血糖水平时间序列预测

第二小组：2051498 储岱泽 2051828 莫益萌 2151615 沈书勤 2151641 王佳垚

---

1. 项目背景

在当今社会，糖尿病已成为一种全球性的公共卫生问题，对人们的生活质量和健康造成了严重影响。根据国际糖尿病联合会（IDF）的报告，糖尿病患者数量正以惊人的速度增长，给医疗系统和患者带来了沉重的负担。因此，开发有效的工具和方法来预测和管理血糖水平对于改善糖尿病患者的治疗效果和生活质量至关重要。

2. 领域知识调研

2.1 数据集选择

本项目旨在通过数据分析与数据挖掘技术，探索和实现一个精确的血糖水平时间序列预测系统。我们选择的数据集是来源于论文“Chinese diabetes datasets for data-driven machine learning”的上海T1DM和上海T2DM数据集，这是一个专门针对中国糖尿病患者的综合性数据集，包含了I型糖尿病患者和II型糖尿病患者的血糖变化随时间的情况，包含了丰富的个人健康记录、生活习惯、饮食日记等信息。该数据集不仅为我们提供了一个深入了解糖尿病患者群体的机会，而且为我们研究血糖水平预测模型提供了宝贵的数据资源。数据集包括中国上海的1型（n=12）和2型（n=100）糖尿病患者。

2.2 I型糖尿病与II型糖尿病

1型糖尿病发生在身体的防御系统错误地攻击胰腺中产生胰岛素的细胞时，这类患者需要终身注射胰岛素，通常从年轻时开始。2型糖尿病更为常见，与超重和缺乏运动等因素有关，导致身体对胰岛素产生抵抗或胰岛素分泌不足，但通常可以通过生活方式的改变、药物或必要时的胰岛素注射来控制。

3. 数据预处理与特征提取

在开展模型训练前，我们对数据集进行了预处理。数据集中主要包含两部分内容：

1、**动态特征：**T1、T2型患者的血糖随时间动态变化的情况，受到饮食、胰岛素注射、其他降血糖药物摄入等因素的影响。

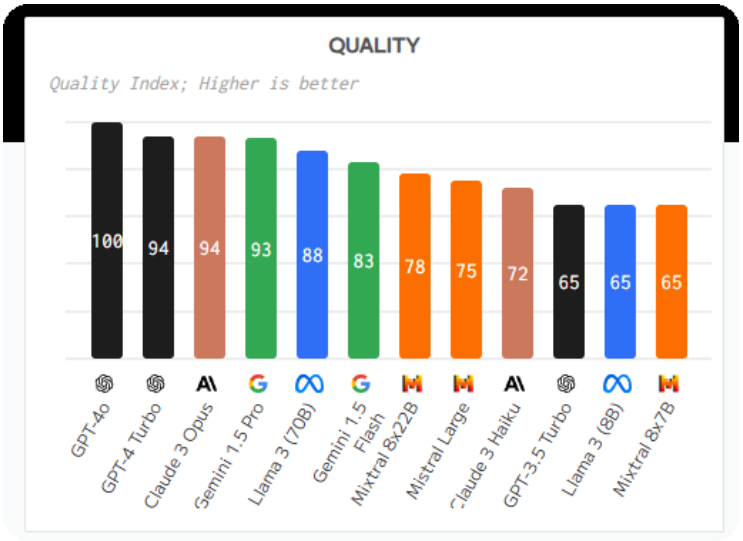
2、**静态特征：**包含各个患者的性别年龄等身体状况、吸烟饮酒史、糖尿病持续时间、并发症、合并症、使用的降血糖药、空腹以及餐后两小时的血糖含量等。

为了选取特征以及进行模型训练，对于这两部分数据我们分别进行了处理

3.1 静态数据处理

3.1.1 根据领域知识删除不重要特征

经过对数据的分析和观察，我们发现除了我们需要进行预测的CGM值，数据集中还有很多特征，这些特征有的会对我们需要预测的值有比较大的影响，有的可能并没有太大的影响，因此我们需要对我们的数据集进行降维，删除多余的特征而保留与我们需要预测的CGM变化相关性最高的特征，从而避免因为数据维度太高而导致的维度灾难和模型过拟合问题。



大模型质量排行榜https://artificialanalysis.ai/

我们首先将数据集中的静态特征喂给目前大模型排行榜上前三的大模型进行询问，分别各问一次，综合三个模型的建议，发现他们都认为在静态数据中以下特征一般来说对血糖都有较大的影响：

```
# 要保留的列
columns_to_keep = [
    "Patient Number",
    "BMI (kg/m2)",
    "Duration of Diabetes (years)",
    "Fasting Plasma Glucose (mg/dl)",
    "2-hour Postprandial Plasma Glucose (mg/dl)",
    "Fasting Insulin (pmol/L)",
    "2-hour Postprandial Insulin (pmol/L)",
    "HbA1c (mmol/mol)",
    "Glycated Albumin (%)",
    "Hypoglycemic Agents",
]
```

因此，我们进行初步的筛选保留以上这些数据特征。

3.1.2 数据预处理

为了使得数据的格式便于处理，我们首先对提取出来的列的数据进行了数据清洗：

1. 观察数据集发现一些列中本来应当是有浮点数类型的数据，但是出现了意义不明的“/”符号，这可能是因为这个地方有数据缺失，然而如果直接将数据缺失的行给暴力删除，将会丢失很多宝贵的数据，所以我们将这些缺失的数据用NaN填充，然后转换成浮点类型，再用中位数填充。

lasma (mg/dl)	2-hour Postprandial Plasma Glucose (mg/dl)	Fasting C-peptide (nmol/L)	2-hour Postprandial C-peptide (nmol	Fasting Insulin (pmol/L)	2-hour Postprandial Insulin (pmo	HbA1c (mmol/mol)	Glycated Albumin (%)	Total Cholesterol (mmol/L)	Triglyceride (mmol/L)	High-Density Lipoprotein Chole ol	
0	348.84	0.050	0.050	/	/	115	40.70	3.59	1.02	0.86	
0	258.84	0.016	0.016	543.38	754.71	69	19.60	4.78	2.20	0.93	
0	258.84	0.016	0.016	543.38	754.71	69	19.60	4.78	2.20	0.93	
0	/	0.016	0.016	78.37	74.39	73	25.10	3.49	1.82	0.84	
0	248.76	0.100	0.120	/	/	122	46.60	5.61	1.14	1.08	
0	305.10	0.090	0.280	/	/	125	37.60	4.57	0.91	1.27	
0	370.44	0.020	0.060	238.09	550.40	68	25.70	4.05	0.46	1.57	
8	193.68	0.007	0.007	30.18	231.60	68	29.20	4.44	0.68	1.97	
0	245.52	0.013	0.010	17.04	445.80	64	27.00	5.12	0.64	1.88	
0	72.54	0.007	0.007	21.66	540.60	61	26.60	4.27	0.54	1.72	
0	297.00	0.140	0.270	14.49	28.74	54	18.30	5.09	0.64	1.84	
0	372.96	0.016	0.016	/	/	63	20.10	4.55	1.57	0.77	
0	176.58	0.150	0.230	311.76	/	78	24.90	5.46	1.77	0.90	
0	329.76	0.070	0.160	61.55	240.58	166	71.10	5.15	0.68	1.98	
0	283.86	0.016	0.016	7.28	282.11	54	21.00	4.64	1.03	1.48	
0	/	/	/	/	/	/	/	4.69	0.74	2.33	

用中位数填充缺失值可以保持数据集的代表性，因为中位数不会受到极端值的影响，是数据集中相对稳定的值。这种填充方式可以减少因缺失值填充而引入的偏差。

```
# 定义需要处理的列
columns_to_process = [
    "Fasting Plasma Glucose (mg/dl)",
    "2-hour Postprandial Plasma Glucose (mg/dl)",
    "Fasting Insulin (pmol/L)",
    "2-hour Postprandial Insulin (pmol/L)",
    "HbA1c (mmol/mol)",
    "Glycated Albumin (%)",
]
# 替换特殊字符 '/' 为 NaN 并转换为浮点数类型，然后用中位数填充缺失值
for column in columns_to_process:
    dataT1DM[column] = dataT1DM[column].replace("/", np.nan).astype(float)
    dataT2DM[column] = dataT2DM[column].replace("/", np.nan).astype(float)
    dataT1DM[column] = dataT1DM[column].fillna(dataT1DM[column].median())
    dataT2DM[column] = dataT2DM[column].fillna(dataT2DM[column].median())
```

2. 通过观察我们发现在Hypoglycemic Agents（降糖药物）列有很多多余的空格，为了方便后续的处理，我们将这些空格删去。

cute Diabetic Complications	Diabetic Macrovascular Complications	Diabetic Microvascular Complications	Comorbidities	Hypoglycemic Agents	
none	l disease, cerebro	neuropathy,	hypertension,	Humulin R, insulin detemir, acarbose	cobalamin, calcit
etic ketoacidosis	coronary heart	neuropathy,	hypertension,	insulin aspart, insulin detemir	olmesartan medo
etic ketoacidosis	coronary heart	neuropathy,	hypertension,	Novolin R, insulin glargine	olmesartan medo
none	coronary heart	neuropathy,	hypertension,	Novolin R	olmesartan medo
etic ketoacidosis	none	neuropathy	leucopenia,	Novolin 50R, acarbose	
none	peripheral arterial	neuropathy	hypocalcemia,	insulin detemir, Novolin R, acarbose	calcitriol, calc
none	none	none	thyroid nodule,	insulin aspart 70/30	
none	none	none	none	voglibose, Humulin R, insulin degludec	

```
# 去除 'Hypoglycemic Agents'（降糖药物） 列中药物名称的多余空格
combined_data["Hypoglycemic Agents"] = (
    combined_data["Hypoglycemic Agents"]
    .str.replace(r"\s+", " ", regex=True)
    .str.strip()
)
```

3. 由于每一个患者可能会使用不同的药物组合，所以我们对药物名称进行了独热编码，将分类数据转换为模型可以理解的数字格式。这样，我们就可以将每种药物或药物组合表示为一个二进制向量，其中每个药物或组合对应一个特定的二进制位。这种表示方式使得机器学习模型能够更好地理解药物之间的关系，从而更准确地预测糖尿病患者的情况或进行其他类型的分析。

```
# 对 'Hypoglycemic Agents' 列进行独热编码
drug_dummies = combined_data["Hypoglycemic Agents"].str.get_dummies(sep=", ")
# 删除包含'None'或空值的列
if "none" in drug_dummies.columns:
    drug_dummies.drop(columns=["none"], inplace=True)
# 添加 'Patient Number' 列
drug_dummies.insert(0, "Patient Number", combined_data["Patient Number"])
# 将独热编码表单独保存为新的CSV文件
drug_dummies.to_csv("data/hypoglycemic_agents.csv", index=False, encoding="utf-8")
```

Patient Nur	Gansulin 40	Gansulin R	Humulin 70	Humulin R	Novolin 30I	Novolin 50I	Novolin R	acarbose	canagliflozi	dapagliflozi	gliclazide	glimepiride	gliquid
1001_0_202	0	0	0	1	0	0	0	1	0	0	0	0	
1002_0_202	0	0	0	0	0	0	0	0	0	0	0	0	
1002_1_202	0	0	0	0	0	0	1	0	0	0	0	0	
1002_2_202	0	0	0	0	0	0	1	0	0	0	0	0	
1003_0_202	0	0	0	0	0	1	0	1	0	0	0	0	
1004_0_202	0	0	0	0	0	0	1	1	0	0	0	0	
1005_0_202	0	0	0	0	0	0	0	0	0	0	0	0	
1006_0_202	0	0	0	1	0	0	0	0	0	0	0	0	
1006_1_202	0	0	0	1	0	0	0	0	0	0	0	0	
1006_2_202	0	0	0	1	0	0	0	0	0	0	0	0	
1007_0_202	0	0	0	0	0	0	1	0	0	0	0	0	
1008_0_202	0	0	0	0	0	0	1	0	0	0	0	0	
1009_0_202	0	0	0	0	0	0	0	0	0	1	0	0	
1010_0_202	0	0	0	0	0	0	1	0	0	0	0	0	
1011_0_202	0	0	0	0	0	0	1	0	0	0	0	0	
1012_0_202	0	0	0	0	0	0	1	0	0	0	0	0	
2000_0_202	1	0	0	0	0	0	0	0	0	0	0	0	

hypoglycemic_agents.csv展示药物独热编码结果

3.1.3 相关性分析与特征选择

1. 计算统计特征

首先，我们计算给定文件夹中的每个 CSV 文件中 "CGM (mg / dl)" 列的统计特征，并将结果保存到指定的输出文件中。

```python
if "CGM (mg / dl)" in df.columns:
    patient_id = os.path.splitext(file_name)[0]
    mean_value = df["CGM (mg / dl)"].mean()      #计算平均值
    median_value = df["CGM (mg / dl)"].median() #计算中位数
    std_value = df["CGM (mg / dl)"].std()        #计算标准差
    min_value = df["CGM (mg / dl)"].min()        #计算最小值
    max_value = df["CGM (mg / dl)"].max()        #计算最大值
```

