تابع read_forexfactory_data:

تابع `read_forexfactory_data` یک تابع است که برای خواندن و پردازش داده‌های فارکس فکتوری از فایل‌های اکسل طراحی شده است. این تابع دو ورودی دریافت می‌کند: `path` که مسیر دایرکتوری فایل اکسل را مشخص می‌کند و `file_name` که نام فایل اکسل (بدون پسوند) را مشخص می‌کند.

در ابتدا، تابع فایل اکسل را با استفاده از `pd.read_excel` می‌خواند و در یک DataFrame به نام `df` ذخیره می‌کند. سپس، تاریخ‌ها را از ستون 'History' استخراج می‌کند و به عنوان نمایه (index) برای DataFrame تنظیم می‌کند. برای این کار از تابع `pd.to_datetime` استفاده می‌کند.

سپس، یک ستون جدید به نام '[Actual file_name]' ایجاد می‌کند و مقادیر ستون 'Actual' اصلی را در آن قرار می‌دهد. ستون‌های غیرضروری مانند 'Previous'، 'Forecast'، 'Date'، 'History' و 'Actual' را از DataFrame حذف می‌کند تا فقط داده‌های مورد نیاز باقی بمانند.

در نهایت، DataFrame اصلی را با یک DataFrame خالی به نام 'final_df' ادغام کرده و آن را به عنوان نتیجه برمی‌گرداند.

توجه داشته باشید که DataFrame نهایی حاوی داده‌های فارکس فکتوری پردازش شده است.

مثال استفاده از این تابع به صورت زیر است:

```
forex_data = read_forexfactory_data('/path/to/data', 'file1')
print(forex_data.head())
```

این تابع به شما امکان می‌دهد فایل‌های اکسل حاوی داده‌های فارکس فکتوری را برای تحلیل یا مدل‌سازی به صورت ساده‌تر و کارآمدتری بخوانید و پردازش کنید.

تابع convert_monthly_to_daily:

تابع `convert_monthly_to_daily` یک تابع است که یک DataFrame از داده‌های با فرکانس ماهانه(مانند دیتا های اخبار) را می‌گیرد و همان داده‌ها را با فرکانس روزانه برمی‌گرداند. به عنوان مثال اگر خبر NFP در تاریخ 5ام ماه آئریل منتشر شده باشد و در

ماه بعدی یعنی ماه می در روز 8ام ماه منتشرشده باشد با کمک این تابع میتوانیم از 5ام ماه آئریل تا 8ام ماه می برچسب ها را اعداد NFP بزنیم.

این کار را با ایجاد یک DataFrame جدید با فرکانس روزانه بین هر ماه و تبدیل داده‌های ماهانه به داده‌های روزانه انجام می‌دهد.

پارامترها:

-final_df: یک DataFrame از داده‌های با فرکانس ماهانه.

خروجی:

- `pd.DataFrame`: یک DataFrame از داده‌های با فرکانس روزانه.

مراحل پردازش داده:

- تابع یک DataFrame خالی به نام 'new_df' ایجاد می‌کند.

- سپس از روی شاخصهای DataFrame ورودی 'final_df'، که فرض می‌شود یک DatetimeIndex با فرکانس ماهانه باشد، عبور می‌کند.

- برای هر ماه، تابع تاریخ‌های روزانه بین ماه جاری و ماه بعد را با استفاده از 'pd.date_range' تولید می‌کند.

- سپس از روی ستون‌های 'final_df' عبور می‌کند و یک DataFrame جدید به نام 'daily_df' با داده‌های فرکانس روزانه برای هر ستون ایجاد می‌کند.

- داده‌های روزانه با تکرار مقدار ماهانه برای هر روز در ماه تولید می‌شوند.

- سپس 'daily_df' با 'new_df' ادغام می‌شود تا مجموعه داده‌های فرکانس روزانه ساخته شود.

- تابع این فرآیند را برای همه ماه‌ها در DataFrame ورودی ادامه می‌دهد.

- 'new_df' حاوی همان داده‌های ورودی 'final_df' است اما با فرکانس روزانه.

مثال استفاده از این تابع به صورت زیر است:

```
monthly_data = pd.DataFrame({
    'Date': ['2022-01-01', '2022-02-01', '2022-03-01'],
    'Value': [100, 110, 120]
})
monthly_data['Date'] = pd.to_datetime(monthly_data['Date'])
```

```
monthly_data.set_index('Date', inplace=True)

daily_data = convert_monthly_to_daily(monthly_data)

print(daily_data.head())
```

```

توجه: این تابع مفید برای تبدیل داده‌های با فرکانس ماهانه به فرکانس روزانه است.

## تابع combine_daily_data:

تابع `combine_daily_data` یک تابع است که DataFrame های روزانه فایل‌های اکسل را به یک DataFrame نهایی ترکیب می‌کند.

این تابع دو پارامتر می‌گیرد: `files` که یک لیست از نام‌های فایل‌های اکسل برای خواندن است و `path` که مسیر دایرکتوری است که در آن فایل‌ها قرار دارند.

تابع یک DataFrame تکی را که تمام DataFrame های روزانه خوانده شده از فایل‌های اکسل مشخص شده را ترکیب می‌کند، برمی‌گرداند.

مراحل پردازش داده:

- تابع ابتدا یک DataFrame خالی به نام 'combined_daily_data' ایجاد می‌کند.

- سپس بر روی هر نام فایل در لیست 'files' حلقه می‌زند و داده‌های Forex Factory را از فایل اکسل با استفاده از تابع 'read_forexfactory_data' می‌خواند.

- سپس داده‌های ماهانه را با استفاده از تابع 'convert_monthly_to_daily' به داده‌های روزانه تبدیل می‌کند.

- DataFrame روزانه حاصل را با استفاده از متد 'join' با 'combined_daily_data' DataFrame ترکیب می‌کند.

- در نهایت، تابع 'combined_daily_data' DataFrame حاوی داده‌های روزانه Forex Factory از تمام فایل‌های اکسل مشخص شده را برمی‌گرداند.

پارامترها:

- files: یک لیست از نام‌های فایل‌ها (بدون پسوند) برای پردازش.

- path: مسیر دایرکتوری که فایل‌های اکسل در آن قرار دارند.
```

خروجی:

- `pd.DataFrame`: یک DataFrame ترکیبی حاوی داده‌های روزانه Forex Factory.

مثال استفاده از این تابع به صورت زیر است:

```
files_to_process = ['file1', 'file2']
data_path = '/path/to/excel/files'
combined_data = combine_daily_data(files_to_process, data_path)
print(combined_data.head())
```

توجه: این تابع برای تلفیق داده‌های روزانه Forex Factory از چندین فایل اکسل به یک DataFrame تکی برای تجزیه و تحلیل یا مدل‌سازی مفید است.

تابع read_investing_daily_data:

تابع read_investing_daily_data، یک فایل CSV حاوی داده‌های روزانه از investing.com را می‌خواند، داده‌ها را تمیز می‌کند و ستون‌های اضافی را حذف می‌کند. دو پارامتر path و file_name دریافت می‌کند که به ترتیب مسیر و نام فایل CSV را مشخص می‌کنند. تابع یک شیء DataFrame از Pandas با داده‌های تمیز شده را برمی‌گرداند.

مراحل تمیز کردن داده‌ها:

- فایل CSV را به یک شیء DataFrame از Pandas می‌خواند.
- سعی می‌کند ستون 'Date' را به یک شیء datetime تبدیل کند با استفاده از فرمت‌های تاریخ مختلف.
- ستون 'Date' را به عنوان شاخص DataFrame تنظیم می‌کند.
- DataFrame را بر اساس شاخص به ترتیب صعودی مرتب می‌کند.
- یک ستون جدید به نام 'file_name' ایجاد می‌کند که شامل مقادیر ستون 'Price' است.
- ستون‌های غیرضروری از DataFrame را حذف می‌کند، مانند 'Change'، 'Low'، 'High'، 'Open'، 'Price'، 'Date' % و 'Vol'.

