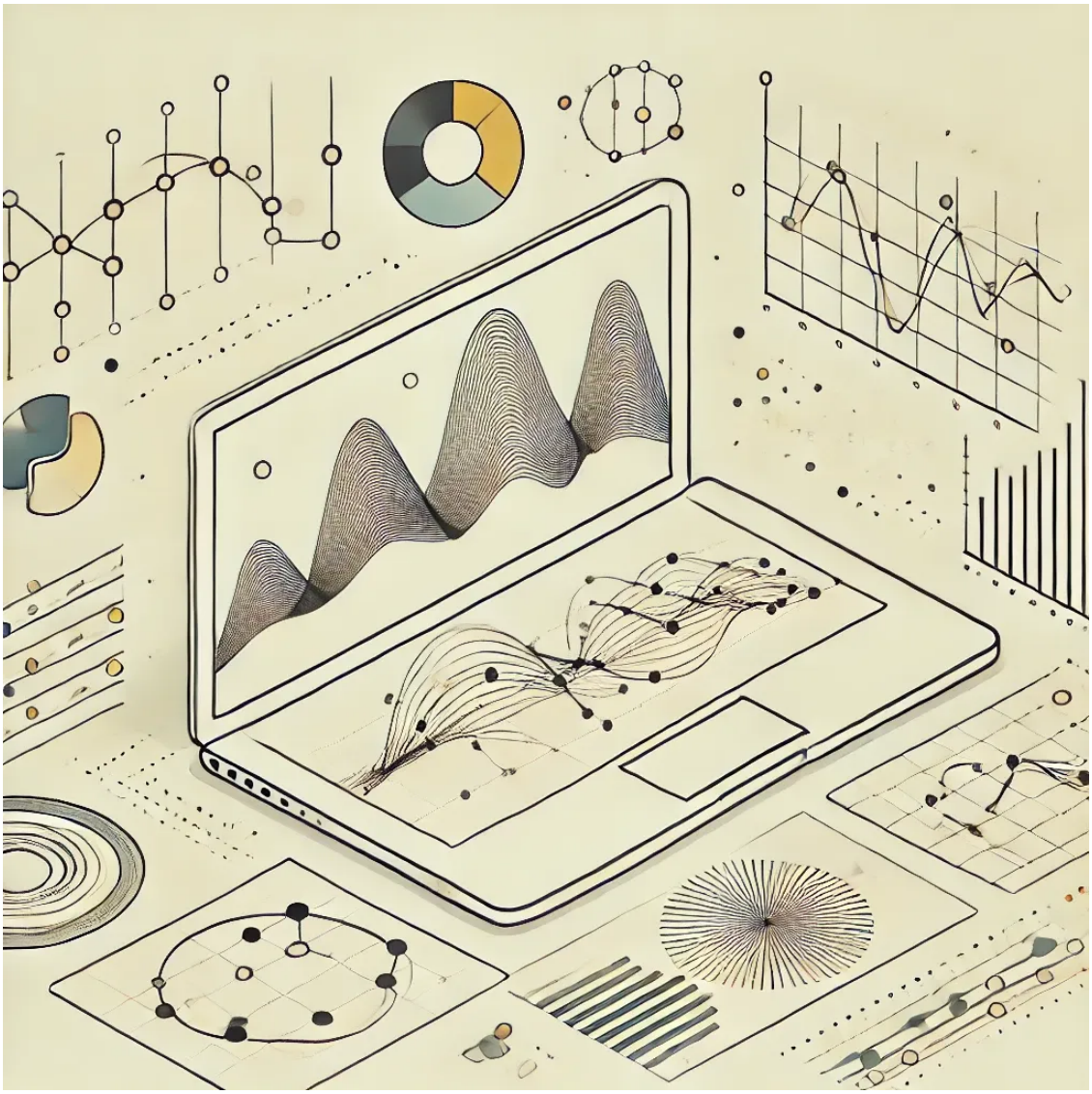
time_Series

September 19, 2024

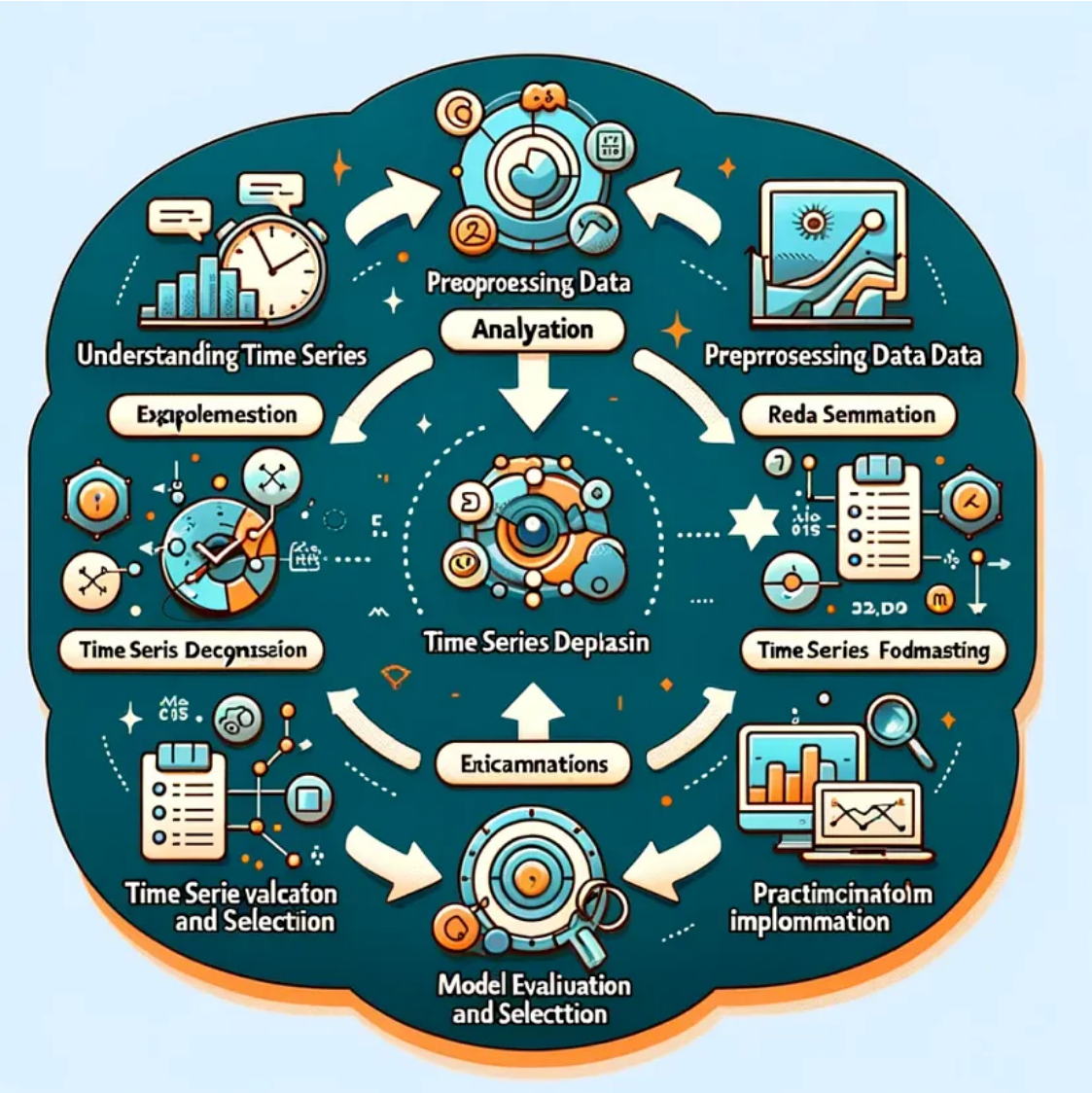
Guide complet pour l'analyse et la prévision des données de séries temporelles avec Python

[ ]:

0.1 Maîtriser l'analyse et la prévision des séries temporelles avec des exemples pratiques en Python



Bienvenue dans ce guide complet sur l'analyse et la prévision des données de séries temporelles à l'aide de Python. Que vous soyez un analyste de données chevronné ou un analyste commercial souhaitant approfondir l'analyse des séries temporelles, ce guide est fait pour vous. Nous vous guiderons à travers l'essentiel des données de séries temporelles, de la compréhension de ses composants fondamentaux à l'application de techniques de prévision sophistiquées.