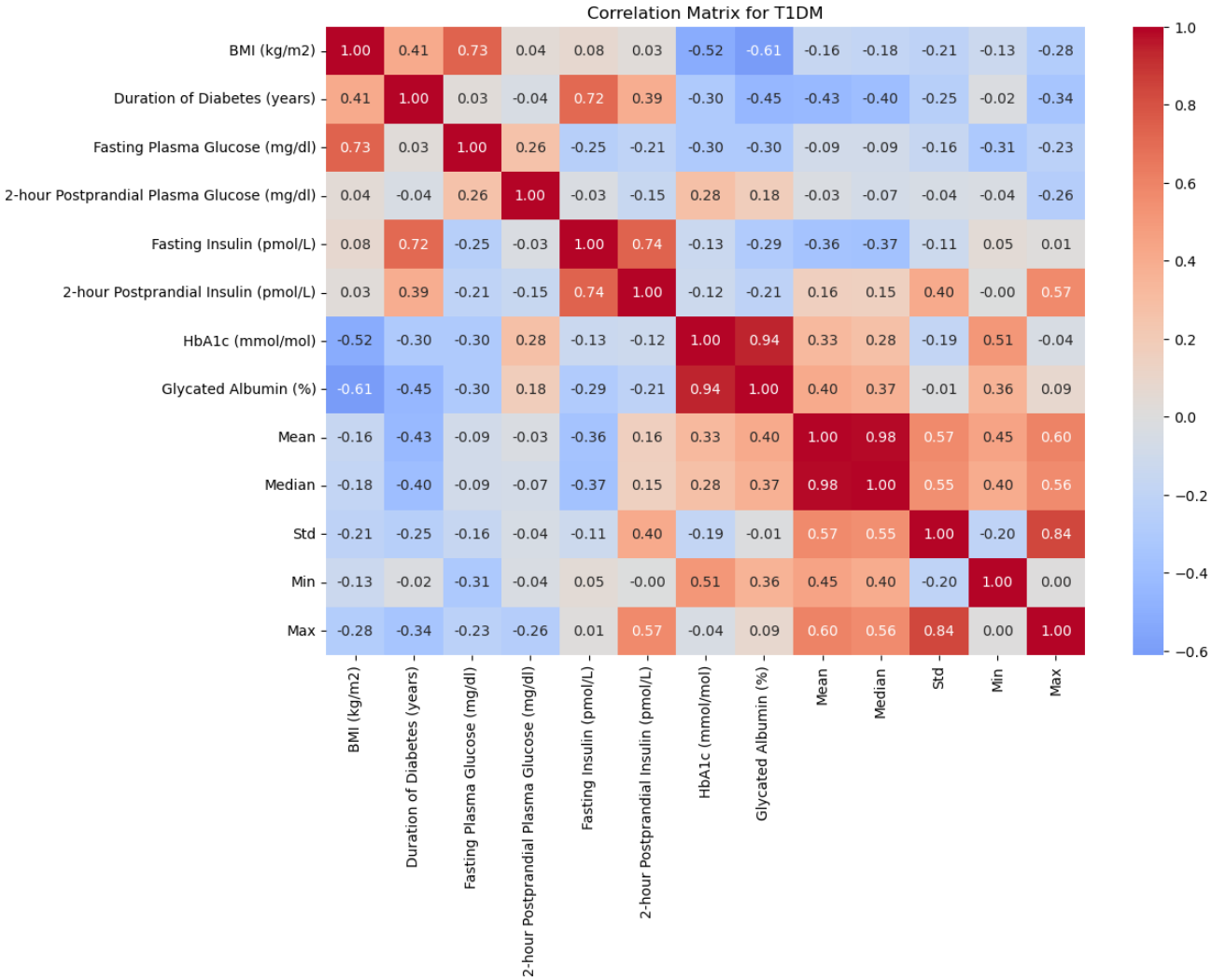
2. 探索特征之间的相关性

分别对 T1DM 和 T2DM 的数据进行相关性分析并绘制热图。相关性热图提供了数据集中各个变量之间相关性强度的可视化表示。以下是对提供的T1DM（1型糖尿病）和T2DM（2型糖尿病）相关性热图的分析和结论，矩阵中的每个单元格代表两个变量之间的**Pearson相关系数**，范围从-1到1，其中：

- **1**表示完全正相关，
- **-1**表示完全负相关，
- **0**表示没有线性相关。

矩阵中使用颜色编码，红色表示正相关，蓝色表示负相关，颜色越深，相关性越强。

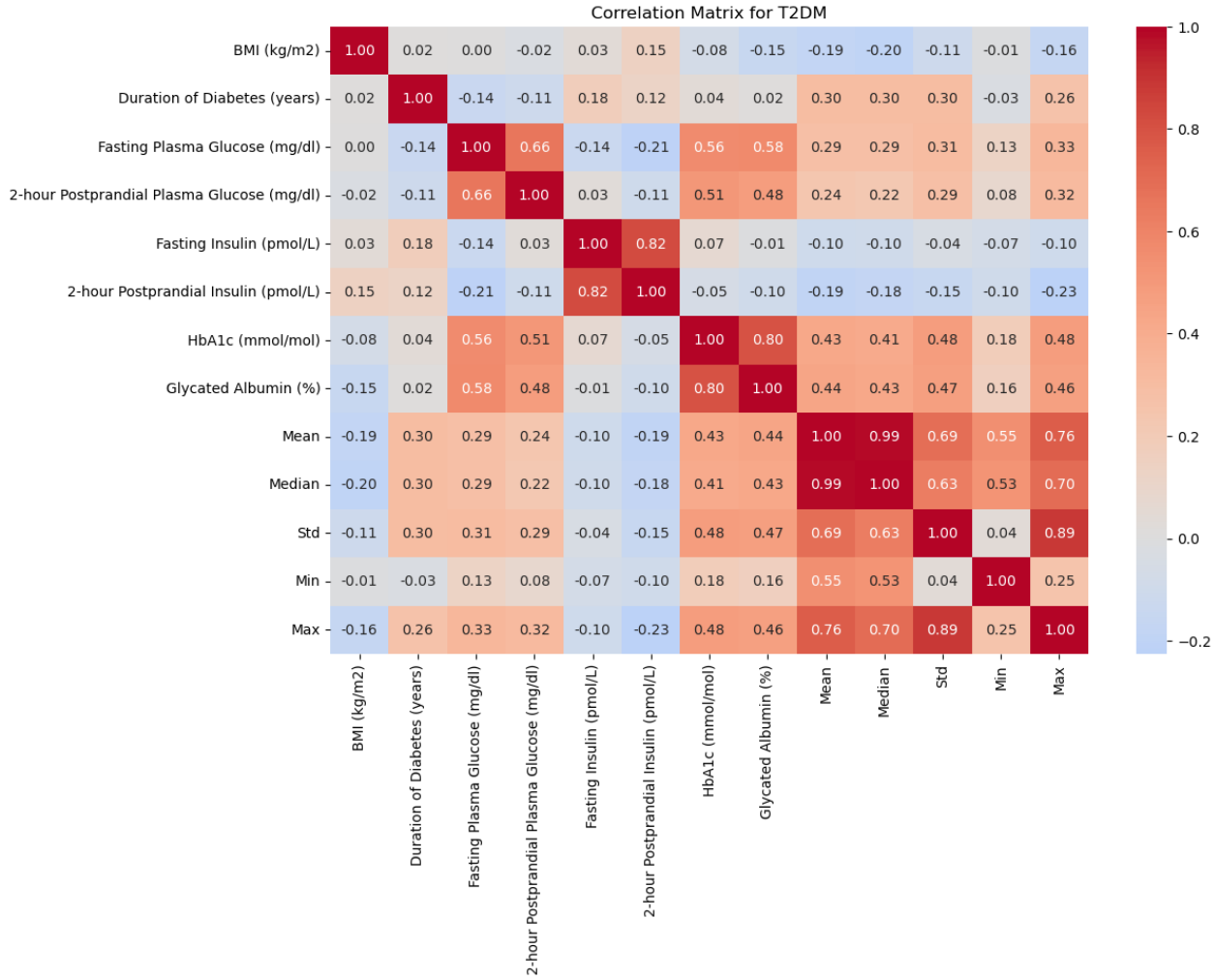
T1DM相关性热图分析



T1DM相关性矩阵

1. **HbA1c（糖化血红蛋白）与Glycated Albumin（糖化白蛋白）**：两者之间的相关性非常高（0.94），表明它们之间存在强烈的正相关关系。这符合预期，因为Glycated Albumin（糖化白蛋白）通常与HbA1c（糖化血红蛋白）一样，是血糖控制的指标。糖化白蛋白反映了过去2到4周内的平均血糖水平。与HbA1c（糖化血红蛋白）不同，后者反映的是过去2到3个月的平均血糖水平。
2. **Fasting Insulin (空腹胰岛素) 与 2-hour Postprandial Insulin (餐后2小时胰岛素):** 相关性极高（0.74），这表明这两个变量在大多数情况下是同时升高或降低的。胰岛素是调节血糖的重要激素，空腹和餐后胰岛素水平通常都受到胰腺功能和胰岛素抵抗程度的影响。因此，它们之间有显著的相关性。
3. **BMI (体重指数) 与 Fasting Plasma Glucose (空腹血糖) :**有较高的正相关性（0.73）。高BMI通常与肥胖相关，肥胖是2型糖尿病和胰岛素抵抗的主要风险因素之一。较高的体重指数通常伴随着更高的空腹血糖水平，这是因为肥胖可能导致胰岛素抵抗，从而引起血糖水平升高。

T2DM相关性热图分析



T2DM相关性分析

根据T2DM的相关性矩阵分析，我们发现**Fasting Insulin (空腹胰岛素) 与 2-hour Postprandial Insulin (餐后2小时胰岛素)**同样有很高的相关性，高达0.82。**HbA1c（糖化血红蛋白）与Glycated Albumin（糖化白蛋白）**同样也有很高的相关性，高达0.80。但是反常的是，与T1DM中的数据不同，在T2DM中**BMI (体重指数) 与 Fasting Plasma Glucose (空腹血糖)** 没有相关性，为0。这说明1型糖尿病和2型糖尿病在病理上有很大的差异。

3. 静态特征选取

接下来我们在这些静态特征中进行特征的选取，我们计算每个静态特征与五个CGM统计特征（Min，Max，Std，Median，Mean）之间的相关性总和，并按相关性总和降序排序。

```
{'Duration of Diabetes (years)': 1.4461835599668196,
 'HbA1c (mmol/mol)': 1.3381624365752587,
 '2-hour Postprandial Insulin (pmol/L)': 1.2870104268929974,
 'Glycated Albumin (%)': 1.2292193902414628,
 'BMI (kg/m2)': 0.955430280946018,
 'Fasting Insulin (pmol/L)': 0.9032887296642582,
 'Fasting Plasma Glucose (mg/dl)': 0.8853858985415652,
 '2-hour Postprandial Plasma Glucose (mg/dl)': 0.445899151433402}
```

T1DM静态特征与CGM统计特征相关性总和

```
{'HbA1c (mmol/mol)': 1.9837058192223456,
 'Glycated Albumin (%)': 1.9526663368754618,
 'Fasting Plasma Glucose (mg/dl)': 1.357040941918417,
 'Duration of Diabetes (years)': 1.1807050119333464,
 '2-hour Postprandial Plasma Glucose (mg/dl)': 1.153357244759583,
 '2-hour Postprandial Insulin (pmol/L)': 0.8380879462291625,
 'BMI (kg/m2)': 0.6677050000051548,
 'Fasting Insulin (pmol/L)': 0.4171549000707161}
```


T2DM静态特征与CGM统计特征相关性总和

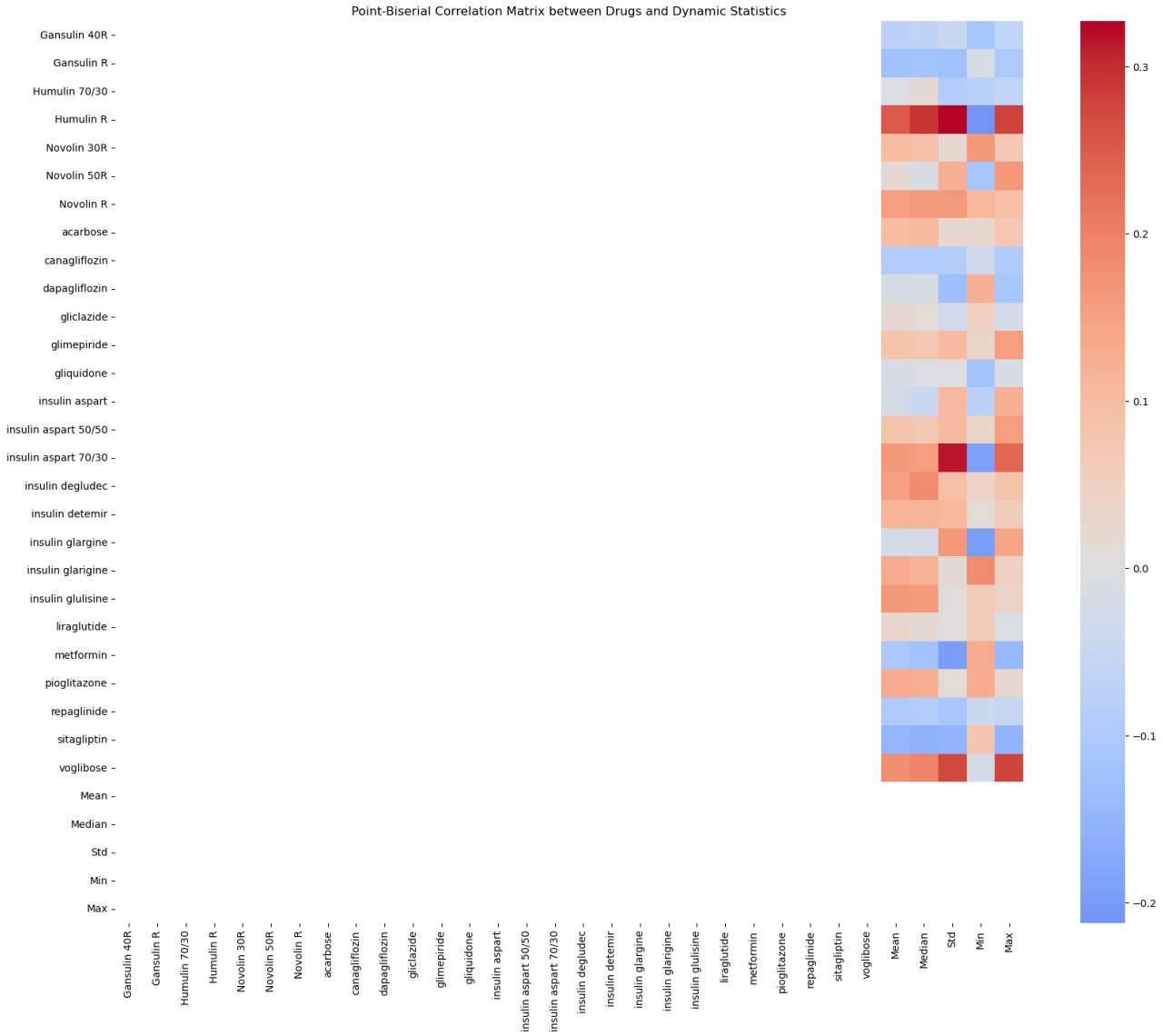
然后我们将每一个静态特征在T1DM和T2DM中的相关性总和相加，如果总和大于2则说明其和CGM的相关性极强，则将其作为是我们选择的特征。最后我们选择了以下这五个作为我们的静态特征：

