Obed's stock market anaylsis using weather

April 4, 2024

[1]:
```
%%capture
%pip install tqdm seaborn skillsnetwork scikit-learn==0.24
```

[2]:
```
from functools import reduce
from copy import deepcopy
import tqdm
import numpy as np
from scipy.signal import periodogram
from scipy.stats import binomtest
import pandas as pd
import skillsnetwork
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
import seaborn as sns
%matplotlib inline

# Float format for pandas display
pd.set_option('display.float_format', lambda x: '%.8f' % x)

# Suppress unneeded warnings:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

sns.set_context('notebook')
sns.set(style="darkgrid")
```

[3]:
```
# Import weather data
# Note that all of the columns are imported as strings
```

```
# This is generally the safest option, but requires additional processing later␣
↪on

await skillsnetwork.download_dataset(
    'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMSkillsNetwork-GPXX0K1YEN/laguardia.csv'
)
laguardia = pd.read_csv('laguardia.csv', dtype='str')

# Import DOW Jones Industrial Average historical data

await skillsnetwork.download_dataset(
    'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMSkillsNetwork-GPXX0K1YEN/dow_jones.csv'
)
dow = pd.read_csv('dow_jones.csv', dtype='str')
```

```
Downloading laguardia.csv:   0%|          | 0/34039379 [00:00<?, ?it/s]

Saved as 'laguardia.csv'

Downloading dow_jones.csv:   0%|          | 0/1635081 [00:00<?, ?it/s]

Saved as 'dow_jones.csv'
```

[4]:
```
# Weather data
laguardia['DATE'] = pd.to_datetime(laguardia.DATE)
laguardia[['wind',
           'dew_point',
           'temp', 'pressure',
           'cloud_cover']] = laguardia[['wind',
                                        'dew_point',
                                        'temp',
                                        'pressure',
                                        'cloud_cover']].astype(float)

# Market data
dow['DATE'] = pd.to_datetime(dow.DATE)
# Drop missing value rows
dow = dow.loc[dow.Open != '            na']
dow[[i for i in dow.columns if i != 'DATE']] = dow[[i for i in dow.columns if i␣
↪!= 'DATE']].astype(float)
dow['Volume'] = dow.Volume.astype(int)
```

[5]:
```
laguardia = laguardia.loc[:, ['DATE', 'temp', 'cloud_cover']]
dow = dow.loc[:, ['DATE', 'Close']]
```

[6]:
```
# Print the `DATE` field in the `laguardia` dataset:
print("laguardia 'DATE' field head")
print(laguardia.DATE.head())

# The following code shows the hours for which data is available
print("\n laguardia 'DATE' field hour availability")
print(sorted(laguardia.DATE.dt.hour.unique()))

# The following code shows the minutes for which data is available
print("\n laguardia 'DATE' field minute availability")
print(sorted(laguardia.DATE.dt.minute.unique()))
```

```
laguardia 'DATE' field head
0   1948-07-01 11:00:00
1   1948-07-01 12:00:00
2   1948-07-01 13:00:00
3   1948-07-01 14:00:00
4   1948-07-01 15:00:00
Name: DATE, dtype: datetime64[ns]

 laguardia 'DATE' field hour availability
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23]

 laguardia 'DATE' field minute availability
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
```

[7]:
```
# Print the `DATE` field in the `dow` dataset:
print("dow 'DATE' field head")
print(dow.DATE.head())

# The following code shows the hours for which data is available
print("\n dow 'DATE' field hour availability")
print(sorted(dow.DATE.dt.hour.unique()))

# The following code shows the minutes for which data is available
print("\n dow 'DATE' field minute availability")
print(sorted(dow.DATE.dt.minute.unique()))
```

```
dow 'DATE' field head
0   2012-12-31
1   2012-12-28
2   2012-12-27
3   2012-12-26
5   2012-12-24
Name: DATE, dtype: datetime64[ns]
```

```
 dow 'DATE' field hour availability
[0]

 dow 'DATE' field minute availability
[0]
```

[8]:
```
# The following code shows the frequency counts for minutes in `laguardia`:
print("\n laguardia 'DATE' field minute frequency (head):")
print(laguardia.DATE.dt.minute.value_counts().head())
```

```

 laguardia 'DATE' field minute frequency (head):
0     503480
51    235096
59     10846
49      3229
30      1714
Name: DATE, dtype: int64
```

[9]:
```
print("'laguardia' duplicated:")
print(laguardia.DATE.duplicated().value_counts())
```

```
'laguardia' duplicated:
False    799794
True       2145
Name: DATE, dtype: int64
```

[10]:
```
print("'dow' duplicated:")
print(dow.DATE.duplicated().value_counts())
```

```
'dow' duplicated:
False    20962
Name: DATE, dtype: int64
```

[11]:
```
print("'laguardia' missing:")
print(laguardia.isna().max())
```

```
'laguardia' missing:
DATE           False
temp            True
cloud_cover     True
dtype: bool
```

[12]:
```
pd.set_option('display.float_format', lambda x: '%.2f' % x)
print("'laguardia' description:")
print(laguardia.describe())
```

```
'laguardia' description:
            temp  cloud_cover
count 717298.00    646597.00
mean      13.04         0.60
std        9.76         0.39
min      -19.40         0.00
25%        5.60         0.25
50%       13.30         0.75
75%       21.10         1.00
max       39.40         1.00
```

