**第一步：安装必要库**

在Jupyter Notebook的第一个单元格中运行

!pip install yfinance pandas-datareader numpy pandas matplotlib seaborn scipy statsmodels scikit-learn tensorflow shap plotly streamlit prophet nltk textblob jieba geopandas

!pip install --upgrade pandas-datareader # 确保是最新版本

**第二步：完整项目代码框架**

创建一个新的Python笔记本文件，按顺序添加以下单元格：

**1. 导入所有必要的库**

# 基础数据处理

import numpy as np

import pandas as pd

from datetime import datetime, timedelta

import warnings

warnings.filterwarnings('ignore')

# 数据获取

import yfinance as yf

from pandas\_datareader import data as pdr

import pandas\_datareader.fred as fred

# 可视化

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

from wordcloud import WordCloud

# 机器学习与统计分析

from scipy import stats

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.vector\_ar.var\_model import VAR

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from sklearn.model\_selection import TimeSeriesSplit

import shap

# 深度学习

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout, MultiHeadAttention, LayerNormalization

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

# NLP处理

import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

from textblob import TextBlob

import jieba

# 时间序列预测

from prophet import Prophet

# 仪表盘

import streamlit as st

# 地理可视化

import geopandas as gpd

**2. 数据获取与预处理**

def fetch\_extended\_data():

"""

获取扩展数据集（股票、宏观、事件、社交媒体）

"""

print("="\*50)

print("开始获取扩展数据...")

# 基本设置

end\_date = datetime.now()

start\_date = datetime(2020, 1, 1)

# 1. 股票数据（从Yahoo Finance获取）

indices = {

"CSI300": "000300.SS", # 沪深300

"SP500": "^GSPC", # 标普500

"STOXX50": "^STOXX50E" # 欧洲STOXX50

}

print("从Yahoo Finance下载股票数据...")

stock\_data = yf.download(

tickers=list(indices.values()),

start=start\_date,

end=end\_date,

group\_by='ticker'

)

# 重组股票数据结构

stock\_dfs = []

for name, ticker in indices.items():

df = stock\_data[ticker].copy()

df.columns = [f"{name}\_{col}" for col in df.columns]

stock\_dfs.append(df)

merged\_stocks = pd.concat(stock\_dfs, axis=1)

# 添加波动率指标

for name in indices.keys():

close\_col = f"{name}\_Close"

if close\_col in merged\_stocks.columns:

merged\_stocks[f"{name}\_Volatility"] = merged\_stocks[close\_col].pct\_change().rolling(30).std()

# 2. 宏观经济数据（从FRED获取）

macro\_indicators = {

'USDCNY': 'DEXCHUS', # 人民币汇率

'USDEUR': 'DEXUSEU', # 欧元汇率

'USCPI': 'CPIAUCSL', # 美国CPI

'FEDFUNDS': 'FEDFUNDS',# 联邦基金利率

'GPRC': 'GPRC', # 地缘政治风险指数

}

print("从FRED获取宏观经济数据...")

macro\_data = pdr.get\_data\_fred(

list(macro\_indicators.values()),

start=start\_date,

end=end\_date

)

# 3. 模拟事件数据（实际项目应接入GDELT API）

print("生成模拟事件数据...")

date\_range = pd.date\_range(start=start\_date, end=end\_date, freq='D')

event\_data = pd.DataFrame(index=date\_range)

# 模拟地缘政治事件强度

event\_data['Event\_Intensity'] = np.random.uniform(0, 1, len(date\_range))

# 模拟重要事件标记（0-无事件，1-有事件）

event\_data['Major\_Event'] = np.random.choice([0, 1], len(date\_range), p=[0.95, 0.05])

# 4. 模拟社交媒体数据（实际项目应接入Twitter/微博API）

print("生成模拟社交媒体数据...")

social\_data = pd.DataFrame(index=date\_range)

# 模拟情感得分（-1到1）

social\_data['Sentiment'] = np.random.uniform(-1, 1, len(date\_range))

