In [1]:
```python
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import sklearn as skl
import datetime as dt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, bds
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pmdarima.arima import auto_arima
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

Tratamento inicial dos dados

In [2]:
```python
data = pd.read_csv('Month_Value_1.csv')
data # Period mal formatado
     # a partir de 1/05/2020 os valores não existem
```

Out[2]:

	Period	Revenue	Sales_quantity	Average_cost	The_average_annual_payroll_of_the_region
0	01.01.2015	1.601007e+07	12729.0	1257.763541	30024676.0
1	01.02.2015	1.580759e+07	11636.0	1358.507000	30024676.0
2	01.03.2015	2.204715e+07	15922.0	1384.697024	30024676.0
3	01.04.2015	1.881458e+07	15227.0	1235.606705	30024676.0
4	01.05.2015	1.402148e+07	8620.0	1626.621765	30024676.0
...	...	...	...	...	...
91	01.08.2022	NaN	NaN	NaN	NaN
92	01.09.2022	NaN	NaN	NaN	NaN
93	01.10.2022	NaN	NaN	NaN	NaN
94	01.11.2022	NaN	NaN	NaN	NaN
95	01.12.2022	NaN	NaN	NaN	NaN

96 rows × 5 columns

Period está mal formatado, devemos transformar para formato de data e, posteriormente, utilizá-lo como index.

A partir de 01/05/2020 os valores não existem. Vamos apagá-los.

Além disso, vamos alterar o nome das variáveis.

In [3]:
```python
# renomeando as variáveis
data = data.rename(columns = {'Period': 'Period',
                              'Revenue': 'Revenue',
                              'Sales_quantity': 'Sales',
                              'Average_cost': 'Average_cost',
