人工智能与数据挖掘课程设计

报告题目

**组员信息：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 姓名 | 学号 | 成员贡献 | 实践成绩 |
| 1 | 范宣哲 | 2021522002 | 30% | 7 |
| 2 | 张泊晗 | 2021522011 | 30% | 7 |
| 3 | 周思达 | 2021522003 | 20% | 7 |
| 4 | 吕致成 | 2021522007 | 20% | 7 |

# **引言**

随着城市交通日益繁忙和车辆数量的不断增加，对于车辆速度预测技术的需求愈发迫切。在过去的几年里，人工智能和数据挖掘技术的迅猛发展为解决交通管理和规划中的挑战提供了新的可能性。车辆速度预测作为交通流管理和智能交通系统的重要组成部分，对于提高交通效率、减缓交通拥堵、提升交通安全具有重要意义。

传统的车辆速度预测方法主要依赖于基于统计学的模型和传感器数据。然而，这些方法在处理复杂的城市交通环境时面临着挑战，包括交叉口拥堵、突发事件、道路工程等因素的影响。人工智能技术的引入为解决这些复杂问题提供了新的途径。通过深度学习、机器学习和数据挖掘等技术，可以更精准地捕捉车辆行为模式和交通动态，从而实现更准确、实时的车辆速度预测。

本研究旨在探讨人工智能在车辆速度预测领域的应用，通过深入挖掘大量的交通数据，结合先进的数据挖掘技术，建立高效准确的车辆速度预测模型。通过这一研究，我们希望为城市交通管理和规划提供更为智能化和可持续的解决方案，从而为城市居民提供更顺畅、高效的出行体验。这也将推动人工智能技术在智能交通系统领域的更广泛应用，为未来的城市交通发展奠定坚实的基础。

# **实验数据**

## 数据获取或预处理

## 数据读取

###### 读取文件中的数据，重置数据框的索引，并显示前5行数据。

data1 = pd.read\_csv("./speed.csv")  
data1.index = range(len(data1))  
data1.head()

时间

车流量

温度

湿度

气压

道路拥堵指数

车速(m/s)

##### 生成一个位于区间[-1, 1]内的随机浮点数。

import random  
random.uniform(-1,1)

0.9191670780704275

###### 读取文件中的数据，重置数据框的索引，并显示前5行数据。

data = pd.read\_csv("./车速.csv")  
data.index = range(len(data))  
data.head()

## 数据预处理

###### 对数据集进行预处理，将特征值进行最小-最大缩放处理，返回处理后的数据集X的形状。

# 数据归一化  
# 时间序列对数据量纲是十分敏感的，所以必须进行数据归一化  
from sklearn.preprocessing import MinMaxScaler, StandardScaler  
# dataset = df['车速(m/s)'].values  
dataset = df.values  
dataset = dataset.astype('float32')  
# dataset = dataset.reshape(-1, 1)  
scaler = MinMaxScaler()  
sc\_model\_X = scaler.fit(dataset[:,:-1])  
dataset\_X= sc\_model\_X.transform(dataset[:,:-1])  
dataset\_X.shape

(7201, 5)

###### 对数据集中的目标变量进行最小-最大缩放处理，返回处理后的数据集Y的形状。

scaler = MinMaxScaler()  
sc\_model\_y = scaler.fit(dataset[:,-1].reshape(-1,1))  
dataset\_y= sc\_model\_y.transform(dataset[:,-1].reshape(-1,1))a  
dataset\_y.shape

(7201, 1)

###### 将数据集划分为训练集和测试集，前70%作为训练集，剩余30%作为测试集，并分别打印出训练集和测试集的样本数。

# 训练和测试的划分  
# 7:3的比例分为训练集和测试集  
train\_size = int(len(dataset) \* 0.7) ##前70%作为训练，30%作为测试  
test\_size = len(dataset) - train\_size  
train\_X, test\_X = dataset\_X[0:train\_size,:], dataset\_X[train\_size:len(dataset\_X),:]  
train\_Y, test\_Y = dataset\_y[0:train\_size], dataset\_y[train\_size:len(dataset\_y)]  
  
print(len(train\_X), len(test\_X))

5040 2161

## 数据统计

##### 复制数据框data，并打印出复制后数据框的形状信息，其中第一个元素表示行数，第二个元素表示列数。

# 数据维度  
df = data.copy()  
print(df.shape)

(7201, 6)

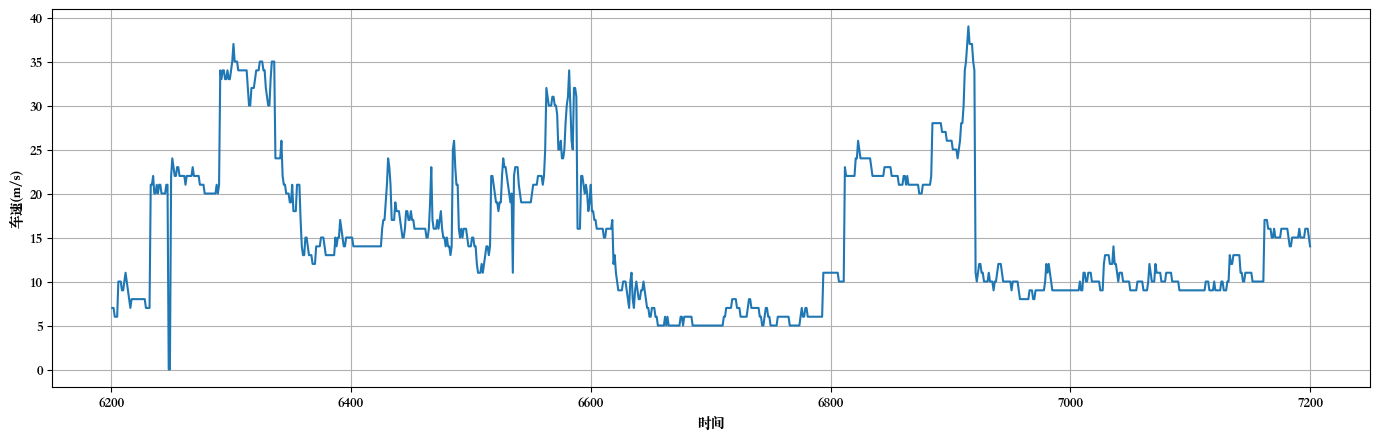
###### 打印出数据框的基本信息摘要，包括列名、非空值数量、每列的数据类型等信息。

