抖音直播间数据分析与预测——作业报告

3018209202 张驰 独自一组～

代码URL:https://github.com/ScofieldaCreep/ECommerce_Performance_LSTM

1. 任务思路：

- 首先，我通过pandas的常规方法导入所需依赖包与数据集，并进行了数据预处理，包括变量降维处理、缺失值处理、异常值处理，以及通过sklearn进行数据集切分，使用sklearn对数据进行标准化处理。之后通过Matplotlib、Seaborn进行了数据可视化分析，探究了时间特征（工作日/节假日与早/中/晚）对目标变量的交叉影响关系，并尝试了statsmodels包中的季节趋势分析功能。
- 之后，我利用Tensorflow框架和Keras建立了一个简易的单变量LSTM+Dense模型，通过使用"每分钟进入直播间人数"进行了LSTM模型训练，分析了模型的表现并给出解决思路。由此我建立了第二个优化后的LSTM模型，得到了更好的效果。
- 由于数据资源充足，我在其后选择进行多变量LSTM模型预测方式，比较了其与第二个模型的数据表现差异。并在最后给出了对本次作业的反思和提升思路。

2. 代码过程

2.1 导入依赖包：

1. 在前文'1.任务思路'部分，我已列出任务主体框架所使用的工具包。此外，我还用了如下包，实现了一些功能：
    - 通过warnings的filterwarnings和os的environ来使notebook的呈现更加清晰。
    - 通过sklearn的mean_squared_error和r2_score来对模型结果进行评价和比较。
2. 在一些包的功能选择上，我有以下思考：
    - 为什么使用MinMaxScaler而不是StandardScaler?