                              'The_average_annual_payroll_of_the_region': 'Average_payroll'})
data
```

Out[3]:

	Period	Revenue	Sales	Average_cost	Average_payroll
0	01.01.2015	1.601007e+07	12729.0	1257.763541	30024676.0
1	01.02.2015	1.580759e+07	11636.0	1358.507000	30024676.0
2	01.03.2015	2.204715e+07	15922.0	1384.697024	30024676.0
3	01.04.2015	1.881458e+07	15227.0	1235.606705	30024676.0
4	01.05.2015	1.402148e+07	8620.0	1626.621765	30024676.0
...	...	...	...	...	...
91	01.08.2022	NaN	NaN	NaN	NaN
92	01.09.2022	NaN	NaN	NaN	NaN
93	01.10.2022	NaN	NaN	NaN	NaN
94	01.11.2022	NaN	NaN	NaN	NaN
95	01.12.2022	NaN	NaN	NaN	NaN

96 rows × 5 columns

In [4]:
```python
data['Period'] = pd.to_datetime(data['Period'], format = '%d.%m.%Y') # transformando Period em index
data.set_index('Period', inplace = True)
data
```

Out[4]:

	Revenue	Sales	Average_cost	Average_payroll
Period				
2015-01-01	1.601007e+07	12729.0	1257.763541	30024676.0
2015-02-01	1.580759e+07	11636.0	1358.507000	30024676.0
2015-03-01	2.204715e+07	15922.0	1384.697024	30024676.0
2015-04-01	1.881458e+07	15227.0	1235.606705	30024676.0
2015-05-01	1.402148e+07	8620.0	1626.621765	30024676.0
...	...	...	...	...
2022-08-01	NaN	NaN	NaN	NaN
2022-09-01	NaN	NaN	NaN	NaN
2022-10-01	NaN	NaN	NaN	NaN
2022-11-01	NaN	NaN	NaN	NaN
2022-12-01	NaN	NaN	NaN	NaN

96 rows × 4 columns

In [5]:
```python
data = data[data.isna().any(axis = 1) == False] # apagando os dados inexistentes
data
```

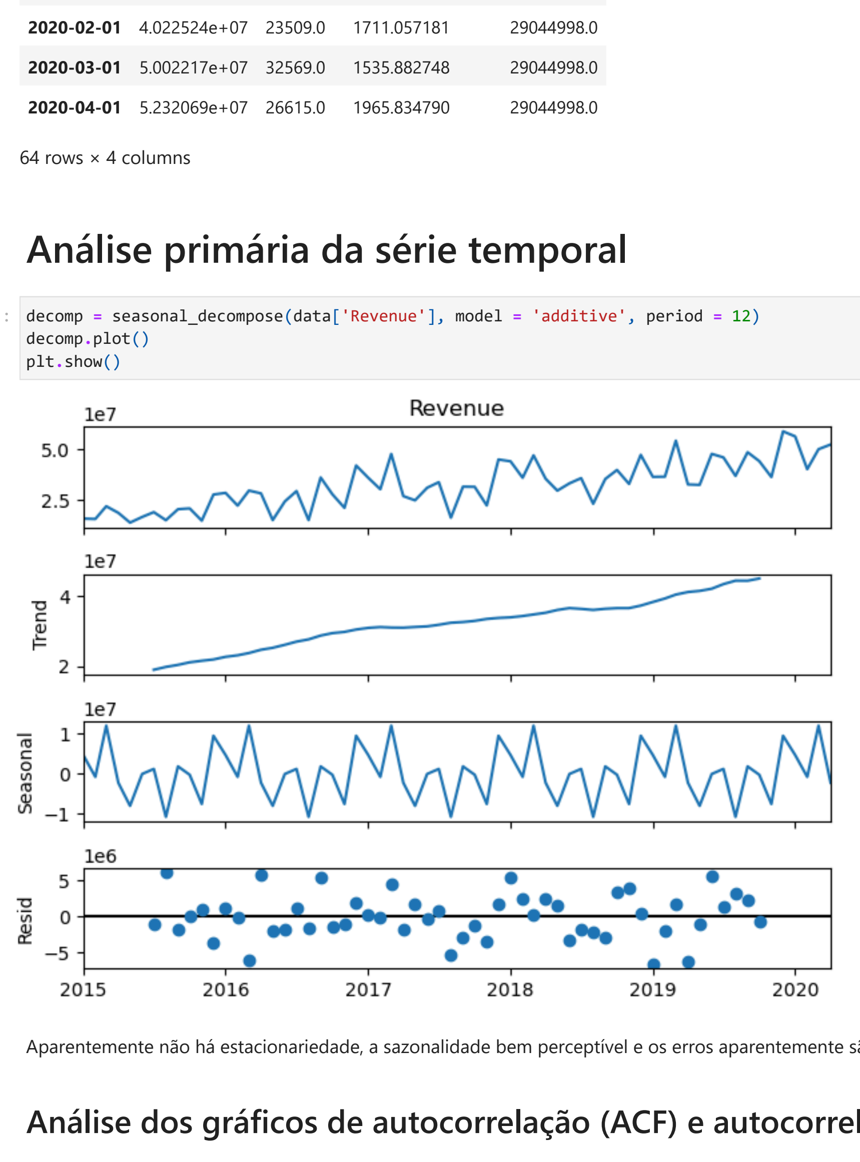
Out[5]:

	Revenue	Sales	Average_cost	Average_payroll
