Návrh systémů IoT

4. Zpracování časových řad.

Stanislav Vítek
Katedra radioelektroniky
České vysoké učení technické v Praze

Časové řady

- Časovou řadu lze považovat za uspořádanou posloupnost hodnot proměnné ve (ideálně) stejně dlouhých časových intervalech.
- Experimentální data v podobě časových řad získaných pozorováním reálných dynamických systémů mohou být využita pro konstrukci matematických modelů těchto systémů.
- K modelování takových dat lze použít analýzu časových řad (TSA).
- TSA zohledňuje skutečnost, že datové body pořízené v průběhu času mohou mít vnitřní strukturu (např. autokorelaci, trend nebo sezónní výkyvy)

Co lze provedením TSA získat?

- **Popis** Identifikujte vzorce v korelovaných datech, jako jsou trendy a sezónní výkyvy.
- **Porozumění** Tyto vzorce/modely umožňují rozumět operacím a vazbám, které ovlivňují vznik hodnot časových řad.
- **Predikce** Při modelování dat lze získat přesné předpovědi budoucích (krátkodobých) trendů.
- **Intervenční analýza** Lze zkoumat, jak (jednotlivé) události ovlivnily časové řady.
- **Kontrola kvality** Odchylky v časové řadě mohou naznačovat problémy v procesu, který se v datech odráží.

Oblasti uplatnění analýzy časových řad

- Ekonomické prognózy
- Prognózování prodeje
- Rozpočtová analýza
- Analýza akciového trhu
- Odhady výnosů
- Řízení procesů a kvality
- Inventarizační studie
- Odhady pracovního zatížení
- Studie užitečnosti
- Analýza sčítání lidu
- Strategické plánování pracovních sil

Nástroje pro analýzu (1/2)

Řekněme, že máme časovou řadu uloženou v CSV / SQL / NoSQL. Chceme:

- Umět data načíst
- Vypočítat statistické údaje a odpovědět na otázky týkající se dat, např.
  - Jaký je průměr, medián, maximum nebo minimum každého sloupce?
  - Souvisí sloupec A se sloupcem B?
  - Jak vypadá rozložení dat ve sloupci C?
- Vyčistit data -- odstranění chybějících hodnot a filtrování řádků nebo sloupců podle určitých kritérií.
- Vizualizovat data -- vykreslení sloupců, čár, histogramů, bublin, ...
- Uložit vyčištěná a transformovaná data zpět do CSV (nebo databáze).

Nástroje pro analýzu (2/2)

- Co tak třeba Matlab?
- Nebyl by lepší třeba Python?
- Pandas
  - Postaveno na základech knihovny NumPy
  - Vizualizace v Matplotlib
  - Algoritmy strojového učení ze Scikit-learn

Tabulky

- Základní datový typ, který Pandas nabízí, je tabulka - DataFrame
- Jednotlivé záznamy jsou v ní uvedeny jako řádky
- Nejpoužívanější způsob, jak naplnit první DataFrame, je načtení ze souboru.
  - Funkce začínají **read_**, např. read_csv
  - Některé funkce potřebují další knihovny (viz dokumentace)

```
actors = pandas.read_csv('files/actors.csv', index_col=None)
```

- Z CSV se nepřenáší indexy, lze určit, který sloupec bude indexem

```
actors = pandas.read_csv('files/actors.csv', index_col=0)
```

Čtení zdrojových dat z databáze

- K SQL databázi je třeba se nejprve připojit

```
import sqlite3
import pandas as pd

con = sqlite3.connect("iot.db")
df = pd.read_sql_query("SELECT * FROM hodnoty", con)

# index lze nastavit samostatnym prikazem
df = df.set_index('id')
```

Vytváření tabulek ze složených datových typů

- Tabulku lze vytvořit ze seznamu seznamů:

```
items = pandas.DataFrame([
    ["Book", 123],
    ["Computer", 2185],
])
```

- Nebo seznamu slovníků:

```
items = pandas.DataFrame([
    {"name": "Book", "price": 123},
    {"name": "Computer", "price": 2185},
])
```

Informace o tabulkách

- Základní informace o tabulce se dají získat metodou info

```
actors.info()
```

- Tabulka má
  - 6 řádků indexovaných pomocí automaticky vygenerovaného indexu
  - 3 sloupce: jeden s objekty, jeden s **int64** a jeden s **bool**
- Datové typy (dtypes) se doplnily automaticky podle hodnot ve sloupcích tabulky.