# 合并所有数据

merged\_data = pd.merge(merged\_stocks, macro\_data, left\_index=True, right\_index=True, how='left')

merged\_data = pd.merge(merged\_data, event\_data, left\_index=True, right\_index=True, how='left')

merged\_data = pd.merge(merged\_data, social\_data, left\_index=True, right\_index=True, how='left')

print("数据获取完成!")

print(f"数据集形状: {merged\_data.shape}")

print("包含特征:", merged\_data.columns.tolist())

return merged\_data

# 获取数据

full\_data = fetch\_extended\_data()

**3. 高级特征工程**

def advanced\_feature\_engineering(df):

"""

高级特征工程

"""

print("="\*50)

print("进行高级特征工程...")

# 1. 技术指标

for prefix in ['CSI300', 'SP500', 'STOXX50']:

close\_col = f'{prefix}\_Close'

if close\_col in df.columns:

# 移动平均线

df[f'{prefix}\_MA\_50'] = df[close\_col].rolling(window=50).mean()

df[f'{prefix}\_MA\_200'] = df[close\_col].rolling(window=200).mean()

# RSI相对强弱指数

delta = df[close\_col].diff()

gain = delta.where(delta > 0, 0)

loss = -delta.where(delta < 0, 0)

avg\_gain = gain.rolling(window=14).mean()

avg\_loss = loss.rolling(window=14).mean()

rs = avg\_gain / avg\_loss

df[f'{prefix}\_RSI'] = 100 - (100 / (1 + rs))

# 2. 宏观经济变化率

macro\_cols = ['USDCNY', 'USDEUR', 'USCPI', 'FEDFUNDS', 'GPRC']

for col in macro\_cols:

if col in df.columns:

df[f'{col}\_Change'] = df[col].pct\_change()

# 3. 事件影响指标

if 'Event\_Intensity' in df.columns:

# 事件影响滞后效应

for lag in [1, 3, 7]:

df[f'Event\_Impact\_Lag{lag}'] = df['Event\_Intensity'].shift(lag)

# 4. 社交媒体情感趋势

if 'Sentiment' in df.columns:

# 情感移动平均

df['Sentiment\_MA7'] = df['Sentiment'].rolling(window=7).mean()

df['Sentiment\_MA30'] = df['Sentiment'].rolling(window=30).mean()

# 情感波动率

df['Sentiment\_Volatility'] = df['Sentiment'].rolling(window=14).std()

# 5. 市场相关性特征

for i, prefix1 in enumerate(['CSI300', 'SP500', 'STOXX50']):

for prefix2 in ['CSI300', 'SP500', 'STOXX50'][i+1:]:

col1 = f'{prefix1}\_Close'

col2 = f'{prefix2}\_Close'

if col1 in df.columns and col2 in df.columns:

# 滚动相关性

df[f'{prefix1}\_{prefix2}\_Corr'] = df[col1].rolling(window=30).corr(df[col2])

# 处理缺失值

df = df.ffill().bfill()

print("高级特征工程完成!")

print(f"新增特征数: {len(df.columns) - len(df.columns)}")

return df

# 应用特征工程

enhanced\_data = advanced\_feature\_engineering(full\_data)

**4. LSTM-Transformer混合模型**

class LSTMTransformer(tf.keras.Model):

"""LSTM-Transformer混合模型"""

def \_\_init\_\_(self, num\_features, num\_timesteps=30, d\_model=64, num\_heads=4, ff\_dim=128, dropout=0.1):

super(LSTMTransformer, self).\_\_init\_\_()

# LSTM部分

self.lstm = LSTM(128, return\_sequences=True, input\_shape=(num\_timesteps, num\_features))

self.lstm\_dropout = Dropout(dropout)

# Transformer部分

self.attention = MultiHeadAttention(num\_heads=num\_heads, key\_dim=d\_model)

self.dropout1 = Dropout(dropout)

self.layernorm1 = LayerNormalization(epsilon=1e-6)

self.ffn = Sequential([

Dense(ff\_dim, activation='relu'),

Dense(d\_model)

])

self.dropout2 = Dropout(dropout)

self.layernorm2 = LayerNormalization(epsilon=1e-6)