پارامترها:

- files: یک لیست از نام‌های فایل‌ها (بدون پسوند) برای پردازش.

- path: مسیر دایرکتوری که فایل‌های اکسل در آن قرار دارند.

خروجی:

- pd.DataFrame: یک شیء DataFrame از Pandas حاوی داده‌های روزانه از فایل CSV تمیز شده.

مثال استفاده:

```
data_path = '/path/to/data/directory'

file_to_read = 'sample_data'

daily_data = read_investing_daily_data(data_path, file_to_read)

print(daily_data.head())

sample_data

Date          Price

2022-01-01 100.50

2022-01-02 101.20

2022-01-03 102.00

2022-01-04 104.50

2022-01-05 105.70
```

توجه: در این مثال، تابع داده‌های روزانه را از یک فایل CSV می‌خواند و مراحل تمیز کردن داده‌ها را انجام داده تا یک شیء DataFrame از Pandas با فرمت مورد نظر به دست آید.

تابع return_files_name :

تابع return_files_name نام‌های تمام فایل‌های موجود در یک دایرکتوری مشخص را برمی‌گرداند.

پارامترها:

- path (str): مسیر به دایرکتوری که می‌خواهید نام فایل‌های آن را به دست آورید.

خروجی:

- list: یک لیست شامل نام‌های فایل‌های موجود در دایرکتوری مشخص شده.

مثال استفاده:

```
directory_path = '/path/to/directory'
files_in_directory = return_files_name(directory_path)
print(files_in_directory)
['file1.txt', 'file2.csv', 'file3.xlsx']
```

توجه: این تابع برای به دست آوردن لیستی از نام‌های فایل در یک دایرکتوری کاربرد دارد که می‌توان از آن برای عملیات مرتبط با فایل استفاده کرد.

تابع monthly_features:

تابع `monthly_features` یک لیست از فایل‌های اکسل را می‌خواند و از آنها ویژگی استخراج می‌کند. میدانیم که اخبار اقتصادی حاوی مقدار های actual, forecast و previous هستند. برای حالت بندی اخبار از دو دسته بندی استفاده کردیم.

دسته اول: 6 حالت که این سه مقدار نسبت به هم میتوانند داشته باشند از 1 تا 6 مقدار دهی میشود. به عنوان مثال اگر actual بیشتر از forecast و previous منتشر شود به آن برچسب 1 میدهیم. این دست در داخل کد کامنت شده و برای استفاده باید آنکامنت شود و دسته ی دوم کامنت شود.

دسته دوم: 2 حالت برای این دسته در نظر گرفته شده. حالت اول, اگر actual بیشتر از forecast بیاید و حالت دوم اگر actual کمتر از forecast بیاید.

پارامترها:

- files: یک لیست از نام‌های فایل‌ها (بدون پسوند) که باید پردازش شوند.

- path: مسیر به دایرکتوری که فایل‌های اکسل در آن قرار دارند. مقدار پیش‌فرض `data/fund_model/monthly/../..` است.

خروجی:

- `pd.DataFrame`: یک شیء DataFrame از Pandas که حاوی داده‌های پردازش شده و ویژگی‌هاست.

ویژگی‌ها:

- تابع یک ویژگی دودویی به نام 'Actual_[file_name]' برای هر فایل در لیست 'files' محاسبه می‌کند بر اساس رابطه بین مقادیر 'Actual' و 'Forecast':

  - اگر 'Actual' بیشتر از 'Forecast' باشد، 'Actual_[file_name]' برابر با 1 تنظیم می‌شود.

  - اگر 'Actual' کمتر از 'Forecast' باشد، 'Actual_[file_name]' برابر با 0 تنظیم می‌شود.

مثال استفاده:

```

files_to_process = ['file1', 'file2']

features_df = monthly_features(files_to_process)

```

توجه: این تابع فایل‌های اکسل حاوی داده‌های اخبار را می‌خواند، آنها را پردازش می‌کند و با توجه به دسته بندی خاص داده ها را برای تحلیل استخراج می‌کند.

تابع discrete_to_continuous:

تابع `discrete_to_continuous` یک DataFrame از Pandas با فواصل زمانی گسسته را می‌گیرد و آن را با فواصل زمانی پیوسته تغییر می‌دهد، و یک DataFrame جدید با فواصل زمانی پیوسته را برمی‌گرداند. این تابع با حلقه‌ای روی هر ردیف از DataFrame و فاصله زمانی بین آن و ردیف بعدی کار می‌کند، سپس یک DataFrame جدید با فواصل زمانی روزانه برای آن فاصله ایجاد می‌کند.

سپس از روش forward filling برای تکمیل مقادیر در DataFrame جدید استفاده می‌کند و در نهایت با یک DataFrame نهایی که تمام فواصل زمانی روزانه را ترکیب می‌کند، ترکیب می‌کند.

DataFrame ورودی باید یک شاخص datetime داشته باشد و فواصل زمانی بین ردیف‌ها باید به طول ثابت باشد.

DataFrame خروجی یک شاخص datetime با فواصل زمانی پیوسته دارد و هر مقدار ناپیدا را با استفاده از forward filling پر می‌کند.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی پاندا با فواصل زمانی گسسته.

خروجی:

- `pd.DataFrame`: یک DataFrame جدید با فواصل زمانی پیوسته و مقادیر تکمیل شده.

مثال استفاده:

```

import pandas as pd

from datetime import datetime
```

```
data = {'Value': [10, 20, 30],
        'Date': [datetime(2022, 1, 1), datetime(2022, 1, 3), datetime(2022, 1, 5)]}
df = pd.DataFrame(data)
df.set_index('Date', inplace=True)

continuous_df = discrete_to_continuous(df)
print(continuous_df)
```

توجه: در این مثال، تابع مقادیر برای فواصل زمانی پیوسته بین فواصل زمانی گسسته در DataFrame ورودی تکمیل می‌کند.

تابع combine_investing_data:

تابع `combine_investing_data` یک لیست از نام‌های فایل‌ها و یک مسیر به دایرکتوری حاوی فایل‌های داده سایت investing را به عنوان ورودی می‌گیرد و یک DataFrame پاندا ترکیب شده از داده‌های روزانه سایت investing را برمی‌گرداند.

تابع ابتدا یک شیء خالی DataFrame پاندا به نام `combined_daily_data` ایجاد می‌کند.

سپس با حلقه‌ای روی هر نام فایل در لیست `files_name` و با استفاده از تابع `read_investing_daily_data`، داده‌های روزانه investing را می‌خواند.

DataFrame حاصل سپس با استفاده از روش join با `combined_daily_data` ترکیب می‌شود.

پس از پردازش تمام فایل‌ها، DataFrame نهایی `combined_daily_data` برگردانده می‌شود که حاوی داده‌های روزانه investing ترکیب شده است.

پارامترها:

- files: یک لیست از نام‌های فایل‌ها برای پردازش و ترکیب.

- path: مسیر به دایرکتوری حاوی فایل‌های داده سرمایه‌گذاری.

خروجی:

- `pd.DataFrame`: یک DataFrame ترکیبی که حاوی داده‌های روزانه investing از تمام فایل‌های مشخص شده است.

مثال استفاده:

```

file_names = ['stock_data_1.csv', 'stock_data_2.csv', 'stock_data_3.csv']

data_path = '/path/to/data/directory'

combined_data = combine_investing_data(file_names, data_path)

print(combined_data.head())

```

توجه: در این مثال، تابع داده‌های روزانه سایت investing را از تمام فایل‌های مشخص شده می‌خواند و آنها را در یک DataFrame ترکیب می‌کند.

تابع convert_str_to_float:

تابع `convert_str_to_float` یک DataFrame از Pandas را به عنوان ورودی می‌گیرد و نوع داده‌های رشته‌ای در هر ستون را به float تبدیل می‌کند.

تابع از طریق حلقه‌ای روی هر ستون در DataFrame حرکت می‌کند و برای هر ستون، از طریق حلقه‌ای روی هر سطر در آن ستون حرکت می‌کند.

