基于美国原油数据的时间序列分析

司徒婉儿 21068121

摘要：
本文旨在对美国原油价格时间序列进行分析和预测。首先，读入数据文件，对原数据进行描述性统计分析和趋势分析。然后，通过数据预处理得到三个不同的数据集：原始数据、按月累和聚合后的数据、滑动平均数据。接下来，对这三个数据集分别进行ADF和KPSS平稳性检验，并在一阶差分后确保平稳。通过绘制ACF和PACF图，识别适合的时间序列模型，并使用AIC准则进行模型定阶。最后，建立模型进行预测，并通过滚动预测评估模型的效果，对模型进行残差分析。结果显示，滚动预测模型能够较好地捕捉原数据特征，月累和聚合数据在模型拟合上表现较为优越。

数据预处理

本文使用的数据为1997-01-03到2010-09-24的世界原油和美国原油价格数据，每周取样，共717条数据。研究目标包括：时间序列描述性统计、数据预处理、模型建立及评估。

In [47]:
```
#导入所需要的库和函数
import datetime
import warnings
import itertools
import pmdarima as pm
import plotly.express as px
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, kpss, coint
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
```

In [57]:
```
#读入数据
dat = pd.read_csv('w-petroprice.txt', delim_whitespace=True,parse_dates={'Date':
dat.set_index('Date', inplace=True)
Dat = dat.reset_index()
```

In [8]:
```
#重构数据
dat_monthly = dat.resample('M').sum()
rollmean = dat.rolling(window=12).mean()
rollmean.dropna(inplace=True)
```

平稳性检验

ADF单位根检验：用于确定时间序列数据是否具有单位根。拒绝单位根假设表示序列平稳。
KPSS检验：用于评估时间序列数据的平稳性。原假设为序列平稳，拒绝原假设表示序列非平稳。

In [12]:
```
# ADF单位根检验
def adf_test(series, title=''):
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series, autolag='AIC')
    labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observat
    out = pd.Series(result[0:4], index=labels)
    for key, value in result[4].items():
        out[f'Critical Value ({key})'] = value
    print(out.to_string())
    if result[1] <= 0.05:
        print("=> Strong evidence against the null hypothesis, reject the null h
    else:
        print("=> Weak evidence against the null hypothesis, time series has a u
# KPSS检验
def kpss_test(series, title=''):
    print(f'KPSS Test: {title}')
    with warnings.catch_warnings():
        warnings.simplefilter('ignore')
        result = kpss(series, regression='c', nlags='auto')
    labels = ['KPSS Test Statistic', 'p-value', 'Lags Used']
    out = pd.Series(result[0:3], index=labels)
    for key, value in result[3].items():
        out[f'Critical Value ({key})'] = value
    print(out.to_string())
    if result[1] >= 0.05:
        print("=> Weak evidence against the null hypothesis, fail to reject the
    else:
        print("=> Strong evidence against the null hypothesis, reject the null h
```

In [13]:
```
# 对WUS石油价格进行平稳性检测
adf_test(dat['US'], title='US Petro Price')
kpss_test(dat['US'], title='US Petro Price')
```

```
Augmented Dickey-Fuller Test: US Petro Price
ADF Test Statistic              -2.126284
p-value                          0.234078
#Lags Used                      14.000000
Number of Observations Used    702.000000
Critical Value (1%)             -3.439700
Critical Value (5%)             -2.865666
Critical Value (10%)            -2.568967
=> Weak evidence against the null hypothesis, time series has a unit root, indica
ting it is non-stationary
KPSS Test: US Petro Price
KPSS Test Statistic       3.156889
p-value                   0.010000
Lags Used                17.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
=> Strong evidence against the null hypothesis, reject the null hypothesis. Data
is not stationary
```

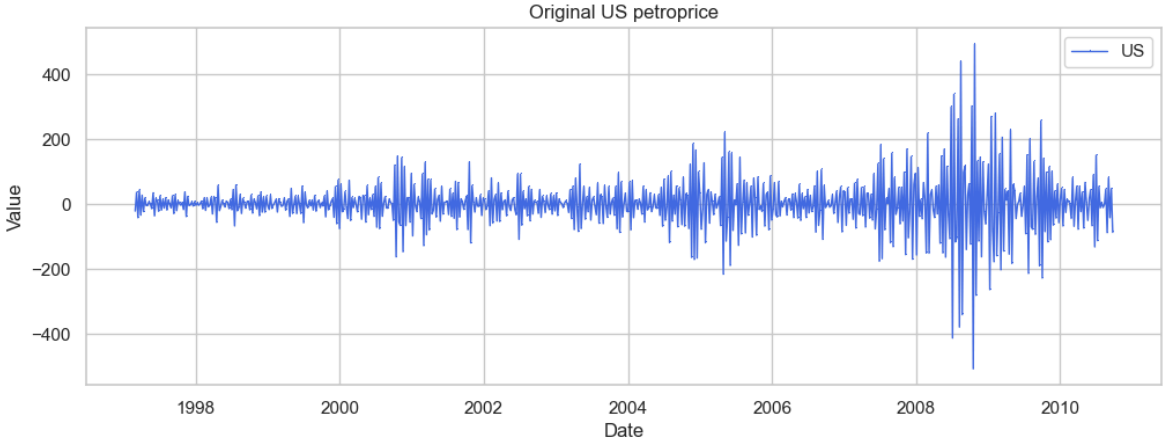
经过平稳性检验发现，目前的三个数据都不符合平稳性要求，因此对其进行差分转平。

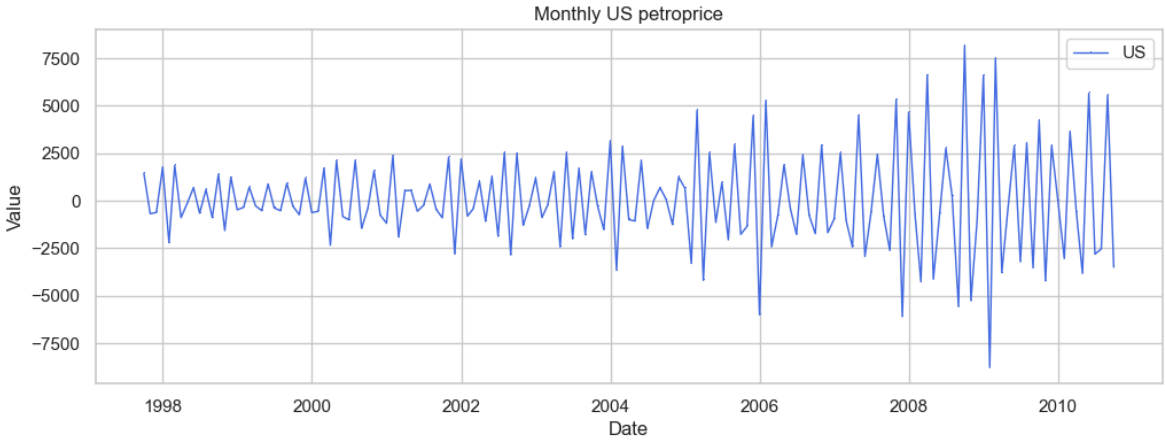
In [15]:
```
#数据整合与备份
dat1 = dat.copy()
dat2 = dat_monthly.copy()
dat3 = rollmean.copy()
```

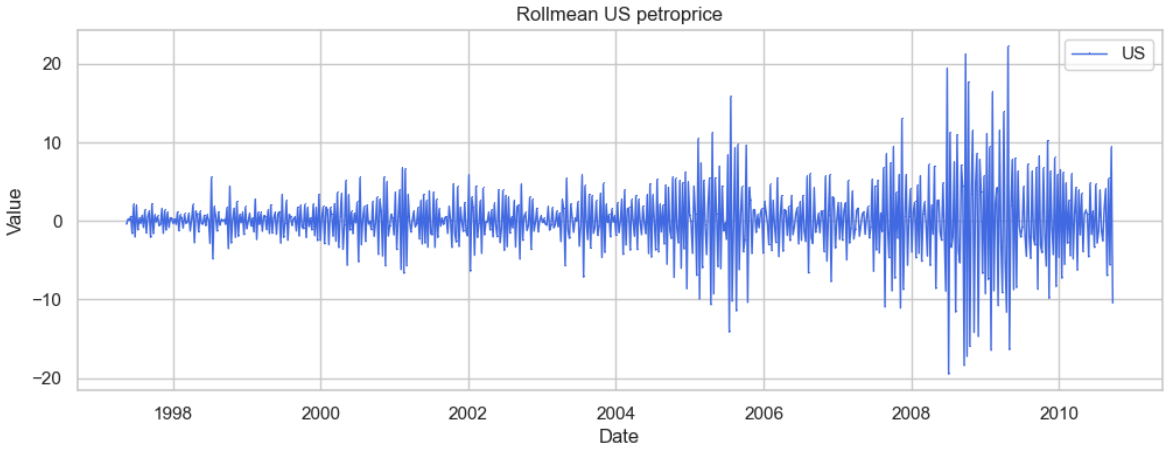
In [86]:
```
#定义差分
def difference(series, n=1):
    return series.diff(n).dropna()
#一阶差分
dat1 = difference(dat1, n=1)
dat2= difference(dat2, n=1)
dat3 = difference(dat3, n=1)
dat1.dropna(inplace=True)
dat2.dropna(inplace=True)
dat3.dropna(inplace=True)
#一阶差分时间序列
plt.figure(figsize=(12,4), dpi=100)
plt.plot(dat1.index, dat1['US'], color='royalblue', marker=',', linestyle='-', l
plt.title('Original US petroprice')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.figure(figsize=(12,4), dpi=100)
plt.plot(dat2.index, dat2['US'], color='royalblue', marker=',', linestyle='-', l
plt.title('Monthly US petroprice')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.figure(figsize=(12,4), dpi=100)
plt.plot(dat3.index, dat3['US'],color='royalblue', marker=',', linestyle='-', li
plt.title('Rollmean US petroprice')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
```

```
plt.grid(True)
#再检验
adf_test(dat1['US'], title='US Petro Price')
kpss_test(dat1['US'], title='US Petro Price')
```

```
Augmented Dickey-Fuller Test: US Petro Price
ADF Test Statistic             -22.088136
p-value                          0.000000
#Lags Used                      20.000000
Number of Observations Used    688.000000
Critical Value (1%)             -3.439891
Critical Value (5%)             -2.865750
Critical Value (10%)            -2.569012
=> Strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root and is stationary
KPSS Test: US Petro Price
KPSS Test Statistic       0.036886
p-value                   0.100000
Lags Used                38.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
=> Weak evidence against the null hypothesis, fail to reject the null hypothesis. Data is stationary
```





