Что такое данные временных рядов?

Данные временного ряда (данные с отметками времени) представляют собой последовательность точек данных, проиндексированных во временном порядке. Временные метки — это данные, собранные в разные моменты времени.

Эти точки данных обычно состоят из последовательных измерений, сделанных из одного и того же источника в течение определенного интервала времени, и используются для отслеживания изменений во времени.

Date	Spending
2004-01-01	7987.4
2004-02-01	8019.8
2004-03-01	8076.0
2004-04-01	8088.6
2004-05-01	8163.2
2004-06-01	8147.2
2004-07-01	8218.9
2004-08-01	8253.1
2004-09-01	8321.1
2004-10-01	8374.6

Каково использование данных временных рядов?

Используя данные временных рядов, мы можем найти закономерности в данных, а затем их можно использовать для прогнозирования/прогноза значений любой заданной переменной.

Это тоже имеет свои ограничения, если данная переменная сильно зависит от каких-либо внешних факторов, наша Модель не даст наилучших результатов.

Однако прогнозирование временных рядов является важной областью машинного обучения, поскольку существует множество задач прогнозирования, включающих временной компонент.

1. Получение данных временных рядов в правильном формате

Мы будем работать с набором данных о личных расходах, в котором есть личные расходы мужчины с 01.01.2004 по 01.01.2007, где данные собираются периодически 1-го числа каждого месяца.

Передайте переменную с данными Timestamp в index_col и установите parse_date=True , чтобы считать данные в правильном формате.
Проверьте частоту нашего набора данных (может быть ежедневно, ежемесячно, ежегодно и т. д.) и соответственно установите частоту индекса.

```
data = pd.read_csv("PCEPersonalSpending.csv",parse_dates=True,index_col="Date")
data.index.freq="MS" # As Our data Is monthly Spend Data.

data.head(10)
```

Date	Spending
2004-01-01	7987.4
2004-02-01	8019.8
2004-03-01	8076.0
2004-04-01	8088.6

2. EDA

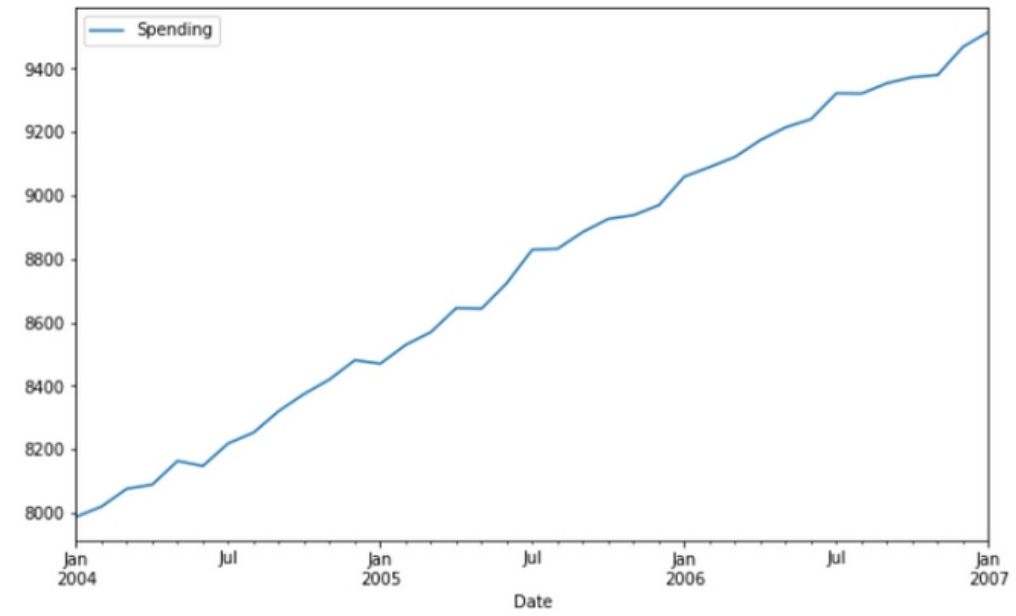
У нас есть месячные данные, поэтому, поскольку у нас всего 37 строк, общие данные, которые у нас есть, относятся к 3 годам и 1 месяцу (с 01.01.2004 по 01.01.2007).

```
data.shape
```

```
(37, 1)
```