اگر سلول خاصی در DataFrame نوع رشته‌ای داشته باشد، تابع هرگونه کاما را در رشته حذف می‌کند تا آن را به یک مقدار عددی float تبدیل کند.

در نهایت، تابع با استفاده از روش astype، ستون را به float تبدیل می‌کند.

تابع DataFrame ورودی را در محل تغییر می‌دهد و DataFrame تغییر یافته را به عنوان خروجی برمی‌گرداند.

توجه کنید که تابع فرض می‌کند هر مقدار رشته‌ای در DataFrame را پس از حذف کاما می‌توان به float تبدیل کرد.

اگر یک مقدار نتواند به float تبدیل شود، تابع یک خطا را برمی‌گرداند.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.

خروجی:

- `DataFrame`: pd.DataFrame تغییر یافته که مقادیر رشته‌ای به float تبدیل شده‌اند.

مثال استفاده:

```
```

```
import pandas as pd

data = {'Price': ['1,000.50', '2,000.75', '3,500.25', '4,200.00', '5,800.90'],
        'Quantity': ['1,000', '2,500', '3,200', '4,000', '5,100']}
df = pd.DataFrame(data)
converted_df = convert_str_to_float(df)
print(converted_df)
```

توجه: در این مثال، تابع مقادیر رشته‌ای در ستون‌های 'Price' و 'Quantity' را به float تبدیل می‌کند و در این فرآیند کاماها را حذف می‌کند.

تابع return_price:

تابع `return_price` یک DataFrame از Pandas به نام `df` را به عنوان ورودی می‌گیرد و یک DataFrame جدید از Pandas حاوی قیمت بازگشتی محاسبه شده را برمی‌گرداند.

تابع با استفاده از روش `pct_change()` روی هر ستون DataFrame ورودی، قیمت بازگشتی را محاسبه می‌کند.

در واقع، برای هر ستون، این روش تغییر درصد بین عضو فعلی و عضو قبلی را محاسبه می‌کند که قیمت بازگشتی را نشان می‌دهد.

تابع یک DataFrame جدید از Pandas به نام `new_df` با همان شاخص DataFrame ورودی ایجاد می‌کند.

برای هر ستون در DataFrame ورودی، ستون جدیدی با پسوند '_return_price' به نام ستون در `new_df` ایجاد می‌شود.

ستون‌های جدید، قیمت بازگشتی محاسبه شده را برای هر ستون متناظر در DataFrame ورودی شامل می‌شوند.

توجه کنید که DataFrame ورودی نباید شامل هیچ مقدار ناقصی باشد. اگر مقادیر ناقصی وجود داشته باشد، تابع یک DataFrame با مقادیر ناقص را برمی‌گرداند.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.

خروجی:

- `DataFrame`: pd.DataFrame جدید از Pandas حاوی قیمت بازگشتی محاسبه شده.

مثال استفاده:

```

import pandas as pd

data = {'Stock_A': [100, 105, 110, 115, 120],
        'Stock_B': [50, 55, 60, 65, 70]}
df = pd.DataFrame(data)
return_prices_df = return_price(df)
print(return_prices_df)
```

توجه: در این مثال، تابع قیمت بازگشتی برای ستون‌های 'Stock_A' و 'Stock_B' را در DataFrame ورودی محاسبه می‌کند.

تابع get_redundant_pairs:

تابع `get_redundant_pairs` یک DataFrame از Pandas به نام `df` را به عنوان ورودی می‌گیرد و مجموعه‌ای از جفت‌های تکراری از ستون‌ها را برمی‌گرداند.

تابع برای هر ستون از DataFrame، برای تمام جفت‌های ممکن از ستون‌ها، از طریق حلقه‌ها به دست می‌آورد.

سپس یک مجموعه از تمام جفت‌های ستون‌ها را ایجاد می‌کند که در قطر ماتریس همبستگی یا پایین آن قرار دارند.

از آنجا که ماتریس همبستگی تقارن دارد، این کار باعث می‌شود تا همه جفت‌های تکراری ستون‌ها را دریافت کنیم.

در نهایت، تابع مجموعه جفت‌های تکراری را برمی‌گرداند.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.

خروجی:

- `pd.DataFrame`: مجموعه‌ای از جفت‌های نام ستون‌ها که نشان‌دهنده جفت‌های تکراری ستون‌ها در DataFrame است.

مثال استفاده:

```

import pandas as pd
```

```
data = {'A': [1, 2, 3, 4, 5],
        'B': [0.1, 0.2, 0.3, 0.4, 0.5],
        'C': [0.5, 0.4, 0.3, 0.2, 0.1]}
df = pd.DataFrame(data)
redundant_pairs = get_redundant_pairs(df)
print(redundant_pairs)
```

توجه: در این مثال، تابع مجموعه‌ای از جفت‌های تکراری ستون‌ها را در DataFrame برمی‌گرداند که شامل جفت‌های قطری و جفت‌های مثلثی پایین است.

تابع get_top_abs_correlations:

تابع `get_top_abs_correlations` یک DataFrame از Pandas به نام `df` و یک عدد صحیح `n` را به عنوان ورودی می‌گیرد و n بالاترین همبستگی‌های مطلق در DataFrame را برمی‌گرداند.

تابع ابتدا ماتریس همبستگی برای DataFrame را با استفاده از متد `corr()` محاسبه می‌کند.

سپس برای هر جفت ستون، همبستگی مطلق را با استفاده از متد `abs()` و تابع `unstack()` محاسبه می‌کند.

سپس با استفاده از تابع `get_redundant_pairs()` که جفت‌های تکراری در ماتریس همبستگی را برمی‌گرداند، جفت‌های تکراری همبستگی‌ها را حذف می‌کند.

در نهایت، تابع جفت‌های باقی‌مانده همبستگی‌ها را به ترتیب نزولی مرتب می‌کند و n جفت برتر را با استفاده از ایندکسینگ برمی‌گرداند.

خروجی یک شیء سری از Pandas با n جفت برتر همبستگی‌ها و مقادیر متناظر آنها است.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.
- n: تعداد بالاترین همبستگی‌های مطلق برای بازگشت.

خروجی:

- `pd.Series`: یک شیء سری Pandas که شامل n جفت برتر همبستگی‌ها و مقادیر متناظر آنها است، به ترتیب نزولی مرتب شده است.

مثال استفاده:

```
import pandas as pd

data = {'A': [1, 2, 3, 4, 5],
        'B': [0.1, 0.2, 0.3, 0.4, 0.5],
        'C': [0.5, 0.4, 0.3, 0.2, 0.1]}
df = pd.DataFrame(data)
top_correlations = get_top_abs_correlations(df, 2)
print(top_correlations)
```

توجه: در این مثال، تابع n بالاترین همبستگی‌های مطلق را در DataFrame برمی‌گرداند.

تابع get_top_corr_with_gold:

تابع `get_top_corr_with_gold` یک DataFrame از Pandas به نام `df` و یک رشته `target_file_name` را به عنوان ورودی می‌گیرد و یک DataFrame حاوی مقادیر همبستگی بین `target_file_name` و سایر ویژگی‌ها در DataFrame را به ترتیب نزولی برمی‌گرداند.

تابع ابتدا سعی می‌کند همبستگی بین `target_file_name` و سایر ویژگی‌ها در DataFrame ورودی را با استفاده از متد `corr()` بدست آورد، و در صورت شکست، سعی می‌کند همبستگی بین `target_file_name` و ویژگی‌هایی که با `target_file_name` به پایان یک رشته `_labeled` می‌چسبند را بدست آورد.

سپس مقادیر همبستگی را به ترتیب نزولی مرتب می‌کند و آنها را در یک DataFrame جدید به همراه نام ویژگی متناظر ذخیره می‌کند. ردیف اول حذف می‌شود زیرا مربوط به همبستگی `target_file_name` با خودش است. سپس DataFrame نتیجه‌ی حاصل را برمی‌گرداند.

پارامترها:

- df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.

- target_file_name: نام ویژگی هدف که همبستگی‌ها برای آن محاسبه خواهد شد.