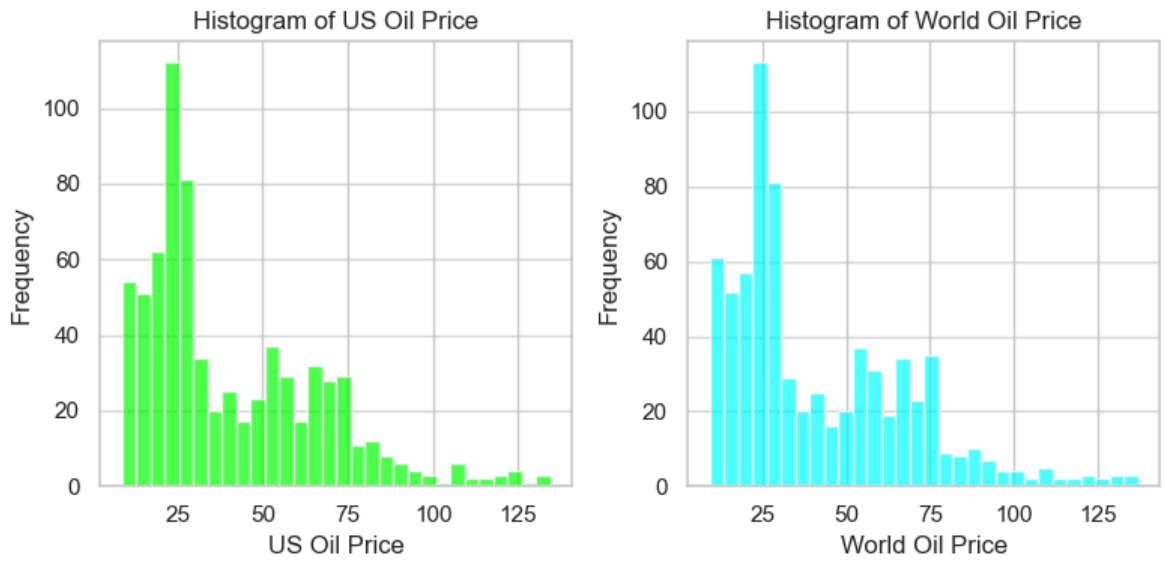


经检验，一阶差分后的三个备选数据都通过了平稳性检验。绘制其时间序列图像，可以观察到，一阶差分后，按月聚合数据与原始数据形状相似但取值大小不同，且前者比后者线条趋势干净。

数据描述性统计

In [133…

```
#数据直方图
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)  # 1行2列的第一个
plt.hist(dat['US'], bins=30, color='lime', alpha=0.7)
plt.title('Histogram of US Oil Price')
plt.xlabel('US Oil Price')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.subplot(1, 2, 2)  # 1行2列的第二个
plt.hist(dat['World'], bins=30, color='cyan', alpha=0.7)
plt.title('Histogram of World Oil Price')
plt.xlabel('World Oil Price')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.tight_layout()  # 调整子图间距
plt.show()
```



In [5]:

```
#数据包含十分相似的两列
```

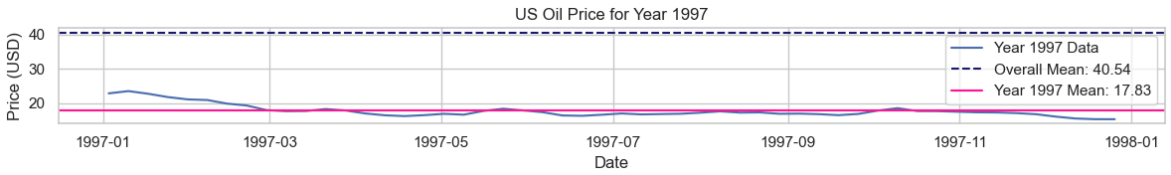
```
dat.head()
```

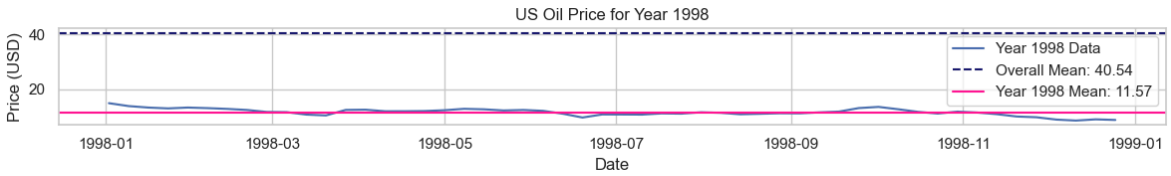
Out[5]:

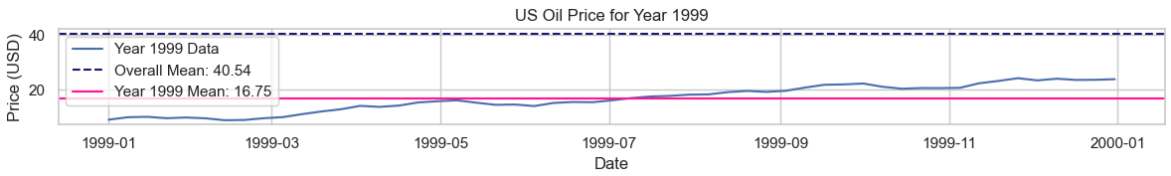
Date	World	US
1997-01-03	23.18	22.90
1997-01-10	23.84	23.56
1997-01-17	22.99	22.79
1997-01-24	22.05	21.83
1997-01-31	21.87	21.16

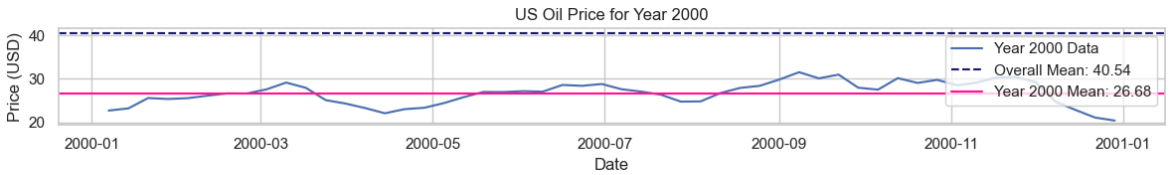
In [91]:

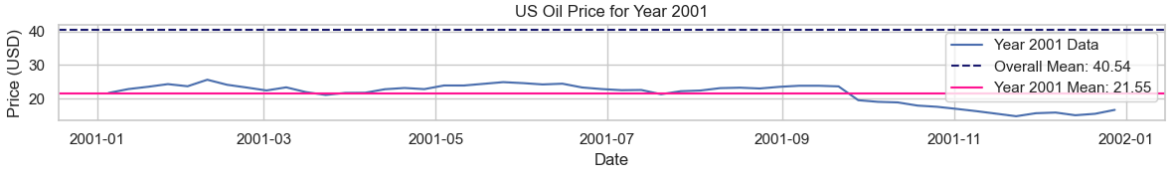
```
unique_years = dat.index.year.unique()
overall_mean = dat['US'].mean()
for i, year in enumerate(unique_years):
    plt.figure(figsize=(12, 2))  # 设置图形大小
    us_year_data = dat.loc[dat.index.year == year, 'US']
    plt.plot(us_year_data.index, us_year_data, label=f'Year {year} Data')
    plt.axhline(overall_mean, color='midnightblue', linestyle='--', label=f'Over
    year_mean = us_year_data.mean()
    plt.axhline(year_mean, color='deeppink', linestyle='-', label=f'Year {year}
    plt.title(f'US Oil Price for Year {year}')
    plt.xlabel('Date')
    plt.ylabel('Price (USD)')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

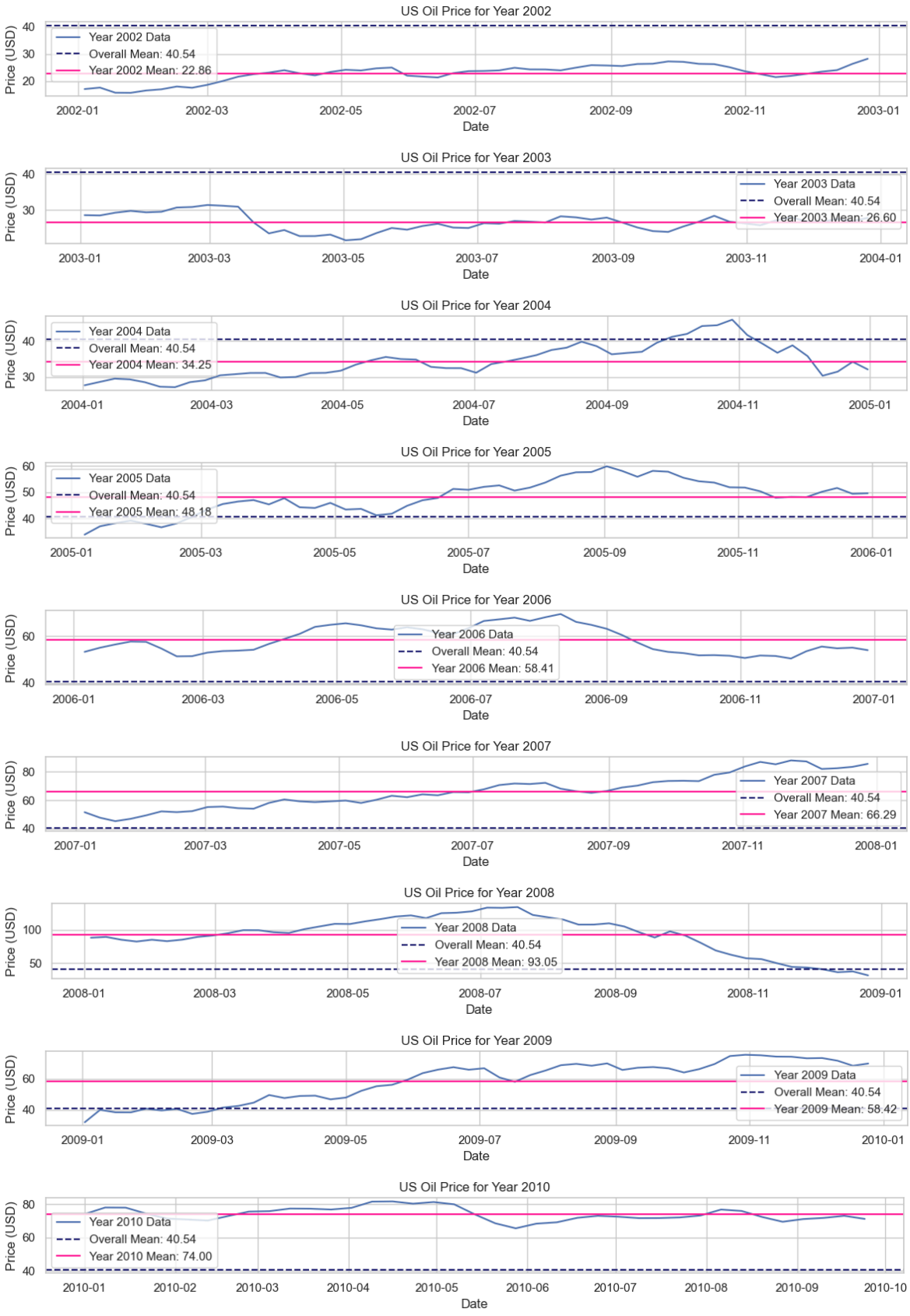












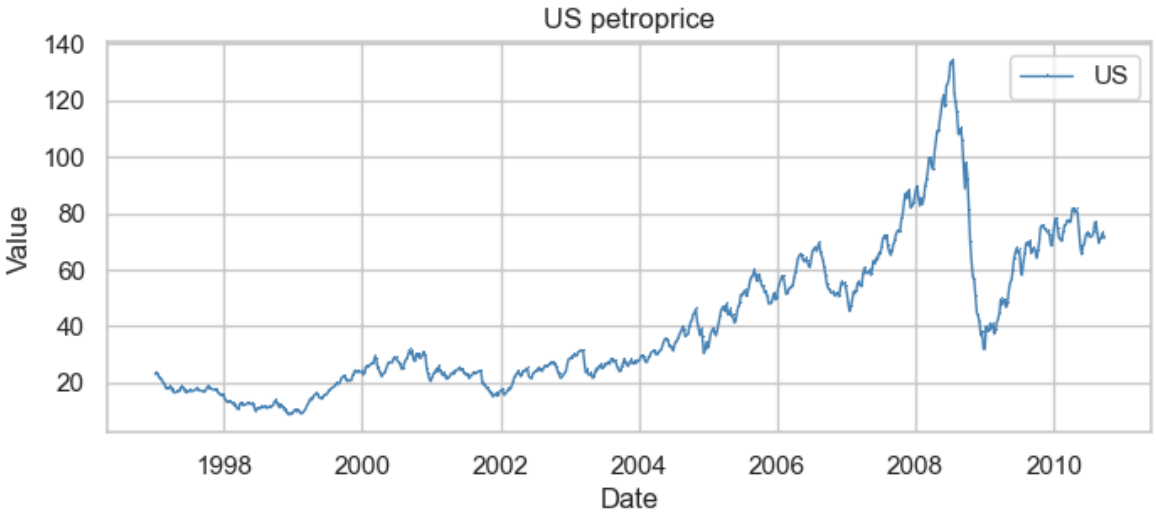
以年为时间段查看数据趋势，价格总均值为74。

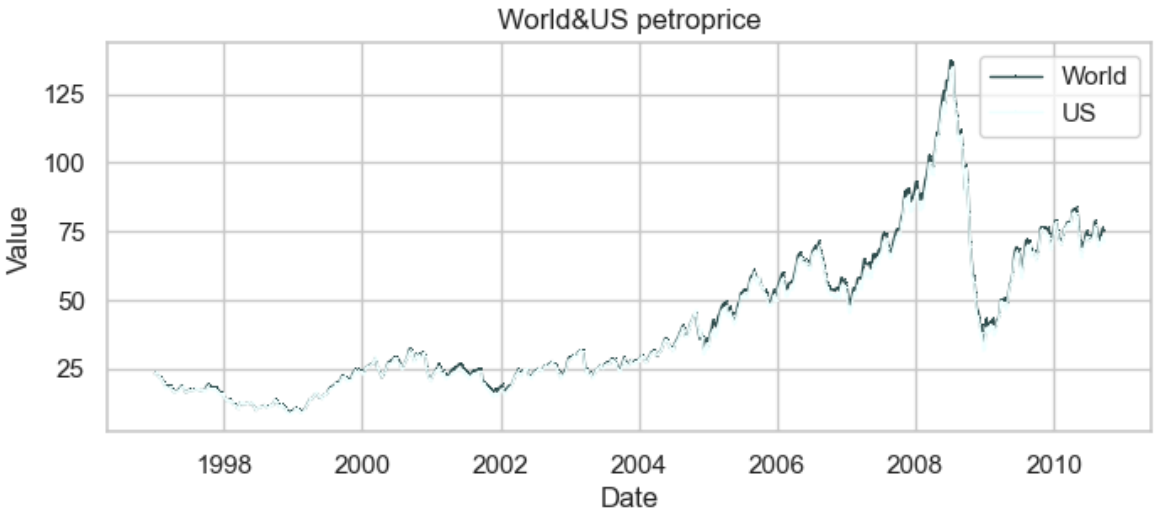
对比总均值和年均值。1997、1998年价格在偶数月份高于奇数月份，在年末下降；1999年价格有上升趋势，从7月份开始高于年均值；2000年大概在四个季度的中间时间价格高于年均值，过渡时间较低；2001年前9个月价格较高，之后下降至低谷；2002年重新上升；2003年5月达到低谷。

2004年趋势性上升，并第一次超过总均值；2005年年均值第一次超过总均值，大致可分为1-3月份一个阶段，3-6月份一个阶段，7-12月份一个阶段，逐次上升；2006年年均值

超过总均值更为明显，年价格波动性大；2007年价格保持上涨；2008年7、8月达到峰值后以较快趋势下降至低于总平均线的程度；2009年总体保持略微上涨；2010年价格相较平稳在75左右。

In [99]:
```
#原数据时间序列图
plt.figure(figsize=(8,3), dpi=100)
plt.plot(dat.index, dat['US'], color='steelblue', marker=',', linestyle='-', lin
plt.title('US petroprice')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.figure(figsize=(8,3), dpi=100)
plt.plot(dat.index, dat['World'], color='darkslategray', marker=',', linestyle='
plt.plot(dat.index, dat['US'], color='azure', marker=',', linestyle='-', linewid
plt.title('World&US petroprice')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
```





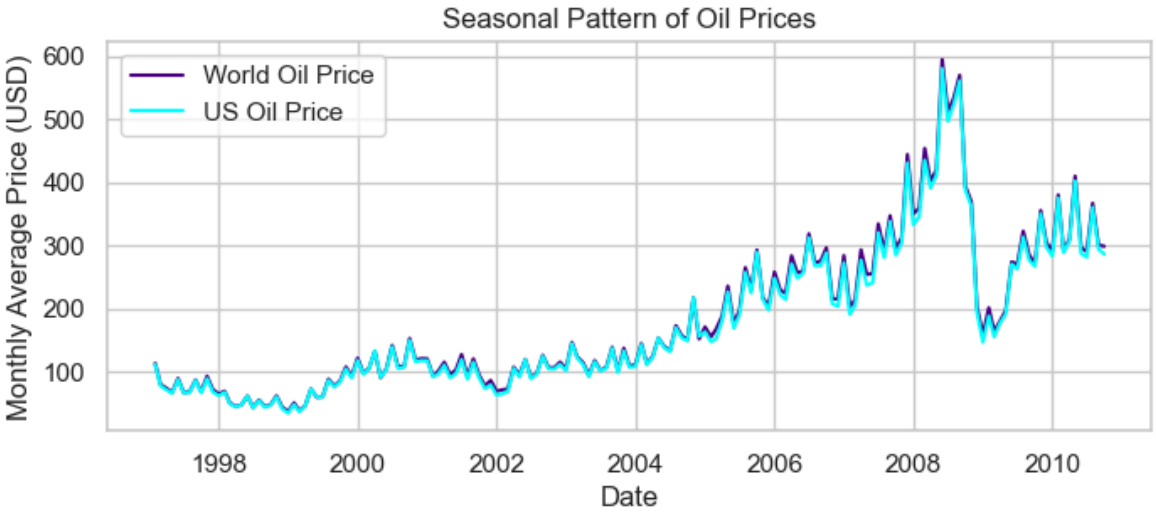
In [62]:
```
#相关性分析
correlation = dat['US'].corr(dat['World'])
print(f'Correlation coefficient: {correlation}')
```

```
Correlation coefficient: 0.9995631692105087
```

可以看出，美国原油价格和世界原油价格时间序列趋势相近，高度相关，似有重合。

In [67]:
```
#交互可放大时间序列图
warnings.filterwarnings('ignore')
fig = px.line(Dat, x='Date', y='US')
fig.show()
```

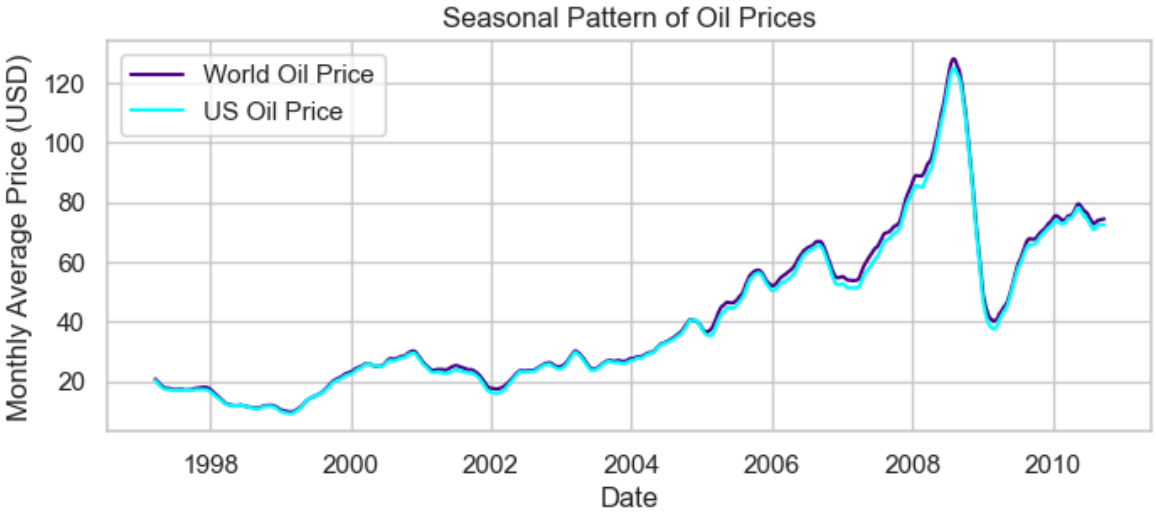
In [117…
```
#月聚合数据
plt.figure(figsize=(8,3))
plt.plot(dat_monthly['World'], label='World Oil Price', color='indigo')
plt.plot(dat_monthly['US'], label='US Oil Price', color='aqua')
plt.title('Seasonal Pattern of Oil Prices')
plt.xlabel('Date')
plt.ylabel('Monthly Average Price (USD)')
plt.legend()
plt.show()
```



上图为原始数据按月聚合后的数据。波动细节更少。不同聚合函数的选择代表的含义不同，如sum()为区间段累计值，可以反应每周或每月的原油价格总和，所得曲线具有一定局部规律性，mean()为时间段均值，综合考虑所有值包括异常值， median()则取中位数值，对异常值不敏感。
由于选取的是累和函数，所以会有一定的周期性峰值。

In [118…