	L	M	N	O	P
_c	Duration of Diabetes (years)	Fasting Plasma Glucose (m	2-hour Postprandial Insulin (p	HbA1c (mmol/mol)	Glycated Albumin (%)
2	10	352.8	363.955	115.311	40.7

5个静态特征

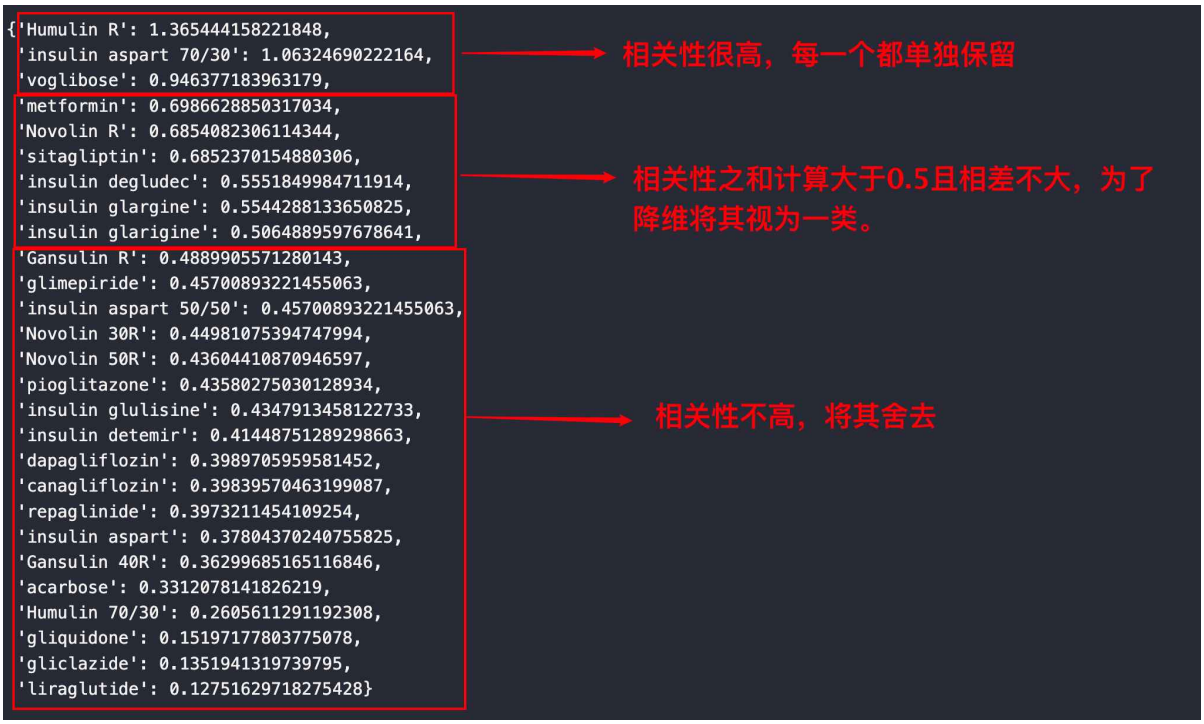
4. 对药物的数据处理

由于患者使用的药物我们在数据处理时采用了独热编码，所以我们需要用特殊的方法对其对五个统计量的相关性进行分析，下面是我们对于每一个药物的使用对CGM相关性的分析的热力图。



Hypoglycemic Agents与CGM相关性分析

之后我们将每一个药物关于五个统计值的相关性相加得到下面的结果，我们发现Humulin R，insulin aspart 70/30，voglibose这三种药的相关性值很高，将其每一个都单独保留；而下面的六种药和其形成了一种断层，但是高于0.5且相差不是很大，为了使得数据的维度变低，我们可以将它们视作一类；接下来下面的相关性就很低，可以将它们舍去。



将每个药物相关于后面五个统计量的相关性进行相关的结果

3.2 动态数据处理

3.2.1 根据领域知识删除不重要特征

同样，跟前面的静态数据处理一样，我们通过查阅资料以及询问目前top3的大模型，先进行特征的筛选：

```
# 要保留的列
required_columns = [
    "Date",
    "CGM (mg / dl)",
    "Dietary intake",
    "Insulin dose - s.c.",
    "CSII - bolus insulin (Novolin R, IU)",
    "CSII - basal insulin (Novolin R, IU / H)",
]
```

根据领域知识筛选得到的特征

第一轮先留下这些特征。

3.2.2 数据预处理

3.2.2.1 数据清洗

首先我们先对动态的数据进行了清洗：

- 将所有的Date数据转换成datetime类型。
- Insulin dose - s.c是指皮下注射的胰岛素剂量，该变量在数据集中存储的形式是类似于“Novolin R, 6 IU”这样的形式，我们只考虑其注射剂量的多少，所以在预处理的时候将数值部分提取出来，其他部分舍弃，同时将空值替换成0，显式的转换成整数类型。

```
# 检查列是否存在
if "Insulin dose - s.c." in df.columns:
    # 提取数值部分并转换为整数类型
    def extract_int(value):
        match = re.search(r"(\d+)\s*IU", str(value))
        return int(match.group(1)) if match else None
    df["Insulin dose - s.c."] = df["Insulin dose - s.c."].apply(extract_int)
    df["Insulin dose - s.c."] = df["Insulin dose - s.c."].fillna(0)# 将空值替换为0
    df["Insulin dose - s.c."] = df["Insulin dose - s.c."].astype(int)# 显式地转换为整数类型
```

3.2.2.2 获取碳水摄入

动态数据集的Dietary intake字段记录了病人在连续血糖监测期间的饮食摄入。当病人从食物中摄入碳水化合物时，碳水化合物会被消化成葡萄糖，并释放到血液中，从而影响血糖的变化，可能会对血糖动态变化的预测结果产生影响。考虑到这一点，我们希望能从饮食中推算病人摄入的碳水含量。

该字段的数据以食物名-摄入量的文本形式存储，以逗号分隔，未记录的进食行为以data not available标记。对于有效的记录，我们使用了Nutritionix API来辅助处理。该API提供了将自然语言记录的饮食解析为食物名称与摄入量，并按照其自有数据库的记录，估算营养信息，返回为JSON格式的数据。我们解析了其中的食物名称与碳水含量，作为新字段加入到原始数据表格中。主要的处理过程如下：

```
for index, row in df.iterrows():
    dietary_intake = row['Dietary intake']
    if pd.isna(dietary_intake) or dietary_intake == "data not available" or dietary_intake == "Data not available":
        food_segmentations.append(None)
        carbohydrate_totals.append(None)
        continue

    print(dietary_intake)
    # 发送Dietary intake单元格内容给Nutritionix API
    nutrition_data = get_nutritionix_data(dietary_intake)
    if nutrition_data:
        food_names = [food['food_name'] for food in nutrition_data['foods']]
        total_carbohydrates = sum(food.get('nf_total_carbohydrate', 0) for food in nutrition_data['foods'])
```

```
14
15          food_segmentations.append(", ".join(food_names))
16          carbohydrate_totals.append(total_carbohydrates)
17      else:
18          food_segmentations.append("Unmatched")
19          carbohydrate_totals.append(None)
20
21  df['food segmentation'] = food_segmentations
22  df['Carbohydrate/g'] = carbohydrate_totals
```

3.2.2.3 数据处理和归一化

首先，拆分了注射速率中的合并单元格，并规范化了日期信息。为CGM（连续葡萄糖监测）字段计算了一小时滚动均值。原始数据中的血糖水平可能存在短期的波动和噪声，通过计算一小时滚动均值，可以平滑这些波动，使得血糖数据更具可读性，减少随机波动的影响，显示较为整体的趋势。

对于剂量、碳水等字段，进行了空值填充，去除了错误值，并去掉了文本记录的饮食等字段。

对于Non-insulin hypoglycemic agents 字段和Insulin dose - s.c.字段，由于也是用药物名+剂量的文本形式存储的，因此对该字段也进行了拆分解析，分为剂量、药物名两个字段。主要的处理函数如下所示。

```
1  # 提取和转换药物名称和剂量信息
2  def extract_dose(drug_string):
3      try:
4          dose = float(drug_string.split(',')[-1].split(' ')[-2])
5          return dose
6      except:
7          return np.nan
8
9  def extract_drug_name(drug_string):
10     try:
11         name = drug_string.split(',')[0].strip()
12         return name
13     except:
14         return np.nan
```

为进一步的相关性分析与模型训练，选取了一些字段，进行了归一化操作。

```
1  columns_to_normalize = [
2      'CGM',
3      'CBG (mg / dl)',
4      'Blood Ketone (mmol / L)',
5      'Insulin dose - s.c. (IU)',
```

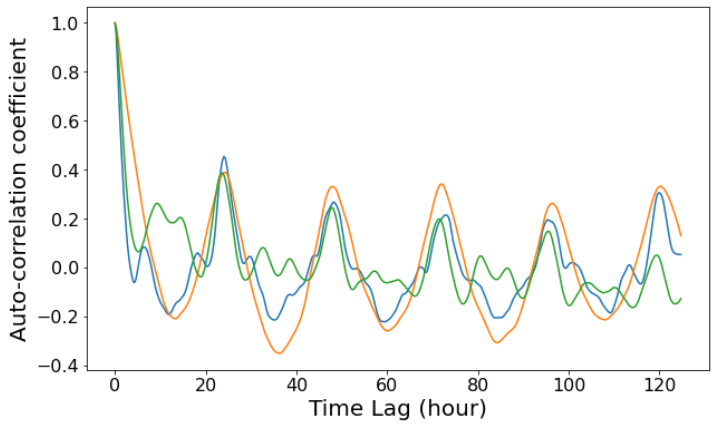
```
    'CSII - bolus insulin (Novolin R, IU)',
    'CSII - basal insulin (Novolin R, IU / H)',
    'rolling_mean_CGM',
    'rolling_std_CGM'
]

scaler = StandardScaler()
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])
```

3.2.3 相关性分析与特征选取

1. 患者CGM序列自回归成明显周期性

我们使用了随机三个患者的CGM时间序列进行自相关性分析，并绘制自相关函数（ACF）图，以分析血糖值的时间序列特性。

我们发现数据呈现明显的周期性，并且大概以24小时为一个周期，因为发现间隔24小时的倍数的数据之间呈现很强的相关性。



- 横坐标（Time Lag）表示在时间序列中，数据点之间的时间间隔。例如，如果采样间隔为15分钟，滞后4表示1小时的时间间隔。
- 纵坐标（Auto-correlation coefficient）表示当前数据点与滞后数据点之间的相关程度。高正值表示强正相关，高负值表示强负相关，接近零表示无明显相关性。

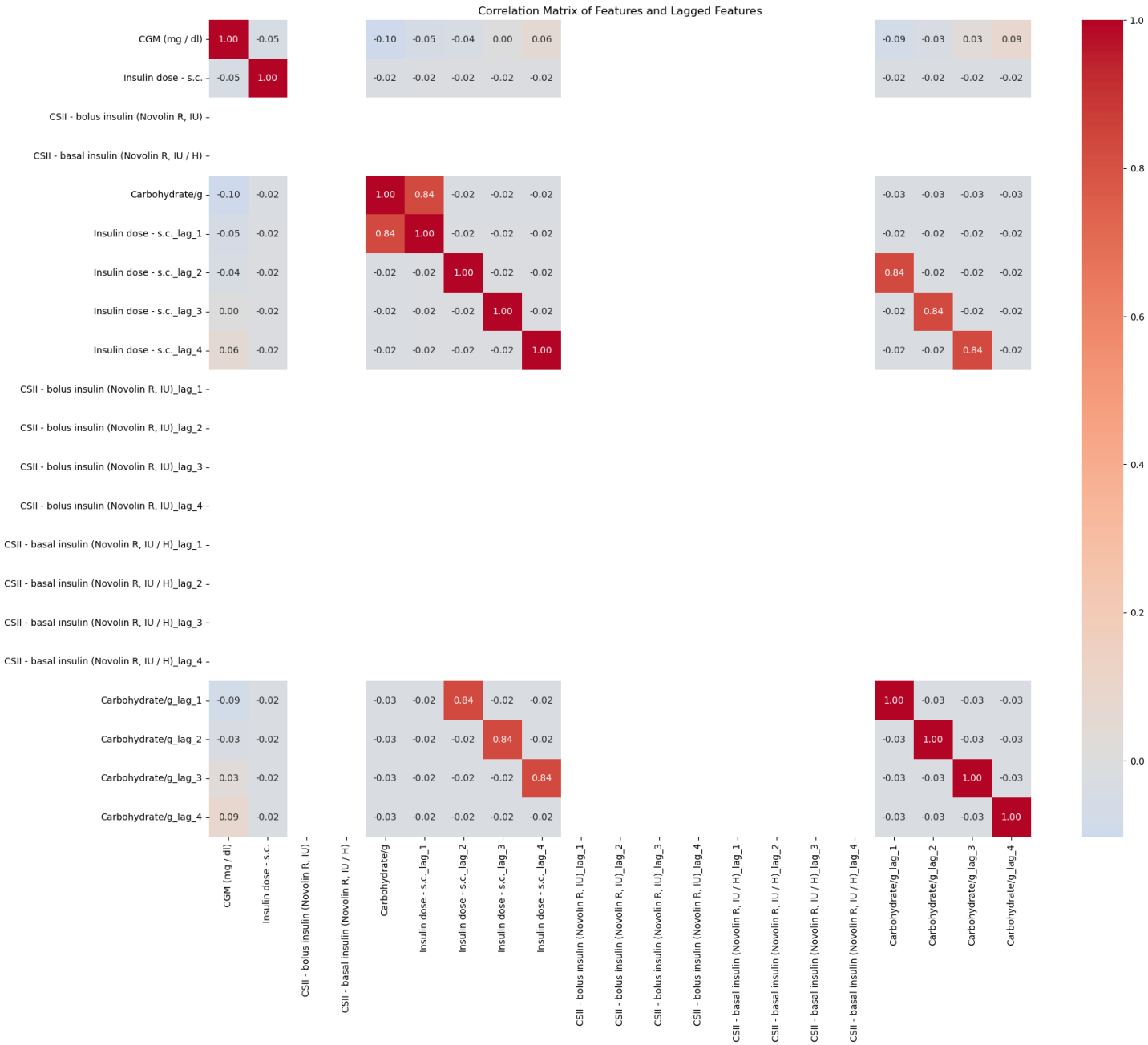
2. 创建滞后特征并寻找其与其他特征的相关性

我们对病人的时间序列以15分钟一个点创建滞后特征，在一个时间序列中，当前时间点的值可能受到之前时间点的影响。通过创建滞后特征，我们可以捕获这种时间上的动态关系，从而更好的帮助模型理解数据的时间依赖性。同时，滞后特征提供了更多关于数据的历史信息，这有助于模型更好地理解数据的上下文和趋势。这些信息可能包括趋势、周期性或季节性等。