# 数据字段类型  
print(df.info())

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7201 entries, 0 to 7200  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 车流量 7201 non-null float64  
 1 温度 7201 non-null float64  
 2 湿度 7201 non-null float64  
 3 气压 7201 non-null float64  
 4 道路拥堵指数 7201 non-null float64  
 5 车速(m/s) 7201 non-null int64   
dtypes: float64(5), int64(1)  
memory usage: 337.7 KB  
None

###### 取数据框的最后1000个索引值，数据框中’车速(m/s)’列的最后1000个值，其中横坐标轴表示”时间”，纵坐标轴表示”车速(m/s)“，绘制出车速的时间序列走势。

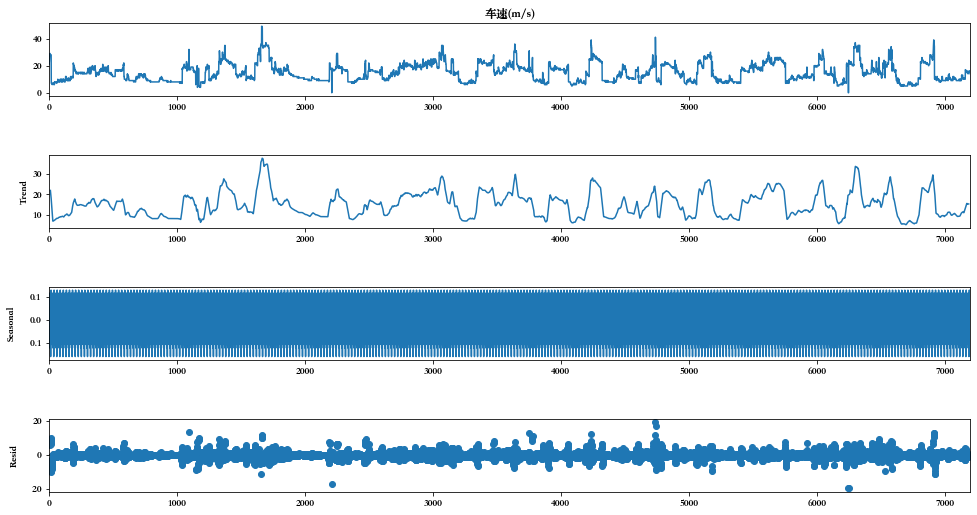
# 车速的时间序列走势  
plt.figure(figsize=(17,5), dpi=100)  
plt.plot(df.index[-1000:], df['车速(m/s)'][-1000:])  
plt.grid()  
plt.xlabel("时间")  
plt.ylabel("车速(m/s)")  
plt.show()



###### 使用季节性分解对名为’车速(m/s)’的时间序列数据进行分解，季节性的周期为24，并绘制出分解后的趋势、季节性等成分。

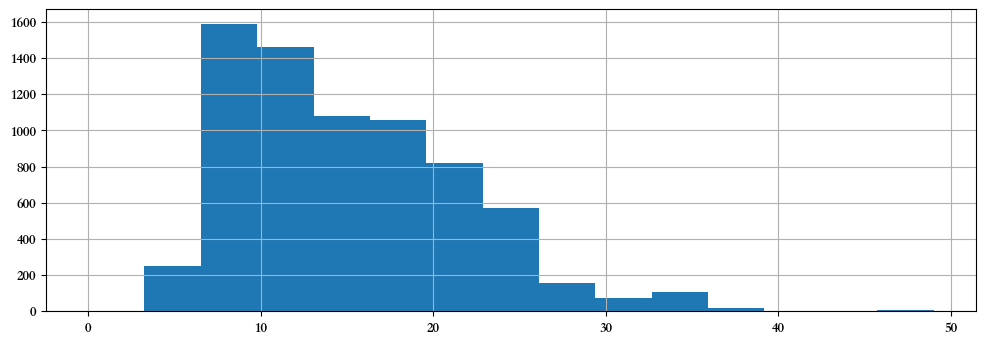
# 可见数据存在一定的时间周期  
# 进一步的分析时间的周期性、季节趋势等  
decomposition = seasonal\_decompose(df['车速(m/s)'], period=24)   
#   
fig = plt.figure()   
fig = decomposition.plot()   
fig.set\_size\_inches(15, 8)  
plt.show()

<Figure size 432x288 with 0 Axes>



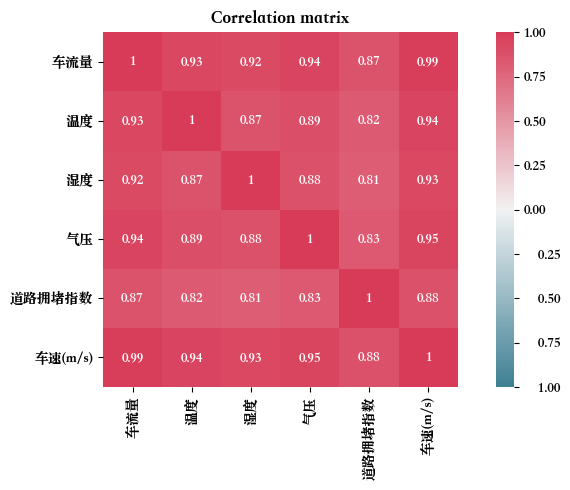
###### 取数据框中’车速(m/s)’列的所有值，并将数据分为15个区间进行统计，绘制出车速的柱状图，根据图像可以看出车速大部分分布在10和20左右。