Постройте данные, чтобы получить представление: на этом графике мы видим, что в данных явно прослеживается восходящий тренд, а также некоторая сезонность. Эти два термина очень важны в анализе временных рядов, поэтому мы увидим, что они на самом деле означают.

```
data.plot(figsize=(10,6));
```



Важные концепции

1. Тренд: тренд показывает общую тенденцию данных к увеличению или уменьшению в течение длительного периода времени. Тренд – это плавная, общая, долгосрочная, усредненная тенденция. Не всегда необходимо, чтобы увеличение или уменьшение было в одном и том же направлении в течение данного периода времени. 3 типа тренда: 1. Восходящий 2. Нисходящий 3. Горизонтальный/Стационарный
2. Сезонность. Сезонность — это характеристика временного ряда, в котором данные претерпевают регулярные и предсказуемые изменения, повторяющиеся каждый календарный год. Любые предсказуемые колебания или закономерности, которые повторяются или повторяются в течение одного года, называются сезонными.
3. Стационарность: говорят, что данные временного ряда являются стационарными, если их статистические свойства, такие как среднее значение и дисперсия, остаются постоянными во времени. Большинство моделей временных рядов работают в предположении, что данные стационарны, поэтому, если наши данные нестационарны, нам придется преобразовать их в стационарные данные. Еще одним преимуществом преобразования данных в стационарные является то, что теории, связанные со стационарными данными, являются более зрелыми и простыми в реализации.

Глядя на график выше, мы можем сказать, что в наших данных есть четкий восходящий тренд, а также некоторая сезонность. Но это может быть не так просто для всех наборов данных. Поэтому мы можем использовать сезонное_декомпозирование, чтобы узнать, есть ли у наших данных сезонность, тренд.

Сезонное разложение

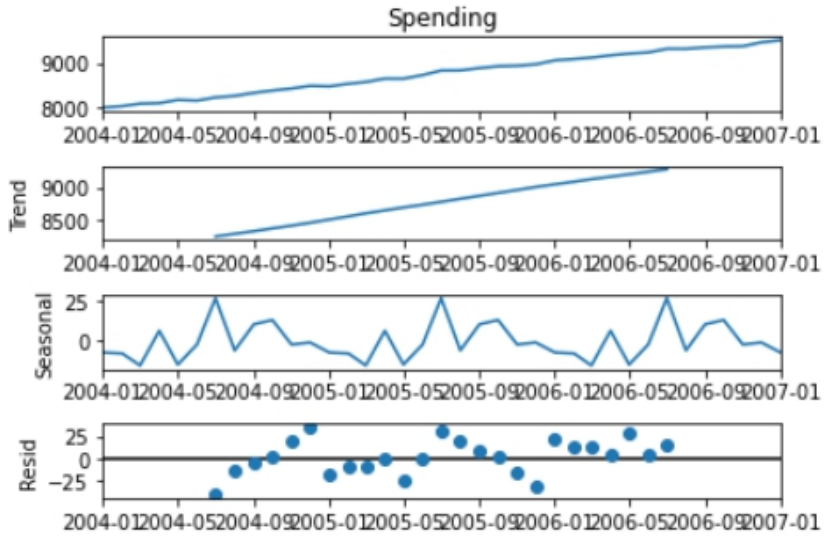
Seasonal Decompose вернет 4 вещи:

1. Наблюдалось (Исходные данные)
2. Тренд (Общий тренд)
3. Сезонные данные
4. Ошибка/Остаток (данные, которые нельзя объяснить ни сезонностью, ни тенденцией)

Есть 2 варианта сезонного разложения:

1. Аддитивный (когда тренд более линейный, сезонность и тренд кажутся постоянными)
2. Мультипликативный (тренд более нелинейный)

```
from statsmodels.tsa.seasonal import seasonal_decompose
results = seasonal_decompose(data["Spending"], model="Additive")
results.plot();
```



На этом изображении мы видим, что сезонное разложение ясно говорит нам о том, что в данных есть некоторый восходящий тренд и некоторая сезонность.

3.Проверьте стационарность.

Функция Дики-Фуллера: проверьте, являются ли данные стационарными.