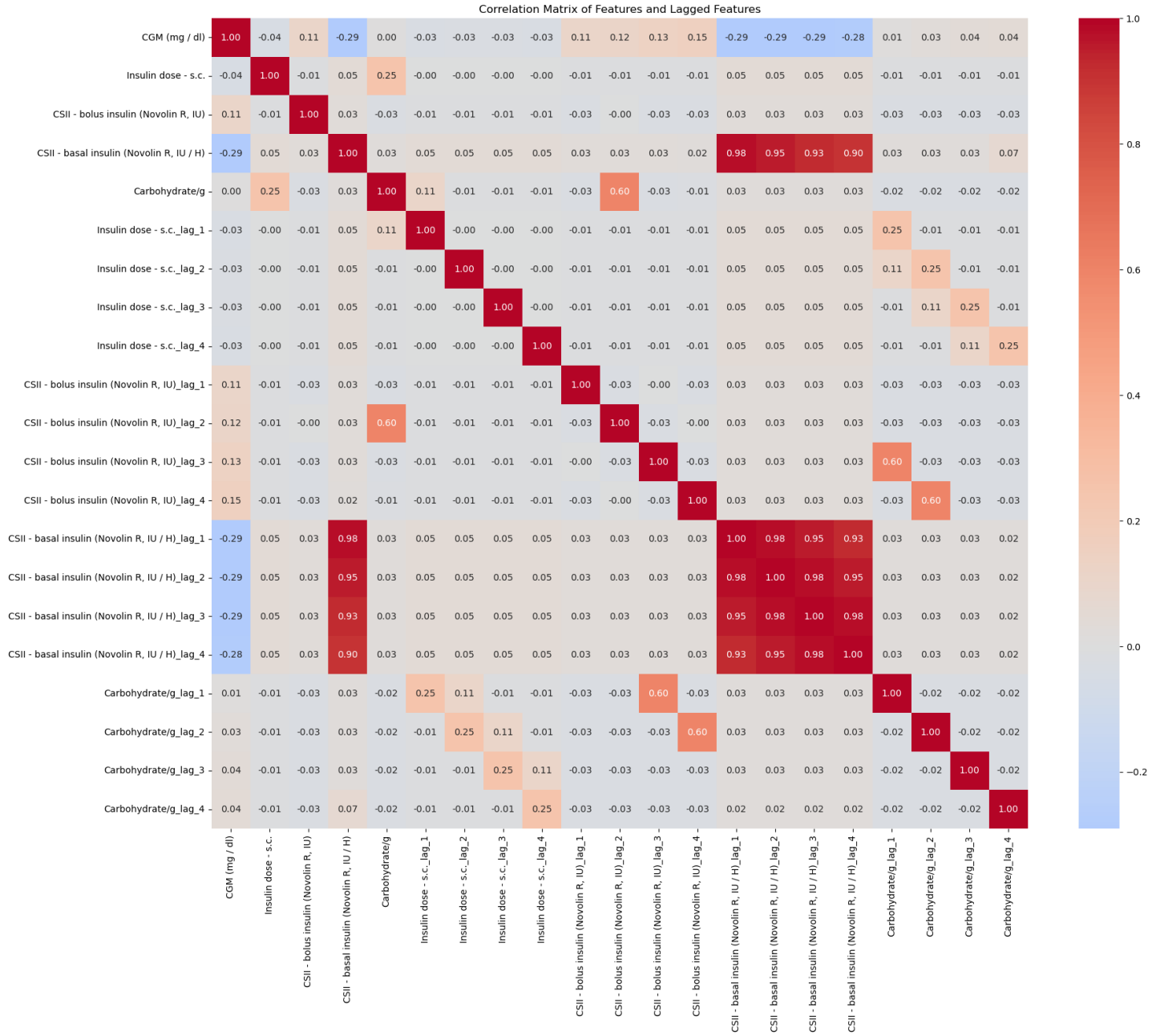
```
# 创建滞后特征
lagged_features = []
window_size = 4  # 假设每小时4个点，即每15分钟一个点
for feature in features:
    if feature != target:  # 排除目标变量本身
        for lag in range(1, 5):  # 创建1到4个时间点的滞后特征
            data[f"{feature}_lag_{lag}"] = data[feature].shift(lag)
            lagged_features.append(f"{feature}_lag_{lag}")
```

```
# 删除包含NaN值的行
data = data.dropna()
# 将滞后特征和目标变量分开
X = data[features + lagged_features]
y = data[target]
```

创建完了之后，我们使用 Pandas 的 `corr()` 函数计算特征及其滞后版本之间的皮尔逊相关系数，以评估它们之间的线性相关性，并绘制相关性热图，我们以2000_0_20201230和2003_0_20210615号两个患者为例进行了热图的绘制，我们选择这两个患者的原因是这两个患者是比较典型的例子，他们一个是没有采用basal胰岛素（餐时胰岛素）和bolus胰岛素（基础胰岛素）的（2000_0_20201230），而另一个是采用了basal胰岛素和bolus胰岛素的（2003_0_20210615）：



2000_0_20201230患者（未使用basal胰岛素和bolus胰岛素）的特征与其滞后版本之间的相关性热图



2003_0_20210615患者使用（basal胰岛素和bolus胰岛素）的特征与其滞后版本之间的相关性热图

我们发现无论患者是否使用了basal胰岛素和bolus胰岛素，其在用餐时摄入的Carbonhydrate都和insulin dose有一定的正相关。

然后对特征进行归一化之后利用随机森林模型获取特征的重要性。随机森林模型能够评估每个特征对模型预测的贡献。它通过测量每个特征在分裂节点时的贡献来计算特征重要性。重要性的度量通常基于特征在所有树中的分裂点上所减少的均方误差（对于回归任务）或基尼不纯度（对于分类任务）的总和。

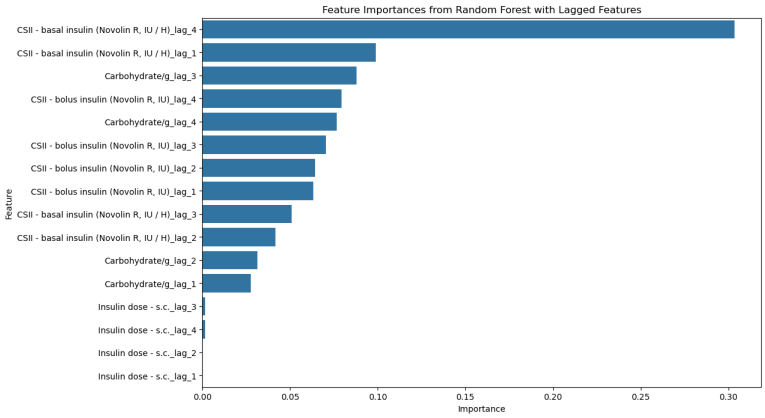
```
# 训练随机森林回归模型
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_scaled, y)
# 获取特征重要性
importances = model.feature_importances_
# 特征重要性排序
```

```python
feature_importances = pd.DataFrame(
    {"Feature": lagged_features, "Importance": importances}
    ).sort_values(by="Importance", ascending=False)
# 打印特征重要性
print(feature_importances)
```

以下为打印的结果和重要性的可视化：

	Feature	Importance
11	CSII - basal insulin (Novolin R, IU / H)_lag_4	0.303550
8	CSII - basal insulin (Novolin R, IU / H)_lag_1	0.098965
14	Carbohydrate/g_lag_3	0.088013
7	CSII - bolus insulin (Novolin R, IU)_lag_4	0.079228
15	Carbohydrate/g_lag_4	0.076622
6	CSII - bolus insulin (Novolin R, IU)_lag_3	0.070449
5	CSII - bolus insulin (Novolin R, IU)_lag_2	0.064371
4	CSII - bolus insulin (Novolin R, IU)_lag_1	0.063413
10	CSII - basal insulin (Novolin R, IU / H)_lag_3	0.050953
9	CSII - basal insulin (Novolin R, IU / H)_lag_2	0.041680
13	Carbohydrate/g_lag_2	0.031549
12	Carbohydrate/g_lag_1	0.027740
2	Insulin dose - s.c._lag_3	0.001579
3	Insulin dose - s.c._lag_4	0.001461
1	Insulin dose - s.c._lag_2	0.000229
0	Insulin dose - s.c._lag_1	0.000197

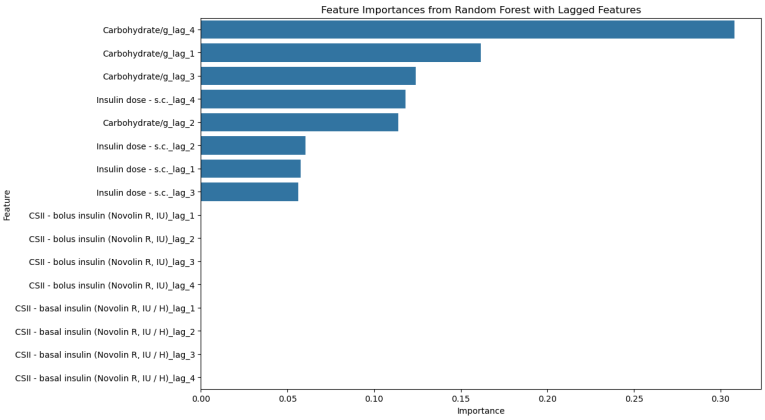
对2003_0_20210615患者而言特征重要性进行打印



2003_0_20210615号患者的特征重要性可视化

	Feature	Importance
15	Carbohydrate/g_lag_4	0.308067
12	Carbohydrate/g_lag_1	0.161572
14	Carbohydrate/g_lag_3	0.123905
3	Insulin dose - s.c._lag_4	0.118049
13	Carbohydrate/g_lag_2	0.114052
1	Insulin dose - s.c._lag_2	0.060507
0	Insulin dose - s.c._lag_1	0.057551
2	Insulin dose - s.c._lag_3	0.056298
4	CSII - bolus insulin (Novolin R, IU)_lag_1	0.000000
5	CSII - bolus insulin (Novolin R, IU)_lag_2	0.000000
6	CSII - bolus insulin (Novolin R, IU)_lag_3	0.000000
7	CSII - bolus insulin (Novolin R, IU)_lag_4	0.000000
8	CSII - basal insulin (Novolin R, IU / H)_lag_1	0.000000
9	CSII - basal insulin (Novolin R, IU / H)_lag_2	0.000000
10	CSII - basal insulin (Novolin R, IU / H)_lag_3	0.000000
11	CSII - basal insulin (Novolin R, IU / H)_lag_4	0.000000

对2000_0_20201230患者而言特征重要性进行打印



2000_0_20201230号患者的特征重要性可视化

经过综合考量，最终我们的动态特征选择CGM，Insulin dose - s.c，CSII - basal insulin (Novolin R, IU/H)_lag_4，CSII - basal insulin (Novolin R, IU/H)_lag_1，Carbonhydrate。

4. LSTM法预测血糖时间序列

4.1 模型训练

4.1.1 模型选择

在本项目中，我们选择使用长短期记忆网络（LSTM）来进行糖尿病患者血糖水平的预测。长短期记忆网络（Long Short-Term Memory, LSTM）是一种特殊类型的递归神经网络（Recurrent Neural Network, RNN），它能够学习和记忆长时间序列数据中的模式。LSTM网络通过引入一组“门”机制

（输入门、遗忘门和输出门）来控制信息的流动，从而解决了传统RNN在处理长时间序列数据时存在的梯度消失和梯度爆炸问题。

LSTM在处理时间序列数据方面具有天然优势，因为它能够捕捉数据中的时间依赖关系，并通过其门机制有效地记忆和利用过去的信息。此外，糖尿病患者的血糖水平变化受多种因素影响，包括饮食、药物、运动等，这些因素在时间上具有长时间的依赖关系。LSTM通过其设计可以有效地建模这些长时间的依赖关系，从而提高预测的准确性。并且，传统的RNN在处理长时间序列数据时容易出现梯度消失或梯度爆炸问题，导致模型难以训练，而LSTM通过其特殊的结构设计，能够缓解甚至解决这些问题，使得模型在训练过程中更加稳定和有效。因此，我们选择LSTM作为我们糖尿病预测的模型。

4.1.2 数据准备

4.1.2.1 特征提取

在数据准备阶段，我们从指定路径读取了一型糖尿病（T1DM）和二型糖尿病（T2DM）患者的总结数据，并分别添加了一个标识字段（type）表示糖尿病类型。我们还读取了每个患者的详细数据，并根据数据预处理中得到血糖与数据相关度提取了特定的时间序列特征（如CGM值、胰岛素剂量、饮食等）和静态特征（如年龄、体重、糖尿病病程等）。

我们最终提取的特征如下：

```
time_series_features = [
    'CGM (mg / dl)',
    'Insulin dose - s.c.',
    'CSII - bolus insulin (Novolin R, IU)',
    'Carbohydrate/g'
]
static_features = [
    'type','patient_id','Age (years)', 'Weight (kg)', 'BMI (kg/m2)', 'Duration of Diabetes (years)',
    'HbA1c (mmol/mol)', 'Fasting Plasma Glucose (mg/dl)', '2-hour Postprandial C-peptide (nmol/L)',
    'Fasting C-peptide (nmol/L)', 'Glycated Albumin (%)', 'Acute Diabetic Complications',
    'Diabetic Macrovascular Complications', 'Diabetic Microvascular Complications',
    'Comorbidities', 'Hypoglycemic Agents', 'Other Agents'
]
```

