Deep Learning：Kaggle 房價預測競賽

Group 8
楊怡芳 鄭建澤 陳譽升 胡心玫

TABLE OF CONTENTS

PROBLEM DEFINITION

今年巧遇肺炎疫情挑戰，各國央行積極地進行量化寬鬆政策，市場流入大量熱錢，導致房價狂漲，而我們需要一種方式針對特定房地產物件的價值進行判斷，以期能夠正確估算該物件的真實價值

DATASET INTRODUCTION by Kaggle

Train Set

1460 筆

79 項特徵變數

SalePrice 目標變數

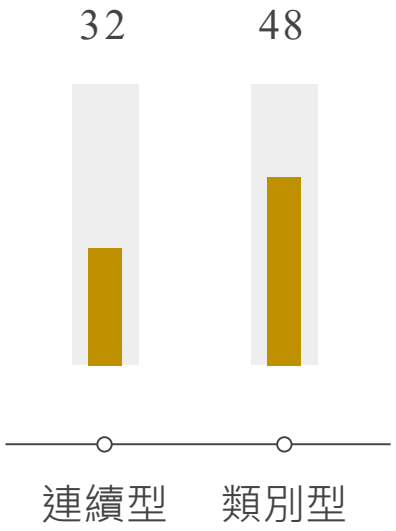
Test Set

1459 筆

79 項特徵變數

於 Kaggle 上傳驗證

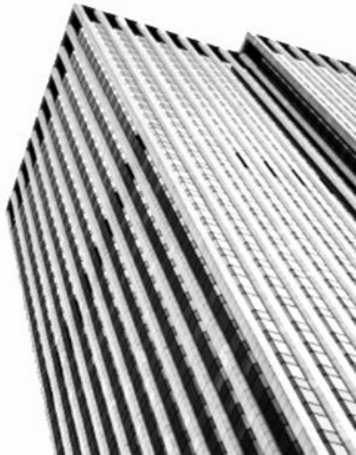
變數型態



美國愛荷華州住宅房價預測

VARIABLES SUMMARY at Train.csv

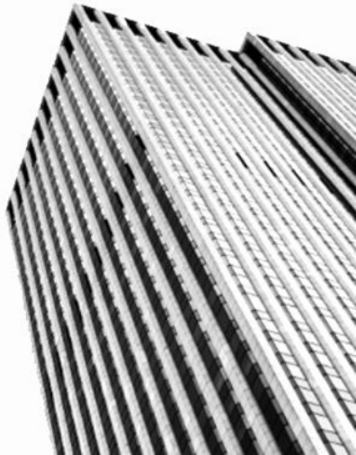
Feature	Description	Type
SalePrice	the property's sale price in dollars (target variable that you're trying to predict)	continuous
LotFrontage	Linear feet of street connected to property	
LotArea	Lot size in square feet	
…	…	
1stFlrSF	First Floor square feet	
2ndFlrSF	Second floor square feet	
LowQualFinSF	Low quality finished square feet (all floors)	



▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ 預測 SalePrice 房屋價格目標變數

VARIABLES SUMMARY at Train.csv

Feature	Description	Type
MSSubClass	The building class	category
MSZoning	The general zoning classification	
Street	Type of road access	
…	…	
YrSold	Year Sold	
SaleType	Type of sale	
SaleCondition	Condition of sale	



▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ 自變數共 79 項變數，但非每項皆與SalePrice有關

