**基于Spark与大语言模型的融合应用**

# **一、实验环境**

Ubuntu 16.04

Hadoop 3.1.3

Spark 3.0.0

PySpark 3.0.0

Python 3.12.4

JDK 1.8.0\_421

大语言模型的环境如下：

CUDA 11.8

Pytorch 2.2.2

Transformers 4.48.0

Bitsandbytes 0.45.0

Sqlparse 0.5.2

# **二、数据预处理**

## **2.1数据集简介**

本次实验采用的数据集是贵州茅台公司从2004年1月1日到2024年5月10日期间的每日股票行情数据，使用开源股票数据接口Tushare和Baostock获得。该数据集的各个字段说明如下：



## **2.2数据集处理**

本次实验使用pandas读取数据并且进行数据收集和预处理。具体步骤如下：

1.开源股票数据接口所提供的数据并不齐全，并存在数据缺失的现象，因此本次实验结合两个开源股票数据接口来获得数据，两个数据接口各自特点如下：（1）Tushare：常规数据（例如开盘价、交易量等）齐全，其在服务端已经过手工调整，数据中不存在缺失的现象，但所提供的数据种类不够丰富。（2）Baostock：数据种类更加丰富，包括peTTM等扩展数据，但其数据中存在较多缺失及错误。我们调用两个接口分别获取字段，对于常规数据使用Tushare的结果，并进一步使用Baostock获取更丰富的数据，如peTTM、pbMRQ、psTTM、pcfNcfTTM字段。

2.接下来我们需要使用pandas将两种数据来源的结果进行合并，在两种数据集中日期格式表示不一致，前者为存数字的20240516形式，而后者为2024-05-16形式，这里首先使用pandas的to\_datetime和匿名函数进行转换，再用pandas的merge功能按照日期进行合并，代码如下：

1. *# 使用pandas进行格式化统一不同数据来源的日期格式以便下一步合并DataFrame*
2. k\_data\_df['date'] = pd.to\_datetime(k\_data\_df['date'])
3. k\_data\_df['trade\_date'] = k\_data\_df['date'].apply(lambda x: x.strftime('%Y%m%d'))
4. k\_data\_df = k\_data\_df.drop(['date', 'code'], axis=1)
5. *# 使用pandas合并两个DataFrame*
6. *# 将每日K线数据和每日滚动市盈率、滚动市销率合并到一个DataFrame中*
7. hist\_data = pd.merge(hist\_data, k\_data\_df, on='trade\_date')

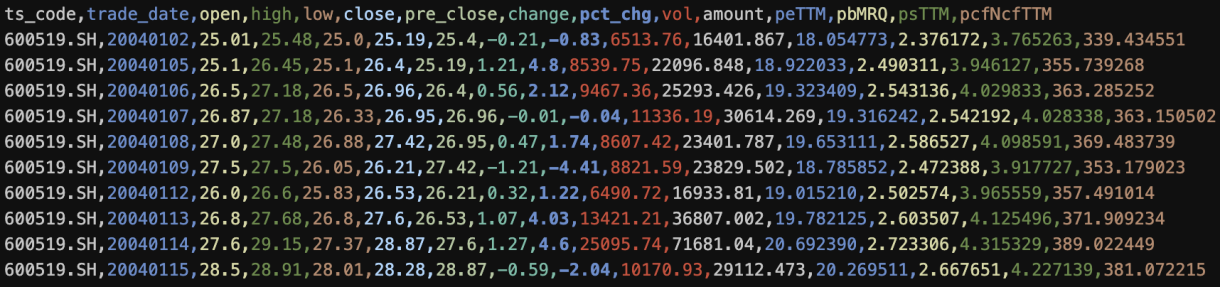
language-python

3.由于合并后的数据在部分字段存在数据缺失现象，我们按照实际分析股票的思路，使用前值填充的方式对数据进行补全。注意到数据中还存在异常的0值（某些字段的数值是不能为0的），我们同样使用pandas对这些值进行处理。代码如下：

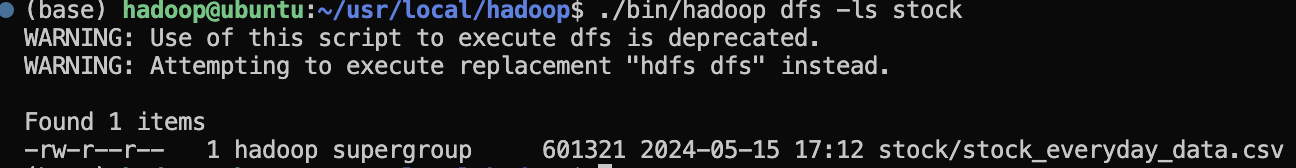
1. *# 部分字段存在缺失现象，根据股票数据的特点，使用前值填充的方式填充缺失值*
2. hist\_data['peTTM'].fillna(method='ffill', inplace=True)
3. hist\_data['pbMRQ'].fillna(method='ffill', inplace=True)
4. hist\_data['psTTM'].fillna(method='ffill', inplace=True)
5. hist\_data['pcfNcfTTM'].fillna(method='ffill', inplace=True)
6. *# 部分股票数据存在异常0值（这些值不应该为0），仍使用前值填充的方式填充0值*
7. hist\_data['peTTM'].replace(0, method='ffill', inplace=True)
8. hist\_data['pbMRQ'].replace(0, method='ffill', inplace=True)
9. hist\_data['psTTM'].replace(0, method='ffill', inplace=True)
10. hist\_data['pcfNcfTTM'].replace(0, method='ffill', inplace=True)

language-python

4.处理后的数据样例如下：

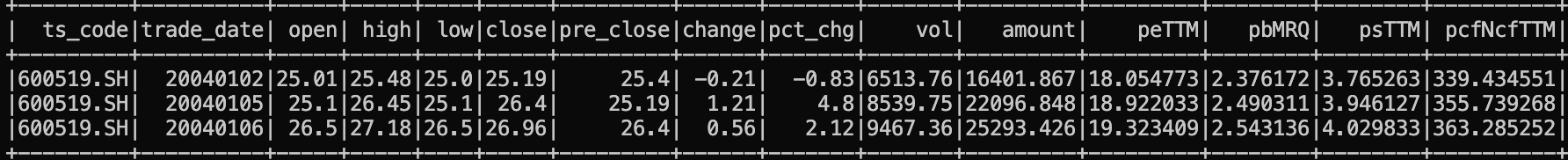


5.上传数据到HDFS：首先./bin/hadoop dfs -mkdir stock创建一个目录用于存储股票数据，然后使用./bin/hadoop dfs -put /home2/hadoop/stock\_everyday\_data.csv stock上传本地数据集到HDFS，使用ls命令查看上传结果：