4.1.2.2 时间序列准备

在时间序列准备阶段，我们为每个患者创建长度为15的滑动窗口，将数据划分为多个时间步长序列。具体步骤如下：

滑动窗口：

- 将每个患者的时间序列数据分割成连续的长度为15的窗口。每个窗口代表模型的一个输入序列。
- 滑动窗口的步长为1，即每次移动一个时间步，生成新的窗口。

处理缺失值：

- 对于时间序列中的缺失值（NaN），我们使用中位数填充，以保证数据的完整性。
- 在创建窗口时，如果发现窗口内有缺失值，则用该特征在整个时间序列中的中位数进行填充。

我们在构建时间序列时特别注意了静态数据的协同，确保每一个时间序列都对应一个病人的静态数据。

```
def create_sequences(patient_id, patient_data, static_data,
time_series_features, static_features, target, time_steps):
    sequences = []
    targets = []
    static = static_data[static_features].values[0]
    # 填充静态数据中的 NaN 值
    if pd.isna(static).sum() > 0:
        nan_indices = np.where(pd.isna(static))[0]
        for idx in nan_indices:
            feature_name = static_features[idx]
            static[idx] = t1dm_medians[feature_name]
    for i in range(len(patient_data) - time_steps):
        seq = patient_data.iloc[i:i + time_steps][time_series_features].values
        label = patient_data.iloc[i + time_steps][target]

        # 填充时间序列数据中的 NaN 值
        if np.isnan(seq).sum() > 0:
            nan_indices = np.where(np.isnan(seq))
            for row, col in zip(*nan_indices):
                feature_name = time_series_features[col]
                seq[row, col] = patient_data[feature_name].dropna().median()
        # 填充目标数据中的 NaN 值
        if np.isnan(label):
            label = patient_data[target].mean()
        sequences.append((seq, static))
        targets.append(label)
    return sequences, targets
```

4.1.3 模型结构和训练流程

我们使用了一个包含多层LSTM单元的神经网络模型，该模型能够处理时序数据并生成预测。具体架构如下：

```python
def build_model(time_steps, time_series_features, static_features):
    input_time_series = Input(shape=(time_steps, len(time_series_features)))
    input_static = Input(shape=(len(static_features),))
    # LSTM层
    lstm_out = LSTM(units=256, return_sequences=True, kernel_initializer=HeNormal())(input_time_series)
    lstm_out = BatchNormalization()(lstm_out)
    lstm_out = Dropout(0.4)(lstm_out)
    lstm_out = LSTM(units=256, return_sequences=True, kernel_initializer=HeNormal())(lstm_out)
    lstm_out = BatchNormalization()(lstm_out)
    lstm_out = Dropout(0.4)(lstm_out)
    lstm_out = LSTM(units=128, kernel_initializer=HeNormal())(lstm_out)
    lstm_out = BatchNormalization()(lstm_out)
    lstm_out = Dropout(0.4)(lstm_out)
    concat = concatenate([lstm_out, input_static])
    dense_out = Dense(512, activation='relu', kernel_initializer=GlorotUniform())(concat)
    dense_out = BatchNormalization()(dense_out)
    dense_out = Dropout(0.4)(dense_out)
    dense_out = Dense(256, activation='relu', kernel_initializer=GlorotUniform())(dense_out)
    dense_out = BatchNormalization()(dense_out)
    dense_out = Dropout(0.4)(dense_out)
    dense_out = Dense(128, activation='relu', kernel_initializer=GlorotUniform())(dense_out)
    dense_out = BatchNormalization()(dense_out)
    dense_out = Dropout(0.4)(dense_out)
    output = Dense(1, kernel_initializer=GlorotUniform())(dense_out)
    model = Model(inputs=[input_time_series, input_static], outputs=output)
    model.compile(optimizer=tf.keras.optimizers.AdamW(learning_rate=0.0001), loss='mean_squared_error', metrics=['mae'])
    return model
```

我们使用多层LSTM来捕捉时间序列数据中的长时间依赖关系。每层LSTM后接Batch Normalization和Dropout，以稳定训练并防止过拟合。

在全连接层，我们将LSTM的输出与静态特征结合，经过多层全连接层进一步处理。每层全连接层后接Batch Normalization和Dropout，以增强模型的泛化能力。

最后，我们在最后一层Dense层输出一个标量，作为血糖水平的预测值。

我们选择使用 `ReduceLROnPlateau` 回调函数来动态调整学习率，以提高训练效果。训练数据和验证数据的划分比例为80:20。

训练过程中的主要步骤如下：

1. **创建模型并编译**：

- 使用定义的LSTM模型架构，并编译模型，设置优化器为 `AdamW` ，损失函数为均方误差（MSE），评估指标为平均绝对误差（MAE）。

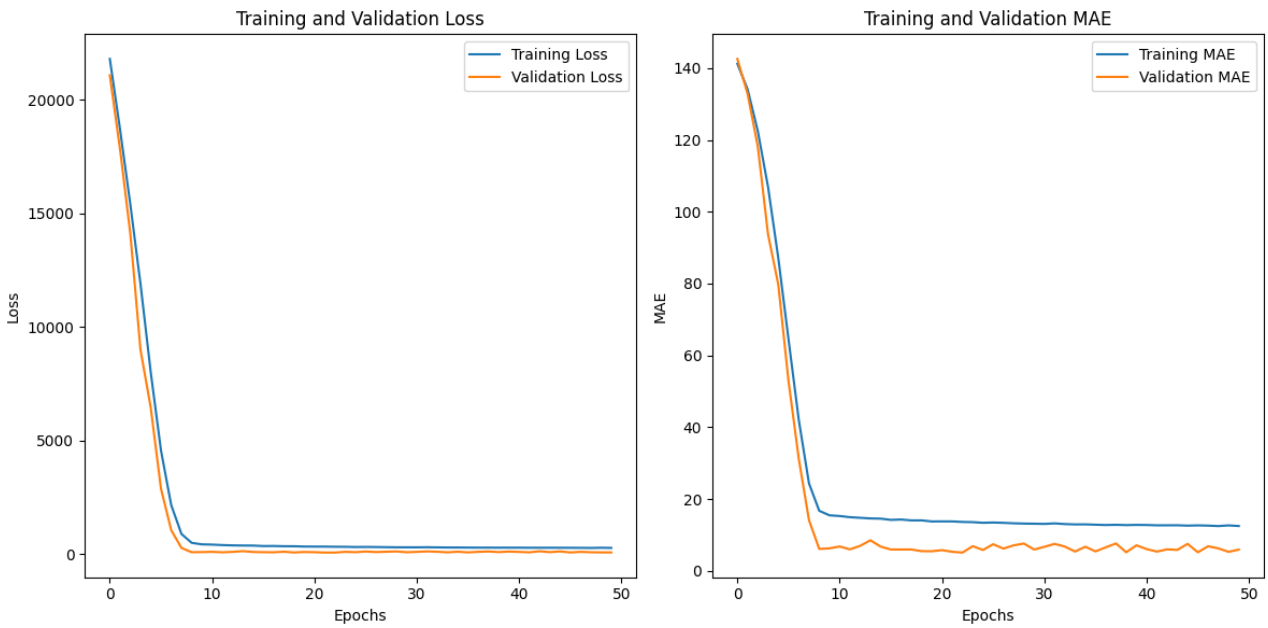
2. **数据划分**：
   - 将数据分为训练集和验证集，训练集占80%，验证集占20%。
3. **训练模型**：
   - 使用训练集进行模型训练，并在训练过程中使用验证集进行验证。通过回调函数动态调整学习率以优化训练效果。
4. **保存模型**：
   - 在训练完成后，保存最终的模型以备后续使用。

4.2 训练结果

4.2.1 模型结果

```
Mean Squared Error (MSE): 71.91808362785893
Mean Absolute Error (MAE): 5.904551408513383
```

我们模型最终的MSE为71.91，MAE为5.90455。仍有提高的空间。



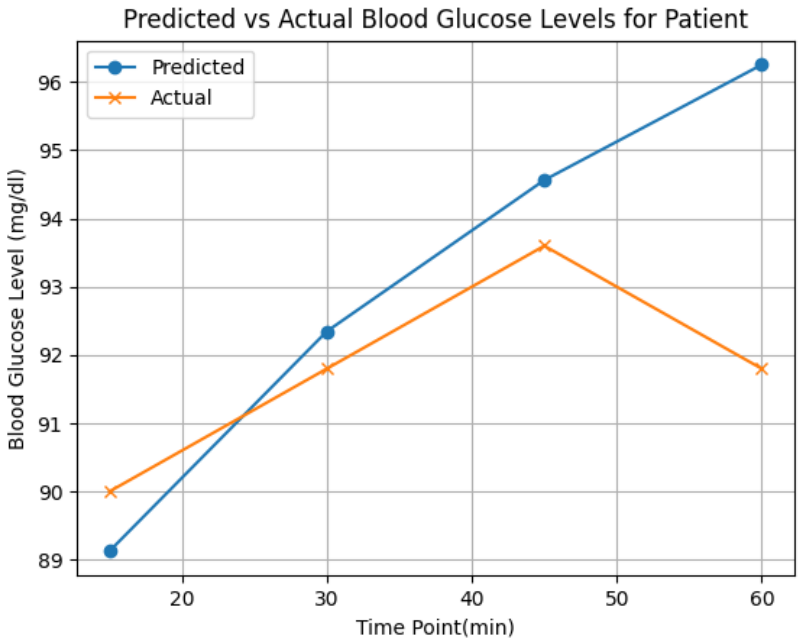
损失（Loss）和平均绝对误差（MAE）的示意图

这是我们的损失（Loss）和平均绝对误差（MAE）的示意图。从图中可以看到，Loss和MAE在训练初期迅速下降，在大约10个epoch之后趋于平稳。这表明模型在这段时间内快速学习并适应了训练数据，同时验证集上的表现也趋于稳定，表明模型没有发生过拟合或欠拟合的问题。

左侧图展示了训练和验证的损失曲线。可以观察到，训练损失和验证损失在初期有明显的下降趋势，并在较低的数值上稳定下来，表明模型训练过程收敛良好。

右侧图展示了训练和验证的MAE曲线。类似于损失曲线，训练MAE和验证MAE也在初期迅速下降，并趋于平稳，说明模型在训练过程中不断优化参数，逐渐提高预测精度。

4.2.2 预测结果



我们随机选取了一段现实中的数据集进行预测，由于我们的模型输出维度只有1，我们采用了逐步预测的形式。从图中可以看到，模型的预测结果（Predicted）和实际值（Actual）在短时间内具有较高的一致性，但随着时间的推移，预测值和实际值之间的差距逐渐增大。

在较短的时间段内，模型的预测结果与实际值非常接近，说明模型能够有效地捕捉短时间内的血糖变化趋势。这表明LSTM模型在处理较短时间序列数据时具有较好的性能，能够准确地预测血糖水平。

然而，随着时间的推移，模型的预测结果与实际值之间的误差逐渐增大。图中可以看到，在第40到60分钟之间，模型的预测值显著高于实际值。

一方面，由于在预测的时候我们只输入最开始的那一组数值，而在预测期间的这一个小时中用户可能进行了饮食、注射药物等活动，这些是我们没有办法在其预测的过程中捕捉的，但是这些因素在长时间内对血糖水平的影响较大，所以会导致和实际结果产生误差。

另一方面，我们使用预测值参与下一次的预测，会大大增大最终预测的误差，因此，每一次预测中的小误差都会在后续的预测中被放大，导致整体预测误差的逐步积累和扩大。这种误差累积现象使得模型在长时间预测中的准确性下降。

5. Transformer法预测血糖时间序列

鉴于T1DM患者和T2DM患者的数据存在较大波动上的差异，我们选择先将T1DM和T2DM患者的数据按相同比例混合进行预训练，然后再分别在T1DM和T2DM数据上进行微调。这样做的依据是，通过预训练阶段的混合数据处理，可以为模型提供更全面的特征表示，从而提高模型的泛化能力。而在微调阶

段分别处理T1DM和T2DM数据，可以使模型更好地适应两类患者的特定数据特点，进一步提升预测精度。

我们之后发现微调后两个模型都有提升，比合起来的预训练好，并且将基于T2DM微调的模型应用于T1DM数据会发现预测的水平较预训练模型有很大降低，侧面证明了T1DM和T2DM的血糖波动模式有很大差异和微调的可行性。

```
Average MAE at time point 1: 10.145940486636004
Average MAE at time point 2: 11.114699643624466
Average MAE at time point 3: 13.15510129346707
Average MAE at time point 4: 15.55509197061155
Overall MAE: 12.492708348584772
Average MSE at time point 1: 191.3012215237218
Average MSE at time point 2: 223.15465515479406
Average MSE at time point 3: 267.8456661589419
Average MSE at time point 4: 313.04296064774957
Overall MSE: 248.83612587130187
```

上图为T2DM微调模型在T1DM数据上的表现，可以和之后的预训练模型和微调模型的MAE和MSE数据做对比。

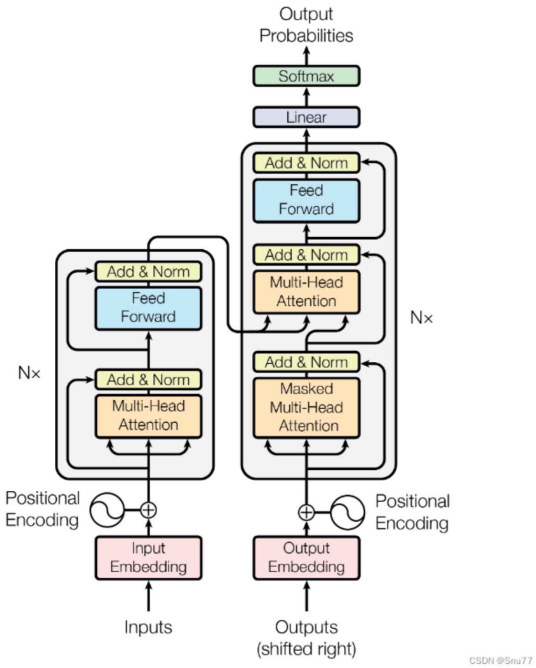
5.1 模型训练

5.1.1 模型选择

我们在使用了LSTM进行了血糖时间序列的预测之后又使用了transformer进行实验，这种模型最初是为了处理自然语言处理（NLP）任务而设计的，但由于其独特的架构和能力，它也被用于时间序列分析。