OUR SERVICES– 房價預測

- **What**

  美國愛荷華州
  住宅房價預測

- **Who**

  想擁有自家住宅或
  欲販售住宅之民眾

- **Why**

  市場泡沫，房地產價值狂漲，
  導致人民難以預測房價行情

- **When**

  欲購買、販賣住宅與
  了解房地產行情時

- **Where**

  美國愛荷華州

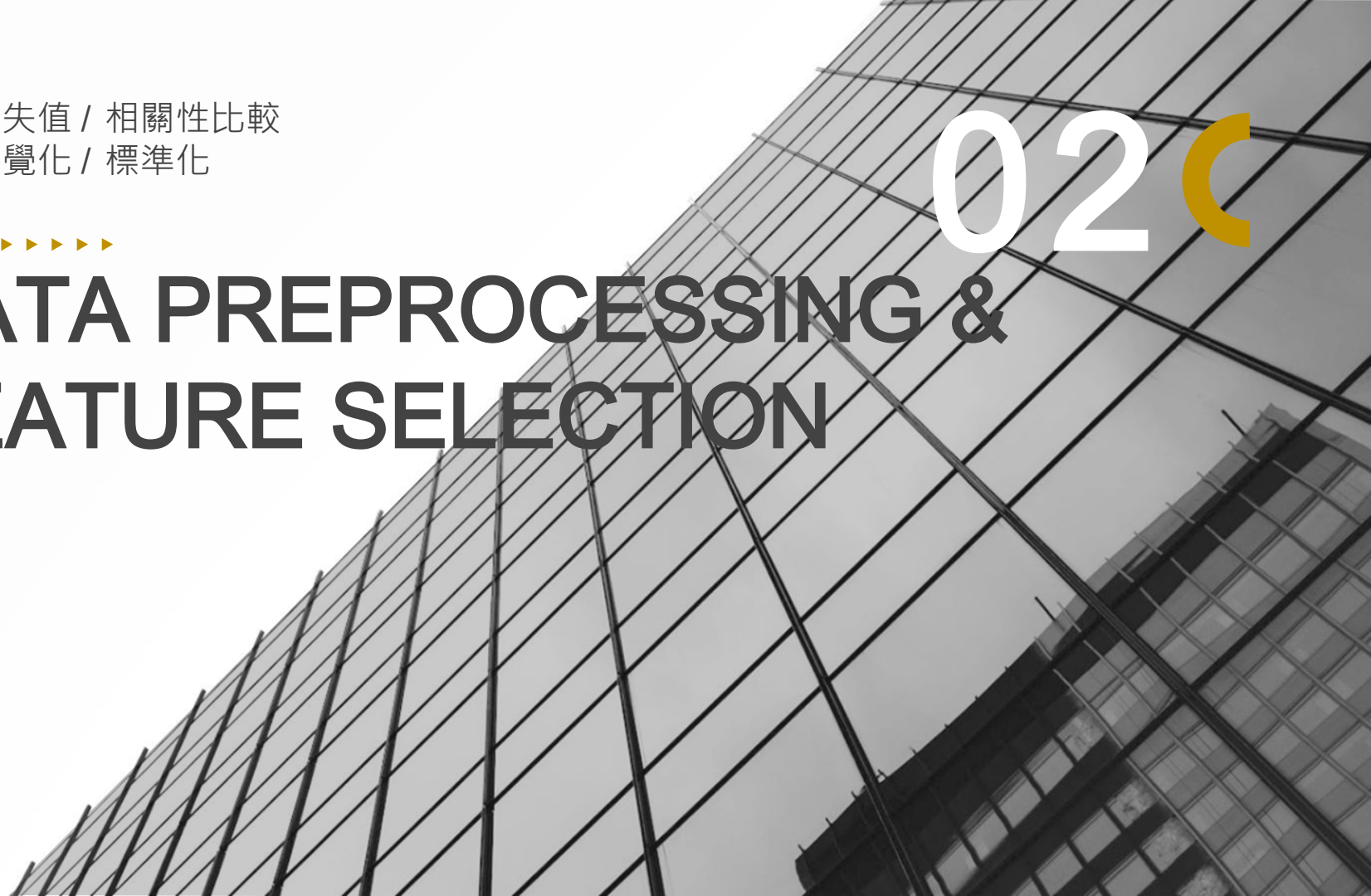
- **How**

  資料欲處理/ 視覺化、
  機器學習、深度學習

觀察缺失值 / 相關性比較
資料視覺化 / 標準化

02

DATA PREPROCESSING & FEATURE SELECTION



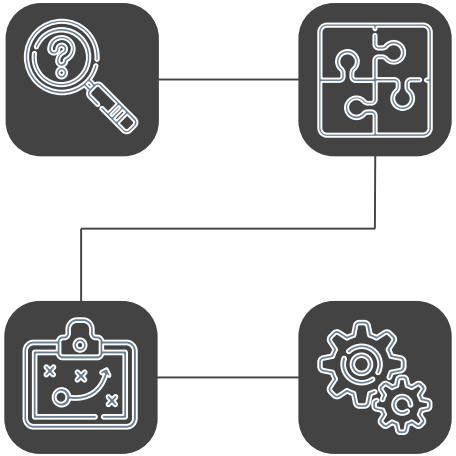
FLOW OF DATA PREPROCESSING

Step 1

觀察各變數
缺失值

Step 2

自變數與目標變數
相關性比較



Step 3

特徵變數
資料視覺化

Step 4

特徵變數
資料標準化

DATA PREPROCESSING step 1

- **Step 1**

  觀察各變數缺失值

存在過多缺失值之變數

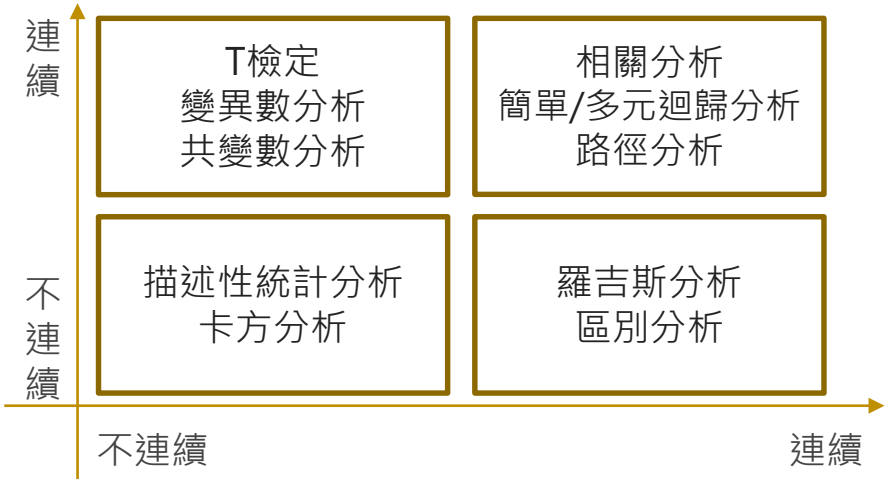
特徵變數	Train (共1460筆)	Test (共1459筆)
Alley	1369	1352
FireplaceQu	690	730
PoolQC	1453	1456
Fence	1179	1169
MiscFeature	1406	1408

▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ 將上述5個缺失值超過半數之變數不納入訓練模型

DATA PREPROCESSING Step 2