#车速(m/s)的直方图分布\  
plt.figure(figsize=(12,4), dpi=100)  
plt.hist(df['车速(m/s)'], bins=15)  
plt.grid()  
plt.show() # 分布比较均衡,大多30和40左右



###### 根据数据框中车流量、温度、湿度等变量得出它们之间的相关系数，并绘制出方形矩阵热力图，根据热力图可以看出各个变量之间的相关性都比较高。

fig, ax=plt.subplots(figsize=(8,5), dpi=100)  
cmap = sns.diverging\_palette(220, 6, as\_cmap=True)  
sns.heatmap(df.corr(), vmax=1, vmin=-1, center=0, square=True, ax=ax, cmap=cmap, annot=True)  
ax.set\_title('Correlation matrix', fontweight='bold', fontsize=13)  
plt.tight\_layout()  
plt.show()



# 和各个变量之间的相关性高

# **方法/模型**

## LSTM模型

###### 定义create\_dataset（）函数，将训练集和测试集转换为 LSTM 模型可接受的格式。

def creat\_dataset(data\_X,data\_Y, lagTerm=1):  
 # 创建数据集以供LSTM使用，转换为LSTM所需要的数据格式  
 # lagTerm: 滑动窗口大小，认为前lagTerm条数据将会影响lagTerm+1的数据  
 dataX, dataY = [],[]  
 for i in range(len(data\_X) - lagTerm):  
 cache = data\_X[i:(i+lagTerm),:]  
 dataX.append(cache)  
 dataY.append(data\_Y[i+lagTerm])  
 return np.array(dataX), np.array(dataY)  
look\_back = 10  
trainX,trainY = creat\_dataset(train\_X,train\_Y, look\_back)  
testX, testY = creat\_dataset(test\_X,test\_Y, look\_back)

###### 打印训练集和测试集的特征和目标变量数组的形状。其中trainX.shape表示训练集特征数组的形状（即行数和列数），trainY.shape表示训练集目标变量数组的形状，testX.shape和testY.shape表示测试集特征和目标变量数组的形状。

#训练和测试的数据维度转换 [samples, time\_steps, features]  
# trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))  
# testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))  
print(trainX.shape, trainY.shape)  
print(testX.shape, testY.shape)

(5030, 10, 5) (5030, 1)  
(2151, 10, 5) (2151, 1)

###### 定义包含多个LSTM层的模型，并使用Adam优化器编译和训练该模型模型。

from tensorflow.keras.optimizers import Adam  
def lstm\_model(numEpoch,batchSize,X,y,callBack):  
 model = Sequential()  
 model.add(LSTM(  
 120,  
 input\_shape=(X.shape[1], X.shape[2]),  
 return\_sequences=True))  
 model.add(Dropout(0.3))  
 model.add(LSTM(  
 128,  
 return\_sequences=True))  
 model.add(Dropout(0.3))  
 model.add(LSTM(units=64, activation='relu', return\_sequences=True))  
 model.add(Dropout(0.2))  
 model.add(LSTM(units=32, activation='relu', return\_sequences=False))  
 model.add(Dropout(0.2))  
 model.add(Dense(1))  
 model.add(Activation("relu"))  
 model.compile(optimizer=Adam(learning\_rate=0.01), loss='mae', metrics=['mse'])  
 history = model.fit(X,y,epochs = numEpoch, batch\_size=32, verbose=1, validation\_split=0.25, callbacks=callBack)  
 return model,history

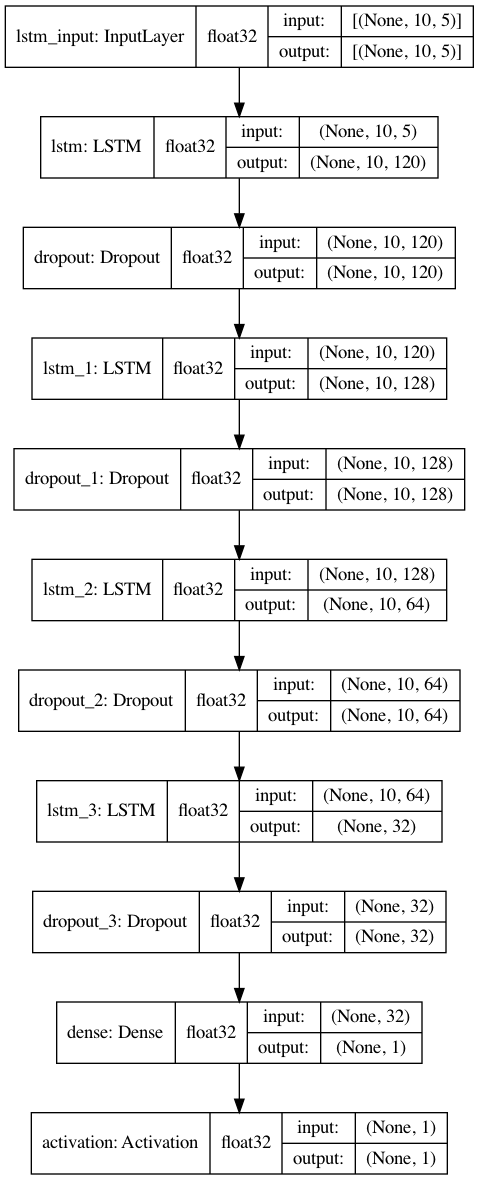
###### 对LSTM模型进行训练训练次数为100次，并使用EarlyStopping和ModelCheckpoint这两个回调函数来实现在训练过程中的自动提前停止和模型保存功能。当连续15个epoch验证集上的损失值都没有改善时，训练将会提前停止，并恢复到最佳的模型权重。model\_save（）函数用来监视验证集上的损失值，当出现更好的模型时，会将模型保存到”lstm\_15min.h5”的文件中。

import tensorflow as tf  
# Early stop callback function.  
  
