CPSC 330 Lecture 20: Time series

Firas Moosvi

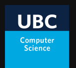
Lecture learning objectives 🎯

- Recognize when a problem is a time series problem and what makes time-ordered data unique.
- Apply proper train/test splits for time series.
- Engineer key time-based features
- Understand strategies for multi-step forecasting
- Describe trend and its impact on time series behavior

Recap: iClicker questions

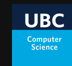
Select all of the following statements which are TRUE.

- a. In multinomial logistic regression, the model learns a separate weight vector and bias for each class.
- b. Neural networks are powerful models, so it's usually a good idea to start with them on any new machine learning problem.
- c. The main reason we add hidden layers is to allow the model to learn increasingly complex representations.
- d. Convolutional neural networks (CNNs) use filters that slide over the image to detect local patterns.
- e. Using a pre-trained network as a feature extractor typically requires less data than training a deep network from scratch.



Recap: What type of model would be appropriate?

Scenario	Model/Method
You have user–item ratings (e.g., movie ratings) and want to predict missing ratings.	?
You have a collection of documents without any labels and want to group them into themes.	?
You want to classify the emotion of a set of text messages, but you do not have any labeled data.	?
You have a small dataset with ~500 images containing pictures and names of 20 different Computer Science faculty members from UBC. Your goal is to develop a reasonably accurate multi-class classification model for this task.	?



Motivation

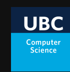
- **Time series** is a collection of data points indexed in time order.
- Time series is everywhere:
  - Physical sciences (e.g., weather forecasting)
  - Economics, finance (e.g., stocks, market trends)
  - Engineering (e.g., energy consumption)
  - Social sciences
  - Sports analytics

Loan default prediction (tabular data)

You work for a financial institution and have a dataset where each row represents a customer applying for a loan. What type of model would you use?

customer_id	income_k	credit_utilization	late_payments	employment_
1	95	22	0	9
2	45	78	3	2
3	120	30	1	7
4	60	65	2	3
5	85	40	0	10
6	55	90	4	1
7	130	28	0	6
8	40	82	2	1

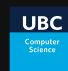
Rows are independent ⟶ order does not matter ⟶ time does not matter



Predict demand for bike rentals

Suppose you want to predict the bike rental demand for the next three-hour period at a station in New York City.

What features might help us make an accurate prediction?



citibike data

```
starttime
2015-08-01 00:00:00     3
2015-08-01 03:00:00     0
2015-08-01 06:00:00     9
2015-08-01 09:00:00    41
2015-08-01 12:00:00    39
2015-08-01 15:00:00    27
2015-08-01 18:00:00    12
2015-08-01 21:00:00     4
2015-08-02 00:00:00     3
2015-08-02 03:00:00     4
2015-08-02 06:00:00     6
2015-08-02 09:00:00    30
2015-08-02 12:00:00    46
2015-08-02 15:00:00    27
2015-08-02 18:00:00    28
2015-08-02 21:00:00     6
2015-08-03 00:00:00     3
2015-08-03 03:00:00     2
2015-08-03 06:00:00    21
2015-08-03 09:00:00     9
Freq: 3h, Name: one, dtype: int64
```

- Only feature: datetime
- The data is collected at regular intervals (every three hours)
- Target: rentals in the next 3-hour period
- Goal: Given past rental counts, predict the number of rentals at a specific future time.

Using only the tools in your current toolbox, what model would you choose, and what challenges might you run into?

Why different treatement?

```
starttime
2015-08-01 00:00:00     3
2015-08-01 03:00:00     0
2015-08-01 06:00:00     9
2015-08-01 09:00:00    41
2015-08-01 12:00:00    39
2015-08-01 15:00:00    27
2015-08-01 18:00:00    12
2015-08-01 21:00:00     4
2015-08-02 00:00:00     3
2015-08-02 03:00:00     4
2015-08-02 06:00:00     6
2015-08-02 09:00:00    30
2015-08-02 12:00:00    46
2015-08-02 15:00:00    27
2015-08-02 18:00:00    28
2015-08-02 21:00:00     6
2015-08-03 00:00:00     3
2015-08-03 03:00:00     2
2015-08-03 06:00:00    21
2015-08-03 09:00:00     9
Freq: 3h, Name: one, dtype: int64
```

- This type of data is distinctive because it is **inherently sequential**.
- The number of bikes available at a station at one point in time is often related to the number of bikes at earlier times.
- This is a **time-series forecasting** problem.

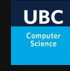
Today's lecture

- What is time series?
- How do we know a problem is a time series problem?
- Why do standard ML models struggle with time-dependent data?
- How can we adapt ML models to handle time series?

Models for time series

The ML models we've used so far **do not have a built-in concept of time**.
There are **two broad strategies** for modeling time series:

- Use models designed for sequential data which explicitly capture temporal dependencies (e.g., Hidden Markov Models, Transformer architectures etc.)
- **Use tabular ML models with engineered temporal features (e.g., Linear models, Random Forests, Gradient Boosted Trees)**



citibike data visualization

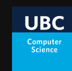
```
Start date: 2015-08-01 00:00:00, End date: 2015-08-31 21:00:00
```

Number of bike rentals over time for a selected bike station

Rentals: 0, 10, 20, 30, 40, 50, 60

Date: Sat 08-01, Sun 08-02, Mon 08-03, Tue 08-04, Wed 08-05, Thu 08-06, Fri 08-07, Sat 08-08, Sun 08-09, Mon 08-10, Tue 08-11, Wed 08-12, Thu 08-13, Fri 08-14, Sat 08-15, Sun 08-16, Mon 08-17, Tue 08-18, Wed 08-19, Thu 08-20, Fri 08-21, Sat 08-22, Sun 08-23, Mon 08-24, Tue 08-25, Wed 08-26, Thu 08-27, Fri 08-28, Sat 08-29, Sun 08-30, Mon 08-31