- **Step 2**

  自變數與目標變數
  相關性比較



▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ 分別利用「相關分析、T檢定」進行各特徵欄位與目標變數的相關性研究

Step 2 相關性比較

- **相關分析**

  自變數：連續型
  目標變數：連續型
  By python pandas

	SalePrice
GrLivArea	0.708624
GarageArea	0.623431
TotalBsmtSF	0.613581
1stFlrSF	0.605852
FullBath	0.560664
TotRmsAbvGrd	0.533723
YearBuilt	0.522897
YearRemodAdd	0.507101
GarageYrBlt	0.486362
MasVnrArea	0.477493
Fireplaces	0.466929
BsmtFinSF1	0.38642
LotFrontage	0.351799
WoodDeckSF	0.324413
2ndFlrSF	0.319334

OpenPorchSF	0.315856
HalfBath	0.284108
LotArea	0.263843
BsmtFullBath	0.227122
BsmtUnfSF	0.214479
BedroomAbvGr	0.168213
ScreenPorch	0.111447
PoolArea	0.092404
3SsnPorch	0.044584
BsmtFinSF2	-0.01138
BsmtHalfBath	-0.01684
MiscVal	-0.02119
LowQualFinSF	-0.02561
EnclosedPorch	-0.12858
KitchenAbvGr	-0.13591

▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ 自變數與目標變數之相關系數皆有0.6以上之表現，更深入進行視覺化分析

Step 2 相關性比較

- **T檢定分析**

  自變數：類別型
  目標變數：連續型
  By R regression

	t value	Siginificant
FunctionalMaj2	-2.237	*
LotConfigFR2	-3.067	**
NeighborhoodEdwards	-2.815	**
BsmtExposureNo	-2.676	**
…	…	…
GarageCondPo	2.729	**
GarageCondTA	2.736	**
NeighborhoodNoRidge	5.955	***

▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ 因kaggle 僅有1460筆，故我們將超過2顆*的變數皆納入我們的訓練因子

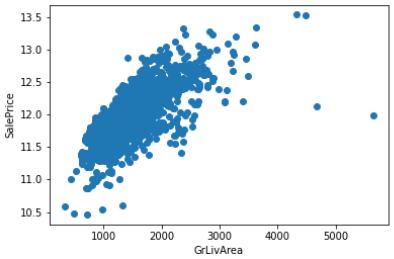
DATA PREPROCESSING step 3 特徵變數資料視覺化

連續型變數：GrLivArea、 GarageArea、TotalBsmtSF 、1stFlrSF與 SalePrice 之散佈圖

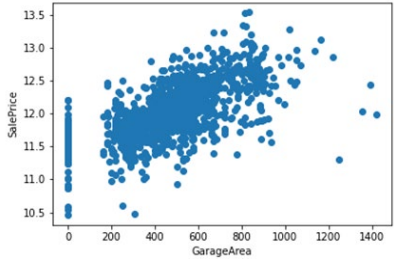
LotConfig 、GarageQual、OverallQual、BsmtQual 與 SalePrice 之長條圖與散佈圖：類別型變數

▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ ▸ 透過資料視覺化，確認各房屋特徵與目標變數之趨勢

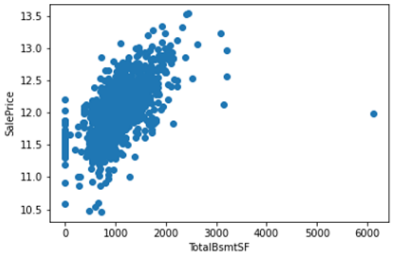
Step 3 連續型變數視覺化



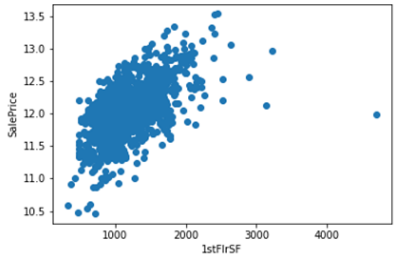
GrLivArea



GarageArea



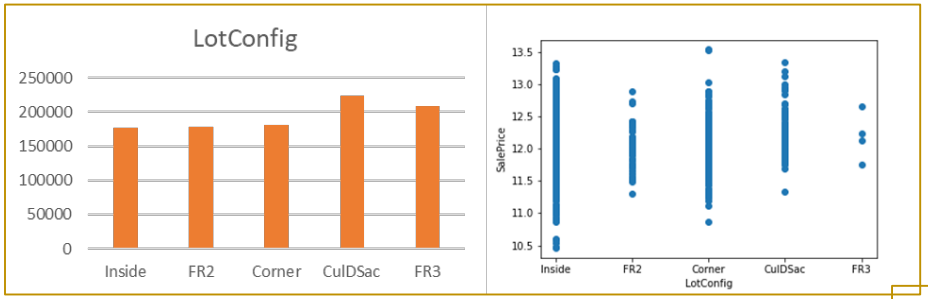
TotalBsmtSF



1stFlrSF

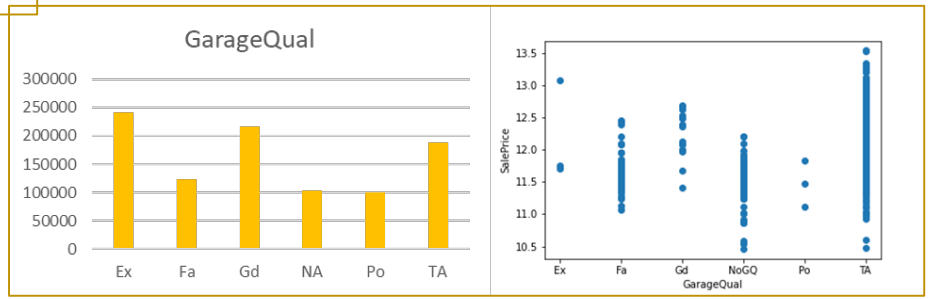
- 可以發現變數數值越大時，SalePrice 之價格趨勢向上，同時呼應前述相關分析之結果

