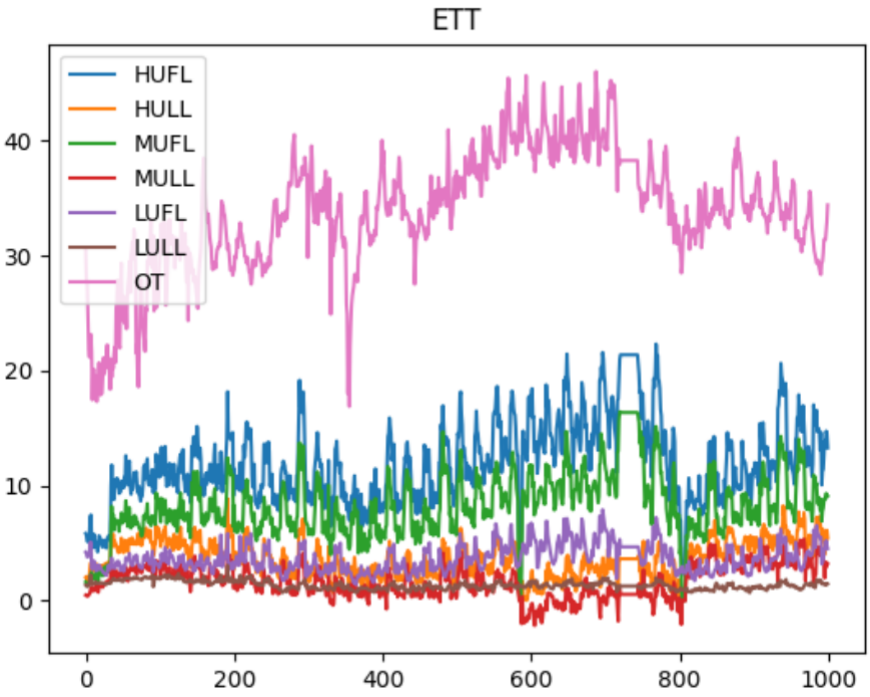
hw1实验报告

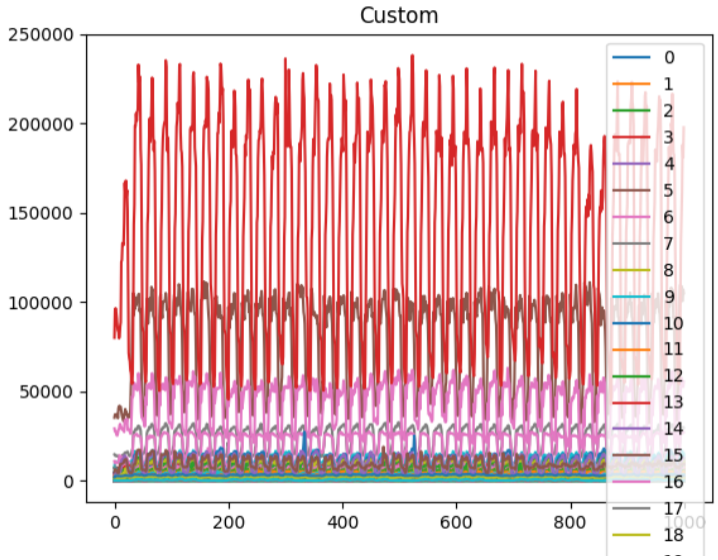
201300086史浩男

一、数据集

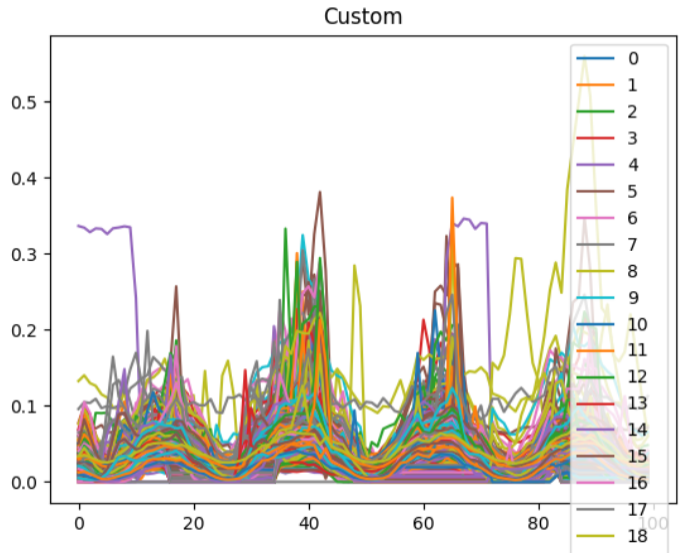
可视化1000个数据点：



`electricity.csv`



`traffic.csv`



二、转换

代码中，如果同一个transform被多次使用，注意是否更新存储的值，可以添加一个update参数用于记录

```python
def transform(self, data, update=False):
    if update:
        self.mean=data.mean()
        self.std=data.std()
    #先检测数据的标准差是否为0
    if self.std==0:
        return data-self.mean
    #将数据标准化到0，1之间
    norm_data=(data-self.mean)/self.std
    return norm_data
```

仅在训练集上update，test_X则不update：

```python
def train(self):
    train_X = self.dataset.train_data
    t_X = self.transform.transform(train_X,update=True)
    self.model.fit(t_X)
```

```python
test_X = self.transform.transform(test_X)
fore = self.model.forecast(test_X, pred_len=pred_len)
# test_Y = self.transform.transform(test_Y)
fore = self.transform.inverse_transform(fore)
```

数学公式

1. **归一化 (Normalization)**
   - **变换 (Transform):**
     - 如果 `max-min` 不为0，变换公式为:$\text{norm\_data} = \frac{\text{data}-\min}{\max-\min}$
     - 如果 `max-min` 为0，则返回 `data - mean`。
   - **逆变换 (Inverse Transform):**
     - 如果 `data` 的极差不为0，逆变换公式为:
       $\text{inverse\_data} = \text{data} \times (\max - \min) + \min$
     - 如果 `data` 的极差为0，则返回 `data * mean`。
2. **标准化 (Standardization)**
   - **变换 (Transform):**
     - 如果 `std` 不为0，变换公式为:$\text{norm\_data} = \frac{\text{data}-\text{mean}}{\text{std}}$
     - 如果 `std` 为0，则返回 `data - mean`。

- **逆变换 (Inverse Transform):**
  - 如果 `data` 的标准差不为0，逆变换公式为:$\text{inverse\_data} = \text{data} \times \text{std} + \text{mean}$
  - 如果 `data` 的标准差为0，则返回 `data + mean`。

3. **均值归一化 (Mean Normalization)**
   - **变换 (Transform):**
     - 如果 `max-min` 不为0，变换公式为:$\text{norm\_data} = \frac{\text{data} - \text{mean}}{\text{max} - \text{min}}$
     - 如果 `max-min` 为0，则返回 `data - mean`。
   - **逆变换 (Inverse Transform):**
     - 如果 `data` 的极差不为0，逆变换公式为:
       $\text{inverse\_data} = \text{data} \times (\text{max} - \text{min}) + \text{mean}$
     - 如果 `data` 的极差为0，则返回 `data * mean`。
4. **Box-Cox 变换 (BoxCox Transform)**
   - **变换 (Transform):**
     - 数据首先转化为正数，然后应用 Box-Cox 变换，公式为:
       $\text{norm\_data} = \text{boxcox}(\text{data}, \text{lam})$
   - **逆变换 (Inverse Transform):**
     - 应用 Box-Cox 的逆变换，公式为:$\text{inverse\_data} = \text{inv\_boxcox}(\text{data}, \text{lam})$