Transformer应用于时间序列分析中的基本思想是：Transformer 在时间序列分析中的应用核心在于其自注意力机制，这使其能够有效捕捉时间序列数据中的长期依赖关系。

通过并行处理能力和位置编码，Transformer 不仅提高了处理效率，而且确保了时间顺序的准确性。其灵活的模型结构允许调整以适应不同复杂度的数据，而编码器-解码器架构则特别适用于预测未来的时间点。



Transformer模型架构

5.1.2 数据准备

对上面挑选好的特征，我们需要进行以下数据准备工作：

1. 滑动窗口处理

首先，利用滑动窗口原理对数据进行处理。窗口大小设为 n，滑动步数为1。具体步骤如下：

- 使用前 n 个时间点的数据来进行预测。
- 将这 n 个时间点中的后4个时间点的血糖值作为标签。
- 依次滑动窗口，最终得到的数据集格式如下：
  - `x` 的第一维度代表数据条数，第二维度代表预测窗口大小，第三维度代表特征。
  - `y` 的第一维度代表数据条数，第二维度代表之后四个时间点的血糖值。

```
All X shape: (127170, 4, 15)
All y shape: (127170, 4)
Sample X: [[[ 1.13400000e+02  0.00000000e+00  3.00000000e-01  0.00000000e+00
   -9.44089020e-01 -3.29690645e-01  1.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  1.00000000e+01  3.52800000e+02
    3.63955000e+02  1.15311000e+02  4.07000000e+01]
  [ 1.24200000e+02  0.00000000e+00  3.00000000e-01  0.00000000e+00
   -9.63630453e-01 -2.67238376e-01  1.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  1.00000000e+01  3.52800000e+02
    3.63955000e+02  1.15311000e+02  4.07000000e+01]
  [ 1.29600000e+02  0.00000000e+00  3.00000000e-01  6.65100000e+01
   -9.79045472e-01 -2.03641751e-01  1.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  1.00000000e+01  3.52800000e+02
    3.63955000e+02  1.15311000e+02  4.07000000e+01]
  [ 1.42200000e+02  0.00000000e+00  3.00000000e-01  0.00000000e+00
   -9.90268069e-01 -1.39173101e-01  1.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00  1.00000000e+01  3.52800000e+02
    3.63955000e+02  1.15311000e+02  4.07000000e+01]]]
Sample y: [[156.6 162.  163.8 165.6]]
```

2. 时间顺序检查

为了确保文件中的时间按顺序排列且每个时间点相差15分钟，我们编写了代码进行检查。发现有个别数据点间隔为30分钟，将其视为噪音并不做处理。

3. 时间特征处理

处理时间特征时，将时间的日期部分去除，转换为一天中的分钟数。由于血糖水平具有明显的24小时周期性，但季节性影响基本可以从静态特征中看出近期的血糖走势，并且我们预测的是短期的血糖变化，因此可以忽略季节性因素。接着，将分钟数进行了sin和cos形式编码，以更好地体现数据中的周期性特征。

4. 数据集平衡

预训练模型所用的数据集结合了T1DM和T2DM数据。具体处理方法如下：

- 对T1DM的数据进行上采样。
- 对T2DM的数据进行下采样。
- 这样可以确保在训练过程中，这两类病人的数据比例基本一致。

```
Data proportion from dataset/T1DM: 46.57%
Data proportion from dataset/T2DM: 53.43%
```

通过上述步骤，我们完成了数据的准备工作，确保数据集能够有效地用于模型训练和预测。

5.1.3 模型结构和训练过程

```
class TransformerModel(nn.Module):
    def __init__(
        self,
        input_dim,
        output_dim,
        d_model=32,
        nhead=4,
        num_encoder_layers=3,
        num_decoder_layers=3,
        dim_feedforward=128,
        dropout=0.1,
    ):
        super(TransformerModel, self).__init__()
        self.d_model = d_model
        self.embedding = nn.Linear(input_dim, d_model)
        self.transformer = nn.Transformer(
            d_model,
            nhead,
            num_encoder_layers,
            num_decoder_layers,
            dim_feedforward,
            dropout,
            batch_first=True,
        )
        self.decoder_embedding = nn.Linear(output_dim, d_model)
        self.fc_out = nn.Linear(d_model, output_dim)
        self.init_weights()

    def init_weights(self):
        # 使用Xavier初始化适用于所有线性层
        nn.init.xavier_uniform_(self.embedding.weight)
        nn.init.xavier_uniform_(self.fc_out.weight)
        nn.init.xavier_uniform_(self.decoder_embedding.weight)

    def forward(self, src, tgt):
        src = self.embedding(src)
        tgt = self.decoder_embedding(tgt)
        output = self.transformer(src, tgt)
        output = self.fc_out(output)
        return output

```

模型结构

我们的模型是一个基于Transformer架构的神经网络，用于处理时间序列数据。模型的详细结构如下：

1. **输入层和嵌入层**：
   - `input_dim`：输入特征的维度，模型将其映射到更高维度的表示（`d_model`）。
   - `embedding`：使用线性层将输入特征转换为`d_model`维度的嵌入向量。
2. **Transformer模块**：
   - 使用`nn.Transformer`模块，包括编码器和解码器部分。
   - `d_model`：嵌入向量的维度。
   - `nhead`：多头注意力机制的头数。
   - `num_encoder_layers`：编码器的层数。
   - `num_decoder_layers`：解码器的层数。
   - `dim_feedforward`：前馈神经网络的维度。
   - `dropout`：Dropout率。
3. **解码器嵌入层和输出层**：
   - `decoder_embedding`：将目标值（标签）转换为`d_model`维度的嵌入向量。
   - `fc_out`：最终的线性层，将Transformer的输出映射为目标维度（`output_dim`）。
4. **权重初始化**：
   - 使用Xavier初始化方法对所有线性层的权重进行初始化。

前向传播过程

1. 输入数据（`src`）经过嵌入层转换为`d_model`维度的向量。
2. 目标数据（`tgt`）经过解码器嵌入层转换为`d_model`维度的向量。
3. 经过Transformer模块进行编码和解码，得到输出向量。
4. 输出向量经过最终的线性层，得到预测值。

```
def train(
    model,
    train_loader,
    val_loader,
    criterion,
    optimizer,
    scheduler,
    num_epochs=20,
):
```

```
    model = model.to(device)
    criterion = criterion.to(device)

    best_val_loss = float("inf")  # 用于保存最佳验证损失

    step = 0
    for epoch in range(num_epochs):
        model.train()
        epoch_loss = 0

        print("Size of the loader: ", len(train_loader))
        for src, tgt in tqdm(train_loader):
            src = src.to(device)
            tgt = tgt.to(device)
            optimizer.zero_grad()
            tgt_input = torch.zeros((tgt.size(0), 1, 1), device=src.device)
            loss = 0
            for t in range(tgt.size(1)):
                output = model(src, tgt_input)
                loss += criterion(output[:, -1, :], tgt[:, t : t + 1])
                tgt_input = torch.cat((tgt_input, output[:, -1:, :]), dim=1)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
            step += 1
            writer.add_scalar("Training Loss", loss.item(), step)
        scheduler.step()
        avg_train_loss = epoch_loss / len(train_loader)
        print(f"Epoch {epoch+1}, Training Loss: {avg_train_loss}")
        # 验证集上的损失
        model.eval()
        val_loss = 0
        with torch.no_grad():
            for src, tgt in val_loader:
                src = src.to(device)
                tgt = tgt.to(device)
                tgt_input = torch.zeros((tgt.size(0), 1, 1), device=src.device)
                loss = 0
                for t in range(tgt.size(1)):
                    output = model(src, tgt_input)
                    loss += criterion(output[:, -1, :], tgt[:, t : t + 1])
                    tgt_input = torch.cat((tgt_input, output[:, -1:, :]),
dim=1)
                val_loss += loss.item()
        avg_val_loss = val_loss / len(val_loader)
        print(f"Epoch {epoch+1}, Validation Loss: {avg_val_loss}")
        writer.add_scalar("Validation Loss", avg_val_loss, epoch)  # 记录验证损失
```

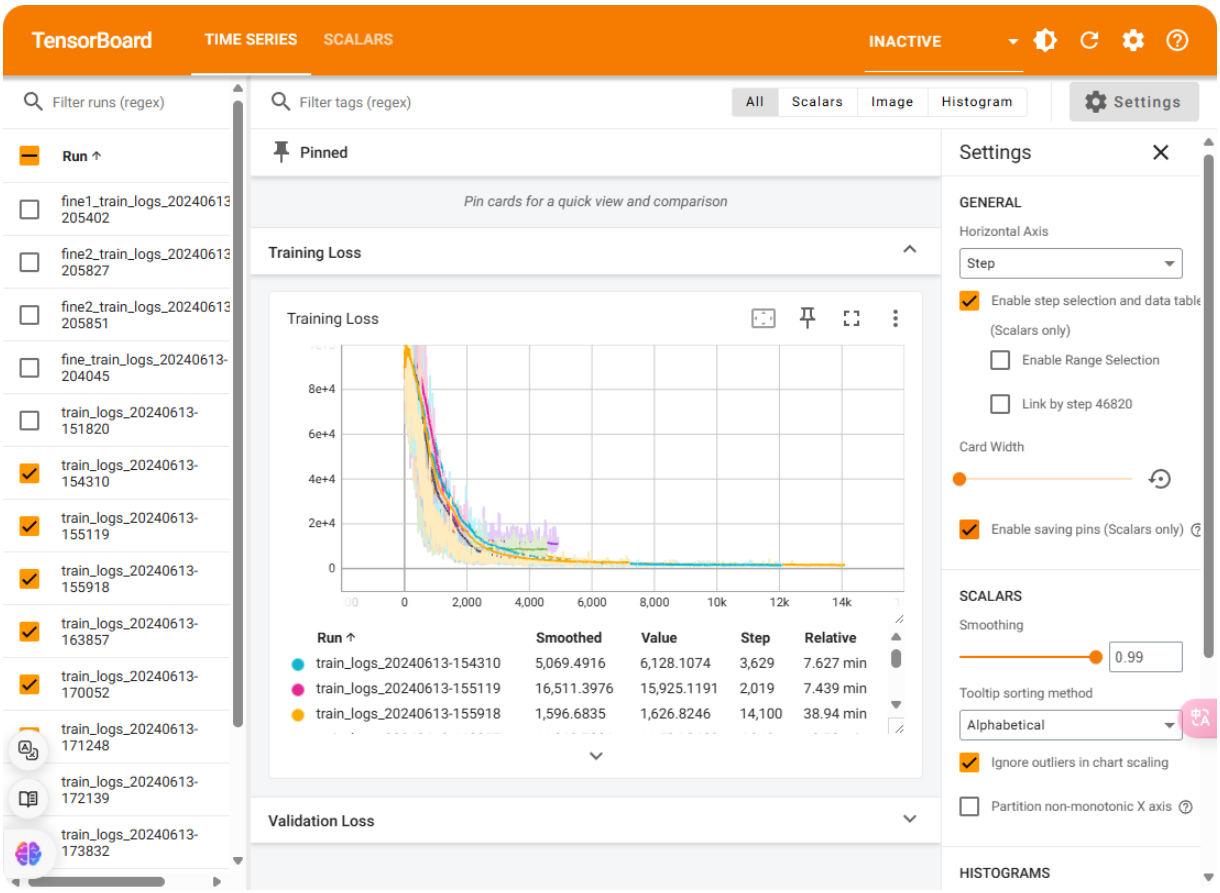
```
56
57          # 如果模型在验证集上表现更好，保存模型
58          if avg_val_loss < best_val_loss:
59              best_val_loss = avg_val_loss
60              torch.save(model.state_dict(),
    f"models/best_model_{timestamp}.pth")
61              print("Model Saved")
```

训练过程

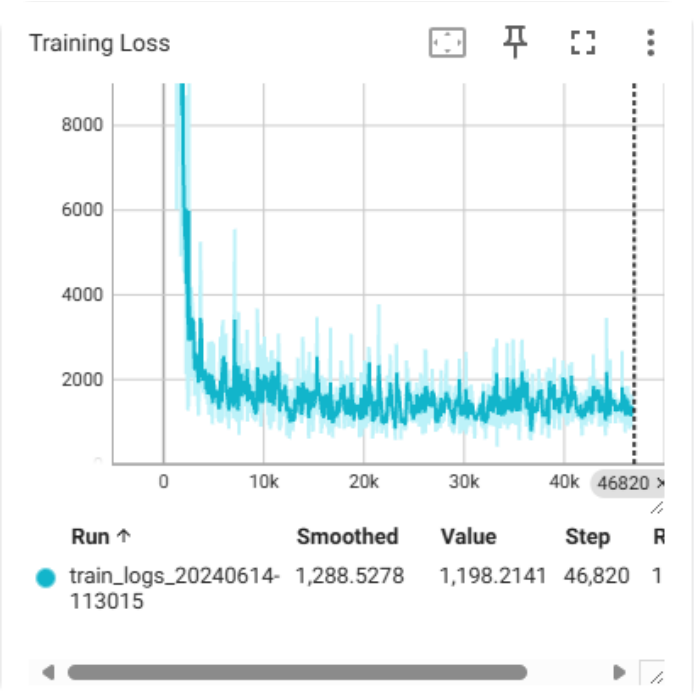
1. **数据加载**：
   - 使用 `getLoader` 函数获取训练和验证数据加载器。(我们的数据集以9：1划分训练集和验证集)
2. **训练循环**：
   - 每个epoch中，对每个批次的数据进行前向传播、计算损失、反向传播和参数更新。
   - 使用滑动窗口的方法，逐步预测目标时间点的血糖值，累加要预测的四个时间点的损失值并更新。
3. **学习率调度与损失函数**：
   - 使用余弦退火学习率调度器（CosineAnnealingLR）调整学习率。
   - 优化器采用Adam，并且使用了L2正则化。
   - 预训练中损失函数选择MSE，因为预训练的目标是抓住大体特征，所以对较大误差要给予更大的惩罚，在微调的时候我们选择MAE，因为对或大或小的误差都有线性的关注度，比较精确。
6. **模型验证**：
   - 在每个epoch结束后，使用验证集数据计算验证损失，并记录下来。
   - 若当前epoch的验证损失优于之前的最佳验证损失，保存模型权重。
7. **模型保存**：
   - 保存验证损失最小的模型，以便后续使用。
8. **记录损失值**：
   - 使用TensorBoard记录训练和验证过程中的损失值，便于可视化和分析。