# **三、数据分析**

本次实验我们使用Spark进行数据分析（编程语言使用Python），首先我们使用pyspark库读取数据并创建视图（以利用 SQL 来进行数据分析），通过show(3)我们可以验证数据是否读取成功：



1.分析每年贵州茅台股票的平均交易量

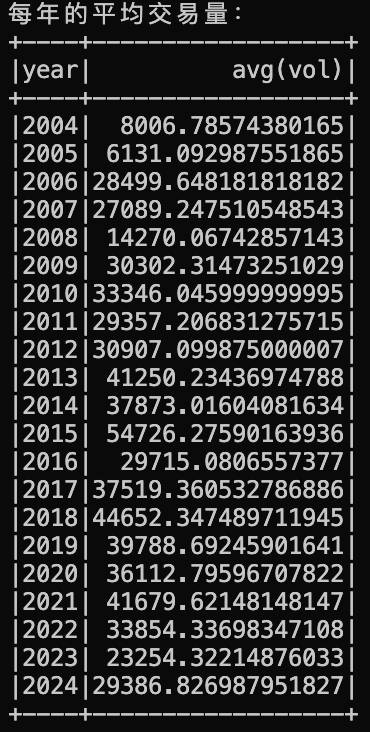
首先使用spark的year函数提取所有条目对应的年份，并增添新的一列保存年份。接下来，使用groupBy方法按年份对DataFrame进行分组，并使用agg方法对每个组（即每一年）应用聚合函数。在这种情况下，聚合函数是avg，用于计算每年交易量的平均值。最后，结果按年份排序，以便从最早到最近的顺序显示每年的平均交易量。

代码如下：

1. *# 将日期字段转换为年份*
2. df = df.withColumn("year", year("trade\_date"))
3. *# 计算每年的平均交易量*
4. yearly\_avg\_volume = df.groupBy("year").agg({"vol": "avg"}).orderBy("year")
5. *# 显示结果*
6. print("每年的平均交易量：")
7. yearly\_avg\_volume.show(100)
8. *#保存在文件中*
9. yearly\_avg\_volume.toPandas().to\_csv("result/yearly\_avg\_volume.csv", index=True)

language-python

计算结果如下：



2.分析股票收盘价的波动：

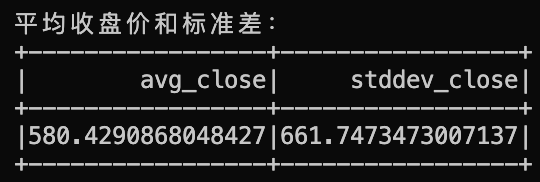
使用 select 方法选择收盘价列，并使用 avg 和 stddev 聚合函数计算收盘价的平均值和标准差。

代码如下：

1. *# 计算收盘价的平均值和标准差*
2. closing\_stats = df.select(avg("close").alias("avg\_close"), stddev("close").alias("stddev\_close"))
3. *# 显示结果*
4. print("平均收盘价和标准差：")
5. closing\_stats.show()

language-python

分析结果如下：



3.分析涨跌幅与各数据的相关性：

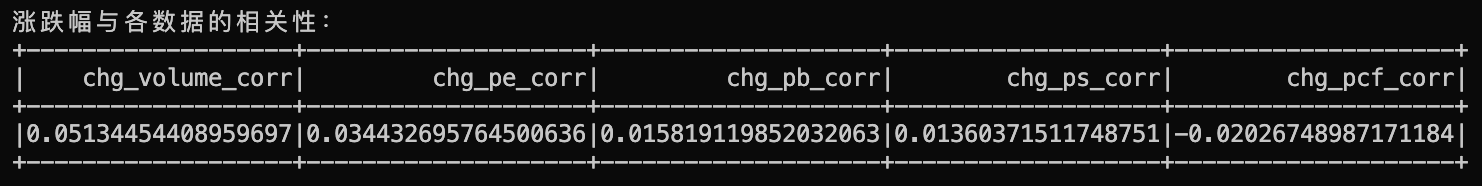
联合使用 select 方法和 corr 函数计算涨跌幅与成交量、涨跌幅、peTTM、pbMRQ、psTTM 和 pcfNcfTTM 的相关性。结果将被合并到一张表中，以便比较股价涨跌幅与不同数据之间的相关性。

代码如下：

1. *# 计算涨跌幅与成交量的相关性*
2. chg\_volume\_corr = df.select(corr("pct\_chg", "vol").alias("chg\_volume\_corr"))
3. *# 计算涨跌幅和peTTM的相关性*
4. chg\_pe\_corr = df.select(corr("pct\_chg", "peTTM").alias("chg\_pe\_corr"))
5. *# 计算涨跌幅和pbMRQ的相关性*
6. chg\_pb\_corr = df.select(corr("pct\_chg", "pbMRQ").alias("chg\_pb\_corr"))
7. *# 计算涨跌幅和psTTM的相关性*
8. chg\_ps\_corr = df.select(corr("pct\_chg", "psTTM").alias("chg\_ps\_corr"))
9. *# 计算涨跌幅和pcfNcfTTM的相关性*
10. chg\_pcf\_corr = df.select(corr("pct\_chg", "pcfNcfTTM").alias("chg\_pcf\_corr"))
11. *#合并上述结果在一张表中*
12. correlation = chg\_volume\_corr.crossJoin(chg\_pe\_corr).crossJoin(chg\_pb\_corr).crossJoin(chg\_ps\_corr).crossJoin(chg\_pcf\_corr)
13. *#显示结果*
14. print("涨跌幅与各数据的相关性：")
15. correlation.show()
16. *#保存在文件*
17. correlation.toPandas().to\_csv("result/correlation.csv", index=True

language-python

分析结果如下：



可以看出股价与每日成交量具有一定的相关性，并且和pcfNcfTTM负相关，投资者在进行决策时可主要参考成交量指标。

4.计算各个月份股票的涨停和跌停次数：

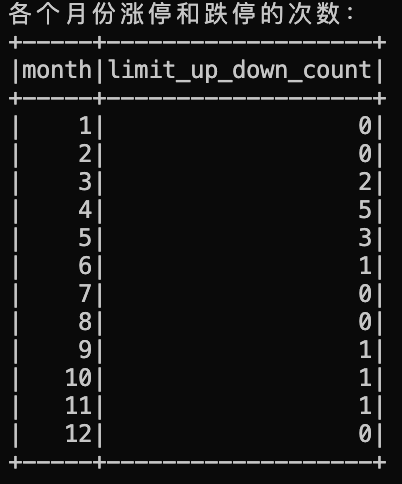
首先使用spark的month函数提取所有条目对应的月份，并增添新的一列保存月份。接下来，使用groupBy方法按月份对DataFrame进行分组，并使用agg方法对每个组应用聚合函数。在每一组中，首先使用when判断是否涨跌停，仅当涨跌停时值为1，使用sum函数计算1的数量从而实现对涨跌停次数的计算。