Sloupce

- Sloupec, neboli Series, je druhý základní datový typ v Pandas.
- Obsahuje sérii hodnot, jako seznam, ale navíc má jméno, datový typ a *index*, který jednotlivé hodnoty pojmenovává.
- Sloupce (a informace o nich) se dají získat vybráním z tabulky:

```
birth_years = actors['birth']

type(birth_years)

birth_years.name

birth_years.index

birth_years.dtype
```

Operace se sloupci

- Základní aritmetické operace se sloupcem a skalární hodnotou nebo porovnání provedou danou operaci nad každou hodnotou ve sloupci.
- Výsledek je nový sloupec

```
ages = 2023 - birth_years

century = birth_years // 100 + 1

birth_years > 1940

birth_years == 1940

# operace s iterovatelnym datovym typem
actors['name'] + [' (1)', ' (2)', ' (3)', ' (4)', ' (5)', ' (6)']
```

Výběr prvků

- Ze sloupců lze vybírat prvky či podsekvence podobně jako třeba ze seznamů

```
birth_years[2]

birth_years[2:-2]
```

- Výběr odpovídající podmínce

```
birth_years[birth_years > 1940]

birth_years[(birth_years > 1940) & (birth_years < 1943)]
```

Agregační funkce

- Vestavěné funkce, základní statistika

```
print('Součet: ', birth_years.sum())

print('Průměr: ', birth_years.mean())

print('Medián: ', birth_years.median())

print('Počet unikátních hodnot: ', birth_years.nunique())

print('Koeficient špičatosti: ', birth_years.kurtosis())
```

- Možnost spouštění vlastních funkcí

```
actors['name'].apply(lambda x: ''.join(reversed(x)))

actors['alive'].apply({True: 'alive', False: 'deceased'}.get)
```

Vybírání prvků ze sloupců

- Prvky ze sloupců jdou vybírat jako u seznamů.
- Ale z tabulek v Pandas jde vybírat spoustou různých způsobů.
- Tradiční hranaté závorky plní několik funkcí najednou, takže někdy není na první pohled jasné, co jaké indexování znamená.

```
actors['name']  # Jméno sloupce

actors[1:-1]  # Interval řádků

actors[['name', 'alive']]  # Seznam sloupců
```

Indexer loc

- Indexer **loc** zprostředkovává řádky podle indexu, tedy hlaviček tabulky.
- Není to metoda, používají se s ním hranaté závorky.

```
actors.loc[2]
```

- Použijeme-li k indexování n-tici, prvním prvkem se indexují řádky a druhým sloupce

```
actors.loc[2, 'birth']
```

- Lze pracovat s intervaly

```
actors.loc[2:4, 'birth':'alive']
```

Indexer iloc

- Umí to samé co **loc**, jen nepracuje s klíčem, ale s pozicemi řádků či sloupců.

```
actors.iloc[0, 0]

actors.iloc[-1, 1]

actors.iloc[:, 0:1]
```

- Jak **loc** tak **iloc** fungují i na sloupcích (Series), takže se dají kombinovat:

```
actors.iloc[-1].loc['name']
```

Příklad 1

Data z databáze IMDB

Načtení a prohlížení dat

- Načtení dat

```
movies_df = pd.read_csv("files/imdb.csv", index_col="Title")
```

- Prvních 5 (nebo N) řádek tabulky

```
movies_df.head()
movies_df.head(10)
```

- Posledních 5 nebo N řádek tabulky

```
movies_df.tail()
movies_df.tail(10)
```

Hledání duplikátů, práce s daty

- V tabulce žádné duplikáty nejsou, tak nějaké vyrobíme

```
temp_df = movies_df.append(movies_df)
```

- Vyčištění duplikátů

```
temp_df = temp_df.drop_duplicates()
```

- Sloupce tabulky a jejich přejmenování

```
movies_df.columns
movies_df.rename(columns={
        'Runtime (Minutes)': 'Runtime',
        'Revenue (Millions)': 'Revenue_millions'
}, inplace=True)
# nebo treba pro prejmenovani vsech na lower case
movies_df.columns = [col.lower() for col in movies_df]
```

Chybějící data

- Zajímá nás, které řádky mají chybějící údaje
- Takových řádků se buď zbavíme, nebo tam nějaká data vložíme

```
# ktere prvky v DF jsou prázdné - vraci True/False
movies_df.isnull()
# agregovana funkce
movies_df.isnull().sum()
```

- Zahození **řádků** s prázdnými položkami

```
temp_df = movies_df.dropna()
```

- Zahození **sloupců** s prázdnými položkami

```
temp_df = movies_df.dropna(axis=1)
```