其他Scaler

除了手动实现的这些，sklearn中还提供了可以直接调用的常用Scaler：

1. **StandardScaler**:
   对于每个特征 x，StandardScaler 首先计算特征的均值 μ 和标准差 σ，然后应用以下公式：
   $x_{\text{scaled}} = \frac{x-\mu}{\sigma}$
2. **MinMaxScaler**:
   MinMaxScaler 将每个特征的值缩放到指定的范围内（通常是 0 到 1）。对于每个特征 x，应用以下公式：
   $x_{\text{scaled}} = \frac{x-x_{\min}}{x_{\max}-x_{\min}}$
   其中，$x_{\min}$ 和 $x_{\max}$ 分别是特征 x 的最小值和最大值。
3. **RobustScaler**:
   RobustScaler 使用中位数和四分位数范围对数据进行缩放，以减少离群值的影响。对于每个特征 x，应用以下公式：
   $x_{\text{scaled}} = \frac{x-Q_1(x)}{Q_3(x)-Q_1(x)}$
   其中，$Q_1(x)$ 和 $Q_3(x)$ 分别是特征 x 的第一四分位数和第三四分位数。
4. **MaxAbsScaler**:
   MaxAbsScaler 通过除以每个特征的最大绝对值来缩放数据。对于每个特征 x，应用以下公式：
   $x_{\text{scaled}} = \frac{x}{\max(|x|)}$
5. **Normalizer**:
   Normalizer 对单个样本的特征向量进行缩放，使其具有单位范数（长度）。这通常用于文本分类和聚类。对于样本 x，应用以下公式：
   $x_{\text{scaled}} = \frac{x}{\|x\|_p}$
   其中，$\|x\|_p$ 是 p 范数，常见的 p 包括 1（曼哈顿距离）、2（欧几里得距离）等。

- 如果数据包含许多异常值，使用 `RobustScaler` 可能更合适。
- 当处理稀疏数据时，应谨慎选择 Scaler。例如，`StandardScaler` 和 `MinMaxScaler` 可能会改变数据的稀疏性，而 `MaxAbsScaler` 则保持数据的稀疏结构。

三、指标

1. **均方误差 (MSE - Mean Squared Error):**
   $\text{MSE}(\text{predict}, \text{target}) = \text{mean}((\text{target} - \text{predict})^2)$
2. **平均绝对误差 (MAE - Mean Absolute Error):**
   $\text{MAE}(\text{predict}, \text{target}) = \text{mean}(\text{abs}(\text{target} - \text{predict}))$
3. **平均绝对百分比误差 (MAPE - Mean Absolute Percentage Error):**
   $\text{MAPE}(\text{predict}, \text{target}) = \text{nanmean}\left(\text{abs}\left(\frac{\text{target}_{\text{nonzero}}-\text{predict}_{\text{nonzero}}}{\text{target}_{\text{nonzero}}}\right)\right)$
   为0的情况需要特殊处理
4. **对称平均绝对百分比误差 (sMAPE - Symmetric Mean Absolute Percentage Error):**
   $\text{sMAPE}(\text{predict}, \text{target}) = \text{mean}\left(\frac{2\times\text{abs}(\text{target}-\text{predict})}{\text{abs}(\text{target})+\text{abs}(\text{predict})}\right)$
   为0的情况需要特殊处理