[13]:

```python
# The following resamples all data to an hourly frequency by
# taking an average of all minutes that round to that hour.
laguardia['DATE'] = laguardia['DATE'].dt.round('60min')

# Note that a loop is used to account for the fact that each column contains a
# unique set of missing values:
laguardia_cols = []

for c in laguardia.columns:
    if c == 'DATE':
        continue
    else:
        laguardia_cols.append(
            laguardia[['DATE', c]].dropna().groupby(
                'DATE', as_index=False
            ).agg({c: 'mean'})
        )

# Finally, merge all columns back together again:
laguardia_merged = reduce(
    lambda left, right: pd.merge(left, right, on=['DATE'], how='outer'),
 ↪laguardia_cols
)

# Sort by DATE
laguardia_merged.sort_values('DATE', inplace=True)

# Let's see what the merged data looks like:
laguardia_merged.head()
```

[13]:

	DATE	temp	cloud_cover
0	1948-07-01 11:00:00	22.80	1.00
1	1948-07-01 12:00:00	23.30	0.88
2	1948-07-01 13:00:00	25.00	0.25
3	1948-07-01 14:00:00	26.70	0.00
4	1948-07-01 15:00:00	27.80	0.00

[14]:
```
laguardia_merged.isna().value_counts()
```

[14]:
```
DATE   temp   cloud_cover
False  False  False          569290
              True            44601
       True   False             184
dtype: int64
```

[15]:
```
laguardia_merged[['cloud_cover', 'DATE']].dropna().DATE.diff().value_counts()
```

[15]:
```
0 days 01:00:00    521387
0 days 03:00:00     41070
0 days 02:00:00      6354
0 days 06:00:00       405
0 days 04:00:00        83
0 days 05:00:00        44
0 days 12:00:00        41
0 days 09:00:00        34
0 days 07:00:00        15
0 days 08:00:00        10
0 days 18:00:00         7
1 days 00:00:00         5
0 days 11:00:00         4
0 days 10:00:00         4
0 days 13:00:00         2
0 days 15:00:00         1
0 days 19:00:00         1
1 days 01:00:00         1
0 days 17:00:00         1
0 days 16:00:00         1
1 days 12:00:00         1
0 days 23:00:00         1
2 days 09:00:00         1
Name: DATE, dtype: int64
```

[16]:
```
laguardia_nan_cloud_cover = laguardia_merged.set_index(
    'DATE', drop=True
).sort_index()
laguardia_nan_cloud_cover = laguardia_nan_cloud_cover.reindex(
    pd.date_range(
        start=laguardia_merged.DATE.min(),
        end=laguardia_merged.DATE.max(),
        freq='1H'
    )
)
laguardia_nan_cloud_cover = laguardia_nan_cloud_cover.loc[
    laguardia_nan_cloud_cover.cloud_cover.isna()
```

```python
]
laguardia_nan_cloud_cover['datetime'] = laguardia_nan_cloud_cover.index
laguardia_nan_cloud_cover.datetime.dt.hour.value_counts()
```

[16]:
```
19    5871
20    5859
17    5793
22    5766
23    5746
16    5701
14    5650
13    5607
1     5598
11    5543
2     5508
10    5497
4     5422
7     5420
8     5420
5     5416
12     477
18     458
0      350
6      334
9      258
21     238
15     234
3      198
Name: datetime, dtype: int64
```

[17]:
```python
# This should output just one row if there are no missing hours:
print(laguardia_merged.DATE.diff().value_counts())
```

```
0 days 01:00:00    589778
0 days 03:00:00     23445
0 days 02:00:00       850
1 days 00:00:00         1
Name: DATE, dtype: int64
```

[18]:
```python
# Reindex the dataset to remove missing hours
# First, set the `DATE` column as the index:
laguardia_merged.set_index('DATE', drop=True, inplace=True)
# Now reindex
laguardia_merged = laguardia_merged.reindex(
    pd.date_range(
        start=laguardia_merged.index.min(),
        end=laguardia_merged.index.max(),
```

```
        freq='1H'
    )
)
# Set all data types to float:
laguardia_merged = laguardia_merged.astype(float)

# Interpolate
laguardia_merged.interpolate(method='linear', inplace=True)
laguardia_merged.describe()
```

[18]:

```
             temp  cloud_cover
count  661838.00    661838.00
mean       12.92         0.60
std         9.84         0.38
min       -19.40         0.00
25%         5.00         0.25
50%        13.30         0.75
75%        21.10         1.00
max        39.40         1.00
```

[19]:

```
laguardia_merged.isna().value_counts()
```

[19]:

```
temp   cloud_cover
False  False          661838
dtype: int64
```

[20]:

```
# Get weather variables betweem 8am and 9pm
laguardia_merged_avg = laguardia_merged.between_time('8:00', '9:00').
 ↪reset_index()
laguardia_merged_avg.rename({'index': 'DATE'}, axis=1, inplace=True)
laguardia_merged_avg['DATE'] = laguardia_merged_avg['DATE'].dt.round('1D')
laguardia_merged_avg = laguardia_merged_avg.groupby(
    'DATE', as_index=False
).agg({'temp': 'mean', 'cloud_cover': 'mean'}).set_index('DATE')
rename_dict = dict(
    zip(
        laguardia_merged_avg.columns.tolist(),
        [i + '_avg' for i in laguardia_merged_avg.columns]
    )
)
laguardia_merged_avg.rename(rename_dict, axis=1, inplace=True)
df_weather_final = laguardia_merged_avg
df_weather_final.head()
```

[20]:

```
            temp_avg  cloud_cover_avg
DATE
1948-07-02     18.30             0.00
```

```
1948-07-03     19.40             0.69
1948-07-04     19.40             0.00
1948-07-05     22.50             0.06
1948-07-06     23.05             1.00
```