      防止数据不平衡：使用MinMaxScaler可以缩放数据，使其不会因某些特征的值过大而影响模型的结果。

      简单易用，并且可以适用于多种数据类型，如数值和类别型数据。（类别型数据怎么缩放？）

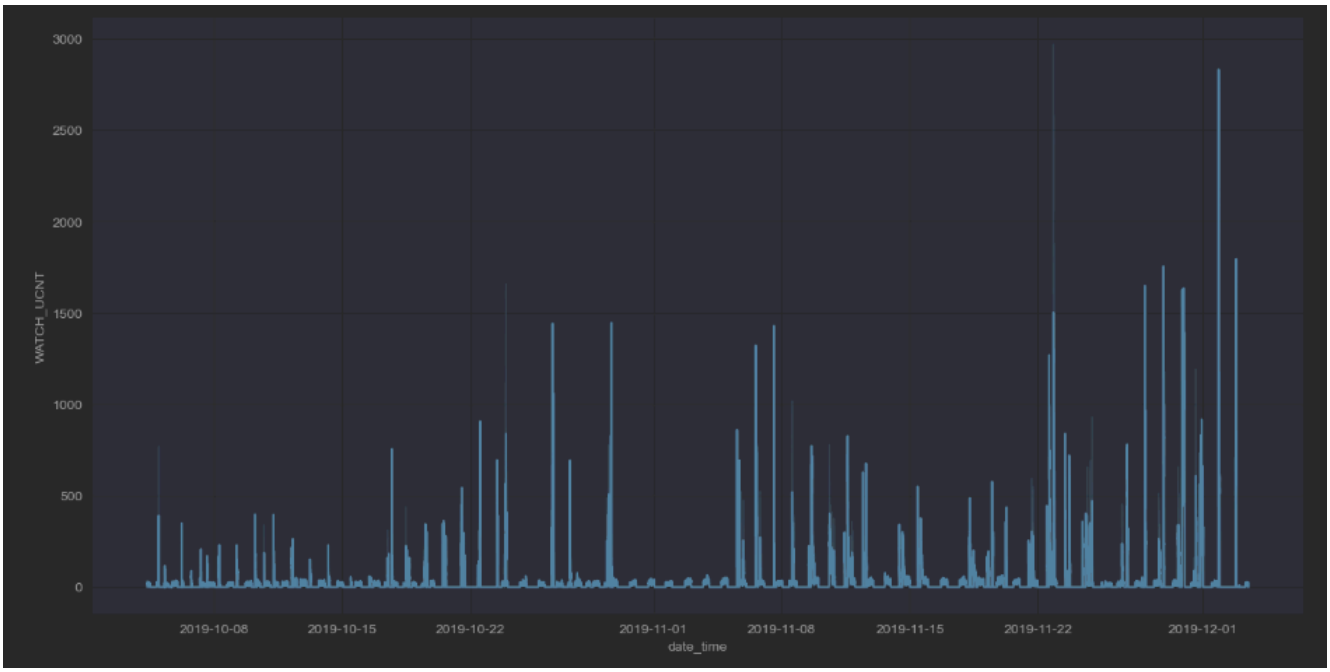
      可解释性高：MinMaxScaler可以很好地保留数据的原始分布

      适用于不同类型的模型：适用于许多不同类型的机器学习模型，并且对于不同类型的模型可以产生不同的结果。（这点不明白）

      其他方法简介：
      StandardScaler：标准化数据，将数据缩放至均值为 0，方差为 1 的范围内。
      RobustScaler：使用中位数和四分位数缩放数据，抗异常值的影响。
      Normalizer：把数据的每个样本的模长归一化为 1，可用于处理高维稠密数据。