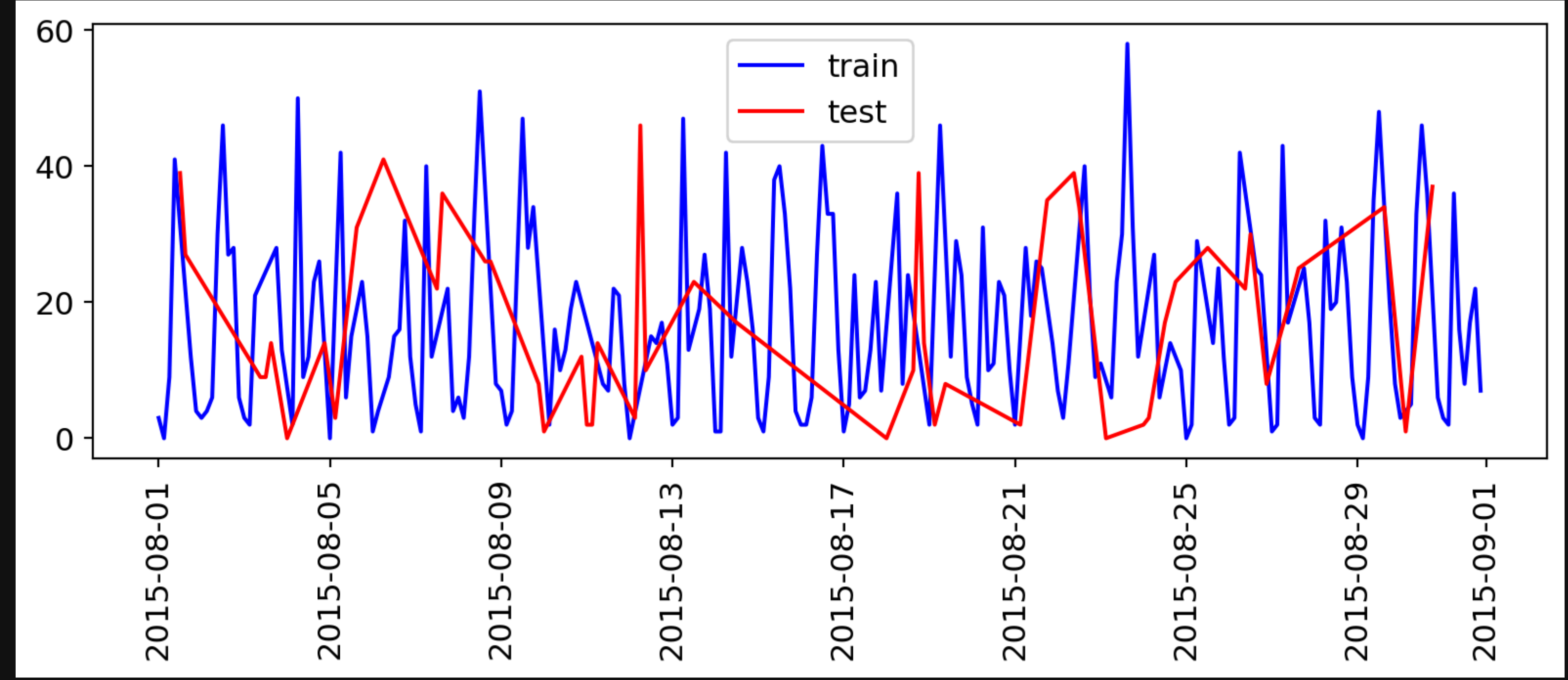
Do you see any daily patterns? Weekly patterns? Noise?



⛔ Incorrect data splitting

```
1 train_df, test_df = train_test_split(citibike, test_size=0.2, random_state=123)
2 print('Train largest date: ', train_df.index.max())
3 print('Test smallest date: ', test_df.index.min())
```

```
Train largest date:  2015-08-31 21:00:00
Test smallest date:  2015-08-01 12:00:00
```

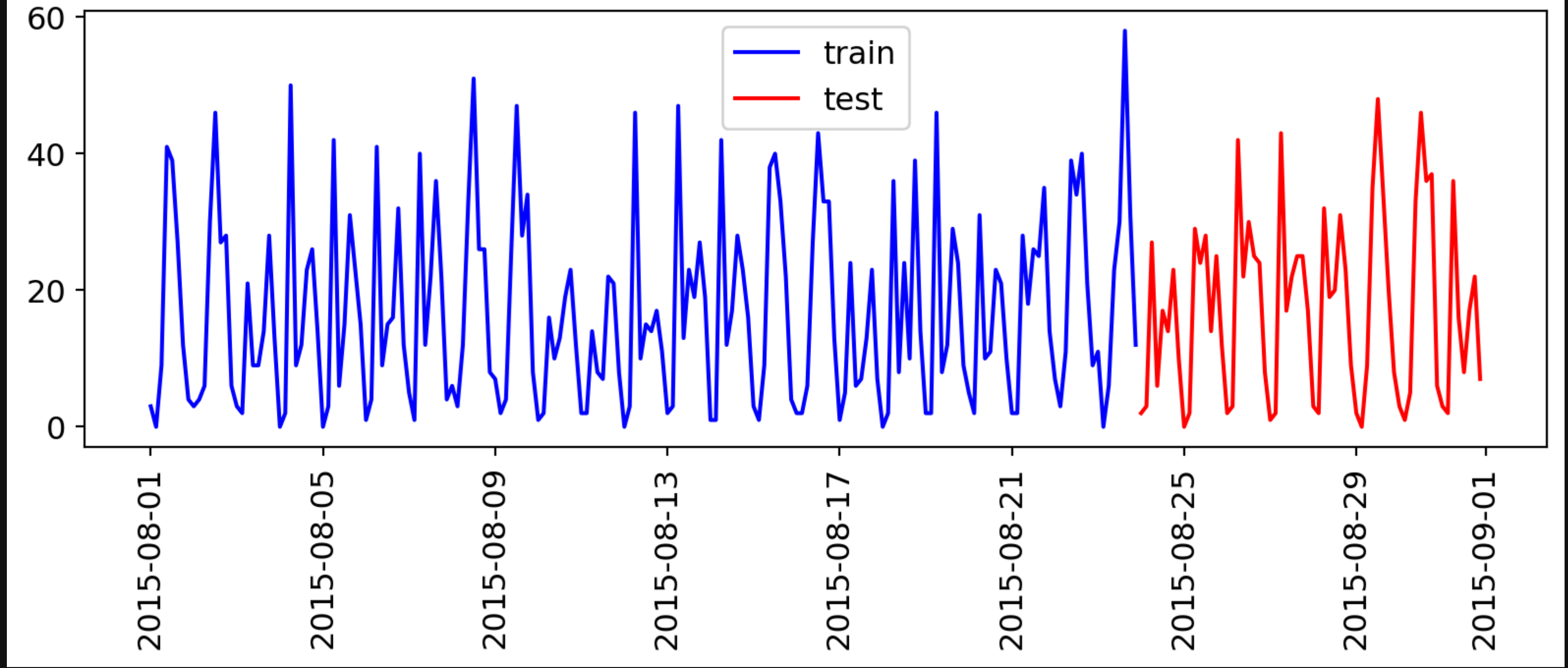


⛔ We should never train on the future to predict the past!

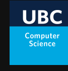
Correct data splitting

In time series, the simplest split is: earlier data ⟶ training and later data ⟶ testing. For example:

```
n_train = 184
train_df = citibike[:184]
test_df = citibike[184:]
```



Feature engineering for time series



Motivation

- In this toy data, we just have a single feature: the date time feature.
- Note that ML models do not have a built-in concept of time. We have to give it to them.
- We will explore different ways to extract informative features from time.

POSIX time feature

- Let’s start with a simplest encoding.
- A common way that dates are stored on computers is using POSIX time, which is the number of seconds since January 1970 00:00:00 (this is beginning of Unix time).
- Let’s start with encoding feature as a single integer representing this POSIX time.

```
1 # convert to POSIX time by dividing by 10**9
2 X = (
3     citibike.index.astype("int64").values.reshape(-1, 1) // 10**9
4 )  # convert to POSIX time by dividing by 10**9
5 y = citibike.values
6 X[:10]
```

```
array([[1438387],
       [1438398],
       [1438408],
       [1438419],
       [1438430],
       [1438441],
       [1438452],
       [1438462],
       [1438473],
       [1438484]])
```

RF on posix features

```
Train-set R^2: 0.85
Test-set R^2: -0.04
```

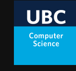
RandomForestRegressor
Features= POSIX time

Rentals
Date

Sat 08-01, Sun 08-02, Mon 08-03, Tue 08-04, Wed 08-05, Thu 08-06, Fri 08-07, Sat 08-08, Sun 08-09, Mon 08-10, Tue 08-11, Wed 08-12, Thu 08-13, Fri 08-14, Sat 08-15, Sun 08-16, Mon 08-17, Tue 08-18, Wed 08-19, Thu 08-20, Fri 08-21, Sat 08-22, Sun 08-23, Mon 08-24, Tue 08-25, Wed 08-26, Thu 08-27, Fri 08-28, Sat 08-29, Sun 08-30, Mon 08-31

train
test
prediction train
prediction test

- The predictions on the training data and training score are pretty good
- But for the test data, a constant line is predicted ...
- What’s going on?



Trees cannot extrapolate!

- Tree-based models only make predictions within the range of values they've seen during training.
- Trees partition the feature space into fixed regions and predictions inside each region are averages of training labels.
- If your future timestamps are larger than the ones in the training set trees cannot "see beyond" the training range and they will flatline or behave unpredictably.

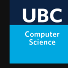
This is exactly what happens with POSIX time encoded as a single numeric feature!

Extracting date and time information

- Note that our index is of this special type: `DateTimeIndex`. We can extract all kinds of interesting information from it.

```
1 print(citibike.index[0])
2 print(citibike.index[0].month_name())
3 print(citibike.index[0].dayofweek)
4 print(citibike.index[0].hour)
```

```
2015-08-01 00:00:00
August
5
0
```



Hour of day

- We noted before that the hour of day and day of week seem quite important.
- Let’s start with hour of day.

```
1 X_hour = citibike.index.hour.values.reshape(-1, 1)
2 X_hour[:10]
```

```
array([[ 0],
       [ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18],
       [21],
       [ 0],
       [ 3]], dtype=int32)
```

RF with hour of day

```python
1  regressor = RandomForestRegressor(n_estimators=100, random_state=0)
2  eval_on_features(X_hour, y, regressor,  xtick_labels, feat_names="Hour of the day")
```

```
Train-set R^2: 0.50
Test-set R^2: 0.60
```

RandomForestRegressor
Features= Hour of the day

Rentals

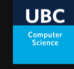
0 20 40 60

Sat 08-01 Sun 08-02 Mon 08-03 Tue 08-04 Wed 08-05 Thu 08-06 Fri 08-07 Sat 08-08 Sun 08-09 Mon 08-10 Tue 08-11 Wed 08-12 Thu 08-13 Fri 08-14 Sat 08-15 Sun 08-16 Mon 08-17 Tue 08-18 Wed 08-19 Thu 08-20 Fri 08-21 Sat 08-22 Sun 08-23 Mon 08-24 Tue 08-25 Wed 08-26 Thu 08-27 Fri 08-28 Sat 08-29 Sun 08-30 Mon 08-31

Date

train
test
prediction train
prediction test

The scores are better when we add hour of day feature!



hour of day + day of week

Now let's add day of week along with hour of day.

```
X_hour_week = np.hstack(
    [
        citibike.index.dayofweek.values.reshape(-1, 1),
        citibike.index.hour.values.reshape(-1, 1),
    ]
)
X_hour_week
```

```
array([[ 5,  0],
       [ 5,  3],
       [ 5,  6],
       [ 5,  9],
       [ 5, 12],
       [ 5, 15],
       [ 5, 18],
       [ 5, 21],
       [ 6,  0],
       [ 6,  3],
       [ 6,  6],
       [ 6,  9],
       [ 6, 12],
       [ 6, 15],
       [ 6, 18],
       [ 6, 21],
       [ 0,  0],
       [ 0,  3],
       [ 0,  6],
       [ 0,  9],
       [ 0, 12],
       [ 0, 15],
       [ 0, 18],
```

```
[ 0, 21],
[ 1,  0],
[ 1,  3],
[ 1,  6],
[ 1,  9],
[ 1, 12],
[ 1, 15],
[ 1, 18],
[ 1, 21],
[ 2,  0],
[ 2,  3],
[ 2,  6],
[ 2,  9],
[ 2, 12],
[ 2, 15],
[ 2, 18],
[ 2, 21],
[ 3,  0],
[ 3,  3],
[ 3,  6],
[ 3,  9],
[ 3, 12],
[ 3, 15],
[ 3, 18],
[ 3, 21],
[ 4,  0],
[ 4,  3],
[ 4,  6],
[ 4,  9],
[ 4, 12],
[ 4, 15],
[ 4, 18],
[ 4, 21],
[ 5,  0],
[ 5,  3],
[ 5,  6],
[ 5,  9],
[ 5, 12],
```

UBC Computer Science

```
[ 5, 15],
[ 5, 18],
[ 5, 21],
[ 6,  0],
[ 6,  3],
[ 6,  6],
[ 6,  9],
[ 6, 12],
[ 6, 15],
[ 6, 18],
[ 6, 21],
[ 0,  0],
[ 0,  3],
[ 0,  6],
[ 0,  9],
[ 0, 12],
[ 0, 15],
[ 0, 18],
[ 0, 21],
[ 1,  0],
[ 1,  3],
[ 1,  6],
[ 1,  9],
[ 1, 12],
[ 1, 15],
[ 1, 18],
[ 1, 21],
[ 2,  0],
[ 2,  3],
[ 2,  6],
[ 2,  9],
[ 2, 12],
[ 2, 15],
[ 2, 18],
[ 2, 21],
[ 3,  0],
[ 3,  3],
[ 3,  6],
```

UBC Computer Science

```
[ 3,  9],
[ 3, 12],
[ 3, 15],
[ 3, 18],
[ 3, 21],
[ 4,  0],
[ 4,  3],
[ 4,  6],
[ 4,  9],
[ 4, 12],
[ 4, 15],
[ 4, 18],
[ 4, 21],
[ 5,  0],
[ 5,  3],
[ 5,  6],
[ 5,  9],
[ 5, 12],
[ 5, 15],
[ 5, 18],
[ 5, 21],
[ 6,  0],
[ 6,  3],
[ 6,  6],
[ 6,  9],
[ 6, 12],
[ 6, 15],
[ 6, 18],
[ 6, 21],
[ 0,  0],
[ 0,  3],
[ 0,  6],
[ 0,  9],
[ 0, 12],
[ 0, 15],
[ 0, 18],
[ 0, 21],
[ 1,  0],
```

UBC Computer Science

```
[ 1,  3],
[ 1,  6],
[ 1,  9],
[ 1, 12],
[ 1, 15],
[ 1, 18],
[ 1, 21],
[ 2,  0],
[ 2,  3],
[ 2,  6],
[ 2,  9],
[ 2, 12],
[ 2, 15],
[ 2, 18],
[ 2, 21],
[ 3,  0],
[ 3,  3],
[ 3,  6],
[ 3,  9],
[ 3, 12],
[ 3, 15],
[ 3, 18],
[ 3, 21],
[ 4,  0],
[ 4,  3],
[ 4,  6],
[ 4,  9],
[ 4, 12],
[ 4, 15],
[ 4, 18],
[ 4, 21],
[ 5,  0],
[ 5,  3],
[ 5,  6],
[ 5,  9],
[ 5, 12],
[ 5, 15],
[ 5, 18],
```

UBC Computer Science

```
[ 5, 21],
[ 6,  0],
[ 6,  3],
[ 6,  6],
[ 6,  9],
[ 6, 12],
[ 6, 15],
[ 6, 18],
[ 6, 21],
[ 0,  0],
[ 0,  3],
[ 0,  6],
[ 0,  9],
[ 0, 12],
[ 0, 15],
[ 0, 18],
[ 0, 21],
[ 1,  0],
[ 1,  3],
[ 1,  6],
[ 1,  9],
[ 1, 12],
[ 1, 15],
[ 1, 18],
[ 1, 21],
[ 2,  0],
[ 2,  3],
[ 2,  6],
[ 2,  9],
[ 2, 12],
[ 2, 15],
[ 2, 18],
[ 2, 21],
[ 3,  0],
[ 3,  3],
[ 3,  6],
[ 3,  9],
[ 3, 12],
```

UBC Computer Science

```
       [ 3, 15],
       [ 3, 18],
       [ 3, 21],
       [ 4,  0],
       [ 4,  3],
       [ 4,  6],
       [ 4,  9],
       [ 4, 12],
       [ 4, 15],
       [ 4, 18],
       [ 4, 21],
       [ 5,  0],
       [ 5,  3],
       [ 5,  6],
       [ 5,  9],
       [ 5, 12],
       [ 5, 15],
       [ 5, 18],
       [ 5, 21],
       [ 6,  0],
       [ 6,  3],
       [ 6,  6],
       [ 6,  9],
       [ 6, 12],
       [ 6, 15],
       [ 6, 18],
       [ 6, 21],
       [ 0,  0],
       [ 0,  3],
       [ 0,  6],
       [ 0,  9],
       [ 0, 12],
       [ 0, 15],
       [ 0, 18],
       [ 0, 21]], dtype=int32)
```

RF with hour of day + day of week

```python
eval_on_features(X_hour_week, y, regressor,  xtick_labels, feat_names = "hour of day + day of week")
```

```
Train-set R^2: 0.89
Test-set R^2: 0.84
```

RandomForestRegressor
Features= hour of day + day of week

Rentals

Date

train
test
prediction train
prediction test

Sat 08-01, Sun 08-02, Mon 08-03, Tue 08-04, Wed 08-05, Thu 08-06, Fri 08-07, Sat 08-08, Sun 08-09, Mon 08-10, Tue 08-11, Wed 08-12, Thu 08-13, Fri 08-14, Sat 08-15, Sun 08-16, Mon 08-17, Tue 08-18, Wed 08-19, Thu 08-20, Fri 08-21, Sat 08-22, Sun 08-23, Mon 08-24, Tue 08-25, Wed 08-26, Thu 08-27, Fri 08-28, Sat 08-29, Sun 08-30, Mon 08-31

Ridge with Hour of day + Day of week

```
Train-set R^2: 0.16
Test-set R^2: 0.13
```

Ridge
Features= hour of day + day of week

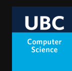
Rentals
0
20
40
60

Sat 08-01
Sun 08-02
Mon 08-03
Tue 08-04
Wed 08-05
Thu 08-06
Fri 08-07
Sat 08-08
Sun 08-09
Mon 08-10
Tue 08-11
Wed 08-12
Thu 08-13
Fri 08-14
Sat 08-15
Sun 08-16
Mon 08-17
Tue 08-18
Wed 08-19
Thu 08-20
Fri 08-21
Sat 08-22
Sun 08-23
Mon 08-24
Tue 08-25
Wed 08-26
Thu 08-27
Fri 08-28
Sat 08-29
Sun 08-30
Mon 08-31

Date

train
test
prediction train
prediction test

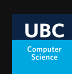
Why is `Ridge` performing poorly on the training data as well as test data?



Encoding hour and day with OHE

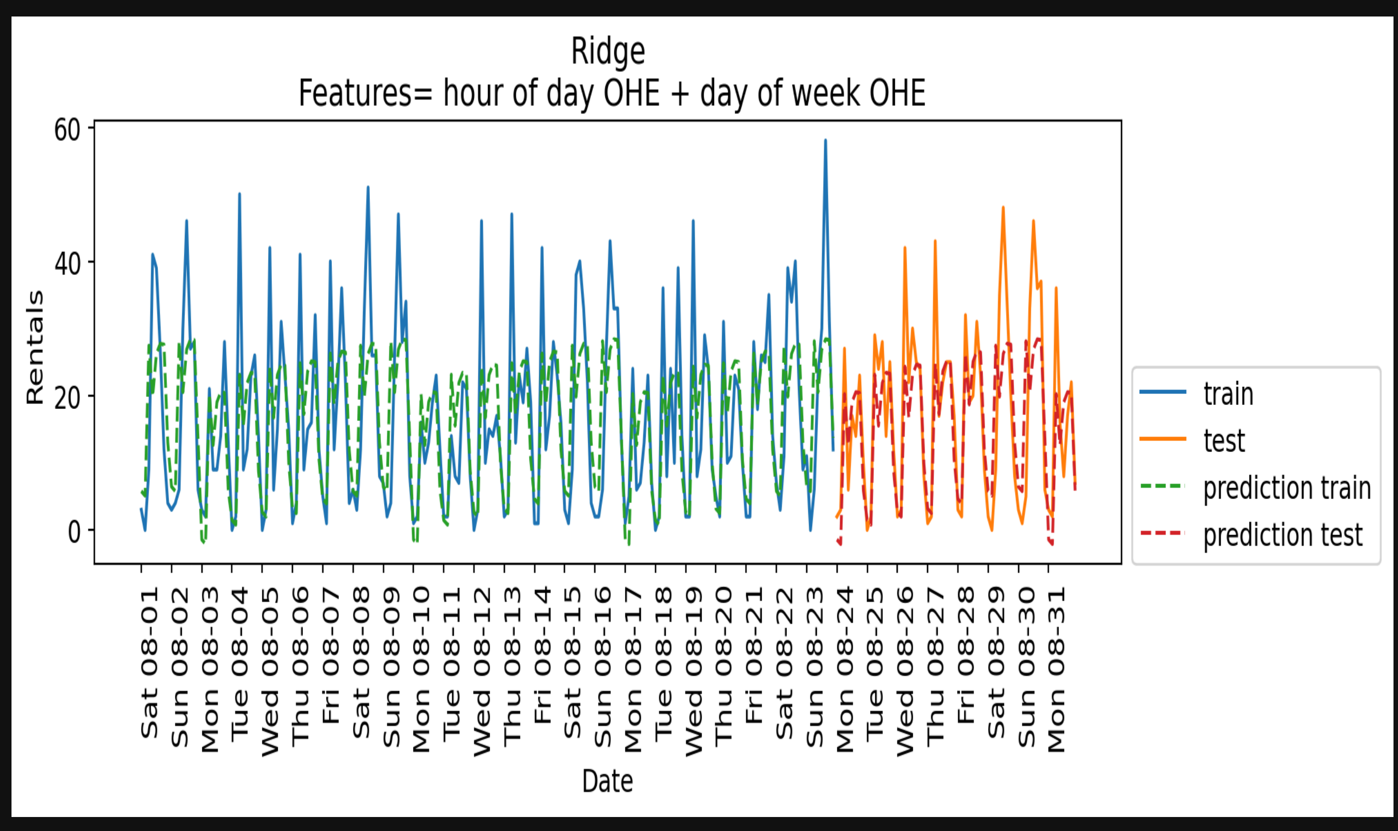
```
1 enc = OneHotEncoder()
2 X_hour_week_onehot = enc.fit_transform(X_hour_week).toarray()
3 hour = ["%02d:00" % i for i in range(0, 24, 3)]
4 day = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
5 features = day + hour
6 pd.DataFrame(X_hour_week_onehot, columns=features).head(6)
```

	Mon	Tue	Wed	Thu	Fri	Sat	Sun	00:00	03:00	06:00	09:00	12
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.(
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.(
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.(
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.(
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.(
5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.(



Ridge with hour of day OHE + day of week OHE

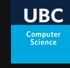
```
Train-set R^2: 0.53
Test-set R^2: 0.62
```



- The scores are a bit better!
- Can we improve them further?

Add interaction features

	Mon	Tue	Wed	Thu	Fri	Sat	Tue 00:00	Tue 03:00	Tue 06:00	Tue 09:00	Tue 18:00
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0



Ridge with hour of day OHE + day of week OHE + interaction features

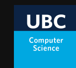
```
Train-set R^2: 0.87
Test-set R^2: 0.85
```

Ridge
Features= hour of day OHE + day of week OHE + interaction feats

Rentals

Date

train
test
prediction train
prediction test



Interpretation

Since we are using a linear model, we can examine the coefficients learned by `Ridge`.

	Coefficient
Sat 09:00	15.196739
Wed 06:00	15.005809
Sat 12:00	13.437684
Sun 12:00	13.362009
Thu 06:00	10.907595
...	...
Sat 21:00	-6.085150
00:00	-11.693898
03:00	-12.111220
Sat 06:00	-13.757591
Sun 06:00	-18.033267

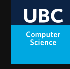
71 rows × 1 columns

Do these coefficients make sense?

iClicker 19.2

Select all of the following statements which are TRUE.

- a. One-hot encoding the hour of day allows linear models to treat each hour as a separate category and capture cyclic patterns effectively.
- b. Success in time-series modeling often depends more on choosing the right features and right encoding for the given model.
- c. Tree-based models are good at predicting values beyond the range of the training set because they can extrapolate trends.



Lag-based features

Creating lag features

```
1 def create_lag_df(df, lag, cols):
2     return df.assign(
3         **{f"{col}-{n}": df[col].shift(n) for n in range(1, lag + 1) for col in cols}
4     )
5 rentals_lag5 = create_lag_df(rentals_df, 5, ['n_rentals'] )
6 rentals_lag5.head(8)
```

	n_rentals	n_rentals-1	n_rentals-2	n_rentals-3	n_rentals-4	n_re
starttime						
2015-08-01 00:00:00	3	NaN	NaN	NaN	NaN	NaN
2015-08-01 03:00:00	0	3.0	NaN	NaN	NaN	NaN
2015-08-01 06:00:00	9	0.0	3.0	NaN	NaN	NaN

	n_rentals	n_rentals-1	n_rentals-2	n_rentals-3	n_rentals-4	n_re
starttime						
2015-08-01 09:00:00	41	9.0	0.0	3.0	NaN	NaN
2015-08-01 12:00:00	39	41.0	9.0	0.0	3.0	NaN
2015-08-01 15:00:00	27	39.0	41.0	9.0	0.0	3.0
2015-08-01 18:00:00	12	27.0	39.0	41.0	9.0	0.0
2015-08-01 21:00:00	4	12.0	27.0	39.0	41.0	9.0

UBC Computer Science

Ridge with lag features

```
Train-set R^2: 0.25
Test-set R^2: 0.37
```

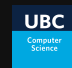
Ridge
Features= Lag features

Rentals

Date

train
test
prediction train
prediction test

The model is performing poorly.



Random Forest with lag features

```
Train-set R^2: 0.94
Test-set R^2: 0.69
```

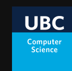
RandomForestRegressor
Features= Lag features

Rentals

Date

train
test
prediction train
prediction test

Better than the linear model but worse than our best model.



Ridge with all features

```
Train-set R^2: 0.87
Test-set R^2: 0.84
```

Ridge
Features= hour of day OHE + day of week OHE + interaction feats + Lag feats

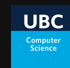
Rentals
60
40
20
0

Sat 08-01
Sun 08-02
Mon 08-03
Tue 08-04
Wed 08-05
Thu 08-06
Fri 08-07
Sat 08-08
Sun 08-09
Mon 08-10
Tue 08-11
Wed 08-12
Thu 08-13
Fri 08-14
Sat 08-15
Sun 08-16
Mon 08-17
Tue 08-18
Wed 08-19
Thu 08-20
Fri 08-21
Sat 08-22
Sun 08-23
Mon 08-24
Tue 08-25
Wed 08-26
Thu 08-27
Fri 08-28
Sat 08-29
Sun 08-30
Mon 08-31

Date

train
test
prediction train
prediction test

Good scores. But lag features don't seem to help much in this case.



RF with all features

```
Train-set R^2: 0.96
Test-set R^2: 0.78
```

RandomForestRegressor
Features= hour of day + day of week + Lag feats

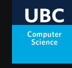
Rentals
0
20
40
60

Sat 08-01
Sun 08-02
Mon 08-03
Tue 08-04
Wed 08-05
Thu 08-06
Fri 08-07
Sat 08-08
Sun 08-09
Mon 08-10
Tue 08-11
Wed 08-12
Thu 08-13
Fri 08-14
Sat 08-15
Sun 08-16
Mon 08-17
Tue 08-18
Wed 08-19
Thu 08-20
Fri 08-21
Sat 08-22
Sun 08-23
Mon 08-24
Tue 08-25
Wed 08-26
Thu 08-27
Fri 08-28
Sat 08-29
Sun 08-30
Mon 08-31

Date

train
test
prediction train
prediction test

Lag features seem to make things a bit worse in this case.



Cross-validation with time series

- We can’t do regular cross-validation if we don’t want to be predicting the past.
- There is `TimeSeriesSplit` for time series data.

```
[0 1 2] [3]
[0 1 2 3] [4]
[0 1 2 3 4] [5]
```

Forecasting further into the future

Problem

- So far, our lag features let us predict 3 hours ahead.
- What if we want to predict 15 hours in the future?
- Problem: We do not yet know the rental counts for the required lag timestamps:
  - 2015-09-01 00:00:00
  - 2015-09-01 03:00:00
  - 2015-09-01 06:00:00
  - 2015-09-01 09:00:00
- Without those values, our lag features break 😢

Approach 1: Iterative forecasting

Train one model that predicts 3 hours ahead. At prediction time, move forward step by step, using your own predictions as future lag inputs.

Example:

- Predict rentals at 00:00 on 2015-09-01.
- Use that prediction as the lag to predict rentals at 03:00.
- Use both predictions to predict rentals at 06:00.
- Continue until you reach 12:00.

This method works, but errors accumulate as we step forward. The longer the horizon, the more uncertainty grows.

Approach 2: Direct forecasting (multiple horizons)

- Train separate models for each horizon:
  - Model 1 ⟶ predict 3 hours ahead
  - Model 2 ⟶ predict 6 hours ahead
  - Model 3 ⟶ predict 9 hours ahead

Each model uses lag features that match the required horizon.

(Optional) Approach 3: Multi-output models (joint forecasting)

- Train one model that predicts several future steps at once, e.g.:

```
y = [rentals_in_3h, rentals_in_6h, rentals_in_9h, ...]
```

- These models learn relationships across future time steps.
- Note: Multi-output forecasting is powerful, but outside the scope of CPSC 330.

Seasonality and trends

Time series often show recognizable patterns over time, which we want our models to capture.

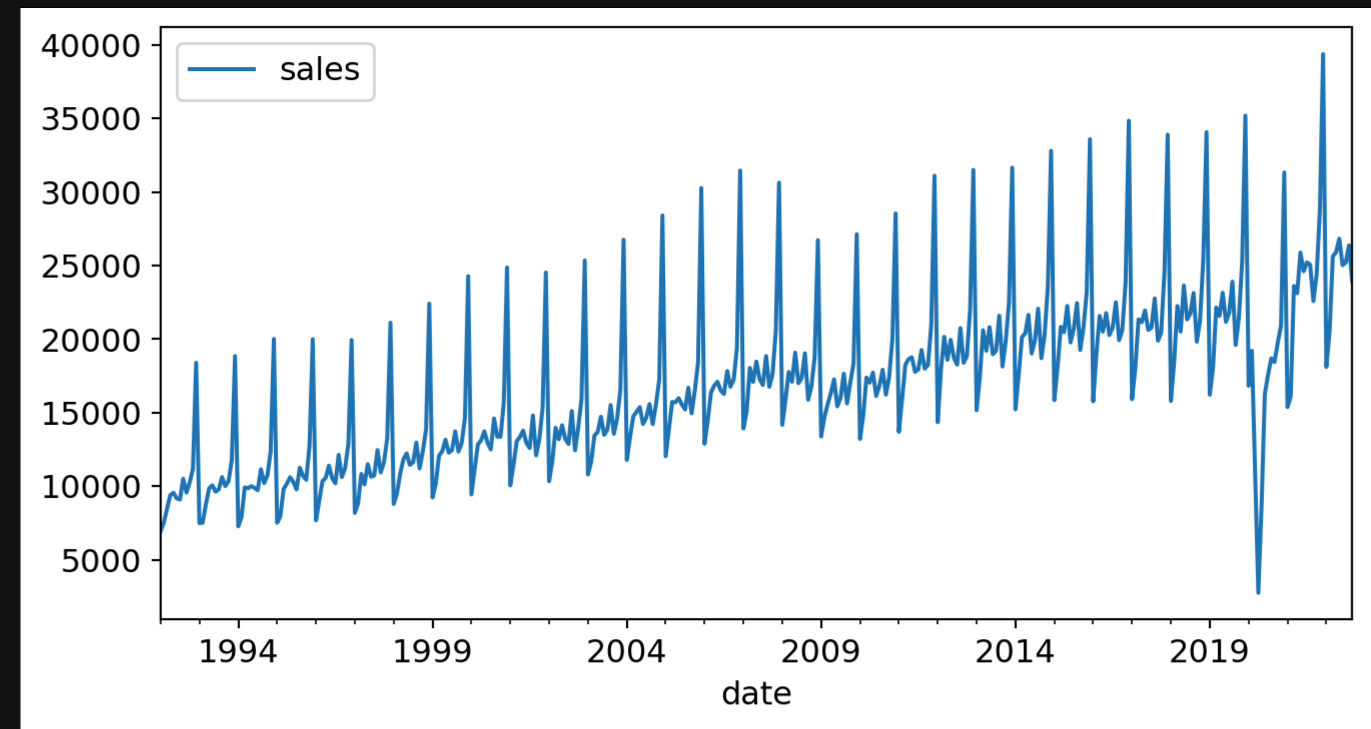
Seasonality

Seasonality refers to a **repeating pattern** that occurs at regular, predictable intervals.

- **Example 1:** Bike rentals tend to peak at the same hours each day (daily seasonality).
- **Example 2:** Retail sales consistently spike before the December holidays (annual seasonality).
- Recognizing seasonality helps a model **anticipate recurring patterns** rather than mistake them for noise.
- We can capture seasonality by adding features such as: `rush_hour`, `is_weekend`, and `is_holiday`

Trends

Let's consider another time series dataset, Retail Sales of Clothing and Clothing Accessory Stores dataset.



It looks like there's a clear trend; retail sales gradually increase over time. A **trend** is a long-term upward or downward movement in a time series.

Question: How might we encode this trend information so our models can use it?

Days since feature

One idea is to create a feature such as “Days_since”

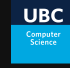
	date	sales	sales-1	sales-2	sales-3	sales-4	sales-5	Days_since
0	1992-01-01	6938	NaN	NaN	NaN	NaN	NaN	0
1	1992-02-01	7524	6938.0	NaN	NaN	NaN	NaN	31
2	1992-03-01	8475	7524.0	6938.0	NaN	NaN	NaN	60
3	1992-04-01	9401	8475.0	7524.0	6938.0	NaN	NaN	91
4	1992-05-01	9558	9401.0	8475.0	7524.0	6938.0	NaN	121

Types of time series

Univariate time series: Bike rentals

A **univariate time series** tracks *one* variable over time. Here, we only measure the number of bike rentals at each timestamp.

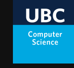
starttime	rentals
2015-08-26 12:00:00	30
2015-08-12 09:00:00	10
2015-08-19 03:00:00	2
2015-08-07 12:00:00	22
2015-08-03 09:00:00	9



Multivariate time series: Bike rentals with weather + traffic

A **multivariate time series** records several variables at each timestamp. Here, we track bike rentals along with weather and local traffic conditions.

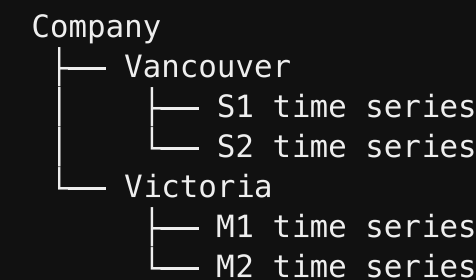
datetime	rentals	temp_C	humidity	wind_kmh	traffic_level
2015-08-01 09:00	12	18	65	10	2
2015-08-01 12:00	28	22	55	8	3
2015-08-01 15:00	35	24	50	12	4
2015-08-01 18:00	30	21	60	20	3
2015-08-02 09:00	10	17	70	15	1



Hierarchical time series: Bike sales across cities and stores

Consists of multiple variables from multiple observations, nested within the larger group categories, recorded sequentially over time (e.g., bike sales from multiple stores over time)

city	store_id	date	bike_sales
Vancouver	S1	2024-01-01	12
Vancouver	S1	2024-01-02	15
Vancouver	S2	2024-01-01	9
Vancouver	S2	2024-01-02	11
Victoria	M1	2024-01-01	7
Victoria	M1	2024-01-02	8
Victoria	M2	2024-01-01	14
Victoria	M2	2024-01-02	13



Group Work: Class Demo & Live Coding

For this demo, each student should click this link to create a new repo in their accounts, then clone that repo locally to follow along with the demo from today.

What did we learn today?

- **Time series** data has an inherent order.
- Correct **train/test splitting** must respect time (no shuffling!).
- **Modeling considerations:**
  - ML models can handle time series *if we engineer meaningful temporal features*.
  - **Linear models** struggle with cyclic numeric features unless encoded properly.
  - **Tree-based models** cannot extrapolate beyond the training range.
- Forecasting further ahead with iterative forecasting
- **Seasonality** = repeating patterns (daily, weekly, annual).
- **Trends** = long-term upward/downward movement

The key to successful time-series modeling: Choose the right features + choose the right model for the temporal structure.