Step 3 類別型變數視覺化

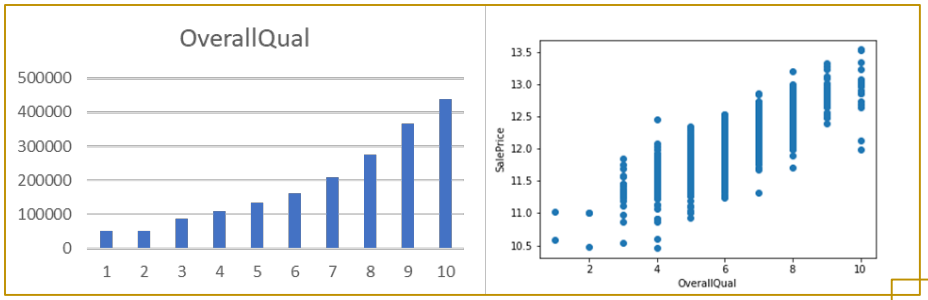


土地面積的分布樣式各類別與 SalePrice 關係

車庫品質各類別與 SalePrice 關係

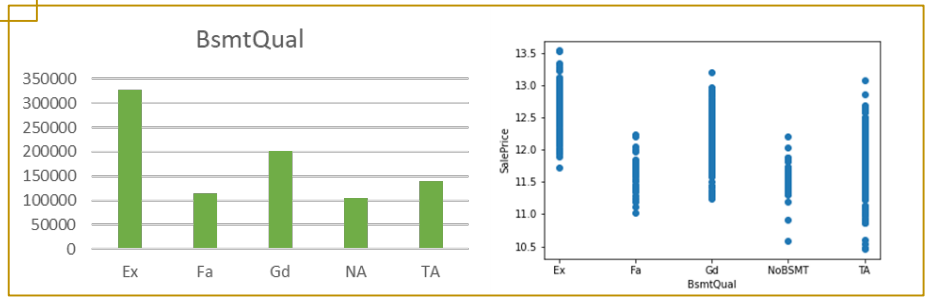


Step 3 類別型變數視覺化



建材與施工品質各類別與 SalePrice 關係

地下室高度各類別與 SalePrice 關係



DATA PREPROCESSING step 4 特徵變數資料標準化

```
## Standardizing numeric features
numeric_features = features.loc[:,['LotFrontage', 'LotArea', 'GrLivArea', 'TotalSF',
                                   'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GarageArea']]
numeric_features_standardized = (numeric_features - numeric_features.mean())/numeric_features.std()
```

```
# MasVnrType NA in all. filling with most popular values
features['MasVnrType'] = features['MasVnrType'].fillna(features['MasVnrType'].mode()[0])

# BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2
# NA in all. NA means No basement
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    features[col] = features[col].fillna('NoBSMT')

# TotalBsmtSF  NA in pred. I suppose NA means 0
features['TotalBsmtSF'] = features['TotalBsmtSF'].fillna(0)

# Electrical NA in pred. filling with most popular values
features['Electrical'] = features['Electrical'].fillna(features['Electrical'].mode()[0])

# KitchenAbvGr to categorical
features['KitchenAbvGr'] = features['KitchenAbvGr'].astype(str)

# KitchenQual NA in pred. filling with most popular values
features['KitchenQual'] = features['KitchenQual'].fillna(features['KitchenQual'].mode()[0])

# FireplaceQu  NA in all. NA means No Fireplace
features['FireplaceQu'] = features['FireplaceQu'].fillna('NoFP')

# GarageType, GarageFinish, GarageQual  NA in all. NA means No Garage
for col in ('GarageType', 'GarageFinish', 'GarageQual'):
    features[col] = features[col].fillna('NoGRG')

# SaleType NA in pred. filling with most popular values
features['SaleType'] = features['SaleType'].fillna(features['SaleType'].mode()[0])

# Year and Month to categorical
features['YrSold'] = features['YrSold'].astype(str)
features['MoSold'] = features['MoSold'].astype(str)
```

```
features['GarageArea'] = features['GarageArea'].astype(float)
features['GarageArea'] = features['GarageArea'].fillna(0.0)

# GarageCars  NA in pred. I suppose NA means 0
features['GarageCars'] = features['GarageCars'].fillna(0).astype(str)

# MSSubClass as str
features['MSSubClass'] = features['MSSubClass'].astype(str)

# MSZoning NA in pred. filling with most popular values
features['MSZoning'] = features['MSZoning'].fillna(features['MSZoning'].mode()[0])

# LotFrontage  NA in all. I suppose NA means 0
features['LotFrontage'] = features['LotFrontage'].fillna(features['LotFrontage'].mean())

# Alley  NA in all. NA means no access
features['Alley'] = features['Alley'].fillna('NOACCESS')

# Converting OverallCond to str
features.OverallCond = features.OverallCond.astype(str)

# MasVnrType NA in all. filling with most popular values
features['MasVnrType'] = features['MasVnrType'].fillna(features['MasVnrType'].mode()[0])
```

```
# Adding total sqfootage feature and removing Basement, 1st and 2nd floor features
features['TotalSF'] = features['TotalBsmtSF'] + features['1stFlrSF'] + features['2ndFlrSF']
```

共挑選出58個變數作為我們的訓練特徵，
並針對特徵變數進行補值與標準化之處理

03

CNN模型建立
超參數優化

THE ILLUSTRATION OF DEEP LEARNING MODEL

MODEL SUMMARY

By python Keras，共架設5個dense層

Layer (type)	Output Shape	Param #
dense_257 (Dense)	(None, 256)	69632
dense_258 (Dense)	(None, 256)	65792
dense_259 (Dense)	(None, 256)	65792
dense_260 (Dense)	(None, 128)	32896
dense_261 (Dense)	(None, 1)	129

Total params: 234,241

Trainable params: 234,241

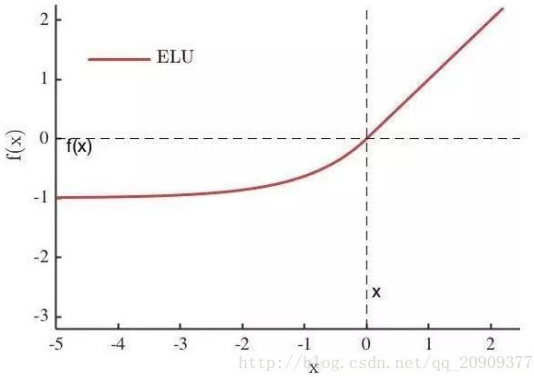
Non-trainable params: 0

```
model = Sequential()

num_features = len(train_features[0])

model.add(Dense(256, input_dim=num_features,
                activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(256, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(256, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(128, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(1, activation='elu',
                bias_regularizer=regularizers.l1(0.1),
                kernel_constraint=tf.keras.constraints.NonNeg()))
```

MODEL SUMMARY by elu activation function

By elu activation function

```
model = Sequential()

num_features = len(train_features[0])

model.add(Dense(256, input_dim=num_features,
                activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(256, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(256, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(128, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(1, activation='elu',
                bias_regularizer=regularizers.l1(0.1),
                kernel_constraint=tf.keras.constraints.NonNeg()))
```