خروجی:

- `DataFrame`: pd.DataFrame حاوی مقادیر همبستگی بین ویژگی هدف و سایر ویژگی‌ها، به ترتیب نزولی مرتب شده است.

مثال استفاده:

```

import pandas as pd


data = {'A': [1, 2, 3, 4, 5],

        'B': [0.1, 0.2, 0.3, 0.4, 0.5],

        'C': [0.5, 0.4, 0.3, 0.2, 0.1]}

df = pd.DataFrame(data)

target_file_name = 'A'

correlation_df = get_top_corr_with_gold(df, target_file_name)

print(correlation_df)

```

توجه: در این مثال، `target_file_name` برابر با 'A' است و تابع یک DataFrame حاوی مقادیر همبستگی بین 'A' و سایر ویژگی‌ها برمی‌گرداند.

تابع create_new_time_features:

تابع `create_new_time_features` یک DataFrame از Pandas به نام `df` را به عنوان ورودی می‌گیرد و ویژگی‌های جدید مبتنی بر زمان ایجاد کرده و آنها را به DataFrame ورودی اضافه می‌کند.

تابع یک DataFrame از Pandas به نام `df` را به عنوان ورودی می‌گیرد و یک DataFrame اصلاح شده را با دو ستون جدید به نام "Day of week" و "Month of year" برمی‌گرداند.

ستون "Day of week" روز هفته را برای هر نقطه داده در شاخص نشان می‌دهد، به طوری که دوشنبه برابر با 0 و یکشنبه برابر با 6 است.

ستون "Month of year" ماه سال را برای هر نقطه داده در شاخص نشان می‌دهد، به طوری که ژانویه برابر با 1 و دسامبر برابر با 12 است.

تابع DataFrame ورودی را با اضافه کردن این دو ستون جدید اصلاح می‌کند و DataFrame اصلاح شده را برمی‌گرداند.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.

خروجی:

- `DataFrame`: pd.DataFrame اصلاح شده با ستون‌های جدید مبتنی بر زمان.

مثال استفاده:

```
import pandas as pd

date_rng = pd.date_range(start='2022-01-01', end='2022-01-05', freq='D')
data = {'Value': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data, index=date_rng)
modified_df = create_new_time_features(df)
print(modified_df)
```

خروجی:

```
            Value  Day of week  Month of year
2022-01-01     10            5              1
2022-01-02     20            6              1
2022-01-03     30            0              1
2022-01-04     40            1              1
```

2022-01-05 50 2 1

```

توجه: در این مثال، تابع یک DataFrame اصلاح شده را برمی‌گرداند که دو ستون جدید "Day of week" و "Month of year" را شامل می‌شود که مقادیر آنها بر اساس زمان است.

## تابع create_nonlinear_features:

تابع `create_nonlinear_features` یک DataFrame از Pandas به نام `df` و یک عدد صحیح به نام `power_upto` را به عنوان ورودی می‌گیرد و یک DataFrame جدید از Pandas را برمی‌گرداند که حاوی ویژگی‌های غیرخطی تا قدرت مشخص شده است. تابع به ازای هر ستون در DataFrame ورودی حلقه زده و ستون‌های جدیدی با ویژگی‌های غیرخطی تا قدرت مشخص شده ایجاد می‌کند. به عنوان مثال، اگر `power_upto` به 3 تنظیم شود، برای هر ستون در DataFrame ورودی، تابع سه ستون جدید با قدرت 1، 2 و 3 از ستون اصلی ایجاد می‌کند.

DataFrame جدید همان شاخص DataFrame ورودی را دارد.

توجه: این تابع می‌تواند برای ایجاد ویژگی‌های چندجمله‌ای برای یک مدل یادگیری ماشین استفاده شود.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.
- `power_upto`: حداکثر توان مجاز برای تبدیل. تابع ستون‌هایی با توان‌های 1 تا power_upto را ایجاد خواهد کرد.

خروجی:

- `DataFrame`: pd.DataFrame جدید با ویژگی‌های غیرخطی تا قدرت مشخص شده.

مثال استفاده:

```

```
import pandas as pd


data = {'A': [1, 2, 3, 4, 5],
        'B': [0.1, 0.2, 0.3, 0.4, 0.5]}
df = pd.DataFrame(data)
nonlinear_features_df = create_nonlinear_features(df, 3)
```

```
print(nonlinear_features_df)
```

```
خروجی:
```

```
   A power1  A power2  A power3  B power1  B power2  B power3
0         1         1         1       0.1      0.01     0.001
1         2         4         8       0.2      0.04     0.008
2         3         9        27       0.3      0.09     0.027
3         4        16        64       0.4      0.16     0.064
4         5        25       125       0.5      0.25     0.125
```

توجه: در این مثال، تابع یک DataFrame جدید را برمی‌گرداند که شامل ستون‌های جدید "A power1"، "A power2"، "A power3"، "B power1"، "B power2" و "B power3" است که مقادیر آنها بر اساس توان‌های مشخص شده است.

تابع create_nonlinear_features_with_power_Q:

تابع `create_nonlinear_features_with_power_Q` یک DataFrame از Pandas به نام `df` و یک عدد صحیح به نام `power_upto` را به عنوان ورودی می‌گیرد و ستون‌های جدیدی را در DataFrame ورودی ایجاد می‌کند با اعمال تبدیلات توان به هر ستون موجود تا حداکثر قدرت مشخص شده. این تابع از طریق هر ستون در DataFrame حلقه می‌زند و با بالا بردن مقادیر ستون به توان برابر با معکوس عدد i برای i در محدوده 0 تا power_upto، ستون‌های جدیدی را ایجاد می‌کند. اگر i برابر با 0 باشد، توان برابر با بی‌نهایت می‌شود. تابع DataFrame به‌روزشده را برمی‌گرداند.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.
- `power_upto` : حداکثر توان مجاز برای تبدیل. تابع ستون‌هایی با توان‌های 0 تا power_upto را ایجاد خواهد کرد.

خروجی:

- `DataFrame`: pd.DataFrame جدید با نسخه‌های قدرتی تبدیل شده از ستون‌های اصلی.

مثال استفاده:

```python
import pandas as pd


data = {'A': [1, 2, 3, 4, 5],
        'B': [0.1, 0.2, 0.3, 0.4, 0.5]}
df = pd.DataFrame(data)
transformed_df = create_nonlinear_features_with_power_Q(df, 3)
print(transformed_df)
```

خروجی:

```
   A    B  A powerinf  B powerinf  A power1.0  B power1.0  A power0.5  B power0.5  A
power0.3333333333333333  B power0.3333333333333333
0  1  0.1         1.0         0.1         1.0         0.1    1.000000    0.316228                1.0
0.464158
1  2  0.2         2.0         0.2         2.0         0.2    1.414214    0.447214                1.0
0.584804
2  3  0.3         3.0         0.3         3.0         0.3    1.732051    0.547723                1.0
0.671582
3  4  0.4         4.0         0.4         4.0         0.4    2.000000    0.632456                1.0
0.741620
4  5  0.5         5.0         0.5         5.0         0.5    2.236068    0.707107                1.0
0.800000
```

توجه: در این مثال، تابع یک DataFrame جدید را برمی‌گرداند که شامل ستون‌های جدید "A"، "B powerinf"، "A powerinf"، "B power0.5"، "A power0.3333333333333333"، "B power1.0"، "power1.0" و "B power0.3333333333333333" است که مقادیر آنها بر اساس توبا توجه به توضیحات، تابع `create_nonlinear_features_with_power_Q` یک DataFrame به نام `df` و یک عدد صحیح به نام `power_upto` را به عنوان ورودی می‌گیرد. سپس برای هر ستون در DataFrame، از توان 0 تا `power_upto` استفاده می‌کند و ستون‌های جدید با نام‌هایی مانند `B power1.0`، `A power1.0`، `B powerinf`، `A powerinf` و غیره را ایجاد می‌کند. این ستون‌های جدید مقدار ستون مربوطه را به توان معکوس عدد `i` برمی‌گردانند. اگر `i` برابر با 0 باشد، توان برابر با بی‌نهایت می‌شود.