```
#滑动平均
plt.figure(figsize=(8,3))
plt.plot(rollmean['World'], label='World Oil Price', color='indigo')
plt.plot(rollmean['US'], label='US Oil Price', color='aqua')
plt.title('Seasonal Pattern of Oil Prices')
plt.xlabel('Date')
plt.ylabel('Monthly Average Price (USD)')
plt.legend()
plt.show()
```



上图是滑动平均处理后的数据，可以看出数据更为平滑，减少细节，保留趋势。

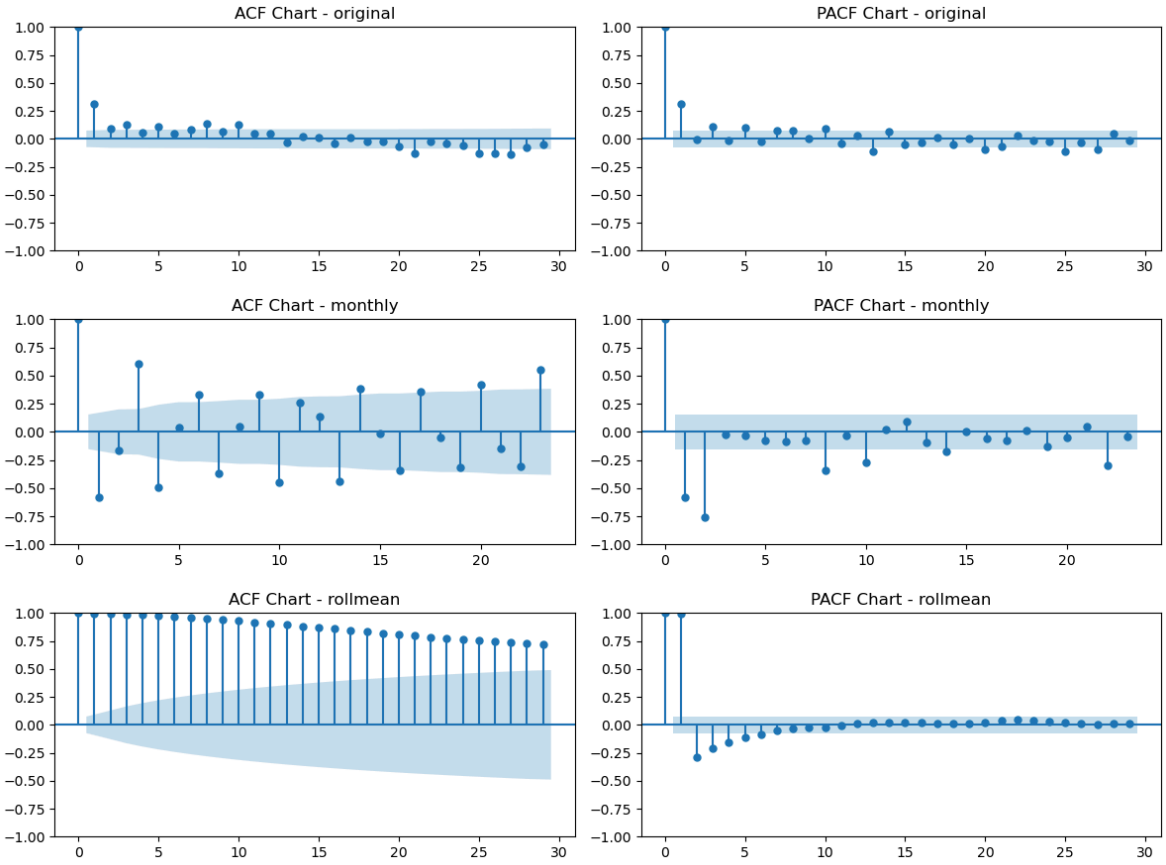
ACF和PACF

ACF（自相关函数）和PACF（偏自相关函数）是时间序列分析中用于识别数据相关性模式的工具。
ACF测量时间序列在不同时间滞后下的值之间的相关性，反映序列自身的相关结构。PACF则测量在控制了序列中其他时间滞后值的影响后，序列在特定滞后下的相关性。

两者通常通过绘制ACF图和PACF图来辅助确定ARIMA模型的参数，是构建时间序列预测模型的关键步骤。

In [173…
```
# 绘制ACF和PACF图
def plot_acf_pacf(series, title):
    plt.figure(figsize=(12,3))
    plt.subplot(121)
    plot_acf(series, ax=plt.gca())
    plt.title(f'ACF Chart - {title}')
    plt.subplot(122)
    plot_pacf(series, ax=plt.gca())
    plt.title(f'PACF Chart - {title}')
    plt.tight_layout()
    plt.show()
```

In [174…
```
plot_acf_pacf(dat1['US'],title='original')
plot_acf_pacf(dat2['US'],title='monthly')
plot_acf_pacf(dat3['US'],title='rollmean')
```



从上至下是dat1,dat2,dat3一阶差分之后的ACF图和PACF图。　　可以看到，原始数据一阶差分后的ACF快速下降，并在滞后2左右显著消失，这表明原始数据的自相关性在一阶差分后得到了较好地消除。PACF图显示了在滞后1处有一个明显的峰值，然后迅速下降到0。考虑使用ARIMA模型。

第二行是按月聚合后数据一阶差分的ACF和PACF，ACF图显示自相关性在滞后数更高的情况下逐渐衰减，表明数据中的季节性成分或周期性更为显著。PACF图在滞后1和滞后12（大致对应一年）处有显著峰值，这表明数据存在季节性和周期性的成分，可能需要季节性差分或使用SARIMA模型来捕捉这些特征。考虑使用SARIMA（季节性ARIMA）模型。

滑动平均后的一阶差分数据，ACF图显示出一种缓慢衰减的模式，这表明数据在进行滑动平均后，仍然存在较强的趋势成分。PACF图在滞后1处有显著峰值，之后逐渐衰减，这表

明滑动平均后数据的一阶差分仍保留了一些短期的自相关性。考虑使用SARIMA模型。
对每组数据的ACF和PACF图进行对比，发现原始数据一阶差分后的ACF和PACF图形态较为简单，表明数据经过一阶差分后，趋势成分和季节性成分被较好地消除。而按月聚合和滑动平均处理后的一阶差分数据，ACF和PACF图形态复杂，显示了更强的周期性和趋势性。
三组图像对比来看，原始数据一阶差分后的ACF和PACF图更接近于白噪声特征，说明其平稳性更好。按月均值聚合和滑动平均处理后的一阶差分数据则表现出较强的周期性，表明需要进一步处理以捕捉其季节性特征。

模型建立及预测

In [20]:
```
dat1 = dat.copy()
dat2 = dat_monthly.copy()
dat3 = rollmean.copy()
```

原数据

AIC准则定阶

AIC（赤池信息量准则）是一种用于模型选择的统计准则，通过平衡模型复杂度和拟合优度来评估模型的优劣。AIC值越小，表示模型在拟合数据的同时保持了较高的简洁性。

In [35]:
```
model1 = pm.auto_arima(dat1['US'], seasonal=False, trace=True,
                       error_action='ignore', suppress_warnings=True,
                       stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=2975.190, Time=0.65 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3056.474, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=2985.021, Time=0.16 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=2986.538, Time=0.17 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3055.254, Time=0.10 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=2978.097, Time=0.51 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=2984.008, Time=0.50 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=1.23 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=2976.947, Time=0.86 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=2986.974, Time=0.21 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=2977.293, Time=0.71 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=2980.246, Time=0.50 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=2978.884, Time=1.65 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=2973.382, Time=0.35 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=2976.292, Time=0.31 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=2982.435, Time=0.29 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=2975.099, Time=0.54 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=2975.138, Time=0.55 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=2985.383, Time=0.14 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=2975.482, Time=0.41 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=2978.596, Time=0.30 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=2977.075, Time=1.12 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 11.347 seconds
```

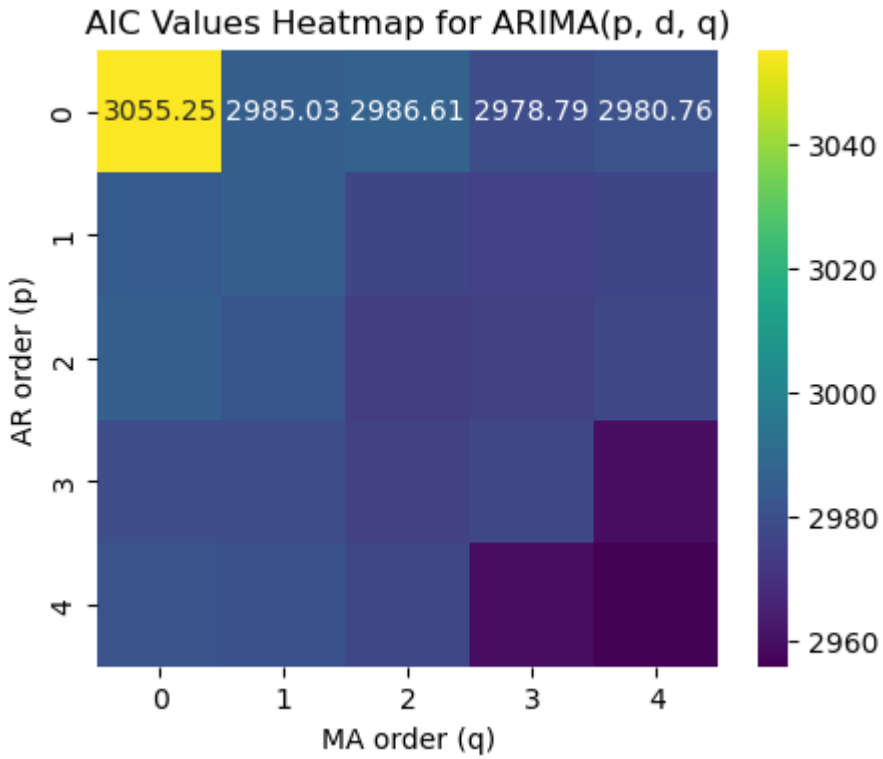
AIC准则阶数热力图

In [210…