# 输出层

self.global\_avg\_pool = tf.keras.layers.GlobalAveragePooling1D()

self.output\_layer = Dense(1)

def call(self, inputs):

# LSTM处理

x = self.lstm(inputs)

x = self.lstm\_dropout(x)

# Transformer处理

attn\_output = self.attention(x, x)

attn\_output = self.dropout1(attn\_output)

out1 = self.layernorm1(x + attn\_output)

ffn\_output = self.ffn(out1)

ffn\_output = self.dropout2(ffn\_output)

x = self.layernorm2(out1 + ffn\_output)

# 输出处理

x = self.global\_avg\_pool(x)

return self.output\_layer(x)

def prepare\_data\_for\_lstm(data, target\_column='SP500\_Close', timesteps=30):

"""

准备LSTM输入数据

"""

# 选择特征和目标

features = data.drop(columns=[target\_column] if target\_column in data.columns else [])

target = data[target\_column] if target\_column in data.columns else None

# 创建时间序列样本

X, y = [], []

for i in range(timesteps, len(features)):

X.append(features.iloc[i-timesteps:i].values)

if target is not None:

y.append(target.iloc[i])

X = np.array(X)

y = np.array(y) if target is not None else None

return X, y

def train\_lstm\_transformer(data, target\_index='SP500'):

"""

训练LSTM-Transformer混合模型

"""

print("="\*50)

print(f"训练LSTM-Transformer模型预测{target\_index}波动率...")

# 准备数据

target\_col = f'{target\_index}\_Volatility'

timesteps = 30

X, y = prepare\_data\_for\_lstm(data, target\_col, timesteps)

# 划分训练测试集

split = int(0.8 \* len(X))

X\_train, X\_test = X[:split], X[split:]

y\_train, y\_test = y[:split], y[split:]

# 创建模型

num\_features = X.shape[2]

model = LSTMTransformer(num\_features, timesteps)

model.compile(

optimizer=Adam(learning\_rate=0.001),

loss='mse',

metrics=['mae']

)

# 训练模型

early\_stop = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

history = model.fit(

X\_train, y\_train,

epochs=100,

batch\_size=32,

validation\_split=0.2,

callbacks=[early\_stop],

verbose=1

)

# 评估模型

test\_loss, test\_mae = model.evaluate(X\_test, y\_test, verbose=0)

print(f"测试集MAE: {test\_mae:.4f}")

# 预测

y\_pred = model.predict(X\_test).flatten()

# 可视化预测结果

plt.figure(figsize=(12, 6))

plt.plot(y\_test, label='真实值')

plt.plot(y\_pred, label='预测值', alpha=0.7)

plt.title(f'{target\_index}波动率预测')

plt.legend()

plt.savefig(f'{target\_index}\_volatility\_prediction.png')

plt.show()

return model, history, y\_test, y\_pred

# 训练模型

model\_sp500, history\_sp500, y\_test\_sp500, y\_pred\_sp500 = train\_lstm\_transformer(enhanced\_data, 'SP500')

**5. SHAP值分析**

def explain\_model\_with\_shap(model, data, sample\_size=100):

"""

使用SHAP解释模型

"""

print("="\*50)

print("使用SHAP分析特征重要性...")

# 准备解释数据

timesteps = 30

X, \_ = prepare\_data\_for\_lstm(data, timesteps=timesteps)

# 采样用于解释

background = X[np.random.choice(X.shape[0], min(sample\_size, X.shape[0]), replace=False)]

# 创建SHAP解释器

explainer = shap.DeepExplainer(model, background)

# 计算SHAP值

shap\_values = explainer.shap\_values(background)

# 特征名称（考虑时间步）

feature\_names = data.columns.tolist()

extended\_names = []

for i in range(timesteps, 0, -1):

extended\_names += [f"{name}\_t-{i}" for name in feature\_names]