    - Tensorflow训练模型结果保存的方法使用：TensorFlow和Keras提供了多种方法来保存和加载模型，tf.keras.callbacks.ModelCheckpoint是TensorFlow中的高级保存模型的方法，专门用于Keras API中训练的模型。它提供了一个简单易用的API，用于在每个训练时期保存模型，并可以通过设置权重文件的命名方式，以自定义保存的频率和位置。

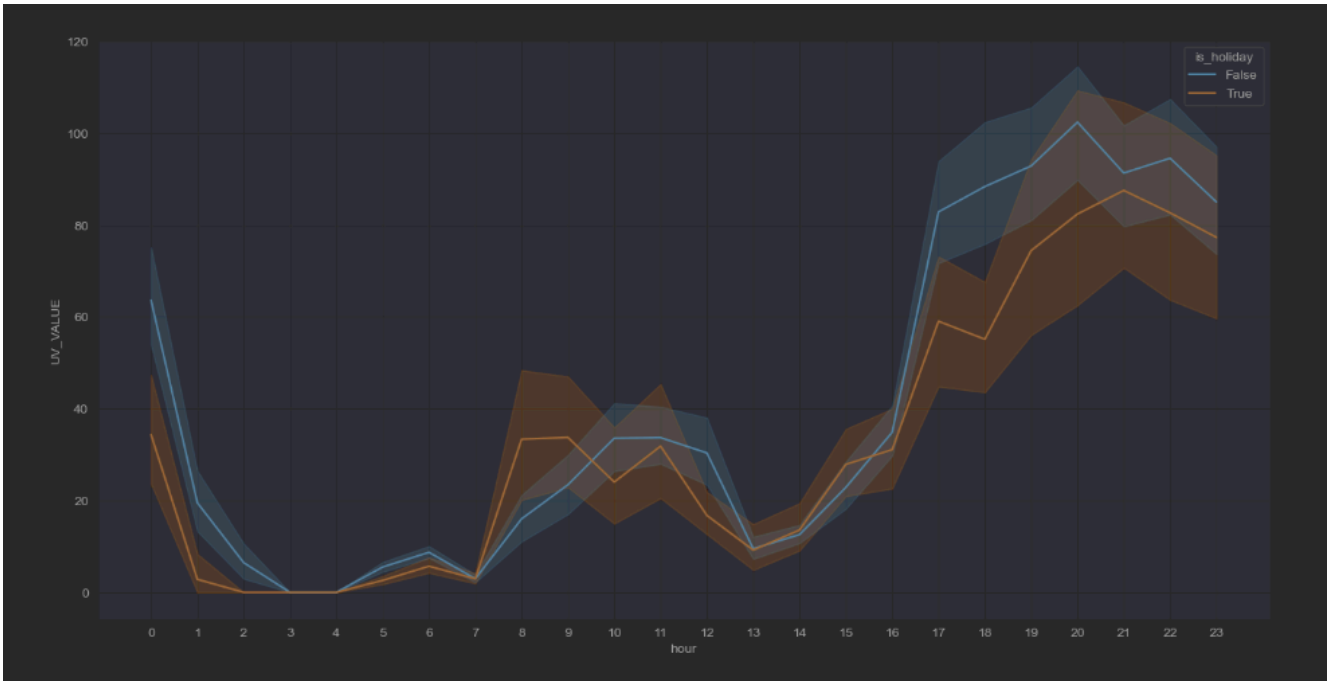
2.2 数据预处理：

1. 数据导入：这一步我通过pandas.dataframe的apply方法将数据降维，解决了'natural_flow_trend_index'等数据维度过高的问题。
2. 缺失值处理：由于数据集中的数据是分散且时长不等的，为方便后续模型学习和分析，我将最早时间与最晚时间之间未发生直播活动的时间均填以0值。此外，数据集中存在的NaN值等情况问题也得到处理。
3. 极端值处理：数据集中存在流量极少的"死直播"，我同样进行了删除操作。