Period				
2015-01-01	1.601007e+07	12729.0	1257.763541	30024676.0
2015-02-01	1.580759e+07	11636.0	1358.507000	30024676.0
2015-03-01	2.204715e+07	15922.0	1384.697024	30024676.0
2015-04-01	1.881458e+07	15227.0	1235.606705	30024676.0
2015-05-01	1.402148e+07	8620.0	1626.621765	30024676.0
...	...	...	...	...
2019-12-01	5.875647e+07	38069.0	1543.420464	29878525.0
2020-01-01	5.628830e+07	27184.0	2070.640850	29044998.0
2020-02-01	4.022524e+07	23509.0	1711.057181	29044998.0
2020-03-01	5.002217e+07	32569.0	1535.882748	29044998.0
2020-04-01	5.232069e+07	26615.0	1965.834790	29044998.0

64 rows × 4 columns

Análise primária da série temporal

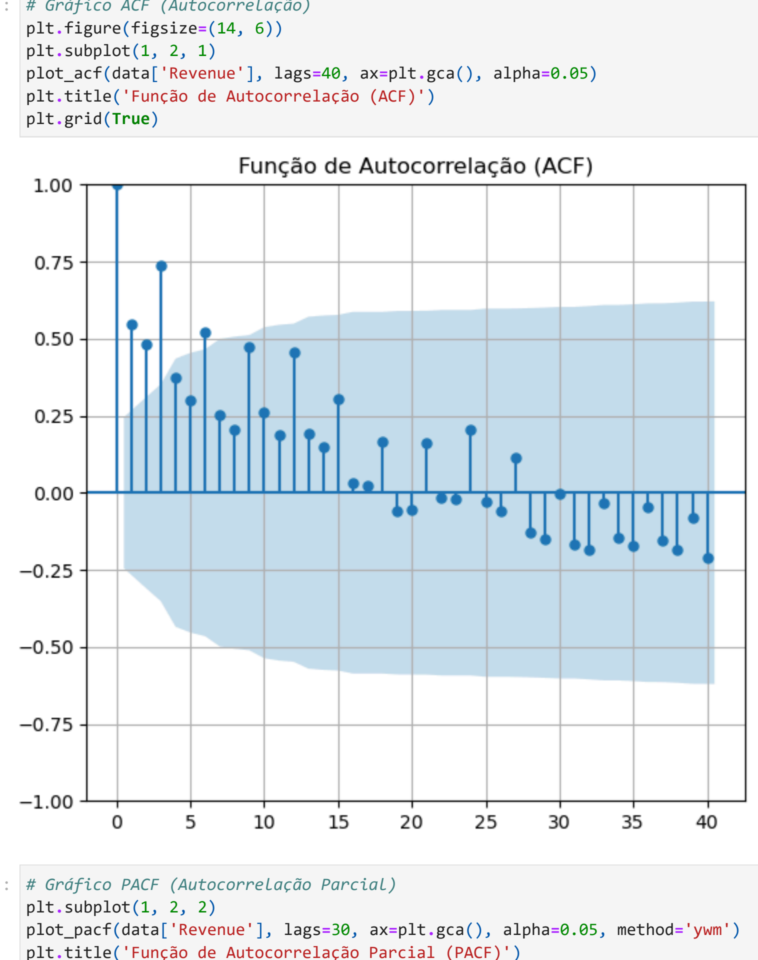
In [6]:
```python
decomp = seasonal_decompose(data['Revenue'], model = 'additive', period = 12)
decomp.plot()
plt.show()
```



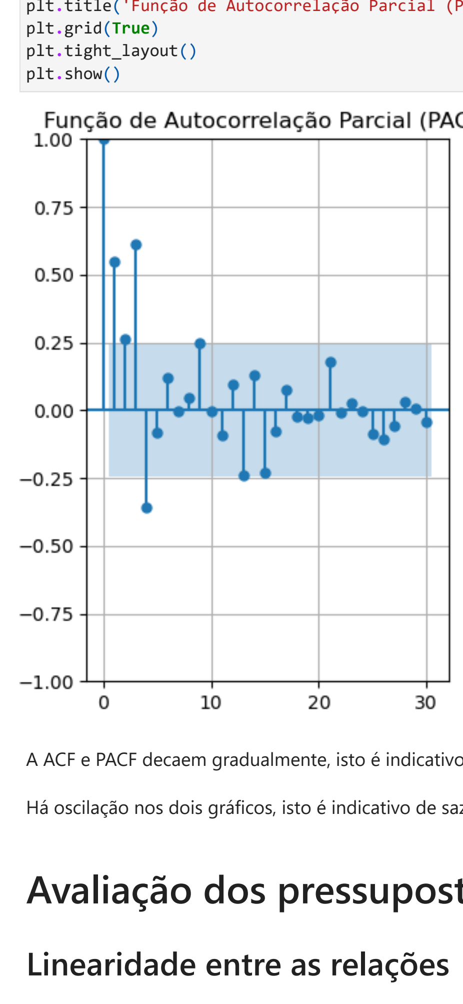
Aparentemente não há estacionariedade, a sazonalidade bem perceptível e os erros aparentemente são homoscedásticos.

Análise dos gráficos de autocorrelação (ACF) e autocorrelação parcial (PACF)

In [7]:
```python
# Gráfico ACF (Autocorrelação)
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plot_acf(data['Revenue'], lags=40, ax=plt.gca(), alpha=0.05)
plt.title('Função de Autocorrelação (ACF)')
plt.grid(True)
```



In [8]:
```python
# Gráfico PACF (Autocorrelação Parcial)
plt.subplot(1, 2, 2)
plot_pacf(data['Revenue'], lags=30, ax=plt.gca(), alpha=0.05, method='ywm')
plt.title('Função de Autocorrelação Parcial (PACF)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