در آخر، DataFrame به‌روزشده را برمی‌گرداند.

لطفاً توجه داشته باشید که مقدار توان برای ستون‌هایی که مقدار اصلی آن‌ها صفر است، برابر با بی‌نهایت (infinity) خواهد بود.

تابع exp_function:

تابع `exp_function` یک DataFrame از Pandas به نام `df` را به عنوان ورودی می‌گیرد و ستون‌های جدیدی را در DataFrame ورودی ایجاد می‌کند با اعمال تابع نمایی بر روی هر ستون موجود. این تابع از طریق هر ستون در DataFrame حلقه می‌زند و مقادیر ستون را به تابع `np.exp` می‌دهد تا نسخه نمایی از ستون ایجاد شود. سپس ستون‌های جدید را به DataFrame جدید اضافه می‌کند و DataFrame به‌روزشده را برمی‌گرداند.

پارامترها:

- `df` (pd.DataFrame): DataFrame ورودی Pandas که حاوی داده‌ها است.

خروجی:

- `pd.DataFrame`: DataFrame جدید با نسخه‌های نمایی از ستون‌های اصلی.

مثال استفاده:

```
import pandas as pd
import numpy as np
```

```
data = {'A': [1, 2, 3, 4, 5],
        'B': [0.1, 0.2, 0.3, 0.4, 0.5]}
df = pd.DataFrame(data)
exponential_df = exp_function(df)
print(exponential_df)
```

خروجی:

```
      A exp     B exp
0   2.718282  1.105171
1   7.389056  1.221403
2  20.085537  1.349859
3  54.598150  1.491825
4 148.413159  1.648721
```

توجه: در این مثال، تابع یک DataFrame جدید را برمی‌گرداند که شامل ستون‌های جدید "A exp" و "B exp" است که مقادیر آنها بر اساس تابع نمایی از ستون‌های اصلی محاسبه شده است.

تابع labeling_target:

تابع `labeling_target` یک مسیر فایل و لیستی از نام‌های فایل را به عنوان پارامترهای ورودی می‌گیرد. این تابع هر فایل را از مسیر مشخص شده خوانده و عملیات زیر را روی هر فایل انجام می‌دهد:

1) علامت درصد (%) را در ستون 'Change %' با یک رشته خالی جایگزین می‌کند و مقادیر را به نوع داده‌ای float تبدیل می‌کند.

2) هر ردیف در ستون 'Change %' را به 1 برچسبگذاری می‌کند اگر مقدار بیشتر یا مساوی با صفر باشد و در غیر اینصورت آن را به 0 برچسبگذاری می‌کند.

3) ستون 'Date' را با استفاده از یکی از فرمت‌های تاریخ مشخص شده به نوع داده‌ای datetime تبدیل می‌کند و آن را به عنوان فهرست اصلی DataFrame تنظیم می‌کند.

4) DataFrame را بر اساس فهرست به ترتیب صعودی مرتب می‌کند.

5) ستون جدیدی با نام نام فایل به اضافه عبارت '_labeled' به DataFrame اضافه می‌کند که شامل مقادیر '% Change' برچسب‌گذاری شده است.

6) ستون‌های غیرضروری را از DataFrame حذف می‌کند.

تابع DataFrame نهایی به‌روزشده را برمی‌گرداند.

پارامترها:

- `path`: مسیری که فایل‌های CSV در آن قرار دارند.

- `files_name`: لیستی از نام‌های فایل (بدون پسوند '.csv') که باید پردازش شوند.

خروجی:

- `DataFrame`: pd.DataFrame نهایی پس از پردازش همه فایل‌ها.

مثال استفاده:

```
path = 'data_folder'
files_name = ['file1', 'file2', 'file3']
final_df = labeling_target(path, files_name)
print(final_df.head())
```

این کد، DataFrame نهایی را ایجاد می‌کند که شامل ستون‌های `file2_labeled`، `file1_labeled` و `file3_labeled` است. DataFrame ایجاد شده شامل مقادیر برچسب‌گذاری شده '% Change' برای هر فایل است. سپس این DataFrame را چاپ می‌کند.

توجه: تابع فرض می‌کند که فرمت تاریخ در فایل‌ها یا "%b %d, %Y"، یا "%m/%d/%Y"، یا "%d/%m/%Y" است. تابع به ترتیب این فرمت‌ها را بررسی می‌کند و تاریخ را به نوع datetime تبدیل می‌کند. در صورتی که هیچکدام از این فرمت‌ها با تاریخ‌ها

مطابقت نداشته باشد، تابع فرض می‌کند که فرمت تاریخ در فایل‌ها "d/%m/%Y%" است. در نهایت، ستون‌های "Date"، "Price"، "Change"، "Low"، "High"، "Open" % و "Vol." از DataFrame حذف می‌شوند.

تابع lag_counter:

تابع `lag_counter` یک DataFrame pandas به نام 'df' و یک عدد صحیح به نام 'number' را به عنوان ورودی دریافت می‌کند. این تابع یک DataFrame جدید با ستون‌هایی که نسخه‌های لگ از ستون‌های اصلی را نشان می‌دهند، تا تعداد مشخص شده ایجاد می‌کند.

پارامترها:

- `df` (pd.DataFrame): DataFrame pandas ورودی که حاوی داده‌ها است.
- `number`: تعداد نسخه‌های لگ برای ایجاد برای هر ستون.

خروجی:

- `DataFrame`: pd.DataFrame جدید با ستون‌های لگ.

مثال استفاده:

```
import pandas as pd
data = {'A': [1, 2, 3, 4, 5],
        'B': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)
lagged_df = lag_counter(df, 2)
print(lagged_df)
```

این کد، DataFrame جدیدی را ایجاد می‌کند که شامل ستون‌های `Blag1`، `Alag2`، `Alag1` و `Blag2` است. DataFrame ایجاد شده شامل نسخه‌های لگ ستون‌های اصلی است. DataFrame ایجاد شده را چاپ می‌کند.

در این تابع، ابتدا یک DataFrame جدید به نام `new_df` ایجاد می‌شود. سپس برای هر ستون در DataFrame ورودی، نام ستون به 'test' تغییر نام داده می‌شود. سپس برای اعداد 1 تا `number`، ستون‌های لگ با نام `(column+'lag'+str(i` در

`new_df` ایجاد می‌شود که مقادیری نسخه لگ شده از ستون 'test' در `df` را نشان می‌دهد. سپس نام ستون 'test' به نام اصلی ستون در `df` تغییر نام می‌دهد. در نهایت، DataFrame جدید `new_df` برگردانده می‌شود.

تابع create_folder:

تابع `create_folder` یک مسیر و یک نام پوشه را به عنوان ورودی دریافت می‌کند و یک پوشه جدید با نام مشخص شده در مسیر مشخص شده ایجاد می‌کند.

پارامترها:

- `path`: مسیری که پوشه جدید باید در آن ایجاد شود.
- `folder_name`: نام پوشه جدیدی که باید ایجاد شود.

خروجی:

- `None`

مثال استفاده:

```
import os

# ایجاد یک پوشه جدید به نام 'results' در مسیر کاری کنونی
create_folder(os.getcwd(), 'results')
```

این کد یک پوشه جدید با نام 'results' در مسیر کاری کنونی ایجاد می‌کند. در این تابع، مسیر پوشه جدید با ترکیب مسیر و نام پوشه مشخص شده ساخته می‌شود. سپس با استفاده از تابع `os.makedirs`، یک پوشه با مسیر و نام مشخص شده ایجاد می‌شود. در صورتی که ایجاد پوشه موفقیت آمیز نباشد، یک پیام خطا چاپ می‌شود. در غیر اینصورت، یک پیام موفقیت آمیز بودن ایجاد پوشه چاپ می‌شود.