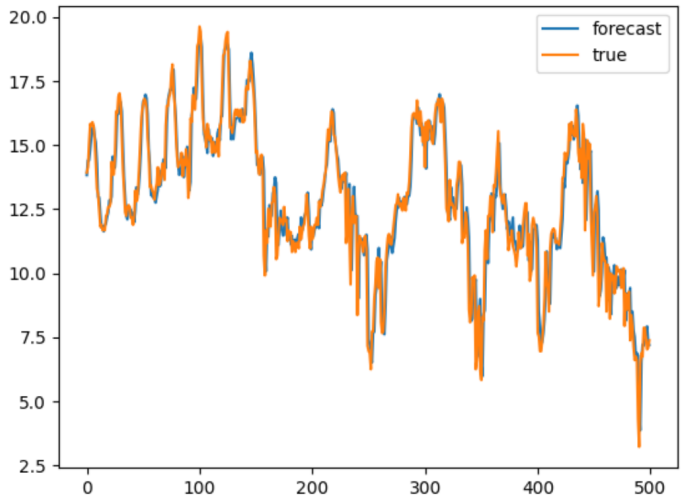
5. **平均绝对误差比例 (MASE - Mean Absolute Scaled Error):**

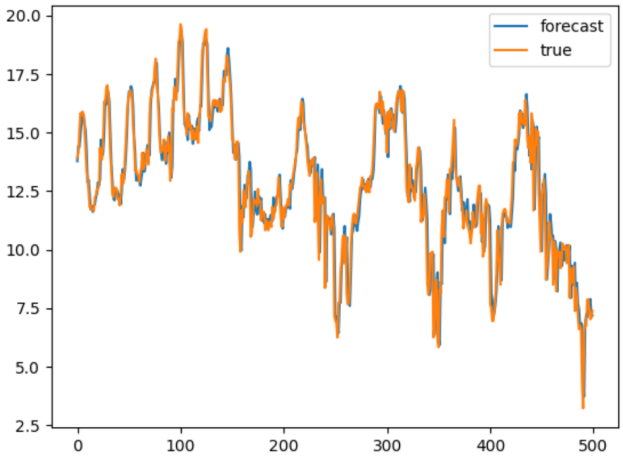
$$\mathrm{MASE}(\mathrm{predict}, \mathrm{target}) = \mathrm{mean}\left(\frac{\mathrm{abs}(\mathrm{target}-\mathrm{predict})}{\mathrm{mean}(\mathrm{abs}(\mathrm{target}_{1:}-\mathrm{target}_{:-1}))}\right)$$

切比雪夫距离：全部维度上的距离中最大值

四、自回归&指数平滑

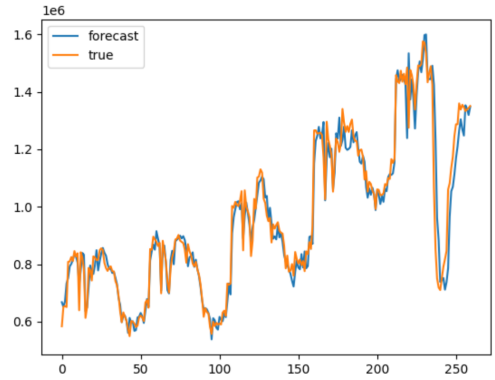
在`ETTh1.csv`上的两种方法预测效果图：（取每个pred_len=32的结果向量的第一维的值，拼接后作为forecast值画图）