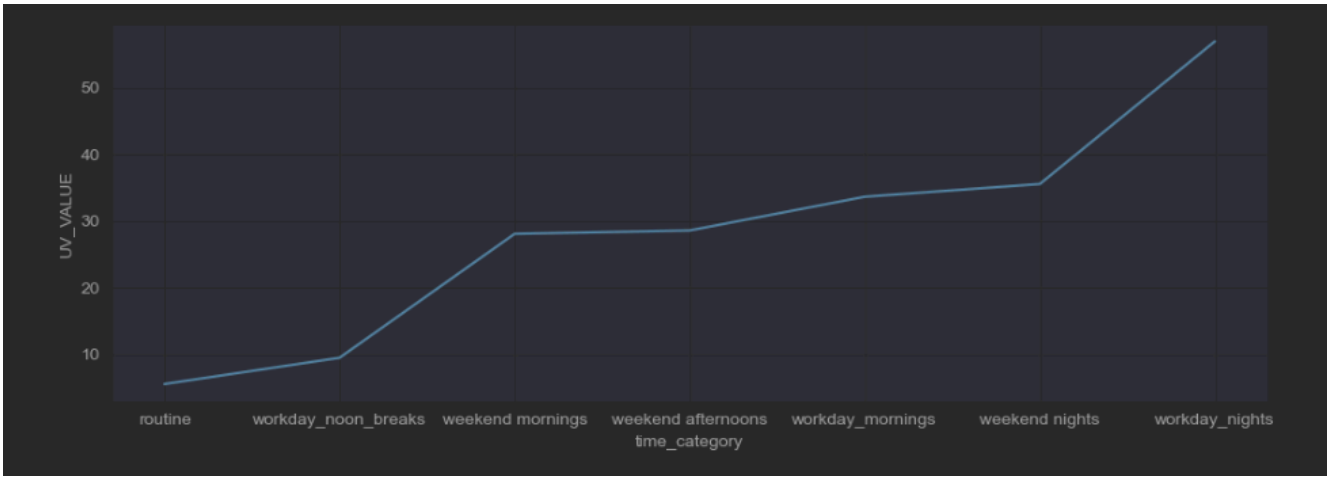


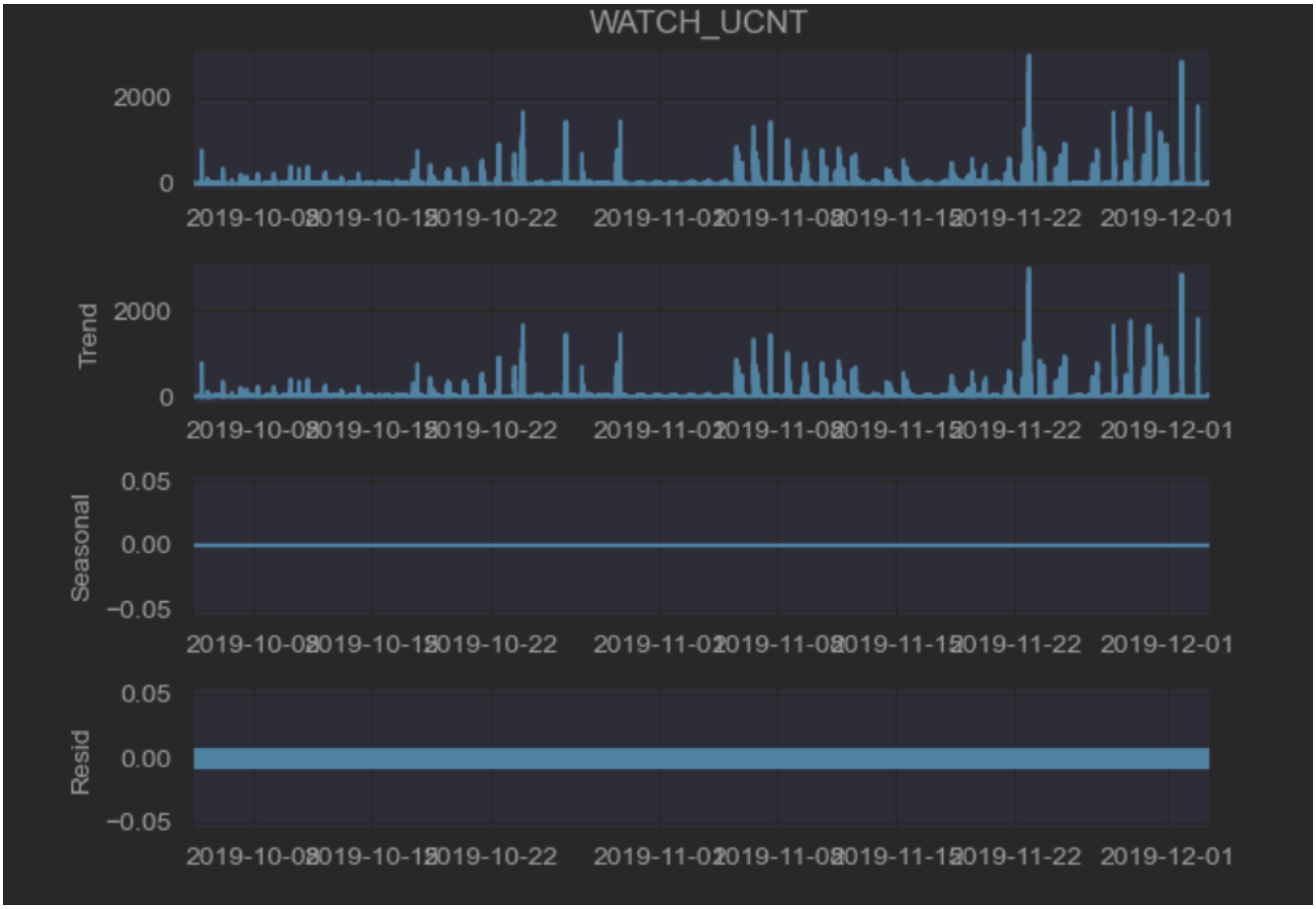
2.3 数据可视化与数据分析；

1. 我绘制了直播间每分钟进入人数的折线图，查看数据的趋势和周期性，对是否工作日、早中晚情况进行了交叉处理，得到了UV值的分布规律。



2. 我分析直播间每分钟进入人数与其他指标（如时间、天气等）的相关性和走势关系，观察到了包括工作日下午、工作日凌晨、周末上午等几个相对反直觉但有合理逻辑支持的高UV时段





2.4 单特征LSTM预测

1. timestep的选择：
    - LSTM 的 timestep 指的是在处理序列数据时，每个时间步携带的信息量。一般而言，LSTM 的每个时间步都是独立的，
      可以使用前一个时间步的输出作为输入，并使用隐藏状态等数据来保留历史信息。每个时间步可以看作一个样本，整个序列可以看作多个时间步组成的数据集。
    - LSTM 的 timestep 大小的选择取决于数据集的性质和模型的目标。
      如果 timestep 太小，可能不能捕捉到数据集的重要特征，并且难以保留历史信息，因此模型的性能会受到影响。
      但是，如果 timestep 太大，模型的学习任务会变得更加困难，并且容易导致过拟合，因此模型的泛化能力会受到影响。
    - 我选择100的心路历程：