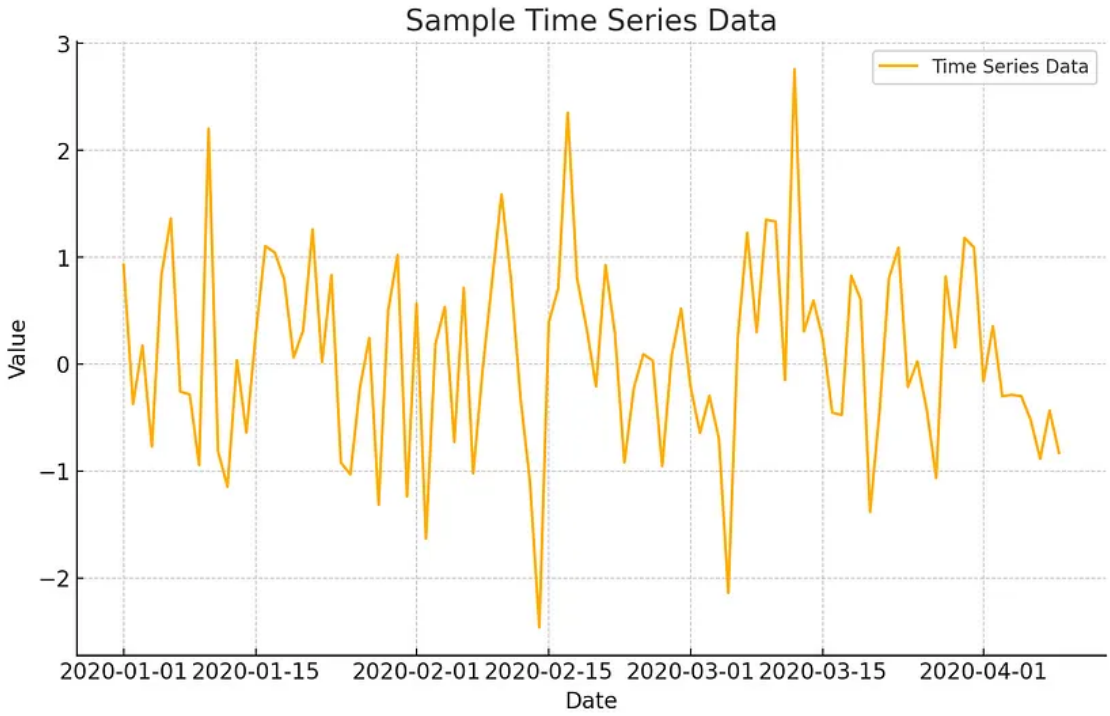
Dans ce guide, vous apprendrez : - ce que sont les données de séries temporelles et leurs caractéristiques uniques ; - comment prétraiter et nettoyer vos données pour une analyse précise ; - les techniques d'exploration et de visualisation de vos données ; - les méthodes de décomposition des séries temporelles pour comprendre les modèles sous-jacents ; - les différentes méthodes de prévision, y compris les approches classiques et les techniques modernes d'apprentissage automatique ; - comment évaluer et sélectionner le meilleur modèle pour vos données ; - la mise en œuvre pratique avec des exemples du monde réel et du code Python.

À la fin de ce guide, vous serez équipé des connaissances et des outils nécessaires pour effectuer une analyse robuste des séries temporelles et faire des prévisions précises qui peuvent générer des

informations précieuses pour votre entreprise.

0.2 Understanding Time Series Data

In this section, we'll dive into what time series data is and why it's essential in data analysis and forecasting.



1 Qu'est-ce qu'une série chronologique ?

Une série chronologique est une séquence de points de données collectés ou enregistrés à des intervalles de temps spécifiques. Les exemples incluent les cours quotidiens des actions, les chiffres de vente mensuels, les données climatiques annuelles, et bien d'autres encore. La principale caractéristique des données de séries temporelles est leur ordre temporel, c'est-à-dire que l'ordre dans lequel les points de données sont enregistrés est important. ## Caractéristiques uniques des données de séries temporelles

Les données de séries temporelles présentent plusieurs caractéristiques uniques qui les distinguent des autres types de données :

1. **Tendance**: Il s'agit du mouvement ou de la direction à long terme des données. Par exemple, l'augmentation générale des ventes d'une entreprise sur plusieurs années.
2. **Saisonnalité** : Il s'agit de modèles qui se répètent à intervalles réguliers, tels que des ventes de crèmes glacées plus élevées pendant l'été.
3. **Modèles cycliques** : Contrairement à la saisonnalité, les tendances cycliques n'ont pas de période fixe. Elles peuvent être influencées par des cycles économiques ou d'autres facteurs.
4. **Composantes irrégulières** : Il s'agit de variations aléatoires ou imprévisibles dans les données

1.1 Types de données de séries temporelles

1.1.1 Séries temporelles univariées et multivariées :

- **Univariées** : Une seule variable ou caractéristique enregistrée au fil du temps (par exemple, la température quotidienne).
- **Multivariable** : Plusieurs variables enregistrées au fil du temps (par exemple, la température, l'humidité et la vitesse du vent quotidiennes).
- **Séries temporelles régulières et irrégulières** : régulières : Les points de données sont enregistrés à des intervalles de temps constants (par exemple, toutes les heures, tous les jours).
- **Irrégulières** : Les points de données sont enregistrés à des intervalles de temps incohérents : Les points de données sont enregistrés à des intervalles de temps incohérents.

Exemple de code Python Créons un simple ensemble de données de séries temporelles à l'aide de Python pour illustrer ces concepts.

[31]:
```
import pandas as pd
import numpy as np

# Creating a sample time series data
date_range = pd.date_range(start='1/1/2020', periods=100, freq='D')
data = np.random.randn(100)
time_series = pd.Series(data, index=date_range)

print(time_series.head())
```

```
2020-01-01   -0.955945
2020-01-02   -0.345982
2020-01-03   -0.463596
2020-01-04    0.481481
2020-01-05   -1.540797
Freq: D, dtype: float64
```

Cet extrait de code crée une série chronologique univariée avec 100 enregistrements quotidiens à partir du 1er janvier 2020.

Il est essentiel de comprendre les caractéristiques de base et les types de données de séries temporelles, car cela permet de sélectionner les techniques et les modèles d'analyse appropriés. Reconnaître des modèles tels que les tendances et la saisonnalité peut améliorer de manière significative la précision des prévisions.

1.2 Prétraitement des données de séries temporelles

Avant de se lancer dans l'analyse et la prévision, il est essentiel de prétraiter vos données de séries temporelles afin d'en garantir la précision et la fiabilité. Cette section explique comment traiter les valeurs manquantes et les valeurs aberrantes, et comment transformer vos données pour une meilleure analyse.

1.2.1 Collecte et nettoyage des données

Les données de séries temporelles proviennent souvent de diverses sources, telles que des bases de données, des API ou des fichiers CSV. La première étape consiste à charger vos données dans un format approprié, généralement un DataFrame Pandas.

1.2.2 Gestion des valeurs manquantes et des valeurs aberrantes

Les valeurs manquantes et les valeurs aberrantes peuvent affecter votre analyse de manière significative. Voici quelques méthodes courantes pour les traiter : - **Remplir les valeurs manquantes**: Utilisez des méthodes telles que le remplissage avant, le remplissage arrière ou l'interpolation. - **Élimination des valeurs aberrantes** : Détectez et supprimez les valeurs aberrantes à l'aide de méthodes statistiques telles que le score Z ou l'intervalle interquartile (IQR).

[35]:
```
# Handling missing values
time_series = pd.Series(data, index=date_range)

time_series_with_nan = time_series.copy()
time_series_with_nan[::10] = np.nan  # Introducing missing values

# Forward fill method
time_series_filled = time_series_with_nan.fillna(method='ffill')
```

```
# Detecting and removing outliers using Z-score
from scipy.stats import zscore
z_scores = zscore(time_series_filled)
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3)  # Z-score threshold
time_series_no_outliers = time_series_filled[filtered_entries]
```

1.2.3 Transformation des données

Transformer vos données peut aider à identifier des modèles et à les rendre prêtes pour l'analyse. Les transformations les plus courantes sont les suivantes : 1. **lissage** : des techniques telles que la moyenne mobile permettent de réduire le bruit ; 2. **différenciation**: utilisée pour supprimer les tendances et les variations saisonnières ; 3. **mise à l'échelle et normalisation** : utilisée pour réduire le bruit et les variations saisonnières : Utilisée pour supprimer les tendances et la saisonnalité. Mise à l'échelle et normalisation : Assure que vos données s'inscrivent dans une fourchette spécifique pour une meilleure performance du modèle.