early\_stopping\_cb = tf.keras.callbacks.EarlyStopping(patience=15, restore\_best\_weights=True)  
# 保存模型  
model\_save = tf.keras.callbacks.ModelCheckpoint(filepath="lstm\_15min.h5", save\_weights\_only=False, monitor='val\_loss',  
 save\_best\_only=True)  
lstm, lstm\_history = lstm\_model(100, 32, trainX, trainY, [model\_save])

2023-12-21 22:42:17.770824: I tensorflow/compiler/jit/xla\_cpu\_device.cc:41] Not creating XLA devices, tf\_xla\_enable\_xla\_devices not set  
2023-12-21 22:42:17.774001: I tensorflow/core/platform/cpu\_feature\_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA  
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.  
2023-12-21 22:42:21.054841: I tensorflow/compiler/mlir/mlir\_graph\_optimization\_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)

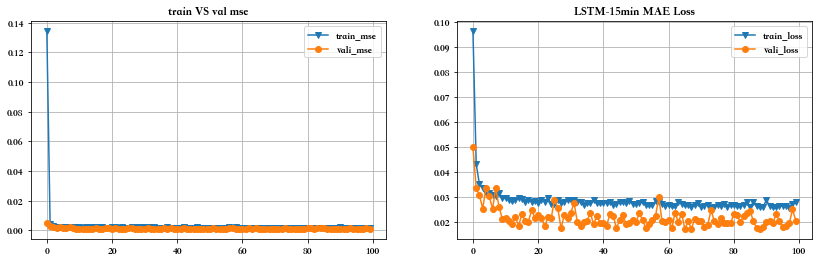
###### 生成 LSTM 模型的可视化结构图，包括每个层的名称、输入输出形状以及数据类型，并将结果保存为名为 “lstm.png” 的图片文件，直观地了解模型的结构和每一层的信息。

plot\_model(lstm, "lstm.png", show\_dtype=True,  
 show\_layer\_names=True, show\_shapes = True)



###### 绘制 LSTM 模型在训练过程中损失值和准确率的变化曲线。从图中可以看出，准确率和损失值的变化曲线刚开始均快速下降说明模型正在逐渐学习并优化自己的预测能力，然而当训练进行到一定程度后均趋于稳定，说明模型已经接近最佳状态。

train\_loss=lstm\_history.history['loss']  
vali\_loss=lstm\_history.history['val\_loss']  
train\_acc = lstm\_history.history['mse']  
vali\_acc = lstm\_history.history['val\_mse']  
plt.figure(figsize=(14,4))  
plt.subplot(1,2,1)  
plt.plot(range(len(train\_acc)),train\_acc,marker='v',label='train\_mse')  
plt.plot(range(len(train\_acc)),vali\_acc,marker='o',label='vali\_mse')  
plt.legend()  
plt.title("LSTM-15min MSE")  
plt.grid()  
plt.title("train VS val mse")  
plt.subplot(1,2,2)  
plt.plot(range(len(train\_loss)),train\_loss,marker='v',label='train\_loss')  
plt.plot(range(len(train\_loss)),vali\_loss,marker='o',label='vali\_loss')  
plt.legend()  
plt.title("train VS val loss")  
plt.grid()  
plt.title("LSTM-15min MAE Loss")  
plt.show()



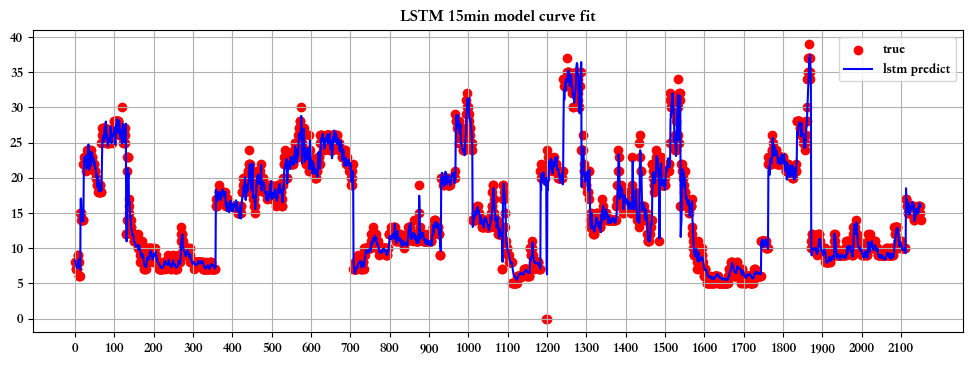
###### 评估训练好的 LSTM 模型在测试集上的性能，并计算其均方根误差（RMSE）和平均绝对误差（MAE）。根据结果可以看出MAE和RMSE的值均比较小，可见LSTM 模型在测试集上的性能比较好。

# LSTM模型在测试上的效果--MSE和MAE以及MAPE  
from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error  
from tensorflow.keras.models import load\_model  
lstm\_best = load\_model("lstm\_15min.h5")  
lstm\_pred = lstm\_best.predict(testX)  
inverse\_lstm\_pred = sc\_model\_y.inverse\_transform(lstm\_pred.reshape(-1,1))  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score  
inverse\_test = sc\_model\_y.inverse\_transform(testY.reshape(-1,1))  
lstm\_rmse = np.sqrt(mean\_squared\_error(inverse\_test, inverse\_lstm\_pred))  
lstm\_mae = mean\_absolute\_error(inverse\_test, inverse\_lstm\_pred)  
  