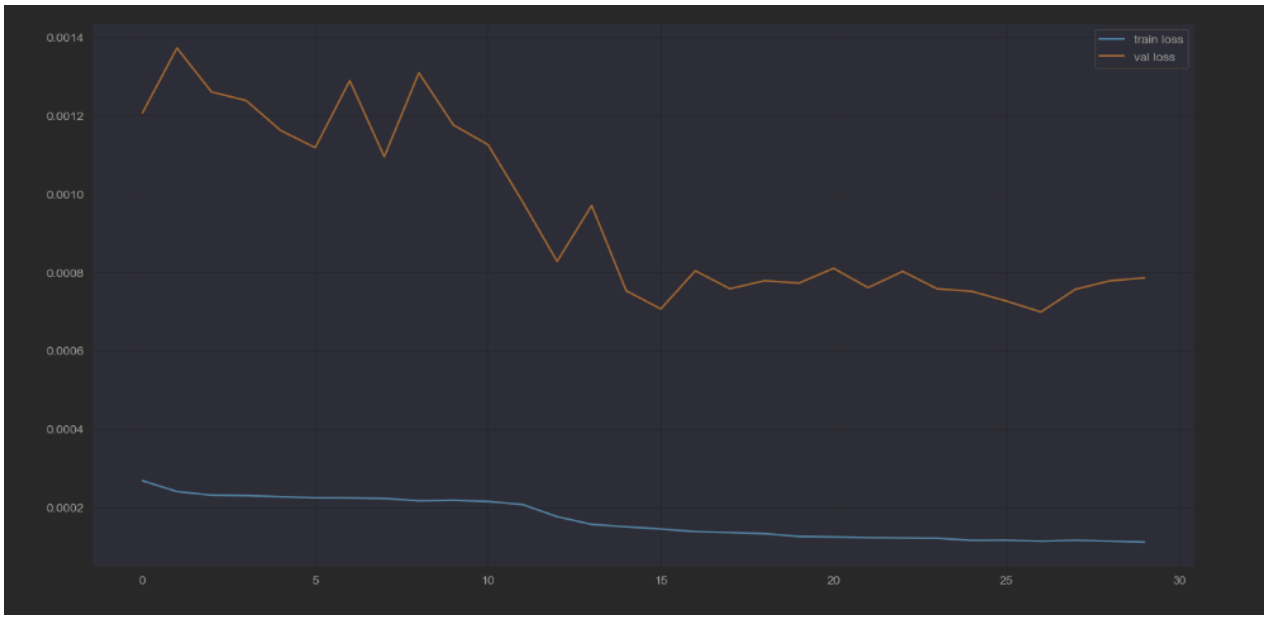
数据复杂性：数据集中的模式需要多个时间步才能被完整捕捉。
模型的性能：通过对不同的 timestep 进行评估，找到性能最佳的一个。
模型的泛化能力：通过对不同的 timestep 进行评估，找到泛化能力最好的一个。
在该任务中，我尝试了10，100，200三种情况，最终选择100作为time_step

2. 学习率算法选择：
    - Adam（Adaptive Moment Estimation）结合了另外两种优化算法（即随机梯度下降法（SGD）和Adagrad）的优点，
      并使用梯度的移动平均值和平方梯度的移动平均值来为每个权重提供自适应学习率，能够优化梯度不明显和噪声过多的情况。
    - AdaGrad (Adaptive Gradient Algorithm) 是一种自适应学习率优化算法，用于梯度下降法优化神经网络的参数。它通过不断调整每个参数的学习率来适应训练数据的复杂度，从而在训练的过程中加快收敛速度，同时避免因更新太快或太慢导致无法收敛的问题。
    - 优点：适用于处理稀疏数据：AdaGrad 对稀疏数据效果比较好，因为它能够根据稀疏数据的特点动态调整每个参数的学习率，使得其能够更快的收敛。能够更好的避免陷入局部最小值：由于 AdaGrad 对学习率进行了动态调整，因此能够更好的避免陷入局部最小值。
    - 可以用在：