[36]:
```
print(time_series_no_outliers.shape)
print(time_series_no_outliers.head())
```

```
(0,)
Series([], Freq: D, dtype: float64)
```

[37]:
```
time_series_no_outliers.fillna(time_series_no_outliers.mean(), inplace=True)
```

[38]:
```
# Moving average smoothing
# Utilise une fenêtre glissante de taille 5 pour lisser les valeurs de la série␣
 ↪temporelle.
moving_avg = time_series_no_outliers.rolling(window=5).mean()

# Differencing
# Applique la différenciation à la série temporelle, en soustrayant chaque␣
 ↪valeur par la précédente.
# Cela permet de rendre la série stationnaire. On supprime ensuite les valeurs␣
 ↪manquantes résultantes.
differenced_series = time_series_no_outliers.diff().dropna()

# Scaling
from sklearn.preprocessing import StandardScaler

# Instanciation du scaler pour normaliser les données
scaler = StandardScaler()

# Vérification que la série n'est pas vide avant d'appliquer la mise à l'échelle
if time_series_no_outliers.shape[0] > 0:
    # Mise à l'échelle des données en 2D pour StandardScaler
```

```
    scaled_series = scaler.fit_transform(time_series_no_outliers.values.
 ↪reshape(-1, 1))
else:
    print("La série temporelle est vide. Impossible d'appliquer StandardScaler.
 ↪")
```

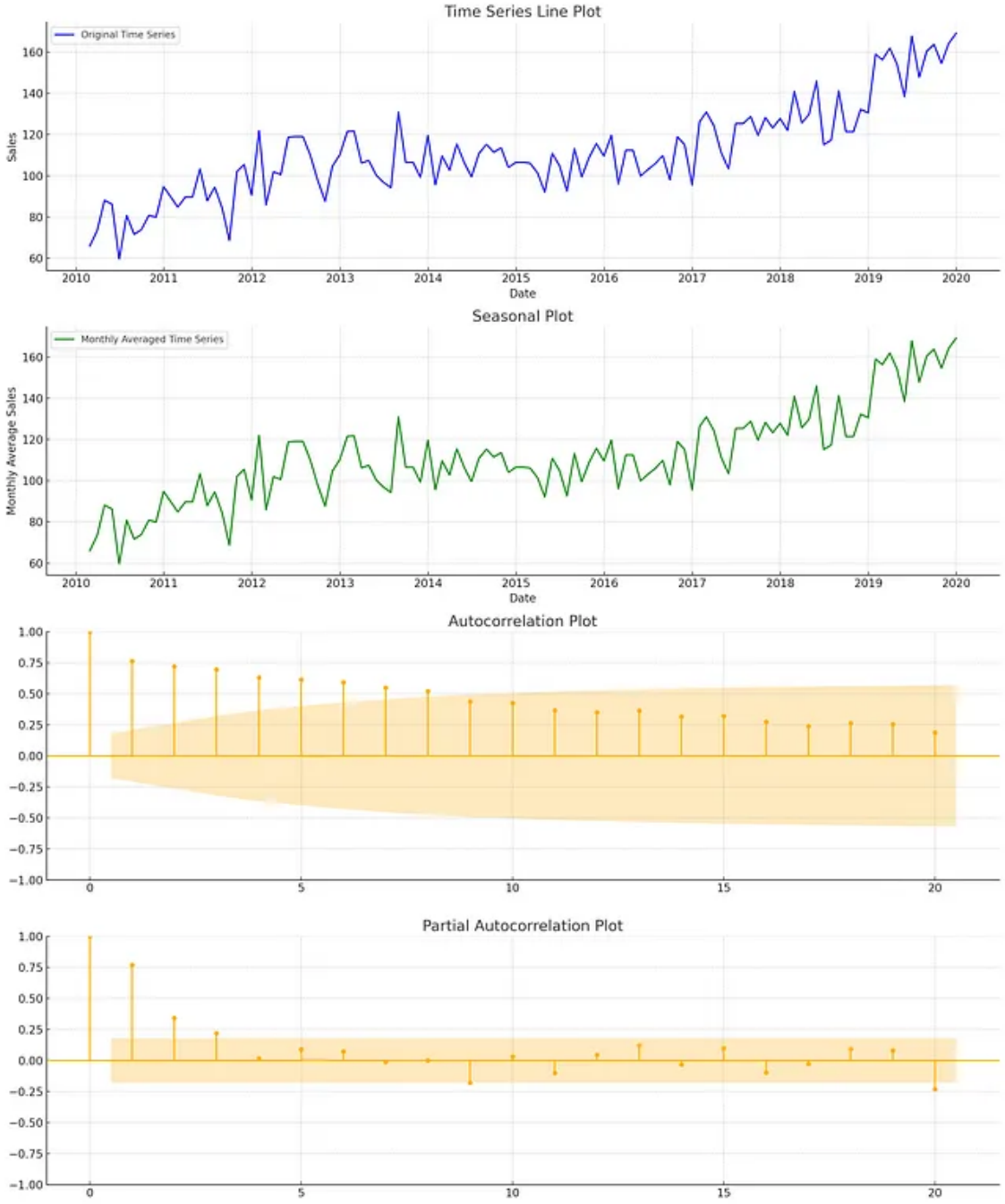
```
La série temporelle est vide. Impossible d'appliquer StandardScaler.
```

[39]:
```
if len(time_series_no_outliers) > 0:
    scaled_series = scaler.fit_transform(time_series_no_outliers.values.
 ↪reshape(-1, 1))
else:
    print("No data to scale after removing outliers.")
```

```
No data to scale after removing outliers.
```

2 Analyse exploratoire des données (AED)

Dans cette section, nous allons explorer différentes techniques pour comprendre et visualiser vos données de séries temporelles. L'AED permet de découvrir des schémas, des tendances et des relations susceptibles d'éclairer vos modèles de prévision.