# 可视化

shap.summary\_plot(

shap\_values[0].reshape(-1, len(feature\_names)\*timesteps)[0],

background.reshape(-1, len(feature\_names)\*timesteps)[0],

feature\_names=extended\_names,

max\_display=20,

show=False

)

plt.title('SHAP特征重要性')

plt.tight\_layout()

plt.savefig('shap\_feature\_importance.png')

plt.show()

# 返回平均SHAP值

mean\_shap = np.abs(shap\_values[0]).mean(0).mean(0)

shap\_df = pd.DataFrame({

'Feature': feature\_names,

'SHAP\_Value': mean\_shap

}).sort\_values('SHAP\_Value', ascending=False)

print("Top 10重要特征:")

print(shap\_df.head(10))

return shap\_df

# 解释SP500模型

shap\_results = explain\_model\_with\_shap(model\_sp500, enhanced\_data)

**6. 创建Streamlit仪表盘**

创建新的Python文件 **dashboard.py**:

# dashboard.py

import streamlit as st

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

# 设置页面

st.set\_page\_config(page\_title="全球股票市场分析", layout="wide")

st.title("全球股票市场多维度分析仪表盘")

# 侧边栏控制

st.sidebar.header("控制面板")

selected\_market = st.sidebar.selectbox("选择市场", ["沪深300", "标普500", "STOXX50"])

time\_range = st.sidebar.slider("选择时间范围", 2020, 2023, (2020, 2023))

show\_shap = st.sidebar.checkbox("显示SHAP分析", value=True)

# 加载数据（修复：为所有市场生成数据）

@st.cache\_data

def load\_data():

date\_range = pd.date\_range('2020-01-01', '2023-12-31', freq='D')

data = pd.DataFrame(index=date\_range)

# 为所有市场生成模拟数据

# 沪深300数据

data['CSI300\_Close'] = np.random.normal(4000, 300, len(date\_range)).cumsum()

data['CSI300\_Volatility'] = np.abs(np.random.normal(0.015, 0.006, len(date\_range)))

# 标普500数据

data['SP500\_Close'] = np.random.normal(3000, 500, len(date\_range)).cumsum()

data['SP500\_Volatility'] = np.abs(np.random.normal(0.01, 0.005, len(date\_range)))

# STOXX50数据

data['STOXX50\_Close'] = np.random.normal(3500, 400, len(date\_range)).cumsum()

data['STOXX50\_Volatility'] = np.abs(np.random.normal(0.012, 0.004, len(date\_range)))

# 通用数据

data['Event\_Intensity'] = np.random.uniform(0, 1, len(date\_range))

data['Sentiment'] = np.random.uniform(-1, 1, len(date\_range))

return data

data = load\_data()

# 过滤时间范围

start\_date = f"{time\_range[0]}-01-01"

end\_date = f"{time\_range[1]}-12-31"

filtered\_data = data.loc[start\_date:end\_date]

# 市场选择

market\_prefix = {

"沪深300": "CSI300",

"标普500": "SP500",

"STOXX50": "STOXX50"

}[selected\_market]

# 主仪表盘

col1, col2 = st.columns(2)

with col1:

st.subheader(f"{selected\_market}价格走势")

fig, ax = plt.subplots(figsize=(10, 4))

ax.plot(filtered\_data.index, filtered\_data[f"{market\_prefix}\_Close"])

ax.set\_xlabel("日期")

ax.set\_ylabel("价格")

st.pyplot(fig)

st.subheader("市场波动率")

fig, ax = plt.subplots(figsize=(10, 4))

ax.plot(filtered\_data.index, filtered\_data[f"{market\_prefix}\_Volatility"], color='red')

ax.set\_xlabel("日期")

ax.set\_ylabel("波动率")

st.pyplot(fig)

with col2:

st.subheader("事件强度与市场波动")

fig, ax = plt.subplots(figsize=(10, 4))

ax.bar(filtered\_data.index, filtered\_data['Event\_Intensity'], alpha=0.5, label='事件强度')

ax.plot(filtered\_data.index, filtered\_data[f"{market\_prefix}\_Volatility"],

color='red', label='波动率')

ax.set\_xlabel("日期")

ax.legend()

st.pyplot(fig)

st.subheader("社交媒体情感分析")

fig, ax = plt.subplots(figsize=(10, 4))

ax.plot(filtered\_data.index, filtered\_data['Sentiment'], color='green')

ax.axhline(0, color='gray', linestyle='--')

ax.set\_xlabel("日期")

ax.set\_ylabel("情感得分")

st.pyplot(fig)