Он возвращает P-значение, если p<0,5: данные стационарны, p>0,5: данные нестационарны.

```
from statsmodels.tsa.stattools import adfuller

def adf_test(series,title=''):

    print('Dickey-Fuller Test :')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles differenced data

    labels = ['ADF test','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype: float64"
    print("*"*100)
    if result[1] <= 0.05:
        print("Data has no unit root and is stationary")
    else:
        print("Data has a unit root and is non-stationary")
```

Итак, после проверки с помощью этой функции мы пришли к выводу, что данные нестационарны.

```
adf_test(data)
```

```
Dickey-Fuller Test :
ADF test                -0.588232
p-value                  0.873619
# lags used              2.000000
# observations          34.000000
critical value (1%)     -3.639224
critical value (5%)     -2.951230
critical value (10%)    -2.614447
****************************************
Data has a unit root and is non-stationary
```

4. Выберите правильную модель и сделайте данные стационарными.

ARIMA: авторегрессивная интегрированная скользящая средняя.

ARIMA — одна из лучших моделей для прогнозирования, подробности здесь.

1. Чтобы эффективно использовать ARIMA, нам нужно понимать стационарность наших данных.
2. ЕСЛИ мы определили, что данные не являются стационарными, нам нужно будет сделать их стационарными, чтобы предсказать их.
    - Один из простых способов сделать это — использовать разность.
3. После того, как данные станут стационарными, нам нужно будет выбрать параметры p, d, q. (p : количество лагов в AR, d : степень различия, q : порядок модели MA)

Здесь мы можем использовать функцию auto_arima, чтобы найти наилучшие значения p, d, q. мы будем.

auto_arima():

```
Automatically discover the optimal order for an ARIMA model.

The auto-ARIMA process seeks to identify the most optimal
parameters for an `ARIMA` model
```

Итак, из auto_arima() мы получили лучшие параметры: p=1, d=1, q=1.

```
from pmdarima import auto_arima

result = auto_arima(data["Spending"],seasonal=True, trace=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=351.132, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=346.030, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=344.364, Time=0.03 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=388.676, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=346.246, Time=0.08 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=346.298, Time=0.05 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=347.760, Time=0.10 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=385.138, Time=0.02 sec

Best model:  ARIMA(0,1,1)(0,0,0)[0] intercept
Total fit time: 0.552 seconds
```

Теперь нам не нужно дифференцировать данные, чтобы сделать их стационарными, передача d=1 в ARIMA продифференцирует их на 1 для нас.

5. Создайте модель ARIMA и предскажите тестовые данные.

В ARIMA() мы передаем order=(p, d, q), мы передали наши p, d, q, которые равны 0, 1, 1.

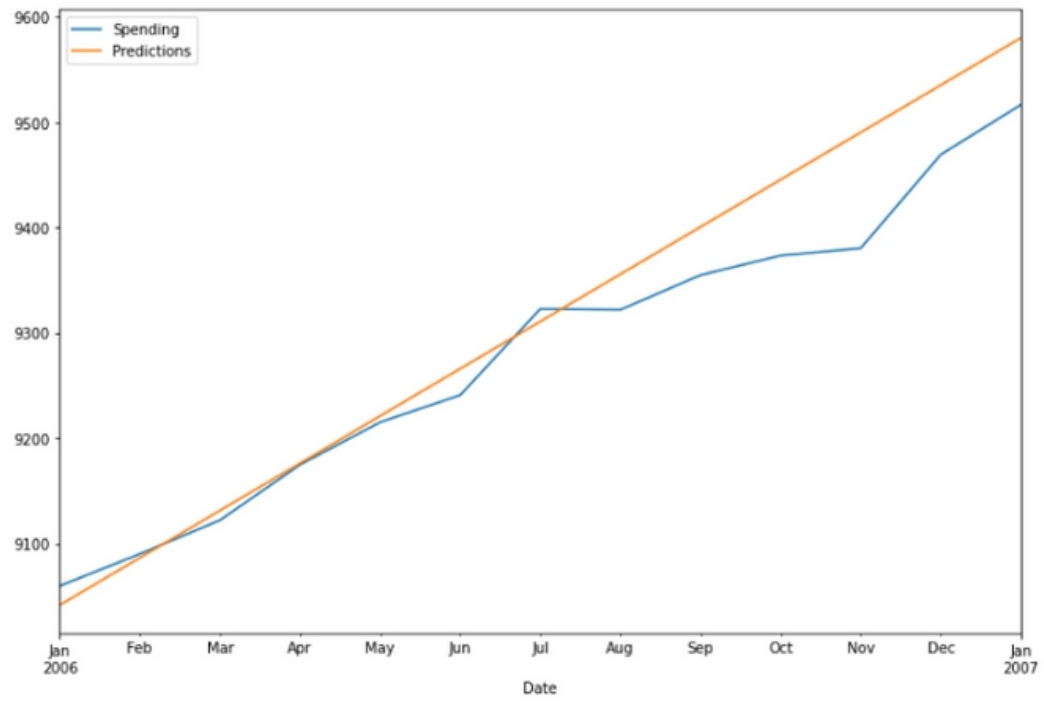
```
from statsmodels.tsa.arima_model import ARIMA

#Build the model
model = ARIMA(Train["Spending"],order=(0,1,1))

#Fit the Model
result = model.fit()

# Predict the test data
start = len(Train) #Where we want to start the prediction
end = len(data)-1  #Where we want to end the prediction

# typ="levels", use this when we difference the Data.As we want the prediction on original data and not th Differenced Data.
pred = result.predict(start=start,end=end,typ="levels", dynamic=False).rename("Predictions")
```