      神经网络的参数优化：AdaGrad 通常用于优化神经网络的参数，在训练神经网络时通常可以加快收敛速度，并且提高模型的准确性。

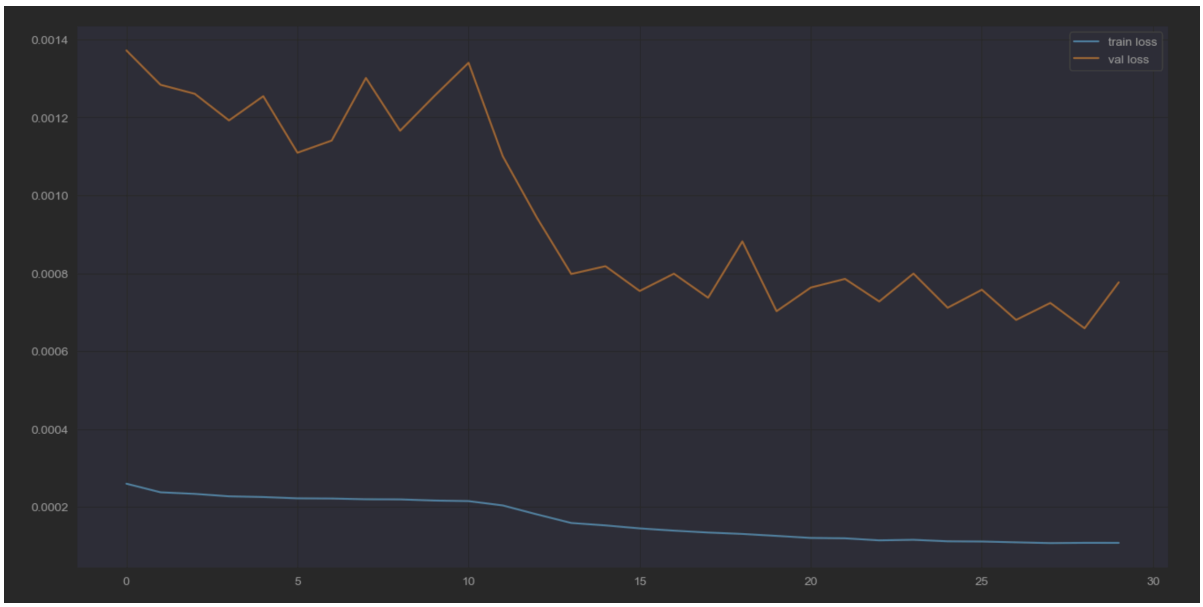
      其他机器学习任务：除了用于神经网络的参数优化，AdaGrad 也可以用于其他机器学习任务，比如说支持向量机等算法的优化。

3. batch_size的选择：
    - 如果 batch_size=1，那么模型每次只使用一个样本进行训练，这种方法称为随机梯度下降(SGD)；
    - 如果 batch_size=64，那么模型每次使用64个样本进行训练，这种方法称为批量梯度下降(BGD)。
    - 通常情况下，BGD能够较快的收敛，但是需要更多的内存；SGD则相反，训练慢一些，但是需要更少的内存。
4. 对于第一个模型，肉眼观察模型效果：
    - 训练集loss减小，但验证集loss没有变化,这种情况通常是出现了过拟合，
      说明模型在训练集上表现很好，但是对新数据的泛化能力不强。
      可以通过采取正则化技巧，例如dropout或者L2正则化来解决过拟合问题。



5. 优化模型：

- 正则化方法选择的思路:
  Dropout 是一种神经网络正则化技术，其原理是在训练过程中随机删除网络中的部分神经元，从而防止模型的过拟合。Dropout 对于简单的模型效果很好，但对于复杂的模型可能不够有效。
  L2 正则化是一种惩罚系数矩阵的平方和，从而防止模型过拟合。L2 正则化通常比 Dropout 好，但其实现较复杂。
  (失效！)在业务上，我认为使用Dropout会损失很多信息，相对于Dropout，L2通过L2引入惩罚系数的方式也许更好！
- 通过调整激活函数，避免过拟合现象的思路：
  - 在LSTM层：
    tanh（双曲正切函数）和线性激活函数比较常见，tanh其输出在-1到1之间，并且具有对称性和单调性，使其适合作为循环单元内部状态的激活函数。
    线性激活函数是一种不改变输入值的激活函数，其可以通过linear指定，它具有简单性和稳定性，但在网络中不会产生额外的非线性效果。
    在这里，我尝试了使用tanh作为新的激活函数，因为实时数据的变化不是简单线性关系。
  - 在Dense全连接层：
    最开始我的想法是这样的：
    ReLU是简单快速的激活函数，对于大多数数据集都是一个很好的选择。然而，在数据中有较多的0值时，模型可能会发生梯度消失的问题，从而影响模型的性能。
    LeakyReLU是ReLU的一种变体，具有较小的负斜率，从而防止梯度消失的问题。在处理具有0值的数据时，LeakyReLU可能更加有效。
    ELU是Exponential Linear Unit的缩写，与LeakyReLU相似，具有负斜率。在处理具有0值的数据时，ELU可能更加有效，因为它的负斜率逐渐增大，直到达到极限值。因此，LeakyReLU和ELU的差别在于其如何处理输入的负值。LeakyReLU通过将负值乘以一个较小的负斜率来处理，而ELU则通过使用负数的指数处理负值来处理负值。因此，在某些情况下，ELU可能比LeakyReLU更有效，因为它在0处具有负平均值。
    在这里，我先尝试了使用relu，但是也出现了训练集loss下降但验证集loss不变的情况，遂改用ELU。
  - 但是我发现效果很差，突然想到我的输入数据全部>=0，而ReLU在这种情况下可能会出现'ReLU Dying'的情况，即Relu激活函数中有一些神经元永远不会被激活，将所有输入设置为零。ELU也可能出现这种情况（虽然我的模型层数并不多...）于是我决定取消使用ELU，看一下效果。