The *exponential linear unit* (ELU) with $0 < \alpha$ is

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}. \qquad (15)$$

elu可以讓optimizer在參數在約等於零附近的區域能有更好的學習效率

MODEL SUMMARY by kernel regularizer

我們觀察到有overfitting 及最後一層bias過大的現象

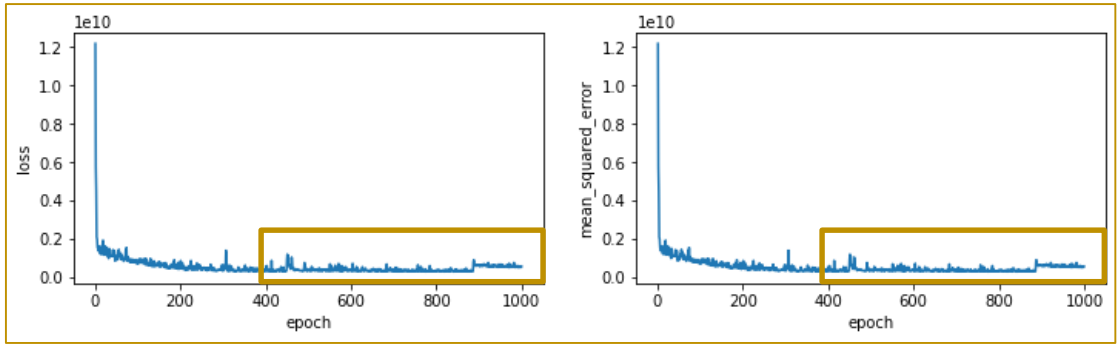
```
model = Sequential()

num_features = len(train_features[0])

model.add(Dense(256, input_dim=num_features,
                activation='elu',
                kernel_initializer='random_uniform'
                kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(256, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(256, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(128, activation='elu',
                kernel_initializer='random_uniform'
                ,kernel_regularizer=regularizers.l2(0.01)
               ))
model.add(Dense(1, activation='elu',
                bias_regularizer=regularizers.l1(0.1)
                kernel_constraint=tf.keras.constraints.NonNeg()))
```

於前後各加入了L2 kernel regularizer 防止模型過擬合

透過L1 biasregularizer 降低最後一層bias ，
讓神經元可正確學習weights

超參數調整 by epochs 迭代次數 (500 vs. 1000)

迭代次數等於1000的收斂過程



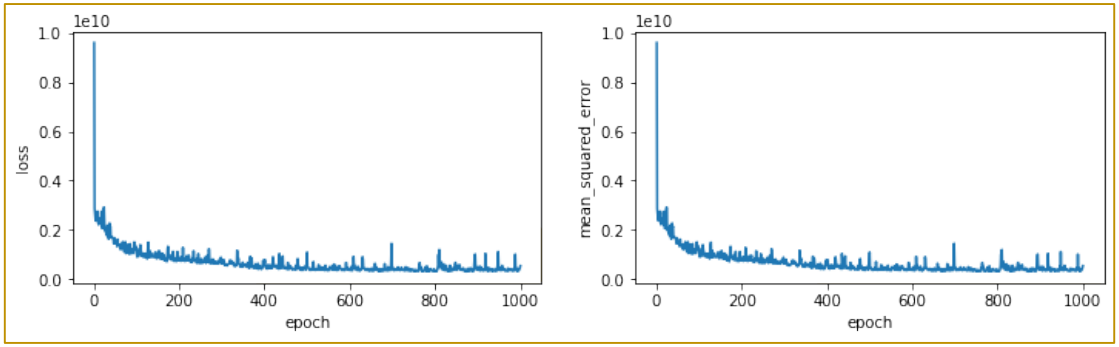
最終模型設定

Epochs = 500

發現該模型在迭代400次後，收斂已趨平緩，而超過900次後loss反倒增加

超參數調整 by hidden layer 神經元數量 (128 vs. 256) / 層數 (2 vs. 4)