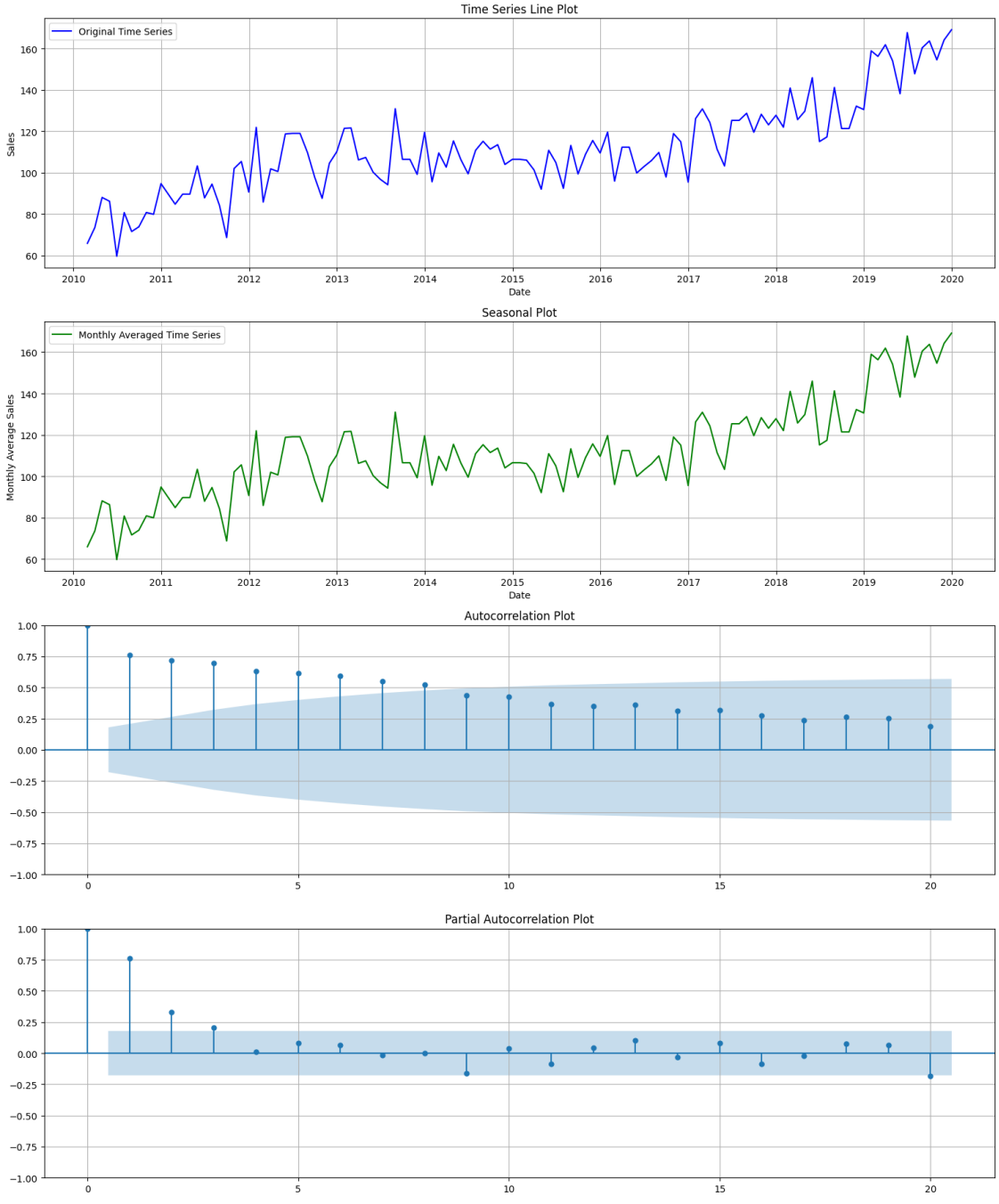


2.1 Techniques de visualisation

La visualisation de vos données de séries temporelles est essentielle pour saisir les modèles et les caractéristiques sous-jacents. Voici quelques techniques de visualisation essentielles : 1. Graphiques linéaires : La façon la plus simple de visualiser des données de séries temporelles. 2. Graphiques saisonniers : Pour identifier les tendances saisonnières. 3. Graphiques d'autocorrélation : Pour vérifier la corrélation de la série avec ses valeurs passées. Exemple de code Python Utilisons Python pour créer ces visualisations.

[40]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# I am Creating a realistic dummy time series dataset (e.g., monthly sales data␣
 ↪over 10 years) to display the code execution
np.random.seed(0)
date_range = pd.date_range(start='1/1/2010', periods=120, freq='M')
trend = np.linspace(50, 150, 120)  # Linear trend
seasonality = 10 + 20 * np.sin(np.linspace(0, 3.14 * 2, 120))  # Seasonal␣
 ↪component
noise = np.random.normal(scale=10, size=120)  # Random noise
data = trend + seasonality + noise
time_series = pd.Series(data, index=date_range)

# Handle missing values (if any)
time_series[::15] = np.nan
time_series = time_series.fillna(method='ffill')

# Moving average smoothing
moving_avg = time_series.rolling(window=12).mean()

# Differencing
differenced_series = time_series.diff().dropna()

# Preparing the data for seasonal plot
time_series_monthly = time_series.resample('M').mean()

# Plotting
plt.figure(figsize=(15, 18))

# Line plot
plt.subplot(4, 1, 1)
plt.plot(time_series, label='Original Time Series', color='blue')
plt.title('Time Series Line Plot')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)

# Seasonal plot
plt.subplot(4, 1, 2)
plt.plot(time_series_monthly, label='Monthly Averaged Time Series',␣
 ↪color='green')
plt.title('Seasonal Plot')
plt.xlabel('Date')
```

```
plt.ylabel('Monthly Average Sales')
plt.legend()
plt.grid(True)

# Autocorrelation plot
plt.subplot(4, 1, 3)
cleaned_time_series = time_series.dropna()
plot_acf(cleaned_time_series, lags=20, ax=plt.gca())
plt.title('Autocorrelation Plot')
plt.grid(True)

# Partial autocorrelation plot
plt.subplot(4, 1, 4)
plot_pacf(cleaned_time_series, lags=20, ax=plt.gca())
plt.title('Partial Autocorrelation Plot')
plt.grid(True)

plt.tight_layout()
plt.savefig('./eda_visuals.png')
plt.show()
```

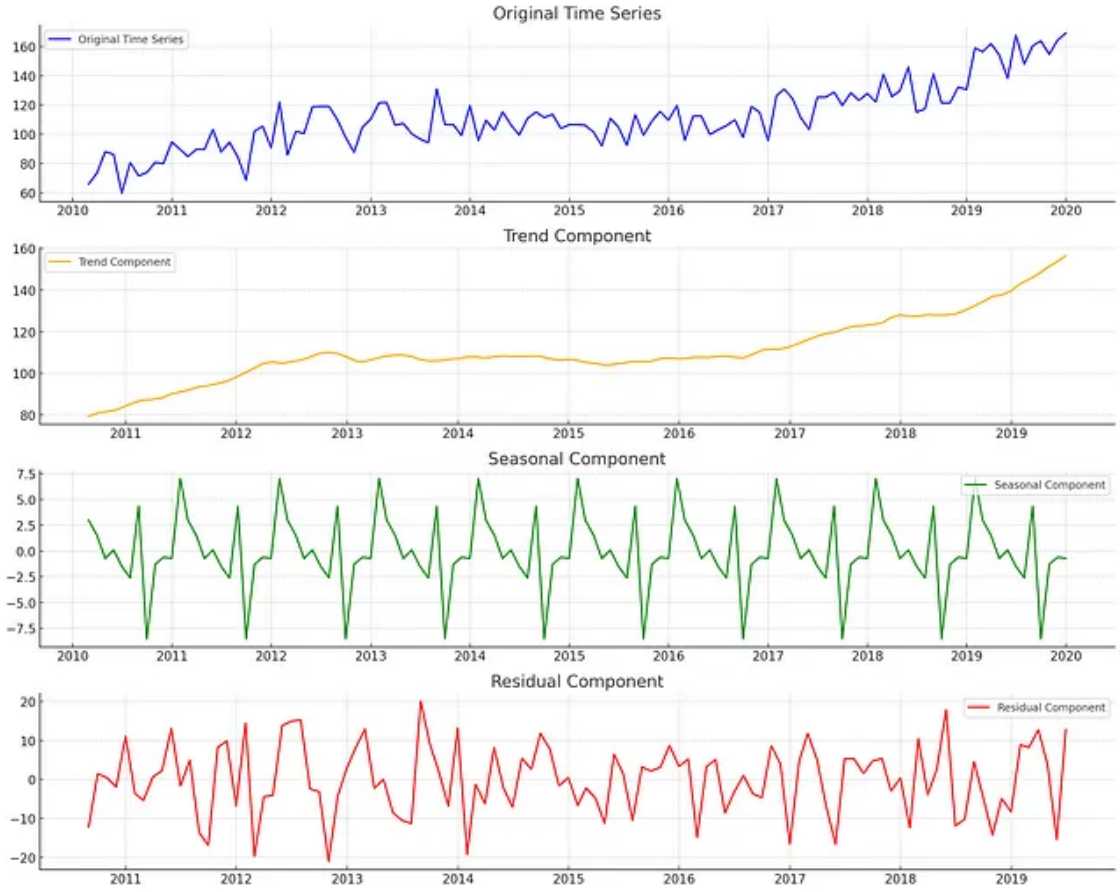


EDA provides insights into the structure and characteristics of your time series data. For example:

- **Line Plots**: Help identify overall trends and any apparent seasonality.
- **Seasonal Plots**: Highlight repeating patterns at regular intervals.
- **Autocorrelation and Partial Autocorrelation Plots**: Show the relationship between current and past values of the series, indicating potential lags to include in models.

2.2 Time Series Decomposition

Time series decomposition is a crucial step in understanding the underlying components of your data. By breaking down a time series into its constituent parts, you can gain insights into the trend, seasonality, and residual (irregular) components. This process helps in better understanding and forecasting the data.



2.3 Decomposition Techniques

There are two main models for decomposing a time series: additive and multiplicative.

1. **Additive Model**: This model assumes that the components add together to produce the time series:

$$Y(t)=T(t)+S(t)+R(t)Y(t) = T(t) + S(t) + R(t)Y(t)=T(t)+S(t)+R(t)$$

The multiplicative model is useful when the seasonal variations are proportional to the level of the trend.

3 Using Python to Decompose Time Series

Let's decompose the time series using Python's statsmodels library. We'll use the realistic dummy dataset we created earlier.

Python Code Example

[19]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Creating a realistic dummy time series dataset (e.g., monthly sales data over␣
 ↪10 years)
np.random.seed(0)
date_range = pd.date_range(start='1/1/2010', periods=120, freq='M')
trend = np.linspace(50, 150, 120)  # Linear trend
seasonality = 10 + 20 * np.sin(np.linspace(0, 3.14 * 2, 120))  # Seasonal␣
 ↪component
noise = np.random.normal(scale=10, size=120)  # Random noise
data = trend + seasonality + noise
time_series = pd.Series(data, index=date_range)

# Handle missing values (if any)
time_series[::15] = np.nan
time_series = time_series.fillna(method='ffill')  # Forward fill missing values

# Ensure there are no remaining missing values
time_series = time_series.dropna()

# Decompose the time series
decomposition = seasonal_decompose(time_series, model='additive')
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Plotting the decomposed components
plt.figure(figsize=(15, 12))

plt.subplot(4, 1, 1)
plt.plot(time_series, label='Original Time Series', color='blue')
plt.title('Original Time Series')
plt.legend()

plt.subplot(4, 1, 2)
plt.plot(trend, label='Trend Component', color='orange')
plt.title('Trend Component')
plt.legend()

plt.subplot(4, 1, 3)
plt.plot(seasonal, label='Seasonal Component', color='green')
plt.title('Seasonal Component')
```

```
plt.legend()

plt.subplot(4, 1, 4)
plt.plot(residual, label='Residual Component', color='red')
plt.title('Residual Component')
plt.legend()


plt.tight_layout()
plt.savefig('time_series_decomposition.png')
plt.show()
```