تابع count_depression_value:

تابع `count_inflation_value` یک DataFrame pandas به نام 'df' و یک رشته به نام 'name' که نام دلخواه مد نظر است را به عنوان ورودی دریافت می‌کند. این نام فقط برای مشخص شدن تکیه ای نام ستون خروجی هست که نشان دهد این بررسی مربوط به کدام کشور است. این تابع مقدار کل depression را برای هر ماه در DataFrame ارائه شده محاسبه می‌کند.در کد ، ابتدا یک

DataFrame خالی به نام 'monthly_totals' با یک فرکانس ماهانه برای شاخص تاریخ زمانی ایجاد می‌شود که دامنه کامل DataFrame اصلی را پوشش می‌دهد.

سپس به تمام ماه‌ها در شاخص 'monthly_totals' حلقه زده, ردیف‌های DataFrame اصلی را با سال و ماه مشابه با حلقه فعلی انتخاب می‌کنیم. ما مجموع مقادیر برای هر ستون در این ردیف‌های انتخاب شده با استفاده از روش `sum` محاسبه می‌کنیم و مقادیر حاصل را باز هم با استفاده از روش `sum` برای همه ستون‌ها جمع می‌کنیم. سپس مقدار کل به عنوان یک ردیف جدید به 'DataFrame 'monthly_totals اضافه می‌شود و شاخص آن به ماه فعلی تنظیم می‌شود.

در پایان حلقه، 'DataFrame 'monthly_totals یک ستون به نام 'total_depression_'+name دارد که شامل مجموع تمام مقادیر ستون‌ها برای هر ماه در DataFrame اصلی است و یک شاخص تاریخ زمانی با یک ردیف برای هر ماه دارد. اگر برخی از ماه‌ها داده‌هایی نداشته باشند (به عبارت دیگر، مقادیر گم شده داشته باشند)، ما این ردیف‌ها را با استفاده از روش `dropna` با `inplace=True` حذف می‌کنیم.

خروجی:

- `pd.DataFrame`: یک DataFrame با یک شاخص تاریخ زمانی که شامل مقادیر کل depression برای هر ماه است.

مثال استفاده:

```
depression_data = pd.read_csv('depression_data.csv')
total_depression = count_depression_value(depression_data, 'usa')
```

این کد مقادیر کل depression را برای هر ماه در داده‌های DataFrame محاسبه می‌کند.

تابع count_inflation_value:

تابع `count_inflation_value` یک DataFrame pandas به نام 'df' و یک رشته به نام 'name' که نام دلخواه مد نظر است را به عنوان ورودی دریافت می‌کند. این نام فقط برای مشخص شدن تکیه ای نام ستون خروجی هست که نشان دهد این بررسی مربوط به کدام کشور است. این تابع مقدار کل inflationرا برای هر ماه در DataFrame ارائه شده محاسبه می‌کند.در کد ، ابتدا یک DataFrame خالی به نام 'monthly_totals' با یک فرکانس ماهانه برای شاخص تاریخ زمانی ایجاد می‌شود که دامنه کامل DataFrame اصلی را پوشش می‌دهد.

سپس به تمام ماه‌ها در شاخص 'monthly_totals' حلقه زده, ردیف‌های DataFrame اصلی را با سال و ماه مشابه با حلقه فعلی انتخاب می‌کنیم. ما مجموع مقادیر برای هر ستون در این ردیف‌های انتخاب شده با استفاده از روش `sum` محاسبه می‌کنیم و مقادیر حاصل را باز هم با استفاده از روش `sum` برای همه ستون‌ها جمع می‌کنیم. سپس مقدار کل به عنوان یک ردیف جدید به DataFrame 'monthly_totals' اضافه می‌شود و شاخص آن به ماه فعلی تنظیم می‌شود.

در پایان حلقه، DataFrame 'monthly_totals' یک ستون به نام 'total_inflation_'+name دارد که شامل مجموع تمام مقادیر ستون‌ها برای هر ماه در DataFrame اصلی است و یک شاخص تاریخ زمانی با یک ردیف برای هر ماه دارد. اگر برخی از ماه‌ها داده‌هایی نداشته باشند (به عبارت دیگر، مقادیر گم شده داشته باشند)، ما این ردیف‌ها را با استفاده از روش `dropna` با `inplace=True` حذف می‌کنیم.

خروجی:

- `pd.DataFrame`: یک DataFrame با یک شاخص تاریخ زمانی که شامل مقادیر کل inflationبرای هر ماه است.

مثال استفاده:

```
inflation_data = pd.read_csv('inflation_data.csv')
total_depression = count_depression_value(depression_data, 'usa')
```

این کد مقادیر کل depression را برای هر ماه در داده‌های DataFrame محاسبه می‌کند.

تابع count_monthly_price_change:

تابع `count_monthly_price_change` یک DataFrame pandas به نام 'df' را به عنوان ورودی دریافت می‌کند و تغییرات قیمت ماهانه را به عنوان مقادیر دودویی (1 برای تغییر مثبت و 0 برای تغییر غیر مثبت/منفی) محاسبه می‌کند.

در این تابع:

- یک DataFrame جدید به نام 'monthly_df' با فرکانس ماهانه و روز اول هر ماه به عنوان شاخص ایجاد می‌شود.

- تغییرات قیمت ماهانه با استفاده از روش 'resample' محاسبه می‌شود.

- تغییرات قیمت را به مقادیر دودویی تبدیل می‌کند، به طوری که 1 تغییر مثبت قیمت را نشان می‌دهد و 0 تغییر غیر مثبت/منفی قیمت را نشان می‌دهد.

- ردیف اول که ماه قبلی برای مقایسه وجود ندارد، از نتیجه حذف می‌شود.

خروجی:

- `pd.DataFrame`: یک DataFrame جدید با اطلاعات تغییرات قیمت ماهانه.

مثال استفاده:

```
daily_prices = pd.DataFrame({'Price': [100, 105, 102, 110, 108]},
                            index=pd.date_range(start='2023-01-01', periods=5, freq='D'))

monthly_changes = count_monthly_price_change(daily_prices)
print(monthly_changes)
```

این کد تغییرات قیمت ماهانه را به عنوان مقادیر دودویی محاسبه می‌کند و نتیجه را در یک DataFrame جدید به نام 'monthly_changes' چاپ می‌کند.

تابع compare_depression_with_price_change:

تابع `compare_depression_with_price_change` دو DataFrame به نام `df_depression_value` و `df_price_change` را به عنوان ورودی دریافت می‌کند و ارزیابی مقادیر depression با تغییرات قیمت ماهانه را انجام می‌دهد و یک DataFrame خلاصه را تولید و آن را به عنوان یک فایل CSV ذخیره می‌کند.

در این تابع:

- ابتدا دو DataFrame `df_depression_value` و `df_price_change` را در هم ترکیب کرده و به عنوان ورودی 'df' استفاده می‌کنیم.

- سپس DataFrame را بر اساس مقادیر در ستون 'total' گروه‌بندی می‌کنیم و تعداد وجود 0 و 1 در ستون 'price_change' را محاسبه می‌کنیم.

- نام ستون‌ها را به 'count_0' و 'count_1' تغییر می‌دهیم تا نشان دهند که تعداد برای 0 و 1 است.

- ستون‌های 'percent_0' و 'percent_1' را اضافه می‌کنیم که درصد 0 و 1 را نشان می‌دهند.

- DataFrame را به فرمت CSV ذخیره می‌کنیم با نام و مسیر داده شده.

- در نهایت، DataFrame خلاصه را برمی‌گردانیم.

خروجی:

- `pd.DataFrame`: DataFrame خلاصه با تعداد و درصدهای 0 و 1 در ستون 'price_change' بر اساس ستون 'total'.

مثال استفاده:

```
depression_df = pd.read_csv('depression_values.csv')
price_change_df = pd.read_csv('price_changes.csv')
result_summary = compare_depression_with_price_change(depression_df, price_change_df,
'results', 'comparison_result')
```

این کد دو DataFrame `depression_df` و `price_change_df` را به عنوان ورودی در تابع `compare_depression_with_price_change` می‌فرستد و نتیجه را در یک DataFrame به نام 'result_summary' ذخیره می‌کند.