A ACF e PACF decaem gradualmente, isto é indicativo de tendência e linearidade.

Há oscilação nos dois gráficos, isto é indicativo de sazonalidade.

Avaliação dos pressupostos iniciais de modelagem SARIMA

Linearidade entre as relações

Usaremos o teste BDS para testar a linearidade. Antes, faremos a diferença para deixar a série estacionária.

In [9]:
```python
data_diff = data['Revenue'].diff(1).dropna()
print('P-valor:', adfuller(data_diff)[1])
```

```
P-valor: 1.244137799553809e-06
```

Com 1 diferença já obtivemos a série estacionária.

As hipóteses são:

H0: a série é linear entre as observações passadas e futuras(os resíduos são iid)

H1: a série não é linear entre as observações passadas e futuras(os resíduos não são iid)

In [10]:
```python
print('Estatística: ', bds(data_diff)[0])
print('P-valor: ', bds(data_diff)[1])
```

```
Estatística:  1.41867946387586O2
P-valor:  0.15599248699934637
```

Não rejeitamos H0, portanto a série é linear.

Criação do modelo preditivo

Separação das bases de treino e de testes

O dataset de teste será o menor intervalo de períodos da base de dados total que deixa os 10% mais recentes da quantidade total de períodos.

Infelizmente a base de teste será pequena, pois o número amostral é pequeno.

In [11]:
```python
n_test = round(len(data['Revenue'])*0.1) + 1 # n_test = 7
train = data[:-n_test] # os primeiros len(data) - 7 períodos
test = data[-n_test:] # os últimos 7 períodos
```

Criação do modelo

In [12]:
```python
model = auto_arima(train['Revenue'],
                   seasonal = True,
                   m = 12,
                   stepwise = True)
```

O melhor modelo é o SARIMAX(0, 1, 0, 12)

In [13]:
```python
model.summary()
```

Out[13]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	57
Model:	SARIMAX(0, 1, 0, 12)	**Log Likelihood**	-761.631
Date:	Sat, 26 Apr 2025	**AIC**	1527.262
Time:	23:04:07	**BIC**	1530.876
Sample:	01-01-2015	**HQIC**	1528.609
	- 09-01-2019		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	5.988e+06	8.11e+05	7.385	0.000	4.4e+06	7.58e+06
sigma2	2.941e+13	6.53e+12	4.505	0.000	1.66e+13	4.22e+13

Ljung-Box (L1) (Q):	0.76	**Jarque-Bera (JB):**	0.14
Prob(Q):	0.38	**Prob(JB):**	0.93
Heteroskedasticity (H):	1.26	**Skew:**	-0.10
Prob(H) (two-sided):	0.66	**Kurtosis:**	2.82

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Avaliação de outros pressupostos do modelo

As estatísticas de teste para os testes de Ljung-Box para autocorrelações, de Jarque-Bera para normalidade e de homoscedasticidade dos resíduos, bem como seus p-valores, foram obtidas através do summary() acima do modelo.