代码如下：

1. *# 将日期字段转换为月份*
2. df = df.withColumn("month", month("trade\_date"))
3. df = df.withColumn("pct\_chg", df["pct\_chg"].cast("float"))
4. df.show(3)
5. *# 计算每月涨停和跌停的次数*
6. monthly\_limit\_up\_down = df.groupBy("month").agg(sum(when((df["pct\_chg"] >= 10) | (df["pct\_chg"] <= -10), 1).otherwise(0)).alias("limit\_up\_down\_count"))
7. *# 显示结果*
8. print("各个月份涨停和跌停的次数：")
9. monthly\_limit\_up\_down.orderBy("month").show()
10. *#保存在文件中*
11. monthly\_limit\_up\_down.orderBy("month").toPandas().to\_csv("result/monthly\_limit\_up\_down.csv", index=True)

language-python

分析结果如下：



可以看见在3、4、5月份该股票常更容易出现涨跌停现象，风险厌恶型的投资者可以避免在该时段购买该股票以规避股价波动。

5.分析不同市场环境下的股票数据：

首先将涨跌幅字段转换为数值型，以便进行条件判断。使用 withColumn 方法根据涨跌幅的大小，定义市场环境规则并分类数据为“牛市”、“熊市”和“波动市”，添加对应列“market\_condition”。

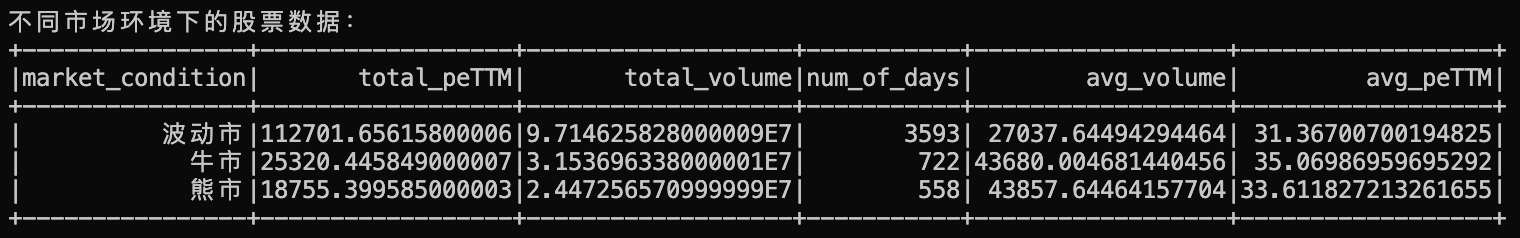
使用 groupBy 方法按市场环境对 DataFrame 进行分组，使用 agg 方法统计不同市场环境下的股票数据，包括交易天数、总交易量和总 peTTM。进一步计算每个市场环境下的平均交易量和平均 peTTM。

代码如下：

1. *#将涨跌幅字段转换为数值型*
2. df = df.withColumn("pct\_chg", df["pct\_chg"].cast("float"))
3. *# 定义市场环境规则并分类数据*
4. df = df.withColumn("market\_condition",
5. when(df["pct\_chg"] > 2, "牛市")
6. .when(df["pct\_chg"] < -2, "熊市")
7. .otherwise("波动市"))
8. *# 统计不同市场环境下的股票数据*
9. market\_summary = df.groupBy("market\_condition")\
10. .agg({"trade\_date": "count", "vol": "sum","peTTM":"sum"})\
11. .withColumnRenamed("count(trade\_date)", "num\_of\_days")\
12. .withColumnRenamed("sum(vol)", "total\_volume")\
13. .withColumnRenamed("sum(peTTM)", "total\_peTTM")
14. *# 计算每个市场环境下的平均交易量*
15. market\_summary = market\_summary.withColumn("avg\_volume", col("total\_volume") / col("num\_of\_days"))
16. *# 计算平均peTTM*
17. market\_summary = market\_summary.withColumn("avg\_peTTM", col("total\_peTTM") / col("num\_of\_days"))
18. *# 显示结果*
19. print("不同市场环境下的股票数据：")
20. market\_summary.show()
21. *#保存在文件中*
22. market\_summary.toPandas().to\_csv("result/market\_summary.csv", index=True)

language-python

分析结果：



可以看见在牛市或熊市，即股价有大幅度波动时，成交额比起波动环境会有显著增加，说明投资者们更偏好在股市有巨大波动时进行交易。

6.分析贵州茅台近20年的最高股价和最低股价：

直接使用 SQL 查询获取贵州茅台最高股价和最低股价对应的日期和数值，并使用 orderBy 方法对结果进行排序。这里注意需要把close转换为数值以确保正确比较。

代码如下：

1. sql\_result = spark.sql("""SELECT trade\_date, CAST(high AS DECIMAL) AS high\_numeric
2. FROM stock\_data
3. ORDER BY high\_numeric DESC
4. LIMIT 1;""")
5. print("20年来贵州茅台最高股价:")
6. sql\_result.show()
8. sql\_result = spark.sql("""SELECT trade\_date, CAST(low AS DECIMAL) AS low\_numeric
9. FROM stock\_data
10. ORDER BY low\_numeric ASC
11. LIMIT 1;""")
12. print("20年来贵州茅台最低股价:")
13. sql\_result.show()

language-python

分析结果如下：



可见在数据集中的最高股价出现在2021年2月18日，为2628元；而最低股价出现在2004年1月2日，为25元。

7.计算上周的平均涨幅：

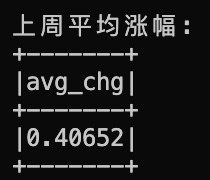
使用 SQL 查询计算上周的平均涨幅，通过 AVG 函数实现，同时结合 BETWEEN 条件来筛选上周的日期范围。