تابع compare_depression_of_2countries:

تابع `compare_depression_of_2countries` دو DataFrame به نام `df1` و `df2` برای نرخ depression دو کشور را به عنوان ورودی دریافت می‌کند. همچنین یک DataFrame به نام `df_price_change` که شامل داده‌های تغییرات قیمت ارز است را نیز دریافت می‌کند. این تابع نرخ depression دو کشور را مقایسه می‌کند و یک DataFrame خلاصه را تولید و آن را به عنوان یک فایل CSV ذخیره می‌کند.

در این تابع:

- ابتدا سه `df2`، `df1` DataFrame و `df_price_change` را در هم ترکیب می‌کنیم و به عنوان ورودی 'df' استفاده می‌کنیم.
- ستون `compare_depression` را به عنوان ستون جدیدی با مقادیر NaN ایجاد می‌کنیم.
- سپس برای هر ردیف در DataFrame، اگر مقدار ستون اول (`df1`) بزرگتر یا مساوی مقدار ستون دوم (`df2`) باشد، مقدار ستون `compare_depression` را 1 قرار می‌دهیم؛ در غیر این صورت، مقدار آن را 0 قرار می‌دهیم.
- سپس DataFrame را بر اساس ستون `compare_depression` و `price_change` گروه‌بندی و تعداد وجود 0 و 1 را در ستون `price_change` محاسبه می‌کنیم.
- نام ستون‌ها را به 'count_0' و 'count_1' تغییر می‌دهیم تا نشان دهند که تعداد برای 0 و 1 است.
- ستون‌های 'percent_0' و 'percent_1' را اضافه می‌کنیم که درصد 0 و 1 را نشان می‌دهند.

- DataFrame را به فرمت CSV ذخیره می‌کنیم با نام و مسیر داده شده.

- در نهایت، DataFrame خلاصه را برمی‌گردانیم.

خروجی:

- `DataFrame`: pd.DataFrame خلاصه با تعداد و درصدهای تغییرات قیمت بر اساس مقایسه نرخ depression.

مثال استفاده:

```
eur_depression_df = monthly_features(files=['EUR_depression'], path='depression_data/')
usd_depression_df = monthly_features(files=['USD_depression'], path='depression_data/')
price_change_df = combine_investing_data(path='price_data/', files_name=['EUR_USD'])
price_change_df = convert_str_to_float(price_change_df)
price_change_df = count_monthly_price_change(price_change_df)
result_summary = compare_depression_of_2countries(eur_depression_df, usd_depression_df,
price_change_df, 'results/', 'depression_comparison')
```

این کد دو `DataFrame` `eur_depression_df` و `usd_depression_df` را به عنوان ورودی در تابع `compare_depression_of_2countries` می‌فرستد و نتیجه را درمتغیر `result_summary` ذخیره می‌کند.

تابع merge_csv:

تابع `merge_csv` یک لیست از دایرکتوری‌ها که هر یک دو فایل CSV را شامل می‌شوند را به عنوان ورودی دریافت می‌کند و آن‌ها را در یک فایل اکسل (XLS) ترکیب می‌کند و نتیجه را ذخیره می‌کند.

در این تابع:

- ابتدا یک فولدر با نام 'merged_files' در مسیر فعلی ایجاد می‌شود تا فایل‌های ترکیب شده در آن ذخیره شوند.

- سپس برای هر دایرکتوری در لیست `files_list`، مسیر و نام فایل‌های CSV موجود در آن را استخراج می‌کنیم.

- پس از خواندن دو فایل CSV، آن‌ها را با استفاده از تابع `concat` و با محور ستون (`axis=1`) به صورت عمودی ترکیب می‌کنیم و نتیجه را در متغیر `result` ذخیره می‌کنیم.

- سپس DataFrame ترکیب شده را به صورت CSV در مسیر `os.getcwd()+'/feature_analysis/merged_files'` با نام `file` و پسوند '_merged.csv' ذخیره می‌کنیم.

- در نهایت، با استفاده از کتابخانه `xlwt` یک فایل اکسل جدید ایجاد می‌کنیم و برای هر فایل CSV در مسیر `os.getcwd()+'/feature_analysis/merged_files'`، یک ورک‌شیت با نام فایل CSV را ایجاد می‌کنیم.

- سپس با باز کردن هر فایل CSV و خواندن سطرها و ستون‌ها، اطلاعات را در ورک‌شیت متناظر ذخیره می‌کنیم.

- در نهایت، فایل اکسل را با نام `saved_file_name` ذخیره می‌کنیم.

خروجی:

- None

مثال استفاده:

```
directories = ['data_set_1', 'data_set_2']
merge_csv(directories, 'merged_output.xls')
```

در این مثال، دو دایرکتوری 'data_set_1' و 'data_set_2' را به عنوان ورودی در تابع `merge_csv` می‌فرستیم و نتیجه را در فایل 'merged_output.xls' ذخیره می‌کند.

تابع news_effect_with_periods:

تابع `news_effect_with_periods` تحلیل اثر اخبار ماهانه بر یک ویژگی خاص را برای چندین دوره مشخص می‌کند.

در این تابع:

- ابتدا مسیر و نام فایل مربوط به ویژگی مالی مورد نظر که در دایرکتوری `affected_feature_path` قرار دارد را استخراج می‌کنیم.

- سپس فایل ویژگی مالی را بارگیری کرده و به یک DataFrame تبدیل می‌کنیم.

- سپس، تبدیل‌های دیگری روی ویژگی مالی اعمال می‌شود، از جمله تبدیل ماهانه به روزانه.

- مجموعه داده‌های ماهانه را نیز بارگیری می‌کنیم و به یک DataFrame تبدیل می‌کنیم.

- یک لیست با نام ستون‌های جدول را تعریف می‌کنیم و سپس به تعداد دوره‌های مشخص شده پس از هر انتشار خبر، نام ستون‌ها را اضافه می‌کنیم.

- یک DataFrame جدید با نام `new_df` ایجاد می‌کنیم.

- در یک حلقه `for` برای هر ردیف در داده‌های ماهانه:

- یک ردیف جدید به `new_df` اضافه می‌کنیم که شامل اطلاعات خبر ماهانه است.

- سپس برای هر دوره مشخص، مجموعه ای از ردیف‌های مربوطه از ویژگی مالی را استخراج می‌کنیم.

- تغییر در مقدار ویژگی را در هر دوره محاسبه کرده و در متغیر `result` ذخیره می‌کنیم.

- اگر تغییر مقدار بیشتر از صفر باشد، به `new_df` مقدار 1 در ستون مربوط به آن دوره اضافه می‌کنیم، در غیر این صورت مقدار 0 را اضافه می‌کنیم.

- در نهایت، `new_df` را به عنوان خروجی برمی‌گردانیم.

خروجی:

- pd.DataFrame: یک DataFrame که شامل اثر اخبار ماهانه بر ویژگی مشخص شده در چندین دوره است.

مثال استفاده:

```
result_df = news_effect_with_periods(affected_feature_file_name='XAU_USD',
                                     monthly_news_file_name='Trade Balance',
                                     periods=3)
```

در این مثال، تحلیل اثر اخبار 'Trade Balance' بر ویژگی 'XAU_USD' برای 3 دوره انجام می‌شود.

تابع feature_analysis:

تابع `feature_analysis` تحلیل ویژگی‌ها را بر روی فایل‌های ورودی انجام می‌دهد و جدولی حاوی ضریب همبستگی هر ویژگی با هدف را بر اساس تعداد تاخیرها برمی‌گرداند.

در این تابع:

- ابتدا مسیر و نام فایل‌های ویژگی‌ها را استخراج می‌کنیم.

- سپس برای هر فایل ویژگی:

- فایل را بارگیری کرده و به یک DataFrame تبدیل می‌کنیم.

- تبدیل‌های دیگری روی ویژگی اعمال می‌شود، از جمله تبدیل به ویژگی‌های غیرخطی و تبدیل به نرخ بازده.

- ویژگی را با داده هدف ترکیب کرده و نسخه‌های تاخیری از هر ویژگی ایجاد می‌کنیم.

- سپس ضریب همبستگی هر ویژگی با هدف را برای هر تاخیر محاسبه کرده و نتایج را در فایل CSV ذخیره می‌کنیم.