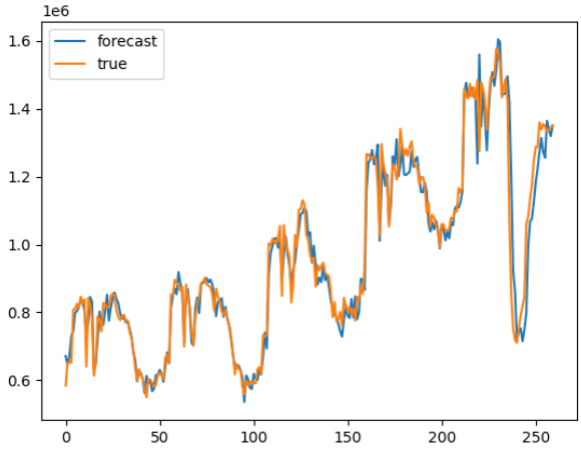


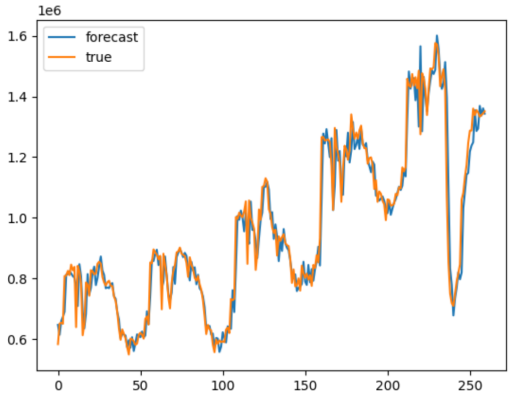


数据集	模型	转换	MSE	MAE	MAPE	SMAPE	MASE
ETTh1.csv	AR	无	3.7236823543	1.4223695605	0.316432467	0.2786462916	3.1206446025
	AR	归一化	3.7236823543	1.4223695605	0.316432467	0.2786462916	3.1206446025
	AR	标准化	3.7236823543	1.4223695605	0.316432467	0.2786462916	3.1206446025
	EMA	无	3.7212477249	1.4213196388	0.3162016966	0.2785964063	3.1183410997
	EMA	归一化	3.7212477249	1.4213196388	0.3162016966	0.2785964063	3.1183410997
	EMA	标准化	3.7212477249	1.4213196388	0.3162016966	0.2785964063	3.1183410997

在 national_illness.csv 上的三种方法预测效果图：（取每个pred_len=32的结果向量的第一维的值，拼接后作为forecast值画图）







数据集	模型	参数	转换	MSE	MAE	MAPE	SMAPE	MASE
illness.csv	AR	无	无	27439193740	108850	0.1034950089	0.1051011881	2.3992371951
	AR	无	归一化	27439193740	108850	0.1034950089	0.1051011881	2.3992371951
	AR	无	标准化	27439193740	108850	0.1034950089	0.1051011881	2.3992371951
	EMA	α=0.9	无	27419252529	108690	0.1033603869	0.1049807442	2.3957124803
	EMA	α=0.9	归一化	27419252529	108690	0.1033603869	0.1049807442	2.3957124803
	EMA	α=0.8	归一化	27416029462	108562	0.1032671931	0.1049089515	2.3928983186
	EMA	α=0.9	标准化	27419252529	108690	0.1033603869	0.1049807442	2.3957124803
	DES	α=0.9 β=0.2	无	36208536448	137220	0.1292560139	0.1356445787	3.0245655263
	DES	α=0.9 β=0.2	归一化	36208531889	137220	0.1292560072	0.135644571	3.0245653866
	DES	α=0.9 β=0.2	标准化	35451148797	136333	0.1282868701	0.1345597513	3.0050141933
	DES	α=0.8 β=0.2	标准化	35477352164	136305	0.1282743197	0.1345584203	3.0043878589
	DES	α=0.9 β=0.1	标准化	35346409865	136005	0.1280267384	0.1342522843	2.9977781036
	DES	α=0.7 β=0.05	标准化	35477666057	135562	0.1277068109	0.1339138782	2.9880239706

> 两参数指数平滑（Double Exponential Smoothing），也被称为霍尔特（Holt's）线性趋势模型，用于具有趋势但无季节性的时间序列数据

结论：

- 不同的转换方法，误差在10位小数范围内没有任何区别
- 关于EMA的理解：
  - 波动剧烈时alpha要设置的大
  - alpha=1等价于不平滑
  - 平滑有助于不稳定带来的累计误差增大现象
- 我还尝试了DES，即考虑了水平和趋势两个平滑参数的指数平滑模型，其性能更差了，但不同归一化方法的表现有了细微差别
- 减少DES的两个参数值，可以有细微的性能提升