代码如下：

1. sql\_result = spark.sql("""SELECT AVG(pct\_chg) AS avg\_chg
2. FROM stock\_data
3. WHERE trade\_date BETWEEN '20240505' AND '20240510';
4. """)
5. print("上周平均涨幅:")
6. sql\_result.show()

language-python

分析结果如下：



可知在20240505至20240510期间贵州茅台的每日平均涨幅约为0.4%。

8.统计每年上涨和下跌天数

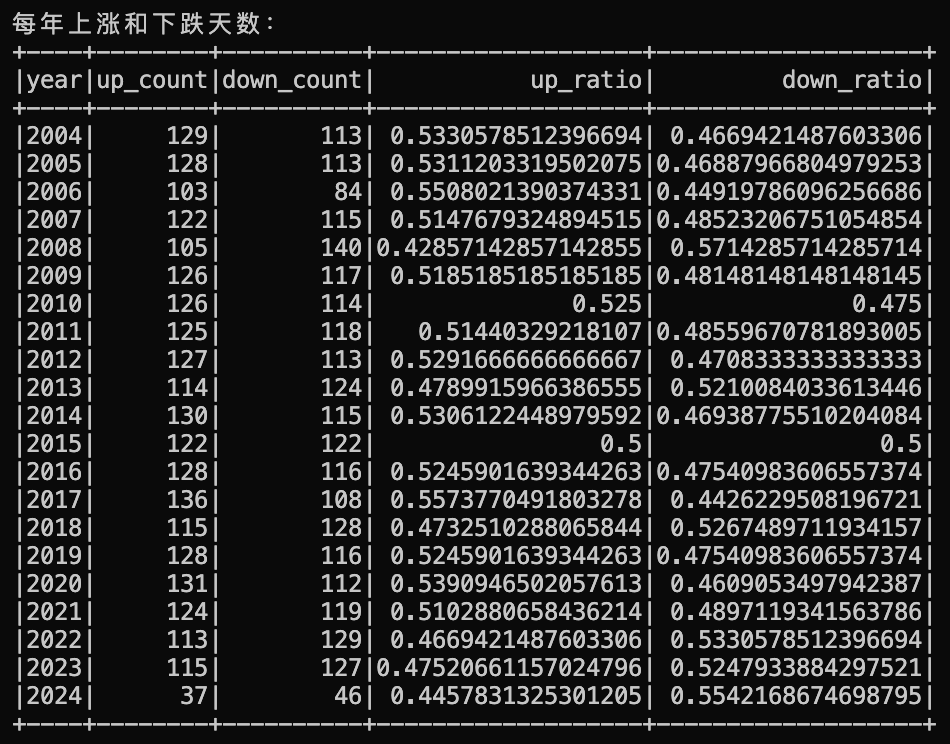
使用 groupBy 方法按年份对 DataFrame 进行分组。使用 agg 方法计算每年上涨和下跌天数，通过 sum 和 when 函数实现，并添加列计算上涨和下跌天数占总天数的比值。

代码如下：

1. *# 计算每月涨停和跌停的次数*
2. year\_up\_days = df.groupBy("year").agg(sum(when((df["pct\_chg"] >= 0), 1).otherwise(0)).alias("up\_count"))
3. year\_down\_days = df.groupBy("year").agg(sum(when((df["pct\_chg"] < 0), 1).otherwise(0)).alias("down\_count"))
4. *#合并在同一张表并按年份排序*
5. year\_up\_down\_days = year\_up\_days.join(year\_down\_days, "year").orderBy("year")
6. *#添加列：上涨天数占总天数的比值*
7. year\_up\_down\_days = year\_up\_down\_days.withColumn("up\_ratio", col("up\_count") / (col("up\_count") + col("down\_count")))
8. *#添加列：下跌天数占总天数的比值*
9. year\_up\_down\_days = year\_up\_down\_days.withColumn("down\_ratio", col("down\_count") / (col("up\_count") + col("down\_count")))
10. print("每年上涨和下跌天数：")
11. year\_up\_down\_days.show(100)
12. *#保存在文件中*
13. year\_up\_down\_days.toPandas().to\_csv("result/year\_up\_down\_days.csv", index=True)

language-python

分析结果如下：



可知贵州茅台每一年的涨跌状况，可以看见近几年其股票呈下跌趋势。

# **四、利用大语言模型（LLM）进行数据分析**

近年来，大型语言模型（Large Language Model, LLM）备受瞩目，其主要优势在于能够理解用户的自然语言指令并据此完成任务。然而，由于大型语言模型本质上是概率模型，在回答问题时可能存在“幻觉”，即出现随意编造内容的情况。特别是在需要回答准确数据的任务中，大型语言模型往往无法满足要求。此外，由于训练大型语言模型需要耗费大量时间和资源，其所学习到的知识可能滞后于现实，并且难以实时更新，因此容易提供过时的数据和知识。

相比之下，数据库具有多样化的查询方法，并且能够实时更新数据，因此可以提供准确的信息。然而，数据库也存在一个缺点，即普通用户缺乏对SQL语句等数据库操作工具的掌握，因此难以直接与数据库交互。

因此，本次实验旨在结合大型语言模型与数据库的优势，以弥补各自的不足，制作一个面向普通用户的数据分析方法。

## **4.1方法**

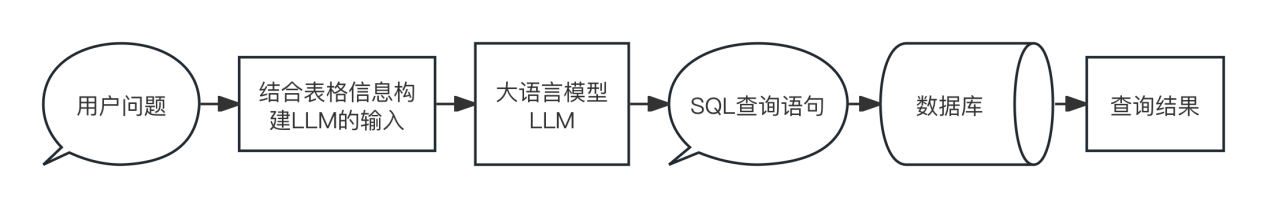
为了实现LLM与数据库的交互，本实验采用SQL查询语句作为纽带，即LLM接收用户的自然语言查询输入，例如“2024年哪一天股价最高”，大语言模型基于该输入生成SQL查询语句：

SELECT MAX(s.high) AS max\_high\_price\_date

FROM a\_stock\_everyday\_data s

WHERE s.trade\_date BETWEEN 20240101 AND 20241231

该SQL查询语句用于查询数据库并返回结果给用户，其流程可归纳为下图：



LLM通常以预训练-微调的范式进行应用，预训练后的LLM通常拥有了通用能力，能对用户的输入进行响应，为了进一步让LLM完成本实验的任务要求，我们需要让其学会以文字问题作为输入，并输出SQL查询语句（即Text2SQL任务）。

我们选择sqlcoder-7b-2作为语言模型，其在大量Text2SQL语料上进行了微调，可很好的实现生成SQL查询语句的任务，以下网址为该模型的Demo：<https://defog.ai/sqlcoder-demo/>