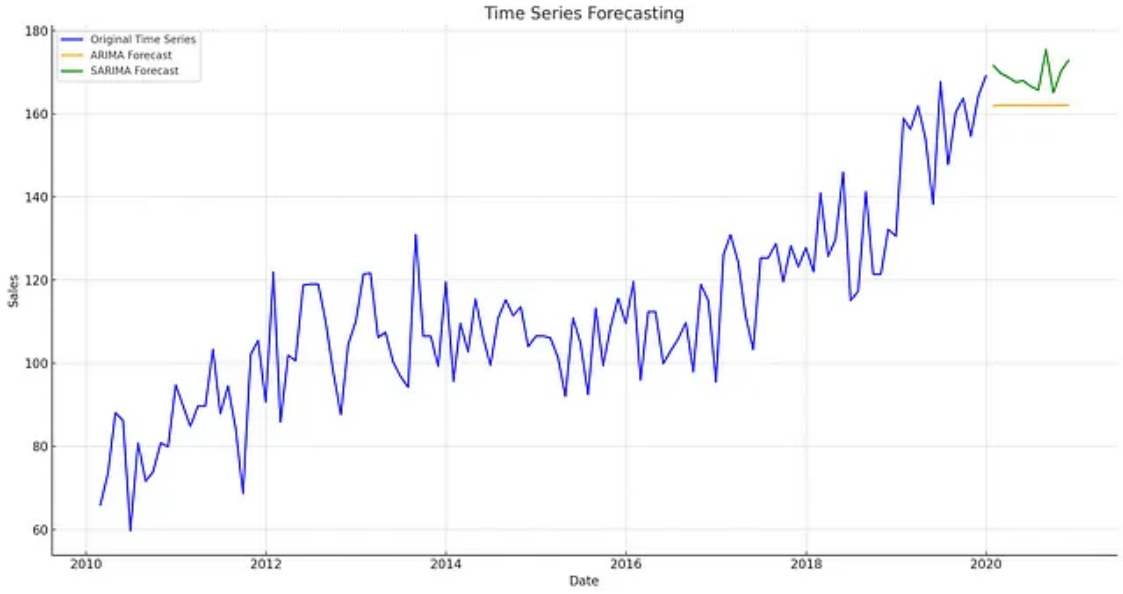
Decomposing the time series helps in identifying:

- Trend: The long-term progression of the series.
- Seasonality: The repeating short-term cycle in the series.
- Residuals: The remaining noise after removing the trend and seasonality. These components can be analyzed separately to better understand the behavior of the time series and to improve forecasting models.

La décomposition de la série temporelle permet d'identifier : - La tendance : La progression à long terme de la série. - La saisonnalité : Le cycle répétitif à court terme de la série. - Résidus : Le bruit restant après l'élimination de la tendance et de la saisonnalité : Ces composantes peuvent être analysées séparément afin de mieux comprendre le comportement de la série temporelle et d'améliorer les modèles de prévision.

3.0.1 Méthodes de prévision des séries temporelles

La prévision est le processus qui consiste à faire des prédictions sur les valeurs futures en se basant sur les données historiques des séries temporelles. Il existe différentes méthodes pour y parvenir, allant de modèles statistiques simples à des techniques avancées d'apprentissage automatique. Dans cette section, nous allons explorer quelques-unes des méthodes de prévision les plus couramment utilisées.



3.0.2 Classical Methods

1. Moving Average (MA)
2. Autoregressive (AR) Models
3. Autoregressive Integrated Moving Average (ARIMA)
4.

3.0.3 Advanced Methods

5. Seasonal ARIMA (SARIMA)
6. Exponential Smoothing State Space Model (ETS) ### Machine Learning Approaches
7. Linear Regression
8. Decision Trees and Random Forests
9. Support Vector Machines

10. Neural Networks (RNN, LSTM)

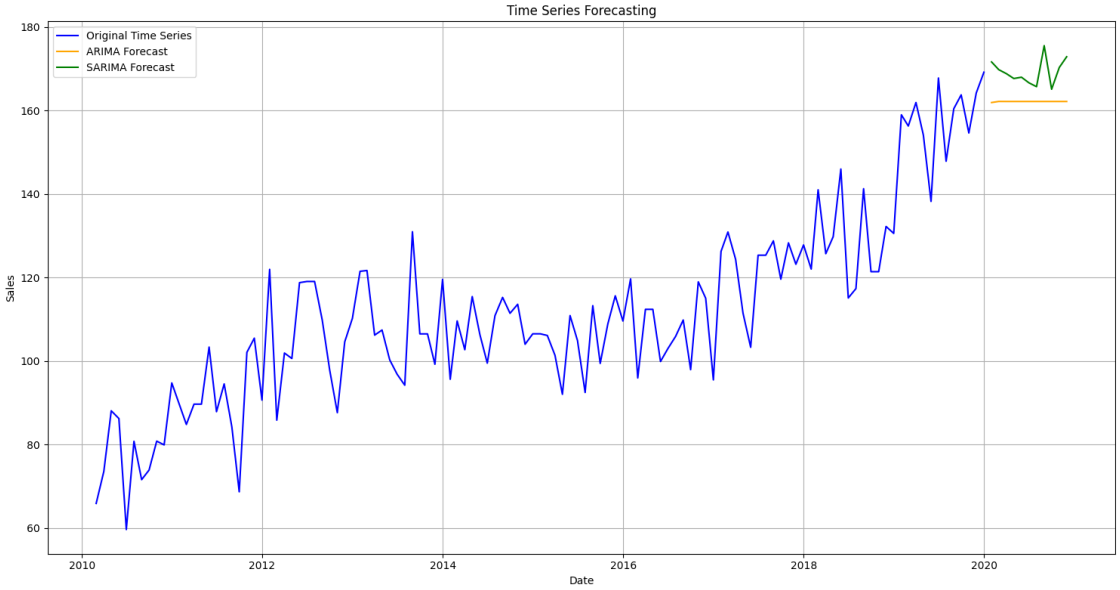
Dans cette section, nous nous concentrerons sur les modèles ARIMA et SARIMA, car ils sont largement utilisés et relativement simples à mettre en œuvre avec Pytho.