```
#AIC热力图
def find_best_arima_model(data, p_max=4, q_max=4, d=1):
    aic_matrix = np.full((p_max + 1, q_max + 1), np.inf)
    best_aic = np.inf
    best_order = None
    for p in range(p_max + 1):
        for q in range(q_max + 1):
            try:
                model = ARIMA(data, order=(p, d, q)).fit()
                aic_matrix[p, q] = model.aic
                if aic_matrix[p, q] < best_aic:
                    best_aic = aic_matrix[p, q]
                    best_order = (p, d, q)
            except Exception as e:
                print(f"Error fitting ARIMA({p}, {d}, {q}): {e}")
    # 绘制AIC热力图
    plt.figure(figsize=(5,4))
    sns.heatmap(aic_matrix, annot=True, fmt=".2f", cmap="viridis",
                xticklabels=range(q_max + 1), yticklabels=range(p_max + 1))
    plt.title("AIC Values Heatmap for ARIMA(p, d, q)")
    plt.xlabel("MA order (q)")
    plt.ylabel("AR order (p)")
    plt.show()
    best_model = ARIMA(data, order=best_order).fit()
    return best_model, best_aic
```

In [211…

```
warnings.filterwarnings('ignore')
find_best_arima_model(dat1['US'])
```



Out[211…

```
(<statsmodels.tsa.arima.model.ARIMAResultsWrapper at 0x1be22ba3190>,
 2955.792388264508)
```

因为自动选阶函数和阶数热力图使用的是不同的建模函数来构造同一模型，因此AIC数值上可能会有一点区别，但是判断所选阶数是大体相同的。

模型预测

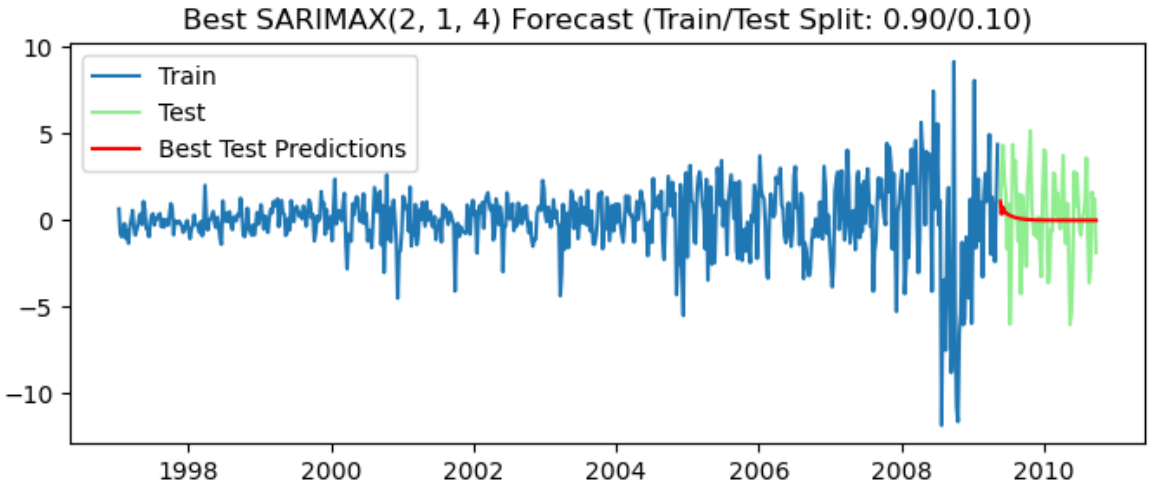
In [214…

```
#训练集/测试集预测
def fit_and_evaluate_sarimax(data, p, d, q, ratios):
    results = []
    n = len(data)
    best_result = None
    best_ratio = None
    for a in ratios:
        train_data = data['US'][0:int(n*a)]
        test_data = data['US'][int(n*a):]
        train_data=difference(train_data, n=d)
        test_data=difference(test_data, n=d)
        train_data.dropna(inplace=True)
        test_data.dropna(inplace=True)
        model = SARIMAX(train_data, order=(p, 0, q))
        model_fit = model.fit(disp=False)
        test_predictions = model_fit.forecast(steps=len(test_data))
        mse = mean_squared_error(test_data, test_predictions)
        rmse = np.sqrt(mse)
        mae = mean_absolute_error(test_data, test_predictions)
        results.append({
            'ratio': a,
            'mse': mse,
            'rmse': rmse,
            'mae': mae
        })
        if best_result is None or rmse < best_result['rmse']:
            best_result = {
                'ratio': a,
                'mse': mse,
                'rmse': rmse,
                'mae': mae,
                'model': model_fit,
                'predictions': test_predictions
            }
            best_ratio = a
    results_df = pd.DataFrame(results)
    print("Best Train/Test Split Ratio: {:.2f}/{:.2f}".format(best_ratio, 1-best
    print("MSE: {:.4f}, RMSE: {:.4f}, MAE: {:.4f}".format(best_result['mse'], be
    if best_result:
        plt.figure(figsize=(8,3))
        plt.plot(train_data.index, train_data, label='Train')
        plt.plot(test_data.index, test_data, label='Test',color='lightgreen')
        plt.plot(test_data.index, best_result['predictions'], label='Best Test P
        plt.legend()
        plt.title('Best SARIMAX{} Forecast (Train/Test Split: {:.2f}/{:.2f})'.fo
        plt.show()
    return results_df
```

In [252…

```
#测试集预测
def generate_series(start, end, step):
    return [start + i * step for i in range(int((end - start) / step) + 1)]
p,d,q=2,1,2
ratios = generate_series(0.75, 0.95, 0.05)
fit_and_evaluate_sarimax(dat1,p,d,q, ratios)
```

```
Best Train/Test Split Ratio: 0.90/0.10
MSE: 6.1715, RMSE: 2.4842, MAE: 1.9565
```



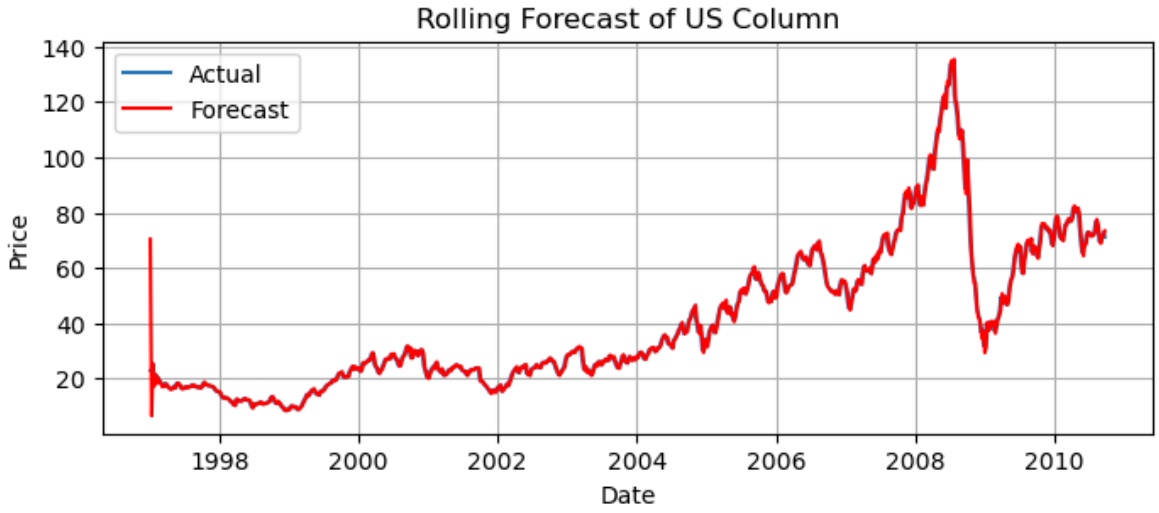
Out[252...

	ratio	mse	rmse	mae
0	0.75	11.687841	3.418748	2.586189
1	0.80	13.447885	3.667136	2.791080
2	0.85	12.503177	3.535983	2.623662
3	0.90	6.171482	2.484247	1.956478

In [225...

```
#dat1滚动预测
dat1 = dat.copy()
order = (2, 1, 2)
predictions1 = []
history = list(dat1['US'].values)
for t in range(len(dat1)):
    model = SARIMAX(history, order=order, seasonal_order=(0, 0, 0, 0))
    model_fit = model.fit(disp=False)
    yhat = model_fit.forecast()[0]
    predictions1.append(yhat)
    history.append(dat1['US'].iloc[t])

plt.figure(figsize=(8,3))
plt.plot(dat1.index, dat1['US'], label='Actual')
plt.plot(dat1.index, predictions1, color='red', label='Forecast')
plt.legend()
plt.title('Rolling Forecast of US Column')
plt.xlabel('Date')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