## **4.2具体代码实现**

本实验在本地构建了LLM与数据库的结合的pipeline，接下来是详细的介绍。

首先我们读入HDFS的文件，对其进行数据格式的转换，并构建视图用于SQL查询，代码如下：

1. spark = SparkSession.builder.getOrCreate()
2. df = spark.read.csv("stock\_everyday\_data.csv",header=True)
3. *#将a\_stock\_everyday\_data中除了trade\_date的字符串都转换为数值，以确保使用sql语句时能正常比较*
4. for col in df.columns:
5. if col != "trade\_date":
6. df = df.withColumn(col, df[col].cast("float"))
7. df.createOrReplaceTempView("a\_stock\_everyday\_data")

language-python

接下来，我们需要在本地加载LLM，模型参数量为7B，大约占用17G显存，这里使用Transformers库进行便捷部署：

1. from transformers import AutoTokenizer, AutoModelForCausalLM
2. model\_name = "defog/sqlcoder-7b-2"
3. tokenizer = AutoTokenizer.from\_pretrained(model\_name)
4. model = AutoModelForCausalLM.from\_pretrained(
5. model\_name,
6. trust\_remote\_code=True,
7. torch\_dtype=torch.float16,
8. *#load\_in\_4bit=True,*
9. device\_map="auto",
10. use\_cache=True,
11. )

language-python

在输入大语言模型时，需要一同给出我们的数据库结构，以便LLM能给出正确的SQL查询语句，因此我们需要构建Prompt模版，指明表格结构，在接收到用户的问题后，替换模版中的{question}内容，从而输入大语言模型，代码如下：

1. prompt = """<|begin\_of\_text|><|start\_header\_id|>user<|end\_header\_id|>
3. Generate a SQL query to answer this question: `{question}`
5. DDL statements:
7. CREATE TABLE a\_stock\_everyday\_data (
8. trade\_date DATE, -- Trading date交易日期，以纯数字形式给出，例如20240515
9. open DECIMAL(10,2), -- Opening price开盘价
10. high DECIMAL(10,2), -- Highest price当日最高价
11. low DECIMAL(10,2), -- Lowest price当日最低价
12. close DECIMAL(10,2), -- Closing price收盘价
13. pre\_close DECIMAL(10,2), -- Previous closing price前一天的收盘价
14. change DECIMAL(10,2), -- Price change股价变化
15. pct\_chg DECIMAL(10,2), -- Price change percentage股价变化率
16. vol BIGINT, -- Trading volume成交额
17. amount DECIMAL(15,2), -- Trading amount成交量
18. );
20. <|eot\_id|><|start\_header\_id|>assistant<|end\_header\_id|>
22. The following SQL query best answers the question `{question}`:
23. ```sql
24. """

language-python

构建完成模型输入后，我们需要调用LLM并生成对应结果，在LLM输出结果后，还需要进行解析和格式化处理，为了便于代码复用，我们将调用流程封装为以下函数：

1. def generate\_query(question):
2. updated\_prompt = prompt.format(question=question)
3. inputs = tokenizer(updated\_prompt, return\_tensors="pt").to("cuda")
4. *# 生成SQL语句的参数，确保do\_sample=False以保证生成的SQL语句是确定性的并且可复现*
5. generated\_ids = model.generate(
6. \*\*inputs,
7. num\_return\_sequences=1,
8. eos\_token\_id=tokenizer.eos\_token\_id,
9. pad\_token\_id=tokenizer.eos\_token\_id,
10. max\_new\_tokens=400,
11. do\_sample=False,
12. num\_beams=1,
13. )
14. outputs = tokenizer.batch\_decode(generated\_ids, skip\_special\_tokens=True)
16. torch.cuda.empty\_cache()
17. torch.cuda.synchronize()
19. return outputs[0].split("```sql")[1].split(";")[0]

language-python

最后，是接收用户输入并调用模型获取SQL查询语句，在通过spark进行SQL查询的代码：

1. while(True):
2. question = input("请输入一个问题：")
3. if question == "exit":
4. break
5. generated\_sql = generate\_query(question)
6. print("->LLM生成的SQL语句为：",end='')
7. print(sqlparse.format(generated\_sql, reindent=True))
8. sql\_result = spark.sql(generated\_sql)
9. print("->查询结果为：")
10. sql\_result.show()

language-python

## **4.2实验结果**

1.查询2024年股价最高的日子

LLM针对问题生成SQL查询语句，该语句直接被用与查询数据库。



2.查询20年来最高的开盘价



3.查询某一天的数据



# **五、数据可视化**

为了使分析结果更加直观，本次实验还根据不同的分析项目设计了不同可视化结果。可视化主要通过matplotlib实现。

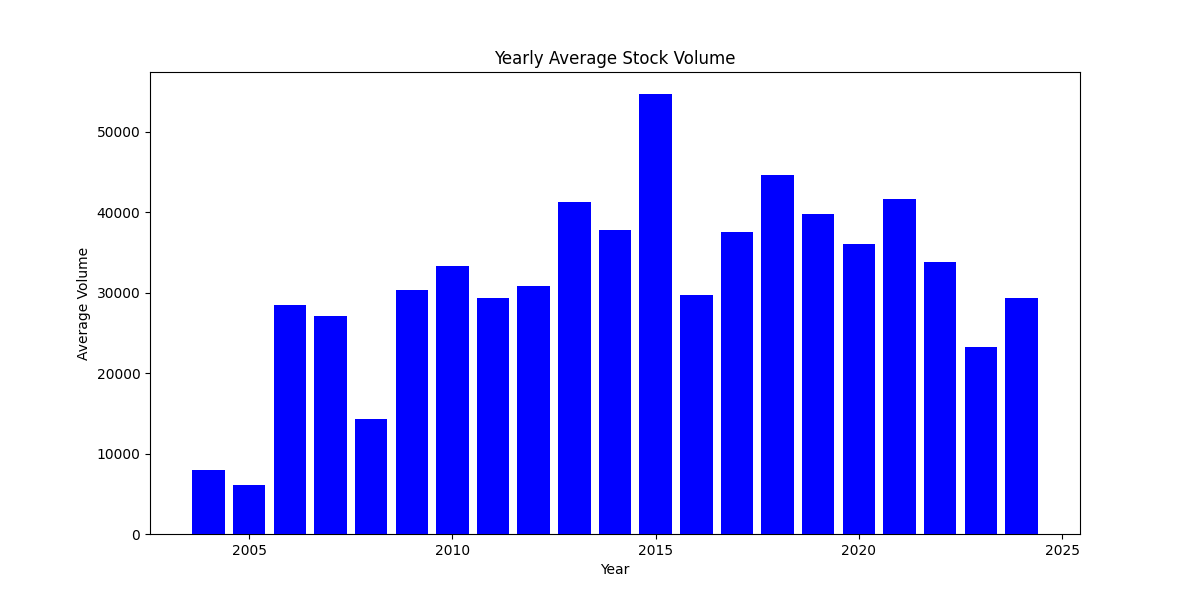
## **1.可视化每年股票的平均交易量**

这里使用柱状图展示每年股票的平均交易量，可以直观体现出每一年的交易活跃度，其中2015年为整个市场的牛市，交易量显著高于其它年份。

代码如下：

1. *#读取文件*
2. yearly\_avg\_volume = pd.read\_csv("result/yearly\_avg\_volume.csv", index\_col=0)
3. *# 绘制柱状图*
4. plt.figure(figsize=(12, 6))
5. plt.bar(yearly\_avg\_volume['year'], yearly\_avg\_volume['avg(vol)'], color='b', label='Average Volume')
6. *# 设置图表标题和标签*
7. plt.title('Yearly Average Stock Volume')
8. plt.xlabel('Year')
9. plt.ylabel('Average Volume')
10. plt.savefig("result/yearly\_avg\_volume.png")