5.2 训练结果

5.2.1 预训练结果



如上，在微调阶段我们调参做了多次实验，最终得到效果较好的模型训练和验证曲线如下：




```
Average MAE at time point 1: 6.671402213150021
Average MAE at time point 2: 7.95160774060437
Average MAE at time point 3: 9.875176345176119
Average MAE at time point 4: 12.893926611774045
Overall MAE: 9.348028227676139
Average MSE at time point 1: 63.227197731966854
Average MSE at time point 2: 141.51063435010627
Average MSE at time point 3: 207.13164857086497
Average MSE at time point 4: 300.7196694878081
Overall MSE: 178.14728753518654
```

这张图显示了模型在不同时间点的平均绝对误差（MAE）和平均平方误差（MSE），以及总体的MAE和MSE。由上面的结果可以看出，Transformer模型在预测性能上并未优于LSTM模型，具体分析将在后文进行。同时，我们还注意到，随着时间点的推移，预测变得更加困难。

在较短的时间段内，模型的预测结果与实际值非常接近，说明模型能够有效地捕捉短时间内的血糖变化趋势。然而，随着时间的推移，模型的预测结果与实际值之间的误差逐渐增大。

造成这一现象的原因有以下几点：

1. **外部因素影响**：

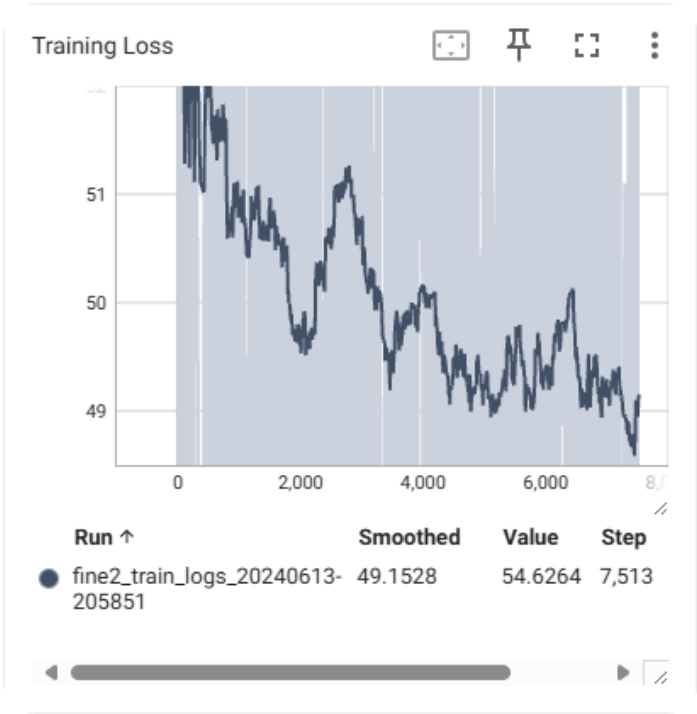
在预测过程中，我们仅输入了最初的那组数值，而在预测期间的这一个小时内，用户可能进行了饮食、注射药物等活动。这些活动无法在模型预测时被捕捉到，但它们对血糖水平的影响较大，从而导致预测结果与实际值之间产生误差。

2. **误差累积效应**：

我们在每一次预测中使用了前一次的预测值，这会大大增大最终预测的误差。每一次预测中的小误差都会在后续的预测中被放大，导致整体预测误差逐步积累和扩大。这种误差累积现象使得模型在长时间预测中的准确性下降。

5.2.2 微调结果

T1DM



T2DM

我们在预训练模型的基础上对T1DM和T2DM的患者进行了微调，最后微调的结果如下：

```
Average MAE at time point 1: 5.807140617395106
Average MAE at time point 2: 7.765837631397089
Average MAE at time point 3: 9.366602121596927
Average MAE at time point 4: 12.766210924759843
Overall MAE: 8.926447823787242
Average MSE at time point 1: 45.91198685878197
Average MSE at time point 2: 101.14474750437948
Average MSE at time point 3: 159.27308845327516
Average MSE at time point 4: 237.34657947705838
Overall MSE: 135.91910057337375
```

T1DM

```
Average MAE at time point 1: 5.1485445535528
Average MAE at time point 2: 7.3331997285383625
Average MAE at time point 3: 8.529456849913943
Average MAE at time point 4: 11.224744900504216
Overall MAE: 8.05898650812733
Average MSE at time point 1: 37.05506862138204
Average MSE at time point 2: 91.74757838637348
Average MSE at time point 3: 138.54299471619544
Average MSE at time point 4: 202.3090953938494
Overall MSE: 117.41368427945008
```

T2DM

可以看出，经过微调后的模型在整体上比预训练的模型表现更好，但对于T1DM患者的改进效果并不明显，而对于T2DM患者则更加显著。这可能是由于以下原因：

1. **数据量差异**：

T2DM患者的数据量较多，在预训练时为了保持数据比例相同，我们对T2DM的数据进行了下采样。因此，在微调过程中，T2DM患者的新数据加入训练，使模型在处理T2DM数据时表现出更好的适应性和精度。

2. **T1DM数据的限制**：

相比之下，T1DM患者的数据量较少，因此在微调过程中，模型能够利用的新信息有限，这导致了改进效果不如T2DM患者明显。

综上所述，微调后的模型在处理T2DM患者的数据时表现出更显著的改进，主要原因是T2DM数据量较大且在微调过程中有新数据的加入，而T1DM患者由于数据量相对较少，改进效果不如T2DM患者明显。

6. 总结

6.1 项目实施中遇到的困难

6.1.1 数据集制作中遇到的问题

在最初的LSTM模型训练过程中，我们将T1DM和T2DM数据混合在一起，并通过额外的特征标识每条数据所属的类别。参考了许多论文后，我们发现大多数研究都是将T1DM和T2DM数据分开处理，于是我们考虑是否也应该分开训练。

在之后分析数据集时，我们发现OhioT1DM数据集大约有5.6万条数据，而我们的T1DM数据有1.9万条，T2DM数据有10.6万条。最初我们计划在OhioT1DM数据集上进行迁移学习，并考虑对每个个体进行微调，以实现个性化的血糖预测。然而，我们注意到OhioT1DM数据集的静态信息非常少（例如抽烟、喝酒、病史等），而我们数据集的静态信息较为丰富。因此，如果从OhioT1DM数据集进行预训练，可能难以捕捉病人的个性化静态信息，这使得对每个人的微调显得尤为重要。

然而，OhioT1DM数据集与我们的数据集存在较大差异，主要体现在缺乏静态信息，这使得迁移学习变得非常复杂。否则，我们的数据集将无法利用静态特征，从而导致不一致的问题。基于此，我们搁置了OhioT1DM的迁移学习计划，转而考虑混合T1DM和T2DM数据进行预训练，以获得泛化的血糖趋势预测模型，然后在各自的数据集上进行微调，分别训练两个模型。

然而，在使用分开的数据集进行第二个Transformer预测实验时，结果比原本的LSTM模型差。尚未确定这种情况是由于分开数据集的原因还是Transformer模型本身的原因。我们接下来会分析Transformer模型可能的劣势，并考虑在LSTM模型上进行分开的实验，效果可能会更好。但由于时间限制，我们暂时无法进行更多的对比实验。

6.1.2 数据预处理中遇见的问题

1. 在动态特征表中，不同表间同一字段名的格式可能不一致，导致了合并处理时多个同名但不同格式的字段的出现。此外，少量字段出现了字段名和内容不匹配的情况。这种问题给数据合并处理带来了困难，如出现了无法处理的文本值等。为了解决这个问题，编写了Python脚本来排查格式问题，统一了命名格式。此外，对于少量错误进行了手动修正。

2. 在进行特征相关性分析时，初期绘制的热力图显示大部分特征与CGM等测量数据的相关性并不高。考虑到可能是因为动态序列具有延迟为了解决这个问题，对原始的CGM数据进行了进一步的计算，获取了滚动均值、小时变化速率等特征，以排除短时间内的小幅波动，从而更准确地评估特征与血糖变化之间的相关性。
3. 关于上述相关性分析，由于各个和血糖相关性不高的特征之间又存在很明显的相关性，我们初步考虑到特征之间可能存在的延迟效应以及协变量混淆/共线性问题（比如说吃饭之前会吃药，这两者效果抵消了，而且这两者都是对之后一段时间才会有影响，并非瞬间的时刻）。为此，我们通过创建延迟特征并采用更复杂的随机森林模型来评估特征的重要性。这样可以更好地捕捉特征的延迟效应以及它们之间非线性的相关性。
4. 在观察静态数据和动态数据的过程中，由于特征太多导致预处理步骤繁琐又困难，考虑到我们缺乏医学知识，所以我们运用了三个顶级的大语言模型来帮我们初步筛选所需的特征，以减少我们数据预处理的工作量。

6.1.3 模型训练中遇见的问题

在使用LSTM和Transformer进行血糖水平预测时，我们面临以下几个困难：

1. **数据质量和缺失值**：

   在我们的数据集中出现了很多缺失值和噪声值，这会影响模型的训练效果。尽管我们在数据准备阶段使用了中位数填充方法处理缺失值，但这仍可能导致信息丢失和预测不准确。数据集的质量会大大影响预测的效果。

2. **特征选择和特征工程**：

   虽然LSTM/Transformer模型能够自动识别时间序列中的特征，但输入的原始特征仍然会影响模型的性能。如何选择和工程这些特征以最大化模型的预测能力是一个复杂的问题。尽管LSTM/Transformer具有一定的特征学习能力，但输入的原始特征质量不高可能会降低模型的效果。

   我们通过手动选出一些相关度比较高的属性，来提高预测性能和降低计算量，但是仍无法彻底解决这一问题，需要大量得实验比对。

3. **时间序列长度和窗口大小**：

   确定合适的时间步长和窗口大小对于捕捉长时间依赖关系至关重要。如果窗口太短，模型可能无法捕捉到足够的信息；如果窗口太长，模型的复杂性和计算成本将显著增加。

   在LSTM模型中，我们选择使用长度为15的时间步长窗口，因为它在捕捉足够信息的同时，保持了模型的复杂性在可控范围内。通过实验，我们发现15的窗口大小在保持信息完整性和计算效率之间取得了相对的平衡。

4. **模型复杂性和训练时间**：

   LSTM模型由于其复杂的结构和参数量大，训练时间较长，且需要大量计算资源。模型的复杂性也增加了过拟合的风险，因此需要仔细调整模型参数和正则化方法。

在训练过程中，我们也通过不断调整学习率和模型结构来降低风险，但平衡时间，空间和训练效果仍然是一个比较困难的点。

5. **个体差异**：

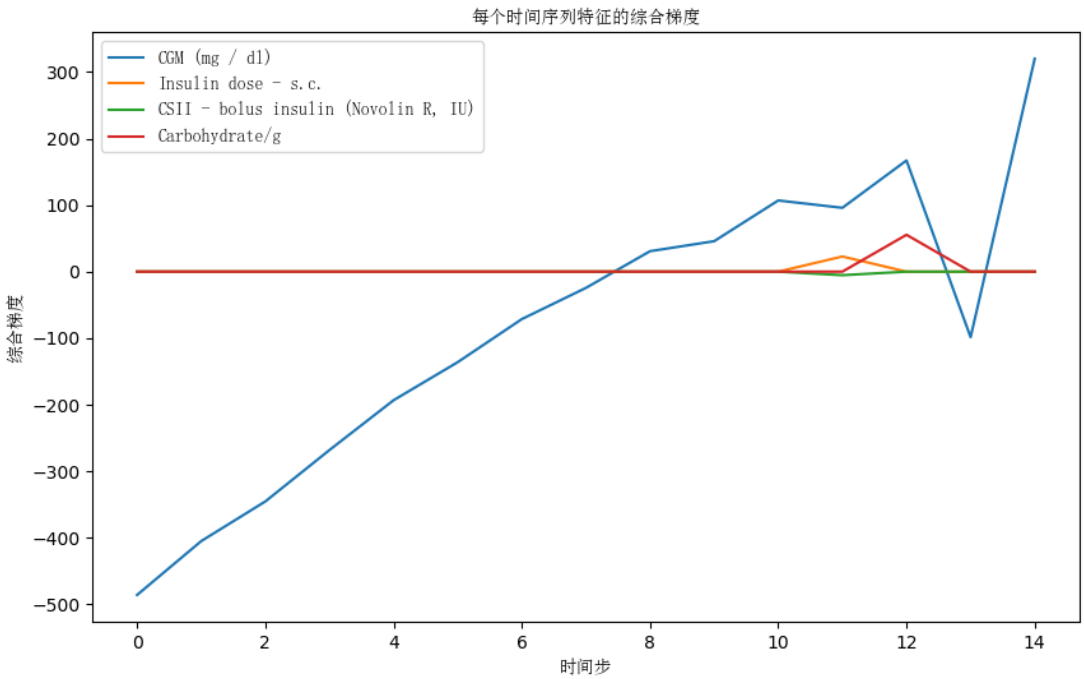
   糖尿病患者个体之间存在显著差异，包括生理特征、生活习惯和治疗方案等。这些个体差异使得模型需要具备较强的泛化能力，以便在不同患者之间有效应用。

6. **T1DM和T2DM数据集的样本数存在很大差异**：

   我们的T1DM数据集一共有有1w9条左右的记录，而T2DM有10w6条左右，而且我们的模型在预测T2DM患者数据的时候也会好一点，说明T1DM数据集还是样本数比较少，可以考虑之后利用ohioT1DM大概5w6条的样本数据（因为ohio是5min记录一次的，转成我们15min记录一次就是5w6条）进行补充，也可以通过数据增强方法来增强数据（比如重复过采样、加入高斯白噪声、使用TimeGAN生成合成少数样本和Mixup方法。）