Doplnění chybějících dat

- Budeme pracovat se sloupcem **Revenue (Millions)**

```
revenue = movies_df['Revenue (Millions)']
```

- Tam, kde chybí data, doplníme střední hodnotu

```
revenue_mean = revenue.mean()
revenue.fillna(revenue_mean, inplace=True)
```

- Otestování počtu položek s chybějícími daty

```
movies_df.isnull().sum()
```

Statistiky

- Statistika přes celý *DataFrame*

```
movies_df.describe()
```

- Statistika vybraného sloupce

```
movies_df['Genre'].describe()
```

- Víme, kolik je tam unikátních žánrů. Co histogram?

```
movies_df['Genre'].value_counts().head(10)
```

- Korelace mezi sloupci

```
movies_df.corr()
```

Hledání v datech

- Výpis sloupců

```
subset = movies_df[['genre', 'rating']]
```

- Hledání hodnot v řádcích

```
prom = movies_df.loc["Prometheus"]
prom = movies_df.iloc[1]
```

- Slicing

```
movie_subset = movies_df.loc['Prometheus':'Sing']
movie_subset = movies_df.iloc[1:4]
```

- Podmíněný výběr

```
condition = (movies_df['director'] == "Ridley Scott")
movies_df[movies_df['rating'] >= 8.6].head(3)
```

Volání funkcí

- Definice hodnotící funkce

```
def rating_function(x):
        if x >= 8.0:
                return "good"
        else:
                return "bad"
```

- Aplikace funkce

```
movies_df["rating_category"] = movies_df["rating"].apply(rating_function)
```

- Lambda funkce

```
movies_df["rating_category"] = movies_df["rating"].apply(lambda x: 'good' if x >= 8.0 else 'bad')
```

Kreslení

```
import matplotlib.pyplot as plt
# nastavení fontu a velikosti okna
plt.rcParams.update({'font.size': 20, 'figure.figsize': (10, 8)})
# plot vykreslí data z tabulky oproti indexu
```

- Vztah mezi obratem a hodnocením filmu

```
movies_df.plot(kind='scatter', x='rating', y='revenue_millions', title='Revenue (millions) vs Rating');
```

- Histogram

```
movies_df['rating'].plot(kind='hist', title='Rating');
```

- Vlastnosti sloupce rating

```
movies_df['rating'].describe()
movies_df['rating'].plot(kind="box");
```

Příklad 2

E-shop

Zadání

- Cílem příkladu je vytvořit tabulku fiktivních prodejů v e-shopu, ve formátu jaký bychom mohli dostat z SQL databáze nebo datového souboru.
- Použijeme k tomu mimo jiné funkci date_range, která vytváří kalendářní intervaly.
- Zde, i v jiných případech, kdy je jasné, že se má nějaká hodnota interpretovat jako datum, nám Pandas dovolí místo objektů datetime zadávat data řetězcem:

Vytvoření dat

```
import itertools
import random
import pandas

random.seed(0)

months = pandas.date_range('2015-01', '2016-12', freq='M')

categories = ['Electronics', 'Power Tools', 'Clothing']

data = pandas.DataFrame(
  [
    {'month': a, 'category': b, 'sales': random.randint(-1000, 10000)}
    for a, b in itertools.product(months, categories) if random.randrange(20) > 0
  ])
```

Co se nám podařilo vygenerovat?

```
# Prvních pár řádků (dá se použít i např. head(10), bylo by jich víc)
data.head()

# Celkový počet řádků
len(data)

# Data ve  sloupci sales
data['sales'].describe()
```

- Pomocí set_index nastavíme, které sloupce budeme brát jako hlavičky:

```
indexed = data.set_index(['category', 'month'])
indexed.head()
```

Manipulace s daty

- Budeme-li chtít z těchto dat vytvořit tabulku, která má v řádcích kategorie a ve sloupcích měsíce, můžeme využít metodu unstack, která "přesune" vnitřní úroveň indexu řádků do sloupců a uspořádá podle toho i data.

```
unstacked = indexed.unstack('month')
unstacked
```

- Teď je sloupcový klíč dvouúrovňový, ale úroveň **sales** je zbytečná. Můžeme se jí zbavit pomocí MultiIndex.droplevel.

```
unstacked.columns = unstacked.columns.droplevel()
unstacked
```

Analýza

- Kolik se celkem utratilo za elektroniku?

```
unstacked.loc['Electronics'].sum()
```

- Jak to vypadalo se všemi elektrickými zařízeními v třech konkrétních měsících?

```
unstacked.loc[['Electronics', 'Power Tools'], '2016-03':'2016-05']
```

- A jak se prodávalo oblečení?

```
unstacked.loc['Clothing']
```

Metody stack a unstack jsou sice asi nejužitečnější, ale stále jen jeden ze způsobů jak v Pandas tabulky přeskládávat. Další možnosti v dokumentaci.