language-python

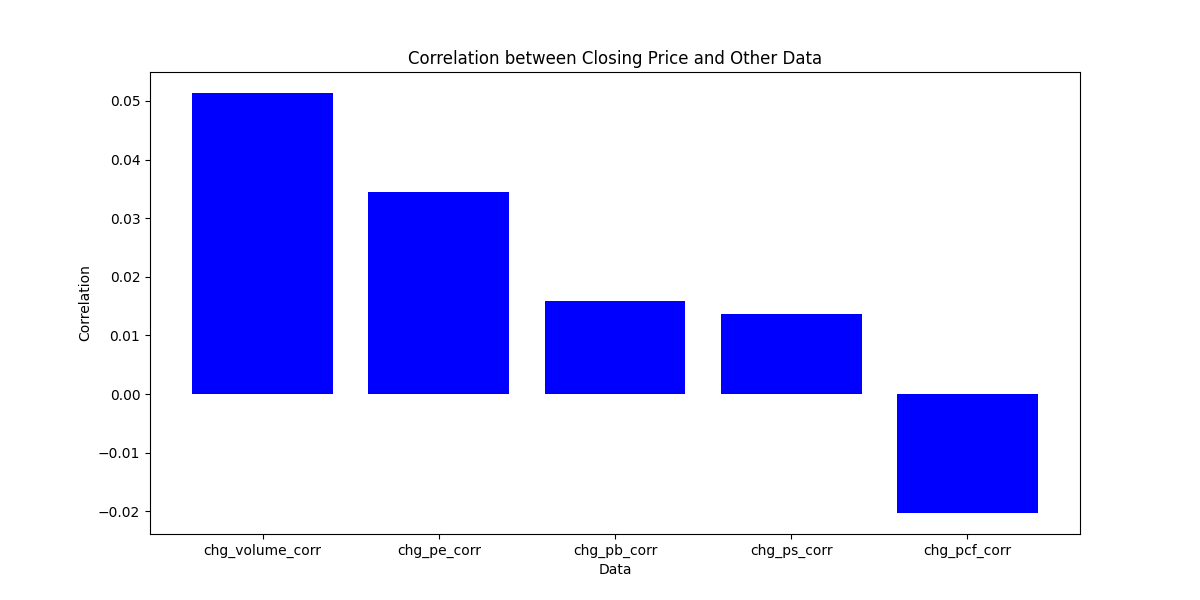


## **2.可视化收盘价与各数据的的相关性**

同样使用柱状图展示，代码如下：

1. *#读取文件*
2. correlation = pd.read\_csv("result/correlation.csv", index\_col=0)
3. *# 绘制柱状图*
4. plt.figure(figsize=(12, 6))
5. plt.bar(correlation.columns, correlation.iloc[0], color='b', label='Correlation')
6. *# 设置图表标题和标签*
7. plt.title('Correlation between Closing Price and Other Data')
8. plt.xlabel('Data')
9. plt.ylabel('Correlation')
10. plt.savefig("result/correlation.png")

language-python



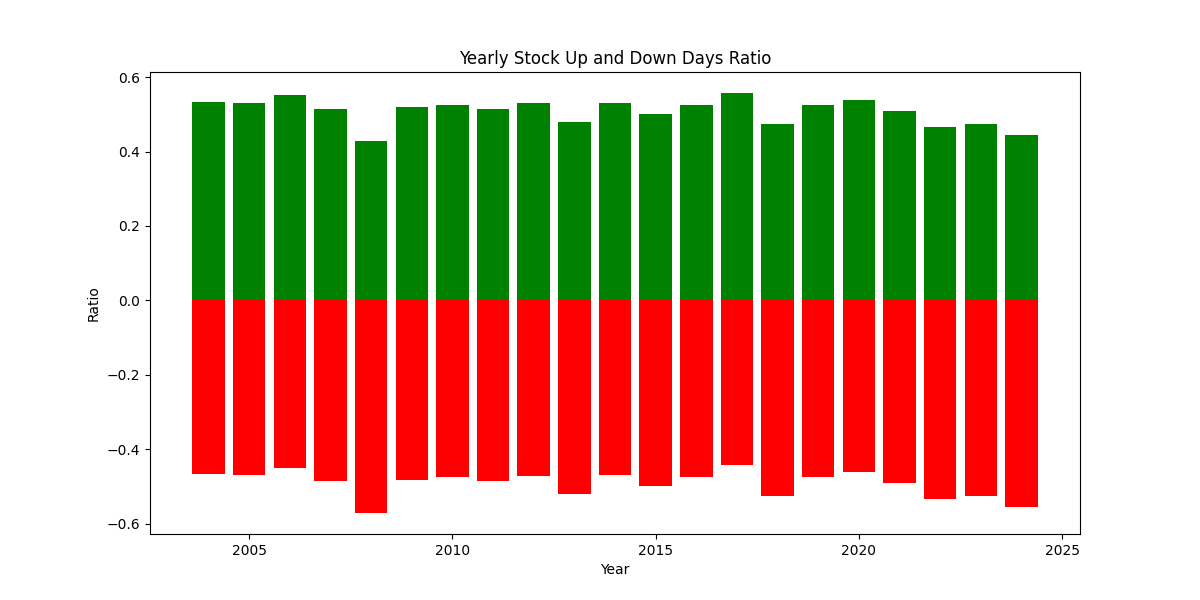
## **3.可视化每年股票上涨和下跌天数的比例**

为了直观展示各年之间上涨的下跌天数的对比，这里采用堆叠的柱状图进行可视化，并以绿色和红色直观的代表上涨和下跌。

代码如下：

1. *# 将下跌天数变为负值*
2. year\_up\_down\_days['down\_ratio'] = -year\_up\_down\_days['down\_ratio']
3. *# 绘制柱状图，对齐红绿交界处*
4. plt.figure(figsize=(12, 6))
5. plt.bar(year\_up\_down\_days['year'], year\_up\_down\_days['up\_ratio'], color='g', label='Up Days Ratio')
6. plt.bar(year\_up\_down\_days['year'], year\_up\_down\_days['down\_ratio'], color='r', label='Down Days Ratio')
7. *# 设置图表标题和标签*
8. plt.title('Yearly Stock Up and Down Days Ratio')
9. plt.xlabel('Year')
10. plt.ylabel('Ratio')
11. plt.savefig("result/year\_up\_down\_days\_ratio.png")