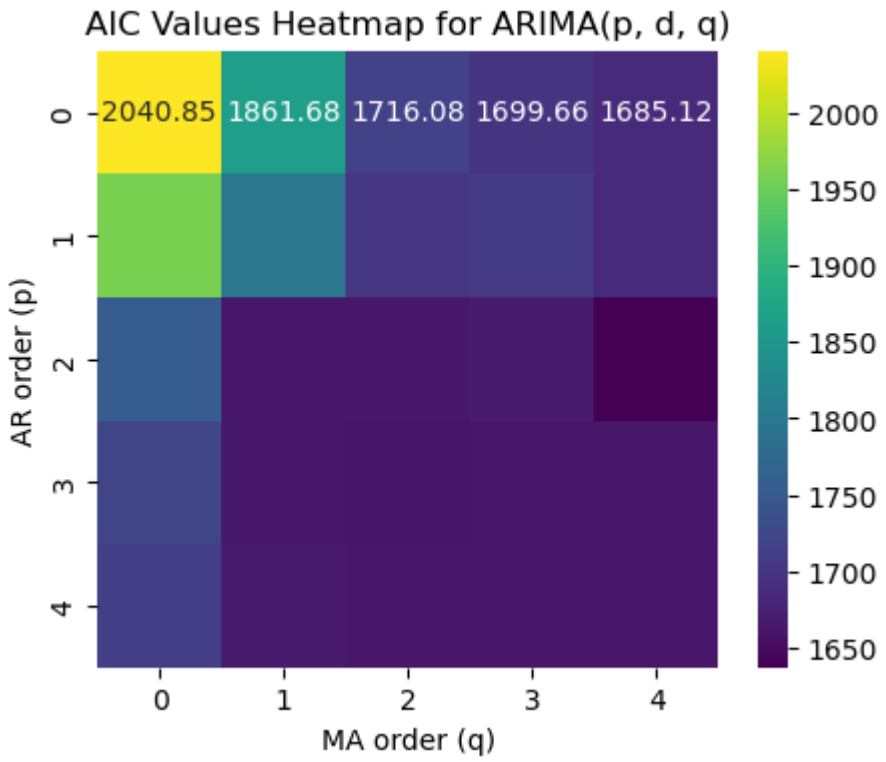
按月聚合数据

In [15]:
```
model2 = pm.auto_arima(dat2['US'], seasonal=False, trace=True,
                       error_action='ignore', suppress_warnings=True,
                       stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.80 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1716.914, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1699.706, Time=0.09 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1697.813, Time=0.12 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1715.007, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1698.651, Time=0.13 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=1684.689, Time=0.22 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1681.420, Time=0.43 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=1649.004, Time=0.47 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=1657.169, Time=0.22 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=1645.972, Time=0.53 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=1639.241, Time=0.84 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=1645.154, Time=0.32 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=1635.958, Time=0.98 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=1634.492, Time=0.90 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=1632.502, Time=0.64 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=1646.680, Time=0.42 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=1645.022, Time=0.22 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=1634.489, Time=0.93 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=1647.005, Time=0.25 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=1641.180, Time=0.74 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=1.12 sec
 ARIMA(4,1,1)(0,0,0)[0]             : AIC=1632.941, Time=0.39 sec

Best model:  ARIMA(4,1,1)(0,0,0)[0] intercept
Total fit time: 10.858 seconds
```

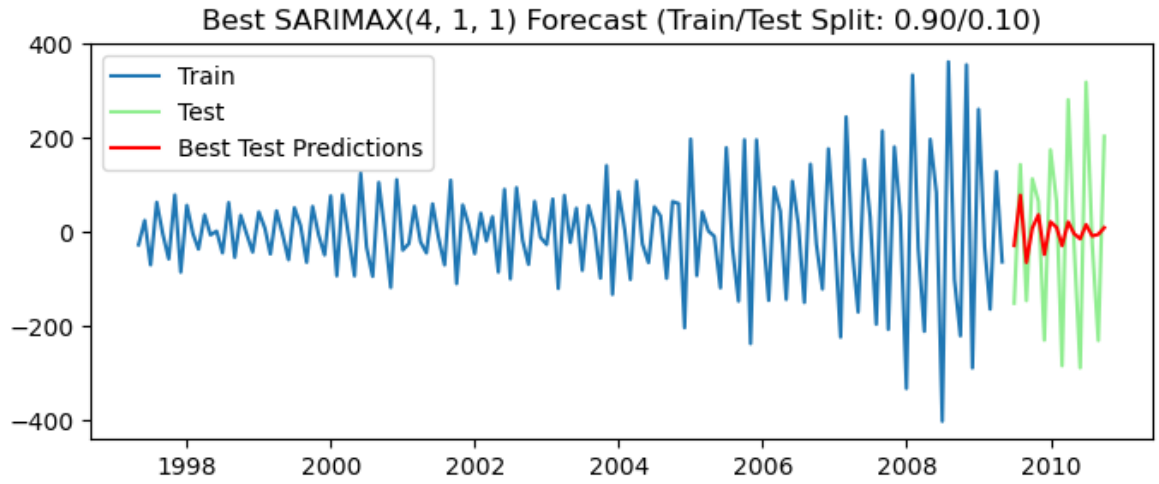
In [212…
```
find_best_arima_model(dat2['US'])
```



Out[212… (<statsmodels.tsa.arima.model.ARIMAResultsWrapper at 0x1be05395ed0>, 1637.0044033675292)

In [216…
```
#一阶差分数据预测
p,d,q=4,1,1
ratios = generate_series(0.75, 0.95, 0.05)
fit_and_evaluate_sarimax(dat2,p,d,q, ratios)
```

```
Best Train/Test Split Ratio: 0.90/0.10
MSE: 30381.4499, RMSE: 174.3028, MAE: 146.5878
```



Out[216…

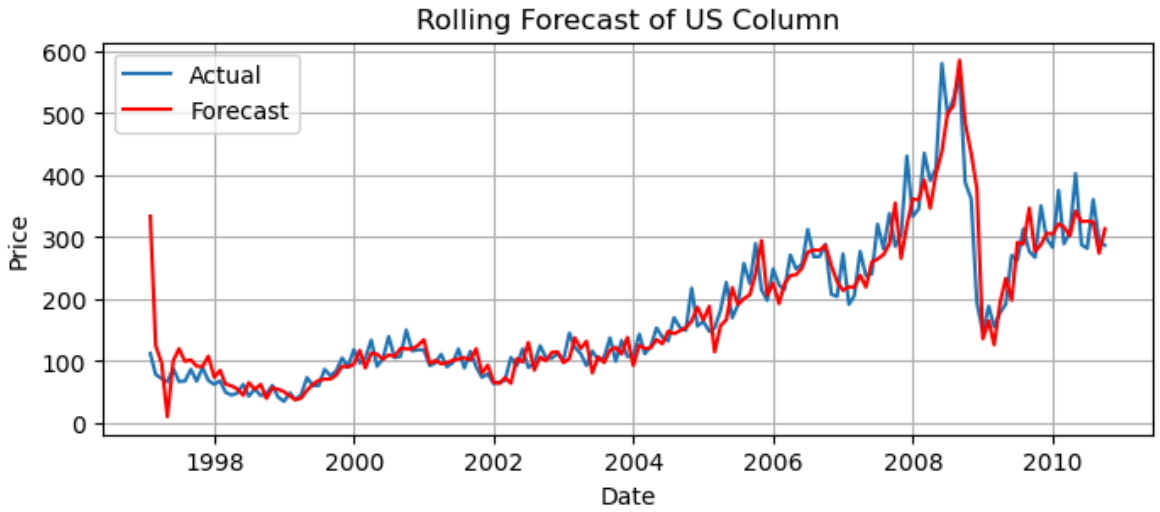
	ratio	mse	rmse	mae
0	0.75	49464.456661	222.406063	192.805352
1	0.80	46813.346342	216.363921	189.957379
2	0.85	46399.414261	215.405233	183.449251
3	0.90	30381.449865	174.302753	146.587763

In [224…

```
#按月聚合数据滚动预测
dat2 = dat_monthly.copy()
order = (4, 1, 2)

predictions2 = []
history = list(dat2['US'].values)
for t in range(len(dat2)):
    model = SARIMAX(history, order=order, seasonal_order=(0, 0, 0, 0))
    model_fit = model.fit(disp=False)
    yhat = model_fit.forecast()[0]
    predictions2.append(yhat)
    history.append(dat2['US'].iloc[t])

plt.figure(figsize=(8,3))
plt.plot(dat2.index, dat2['US'], label='Actual')
plt.plot(dat2.index, predictions2, color='red', label='Forecast')
plt.legend()
plt.title('Rolling Forecast of US Column')
plt.xlabel('Date')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



滑动平均数据

In [29]:

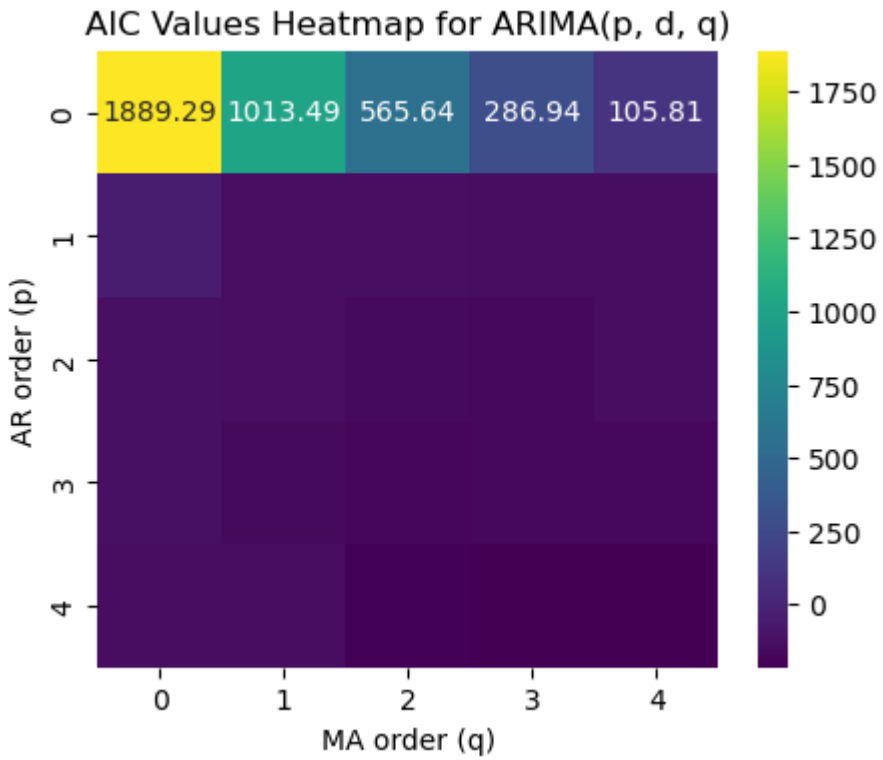
```
model3 = pm.auto_arima(dat3['US'], seasonal=False, trace=True,
                       error_action='ignore', suppress_warnings=True,
                       stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-159.169, Time=3.53 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1886.751, Time=0.25 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-41.566, Time=0.43 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1011.557, Time=0.57 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1889.286, Time=0.16 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-135.469, Time=0.63 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-136.417, Time=0.93 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=-167.409, Time=2.85 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=2.44 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=-193.117, Time=4.02 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=-145.427, Time=2.56 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=4.32 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=4.79 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=-163.669, Time=5.09 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=-164.906, Time=4.25 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=4.56 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=inf, Time=2.89 sec

Best model:  ARIMA(4,1,2)(0,0,0)[0] intercept
Total fit time: 44.322 seconds
```

In [213…

