Comment prévoir les ventes avec Python à l'aide du modèle SARIMA

Un guide pas à pas de statistiques et de prévisions python-série chronologique

Avez-vous déjà imaginé prédire l'avenir? Nous n'y sommes pas encore, mais les modèles de prévision (avec un niveau d'incertitude) nous donnent une excellente orientation pour planifier nos activités avec davantage d'assurance lorsque nous nous tournons vers l'avenir. Dans cet article, nous présenterons une approche permettant de prévoir des séries chronologiques de ventes dans l'industrie automobile à l'aide du modèle SARIMA.

SARIMA est utilisé pour les séries non stationnaires, c'est-à-dire lorsque les données ne fluctuent pas autour de la même moyenne, de la variance et de la co-variance. Ce modèle peut identifier les tendances et la saisonnalité, ce qui le rend si important. SARIMA comprend d'autres modèles de prévision:

AR: Modèle régressif automatique (peut être une régression simple, multiple ou non linéaire)

MA: Modèle de moyennes mobiles. Les modèles à moyenne mobile peuvent utiliser des facteurs de pondération, où les observations sont pondérées par un facteur de compensation (pour les données les plus anciennes de la série) et avec un poids plus élevé pour les observations les plus récentes.

La composition de AR et MA ne porte pas ensemble la ARMA modèle, mais ce modèle est utilisé uniquement pour les séries stationnaires (moyenne, constante de variance dans le temps).

Si la série a une tendance, il faudra utiliser le ARIMA modèle.
ARIMA est utilisé pour les séries non stationnaires. Dans ce modèle, une étape de différenciation I (d) est utilisée pour éliminer la non-stationnarité.
L'élément intégré «I» pour la différenciation permet à la méthode de prendre en charge les séries chronologiques avec tendance. Mais toujours ce modèle n'identifie pas la saisonnalité.

Finalement nous arrivons au SARIMA modèle, qui a une corrélation saisonnière et peut identifier le caractère saisonnier de la série chronologique. Maintenant, nous pouvons passer aux codes!

Nous allons utiliser un ensemble de données sur les ventes de véhicules automobiles que vous pouvez télécharger ici.

```
Avertissements d'importation
importer itertools
importer numpy en tant que np
importer matplotlib.pyplot en tant que plt
warnings.filterwarnings ("ignore")
plt.style.use ('cinq sur trente')
importer des pandas en tant que pd
importer statsmodels.api en tant que sm
importer matplotlib
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'G'df = pd.read_excel ("Retail2.xlsx")
```

Cette étape consiste uniquement à importer les bibliothèques, telles que numpy, pandas, matplotlib et statsmodels, c'est-à-dire la bibliothèque contenant le modèle SARIMA et d'autres fonctionnalités de statistiques. Cette partie du code est utilisée pour configurer les graphiques de matplotlib.

Le jeu de données et le code d'origine sont un peu complexes, mais pour simplifier les choses, le fichier disponible au téléchargement ne contient que des colonnes de date et de vente afin d'éviter le prétraitement des données.

```
y = df.set_index (['Date'])
y.head (5)
```

La commande set_index définit la date de la colonne comme un index et l'en-tête imprime les 5 premières lignes du jeu de données.

```
y.plot (figsize = (19, 4))
plt.show ()
```

En analysant le graphique, nous pouvons observer que la série chronologique présente un modèle de saisonnalité. Octobre a un pic de ventes, au moins pour les 3 dernières années. Il y a aussi une tendance à la hausse au fil des ans.

```
de pylab import rcParams
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose (y, model = 'additive')
fig = decomposition.plot ()
plt.show ()
```

En utilisant la commande "sm.tsa.seasonal_decompose" de la bibliothèque pylab, nous pouvons décomposer la série temporelle en trois composants distincts: tendance, saisonnalité et bruit.

Utilisons SARIMA. La notation des modèles est `SARIMA (p, d, q). (P, D, Q) m`. Ces trois paramètres prennent en compte la saisonnalité, la tendance et le bruit dans les données

```
p = d = q = étendue (0, 2)
pdq = list (itertools.product (p, d, q))
saisonnier_pdq = [(x[0], X[1], X[2], 12) pour x dans la liste (itertools.product (p, d,
q))]
print ('Exemples de paramètres pour SARIMA ...')
print ('SARIMAX: {} x {}'. format (pdq[1], saisonnier_pdq[1]))
print ('SARIMAX: {} x {}'. format (pdq[1], saisonnier_pdq[2]))
print ('SARIMAX: {} x {}'. format (pdq[2], saisonnier_pdq[3]))
print ('SARIMAX: {} x {}'. format (pdq[2], saisonnier_pdq[4]))
```

Exemples de paramètres pour SARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)

```
pour param en pdq:
pour param_seasonal dans season_pdq:
essayer:
mod = sm.tsa.statespace.SARIMAX (y, ordre = param, Season_order = param_seasonal,
enforce_stationarity = False, enforce_invertibility = False)
résultats = mod.fit ()
print ('ARIMA {} x {} 12 - AIC: {}'. format (param, param_seasonal, results.aic))
sauf:
continuer
```