language-python



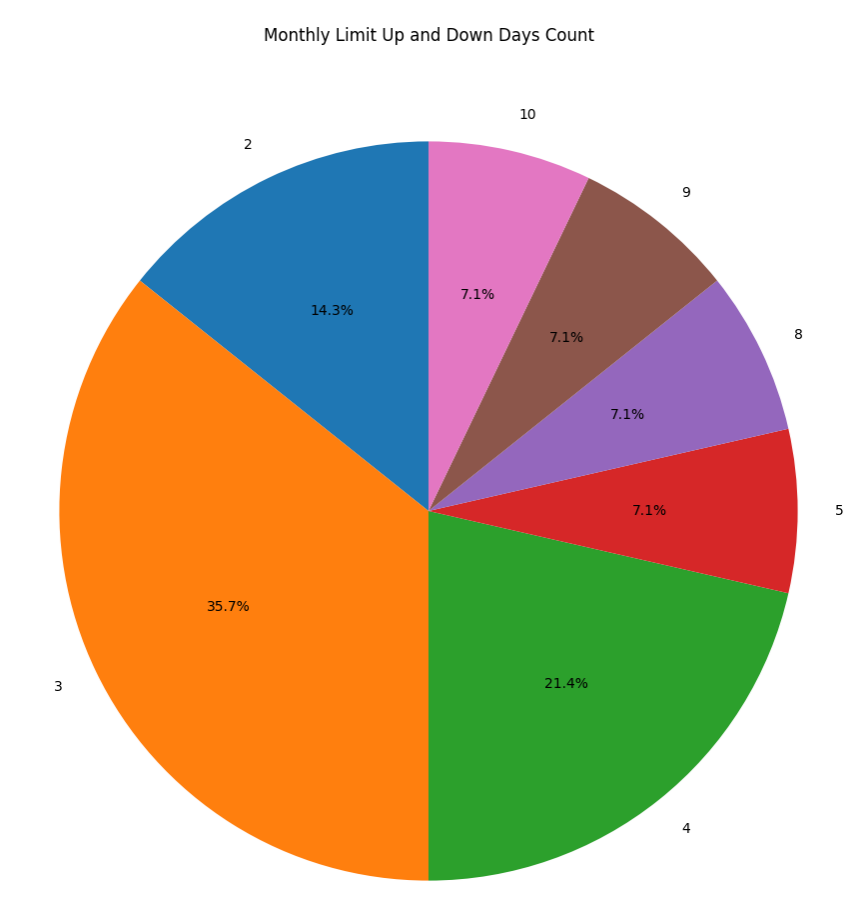
可以看见，在近几年绿色柱逐渐变小，而红色柱逐年增长，说明了下跌天数的占比逐渐变高，贵州茅台股票的行情转弱。

## **4.可视化每个月涨跌停次数**

为了体现出涨跌停与月度关系，并体现出其相关性，这里采用饼状图展示。

1. *# 绘制饼状图*
2. plt.figure(figsize=(12, 12))
3. *#部分月份为0，删去*
4. month\_limit\_up\_down = month\_limit\_up\_down[month\_limit\_up\_down['limit\_up\_down\_count'] != 0]
5. plt.pie(month\_limit\_up\_down['limit\_up\_down\_count'], labels=month\_limit\_up\_down.index, autopct='%1.1f%%', startangle=90)
6. plt.title('Monthly Limit Up and Down Days Count')
7. plt.savefig("result/monthly\_limit\_up\_down.png")

language-python



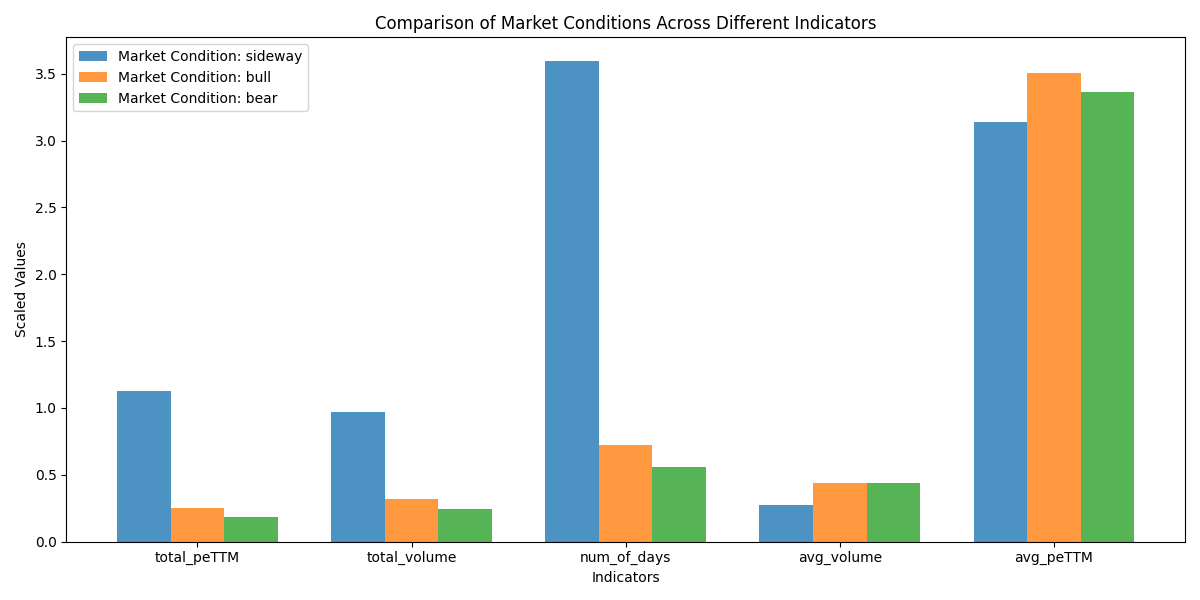
可以直观看出大多数涨跌停出现在三、四月份，说明三四月份贵州茅台行情波动大。

## **5.可视化不同市场环境下的股票数据**

为了体现出不同环境下各个指标的区别，这里选择簇状柱状图进行可视化，同时，考虑到各个指标的数据量级不一样，还需要进行适当放缩，以便其在同一张图上显示。

1. *# 创建图表和坐标轴*
2. fig, ax = plt.subplots(figsize=(12, 6))
3. *# 指标名称*
4. indicators = ['total\_peTTM', 'total\_volume', 'num\_of\_days', 'avg\_volume', 'avg\_peTTM']
5. *# 对某些指标进行缩放*
6. scale\_factors = {'total\_peTTM': 1e-5, 'total\_volume': 1e-8, 'num\_of\_days': 1e-3, 'avg\_volume': 1e-5, 'avg\_peTTM': 1e-1}
7. *# 数据准备*
8. n\_groups = len(indicators)
9. index = np.arange(n\_groups) *# 指标数量*
10. bar\_width = 0.25
11. opacity = 0.8
12. *# 循环绘制每个市场状态*
13. condition\_name = ['sideway','bull','bear']
14. for i, condition in enumerate(market\_env.index):
15. values = market\_env.loc[condition, indicators] \* pd.Series(scale\_factors)
16. plt.bar(index + i \* bar\_width, values, bar\_width,
17. alpha=opacity, label=f'Market Condition: {condition\_name[i]}')
18. *# 设置图表标题和标签*
19. plt.title('Comparison of Market Conditions Across Different Indicators')
20. plt.xlabel('Indicators')
21. plt.ylabel('Scaled Values')
22. plt.xticks(index + bar\_width, indicators)