# 均方根误差、平均绝对误差  
print("LSTM 15min MAE:", lstm\_mae)  
print("LSTM 15min RMSE:", lstm\_rmse)

LSTM 15min MAE: 0.9435141  
LSTM 15min RMSE: 1.7825676

###### 绘制 LSTM 模型在测试集上的预测结果与真实值之间的拟合情况，其中散点表示真实值，折线表示模型的预测值。根据图像可以看出两者之间趋势相似且接近，可见模型的拟合效果比较好。

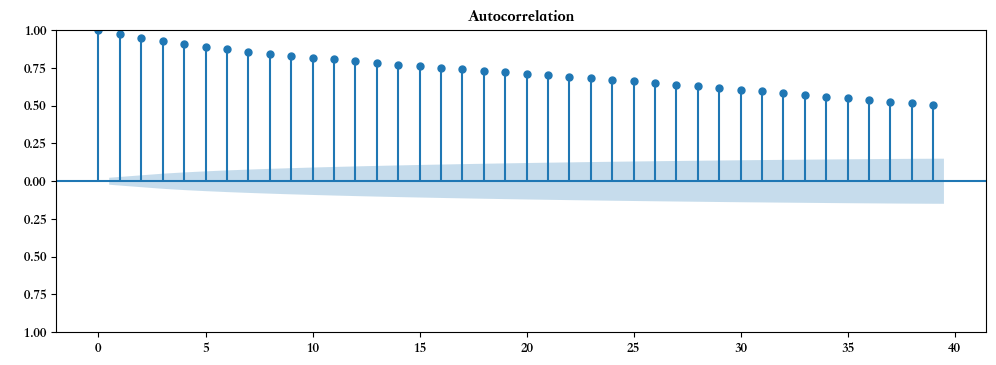
indexs = range(len(df.index[-testX.shape[0]:]))  
# 拟合情况  
plt.figure(figsize = (12,4), dpi=100)  
plt.scatter(indexs, inverse\_test, marker='o', label='true', c='red')  
plt.plot(indexs, inverse\_lstm\_pred, c='blue', label='lstm predict')  
plt.legend()  
plt.title("LSTM 15min model curve fit")  
plt.xticks(np.arange(0, len(df.index[-testX.shape[0]:]), 100))  
plt.grid()  
plt.show() # 可见拟合得很好



## ARIMA

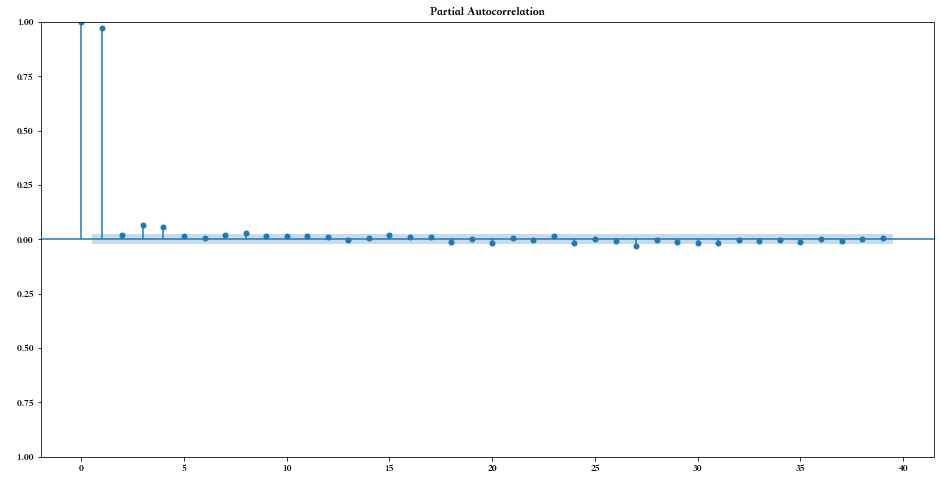
###### 绘制车速时间序列数据的自相关图。从图中可以看出，随着时间的推移，车速的自相关系数逐渐减小，表明车速的变化趋势逐渐减弱。此外还可以看出，自相关系数在时间间隔较小时呈现出较高的值，表明短期内车速的变化具有较强的相关性。

series = np.array(df['车速(m/s)'].values)  
from statsmodels.graphics.tsaplots import plot\_acf  
fig = plt.figure(figsize=(12,4), dpi=100)  
ax1=fig.add\_subplot(111)  
plot\_acf(series,ax=ax1).show()###自相关图检验平稳性



###### 绘制时间序列数据的偏自相关函数图，通过图中我们可以看出车速时间序列具有一定的自相关性，但这种自相关性是有限的。

from statsmodels.graphics.tsaplots import plot\_pacf###偏自相关图  
fig = plt.figure(figsize=(16,8))  
ax2=fig.add\_subplot(111)  
plot\_pacf(series,ax=ax2).show()



###### 对时间序列数据进行单位根检验（ADF检验），结果表明该时间序列数据是平稳的。

# 平稳性检测  
from statsmodels.tsa.stattools import adfuller as ADF  
  
print('原始序列的ADF检验结果为:',ADF(series))

原始序列的ADF检验结果为: (-7.363833723010498, 9.348399679062679e-11, 10, 7190, {'1%': -3.4312598242236394, '5%': -2.861942070882888, '10%': -2.5669840181754524}, 26038.59623096793)

# 本身平稳 不用做差分， d=0

###### 使用 ARIMA 模型对时间序列数据进行建模和预测，通过遍历不同的 ARIMA 模型参数组合，在给定的参数范围内寻找最佳模型。选择最小 AIC 值的模型作为最优模型，并打印出其详细信息。

# 设置，q阶范围  
# product p,q的所有组合  
# 设置最好的aic为无穷大  
# 对范围内的p,q阶进行模型训练，得到最优模型  
import statsmodels.api as sm  
from itertools import product  
import warnings  
warnings.filterwarnings('ignore')  
ps = range(0,5)  
qs = range(0,5)  
parameters = product(ps, qs)  
parameters\_list = list(parameters)  
best\_aic = float('inf')  
results = []  
for param in parameters\_list:  
 try:  
 model = sm.tsa.arima.ARIMA(series, order=(param[0],0, param[1])).fit()  
 except ValueError:  
 print("参数错误：", param)  
 continue  
 aic = model.aic  
 if aic < best\_aic:  
 best\_model = model  
 best\_aic = model.aic  
 best\_param = param  
 results.append([param, model.aic])  
results\_table = pd.DataFrame(results)  
results\_table.columns = ['parameters', 'aic']  
print("最优模型", best\_model.summary())