# SHAP分析

if show\_shap:

st.subheader("特征重要性分析 (SHAP)")

# 模拟SHAP数据

features = ['利率变化', '汇率波动', '事件强度', '市场情绪', 'GDP增长']

shap\_values = np.abs(np.random.normal(0, 1, len(features)))

fig, ax = plt.subplots(figsize=(10, 6))

ax.barh(features, shap\_values, color='skyblue')

ax.set\_xlabel("SHAP值（平均绝对影响）")

st.pyplot(fig)

# 3D因子影响曲面

st.subheader("3D因子影响分析")

col\_x = st.selectbox("选择X轴因子", ['利率', '汇率', '事件强度'])

col\_y = st.selectbox("选择Y轴因子", ['市场情绪', 'GDP增长', '通胀率'])

col\_z = st.selectbox("选择Z轴指标", ['波动率', '收益率', '交易量'])

# 生成模拟数据

x = np.linspace(0, 10, 100)

y = np.linspace(0, 10, 100)

X, Y = np.meshgrid(x, y)

Z = np.sin(X) + np.cos(Y)

# 创建3D图

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.8)

ax.set\_xlabel(col\_x)

ax.set\_ylabel(col\_y)

ax.set\_zlabel(col\_z)

ax.set\_title(f"{col\_x}-{col\_y}对{col\_z}的影响曲面")

st.pyplot(fig)

# 运行仪表盘

if \_\_name\_\_ == "\_\_main\_\_":

st.write("仪表盘加载完成！")

**7. 3D因子影响曲面图**

def create\_3d\_impact\_surface(data, factor1, factor2, target):

"""

创建3D因子影响曲面图

"""

print("="\*50)

print(f"创建{factor1}-{factor2}对{target}的3D影响曲面...")

# 准备数据

x = data[factor1].values

y = data[factor2].values

z = data[target].values

# 创建网格

xi = np.linspace(min(x), max(x), 100)

yi = np.linspace(min(y), max(y), 100)

X, Y = np.meshgrid(xi, yi)

# 插值Z值

from scipy.interpolate import griddata

Z = griddata((x, y), z, (X, Y), method='cubic')

# 创建3D图

fig = plt.figure(figsize=(12, 8))

ax = fig.add\_subplot(111, projection='3d')

# 绘制曲面

surf = ax.plot\_surface(X, Y, Z, cmap='viridis',

rstride=1, cstride=1,

alpha=0.8, linewidth=0,

antialiased=True)

# 添加散点

ax.scatter(x, y, z, c='red', s=20, alpha=0.6)

# 设置标签

ax.set\_xlabel(factor1)

ax.set\_ylabel(factor2)

ax.set\_zlabel(target)

ax.set\_title(f"{factor1}-{factor2}对{target}的影响曲面")

# 添加颜色条

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.tight\_layout()

plt.savefig(f'3d\_impact\_{factor1}\_{factor2}\_{target}.png')

plt.show()

return fig

# 示例：创建利率和事件强度对波动率的影响曲面

create\_3d\_impact\_surface(enhanced\_data, 'FEDFUNDS', 'Event\_Intensity', 'SP500\_Volatility')

**第三步：执行工作流程**

在Jupyter Notebook中按顺序运行以下单元格：

1. **运行安装单元格**：安装所有必要的库（只需运行一次）
2. **运行数据获取单元格**：获取并预处理数据
3. **运行特征工程单元格**：创建高级特征
4. **运行LSTM-Transformer模型单元格**：训练预测模型
5. **运行SHAP分析单元格**：解释模型特征重要性
6. **运行3D曲面图单元格**：可视化因子影响

**第四步：运行仪表盘**

在终端中执行：

streamlit run dashboard.py