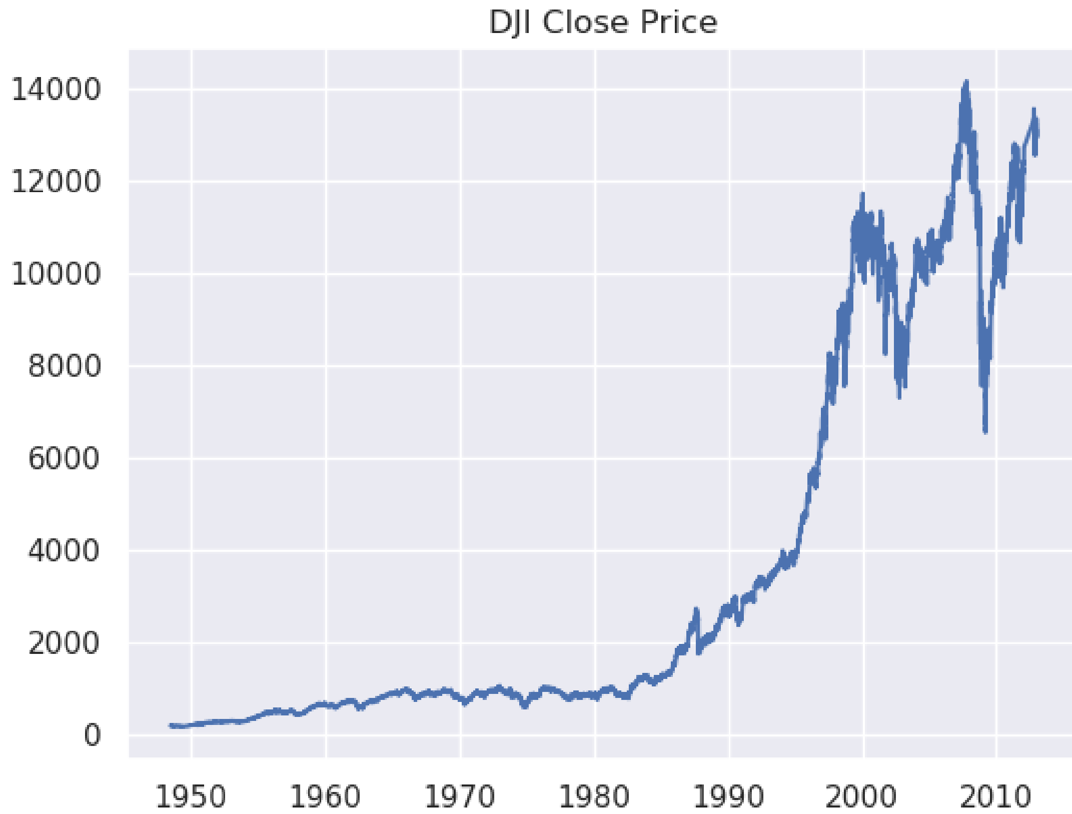
[21]:
```python
# `dow` dataset, gaps between dates (head)
dow.DATE.sort_values().diff().value_counts().head()
```

[21]:
```
1 days    16279
3 days     3880
4 days      464
2 days      325
5 days        8
Name: DATE, dtype: int64
```

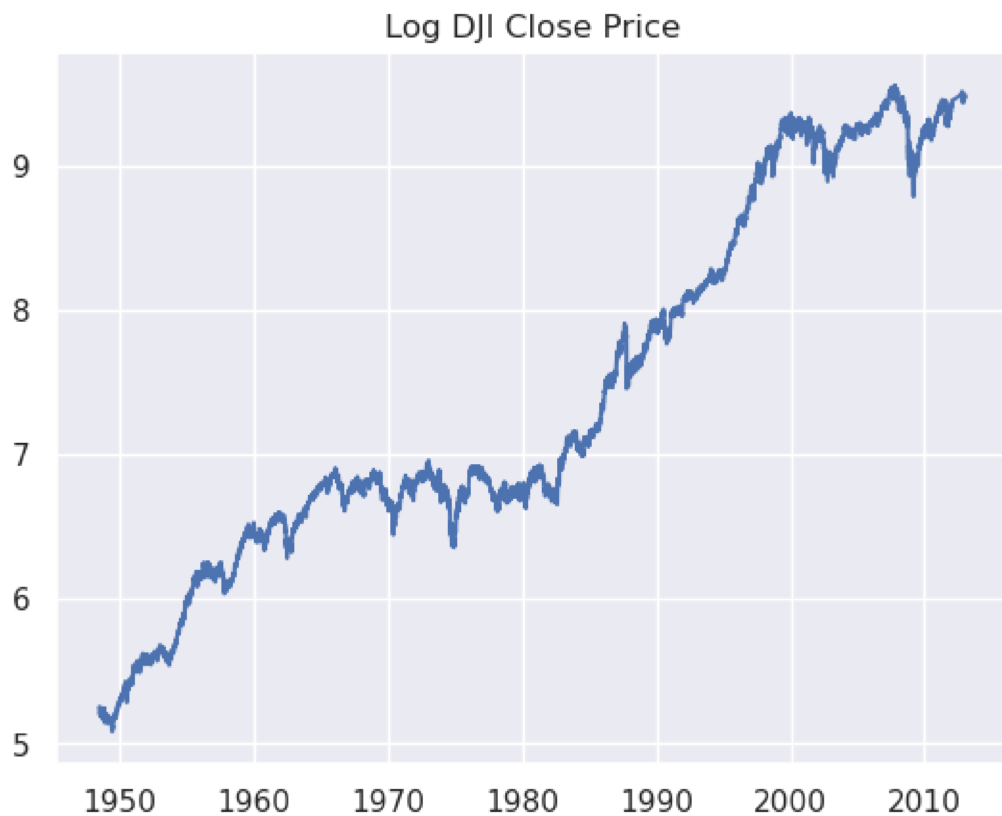
[22]:
```python
dow.sort_values('DATE', inplace=True)
df = dow.merge(df_weather_final,
               how='outer',
               left_on='DATE',
               right_index=True).set_index('DATE').sort_index()
df = df.loc[df.index >= df_weather_final.index[0]]
df.sort_index(inplace=True)
df.head()
```