最优模型 SARIMAX Results   
==============================================================================  
Dep. Variable: y No. Observations: 7201  
Model: ARIMA(4, 0, 3) Log Likelihood -13161.421  
Date: Thu, 21 Dec 2023 AIC 26340.843  
Time: 22:50:18 BIC 26402.781  
Sample: 0 HQIC 26362.155  
 - 7201   
Covariance Type: opg   
==============================================================================  
 coef std err z P>|z| [0.025 0.975]  
------------------------------------------------------------------------------  
const 15.3844 0.998 15.410 0.000 13.428 17.341  
ar.L1 1.7001 0.167 10.173 0.000 1.373 2.028  
ar.L2 -1.3320 0.302 -4.414 0.000 -1.924 -0.741  
ar.L3 1.1076 0.258 4.291 0.000 0.602 1.614  
ar.L4 -0.4817 0.105 -4.583 0.000 -0.688 -0.276  
ma.L1 -0.7543 0.168 -4.500 0.000 -1.083 -0.426  
ma.L2 0.5818 0.174 3.338 0.001 0.240 0.923  
ma.L3 -0.5245 0.111 -4.735 0.000 -0.742 -0.307  
sigma2 2.2637 0.009 249.968 0.000 2.246 2.281  
===================================================================================  
Ljung-Box (L1) (Q): 0.00 Jarque-Bera (JB): 972282.55  
Prob(Q): 0.99 Prob(JB): 0.00  
Heteroskedasticity (H): 1.13 Skew: -0.44  
Prob(H) (two-sided): 0.00 Kurtosis: 59.92  
===================================================================================  
  
Warnings:  
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

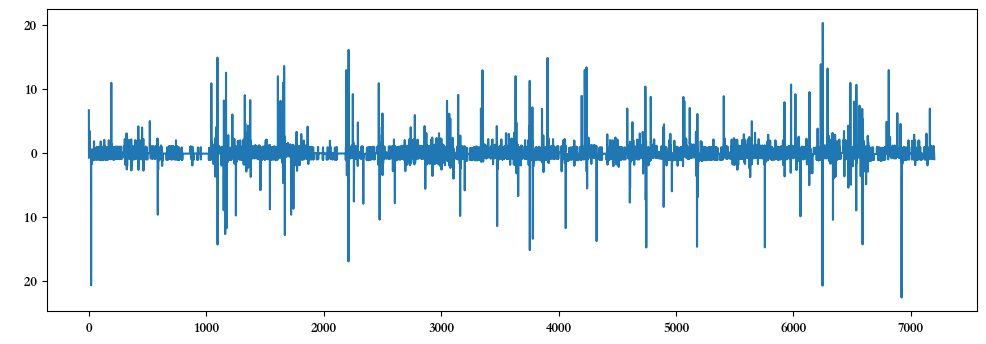
###### 打印最优 ARIMA 模型的信息，包括模型的参数、标准误差、AIC 值、BIC 值、残差等信息。

# 最佳模型ARIMA(3，0,3)  
print(best\_model.summary())

SARIMAX Results   
==============================================================================  
Dep. Variable: y No. Observations: 7201  
Model: ARIMA(4, 0, 3) Log Likelihood -13161.421  
Date: Thu, 21 Dec 2023 AIC 26340.843  
Time: 22:50:18 BIC 26402.781  
Sample: 0 HQIC 26362.155  
 - 7201   
Covariance Type: opg   
==============================================================================  
 coef std err z P>|z| [0.025 0.975]  
------------------------------------------------------------------------------  
const 15.3844 0.998 15.410 0.000 13.428 17.341  
ar.L1 1.7001 0.167 10.173 0.000 1.373 2.028  
ar.L2 -1.3320 0.302 -4.414 0.000 -1.924 -0.741  
ar.L3 1.1076 0.258 4.291 0.000 0.602 1.614  
ar.L4 -0.4817 0.105 -4.583 0.000 -0.688 -0.276  
ma.L1 -0.7543 0.168 -4.500 0.000 -1.083 -0.426  
ma.L2 0.5818 0.174 3.338 0.001 0.240 0.923  
ma.L3 -0.5245 0.111 -4.735 0.000 -0.742 -0.307  
sigma2 2.2637 0.009 249.968 0.000 2.246 2.281  
===================================================================================  
Ljung-Box (L1) (Q): 0.00 Jarque-Bera (JB): 972282.55  
Prob(Q): 0.99 Prob(JB): 0.00  
Heteroskedasticity (H): 1.13 Skew: -0.44  
Prob(H) (two-sided): 0.00 Kurtosis: 59.92  
===================================================================================  
  
Warnings:  
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

###### 绘制最优 ARIMA 模型的残差图。通过图像结果可以看出残差呈现随机且无规律的分布，点都随机地分布在0线周围，表明模型预测值与实际观测值之间没有明显的偏差，模型拟合效果较好。

resid=best\_model.resid #残差  
plt.figure(figsize=(12,4), dpi=100)  
plt.plot(resid)  
plt.show()



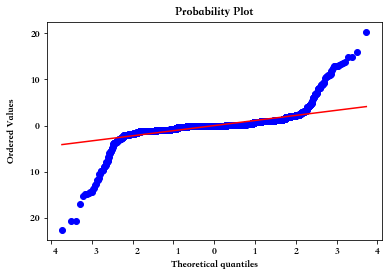
###### 对最优 ARIMA 模型的残差进行正态性检验。根据结果可知， p 值小于显著性水平，所以序列残差不符合正态分布。

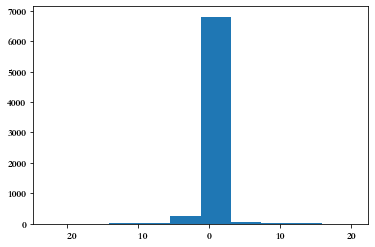
from scipy import stats   
print(stats.normaltest(resid))#检验序列残差是否为正态分布 pvalue< 0.05 拒绝原假设 认为残差符合正太分布