6. Постройте прогнозы и оцените результаты.

Это результат, который мы получили, результат/RMSE достойный, учитывая, что мы использовали простую модель ARIMA, а наши обучающие данные невелики.

```
# Plot Prediction vs Test
Test["Spending"].plot(figsize=(12,8),legend=True)
pred.plot(legend=True)
```

```
<AxesSubplot:xlabel='Date'>
```



```
from statsmodels.tools.eval_measures import rmse

error = rmse(Test['Spending'], pred)
print(f'ARIMA(0,1,1) RMSE Error: {error:11.10}')
```

```
ARIMA(0,1,1) RMSE Error: 48.11754952
```

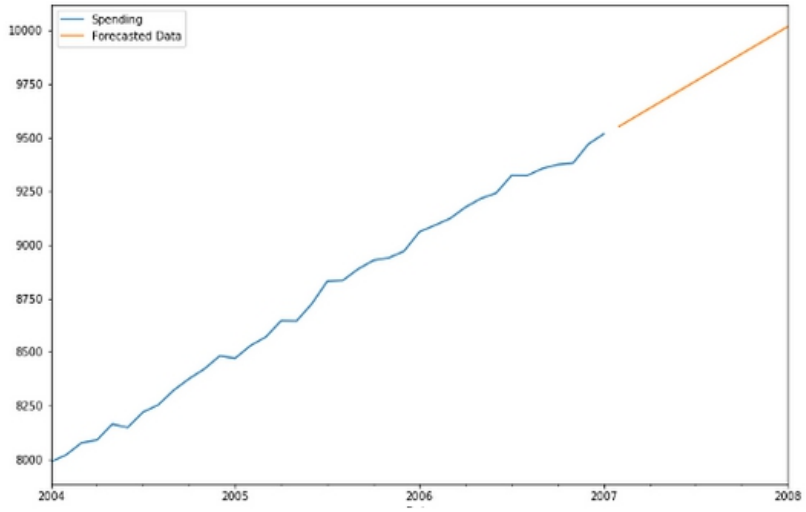
RMSE for the Model.

7. Прогнозируйте будущие данные.

Теперь мы будем тренироваться на всех данных (3 года) и прогнозировать на 1 год вперед.

```
# ReTrain the model on the whole dataset

model = ARIMA(data["Spending"], order=(0,1,1))
result=model.fit()

# forecast using the Model
forecast = result.predict(start = len(data), end = len(data)+11 , typ="levels",dynamic=False).rename("Forecasted Data")


data["Spending"].plot(figsize=(12,8),legend=True)
forecast.plot(legend=True)
```

```
<AxesSubplot:xlabel='Date'>
```



Это для начала и просто для того, чтобы почувствовать прогнозирование временных рядов с использованием простой модели ARIMA, я приведу еще несколько примеров, где мы используем более сложные модели, такие как SARIMA, SARIMAX, а также для использования нескольких столбцов для прогнозирования.