[22]:
```
             Close  temp_avg  cloud_cover_avg
DATE
1948-07-02 190.06     18.30             0.00
1948-07-03    NaN     19.40             0.69
1948-07-04    NaN     19.40             0.00
1948-07-05    NaN     22.50             0.06
1948-07-06 190.55     23.05             1.00
```

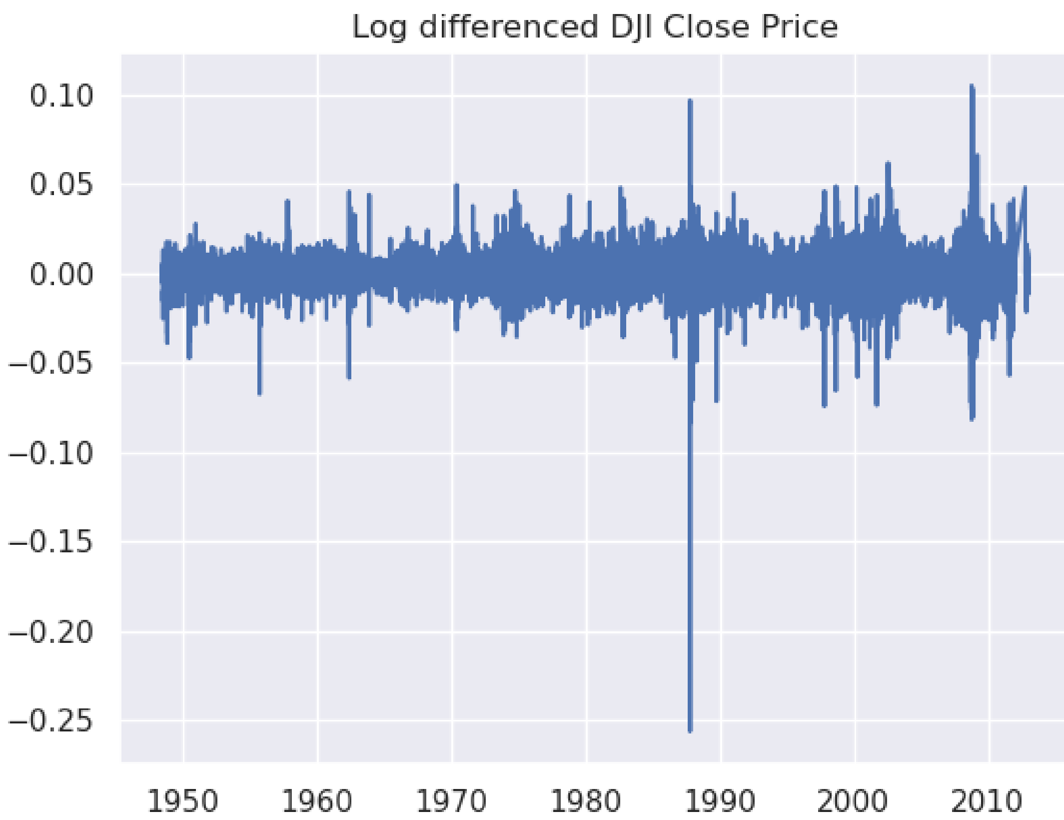
[23]:
```python
_ = sns.lineplot(data=df.Close).set_title('DJI Close Price')
```



[24]:
```
df['log_Close'] = np.log(df.loc[:, 'Close'])
_ = sns.lineplot(data=df.log_Close).set_title('Log DJI Close Price')
```



[25]:
```
log_Close = deepcopy(df.loc[:, 'log_Close'])
log_Close.dropna(inplace=True)
ld_Close = log_Close.diff()
df = df.merge(
    pd.DataFrame(ld_Close).rename({'log_Close':'ld_Close'},axis=1),
    how='left',
    left_index=True,
    right_index=True
)
_ = sns.lineplot(data=df.ld_Close).set_title('Log differenced DJI Close Price')
```



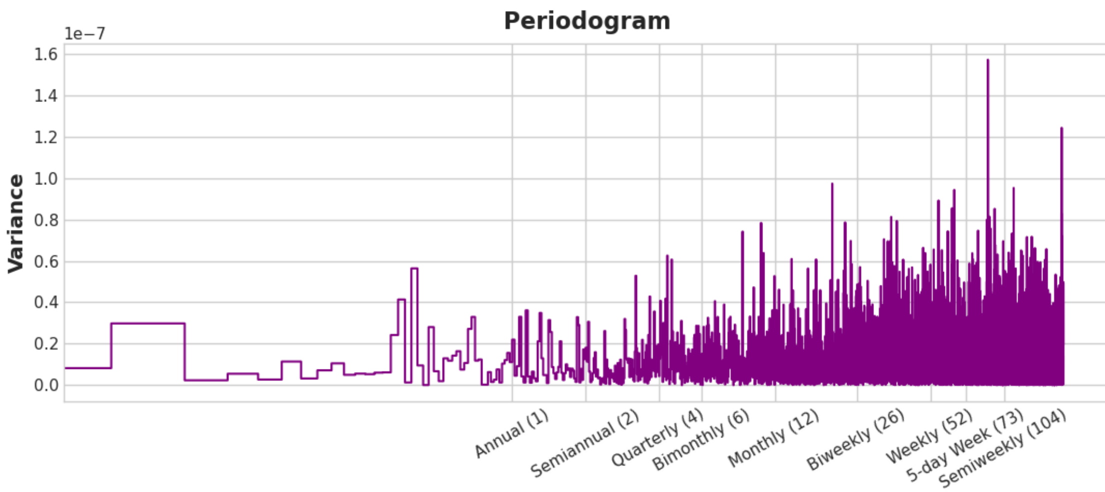
[26]:
```
print('p-value of ADF test:')
print(adfuller(df.ld_Close.dropna())[1])
print('p-value of KPSS test:')
print(kpss(df.ld_Close.dropna())[1])
```

```
p-value of ADF test:
0.0
p-value of KPSS test:
0.1
```

[27]:
```
def plot_periodogram(ts, detrend='linear', ax=None):
    fs = pd.Timedelta("365D6H") / pd.Timedelta("1D")
    freqencies, spectrum = periodogram(
        ts,
        fs=fs,
        detrend=detrend,
        window="boxcar",
        scaling='spectrum',
    )
    if ax is None:
        _, ax = plt.subplots()
```

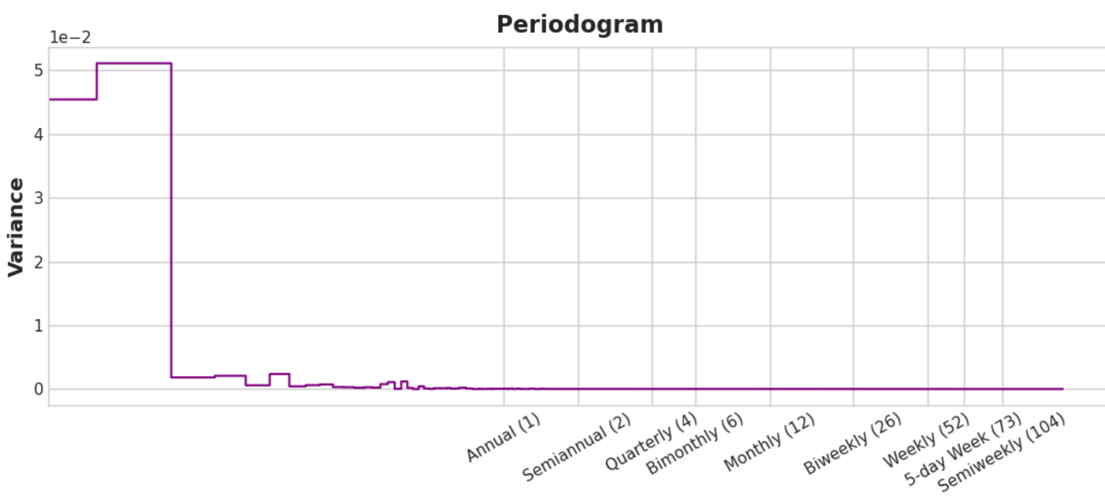
```python
    ax.step(freqencies, spectrum, color="purple")
    ax.set_xscale("log")
    ax.set_xticks([1, 2, 4, 6, 12, 26, 52, 73, 104])
    ax.set_xticklabels(
        [
            "Annual (1)",
            "Semiannual (2)",
            "Quarterly (4)",
            "Bimonthly (6)",
            "Monthly (12)",
            "Biweekly (26)",
            "Weekly (52)",
            "5-day Week (73)",
            "Semiweekly (104)",
        ],
        rotation=30,
    )
    ax.ticklabel_format(axis="y", style="sci", scilimits=(0, 0))
    ax.set_ylabel("Variance")
    ax.set_title("Periodogram")
    return ax

# Set Matplotlib defaults
plt.style.use("seaborn-whitegrid")
plt.rc("figure", autolayout=True, figsize=(11, 5))
plt.rc(
    "axes",
    labelweight="bold",
    labelsize="large",
    titleweight="bold",
    titlesize=16,
    titlepad=10,
)
plot_params = dict(
    color="0.75",
    style=".-",
    markeredgecolor="0.25",
    markerfacecolor="0.25",
    legend=False,
)


plot_periodogram(df.loc[:, 'ld_Close'].dropna())
```

[27]:
```
<AxesSubplot:title={'center':'Periodogram'}, ylabel='Variance'>
```

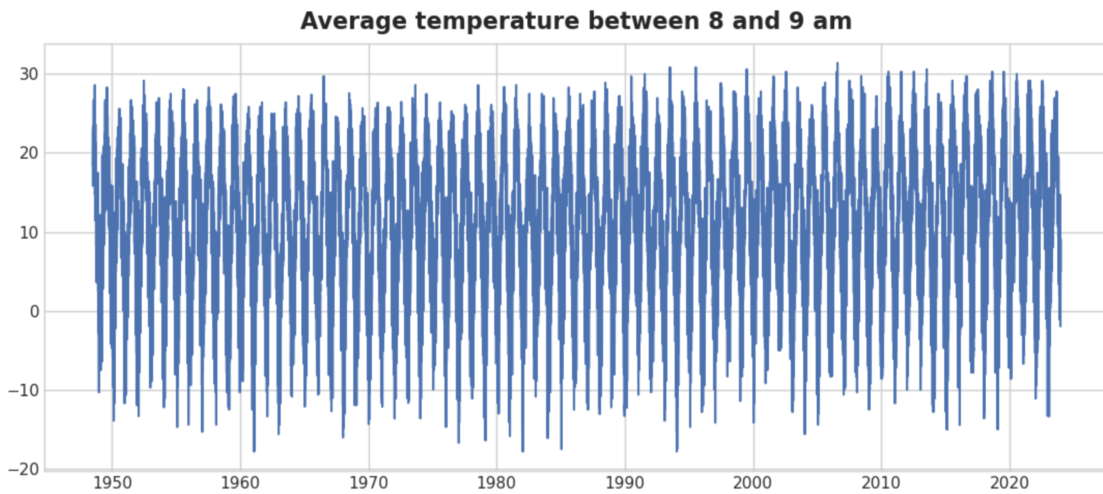


[29]:
```
plot_periodogram(df.loc[:, 'log_Close'].dropna())
```

[29]: <AxesSubplot:title={'center':'Periodogram'}, ylabel='Variance'>

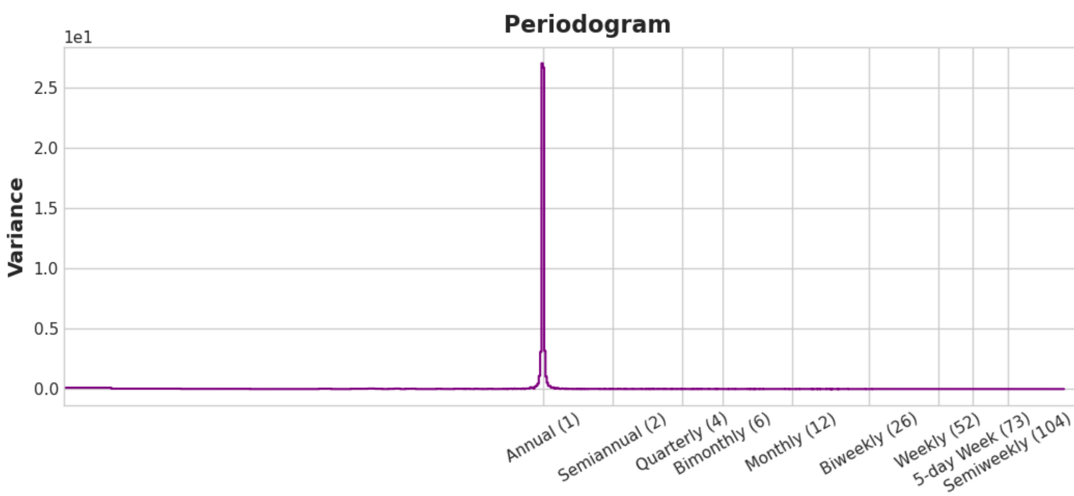


[30]:
```
_ = sns.lineplot(data=df['temp_avg']).set_title('Average temperature between 8␣
↪and 9 am')
```



[31]:
```
plot_periodogram(df.loc[:, 'temp_avg'].dropna())
```

[31]: <AxesSubplot:title={'center':'Periodogram'}, ylabel='Variance'>



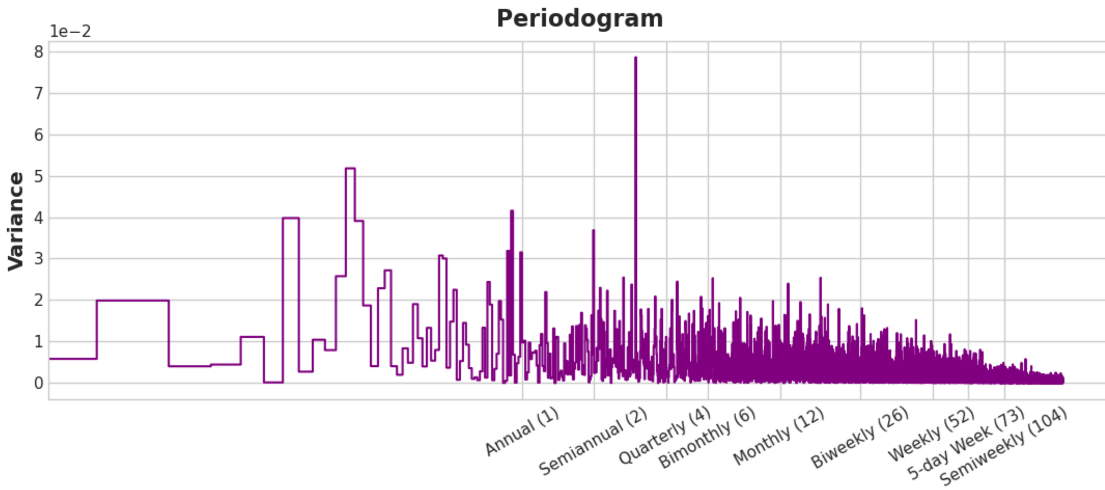
[32]:
```
# Seasonally adjust average temp
y = df.loc[df.index < '1964-01-02', 'temp_avg']
X = [i % 365.25 for i in range(0, len(y.to_numpy()))]
X_full = [i % 365.25 for i in range(0, len(df.temp_avg.to_numpy()))]
degree = 4
coef = np.polyfit(X, y.to_numpy(), degree)
print('Coefficients: %s' % coef)
# create seasonal component
temp_sc_avg = list()
```

```
for i in range(len(X_full)):
    value = coef[-1]
    for d in range(degree):
        value += X_full[i]**(degree-d) * coef[d]
    temp_sc_avg.append(value)

df['temp_sc_avg'] = temp_sc_avg
df['temp_sa'] = df['temp_avg'] - df['temp_sc_avg']
```

```
Coefficients: [-2.27039916e-08  1.89119633e-05 -4.47566270e-03  2.22570025e-01
  1.91433822e+01]
```

[33]:
```
plot_periodogram(df.loc[df.index >= '1964-01-02', 'temp_sa'].dropna())
```

[33]: `<AxesSubplot:title={'center':'Periodogram'}, ylabel='Variance'>`



[34]:
```
print('p-value of ADF test:')
print(adfuller(df.loc[df.index >= '1964-01-02', 'temp_sa'].dropna())[1])
print('p-value of KPSS test:')
print(kpss(df.loc[df.index >= '1964-01-02', 'temp_sa'].dropna())[1])
```

```
p-value of ADF test:
0.0
p-value of KPSS test:
0.01
```

[35]:
```
df['temp_sa_d'] = df['temp_sa'].diff()
print('p-value of ADF test:')
print(adfuller(df.loc[df.index >= '1964-01-02', 'temp_sa_d'].dropna())[1])
print('p-value of KPSS test:')
```

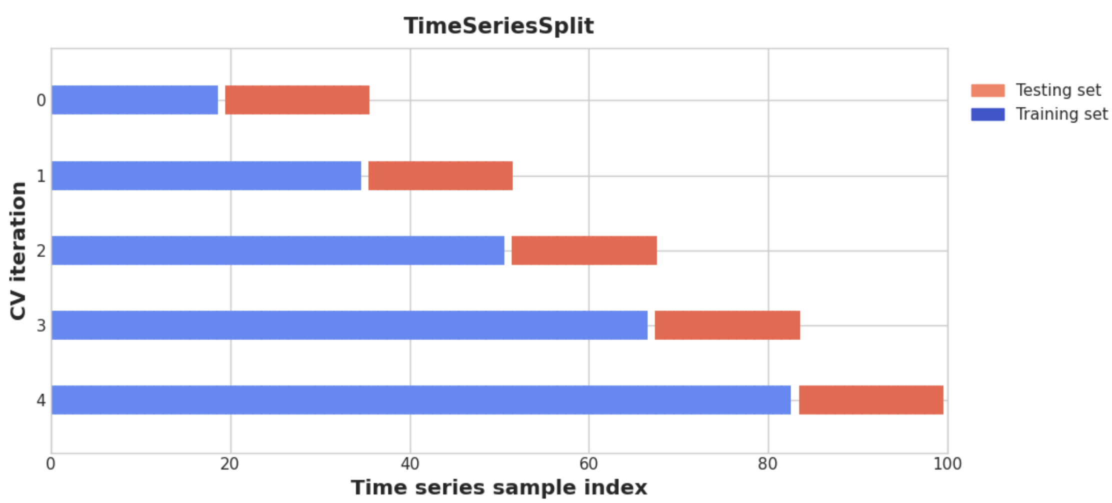
```
print(kpss(df.loc[df.index >= '1964-01-02', 'temp_sa_d'].dropna())[1])
```

```
p-value of ADF test:
0.0
p-value of KPSS test:
0.1
```

[36]:

```
cmap_data = plt.cm.Paired
cmap_cv = plt.cm.coolwarm


def plot_cv_indices(cv, X, y, group, ax, n_splits, lw=20):
    """Create a sample plot for indices of a cross-validation object."""
    # Generate the training/testing visualizations for each CV split
    for ii, (tr, tt) in enumerate(cv.split(X=X, y=y, groups=group)):
        # Fill in indices with the training/test groups
        indices = np.array([np.nan] * len(X))
        indices[tt] = 1
        indices[tr] = 0
        # Visualize the results
        ax.scatter(
            range(len(indices)),
            [ii + 0.5] * len(indices),
            c=indices,
            marker="_",
            s=50,
            lw=lw,
            cmap=cmap_cv,
            vmin=-0.2,
            vmax=1.2,
        )
    # Formatting
    yticklabels = list(range(n_splits))
    ax.set(
        yticks=np.arange(n_splits) + 0.5,
        yticklabels=yticklabels,
        xlabel="Time series sample index",
        ylabel="CV iteration",
        ylim=[n_splits + 0.2, -0.2],
        xlim=[0, 100],
    )
    ax.set_title("{}".format(type(cv).__name__), fontsize=15)
    return ax


fig, ax = plt.subplots()
cv = TimeSeriesSplit(5, gap=1)
```

```
rng = np.random.RandomState(2024)
X = rng.randn(100, 10)
percentiles_classes = [0.33, 0.33, 0.34]
y = np.hstack(
    [[ii] * int(100 * perc) for ii, perc in enumerate(percentiles_classes)]
)
group_prior = rng.dirichlet([2] * 10)
groups = np.repeat(np.arange(10), rng.multinomial(100, group_prior))
plot_cv_indices(cv, X, y, groups, ax, 5)
ax.legend(
        [Patch(color=cmap_cv(0.8)), Patch(color=cmap_cv(0.02))],
        ["Testing set", "Training set"],
        loc=(1.02, 0.8),
)
# Make the legend fit
plt.tight_layout()
fig.subplots_adjust(right=0.7)
```



[37]:
```
# Time series split
tscv = TimeSeriesSplit(n_splits=10, gap=15)
splits = list(tscv.split(df.ld_Close.dropna()))
```

[38]:
```
trues_raw = []
preds_raw = []
results_ols_m = []
for i in tqdm.tqdm_notebook(range(len(splits))):
    mod = sm.OLS(
        df.ld_Close.dropna().to_numpy()[splits[i][0]],
        sm.add_constant(
            df.ld_Close.dropna().to_numpy()[splits[i][0]]
```

```
        )[:, [0]]
    )
    res = mod.fit(disp=False)
    pred = res.predict(
        sm.add_constant(
            df.ld_Close.dropna().to_numpy()[splits[i][1]]
        )[:, [0]]
    )
    preds_raw.append(pred)
    trues_raw.append(df['ld_Close'].dropna().to_numpy()[splits[i][1]])
    results_ols_m.append(res)

trues = np.concatenate(trues_raw)
preds = np.concatenate(preds_raw)
reg_mean_absolute_error = mean_absolute_error(trues, preds)

linreg_mean_mae = []
for i in range(len(trues_raw)):
    linreg_mean_mae.append(mean_absolute_error(trues_raw[i], preds_raw[i]))

print('MAE, regress on constant alone: ' + str(reg_mean_absolute_error))
del mod, res, pred
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

MAE, regress on constant alone: 0.006796108942614544
```

[41]:
```
# ARMA lag order selection using just one fold.
# This code may run for a minute or two.
# Feel free to grab a coffee before continuing!

min_ar_ma = [2,6] # Minimum (p, q)
max_ar_ma = [4,8] # Maximum (p, q)

# Note: according to the AIC criteria, identical AR and MA lags are found if
#       the maximum and minimum bounds are:
#min_ar_ma = [1,1] # Minimum (p, q)
#max_ar_ma = [8,8] # Maximum (p, q)


aic_pd = pd.DataFrame(
    np.empty((max_ar_ma[0]+1-min_ar_ma[0],
              max_ar_ma[1]+1-min_ar_ma[1]),
             dtype=float),
    index=list(range(max_ar_ma[0]+1-min_ar_ma[0])),
    columns=list(range(max_ar_ma[1]+1-min_ar_ma[1]))
)
```

```python
bic_pd = pd.DataFrame(
    np.empty((max_ar_ma[0]+1-min_ar_ma[0],
              max_ar_ma[1]+1-min_ar_ma[1]),
             dtype=float),
    index=list(range(max_ar_ma[0]+1-min_ar_ma[0])),
    columns=list(range(max_ar_ma[1]+1-min_ar_ma[1]))
)

for p in tqdm.tqdm_notebook(range(
        min_ar_ma[0], max_ar_ma[0]+1), position=1, desc='p'):
    for q in range(min_ar_ma[1], max_ar_ma[1]+1):
        if p == 0 and q == 0:
            aic_pd.loc[p, q] = np.nan
            bic_pd.loc[p, q] = np.nan
            continue
        # Estimate the model with no missing datapoints
        mod = sm.tsa.statespace.SARIMAX(
            df['ld_Close'].dropna().iloc[splits[-1][0]],
            order=(p, 0, q),
            trend='c',
            enforce_invertibility=False
        )
        try:
            res = mod.fit(disp=False)
            aic_pd.loc[p, q] = res.aic
            bic_pd.loc[p, q] = res.bic
        except:
            aic_pd.loc[p, q] = np.nan
            bic_pd.loc[p, q] = np.nan

print('AIC: optimal AR order: ' +
      str(aic_pd.min(axis=1).idxmin()) +
      ', optimal MA order: ' +
      str(aic_pd.min().idxmin()))
print('BIC: optimal AR order: ' +
      str(bic_pd.min(axis=1).idxmin()) +
      ', optimal MA order: ' +
      str(bic_pd.min().idxmin()))
```

```
p:   0%|          | 0/3 [00:00<?, ?it/s]
```

```
AIC: optimal AR order: 2, optimal MA order: 8
BIC: optimal AR order: 2, optimal MA order: 7
```

[42]:
```python
trues = []
preds = []
results = []
for i in tqdm.tqdm_notebook(range(len(splits))):
```

```
    mod = sm.tsa.statespace.SARIMAX(
        df['ld_Close'].dropna().to_numpy()[splits[i][0]],
        order=(2, 0, 7),
        trend='c',
        enforce_invertibility=False
    )
    res = mod.fit(disp=False)
    pred = res.predict(
        data=df['ld_Close'].dropna().to_numpy(),
        start=splits[i][1][0],
        end=splits[i][1][-1]
    )
    preds.append(pred)
    trues.append(df['ld_Close'].dropna().to_numpy()[splits[i][1]])
    results.append(res)

trues = np.concatenate(trues)
preds = np.concatenate(preds)

arma_absolute_error = mean_absolute_error(trues, preds)
print('ARMA(2,7) MAE: ' + str(arma_absolute_error))
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```

```
ARMA(2,7) MAE: 0.006796211036333233
```

[46]:

```
trues_rf = []
preds_rf = []
X_orig = df[['ld_Close']].dropna()
features = []
for i in range(1,3):
    features.append(
        df[['ld_Close']].dropna().shift(i).rename(
            {'ld_Close': 'ld_Close_'+str(i)}, axis=1
        )
    )
X = pd.concat(features + [X_orig], axis=1)
y = deepcopy(X_orig[['ld_Close']])
X.drop('ld_Close',axis=1,inplace=True)
for i in tqdm.tqdm_notebook(range(len(splits))):
    regr = RandomForestRegressor(criterion="mae",
                                 n_estimators=10,
                                 max_depth=2,
                                 random_state=2024)
    train_idx = splits[i][0][2:]
    res = regr.fit(X.iloc[train_idx],y.iloc[train_idx])
    pred = regr.predict(X.iloc[splits[i][1]])
    preds_rf.append(pred)
```

```
    trues_rf.append(
        df[['ld_Close']].dropna().ld_Close.to_numpy()[splits[i][1]]
    )

trues_rf = np.concatenate(trues_rf)
preds_rf = np.concatenate(preds_rf)

print(str(mean_absolute_error(trues_rf, preds_rf)))

rf_lags_absolute_error = mean_absolute_error(trues_rf, preds_rf)
print('Random forest with lagged prices MAE: ' +
      str(rf_lags_absolute_error))
del regr, res, pred
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```

```
0.006781133765126767
Random forest with lagged prices MAE: 0.006781133765126767
```

[ ]:

```

```