NormaltestResult(statistic=2691.63816134868, pvalue=0.0)

###### 绘制最优 ARIMA 模型的残差概率图和残差直方图。由图可知qq图散点基本在直线上，同时直方图也呈现正态性。

# p<0.05,接受备择假设，认为残差具有正态性  
stats.probplot(resid, dist="norm", plot=plt)  
plt.show()  
plt.hist(resid,bins=10)  
plt.show()





! pip install statsmodels

Looking in indexes: https://mirrors.aliyun.com/pypi/simple  
Requirement already satisfied: statsmodels in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (0.13.5)  
Requirement already satisfied: numpy>=1.17 in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from statsmodels) (1.19.5)  
Requirement already satisfied: packaging>=21.3 in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from statsmodels) (23.1)  
Requirement already satisfied: scipy>=1.3 in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from statsmodels) (1.10.1)  
Requirement already satisfied: patsy>=0.5.2 in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from statsmodels) (0.5.2)  
Requirement already satisfied: pandas>=0.25 in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from statsmodels) (1.3.4)  
Requirement already satisfied: pytz>=2017.3 in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from pandas>=0.25->statsmodels) (2023.3.post1)  
Requirement already satisfied: python-dateutil>=2.7.3 in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from pandas>=0.25->statsmodels) (2.8.2)  
Requirement already satisfied: six in /Users/huangboyuan/opt/anaconda3/envs/tf2/lib/python3.8/site-packages (from patsy>=0.5.2->statsmodels) (1.16.0)

###### 进行 Durbin-Watson 检验，评估最优 ARIMA 模型的残差序列中是否存在自相关性。DW检验值为 1.9973089496683567，接近2，可以认为在该模型中残差之间没有明显的一阶自相关性即残差之间的相关性较小。

# qq图散点基本在直线上，同时直方图也呈现正态性。  
from statsmodels.stats.stattools import durbin\_watson  
print("DW检验:", durbin\_watson(best\_model.resid)) ##DW检验：靠近2——正常；靠近0——正自相关；靠近4——负自相关

DW检验: 1.9973089496683567

###### 进行ARIMA模型的训练和预测，将数据系列的前70%作为训练集，剩下的30%作为ARIMA测试集。

# DW值靠近2，说明序列不具有相关性  
# 打印模型训练和测试， 计算损失  
from statsmodels.tsa.arima\_model import ARIMA  
from itertools import product  
import tqdm  
size=int(0.7\*len(series)) # 70%训练，30%测试  
train, test\_arima = series[:size], series[size:len(series)]  
history=[k for k in train]  
predictions = list()  
indexs1 = range(len(df.index[size:len(series)]))

###### 对 ARIMA 模型进行循环预测和测试，并将每次的预测结果和真实值打印输出。

c=0  
for t in range(len(test\_arima)-1):  
 c+=1  
 try:  
 model = sm.tsa.arima.ARIMA(history, order=(4,0,3))  
 model\_fit = model.fit()  
 output = model\_fit.forecast()  
 yhat = int(output[0])#预测  
 predictions.append(yhat)  
 obs = test\_arima[t]##真实值  
 history.append(obs)  
 print(f"all {len(test\_arima)} No.{c} {indexs1[c-1]}==> predicted=%d, expected=%d"% (yhat, obs))  
 except:  
 yhat = test\_arima[t]  
 predictions.append(yhat)  
 obs = test\_arima[t]  
 history.append(obs)  
 print(f"all {len(test\_arima)} No.{c} {indexs1[c-1]}==> predicted=%d, expected=%d"% (yhat, obs))

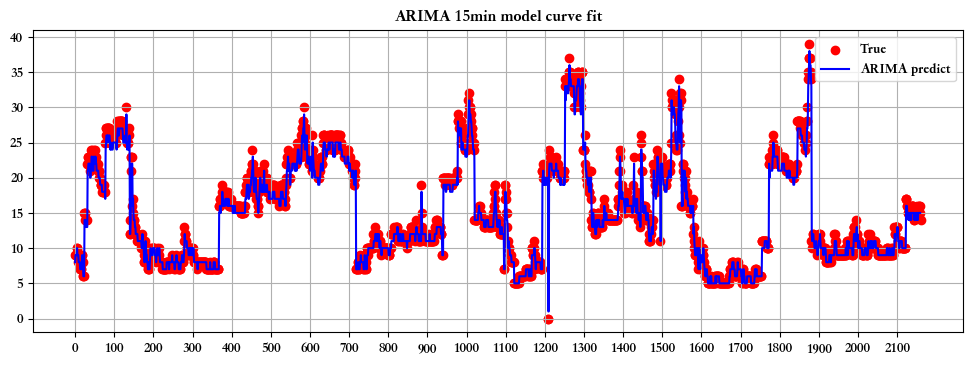
###### 计算 ARIMA 模型的预测结果与测试集实际数值之间的均方根误差（RMSE）和平均绝对误差（MAE）。根据结果可以看出和MAE和RMSE均较小，故表示模型预测的准确性较高。

# ARIMA模型的评价指标  
arima\_rmse = np.sqrt(mean\_squared\_error(test\_arima[:-1], predictions))  
arima\_mae = mean\_absolute\_error(test\_arima[:-1], predictions)  
  