ARIMA (0, 0, 0) x (0, 0, 1, 12) 12 - AIC: 410,521537786262
ARIMA (0, 0, 0) x (1, 0, 0, 12) 12 - AIC: 363.03322198787765
ARIMA (0, 0, 0) x (1, 0, 1, 12) 12 - AIC: 348.13731052896617
ARIMA (0, 0, 0) x (1, 1, 0, 12) 12 - AIC: 237.2069523573001
ARIMA (0, 0, 1) x (0, 1, 1, 12) 12 - AIC: 1005.9793011993983
ARIMA (0, 0, 1) x (1, 0, 0, 12) 12 - AIC: 364,8331172721984
ARIMA (0, 0, 1) x (1, 0, 1, 12) 12 - AIC: 341.5095766512678
ARIMA (0, 0, 1) x (1, 1, 0, 12) 12 - AIC: 239.13525566698246
ARIMA (0, 0, 1) x (1, 1, 1, 12) 12 - AIC: 223,43134436185036

Selon Peterson, T. (2014), le **AIC** (Critère d'information Akaike) est un estimateur de la qualité relative de **statistique** modèles pour un ensemble de données donné. Étant donné une collection de modèles pour les données, **AIC** estime la qualité de chaque modèle par rapport à chacun des autres modèles. Le bas `AIC` valoriser le mieux. Notre production suggère que `SARIMAX (0, 0, 1) x (1, 1, 1, 12)` avec `AIC` la valeur de 223,43 est la meilleure combinaison, nous devrions donc considérer cette option comme optimale.

```
mod = sm.tsa.statespace.SARIMAX (y,
ordre = (0, 0, 1),
Season_order = (1, 1, 1, 12),
enforce_stationarity = False,
enforce_invertibility = False)
résultats = mod.fit ()
print (results.summary (). tables[1])
```

dans la commande "mod = sm.tsa.statespace.SARIMAX", nous devons définir la combinaison choisie.

```
results.plot_diagnostics (figsize = (18, 8))
plt.show ()
```

Avec le diagnostic ci-dessus, nous pouvons visualiser des informations importantes comme la distribution et la fonction de corrélation automatique ACF (corrélogramme). Valeurs à la hausse, le «0» présente une certaine corrélation entre les données de la série temporelle. Les valeurs proches de «1» démontrent la plus forte corrélation.

```
pred = results.get_prediction (start = pd.to_datetime ('2018-06-01'), dynamic = False)
pred_ci = pred.conf_int ()
ax = y['2015':].plot (label = 'observé')
pred.predicted_mean.plot (ax = ax, label = 'Prévisions à un pas d'avance', alpha = .7,
figsize = (14, 4))
ax.fill_between (pred_ci.index,
pred_ci.iloc[:, 0],
pred_ci.iloc[:, 1], couleur = 'k', alpha = .2)
ax.set_xlabel ('Date')
ax.set_ylabel ('Retail_sold')
plt.legend ()
plt.show ()
```

Cette étape consiste à comparer les valeurs vraies avec les prévisions. Nos prévisions correspondent très bien aux vraies valeurs. La commande «pred = results.get_prediction (start = pd.to_datetime (' 2018-06-01 ')» détermine la période que vous souhaitez prévoir en comparant les données réelles.

```
y_forecasted = pred.predicted_mean
y_truth = y['2018-06-01':]
mse = ((y_forecasted - y_truth) ** 2) .mean ()
print ('L'erreur quadratique moyenne est {}'. format (round (mse, 2)))
print ('L'erreur quadratique moyenne est {}'. format (round (np.sqrt (mse), 2)))
```

L'erreur quadratique moyenne est de 595,97.
L'erreur quadratique moyenne est de 24.41

Obs: Dans MSE et RMSE, les valeurs proches de zéro sont meilleures. Ils sont une mesure de la précision.

```
pred_uc = results.get_forecast (étapes = 12)
pred_ci = pred_uc.conf_int ()
ax = y.plot (label = 'observé', figsize = (14, 4))
pred_uc.predicted_mean.plot (ax = ax, label = 'Prévisions')
ax.fill_between (pred_ci.index,
pred_ci.iloc[:, 0],
pred_ci.iloc[:, 1], couleur = 'k', alpha = .25)
ax.set_xlabel ('Date')
ax.set_ylabel ('Ventes')
plt.legend ()
plt.show ()
```

Ici, nous prévoyons les ventes pour les 12 prochains mois. Ce paramètre peut être modifié dans la ligne "pred_uc = results.get_forecast (**étapes** = **12**) ”Du code.

```
y_forecasted = pred.predicted_mean
y_forecasted.head (12)
```

Cette étape indique les valeurs prédites du test que nous avons effectué auparavant.

```
y_truth.head (12)
```

Cette étape indique les valeurs de vérité de l'ensemble de données. Nous pouvons comparer les deux séries ci-dessus pour mesurer la précision du modèle.

```
pred_ci.head (24)
```

Dans le tableau ci-dessus, nous pouvons visualiser les valeurs inférieures et supérieures que le modèle indique comme limites pour la prévision.

```
prévision = pred_uc.predicted_mean
prévision.head (12)
```

Enfin, voici la prévision des ventes pour les 12 prochains mois!

On peut remarquer que les ventes augmentent avec le temps. Janvier et février sont les pires mois, comme les dernières années.

Ceci est juste une expérimentation avec un modèle statistique. Je recommande vraiment d'essayer les réseaux de neurones récurrents pour prévoir, mon conseil est simplement d'utiliser beaucoup plus de données. Mais c'est certainement le thème d'un autre article.

Référence :

Guide de prévision des séries chronologiques avec ARIMA en Python 3