Grafy

```
# Setup
import matplotlib.pyplot

# Plot
unstacked.loc['Clothing'].plot()
matplotlib.pyplot.show()
matplotlib.pyplot.savefig('graph.png')
```

- Jak se postupně vyvíjely zisky z oblečení?

```
# `.T` udělá transpozici tabulky (vymění řádky a sloupce)
# `cumsum()` spočítá průběžný součet po sloupcích
unstacked.T.fillna(0).cumsum().plot()
```

- Jak si proti sobě stály jednotlivé kategorie v březnu, dubnu a květnu 2016?

```
unstacked.loc[:, '2016-03':'2016-05'].plot.bar(legend=False)
```

Práce s časem v Pythonu

Nativní podpora v Pythonu

- Základní objekty jazyka Python pro práci s daty a časy se nacházejí ve vestavěném modulu datetime. Spolu s modulem dateutil jej můžete použít k rychlému provádění řady užitečných funkcí s daty a časy.
- Například můžete ručně sestavit datum pomocí typu datetime:

```
from datetime import datetime
datetime(year=2015, month=7, day=4)
```

```
from dateutil import parser
date = parser.parse("4th of July, 2015")
date.strftime('%A')
```

NumPy

- Nedostatky formátu data v jazyce Python inspirovaly tým NumPy k přidání sady nativních datových typů časových řad do NumPy.
- Datový typ numpy.datetime64 kóduje data jako 64bitová celá čísla, a umožňuje tak velmi kompaktní reprezentaci polí dat.
- Typ numpy.datetime64 vyžaduje velmi specifický vstupní formát:

```
import numpy
date = numpy.array('2015-07-04', dtype=numpy.datetime64)
```

- Jakmile však máme toto datum naformátované, můžeme s ním rychle provádět vektorové operace:

```
date + numpy.arange(12)
```

Pandas

- Pandas poskytuje objekt Timestamp, který kombinuje snadné použití datetime a dateutil s efektivním ukládáním a vektorovým rozhraním numpy.datetime64.
- Ze skupiny těchto objektů Timestamp může Pandas vytvořit DatetimeIndex, který lze použít k indexování dat v Series nebo DataFrame.

```
import pandas
date = pandas.to_datetime("4th of July, 2015")

date + pandas.to_timedelta(np.arange(12), 'D')
```

Časové řady v Pandas - Indexování podle času

- Nástroje časových řad Pandas se stanou skutečně užitečnými, když začnete indexovat data podle časových značek.
- Můžeme například vytvořit objekt Series, který obsahuje časově indexovaná data:

```
index = pandas.DatetimeIndex(['2014-07-04', '2014-08-04',
                              '2015-07-04', '2015-08-04'])
data = pandas.Series([0, 1, 2, 3], index=index)
```

- Nyní, když máme tato data v sérii, můžeme využít některý ze vzorů indexování sérií

```
data['2014-07-04':'2015-07-04']

data['2015']
```

Datové struktury pro podporu časových řad

- Pro časová razítka poskytuje Pandas typ Timestamp.
  - Náhrada nativního typu datetime, založen na efektivnějším datovém typu numpy.datetime64. Přidružená indexová struktura je DatetimeIndex.
- Pro časové periody poskytuje Pandas typ Period.
  - Kóduje interval s pevnou frekvencí, přidružená indexová struktura je PeriodIndex.
- Pro časové delty nebo doby trvání poskytuje Pandas typ Timedelta.
  - Náhradoa nativního typu datetime.timedelta, přidružená indexová struktura je TimedeltaIndex.

Timestamp a DatetimeIndex

- Ačkoli tyto instance tříd lze vyvolat přímo, častěji se používá Pandas funkce to_datetime(), která dokáže analyzovat širokou škálu formátů.
  - pokud má funkce jako argument datum, získáte instanci Timestamp
  - při předání řady dat získáte instance DatetimeIndex

```
dates = pandas.to_datetime([datetime(2015, 7, 3), '4th of July, 2015',
                            '2015-Jul-6', '07-07-2015', '20150708'])
```

- Jakýkoli DatetimeIndex lze převést na PeriodIndex pomocí funkce to_period() s přidáním kódu frekvence
  - zde použijeme 'D' pro označení denní frekvence:

```
dates.to_period('D')
```