Teste de Ljung-Box para autocorrelações dos resíduos

H_0: qualquer grupo de autocorrelações são iguais a 0 (dados não correlacionados).

H_1: os dados são correlacionados.

O p-valor foi igual a 0.38, logo não rejeitamos H_0. Há evidências para dizer que os resíduos não são autocorrelacionados.

Teste de Jarque-Bera para normalidade dos resíduos

H_0: os dados têm assimetria e curtose iguais a de uma distibuição normal (dados tem distribuição normal).

H_1: os dados não têm distribuição normal.

O p-valor foi igual a 0.93, portanto não rejeitamos H_0. Há evidências para dizer que os resíduos são normais.

Teste de homoscedasticidade dos resíduos

H_0: os dados têm variância constante (homoscedásticos).

H_1: os dados não têm variância constante (heteroscedásticos).

O p-valor é igual a 0.66, portanto não rejeitamos H_0. Há evidências para dizer que os resíduos são homoscedásticos.

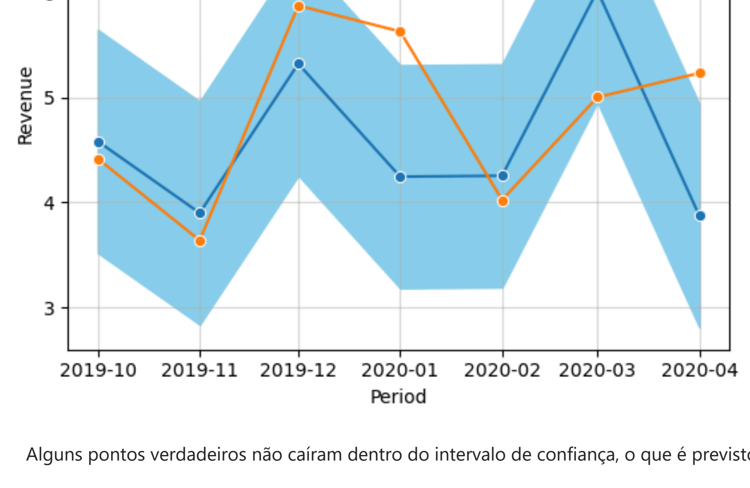
Conclusão

O modelo está validado para uso.

Forecasting

Utilizaremos a base de teste para validar o modelo.

In [14]:
```python
# forecasting
modelfit = model.fit(train['Revenue'])
model_pred, conf_int = modelfit.predict(n_test, return_conf_int = True) # prevendo n_test = 7 períodos à frente
index_fc = pd.date_range(start = test.index[0], end = test.index[len(test) - 1], freq = 'MS') # lista de datas correspondentes às predições
sns.lineplot(model_pred, marker = 'o', label = 'Predições')
sns.lineplot(test['Revenue'], marker = 'o', label = 'Verdadeiro')
plt.fill_between(index_fc, conf_int[:,0], conf_int[:,1], color = 'skyblue') # intervalo de confiança
plt.grid(True, alpha = 0.5)
plt.legend(loc = 'upper left')
plt.show()
```



Alguns pontos verdadeiros não caíram dentro do intervalo de confiança, o que é previsto, pois os intervalos têm um nível de confiança diferente de 100%.

In [15]:
```python
mape = np.mean(np.abs((test['Revenue'] - model_pred) / test['Revenue']))*100
mape
```

Out[15]:
```
13.85710202089875
```

O MAPE (Mean Absolute Percentage Error) é menor que 15%, portanto o modelo é razoável, visto que temos poucas observações.