五、TsfKNN

1、实现自定义距离度量

我的方法是增加一个参数`self.decompose`，用于决定在距离度量和搜索的时候是否考虑季节性和趋势性。具体实现方法为：

1. 对整个序列进行STL分解

```
def _fit(self, X: np.ndarray) -> None:
    self.X = X[0, :, -1]
    if self.decompose:
        self.X_stl = STL(self.X, period=self.period).fit()  # 对整个序列进行STL分解
        plot_STL(self.X_stl,2000)
```

2. 将距离的计算方法改为在趋势和季节分量上的距离

```
def _stl_modified_distance(self, x_component, y_components_series):
    # x_component 是单个时间序列的 STL 分解结果的趋势或季节性组件
    # y_components_series 是多个时间序列的 STL 分解结果的趋势或季节性组件的集合
    distances = []
    for y_component in y_components_series:
        # 计算 x_component 与 y_component 之间的距离
        dist = self.distance(x_component, y_component)
        distances.append(dist)
    return np.array(distances)
```

3. 单独创建一个search函数，需要传递数据分解后的结果

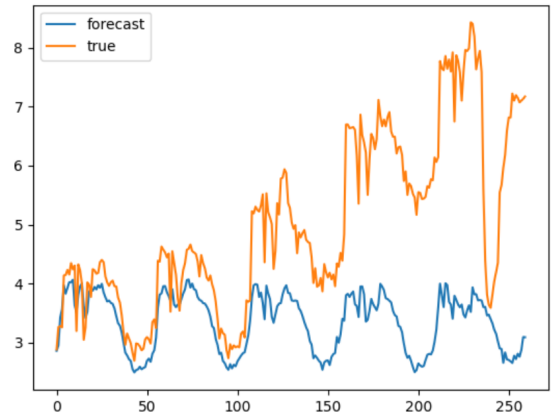
```python
def STL_search(self, x_stl, X_s_seasonal, X_s_resid, seq_len, pred_len):
    # 假设 x_stl 是单个时间序列的 STL 分解结果
    # X_s_trend 和 X_s_seasonal 是训练数据集的趋势和季节性组件的窗口
    if self.approximate_knn == False:
        if self.msas == 'MIMO':
            # 分别计算与 x_stl 的趋势和季节性组件的距离
            # distances_trend = self._stl_modified_distance(x_stl.trend, X_s_trend[:, :seq_len])
            # distances_seasonal = self._stl_modified_distance(x_stl.seasonal, X_s_seasonal[:, :seq_len])
            # distances_resid = self._stl_modified_distance(x_stl.resid, X_s_resid[:, :seq_len])
            # distances = distances_trend + distances_seasonal
            # distances = distances_seasonal + distances_resid

            #使用季节性和残差计算距离
            # distances=self._stl_modified_distance(x_stl.seasonal+ x_stl.resid, X_s_seasonal[:, :seq_len]+X_s_resid[:, :seq_le
            distances = self._stl_modified_distance(x_stl.seasonal, X_s_seasonal[:, :seq_len])
            indices_of_smallest_k = np.argsort(distances)[:self.k]
            neighbor_fore = X_s_resid[indices_of_smallest_k, seq_len:] + X_s_seasonal[indices_of_smallest_k, seq_len:]
            x_fore = np.mean(neighbor_fore, axis=0, keepdims=True)
            return x_fore
```

