Deep Learning

Housing Price Prediction

U Kang
Seoul National University

In This Lecture

- House pricing data
- Time series prediction with LSTM network

Outline

- **Problem Definition**
- Preprocessing Codes

Dataset (1)

- Zillow Economic Data
- Housing and economic data from a variety of public and proprietary sources by Zillow

Dataset (2)

- We will focus on these files
  - 'City_time_series.csv'
  - 'cities_crosswalk.csv'
- In EDA part, we will explore each file
- In prediction part, we will use 'ZHVIPerSqft_AllHomes' column in 'City_time_series.csv'
  - Mean value of the all homes per square feet

Problem Definition

- We will extract time series house pricing data and predict future price
- Input will be series of past house prices
- Output will be series of future house prices

Method

- Input data are time series
- We will use LSTM as main predictor
- We will apply techniques for treating time series data
  - Stationarity
  - Making data for supervised learning

Outline

- ☑ Problem Definition
- ☐ **Preprocessing Codes**

Import libraries

- We will use following libraries
  - Numpy
  - Pandas
  - Sklearn
  - Matplotlib
  - Seaborn # for prettifying graph
  - Plotly # for drawing graph

EDA (1)

- First file: ‘City_time_series.csv’
- In EDA part, we will focus on next columns
  - ‘ZHVIPerSqft_AllHomes’
  - ‘MedianListingPricePerSqft_AllHomes’
  - ‘MedianRentalPricePerSqft_AllHomes’
  - ‘ZHVI_2bedroom’
  - ‘ZHVI_3bedroom’
  - ‘ZHVI_4bedroom’
  - ‘RegionName’

EDA (2)

- ZHVI means 'Zillow Home Value Index'

```
df_city_time_seris = pd.read_csv('./data/City_time_series.csv')
df_city_time_seris.head()
```

	Date	RegionName	InventorySeasonallyAdjusted_AllHomes	InventoryRaw_AllHomes	MedianListingPricePerSqft_1Bedr
0	1996-04-30	abbottstownadamspa	NaN	NaN	I
1	1996-04-30	aberdeenbinghamid	NaN	NaN	I
2	1996-04-30	aberdeenharfordmd	NaN	NaN	I
3	1996-04-30	aberdeenmonroems	NaN	NaN	I
4	1996-04-30	aberdeenmoorenc	NaN	NaN	I

5 rows × 81 columns

EDA (3)

- Second file: ‘cities_crosswalk.csv’
- ‘Unique_City_ID’ column will be used later to draw nation wide map

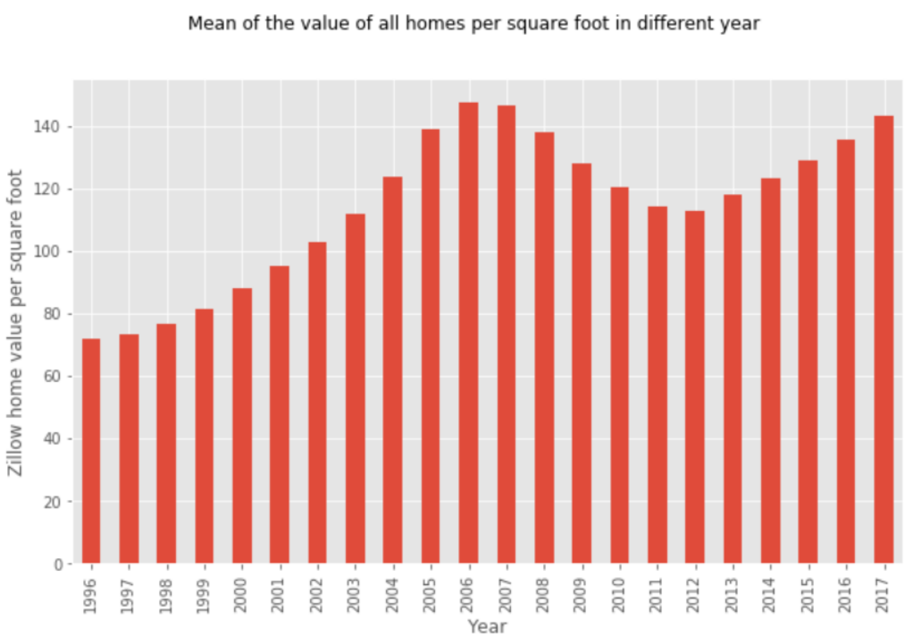
```
df_cities_crosswalk = pd.read_csv('./data/cities_crosswalk.csv')
df_cities_crosswalk.head()
```

	Unique_City_ID	City	County	State
0	oak_grovechristianky	Oak Grove	Christian	KY
1	jarvisburgcurritucknc	Jarvisburg	Currituck	NC
2	mcminnvilleyamhillor	McMinnville	Yamhill	OR
3	union_townshiperiepa	Union Township	Erie	PA
4	oshkoshwinnebagowi	Oshkosh	Winnebago	WI

EDA (4)

- Value of all homes per square in different years
  - Mean of the value of all homes per square foot

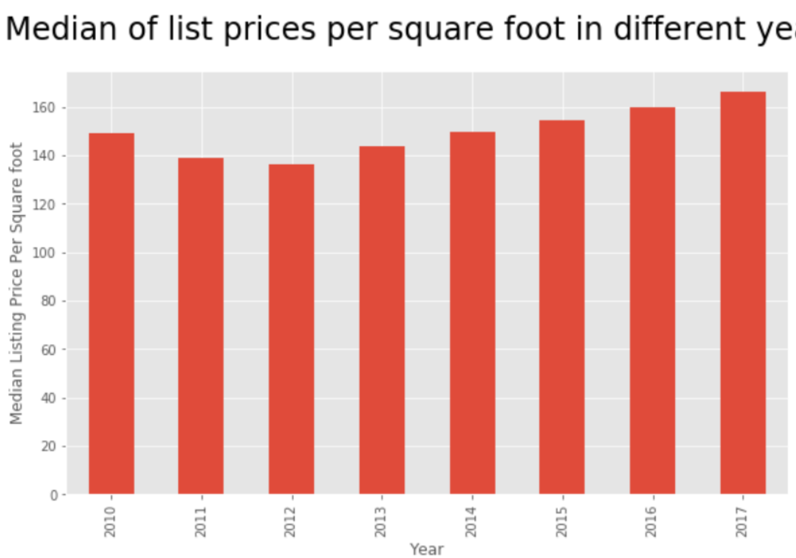
```python
df_city_time_seris.Date = pd.to_datetime(df_city_time_seris.Date)
df_city_time_seris.groupby(df_city_time_seris.Date.dt.year)['ZHVIPerSqft_AllHomes'].mean().plot(
    kind='bar', figsize=(10, 6))
plt.suptitle('Mean of the value of all homes per square foot in different year', fontsize=12)
plt.ylabel('Zillow home value per square foot')
plt.xlabel('Year')
plt.show()
```



EDA (5)

- Median of list prices per square foot in different years

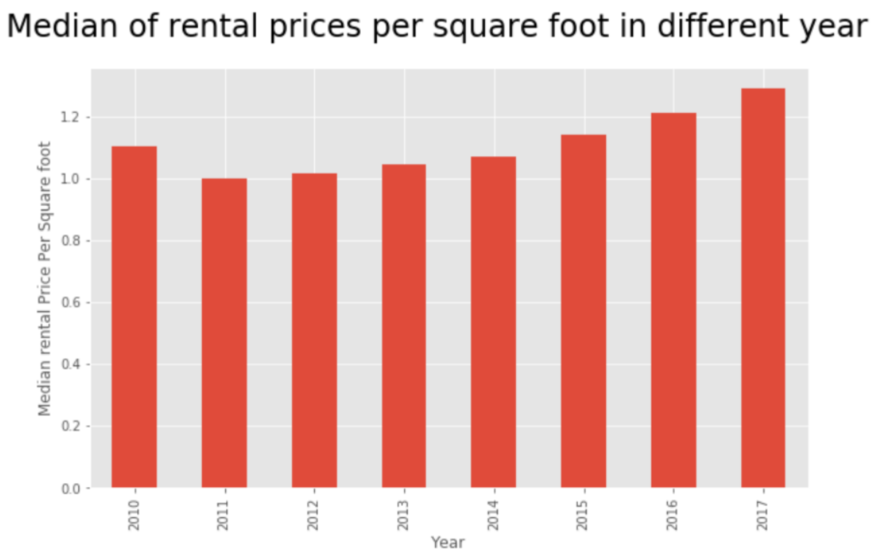
```
df_city_time_seris_without_null = df_city_time_seris.dropna(
    subset=['MedianListingPricePerSqft_AllHomes'], how='any')
df_city_time_seris_without_null \
    .groupby(df_city_time_seris_without_null.Date.dt.year)['MedianListingPricePerSqft_AllHomes'] \
    .mean().plot(kind='bar', figsize=(10, 6))
plt.suptitle('Median of list prices per square foot in different year', fontsize=24)
plt.ylabel('Median Listing Price Per Square foot')
plt.xlabel('Year')
plt.show()
```



EDA (6)

- Median of rental prices per square foot in different years

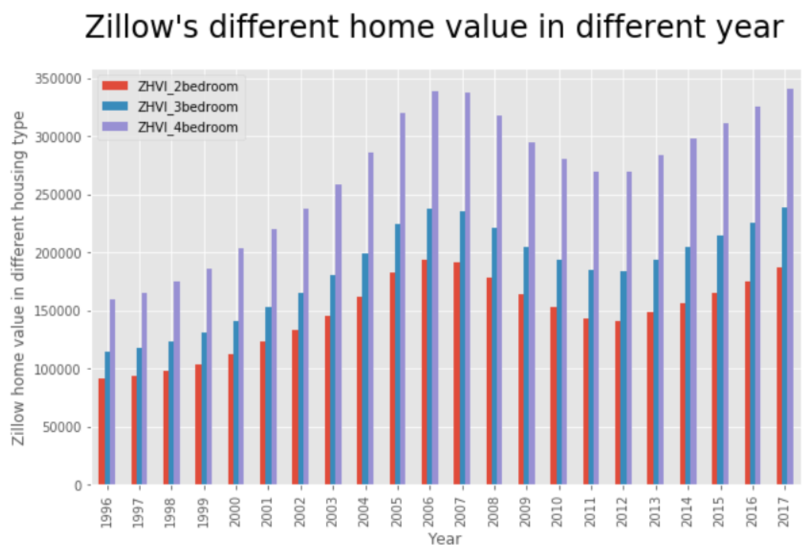
```
df_city_time_seris_without_null_rent = df_city_time_seris.dropna(
    subset=['MedianRentalPricePerSqft_AllHomes'], how='any')
df_city_time_seris_without_null_rent.groupby(df_city_time_seris_without_null_rent.Date.dt.year) \
    ['MedianRentalPricePerSqft_AllHomes'].mean().plot(kind='bar', figsize=(10, 6))
plt.suptitle('Median of rental prices per square foot in different year', fontsize=24)
plt.ylabel('Median rental Price Per Square foot')
plt.xlabel('Year')
plt.show()
```



EDA (7)

- Zillow's different home values in different years
  - Values vary based on housing types, e.g. 2,3,4 bed rooms

```python
df_city_time_seris.groupby(df_city_time_seris.Date.dt.year) \
    [['ZHVI_2bedroom','ZHVI_3bedroom','ZHVI_4bedroom']].mean().plot(kind='bar', figsize=(10, 6))
plt.suptitle("Zillow's different home value in different year", fontsize=24)
plt.ylabel('Zillow home value in different housing type')
plt.xlabel('Year')
plt.show()
```



EDA (8)

- Median of the value of all homes per square foot in different states

```python
# let's replace the regionName column value with State name from cities_crosswalk.csv
df_city_time_seris['RegionName'] = df_city_time_seris['RegionName'] \
    .map(df_cities_crosswalk.set_index('Unique_City_ID')['State'])
# group regionName with ZHVIpersqft mean value
df_regi_zhvi_sq_mean = df_city_time_seris.groupby(df_city_time_seris.RegionName)['ZHVIPerSqft_AllHomes'] \
    .mean().reset_index(name = "ZHVIpersqft_mean")
# drop null values
df_regi_zhvi_sq_mean = df_regi_zhvi_sq_mean.dropna(subset=['ZHVIpersqft_mean'], how='any')
```

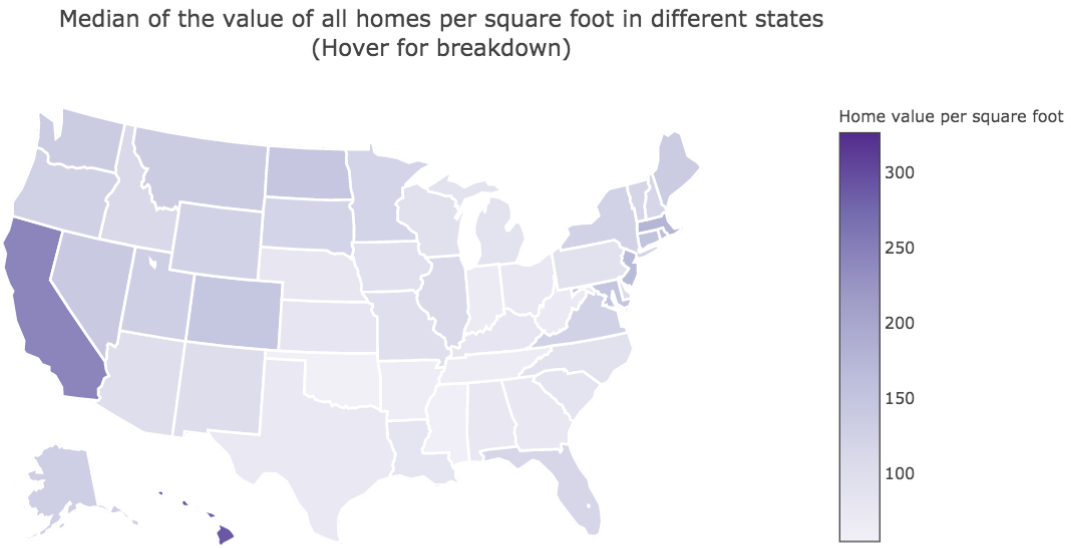
EDA(9)

- Draw graph across the states

```
scl = [
    [0.0, 'rgb(242,240,247)'],
    [0.2, 'rgb(218,218,235)'],
    [0.4, 'rgb(188,189,220)'],
    [0.6, 'rgb(158,154,200)'],
    [0.8, 'rgb(117,107,177)'],
    [1.0, 'rgb(84,39,143)']]


# difine our data for plotting
data = [ dict(
        type='choropleth',
        colorscale = scl,
        autocolorscale = False,
        locations = df_regi_zhvi_sq_mean['RegionName'], # location (states)
        z = df_regi_zhvi_sq_mean['ZHVIpersqft_mean'].astype(float), # Zillow Home value per square foot
        locationmode = 'USA-states', # let's define the location mode to USA_states
        text = 'Median home value per square foot',
        marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            ) ),
        colorbar = dict(
            title = "Home value per square foot")
        ) ]

layout = dict(
    title = 'Median of the value of all homes per square foot in different states<br>(Hover for breakdown)
    geo = dict(
        scope='usa',
        projection=dict( type='albers usa' ),
        showlakes = True,
        lakecolor = 'rgb(255, 255, 255)'),
         )


fig = dict( data=data, layout=layout )
# let's plot
py.iplot( fig, filename='d3-cloropleth-map' )
```

EDA (10)

- Result



Extract feature (1)

- Among many features in ‘City_time_series.csv’, we will use average of ‘ZHVIPerSqft_AllHomes’ per month as feature
- Then, our feature is series of floating point numbers containing average house prices over months

```
df_city_time_series = pd.read_csv('./data/City_time_series.csv',parse_dates=['Date'])
# drop null values in ZHVIPerSqft_AllHomes because we are interested in this column
df_city_time_series = df_city_time_series.dropna(subset=['ZHVIPerSqft_AllHomes'])
df_city_time_series.head()
```

Extract feature (2)

- Corresponding codes are below
- This will plot average price graph over years

```python
# the ZHVIPerSqft_AllHomes column has many value in same date but for different location.
# For this notebook we are not interested in location. We mean all the value in same date
df_zhvi_sqft_all = df_city_time_series.set_index('Date') \
                    .groupby(pd.Grouper(freq='d')).mean().dropna(how='all') \
                    .ZHVIPerSqft_AllHomes

fig, ax = plt.subplots(figsize=(15, 10))
ax.scatter(df_zhvi_sqft_all.index, df_zhvi_sqft_all)
# change x axis year location interval to 1 year. So that it displays data in interval of 1 year
ax.xaxis.set_major_locator(mdates.YearLocator(1))
# Add the title to the graph
plt.title('Zillow Home Value Index in Per Square foot in different year', fontsize=18)
# add xlabel
plt.xlabel('Year', fontsize=18)
# add ylabel
plt.ylabel('Zillow Home Value Index in Per Square foot', fontsize=18)
# beautify the x axis date presentation
fig.autofmt_xdate()
# And finally show the plot in a new window.
plt.show()

```

Stationarity (1)

- So far, we have a series of prices
- Before feeding in these numbers to LSTM, we need to make this time series stationary
- Stationarity in time series means statistical properties of a process generating a time series do not change over time
- We can give stationarity by using difference between current and next elements in series

Stationarity (2)

- The following functions can be helpful with respect to stationarity

```
# create a differenced series
# this is to make time series stationary
# why stationarity? => https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.Series(diff)
```

Make supervised data (1)

- RNN is a network for supervised dataset
- This means that we need both feature and ‘label’
- We only have feature (e.g. differenced series data) and not ‘label’
- We can convert this series to feature and ‘label’

Make supervised data (2)

- We can frame current value as feature and next value as label
- By doing this, we will predict the very next value from past values

```
# frame a sequence as a supervised learning problem
# this methods will create a column and column value will be 1 shift from the data.
# it will make our data to supervised so that we can feed into network
def timeseries_to_supervised(data, lag=1):
    df = pd.DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = pd.concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df
```

Make supervised data (3)

- After timeseries_to_supervised
  - 1^{st} column: shifted value
  - 2^{nd} column: orig. value

	0	0
0	0.000000	-0.034709
1	-0.034709	-0.041769
2	-0.041769	-0.003739
3	-0.003739	-0.011523
4	-0.011523	0.022334
5	0.022334	0.051720
6	0.051720	0.136330
7	0.136330	0.175334
8	0.175334	0.238790
9	0.238790	-0.037459
10	-0.037459	0.308243
11	0.308243	0.216467

Scaler (1)

- Scaler can convert arbitrary range of values to given range
- This is a kind of standardization method and also can improve training quality

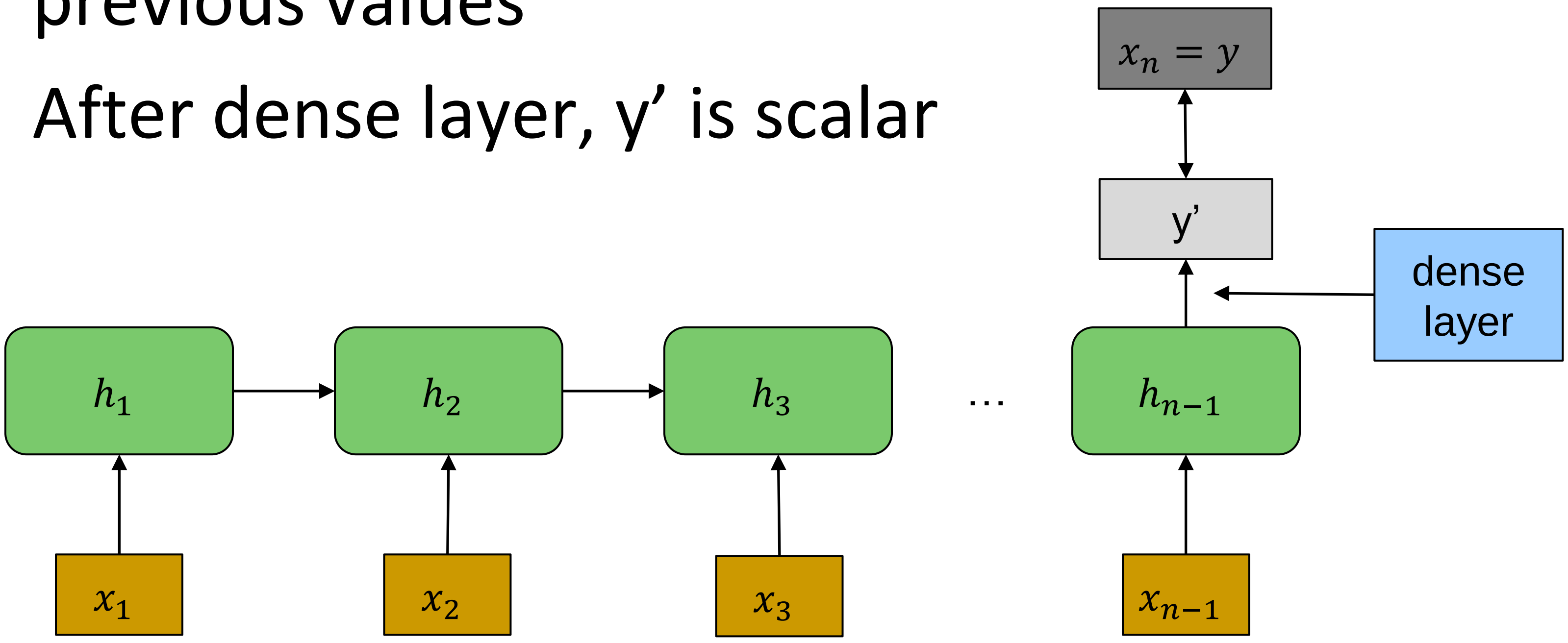
Scaler (2)

- We will use minmax scaler from sklearn

```
# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train_scaled = scaler.transform(train)
    # transform test
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled
```

Model Suggestion

- Our model gets monthly data as input
- As you can see, we predict last value given previous values
- After dense layer, y' is scalar



Questions?