[18]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Creating a realistic dummy time series dataset (e.g., monthly sales data over␣
 ↪10 years)
np.random.seed(0)
date_range = pd.date_range(start='1/1/2010', periods=120, freq='M')
trend = np.linspace(50, 150, 120)  # Linear trend
seasonality = 10 + 20 * np.sin(np.linspace(0, 3.14 * 2, 120))  # Seasonal␣
 ↪component
noise = np.random.normal(scale=10, size=120)  # Random noise
data = trend + seasonality + noise
time_series = pd.Series(data, index=date_range)

# Handle missing values (if any)
time_series[::15] = np.nan
time_series = time_series.fillna(method='ffill')  # Forward fill missing values

# Ensure there are no remaining missing values
time_series = time_series.dropna()

# Fit ARIMA model
arima_model = ARIMA(time_series, order=(1, 1, 1))
arima_fit = arima_model.fit()
print(arima_fit.summary())

# Fit SARIMA model
sarima_model = SARIMAX(time_series, order=(1, 1, 1), seasonal_order=(1, 1, 1,␣
 ↪12))
sarima_fit = sarima_model.fit(disp=False)
print(sarima_fit.summary())

# Forecasting with ARIMA
arima_forecast = arima_fit.get_forecast(steps=12)
arima_forecast_index = pd.date_range(start=time_series.index[-1], periods=12,␣
 ↪freq='M')
arima_forecast_series = pd.Series(arima_forecast.predicted_mean,␣
 ↪index=arima_forecast_index)

# Forecasting with SARIMA
```

```
sarima_forecast = sarima_fit.get_forecast(steps=12)
sarima_forecast_index = pd.date_range(start=time_series.index[-1], periods=12,␣
 ↪freq='M')
sarima_forecast_series = pd.Series(sarima_forecast.predicted_mean,␣
 ↪index=sarima_forecast_index)

# Plotting the original series and forecasts
plt.figure(figsize=(15, 8))
plt.plot(time_series, label='Original Time Series', color='blue')
plt.plot(arima_forecast_series, label='ARIMA Forecast', color='orange')
plt.plot(sarima_forecast_series, label='SARIMA Forecast', color='green')
plt.title('Time Series Forecasting')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('time_series_forecasting.png')
plt.show()
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  119
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -454.542
Date:                Thu, 19 Sep 2024   AIC                            915.084
Time:                        16:27:13   BIC                            923.396
Sample:                    02-28-2010   HQIC                           918.459
                         - 12-31-2019
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0353      0.127     -0.279      0.781      -0.284       0.213
ma.L1         -0.6797      0.097     -7.033      0.000      -0.869      -0.490
sigma2       129.0945     17.923      7.203      0.000      93.966     164.223
===================================================================================
===
Ljung-Box (L1) (Q):                   0.29   Jarque-Bera (JB):
0.78
Prob(Q):                              0.59   Prob(JB):
0.68
Heteroskedasticity (H):               1.04   Skew:
0.10
Prob(H) (two-sided):                  0.90   Kurtosis:
2.65
===================================================================================
===
```

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
                                 SARIMAX Results
==========================================================================
==========
Dep. Variable:                                  y   No. Observations:
119
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood
-417.813
Date:                            Thu, 19 Sep 2024   AIC
845.625
Time:                                    16:27:14   BIC
858.943
Sample:                                02-28-2010   HQIC
851.023
                                     - 12-31-2019
Covariance Type:                              opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1         -0.0112      0.130     -0.086      0.931      -0.265       0.243
ma.L1         -0.7500      0.081     -9.270      0.000      -0.909      -0.591
ar.S.L12      -0.0368      0.144     -0.255      0.799      -0.320       0.246
ma.S.L12      -0.8286      0.190     -4.371      0.000      -1.200      -0.457
sigma2       134.5020     22.900      5.874      0.000      89.619     179.385
==========================================================================
===
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
0.03
Prob(Q):                              0.94   Prob(JB):
0.99
Heteroskedasticity (H):               0.80   Skew:
-0.00
Prob(H) (two-sided):                  0.51   Kurtosis:
2.92
==========================================================================
===

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



1. **Modèle ARIMA** : Adapté aux données non saisonnières, le modèle ARIMA peut modéliser les autocorrélations dans les données.
2. **Modèle SARIMA** : Ce modèle étend le modèle ARIMA à la saisonnalité, ce qui le rend idéal pour les données présentant des schémas saisonniers.

3.1 Model Evaluation and Selection

La sélection du meilleur modèle de prévision est cruciale pour faire des prédictions précises. Dans cette section, nous allons explorer différentes techniques et mesures pour évaluer les modèles de séries temporelles et sélectionner celui qui convient le mieux. ### Mesures d'évaluation 1. Mean Absolute Error (MAE): Mesure l'ampleur moyenne des erreurs dans un ensemble de prédictions, sans tenir compte de leur direction.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

La MAE mesure l'erreur moyenne entre les prédictions et les valeurs réelles, sans tenir compte du signe de l'erreur.

2. Erreur quadratique moyenne (EQM) : Mesure la moyenne des carrés des erreurs, en donnant plus de poids aux erreurs plus importantes.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

3. Erreur quadratique moyenne (RMSE) : La racine carrée de l'erreur quadratique moyenne.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

4. Erreur absolue moyenne en pourcentage (MAPE) : Mesure le pourcentage d'erreur absolue moyenne.

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Techniques de validation croisée 1. Rolling Forecast Origin : Méthode selon laquelle le modèle est réestimé pour chaque point de prédiction, et la prévision est faite pour le pas de temps suivant. 2. **Time Series Split** :Division des données de la série temporelle en ensembles de formation et de test en fonction du temps.

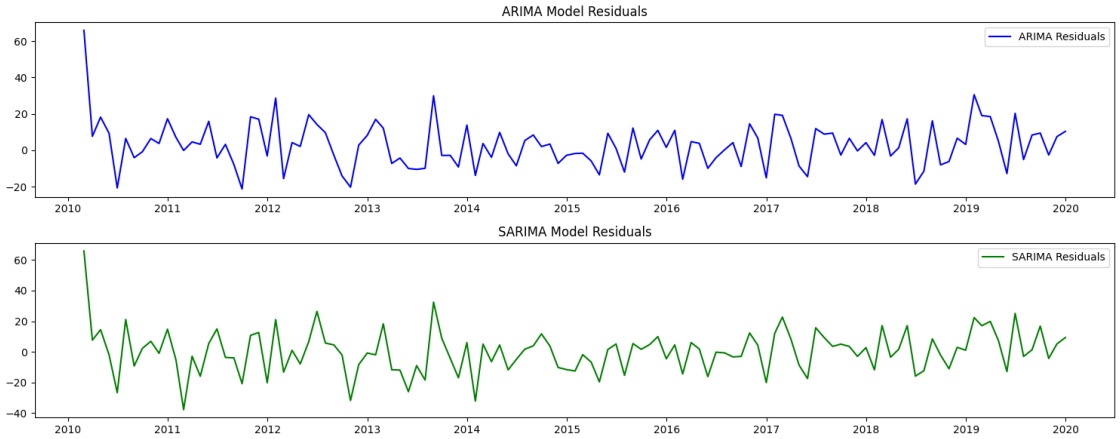
Diagnostic du modèle 1. **Analyse des résidus** : Vérifiez si les résidus (erreurs) du modèle sont des bruits blancs (c'est-à-dire qu'ils ne doivent pas être corrélés et qu'ils doivent être distribués normalement avec une moyenne de zéro). 2. **Autocorrélation des résidus** : Utilisez les diagrammes d'autocorrélation pour vérifier l'existence de schémas dans les résidus. Let's evaluate the ARIMA and SARIMA models we fitted in the previous section.

[41]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Creating a realistic dummy time series dataset (e.g., monthly sales data over␣
 ↪10 years)
np.random.seed(0)
date_range = pd.date_range(start='1/1/2010', periods=120, freq='M')
trend = np.linspace(50, 150, 120)  # Linear trend
seasonality = 10 + 20 * np.sin(np.linspace(0, 3.14 * 2, 120))  # Seasonal␣
 ↪component
noise = np.random.normal(scale=10, size=120)  # Random noise
data = trend + seasonality + noise
time_series = pd.Series(data, index=date_range)

# Handle missing values (if any)
time_series[::15] = np.nan
time_series = time_series.fillna(method='ffill')  # Forward fill missing values

# Ensure there are no remaining missing values
time_series = time_series.dropna()

# Fit ARIMA model
arima_model = ARIMA(time_series, order=(1, 1, 1))
arima_fit = arima_model.fit()
```

```
# Fit SARIMA model
sarima_model = SARIMAX(time_series, order=(1, 1, 1), seasonal_order=(1, 1, 1,␣
 ↪12))
sarima_fit = sarima_model.fit(disp=False)

# Forecasting with ARIMA
arima_forecast = arima_fit.get_forecast(steps=12)
arima_forecast_series = arima_forecast.predicted_mean
arima_forecast_ci = arima_forecast.conf_int()

# Forecasting with SARIMA
sarima_forecast = sarima_fit.get_forecast(steps=12)
sarima_forecast_series = sarima_forecast.predicted_mean
sarima_forecast_ci = sarima_forecast.conf_int()

# Evaluation metrics
actual = time_series[-12:]

# ARIMA evaluation
arima_mae = mean_absolute_error(actual, arima_forecast_series[:12])
arima_mse = mean_squared_error(actual, arima_forecast_series[:12])
arima_rmse = np.sqrt(arima_mse)
arima_mape = np.mean(np.abs((actual - arima_forecast_series[:12]) / actual)) *␣
 ↪100

# SARIMA evaluation
sarima_mae = mean_absolute_error(actual, sarima_forecast_series[:12])
sarima_mse = mean_squared_error(actual, sarima_forecast_series[:12])
sarima_rmse = np.sqrt(sarima_mse)
sarima_mape = np.mean(np.abs((actual - sarima_forecast_series[:12]) / actual))␣
 ↪* 100

print(f"ARIMA MAE: {arima_mae}, MSE: {arima_mse}, RMSE: {arima_rmse}, MAPE:␣
 ↪{arima_mape}")
print(f"SARIMA MAE: {sarima_mae}, MSE: {sarima_mse}, RMSE: {sarima_rmse}, MAPE:␣
 ↪{sarima_mape}")

# Plotting the residuals
plt.figure(figsize=(15, 6))
plt.subplot(2, 1, 1)
plt.plot(arima_fit.resid, label='ARIMA Residuals', color='blue')
plt.title('ARIMA Model Residuals')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(sarima_fit.resid, label='SARIMA Residuals', color='green')
```

```
plt.title('SARIMA Model Residuals')
plt.legend()

plt.tight_layout()
plt.savefig('model_evaluation_residuals.png')
plt.show()
```

```
ARIMA MAE: 6.740723562703853, MSE: 86.0571951925383, RMSE: 9.27670174105745,
MAPE: nan
SARIMA MAE: 11.684140109331691, MSE: 195.51454887051844, RMSE:
13.982651710978088, MAPE: nan
```