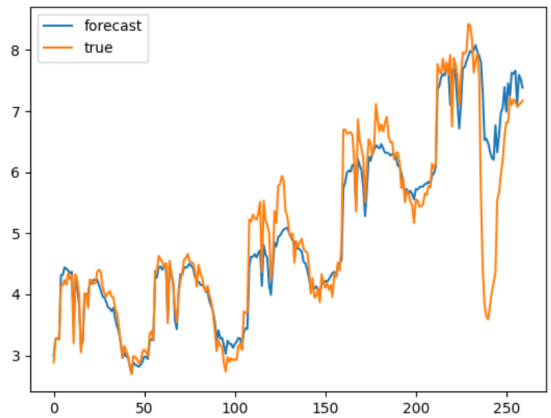
4. 在预测时，使用线性模型预测trend分量，加在预测结果中（重要）

```python
        if self.decompose:
            x_fore = self.STL_search(x_stl,  X_s_seasonal,X_s_resid, seq_len, pred_len)
            #使用线性回归模型，预测x_stl.trend的未来pred_len个时间点的值
            model=LinearRegression()
            model.fit(np.arange(seq_len).reshape((-1,1)),x_stl.trend.reshape((-1,1))[:,-seq_len:])
            x_stl_trend=model.predict(np.arange(seq_len,pred_len+seq_len).reshape((-1,1))).reshape((1,-1))
            x_fore+=x_stl_trend
        else:
            x_fore = self._search(x, X_s, seq_len, pred_len)
```

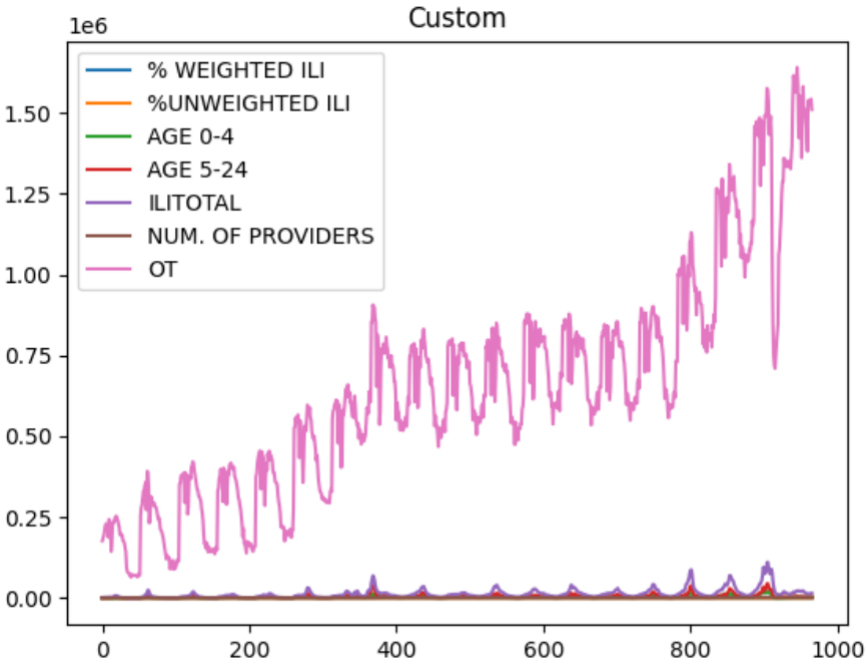
TSFKNN在没见过的trend上，表现不好

使用STL分解后的预测效果对比：





不使用STL会导致预测失效的深层原因：测试数据的分布与训练数据不同，模型没见过测试数据：



2、LSH的尝试（未成功）

尝试一：MinHashLSH

```
MinHashLSH(threshold=0.01, num_perm=Num_perm)
```

> 用于找到jaccard相似性下的近邻，在处理文本数据或离散数据时尤其有效

尝试了很多阈值threshold和num_perm，结果都是无法完成哈希分桶和搜索

尝试二：LSHash

```
self.lsh_models[input_dim] = LSHash(hash_size=10, input_dim=input_dim)
```

> 更通用的LSH方法，可以应用于多种距离度量和数据类型。

LSHash需要在创建时就给定input_dim，但实际使用时需要有1和96两种input_dim，该问题尚未解决

代码方面的收获

内部方法子类重写raise NotImplementedError

滑动窗口可以掉包实现

```
subseries = np.concatenate(([sliding_window_view(v, self.seq_len+1) for v in
X_target]))
```

MAPE在处理0问题时，小常数不行，容易爆掉

动态回归模型可以利用上多个变量