Pravidelné časové řady

- Aby bylo vytváření pravidelných posloupností dat pohodlnější, nabízí Pandas pro tento účel několik funkcí:
  - date_range() pro časové značky,
  - period_range() pro období,
  - timedelta_range() pro časové delty.
- Argumenty funkce date_range() jsou počáteční datum, koncové datum a nepovinný frekvenční kód a vytváří pravidelnou posloupnost dat.
- Ve výchozím nastavení je frekvence jeden den:

```
pandas.date_range('2015-07-03', '2015-07-10')
```

Pravidelné časové řady

- Alternativně lze rozsah dat zadat nikoli pomocí počátečního a koncového bodu, ale pomocí počátečního bodu a několika období:

```
pandas.date_range('2015-07-03', periods=8)
```

- Rozteč lze upravit změnou argumentu **freq**, který má výchozí hodnotu **D**. Například zde vytvoříme rozsah hodinových časových značek:

```
pandas.date_range('2015-07-03', periods=8, freq='H')
```

- Pro vytváření pravidelných posloupností hodnot Period nebo Timedelta jsou užitečné velmi podobné funkce period_range() a timedelta_range(). Zde je několik měsíčních period:

```
pandas.period_range('2015-07', periods=8, freq='M')
```

Frekvence a offsety

- Základním prvkem těchto nástrojů časových řad Pandas je koncept posunu frekvence nebo data. Stejně jako jsme si výše ukázali kódy **D** (den) a **H** (hodina), můžeme pomocí dalších kódů a jejich kombinacemi zadat libovolný frekvenční odstup.

```
pandas.timedelta_range(0, periods=9, freq="2H30T")
```

- Řada těchto zkrácených kódů se odkazuje na pracovní cykly. Například offset pracovního dne můžeme vytvořit přímo následujícím způsobem:

```
from pandas.tseries.offsets import BDay
pandas.date_range('2015-07-01', periods=5, freq=BDay())
```

Příklad 3

Vývoj ceny akcií

Vstupní data

- Indexovananá data mají obecně řadu výhod - automatické zarovnání během operací, intuitivní řezání dat a přístup k nim atd.)
- Pandas poskytuje několik dalších operací specifických pro časové řady.
- V příklad použijeme data cen akcií, dostupné v modulu pandas-datareader

```
# pip install pandas-datareader yfinance
from pandas-datareader import data
import yfinance as yf

yf.pdr_override()

goog = data.DataReader(('GOOG', start='2004-01-01', end='2016-12-31')
goog.head()
```

Vizualizace

- Pro jednoduchost budeme pracovat pouze s hodnotami při zavření burzy

```
goog = goog['Close']
```

- Pro vizualizaci využijeme modul seaborn

```
import matplotlib.pyplot as plt
import seaborn

seaborn.set()

goog.plot();
```

Převzorkování a konverze frekvencí

- Převzorkování na vyšší nebo nižší frekvenci lze u časových řad provést pomocí metody resample() nebo mnohem jednodušší metody asfreq().
- Základní rozdíl mezi nimi spočívá v tom, že resample() je v podstatě agregace dat, zatímco asfreq() je v podstatě výběr dat.

```
goog.plot(alpha=0.5, style='-')

#  BA - Business year end
goog.resample('BA').mean().plot(style=':')
goog.asfreq('BA').plot(style='--');

plt.legend(['input', 'resample', 'asfreq'], loc='upper left');
```

Časové posuny

- Další běžnou operací specifickou pro časové řady je posun dat v čase.
- Pandas má pro tento výpočet metodu shift()
- Posun se zadává v násobcích frekvence.
- **Příklad:** posun o 900 dní:

```
fig, ax = plt.subplots(2, sharey=True)

# apply a frequency to the data
goog = goog.asfreq('D', method='pad')

goog.plot(ax=ax[0])
goog.shift(900).plot(ax=ax[1])
```

Klouzavý průměr

- Klouzavé statistiky jsou třetím typem operací specifických pro časové řady. Ty lze provádět metodou rolling() objektů Series a DataFrame.
- Toto klouzavé zobrazení standardně zpřístupňuje řadu agregačních operací.
- **Příklad:** jednoroční klouzavý průměr a směrodatná odchylka cen akcií Google:

```
rolling = goog.rolling(365, center=True)

data = pd.DataFrame({'input': goog,
                     'one-year rolling_mean': rolling.mean(),
                     'one-year rolling_std': rolling.std()})
ax = data.plot(style=['-', '--', ':'])
ax.lines[0].set_alpha(0.3)
```