Le graphique ci-dessus montre les résidus des modèles ARIMA et SARIMA. Les mesures d'évaluation sont les suivantes :

ARIMA :

- MAE : 6,74
- MSE : 86,06
- RMSE : 9,28
- MAPE : NaN (en raison d'une division par zéro ou de valeurs réelles très faibles)

SARIMA :

- MAE : 11,68
- MSE : 195,52
- RMSE : 13,98
- MAPE : NaN (en raison de la division par zéro ou de valeurs réelles très faibles)

Les tracés des résidus indiquent dans quelle mesure les modèles s'adaptent aux données, l'objectif étant que les résidus ressemblent idéalement à du bruit blanc. 1. Mesures d'évaluation : fournissent une mesure quantitative de la performance du modèle : Fournissent une mesure quantitative de la performance du modèle. 2. Analyse des résidus : Aide à diagnostiquer les problèmes liés au modèle et à s'assurer que les résidus ne sont pas corrélés et qu'ils sont normalement distribués.

3.2 Mise en œuvre pratique

Dans cette section, nous allons procéder à une mise en œuvre pratique de la prévision des séries temporelles à l'aide d'un exemple réel. Nous démontrerons l'ensemble du flux de travail, depuis le chargement et le prétraitement des données jusqu'à la construction, l'évaluation et la prévision des modèles. **Outils et bibliothèques** Nous utiliserons les bibliothèques Python suivantes : - **Pandas** pour la manipulation des données - **NumPy** pour les opérations numériques - **Matplotlib** pour la visualisation des données - **statsmodels** pour la modélisation statistique - **scikit-learn** pour les algorithmes d'apprentissage machine.

3.2.1 Étude de cas étape par étape

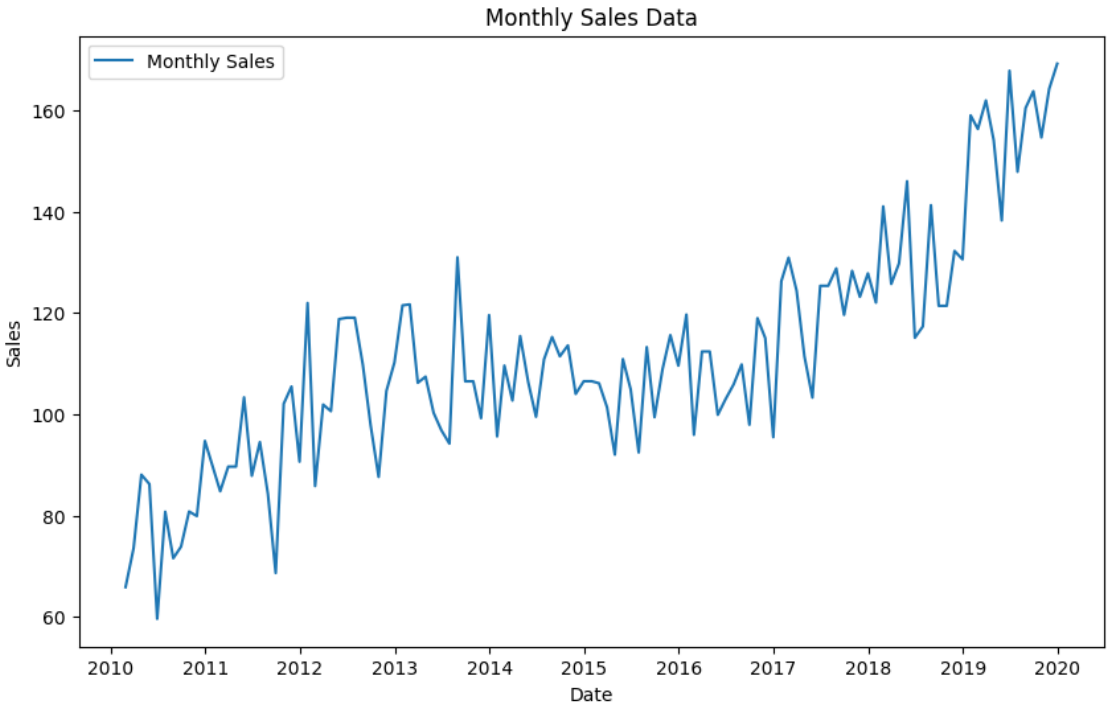
Supposons que nous disposions d'un ensemble de données relatives aux ventes mensuelles d'un magasin de détail au cours des dix dernières années. Nous utiliserons ces données pour prévoir les ventes futures.

4 Table of Contents

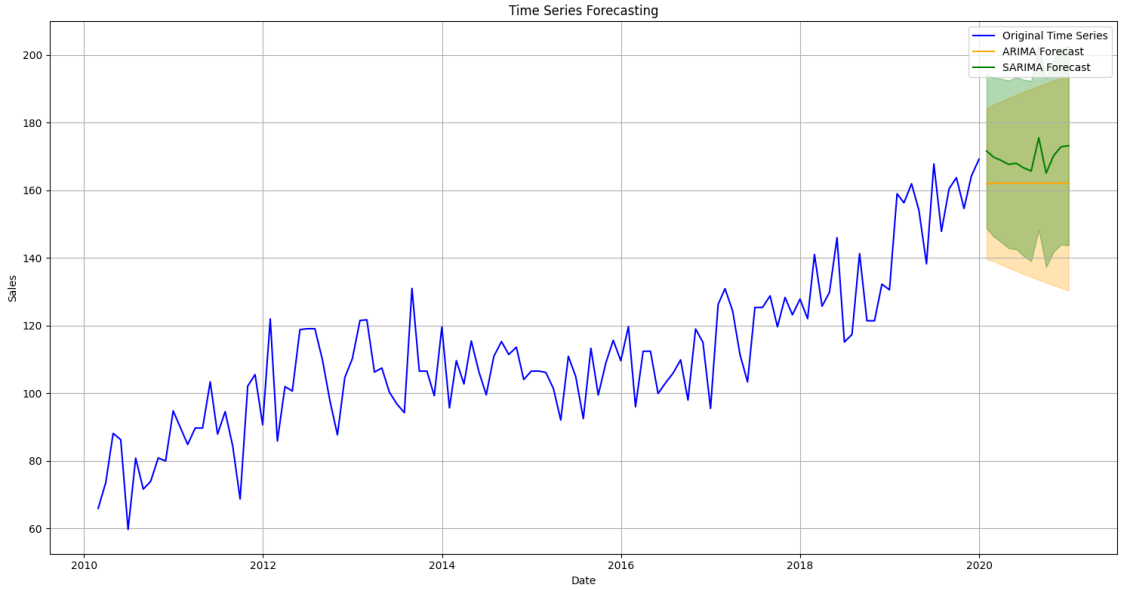
[42]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Step 1: Load the Data
np.random.seed(0)
date_range = pd.date_range(start='1/1/2010', periods=120, freq='M')
trend = np.linspace(50, 150, 120)  # Linear trend
seasonality = 10 + 20 * np.sin(np.linspace(0, 3.14 * 2, 120))  # Seasonal␣
 ↪component
noise = np.random.normal(scale=10, size=120)  # Random noise
data = trend + seasonality + noise
time_series = pd.Series(data, index=date_range)