6. 评价模型
    - r2, coefficient of determination, 是一种用于评估回归模型拟合度的统计度量。它衡量模型根据输入特征的变化解释目标变量的变化的能力。
    - r2的值在0和1之间，接近1表示良好的拟合，接近0表示不佳的拟合。负的r2意味着模型不是很好的拟合，实际上比仅使用目标变量的平均值作为预测更劣。
    - 感觉只能用来横向比较hh，所以我打算一会和多变量LSTM比较一下子！
7. ***完成数据预测任务***
    - 预测结果如下：

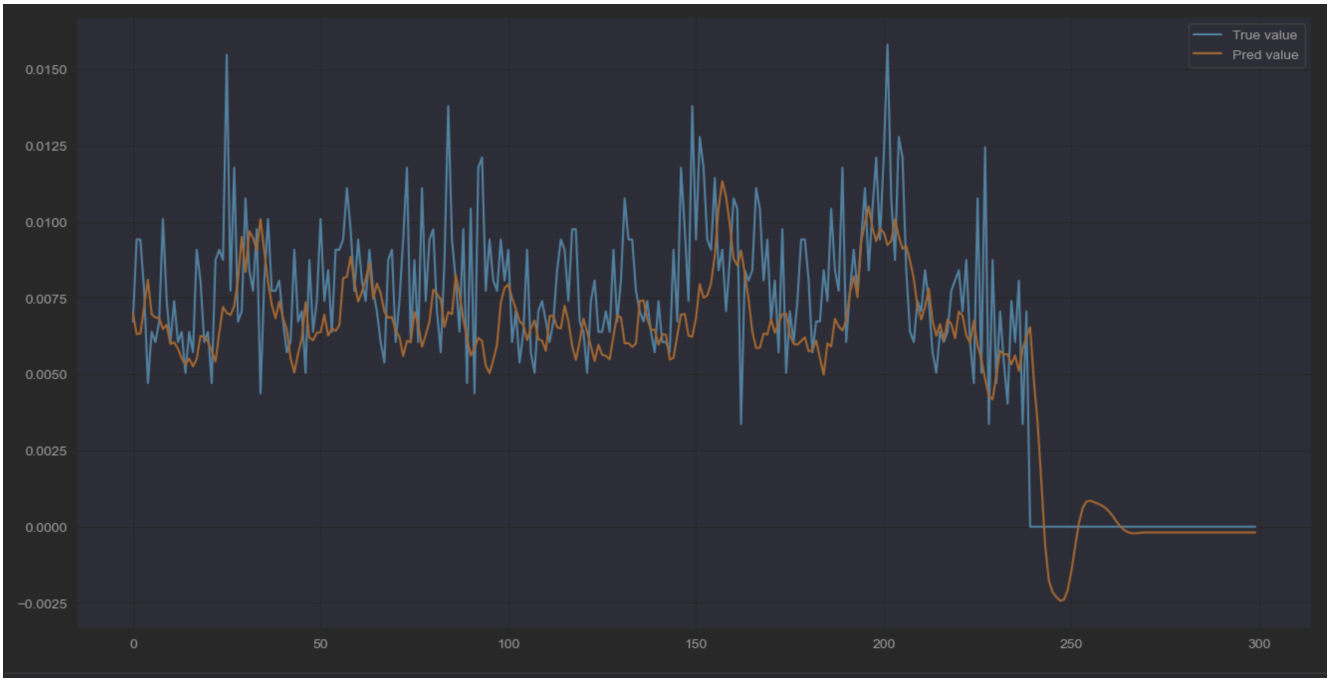
```
[17, 14, 13, 11, 10, 8, 7, 7, 6, 5]
```