# 均方根误差、平均绝对误差  
print("ARIMA 15min MAE:", arima\_mae)  
print("ARIMA 15min RMSE:", arima\_rmse)

ARIMA 15min MAE: 0.8157407407407408  
ARIMA 15min RMSE: 1.7234762459547768

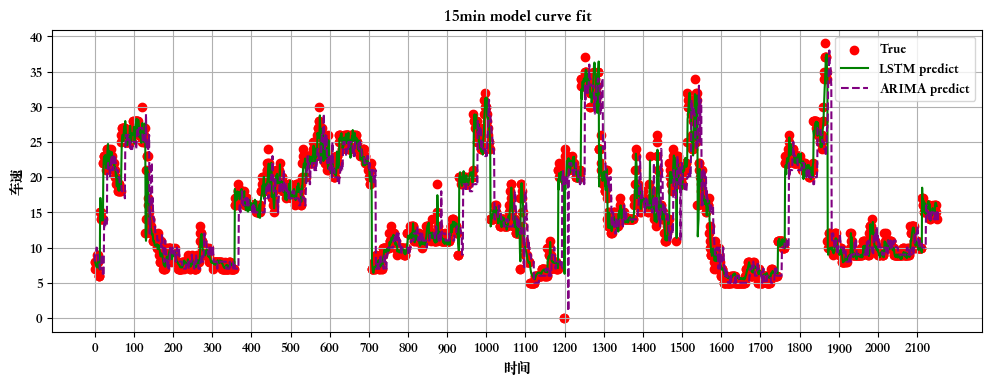
###### 绘制ARIMA模型在测试集上的预测结果与真实值之间的拟合情况，其中散点表示真实值，折线表示模型的预测值。根据图像可以看出两者之间趋势相似且接近，可见模型的拟合效果比较好。

# 拟合情况  
indexs1 = range(len(indexs1))  
plt.figure(figsize = (12,4), dpi=100)  
plt.scatter(indexs1, test\_arima, marker='o', label='True', c='red')  
plt.plot(indexs1[:-1], predictions, c='blue', label='ARIMA predict')  
plt.legend()  
plt.title("ARIMA 15min model curve fit")  
plt.grid()  
plt.xticks(np.arange(0, len(indexs1), 100))  
  
plt.show() # 可见拟合得很好



###### 绘制车速预测模型的结果图，包括真实值、LSTM 模型预测值和 ARIMA 模型预测值。其中散点表示真实值，实折线表示LSTM模型的预测值，虚折现表示ARIMA模型的预测值。根据图像可以看出三者之间趋势相似且接近，可见模型的拟合效果比较好。

# 两个模型的拟合--在一张图上  
# 拟合情况  
plt.figure(figsize = (12,4), dpi=100)  
plt.scatter(indexs, inverse\_test, marker='o', label='True', c='red')  
plt.plot(indexs, inverse\_lstm\_pred, c='green', label='LSTM predict')  
plt.plot(indexs1[:-1], predictions, c='purple', label='ARIMA predict',  
 linestyle='--')  
  
plt.legend()  
plt.title("15min model curve fit")  
plt.xlabel("时间")  
plt.ylabel("车速")  
plt.grid()  
plt.xticks(np.arange(0, len(indexs), 100))  
plt.show() # 可见拟合得很好



###### 绘制LSTM模型和 ARIMA模型在 MAE 和 RMSE 两个指标上的表现对比。根据图像可以看出LSTM比ARIMA在预测精度和稳定性上更优。

# 指标比较  
plt.figure(figsize=(12,5),dpi=100)  
xlabel = ['MAE','RMSE']  
x=np.arange(len(xlabel))  
  
y = [lstm\_mae, lstm\_rmse]  
y1 = [arima\_mae, arima\_rmse]  
bar\_width = 0.35  
rect1=plt.bar(x, y, bar\_width, align="center", color="c", label="LSTM", alpha=0.5)  
rect2=plt.bar(x+bar\_width, y1, bar\_width, color="b", align="center", label="ARIMA", alpha=0.5)  
plt.xlabel("指标")  
plt.ylabel("值")  
plt.xticks(x+bar\_width/2, xlabel)  
  
for rect in rect1: #rects 是三根柱子的集合  
 height = round(rect.get\_height(),2)  
 plt.text(rect.get\_x() + rect.get\_width() / 2, height, str(height), size=15, ha='center', va='bottom')  
for rect in rect2: #rects 是三根柱子的集合  
 height = round(rect.get\_height(),2)  
 plt.text(rect.get\_x() + rect.get\_width() / 2, height, str(height), size=15, ha='center', va='bottom')  
plt.legend()  
plt.grid()  
plt.show()

# **结果评测/展示**

**见视频附件**

# **总结**

工作总结：

在这项研究中，我们致力于通过人工智能和数据挖掘技术来实现车辆速度预测。通过深入挖掘大量的交通数据，我们成功地建立了一套高效而准确的车辆速度预测模型。我们的研究结果表明，利用先进的深度学习和机器学习算法，我们能够更精准地捕捉车辆行为模式，实现对城市交通动态的实时监测和预测。这为城市交通管理和规划提供了有力的工具，有望改善交通效率，减缓交通拥堵，并提高交通系统的可持续性。

通过我们的工作，我们不仅成功地应用了前沿的人工智能技术解决实际的城市交通问题，而且为未来智能交通系统的发展提供了有益的经验。我们的模型不仅在模拟环境中表现出色，而且在实际城市交通场景中也取得了良好的效果，这为将来在其他城市的推广和应用奠定了基础。

局限性分析：

尽管我们取得了一系列令人满意的研究成果，但我们也要认识到研究中存在一些局限性。首先，我们的模型仍然受到交通环境的复杂性和不确定性的制约，例如突发事件、人为因素等，这些因素可能影响预测的准确性。在未来的研究中，我们需要进一步优化模型，提高其对复杂交通场景的适应性。

其次，数据质量和可用性也是我们面临的挑战。虽然我们尽力采用了大量的交通数据，但数据的质量和时效性仍然对模型的性能产生了影响。未来的研究需要更加注重数据的采集和处理，以确保模型在真实场景中的鲁棒性和可靠性。

此外，我们的研究主要集中在车辆速度预测领域，而对于其他交通要素的预测（如交叉口流量、车辆密度等）尚未深入研究。未来的工作应该继续拓展研究范围，以建立更全面、多元化的智能交通系统。

综上所述，尽管存在一些局限性，我们的研究为人工智能在车辆速度预测领域的应用提供了有益的启示，并为未来的研究和实际应用奠定了基础。