- در نهایت، جدول حاوی ضریب همبستگی هر ویژگی با هدف را برمی‌گردانیم.

خروجی:

- pd.DataFrame: یک DataFrame که شامل ضریب همبستگی هر ویژگی با هدف است، مرتب شده بر اساس ضریب همبستگی به ترتیب نزولی.

مثال استفاده:

```
result_df = feature_analysis(path_features='/path/to/features',
                             files_name=['feature1.csv', 'feature2.csv'],
                             power_number=2,
                             path_target='/path/to/target',
                             target_file_name='target.csv',
                             lags_number=3,
                             path_make_folder='/path/to/save/files')
```

در این مثال، تحلیل ویژگی‌ها بر روی فایل‌های 'feature1.csv' و 'feature2.csv' با تاخیرهای 2 و 3 انجام می‌شود. نتایج تحلیل در فایل‌های CSV در مسیر `path/to/save/files/` ذخیره می‌شوند و جدول حاوی ضریب همبستگی هر ویژگی با هدف برگردانده می‌شود.

تابع comparing:

تابع `comparing` یک ویژگی دودویی (binary) را با قیمت طلا (gold) مقایسه می‌کند و نتایج را بر اساس روزهای هفته تحلیل می‌کند.

در این تابع:

- ابتدا لیستی از نام ویژگی‌ها را استخراج می‌کنیم.

- سپس برای هر ویژگی:

- یک DataFrame جدید تشکیل می‌دهیم که شامل ویژگی، قیمت طلا (gold) و روزهای هفته است.

- سپس سطرهایی که دارای مقادیر نامعتبر هستند را حذف می‌کنیم.

- سپس در ستون‌های متناظر با نتایج محاسبات، مقادیر را محاسبه و ذخیره می‌کنیم.

- در نهایت، نتایج را در فایل CSV جداگانه برای هر ویژگی در دایرکتوری 'compare' ذخیره می‌کنیم.

خروجی:

- pd.DataFrame: یک DataFrame که شامل نتایج مقایسه بین ویژگی دودویی و قیمت طلا است، شامل درصد عدم تطابق کلی و عدم تطابق برای هر روز هفته به ترتیب است.

مثال استفاده:

```
import pandas as pd

merged_df = pd.read_csv('merged_data.csv')

state = 1

gold = 0

results_df = comparing(merged_df, state, gold)
```

در این مثال، تابع `comparing` بر روی DataFrame ادغام شده با نام `merged_df` کار می‌کند و ویژگی دودویی `state` را با قیمت طلا `gold` مقایسه می‌کند. نتایج مقایسه در فایل‌های CSV جداگانه برای هر ویژگی در دایرکتوری 'compare' ذخیره می‌شوند و `results_df` DataFrame شامل نتایج مقایسه برگردانده می‌شود.

تابع count_ones_zeros:

تابع `count_ones_zeros` تعداد و درصد ترکیب‌های دودویی از وضعیت‌ها را در یک DataFrame محاسبه می‌کند.

در این تابع:

- ابتدا یک لیست خالی به نام `data` ایجاد می‌کنیم.

- سپس برای هر ستون در DataFrame:

- تعداد موارد `0_0` را محاسبه می‌کنیم که در آن هر دو ستون (ستون اول و ستون مورد بررسی) مقدار 0 دارند.

- تعداد موارد `1_1` را محاسبه می‌کنیم که در آن هر دو ستون (ستون اول و ستون مورد بررسی) مقدار 1 دارند.

- تعداد موارد `0_1` را محاسبه می‌کنیم که در آن ستون اول مقدار 0 دارد و ستون مورد بررسی مقدار 1 دارد.

- تعداد موارد `1_0` را محاسبه می‌کنیم که در آن ستون اول مقدار 1 دارد و ستون مورد بررسی مقدار 0 دارد.

- درصد `0_0` را بر حسب تعداد موارد `0_0` و تعداد کل سطرها محاسبه می‌کنیم و درصد آن را درصد `percent_0_0` ذخیره می‌کنیم.

- به طریقه‌ی مشابه، درصدهای `1_1`، `0_1` و `1_0` را محاسبه می‌کنیم و ذخیره می‌کنیم.

- سپس یک دیکشنری با نام ستون مورد بررسی و تعداد و درصدهای محاسبه شده را به `data` اضافه می‌کنیم.

- در نهایت، یک DataFrame به نام `percent_df` با استفاده از `data` ایجاد می‌کنیم.

- DataFrame حاوی نتایج را در یک فایل CSV در مسیر مشخص شده ذخیره می‌کنیم.

- همچنین، DataFrame نتایج را نیز برمی‌گردانیم.

خروجی:

- pd.DataFrame: یک DataFrame که شامل تعداد و درصد ترکیب‌های دودویی از وضعیت‌ها است.

مثال استفاده:

```
import pandas as pd

data = {'Column1': [1, 0, 1, 0, 0, 1],
        'Column2': [0, 1, 1, 0, 0, 1],
        'Column3': [1, 1, 0, 0, 1, 1]}

df = pd.DataFrame(data)

result_df = count_ones_zeros(df)
```

در این مثال، تابع `count_ones_zeros` بر روی `df` DataFrame کار می‌کند که شامل ستون‌های وضعیت دودویی است. نتایج به صورت یک DataFrame با تعداد و درصد ترکیب‌های دودویی از وضعیت‌ها را محاسبه می‌کند و در یک فایل CSV ذخیره می‌کند.

تابع find_relation:

تابع `find_relation` تعداد و درصد ترکیب‌های دودویی از وضعیت‌ها را در یک DataFrame محاسبه می‌کند. تفاوت این تابع با تابع قبلی در این است که از تابع `read_investing_data` برای خواندن داده‌ها استفاده می‌کند و یک ستون هدف (`df_target`) را نیز مورد استفاده قرار می‌دهد.

در این تابع:

- ابتدا یک لیست خالی به نام `data` ایجاد می‌کنیم.

- سپس برای هر فایل در `file_name`:

- از تابع `read_investing_data` برای خواندن داده‌ها استفاده می‌کنیم و به `df_feature` می‌دهیم.

- سپس `df_feature` را با `df_target` ادغام کرده و در `merged_df` ذخیره می‌کنیم.

- فقط سطرهایی که حاوی مقادیر NaN نیستند را انتخاب می‌کنیم و در `df` ذخیره می‌کنیم.

- در صورت لزوم (برای lag)، ستون اول را یک واحد به پایین منتقل می‌کنیم.

- تعداد موارد `0_0` را محاسبه می‌کنیم که در آن هر دو ستون (ستون اول و ستون مورد بررسی) مقدار 0 دارند.

- تعداد موارد `1_1` را محاسبه می‌کنیم که در آن هر دو ستون (ستون اول و ستون مورد بررسی) مقدار 1 دارند.

- تعداد موارد `1_0` را محاسبه می‌کنیم که در آن ستون اول مقدار 0 دارد و ستون مورد بررسی مقدار 1 دارد.

- تعداد موارد `0_1` را محاسبه می‌کنیم که در آن ستون اول مقدار 1 دارد و ستون مورد بررسی مقدار 0 دارد.

- درصد `0_0` را بر حسب تعداد موارد `0_0` و تعداد کل سطرها محاسبه می‌کنیم و درصد آن را درصد `percent_0_0` ذخیره می‌کنیم.

- به طریقه‌ی مشابه، درصدهای `1_1`، `1_0` و `0_1` را محاسبه می‌کنیم و ذخیره می‌کنیم.

- سپس یک دیکشنری با نام ستون مورد بررسی و تعداد و درصدهای محاسبه شده را به `data` اضافه می‌کنیم.

- در نهایت، یک DataFrame به نام `percent_df` با استفاده از `data` ایجاد می‌کنیم.

- DataFrame حاوی نتایج را در یک فایل CSV در مسیر مشخص شده ذخیره می‌کنیم.

- همچنین، DataFrame نتایج را نیز باز طریق عبارت `return percent_df` برگردانده می‌شود.