6.2 特征的影响力分析

我们对特征对最终预测结果的影响进行了综合梯度分析。以下是我们在一次实验中的预测结果梯度分析示例。在这次实验中，我们选取了一位同时包含多个动态数据的病人信息进行预测，并将分析结果绘制成图表列在下面。在这次实验中，我们只选择了一些动态数据进行分析，包括 CGM、Insulin dose - s.c.、bolus insulin 和 carbohydrate。



从图中可以看出，不同时间序列特征的综合梯度有较大差异。以下是一些详细分析：

1. **CGM (mg/dl)**：
   - 该特征的综合梯度变化非常大，特别是在前几个时间步骤中。这表明血糖监测值（CGM）对模型预测的影响较大，且这种影响在不同时间步长上有显著波动。

    - 在时间步14时，综合梯度急剧上升，这可能表示在预测下一个时间点时，当前时间步的CGM值极为重要。总的来说，CGM是一个关键的动态特征，它对预测的贡献在整个时间序列中都表现得非常显著。
2. **Insulin dose - s.c.** 和 **Carbohydrate/g**：
    - 这两个特征的综合梯度值相对较小，且大部分时间步的综合梯度值几乎为零。这表明这些特征在大多数时间步中对预测的贡献很小。
    - 然而，在特定的时间步（如刚吃完饭或刚注射胰岛素后的时间步），这些特征的综合梯度有小幅上升，表明这些特征在这些特定时间点上可能对模型预测有一定的影响。
3. **CSII - bolus insulin (Novolin R, IU)**：
    - 这个特征的综合梯度在大部分时间步中接近于零，说明对模型预测的影响很小。这可能是因为在这个具体的数据集中，这个特征没有显著变化，或者模型没有正确地捕捉到这个特征的影响。
    - 需要进一步分析和实验来确定该特征是否需要保留或是否需要在数据预处理阶段进行更多的处理。

综合分析

- **CGM (mg/dl)** 是最重要的动态特征，对模型预测的影响最大。其综合梯度的显著波动表明，这个特征在整个时间序列中的每个时间步都很关键。
- **Insulin dose - s.c.** 和 **Carbohydrate/g** 尽管在某些特定时间步有一定的贡献，但总体来看其影响较小。可能需要在模型训练和特征选择过程中进一步优化这些特征的处理方式。
- **CSII - bolus insulin (Novolin R, IU)** 的影响最小，几乎可以忽略。可能需要在进一步的实验中验证其重要性，或者考虑在特征选择过程中剔除该特征。

这种分析可以帮助我们在特征选择过程中进行优化，进一步提升模型的预测性能。在后续的实验中，可以考虑以下几点：

- 增加对CGM (mg/dl)特征的关注，可能通过特征工程增加更多与血糖监测值相关的衍生特征。
- 对于 **Insulin dose - s.c.** 和 **Carbohydrate/g** 特征，可以尝试不同的特征工程方法，增加其对模型的贡献。
- 重新评估 **CSII - bolus insulin (Novolin R, IU)** 特征的必要性，考虑在模型训练中剔除该特征，简化模型结构。

6.3 针对模型选择方面的思考

在本次作业中我们采用了两种方式进行了实验，分别是LSTM和Transformer，我们发现虽然Transformer是一种可以用来处理序列的模型，并且在NLP问题中表现出良好的效果，但是在预测时间序列上的效果并不一定比传统的时间序列模型要来的精准，可能有以下几点原因：

1. **Transformer 模型中对输入的公平处理可能会阻碍时间序列任务**

Transformer 无法看到输入的顺序。为了解决这个缺点，我们通常会在输入序列中添加时间戳信息来明确顺序。无论数据有多旧，Transformer模型都会公平地对待这些输入。这个事实意味着，从 Transformer 模型的角度来看，数据集中最旧的数据与最新的数据一样重要。当然，Transformer 模型可以在注意力架构中学习数据的重要性。但是在学习过程开始时，所有输入都会被公平对待。这个特性是有利的，也是Transformer 模型能够在自然语言任务中表现良好的原因。但对于时间序列任务而言，这可能会导致糟糕的结果，因为数据越新，就越关键。我们也尝试过采用位置编码（虽然我们的数据中本身就有时间信息），寄希望Transformer可以捕捉每个时间点的位置信息，但是最后效果还是没有LSTM好，可能需要更多的训练技巧和实验支持。

2. **与 Transformer模型相比，更简单的模型更适合时间序列任务**

正如之前提到的，Transformer 模型在 NLP 任务中表现更好。这一事实意味着 Transformer 模型对于时间序列任务来说太复杂了。一般来说，NLP 任务比时间序列任务复杂得多。因此，在解决时间序列问题时，不需要庞大而复杂的模型。更简单的模型更好。Transformer更适合处理特征维数很大的数据，因为Transformer模型的核心是自注意力机制（Self-Attention Mechanism），它能够有效地捕捉序列中任意位置的特征之间的依赖关系。这种机制不依赖于特征的顺序，能够在输入的所有特征之间建立直接的关联。这在处理高维特征数据时尤其有用，因为它可以发现并利用特征之间的复杂关系和相互影响。

同时，我们查阅了一些论文，发现在2022年发表的一篇名为《Are Transformers Effective for Time Series Forecasting?》的论文中，作者将 Transformer 模型与 Linear 模型进行了比较（如下图所示）。最好的结果以粗体突出显示，第二好的结果用下划线表示。令人惊讶的是，Linear、NLinear 或 DLinear 模型在几乎所有任务中都优于 Transformer 模型。问题是，尽管线性模型比 Transformers 简单得多，但 Transformers 的表现却不佳，证实了我们的想法。

Methods		IMP.	Linear*		NLinear*		DLinear*		FEDformer		Autoformer		Informer		Pyraformer*		LogTrans		*Repeat**	
Metric		MSE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Electricity	96	27.40%	**0.140**	**0.237**	0.141	**0.237**	**0.140**	**0.237**	0.193	0.308	0.201	0.317	0.274	0.368	0.386	0.449	0.258	0.357	1.588	0.946
	192	23.88%	**0.153**	0.250	0.154	**0.248**	**0.153**	0.249	0.201	0.315	0.222	0.334	0.296	0.386	0.386	0.443	0.266	0.368	1.595	0.950
	336	21.02%	**0.169**	0.268	0.171	**0.265**	**0.169**	0.267	0.214	0.329	0.231	0.338	0.300	0.394	0.378	0.443	0.280	0.380	1.617	0.961
	720	17.47%	**0.203**	0.301	0.210	**0.297**	**0.203**	0.301	0.246	0.355	0.254	0.361	0.373	0.439	0.376	0.445	0.283	0.376	1.647	0.975
Exchange	96	45.27%	0.082	0.207	0.089	0.208	**0.081**	0.203	0.148	0.278	0.197	0.323	0.847	0.752	0.376	1.105	0.968	0.812	**0.081**	**0.196**
	192	42.06%	0.167	0.304	0.180	0.300	**0.157**	0.293	0.271	0.380	0.300	0.369	1.204	0.895	1.748	1.151	1.040	0.851	0.167	**0.289**
	336	33.69%	0.328	0.432	0.331	0.415	**0.305**	0.414	0.460	0.500	0.509	0.524	1.672	1.036	1.874	1.172	1.659	1.081	**0.305**	**0.396**
	720	46.19%	0.964	0.750	1.033	0.780	**0.643**	**0.601**	1.195	0.841	1.447	0.941	2.478	1.310	1.943	1.206	1.941	1.127	0.823	0.681
Traffic	96	30.15%	**0.410**	0.282	**0.410**	**0.279**	**0.410**	0.282	0.587	0.366	0.613	0.388	0.719	0.391	2.085	0.468	0.684	0.384	2.723	1.079
	192	29.96%	**0.423**	0.287	**0.423**	**0.284**	**0.423**	0.287	0.604	0.373	0.616	0.382	0.696	0.379	0.867	0.467	0.685	0.390	2.756	1.087
	336	29.95%	0.436	0.295	**0.435**	**0.290**	0.436	0.296	0.621	0.383	0.622	0.337	0.777	0.420	0.869	0.469	0.734	0.408	2.791	1.095
	720	25.87%	0.466	0.315	**0.464**	**0.307**	0.466	0.315	0.626	0.382	0.660	0.408	0.864	0.472	0.881	0.473	0.717	0.396	2.811	1.097
Weather	96	18.89%	**0.176**	0.236	0.182	**0.232**	**0.176**	0.237	0.217	0.296	0.266	0.336	0.300	0.384	0.896	0.556	0.458	0.490	0.259	0.254
	192	21.01%	**0.218**	0.276	0.225	**0.269**	0.220	0.282	0.276	0.336	0.307	0.367	0.598	0.544	0.622	0.624	0.658	0.589	0.309	0.292
	336	22.71%	**0.262**	0.312	0.271	**0.301**	0.265	0.319	0.339	0.380	0.359	0.395	0.578	0.523	0.739	0.753	0.797	0.652	0.377	0.338
	720	19.85%	0.326	0.365	0.338	**0.348**	**0.323**	0.362	0.403	0.428	0.419	0.428	1.059	0.741	1.004	0.934	0.869	0.675	0.465	0.394
ILI	24	47.86%	1.947	0.985	**1.683**	**0.858**	2.215	1.081	3.228	1.260	3.483	1.287	5.764	1.677	1.420	2.012	4.480	1.444	6.587	1.701
	36	36.43%	2.182	1.036	**1.703**	**0.859**	1.963	0.963	2.679	1.080	3.103	1.148	4.755	1.467	7.394	2.031	4.799	1.467	7.130	1.884
	48	34.43%	2.256	1.060	**1.719**	**0.884**	2.130	1.024	2.622	1.078	2.669	1.085	4.763	1.469	7.551	2.057	4.800	1.468	6.575	1.798
	60	34.33%	2.390	1.104	**1.819**	**0.917**	2.368	1.096	2.857	1.157	2.770	1.125	5.264	1.564	7.662	2.100	5.278	1.560	5.893	1.677
ETTh1	96	0.80%	0.375	0.397	**0.374**	**0.394**	0.375	0.399	0.376	0.419	0.449	0.459	0.865	0.713	0.664	0.612	0.878	0.740	1.295	0.713
	192	3.57%	0.418	0.429	0.408	**0.415**	**0.405**	0.416	0.420	0.448	0.500	0.482	1.008	0.792	0.790	0.681	1.037	0.824	1.325	0.733
	336	6.54%	0.479	0.476	**0.429**	**0.427**	0.439	0.443	0.459	0.465	0.521	0.496	1.107	0.809	0.891	0.738	1.238	0.932	1.323	0.744
	720	13.04%	0.624	0.592	**0.440**	**0.453**	0.472	0.490	0.506	0.507	0.514	0.512	1.181	0.865	0.963	0.782	1.135	0.852	1.339	0.756
ETTh2	96	19.94%	0.288	0.352	**0.277**	**0.338**	0.289	0.353	0.346	0.388	0.358	0.397	3.755	1.525	0.645	0.597	2.116	1.197	0.432	0.422
	192	19.81%	0.377	0.413	**0.344**	**0.381**	0.383	0.418	0.429	0.439	0.456	0.452	5.602	1.931	0.788	0.683	4.315	1.635	0.534	0.473
	336	25.93%	0.452	0.461	**0.357**	**0.400**	0.448	0.465	0.496	0.487	0.482	0.486	4.721	1.835	0.907	0.747	1.124	1.604	0.591	0.508
	720	14.25%	0.698	0.595	**0.394**	**0.436**	0.605	0.551	0.463	0.474	0.515	0.511	3.647	1.625	0.963	0.783	3.188	1.540	0.588	0.517
ETTm1	96	21.10%	0.308	0.352	0.306	0.348	**0.299**	**0.343**	0.379	0.419	0.505	0.475	0.672	0.571	0.543	0.510	0.600	0.546	1.214	0.665
	192	21.36%	0.340	0.369	0.349	0.375	**0.335**	**0.365**	0.426	0.441	0.553	0.496	0.795	0.669	0.557	0.537	0.837	0.700	1.261	0.690
	336	17.07%	0.376	0.393	0.375	0.388	**0.369**	**0.386**	0.445	0.459	0.621	0.537	1.212	0.871	0.754	0.655	1.124	0.832	1.283	0.707
	720	21.73%	0.440	0.435	0.433	0.422	**0.425**	**0.421**	0.543	0.490	0.671	0.561	1.166	0.823	0.908	0.724	1.153	0.820	1.319	0.729
ETTm2	96	17.73%	0.168	0.262	**0.167**	**0.255**	**0.167**	0.260	0.203	0.287	0.255	0.339	0.365	0.453	0.435	0.507	0.768	0.642	0.266	0.328
	192	17.84%	0.232	0.308	**0.221**	**0.293**	0.224	0.303	0.269	0.328	0.281	0.340	0.533	0.563	0.730	0.673	0.989	0.757	0.340	0.371
	336	15.69%	0.320	0.373	**0.274**	**0.327**	0.281	0.342	0.325	0.366	0.339	0.372	1.363	0.887	1.201	0.845	1.334	0.872	0.412	0.410
	720	12.58%	0.413	0.435	**0.368**	**0.384**	0.397	0.421	0.421	0.415	0.433	0.432	3.379	1.338	3.625	1.451	3.048	1.328	0.521	0.465

- Methods* are implemented by us; Other results are from FEDformer [31].

Table 2. Multivariate long-term forecasting errors in terms of MSE and MAE, the lower the better. Among them, ILI dataset is with forecasting horizon $T \in \{24, 36, 48, 60\}$. For the others, $T \in \{96, 192, 336, 720\}$. *Repeat* repeats the last value in the look-back window. The **best results** are highlighted in **bold** and the best results of Transformers are highlighted with a underline. Accordingly, IMP. is the best result of linear models compared to the results of Transformer-based solutions.

7. 参考文献

[1] Zeng, A., Chen, M., Zhang, L., & Xu, Q. (2022). Are Transformers Effective for Time Series Forecasting? arXiv. https://doi.org/10.48550/arXiv.2205.13504

[2] Zhao, Q., Zhu, J., Shen, X. *et al.* Chinese diabetes datasets for data-driven machine learning. *Sci Data* **10**, 35 (2023). https://doi.org/10.1038/s41597-023-01940-7

[3] Deep transfer learning and data augmentation improve glucose levels prediction in type 2 diabetes patients https://www.nature.com/articles/s41746-021-00480-x