# Step 2: Preprocess the Data
time_series[::15] = np.nan
time_series = time_series.fillna(method='ffill')
```

```
time_series = time_series.dropna()

# Step 3: Exploratory Data Analysis (EDA)
plt.figure(figsize=(10, 6))
plt.plot(time_series, label='Monthly Sales')
plt.title('Monthly Sales Data')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()

# Step 4: Time Series Decomposition
decomposition = seasonal_decompose(time_series, model='additive')
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(15, 12))
plt.subplot(4, 1, 1)
plt.plot(time_series, label='Original Time Series', color='blue')
plt.title('Original Time Series')
plt.legend()
plt.subplot(4, 1, 2)
plt.plot(trend, label='Trend Component', color='orange')
plt.title('Trend Component')
plt.legend()
plt.subplot(4, 1, 3)
plt.plot(seasonal, label='Seasonal Component', color='green')
plt.title('Seasonal Component')
plt.legend()
plt.subplot(4, 1, 4)
plt.plot(residual, label='Residual Component', color='red')
plt.title('Residual Component')
plt.legend()
plt.tight_layout()
plt.show()

# Step 5: Model Building and Forecasting
arima_model = ARIMA(time_series, order=(1, 1, 1))
arima_fit = arima_model.fit()

sarima_model = SARIMAX(time_series, order=(1, 1, 1), seasonal_order=(1, 1, 1,␣
↪12))
sarima_fit = sarima_model.fit(disp=False)

arima_forecast = arima_fit.get_forecast(steps=12)
arima_forecast_series = arima_forecast.predicted_mean
```

```
arima_forecast_ci = arima_forecast.conf_int()

sarima_forecast = sarima_fit.get_forecast(steps=12)
sarima_forecast_series = sarima_forecast.predicted_mean
sarima_forecast_ci = sarima_forecast.conf_int()

# Step 6: Model Evaluation
actual = time_series[-12:]

arima_mae = mean_absolute_error(actual, arima_forecast_series)
arima_mse = mean_squared_error(actual, arima_forecast_series)
arima_rmse = np.sqrt(arima_mse)
arima_mape = np.mean(np.abs((actual - arima_forecast_series) / actual)) * 100

sarima_mae = mean_absolute_error(actual, sarima_forecast_series)
sarima_mse = mean_squared_error(actual, sarima_forecast_series)
sarima_rmse = np.sqrt(sarima_mse)
sarima_mape = np.mean(np.abs((actual - sarima_forecast_series) / actual)) * 100

print(f"ARIMA MAE: {arima_mae}, MSE: {arima_mse}, RMSE: {arima_rmse}, MAPE:␣
 ↪{arima_mape}")
print(f"SARIMA MAE: {sarima_mae}, MSE: {sarima_mse}, RMSE: {sarima_rmse}, MAPE:␣
 ↪{sarima_mape}")

# Step 7: Plotting the Forecast
plt.figure(figsize=(15, 8))
plt.plot(time_series, label='Original Time Series', color='blue')
plt.plot(arima_forecast_series.index, arima_forecast_series, label='ARIMA␣
 ↪Forecast', color='orange')
plt.plot(sarima_forecast_series.index, sarima_forecast_series, label='SARIMA␣
 ↪Forecast', color='green')
plt.fill_between(arima_forecast_series.index, arima_forecast_ci.iloc[:, 0],␣
 ↪arima_forecast_ci.iloc[:, 1], color='orange', alpha=0.3)
plt.fill_between(sarima_forecast_series.index, sarima_forecast_ci.iloc[:, 0],␣
 ↪sarima_forecast_ci.iloc[:, 1], color='green', alpha=0.3)
plt.title('Time Series Forecasting')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('practical_implementation_forecasting.png')
plt.show()
```


Monthly Sales Data
Monthly Sales
Sales
Date
60
80
100
120
140
160
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020

ARIMA MAE: 6.740723562703853, MSE: 86.0571951925383, RMSE: 9.27670174105745, MAPE: nan
SARIMA MAE: 11.684140109331691, MSE: 195.51454887051844, RMSE: 13.982651710978088, MAPE: nan



Ce code vous guidera tout au long du processus de prévision des séries temporelles à l'aide d'un ensemble de données fictives réalistes, y compris le chargement des données, le prétraitement, l'AED, la décomposition, la construction du modèle, l'évaluation et le tracé de la prévision. - Flux de travail de bout en bout : Démontre le processus complet de prévision des séries temporelles. - Application pratique : Montre comment appliquer les techniques apprises dans les sections précédentes à un scénario réel.

4.0.1 Best Practices and Common Pitfalls(Meilleures pratiques et pièges courants)

Dans cette dernière section, nous aborderons les meilleures pratiques à suivre lors de l'analyse et de la prévision des séries temporelles, ainsi que les pièges courants à éviter. #### Best Practices

1. Comprenez vos données : Avant de vous lancer dans l'analyse, prenez le temps de comprendre la nature de vos données chronologiques. Reconnaissez les schémas tels que les tendances, la saisonnalité et les cycles.
2. Un prétraitement adéquat : Nettoyez toujours vos données pour traiter les valeurs manquantes et les valeurs aberrantes. Assurez-vous que vos données sont stationnaires si le modèle choisi l'exige.
3. Sélection du modèle : Choisissez le bon modèle en fonction des caractéristiques de vos données. Utilisez des modèles plus simples comme ARIMA pour les données non saisonnières et SARIMA pour les données saisonnières.
4. Validation croisée : Utilisez des techniques de validation croisée pour évaluer les performances de vos modèles. L'origine des prévisions et le fractionnement des séries temporelles sont des méthodes courantes.
5. Analyse des résidus : Après avoir ajusté votre modèle, analysez les résidus pour vous assurer qu'il s'agit de bruit blanc. Cela indique que le modèle a capturé tous les modèles sous-jacents des données.
6. Mises à jour régulières : Les données des séries temporelles peuvent changer au fil du temps. Il est donc essentiel de mettre régulièrement à jour vos modèles avec de nouvelles données

pour maintenir leur précision.

7. Automatiser lorsque c'est possible : Utilisez des outils et des scripts automatisés pour rationaliser votre flux de travail, en particulier pour le prétraitement des données et l'évaluation des modèles.

5 Les pièges les plus fréquents Common pitfalls

1. Surajustement : La création d'un modèle trop complexe peut entraîner un surajustement, c'est-à-dire que le modèle donne de bons résultats sur les données d'apprentissage, mais de mauvais résultats sur les nouvelles données. Pour éviter cela, utilisez des modèles plus simples et des techniques de régularisation.
2. Ignorer la saisonnalité : Ne pas tenir compte des tendances saisonnières peut conduire à des prévisions inexactes. Vérifiez toujours la saisonnalité de vos données et utilisez des modèles appropriés tels que SARIMA si nécessaire.
3. Négliger les diagnostics du modèle : L'omission de l'analyse des résidus et d'autres diagnostics peut conduire à des résultats trompeurs. Effectuez toujours des diagnostics approfondis pour valider votre modèle.
4. Négliger les diagnostics du modèle : L'omission de l'analyse des résidus et d'autres diagnostics peut conduire à des résultats trompeurs. Effectuez toujours des diagnostics approfondis pour valider votre modèle.
5. Data Leakage, Fuite de données : Veiller à ce que les données futures n'influencent pas l'apprentissage du modèle. Cela peut conduire à des estimations de performances trop optimistes. Divisez correctement vos données en ensembles de formation et de test. 6.Des modèles statiques pour des données dynamiques : Les modèles statiques peuvent devenir obsolètes lorsque de nouvelles données sont disponibles. Mettez régulièrement vos modèles à jour pour les adapter aux nouvelles tendances et aux nouveaux modèles.

En suivant ces bonnes pratiques et en évitant les pièges les plus courants, vous pouvez améliorer la précision et la fiabilité de vos prévisions de séries temporelles. N'oubliez pas que la clé d'une analyse réussie des séries temporelles réside dans une compréhension approfondie de vos données et dans une approche disciplinée de l'élaboration et de l'évaluation des modèles.

[ ]:

[ ]:

[ ]:

[ ]: