Electricity Price and Energy Consumption Analysis

Business Problem

**To analyze the impact in electricity prices based on the different types of energy consumption and
their demand in the SE3 region of Sweden.**

Objective

**To perform time-series analysis to forecast energy consumption,
contribution of each energy source to total production and its impact on price changes.
Trend analysis of monthly electricity prices
Yearly Trend Analysis of electricity prices**

Data Collection

1. Web Scraping - NordPool

Scraping the real-time monthly electricity price data and yearly data from the NordPool Market website
where the power trading and day-ahead prices have been fixed for the Baltic and Nordic Countries like Sweden.

2. Svenska Krafnät

Svenska Krafnät Consumption data contains hourly consumption (>50MW), hydroelectric power, wind power, nuclear power,
solar power, and other power production hourly data for the region SE3 of Sweden

Data Set Description

Start Date : Delivery start date from NordPool Market to SE3 region (yyyy-mm-dd)

End Date : Delivery end date from NordPool Market to SE3 region (yyyy-mm-dd)

SE3 (SEK) : SE3 refers to southern sweden which includes Stockholm and Gothenburg,monthly data price

TimePeriod : Timestamp for Hourly time period (yyyy-mm-dd hh:mm:ss)

Hourly Consumption : Hourly energy consumption (>50 MW)

Unspecified Production : Not specified energy production (MWh)

HydroPower Production : Renewable Water Power Production (MWh)

Wind Power Production : Renewable Wind Power Production (MWh)

Nuclear Power Production : Nuclear Power Production (MWh)

Thermal Power Production : Fossil fuel thermal power production (MWh)

Solar Power Production : Renewable Solar Power Production (MWh)

Energy Storage produktion : Stored Energy Production (MWh)

Year : Year from 2019 till 2024

1. Web Scraping - NordPool

In [1]:
```
# importing the necessary libraries
# using selenium lib instead of beautifulsoup to scrape realtime data and to intera
# webdriver acts like a driver for web browser
from selenium import webdriver
from selenium.webdriver.common.by import By
```

```
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import pandas as pd
import seaborn as sns
import time
```

In [2]:
```
# set up Chrome options and disabling cookies popup
chrome_options = Options()
chrome_options.add_argument("--headless")
chrome_options.add_argument("--disable-extensions")
chrome_options.add_argument("--disable-popup-blocking")

# initialize the Selenium WebDriver as the chrome web browser is being controlled b
driver = webdriver.Chrome(options=chrome_options)

def locate_element(driver, locator):
    attempts = 3
    while attempts > 0:
        try:
            return driver.find_element(*locator)
        except Exception as e:
            print(f"Relocating element due to: {e}")
            time.sleep(5)
            attempts -= 1
    raise Exception("Failed to locate element after multiple attempts")

try:
    # open the webpage
    url = 'https://data.nordpoolgroup.com/auction/day-ahead/prices?'\
          'deliveryDate=2024-01-31&currency=SEK&aggregation=MonthlyAggregate&delive
    driver.get(url)

    # wait for the table to load
    WebDriverWait(driver, 40).until(
        EC.presence_of_element_located((By.CLASS_NAME, 'dx-scrollable-container'))
    )

    # locate the scrollable container as the data is loaded dynamically using js
    scrollable_container = locate_element(driver, (By.CLASS_NAME, 'dx-scrollable-co

    # extract and print column names
    header_row = locate_element(driver, (By.CLASS_NAME, 'dx-datagrid-headers'))
    header_cells = header_row.find_elements(By.TAG_NAME, 'td')
    column_names = [cell.text.strip() for cell in header_cells if cell.text.strip()
    print(f"Column Names: {column_names}")

    # scroll and fetch rows based on faced challenges
    last_row_count = 0
    max_scrolls = 10
    scrolls = 0

    while scrolls < max_scrolls:
        # fetch rows and ensure all are loaded
        rows = scrollable_container.find_elements(By.CLASS_NAME, 'dx-data-row')
        if len(rows) > last_row_count:
```

```
                last_row_count = len(rows)
                scrolls = 0
            else:
                scrolls += 1

            # scroll down to load more rows
            driver.execute_script("arguments[0].scrollTop = arguments[0].scrollHeight",
            time.sleep(5)

        # process all loaded rows
        data = []
        for i in range(last_row_count):
            try:
                # locate and interact with the row
                row = locate_element(driver, (By.XPATH, f"(//tr[contains(@class,'dx-dat

                # scroll to the row to ensure it's visible
                driver.execute_script("arguments[0].scrollIntoView(true);", row)
                time.sleep(1)  # Add a short delay to ensure the row is fully loaded

                # get all cells (columns) in the row
                cells = row.find_elements(By.XPATH, './/td')

                # check if there are enough cells to extract
                if len(cells) >= 3:
                    startdate = cells[0].text.strip() if cells[0].text else "N/A"
                    enddate = cells[1].text.strip() if cells[1].text else "N/A"
                    value = cells[2].text.strip() if cells[2].text else "N/A"

                    # handle hidden cells or cells with colspan as the start date is fi
                    if 'dx-hidden-cell' in cells[0].get_attribute('class'):
                        startdate = cells[0].get_attribute('innerHTML').strip()
                    if 'dx-pointer-events-none' in cells[1].get_attribute('class'):
                        enddate = cells[1].get_attribute('innerHTML').strip()

                    data.append([startdate, enddate, value])
                else:
                    print(f"Row {i}: Skipped due to insufficient cells")

            except Exception as e:
                print(f"Error on row {i}: {e}")

finally:
    # close the WebDriver
    driver.quit()
```

```
Column Names: ['Delivery Date Start CET', 'Delivery Date End CET', 'SE3 (SEK)']
```

In [3]:

```
# convert the data into a pandas DataFrame
df = pd.DataFrame(data, columns=["Start Date", "End Date", "SE3 (SEK)"])
```

In [4]:

```
print(df)
```

```
    Start Date    End Date SE3 (SEK)
0   2024-12-01  2024-12-31    582,51
1   2024-11-01  2024-11-30    669,52
2   2024-10-01  2024-10-31    229,73
3   2024-09-01  2024-09-30    164,36
4   2024-08-01  2024-08-31     85,40
5   2024-07-01  2024-07-31    207,18
6   2024-06-01  2024-06-30    272,67
7   2024-05-01  2024-05-31    237,14
8   2024-04-01  2024-04-30    563,14
9   2024-03-01  2024-03-31    594,74
10  2024-02-01  2024-02-29    503,44
11  2024-01-01  2024-01-31    802,95
```

In [5]:
```
df.dtypes
```

Out[5]:
```
Start Date    object
End Date      object
SE3 (SEK)     object
dtype: object
```

In [6]:
```
# convert the Delivery start and end Dates to datetime and SE3 (SEK) to float dtype
df['Start Date'] = pd.to_datetime(df['Start Date'],errors = 'coerce')
df['End Date'] = pd.to_datetime(df['End Date'],errors = 'coerce')
df['SE3 (SEK)'] = df['SE3 (SEK)'].replace({',': '', ' ': ''}, regex=True)
df['SE3 (SEK)'] = (pd.to_numeric(df['SE3 (SEK)'], errors='coerce'))
```

In [7]:
```
df.dtypes
```

Out[7]:
```
Start Date    datetime64[ns]
End Date      datetime64[ns]
SE3 (SEK)              int64
dtype: object
```

In [8]:
```
df
```

Out[8]:

	Start Date	End Date	SE3 (SEK)
0	2024-12-01	2024-12-31	58251
1	2024-11-01	2024-11-30	66952
2	2024-10-01	2024-10-31	22973
3	2024-09-01	2024-09-30	16436
4	2024-08-01	2024-08-31	8540
5	2024-07-01	2024-07-31	20718
6	2024-06-01	2024-06-30	27267
7	2024-05-01	2024-05-31	23714
8	2024-04-01	2024-04-30	56314
9	2024-03-01	2024-03-31	59474
10	2024-02-01	2024-02-29	50344
11	2024-01-01	2024-01-31	80295

2. Svenska Krafnät Consumption data

In [9]:
```
# load the consumption data - Svenska kraftnät
consumption_df = pd.read_excel('energy_consumption.xls')
```

Data Cleaning and Data Preprocessing

In [10]:
```
consumption_df.head()
```

Out[10]:

	tidsperiod	Timmätt förbr exkl. avk.last	Timmätt förbr(>50MW)	Avkopplingsb.last	Timmätta förluster	Ospec.produktion	Va pı
0	2024-01-01 00:00:00	-5654.307921	-688.258409	-46.570925	-271.724827	25.980168	9!
1	2024-01-01 01:00:00	-5636.566433	-706.231609	-45.698261	-283.332135	27.765352	9:
2	2024-01-01 02:00:00	-5573.796854	-717.413409	-46.538559	-272.579932	27.932040	92
3	2024-01-01 03:00:00	-5535.053440	-722.688701	-46.749210	-267.741673	27.522272	9:
4	2024-01-01 04:00:00	-5541.747640	-731.740980	-46.488057	-264.843180	28.502297	9!

In [11]:
```
consumption_df.shape
```

Out[11]: (7320, 14)

In [12]:
```
consumption_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7320 entries, 0 to 7319
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype         
---  ------                        --------------  -----         
 0    tidsperiod                   7320 non-null   datetime64[ns]
 1   Timmätt förbr exkl. avk.last  7320 non-null   float64       
 2   Timmätt förbr(>50MW)          7320 non-null   float64       
 3   Avkopplingsb.last             7320 non-null   float64       
 4   Timmätta förluster            7320 non-null   float64       
 5   Ospec.produktion              7320 non-null   float64       
 6   Vattenkraft produktion        7320 non-null   float64       
 7   Vindkraft produktion          7320 non-null   float64       
 8   Kärnkraft produktion          7320 non-null   float64       
 9   Värmekraft produktion         7320 non-null   float64       
 10  Solkraft produktion           7320 non-null   float64       
 11  Energilager produktion        7320 non-null   float64       
 12  Schablonleverans förbrukning  7320 non-null   float64       
 13  Schablonleverans förluster    7320 non-null   float64       
dtypes: datetime64[ns](1), float64(13)
memory usage: 800.8 KB
```

In [13]:
```
consumption_df.describe()
```

Out[13]:

	Timmätt förbr exkl. avk.last	Timmätt förbr(>50MW)	Avkopplingsb.last	Timmätta förluster	Ospec.produktion	Vattenkr produkt
count	7320.000000	7320.000000	7320.000000	7320.000000	7320.000000	7320.000
mean	-5584.087184	-750.348668	-29.790453	-289.391750	9.950254	1333.736
std	1130.052506	84.176073	18.003755	73.346880	8.638801	317.270
min	-9543.515197	-986.737255	-135.043423	-558.985938	0.127378	486.705
25%	-6275.345760	-811.158695	-33.327144	-332.524901	3.633947	1083.280
50%	-5528.374311	-754.162069	-24.283404	-285.914219	7.265463	1375.884
75%	-4645.086882	-697.087544	-18.471982	-241.776041	14.176709	1605.094
max	-3309.256642	-473.515000	-3.261776	510.343657	100.329072	1957.331

In [14]:
```
consumption_df.columns
```

Out[14]:
```
Index([' tidsperiod', 'Timmätt förbr exkl. avk.last', 'Timmätt förbr(>50MW)',
       'Avkopplingsb.last', 'Timmätta förluster', 'Ospec.produktion ',
       'Vattenkraft produktion ', 'Vindkraft produktion ',
       'Kärnkraft produktion', 'Värmekraft produktion ',
       'Solkraft produktion ', 'Energilager produktion ',
       'Schablonleverans förbrukning', 'Schablonleverans förluster'],
      dtype='object')
```

In [15]:
```
# drop the unnecessary columns
consumption_df.drop(['Timmätt förbr exkl. avk.last','Avkopplingsb.last','Timmätta f
                     'Schablonleverans förbrukning','Schablonleverans förluster'],a
```

In [16]:
```
consumption_df.rename(columns =
                      {' tidsperiod':'TimePeriod','Timmätt förbr(>50MW)':'Hourly Co
                        'Ospec.produktion ':'Unspecified Production','Vattenkraft p
                        'Vindkraft produktion ':'Wind Power Production','Kärnkraft 
                        'Värmekraft produktion ': 'Thermal Power Production','Solkr
                        'Energilager produktion ': 'Energy Storage Production'},inp
```

In [17]:
```
print(consumption_df.head())
```

```
           TimePeriod  Hourly Consumption  Unspecified Production  \
0 2024-01-01 00:00:00         -688.258409               25.980168
1 2024-01-01 01:00:00         -706.231609               27.765352
2 2024-01-01 02:00:00         -717.413409               27.932040
3 2024-01-01 03:00:00         -722.688701               27.522272
4 2024-01-01 04:00:00         -731.740980               28.502297

   HydroPower Production  Wind Power Production  Nuclear Power Production  \
0             951.818289            2061.027236                   6300.99
1             931.126044            2092.080597                   6302.19
2             924.570393            2114.303956                   6301.92
3             933.245079            2170.669133                   6302.21
4             953.262242            2198.205705                   6302.80

   Thermal Power Production  Solar Power Production  Energy Storage Production
0                509.347825                0.439621                        0.0
1                515.332926                0.428982                        0.0
2                517.467926                0.472173                        0.0
3                524.845820                0.440353                        0.0
4                526.772503                0.479922                        0.0
```

In [18]:
```python
print(consumption_df.dtypes)
```

```
TimePeriod                  datetime64[ns]
Hourly Consumption                 float64
Unspecified Production             float64
HydroPower Production              float64
Wind Power Production              float64
Nuclear Power Production           float64
Thermal Power Production           float64
Solar Power Production             float64
Energy Storage Production          float64
dtype: object
```

In [19]:
```python
# frequency of time-series data has been changed to monthly, grouped monthly data a
monthly_consumption = consumption_df.resample('M', on = 'TimePeriod').sum().reset_i
monthly_consumption.dtypes
```

Out[19]:
```
TimePeriod                  datetime64[ns]
Hourly Consumption                 float64
Unspecified Production             float64
HydroPower Production              float64
Wind Power Production              float64
Nuclear Power Production           float64
Thermal Power Production           float64
Solar Power Production             float64
Energy Storage Production          float64
dtype: object
```

Feature scaling

In [20]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [21]:
```
scaling_features = monthly_consumption.drop(columns = ['TimePeriod'])
features = sc.fit_transform(scaling_features)

# convert back to dataframe
mon_consumption = pd.DataFrame(features, columns = scaling_features.columns)
mon_consumption['TimePeriod'] = monthly_consumption['TimePeriod']
mon_consumption.describe()
```

Out[21]:

	Hourly Consumption	Unspecified Production	HydroPower Production	Wind Power Production	Nuclear Power Production	Thermal Pov Product
count	1.000000e+01	1.000000e+01	1.000000e+01	1.000000e+01	1.000000e+01	1.000000e-
mean	-1.221245e-16	1.665335e-16	9.992007e-17	-1.998401e-16	-1.805500e-15	-3.330669e
std	1.054093e+00	1.054093e+00	1.054093e+00	1.054093e+00	1.054093e+00	1.054093e-
min	-1.965703e+00	-7.665087e-01	-1.911650e+00	-1.325800e+00	-1.250203e+00	-1.025026e-
25%	-7.341398e-01	-6.642208e-01	-4.998881e-01	-8.648471e-01	-8.853187e-01	-8.010687e
50%	2.703743e-01	-3.656011e-01	1.780249e-01	8.038220e-02	-2.600285e-01	-5.494328e
75%	5.316635e-01	3.263942e-01	2.676332e-01	5.123673e-01	1.007008e+00	8.504628e
max	1.379359e+00	2.656349e+00	1.581747e+00	2.194181e+00	1.570349e+00	1.591522e-

In [22]:
```
# removing first two rows from pricing data to perform merging
price_df = df.iloc[2:].reset_index(drop= True)
price_df.rename(columns = {'End Date':'TimePeriod'},inplace = True)
price_df.dtypes
```

Out[22]:
```
Start Date    datetime64[ns]
TimePeriod    datetime64[ns]
SE3 (SEK)              int64
dtype: object
```

In [23]:
```
merged_df = pd.merge(price_df, mon_consumption, on = 'TimePeriod')
merged_df
```

Out[23]:

	Start Date	TimePeriod	SE3 (SEK)	Hourly Consumption	Unspecified Production	HydroPower Production	Wind Power Production	Nuclear Power Production	P
0	2024-10-01	2024-10-31	22973	0.549593	0.603380	-0.457563	0.565474	-0.085527	
1	2024-09-01	2024-09-30	16436	0.272391	-0.639972	-0.513997	-1.325800	-1.250203	
2	2024-08-01	2024-08-31	8540	0.268357	0.524824	-1.031260	-0.038827	0.586889	
3	2024-07-01	2024-07-31	20718	1.329406	-0.271769	0.271755	-0.913270	-0.763090	
4	2024-06-01	2024-06-30	27267	1.379359	-0.268895	-1.911650	-0.719580	-1.076295	
5	2024-05-01	2024-05-31	23714	0.477875	-0.672304	0.215029	-1.035094	-0.926062	
6	2024-04-01	2024-04-30	56314	-0.736007	-0.459433	1.581747	0.353048	-0.434530	
7	2024-03-01	2024-03-31	59474	-1.965703	-0.766509	1.449649	0.199592	1.570349	
8	2024-02-01	2024-02-29	50344	-0.728538	-0.705670	0.141021	0.720277	1.231421	
9	2024-01-01	2024-01-31	80295	-0.846734	2.656349	0.255268	2.194181	1.147048	

Checking missing values

In [24]:

```
# as the correlation matrix doesnot show prices, checking for missing values again
merged_df['SE3 (SEK)'].isnull().sum()
```

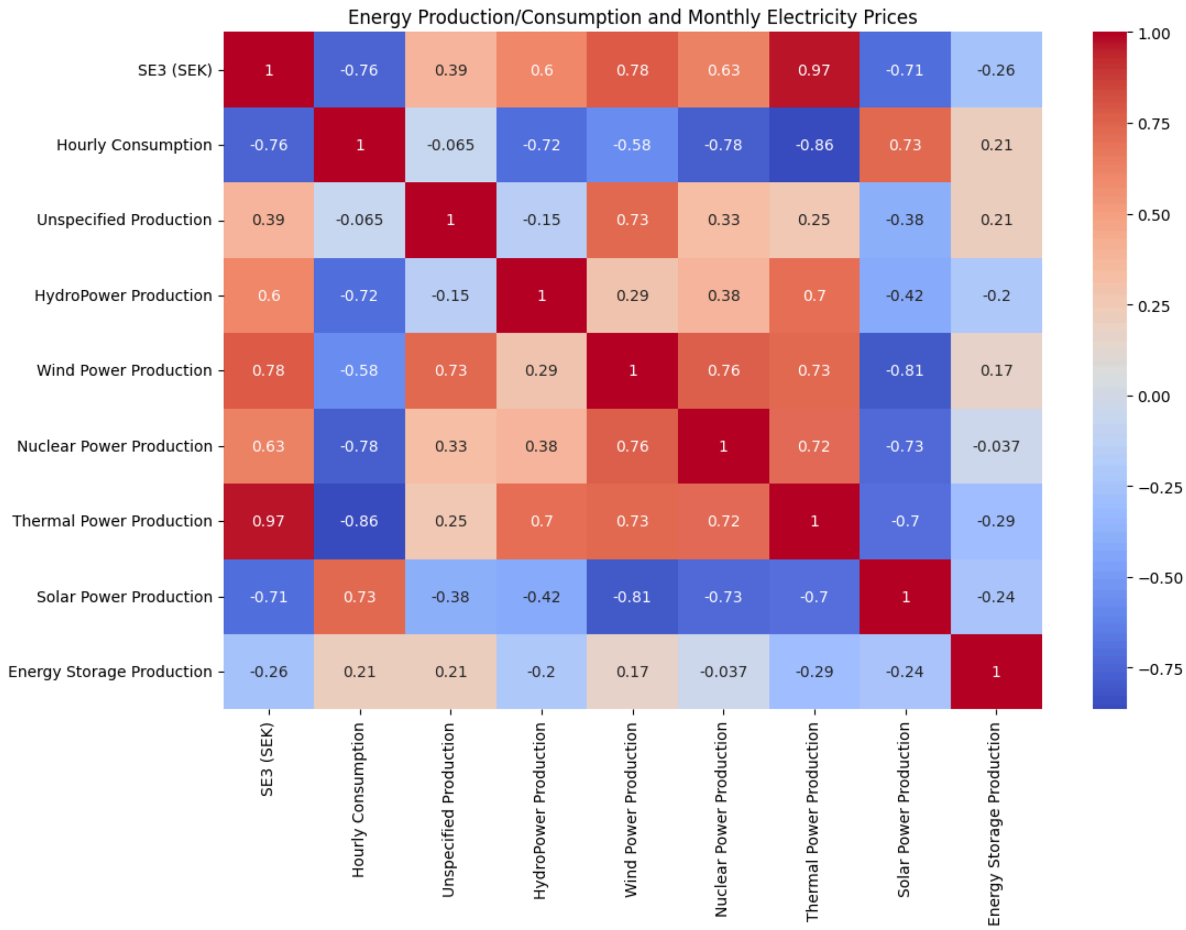
Out[24]: 0

Correlation analysis

In [25]:

```
# calculate correlation matrix
import matplotlib.pyplot as plt
correlation_matrix = merged_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Energy Production/Consumption and Monthly Electricity Prices')
plt.show()
```

```
C:\Users\nklmy\AppData\Local\Temp\ipykernel_24100\4094057435.py:3: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
sion, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  correlation_matrix = merged_df.corr()
```



There is a strong positive correlation between thermal power production and the prices
whereas there is a strong negative correlation between solarpower production and prices
It can impact the demand and supply dynamics of 0.97 and -0.71 respectively

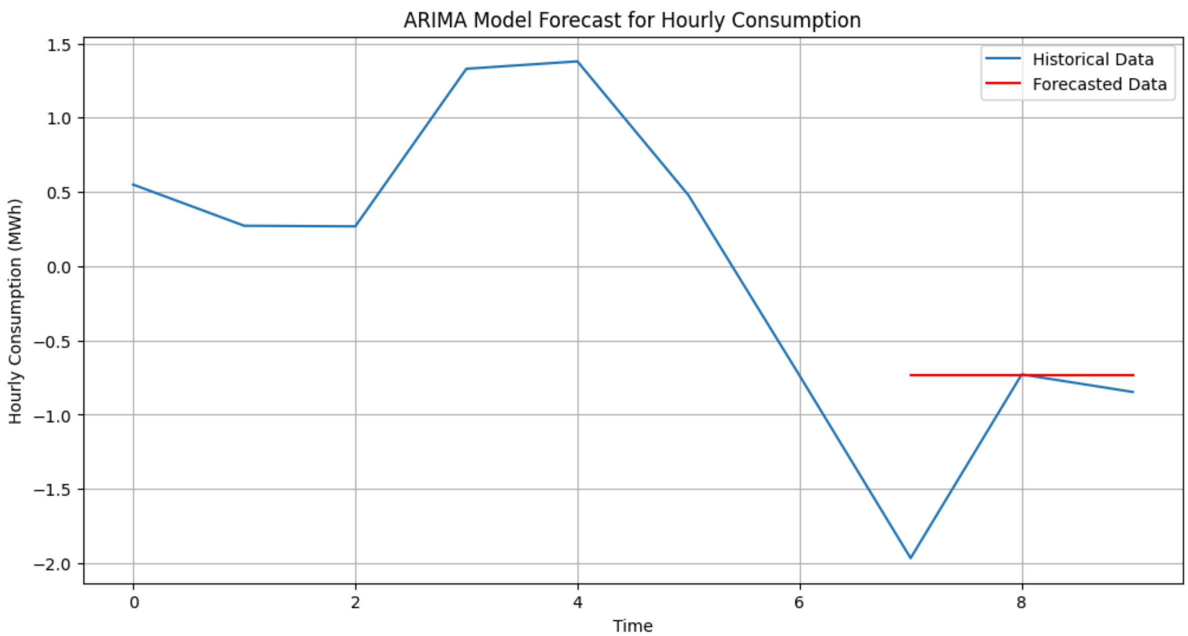
Additionally, hourly consumption and solar power production has a relation of 0.73
It shows that in summer months we can consume more solar power

Forecasting with the Time Series Model ARIMA

Why I used a Chronological Split instead of train_test_split class?
Temporal Dependencies: Time series data points are not independent of each other; they are dependent on the values that came before them. Shuffling would break these dependencies and hence the sequence of observations is essential for making accurate predictions

In [26]:
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np


# drop rows with NaN values
merged_df.dropna(subset=['Hourly Consumption'], inplace=True)

train_size = int(len(merged_df) * 0.7)
train, test = merged_df[:train_size], merged_df[train_size:]


# fit the ARIMA model
model = ARIMA(train['Hourly Consumption'], order=(3, 1, 0))

model_fit = model.fit()

# forecast future consumption
forecast_steps = len(test)
predictions = model_fit.forecast(steps=forecast_steps)

# ensure the predictions are properly assigned to test DataFrame
test['Predictions'] = predictions

# calculate the mean squared error
mse = mean_squared_error(test['Hourly Consumption'], test['Predictions'])
print(f'Mean Squared Error: {mse : .2f}')

rmse = np.sqrt(mse)
print(f'Root Mean Squared Error (RMSE): {rmse: .2f}')


# combine train, test, and predictions for visualization
combined = pd.concat([train, test], axis=0)

# plot for visualizing test data and forecast
plt.figure(figsize=(12, 6))
sns.lineplot(data=combined, x=combined.index, y='Hourly Consumption', label='Histor
sns.lineplot(data=test, x=test.index, y='Predictions', label='Forecasted Data', col
plt.title('ARIMA Model Forecast for Hourly Consumption')
plt.xlabel('Time')
plt.ylabel('Hourly Consumption (MWh)')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\nklmy\AppData\Local\Programs\Python\Python311\Lib\site-packages\statsmode
ls\tsa\statespace\sarimax.py:966: UserWarning: Non-stationary starting autoregress
ive parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
C:\Users\nklmy\AppData\Local\Temp\ipykernel_24100\3382586801.py:24: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['Predictions'] = predictions
Mean Squared Error:  0.51
Root Mean Squared Error (RMSE):  0.71
```

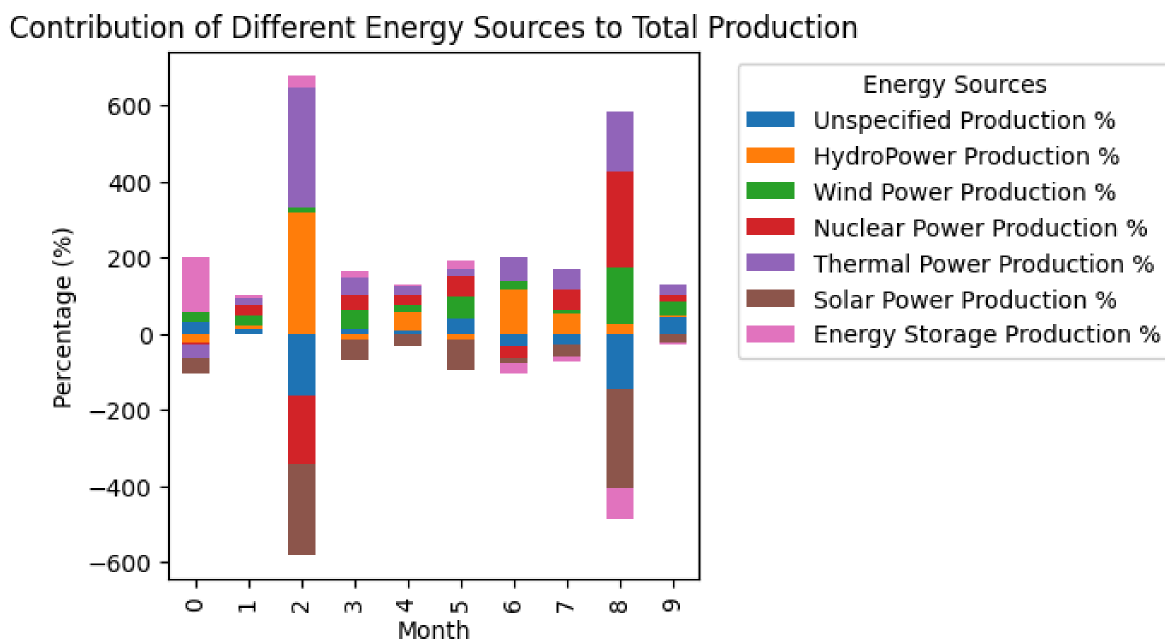


Analysis of contribution of each energy source to total production

In [27]:

```python
# list of all energy production columns
production_columns = ['Unspecified Production', 'HydroPower Production',
                      'Wind Power Production', 'Nuclear Power Production',
                      'Thermal Power Production', 'Solar Power Production', 'Energy

# create a new column 'Total Production' by summing the production columns
merged_df['Total Production'] = merged_df[production_columns].sum(axis=1)

# calculate the percentage contribution of each energy source
for column in production_columns:
    merged_df[f'{column} %'] = (merged_df[column] / merged_df['Total Production'])

# filter columns that contain '%' in their name
percentage_columns = [col for col in merged_df.columns if '%' in col]

# plot the contributions of each energy source
merged_df[percentage_columns].plot(kind='bar', stacked=True, figsize=(4, 4))
```

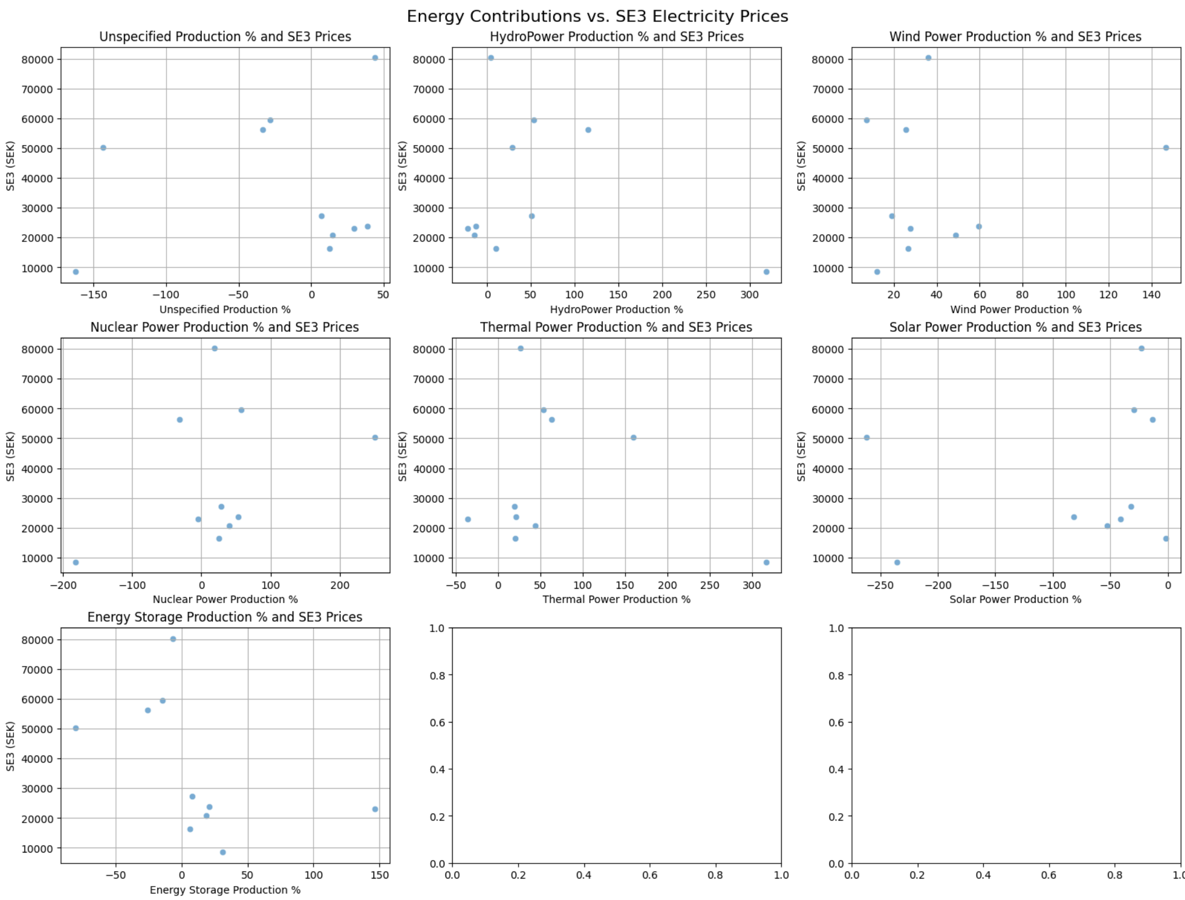
```
plt.title('Contribution of Different Energy Sources to Total Production')
plt.xlabel('Month')
plt.ylabel('Percentage (%)')

# moving the legend outside the plot area to avoid overlapping
plt.legend(title="Energy Sources", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



Analysis of the contribution of each energy source to total production and its impact on prices

In [28]:
```
# Create a grid of scatter plots for energy contributions Vs SE3 electricity prices

num_columns = len(percentage_columns)

# to determine grid size
grid_size = int(np.ceil(np.sqrt(num_columns)))
fig, axes = plt.subplots(grid_size, grid_size, figsize=(16, 12), constrained_layout

# flatten the axes for easy iteration to reduce the dimensionality of multiple subp
axes = axes.flatten()

for i, column in enumerate(percentage_columns):
    sns.scatterplot(
        ax=axes[i],
        data=merged_df,
        x=column,
        y='SE3 (SEK)',
        alpha=0.6
    )
    axes[i].set_title(f'{column} and SE3 Prices')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('SE3 (SEK)')
```

```
    axes[i].grid(True)

plt.suptitle('Energy Contributions vs. SE3 Electricity Prices', fontsize=16)
plt.show()
```



Renewable energy like Hydropower and Solar power generally show a weaker positive relationship with lower prices,
higher renewable energy usage tends to stabilize or lower prices in the SE3 region.
Increase in renewable energy production with less transmission may reduce prices for large consumers

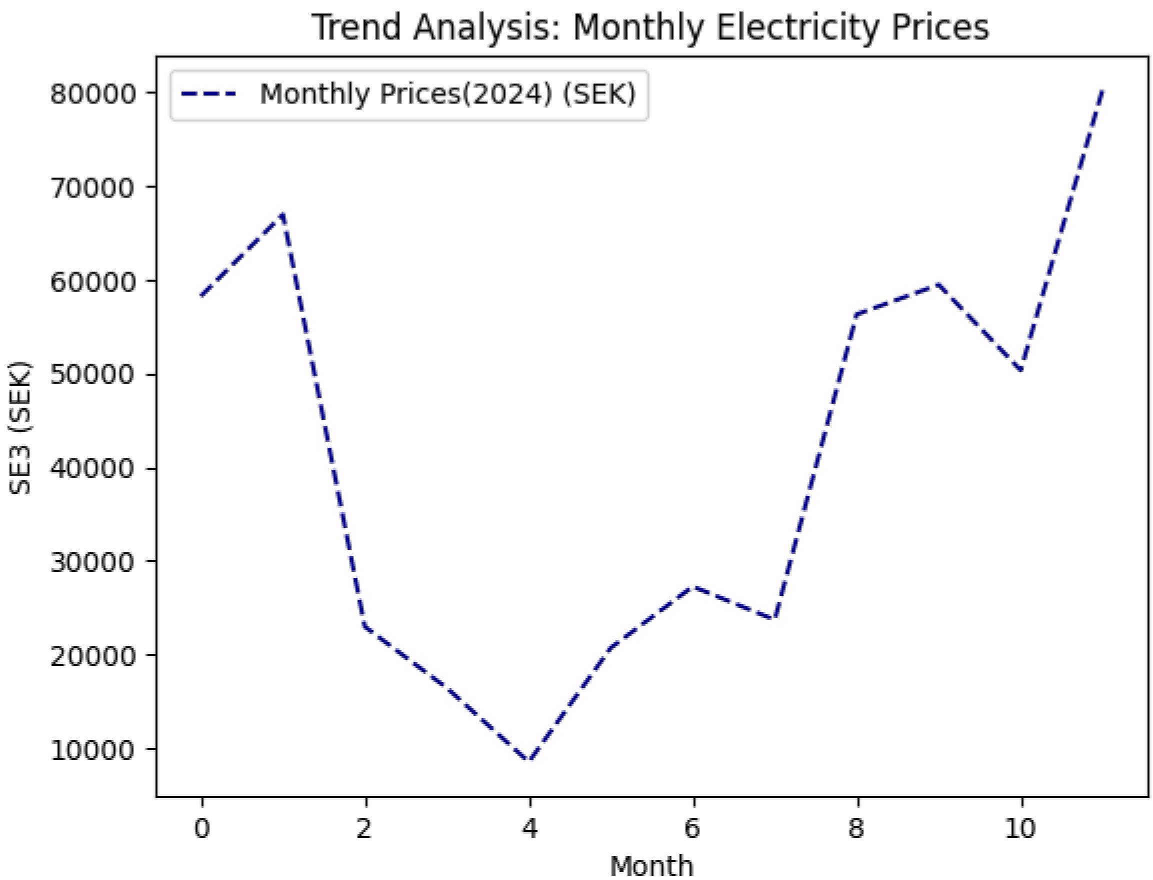
Fossile fuels energy like Nuclear and Thermal power have a direct relationship with prices,
when contributes higher, prices will increase

Wind power contribution in prices is quite unpredictable
Other factors like transporting/transmitting the energy from other regions, government standards also influences the prices

Trend analysis

In [32]:
```
# Plotting the monthly prices
sns.lineplot(x=df.index, y=df['SE3 (SEK)'], label='Monthly Prices(2024) (SEK)', lin
```

```
plt.title('Trend Analysis: Monthly Electricity Prices')
plt.xlabel('Month')
plt.ylabel('SE3 (SEK)')
plt.legend(loc='upper left')

plt.show()
```

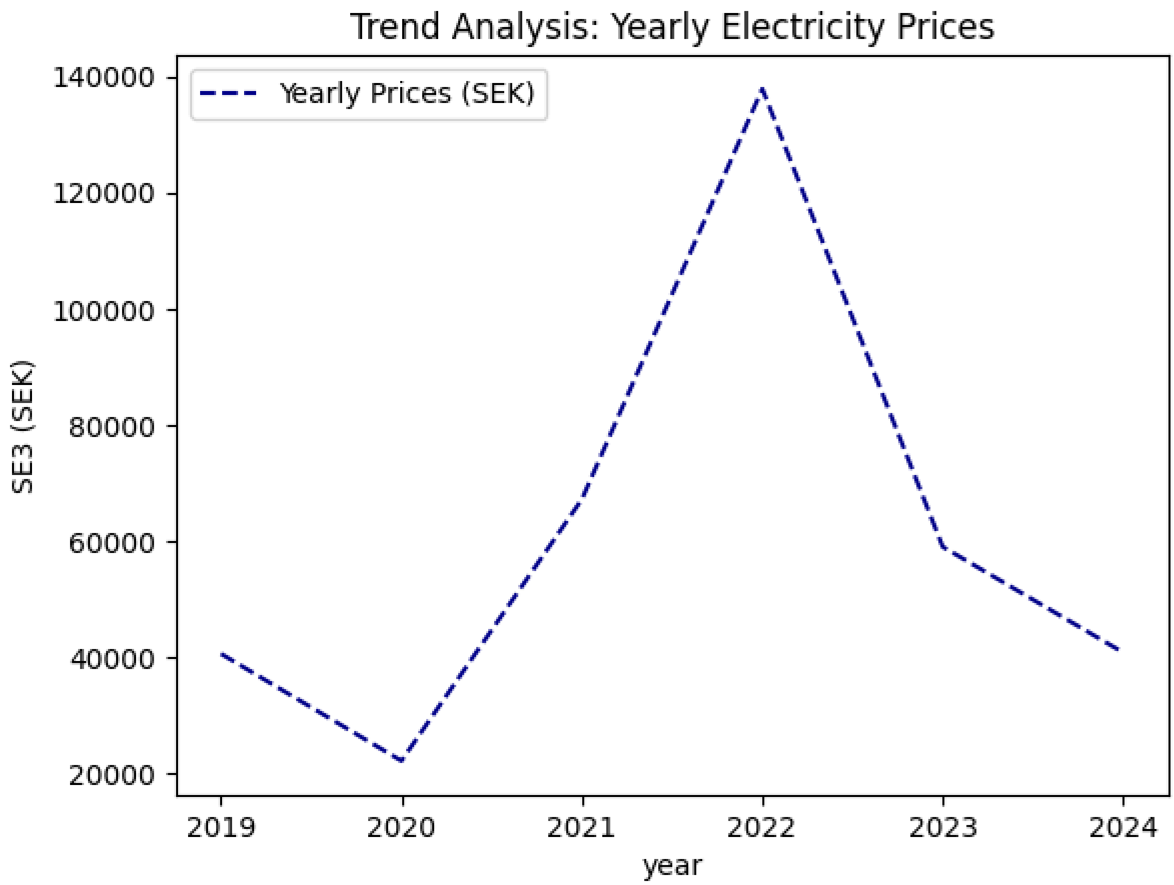


Seasonality increase in prices from the end of autumn till the end of winter as the consumption increases

Yearly Trend analysis

In [30]:
```
# scraped seperately and performed data cleaning
year_df = pd.read_csv('yearly_df.csv')
```

In [31]:
```
# plotting the monthly prices

sns.lineplot(x=year_df['Year'], y=year_df['SE3 (SEK)'], label='Yearly Prices (SEK)'

plt.title('Trend Analysis: Yearly Electricity Prices')
plt.xlabel('year')
plt.ylabel('SE3 (SEK)')
plt.legend(loc='upper left')

plt.show()
```



Electricity prices in the SE3 region show a downward trend as of the 12/31/2024 data, with an exceptional increase in 2022 due to the COVID crisis.
Based on the ablove analysis and its incorporation with this data reflects a balanced ratio of demand and supply,
as well as increase in production to balance the energy consumption in SE3 region of Sweden