    - 在处理数据时，用到的标准化算法的思考：

      minmaxscaler.transform 和 minmaxscaler.fit_transform 的区别是： minmaxscaler.fit_transform: 先找到数据的最大值和最小值，然后计算所有数据的缩放值并进行缩放，同时返回缩放后的数据。 minmaxscaler.transform: 已经计算出了数据的最大值和最小值，然后直接使用已有的最大值和最小值对数据进行缩放，并返回缩放后的数据。 所以，如果没有计算过数据的最大值和最小值，那么必须使用 minmaxscaler.fit_transform。如果已经计算过，则可以直接使用 minmaxscaler.transform。

2.5 多特征LSTM预测

- 多特征LSTM预测的操作类似，在这里我使用了另一种模型搭建的方法，输入Sequential()的参数一步到位生成模型，并使用ModelCheckpoint保存数据。
- 尽管多特征LSTM和单特征LSTM的数据体量和性质不一样，是的，MSE（均方误差）可以用来比较两个数据规模和参数数量不同的模型。 MSE反映了模型预测值与真实值之间的平均误差，与数据规模和参数数量无关，因此可以用来比较不同模型的表现。
  由于我的训练结果最终趋向于过拟合，MSE表现也比不上上一个模型，因此弃用该模型进行业务预测。



3. 总结与建议

3.1 模型表现的反思：

3.1.1 对于模型不能有效预测直播间人流量突变情况的反思：

1. 收集更多数据：如果数据量不够，收集更多数据以提高模型的预测能力。
2. 特征工程：对特征进行选择，提取和构造以更好地描述直播间人流量的变化。
    - 在数据的预处理阶段，我并没有对时间序列的特征进行有效提取，比如在数据可视化分析阶段我发现同一天不同时间段、工作日与周末所反映的UV值的不同，也许会影响到目标变量。因此，也许根据时间序列提取周末/工作日的0-1二元特征，早/中/晚的三元特征等等时间所隐含的特征的提取会有助于业务预测。
3. 模型选择：尝试使用其他类型的模型，例如支持向量机（SVM），随机森林（RF），GBDT等，也许会有更好的预测效果，但是LSTM相对而言更符合业务逻辑和直觉，具体模型效果有待比较。
4. 模型调整：调整模型的超参数以提高预测的准确性。
5. 数据预处理：对数据进行预处理，例如归一化，标准化等。这一块我思考的不够透彻，对于不同的归一化方法的理解仍有欠缺（在我看来MinMaxScaler通吃任何场景呀...）。
6. 融合多个模型：可以尝试融合多个模型，如模型融合，stacking等。
    - 模型融合即通过将多个模型的结果结合起来来提高预测效果。常见的模型融合技术包括加权平均、投票和stacking。
    - Stacking即一种基于预测的模型融合技术，它通过使用一个新的模型来学习如何结合多个模型的预测结果。
    - 为了使LSTM融合多个模型，需要对多个模型进行训练，然后使用模型融合或stacking技术将这些模型的结果结合起来。此外，还可以使用数据增强技术，如随机旋转和翻转等，来提高模型的泛化能力。
    - 这一部分由于我独自成组，个人任务过重，并没有机会在有效时间内完成。（但很好奇这样操作会对模型表现产生多大的影响）
7. 模型的技巧：使用一些技巧，例如时间戳嵌入，回归技巧等，来提高预测结果的准确性。
    - 学习率、激活函数、权重初始化方式、训练次数、隐层数、隐层神经元数、正则化系数等,由于算力限

制，我并没有投入大量的精力进行调优。但是其实我可以通过函数式编程思想和交叉集验证择优模式，在算力充足的情况下自动优化，或手动设置默认参数（这样也能够让代码逻辑更为清晰）。与之相比，本代码作业中的平铺直叙的代码编写方式在后期优化过程中很耗费精力，并不能很方便地比较输入参数对模型表现的影响。

3.2 代码编写过程的反思：

1. 由于没有大量面向函数编程，导致我在后面做了很多重复工作，如对目标的待预测数据集的数据预处理工作。
2. 在变量命名的过程中也比较混乱，即使在第一个代码段已明确列出变量含义，在后面的修改中也很容易覆盖前面的变量，导致回头优化时需要把前面做过的工作推倒重来。
3. 在模型保存的工作上，我也忘记了清晰保存记录每一个训练模型，同样会产生覆盖问题。