```
dat3 = rollmean.copy()
find_best_arima_model(dat3['US'])
```



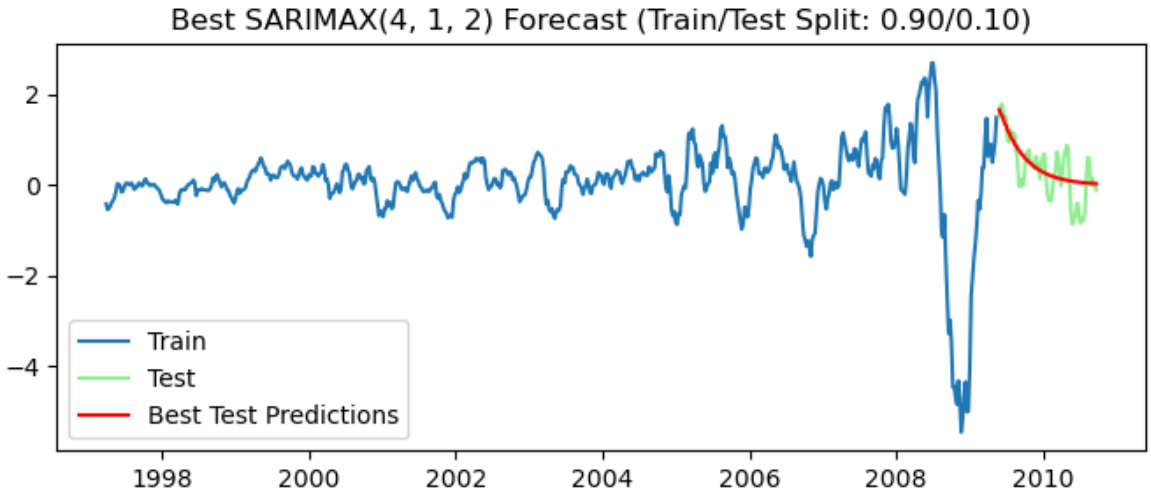
Out[213…

```
(<statsmodels.tsa.arima.model.ARIMAResultsWrapper at 0x1be053b7090>,
 -214.04777243121498)
```

In [217…

```
#一阶差分数据预测
dat3 = rollmean.copy()
p,d,q=4,1,2
ratios = generate_series(0.75, 0.95, 0.05)
fit_and_evaluate_sarimax(dat3,p,d,q, ratios)
```

```
Best Train/Test Split Ratio: 0.90/0.10
MSE: 0.1922, RMSE: 0.4384, MAE: 0.3427
```



Out[217…

	ratio	mse	rmse	mae
0	0.75	2.883109	1.697972	1.183596
1	0.80	3.350281	1.830377	1.277613
2	0.85	2.128270	1.458859	1.010377
3	0.90	0.192209	0.438417	0.342709

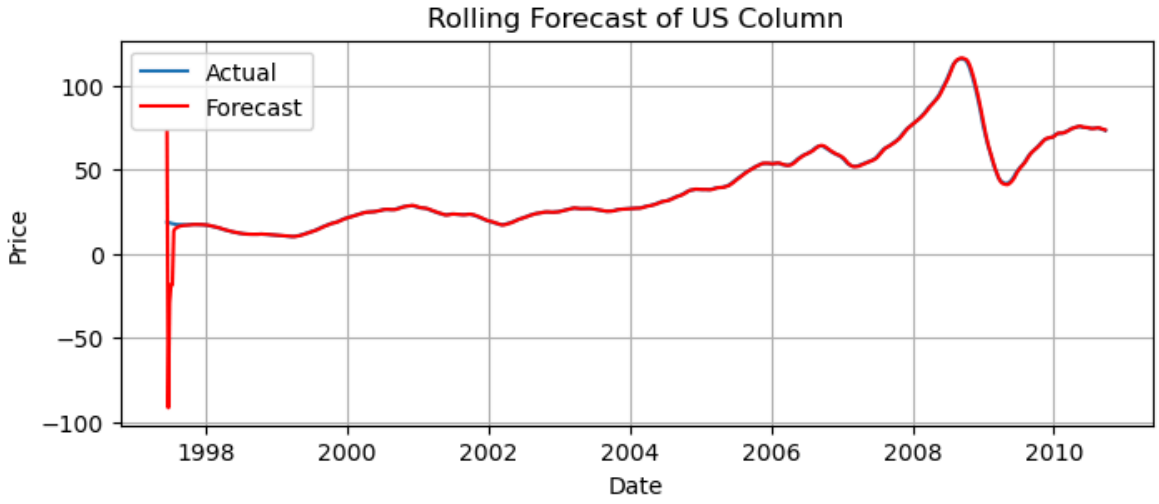
通过修改训练集和测试集的比例，可以看出后续预测波动、精度都有非常明显的不同。三个数据最优比例都是9：1.

In [223…

```
#滑动平均数据滚动预测
dat3 = rollmean.copy()
order = (4, 1, 1)

predictions3 = []
history = list(dat3['US'].values)
for t in range(len(dat3)):
    model = SARIMAX(history, order=order, seasonal_order=(0, 0, 0, 0))
    model_fit = model.fit(disp=False)
    yhat = model_fit.forecast()[0]
    predictions3.append(yhat)
    history.append(dat3['US'].iloc[t])

plt.figure(figsize=(8,3))
plt.plot(dat3.index, dat3['US'], label='Actual')
plt.plot(dat3.index, predictions3, color='red', label='Forecast')
plt.legend()
plt.title('Rolling Forecast of US Column')
plt.xlabel('Date')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



残差分析

残差分析是评估时间序列模型拟合优度的一种方法。它涉及计算实际观测值与模型预测值之间的差异（残差），并检查这些残差是否随机分布且无明显模式。残差分析包括绘制残差图、直方图、自相关图，以及进行Ljung-Box检验等，以确保残差表现为白噪声特性。这有助于确认模型假设是否得到满足，模型是否稳定，以及数据是否被正确地建模。