language-python



可见波动的市场占据大多数天数，并且在波动的市场下，平均交易量和peTTM都会更低。

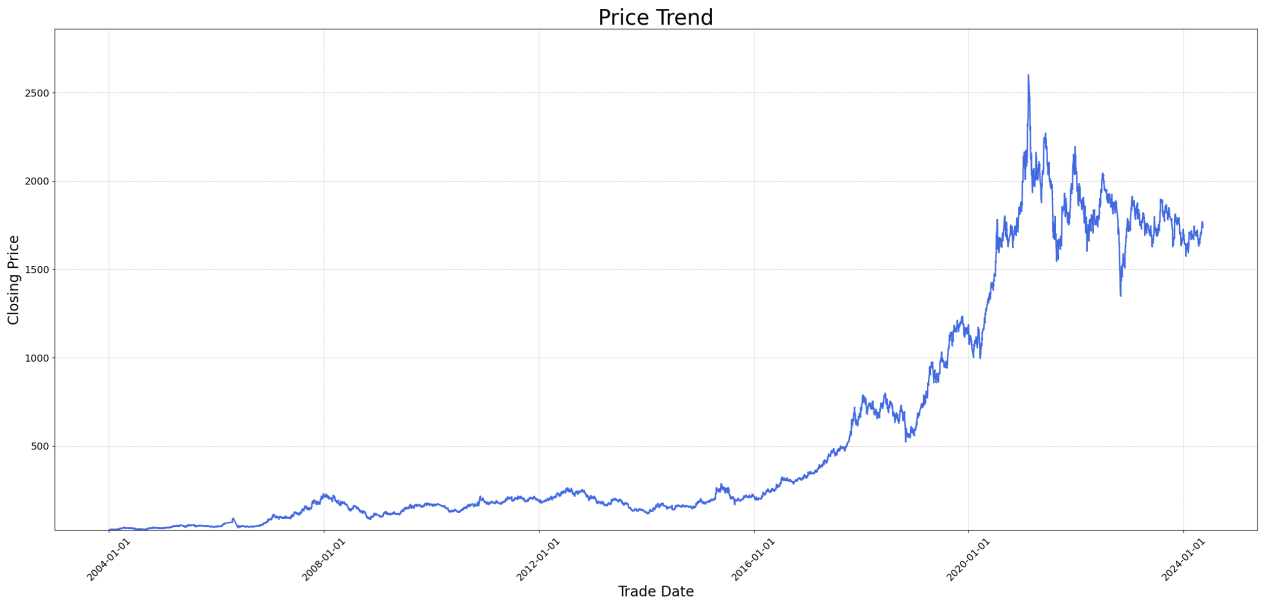
## **6.可视化贵州茅台的股票价格趋势**

当需要反映数据趋势时，通常选择折线图，股价的波动也适合用折线图来表示。

1. *# 获取最大和最小的收盘价*
2. max\_close = pandas\_df['close'].max()
3. min\_close = pandas\_df['close'].min()
4. *# 把pandas\_df['trade\_date']转换为list*
5. trade\_date\_list = pandas\_df['trade\_date'].tolist()
6. close\_list = pandas\_df['close'].tolist()
7. *# 绘制折线图*
8. plt.figure(figsize=(25, 12))
9. plt.plot(trade\_date\_list, close\_list, marker='o', linestyle='-', color='royalblue', markersize=1, linewidth=2, label='Close Price')
10. plt.title('Price Trend', fontsize=30)
11. plt.xlabel('Trade Date', fontsize=20)
12. plt.ylabel('Closing Price', fontsize=20)
13. plt.xticks(rotation=45)
14. plt.grid(True, linestyle='--', linewidth=0.5)
15. plt.tick\_params(axis='both', labelsize=14)
16. plt.ylim(min\_close \* 0.9, max\_close \* 1.1)
17. plt.gca().xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m-%d'))
18. plt.gca().xaxis.set\_major\_locator(mdates.AutoDateLocator())
19. plt.tight\_layout() *# 自动调整布局*

language-python

20年来股价变化趋势：



近一年股价变化趋势：



# **六、总结**

本次实验在Ubuntu 16.04操作系统上，利用Hadoop 3.3.5和Spark 3.2.0框架，结合Python 3.8和JDK 1.8，对贵州茅台公司自2004年1月1日至今的每日股票行情数据进行了深入分析。实验首先通过Tushare和Baostock两个开源接口获取数据，并使用pandas库进行数据预处理，处理了数据缺失和异常值，然后将清洗后的数据上传至HDFS。

数据分析阶段，我们运用Spark和Python对数据进行了多维度的分析，包括计算年均交易量、收盘价波动性、涨跌幅与其他财务指标的相关性、月份涨跌停次数、不同市场环境下的股票表现等。此外，还查询了特定时间段内的最高和最低股价，计算了平均涨幅，并统计了每年的上涨和下跌天数。

为了提高数据分析的准确性和用户交互的便捷性，本实验引入了大型语言模型（LLM），通过生成SQL查询语句，将用户的自然语言问题转化为数据库查询，再通过Spark执行并展示结果。所选模型sqlcoder-7b-2在Text2SQL任务上进行了微调，有效提升了查询的准确性和效率。

数据可视化方面，实验采用了matplotlib库，通过柱状图、饼状图、簇状柱状图和折线图等多种形式，直观展示了分析结果。这些图表不仅增强了数据的可读性，也帮助用户更直观地理解股票市场的趋势和模式。

总结来说，本次实验成功地展示了如何综合运用大数据技术、人工智能和数据可视化工具，对股票市场数据进行高效处理和深入分析。实验结果不仅为投资者提供了有价值的市场洞察，也证明了结合大数据分析和LLM在提升用户体验和数据处理能力方面的潜力。通过本次实验，我也加深了对大数据处理流程的理解，并提高了在实际问题中应用数据分析技术的能力。