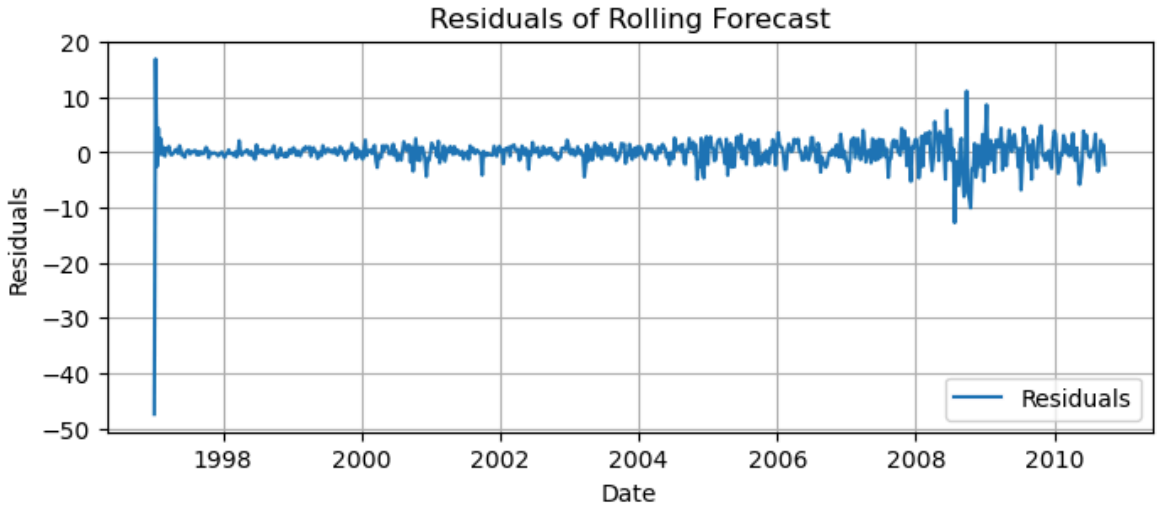
In [120…

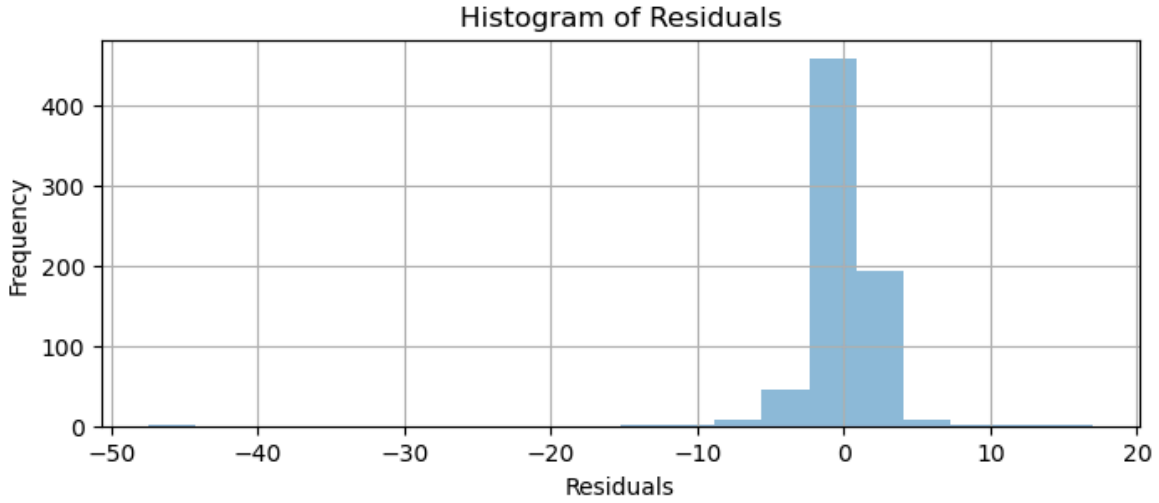
```python
# 残差图
def res(dat,pre):
    residuals = [actual - pred for actual, pred in zip(dat['US'], pre)]
    plt.figure(figsize=(8, 3))
    plt.plot(dat.index, residuals, label='Residuals')
    plt.legend()
    plt.title('Residuals of Rolling Forecast')
    plt.xlabel('Date')
    plt.ylabel('Residuals')
    plt.grid(True)
    plt.show()
    # 残差直方图
    plt.figure(figsize=(8, 3))
    plt.hist(residuals, bins=20, alpha=0.5)
    plt.title('Histogram of Residuals')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

原始数据

In [262…

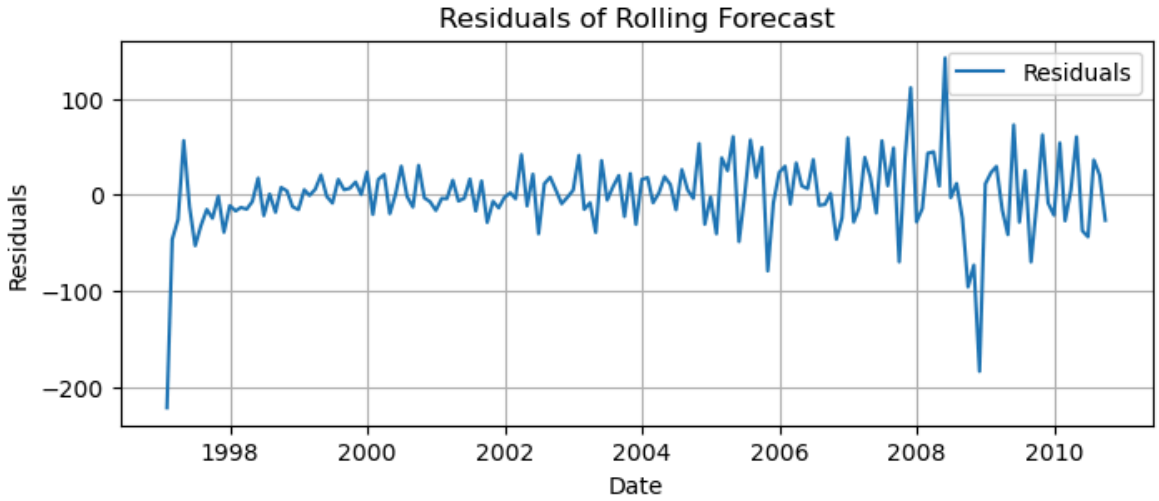
```python
res(dat1,predictions1)
```

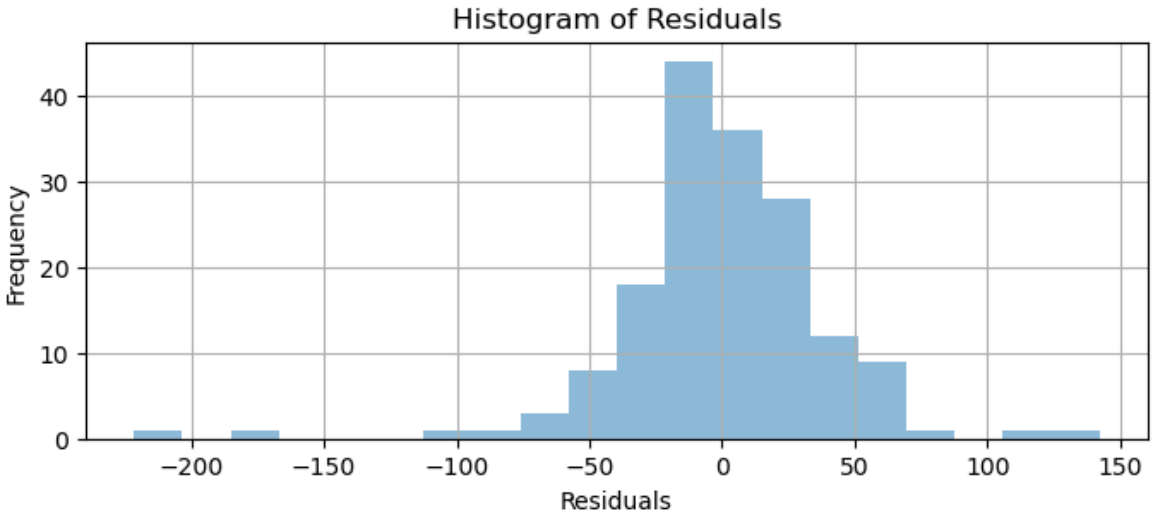




月聚合数据

In [263…
```
res(dat2,predictions2)
```

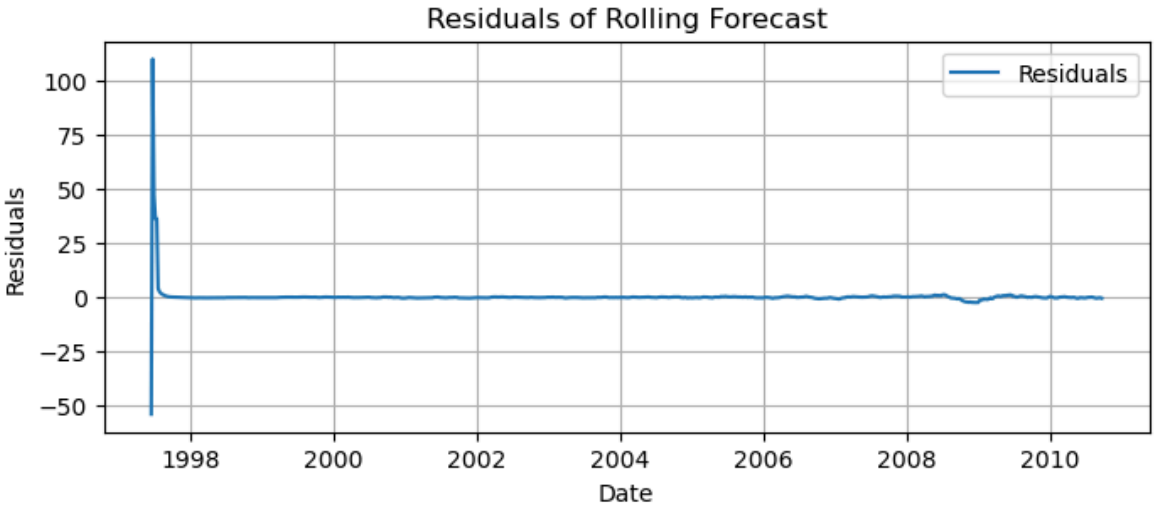


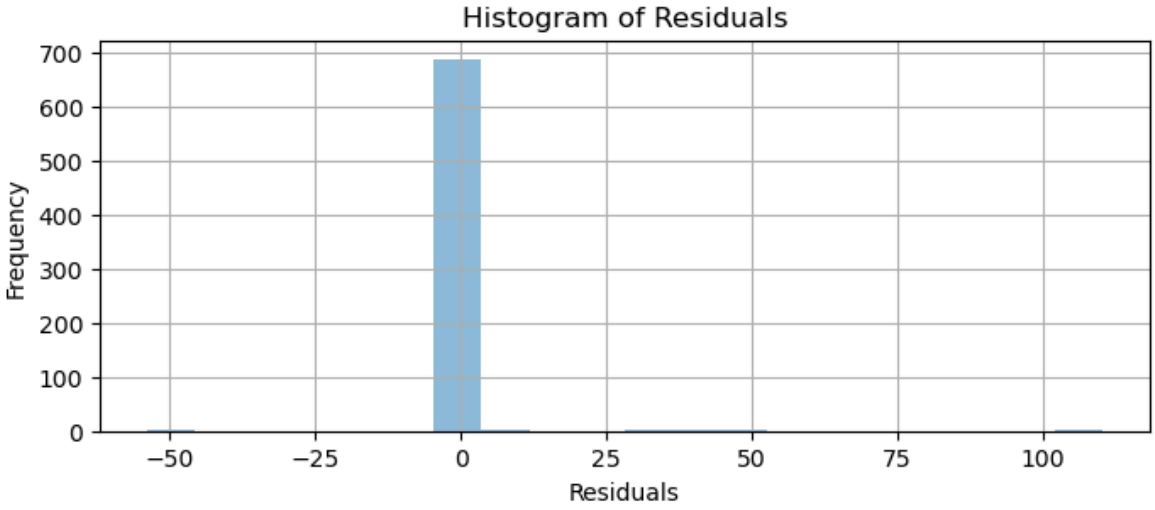


滑动平均数据

In [264…

```
res(dat3,predictions3)
```





直观来看，dat1的拟合模型残差基本符合要求，dat2的残差似乎仍有部分趋势性，dat3残差基本为0。

模型评估

MSE（均方误差）是预测误差平方的平均值，衡量模型预测值与实际观测值差异的平方的平均程度。RMSE（均方根误差）是MSE的平方根，提供了与原始数据相同单位的误差量度，常用于评估模型的预测精度。MAE（平均绝对误差）是预测误差绝对值的平均值，衡量误差的平均大小，对异常值不敏感。这三个指标越小，表示模型预测性能越好。

In [249…
```python
def three(dat,pre):
    mse = mean_squared_error(dat['US'], pre)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(dat['US'], pre)
    return mse,rmse,mae
```

In [250…
```python
mse1, rmse1, mae1 = three(dat1, predictions1)
mse2, rmse2, mae2 = three(dat2, predictions2)
mse3, rmse3, mae3 = three(dat3, predictions3)
results = pd.DataFrame({
    'MSE': [mse1, mse2, mse3],
    'RMSE': [rmse1, rmse2, rmse3],
    'MAE': [mae1, mae2, mae3],
    'Dataset': ['Original', 'Monthly', 'Rollmean']})
print(results)
```

```
           MSE        RMSE         MAE    Dataset
0     7.330924    2.707568    1.404084  Original
1  1552.654434   39.403736   26.190026   Monthly
2    28.875502    5.373593    0.672827  Rollmean
```

通过比较不同模型的MSE、RMSE和MAE，发现滚动预测模型能够较好地捕捉原数据的特征，而月累和聚合数据在模型拟合上表现更为明显。

总结

本文通过对美国原油价格时间序列的分析和预测，展示了时间序列分析的基本流程。通过对不同数据集的处理和平稳性检验，建立了适合的时间序列模型，并通过滚动预测评估了模型效果。研究表明，模型能够较好地捕捉数据特征，但在预测准确性方面仍有改进空间。未来研究可以考虑更广泛的数据集和更复杂的模型，以提高预测的准确性。 本文的优点在于有意识地代码模块化可以套用于不同数据，但仍存在一些局限性。例如，模型定义不够严谨、报告书面不 够美观、可视化部分和解释部分略显单薄、数据集的时间范围可能限制了模型的泛化能力，且模型可能未能充分考虑所有可能影响原油价格的外部因素。未来的研究可以考虑更广泛的数据集和更复杂的模型，以进一步提高预测的准确性。由于US数据和World数据高度相关，可以尝试用US数据预测World数据，充分利用数据。同时也可以尝试将偶数年数据挖掉，进行拟合插补，测试模型发鲁棒性。由于算力的限制，不能一一尝试。　　总体而言，本文不仅为理解时间序列分析的基本过程提供了一个框架，而且为能源市场的分析和预测提供了些许实际的应用价值。