兩層神經元數皆為128之training history



- 兩層128個節點的Dense層 Kaggle 表現已達到 = 0.14816

將節點數加寬至256後，發現Kaggle 表現已達到 = 0.14121

增加一層寬度為256的Dense層，另一層寬度為128的Dense層，發現 Kaggle 表現已達到 = 0.13656

超參數調整 by Adam learning rate & kernel regularizer

Adam learning rate

分別以 0.01及0.001 的學習率訓練出之模型，學習成效並沒有太大差異，可能是因為此model的參數眾多，使得local minimum較不可能出現，因此設定

0.01 learning rate

使最終模型可以更快速收斂

kernel regularizer

嘗試以0.01和0.001調整kernel regularizer參數，，發現兩者對於訓練沒有太大的差異。可能是因為的predict value本身偏大，而0.01和0.001的 L2 regulation懲罰對於Loss function不足為奇，因此設定

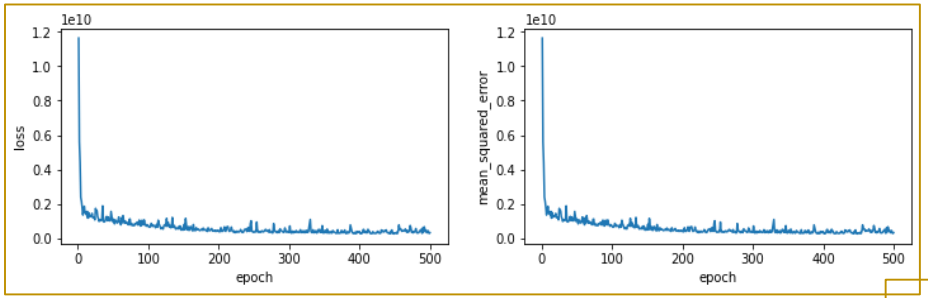
0.01 kernel regularizer

模型分析比較
結論與未來方向

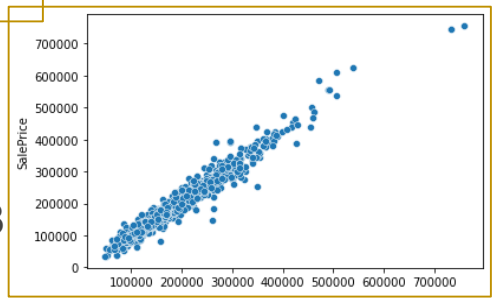
04

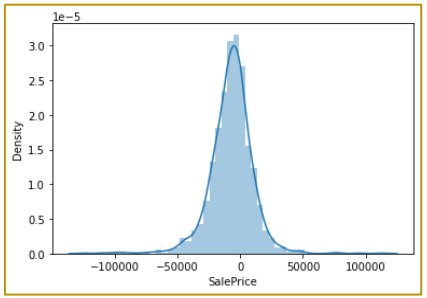
MODEL EVALUATION & CONCLUSION

最終模型表現 by 1387 筆訓練集 / 73 筆測試集



經過500次epoch訓練及超參數調整
最終模型之 Training History





預測值vs.實際值之視覺化結果
呈常態分配，標準差為 17501.68

- R-square達到0.9434，Training Set 之 RMSLE達到 0.10，
- Training Set 之 RMSLE達到 0.13656

最終模型表現 Compared with Linear Regression / SVR

```
from sklearn.linear_model import LinearRegression

lm_model = LinearRegression(fit_intercept=True)

lm_model.fit(train_features_st, train_labels)
```

Linear Regression 之 R-square = 0.897 / RMSLE = 0.189，擬合效果不佳

```
from sklearn.svm import SVR
clf = SVR(kernel='rbf', C=100000, gamma=0.01)
clf.fit(train_features_st, train_labels)
```

```
C=100000, cache_size=200, coef0=0.0, degree=3,
kernel='rbf', max_iter=-1, shrinking=True, tol=
```

```
y_test_raw = clf.predict(train_features_st)
y_test_raw
```

SVR之 R-square = 0.94 / RMSLE = 0.15，
顯見SVR有過擬合的現象產生

最終模型表現

各訓練模型評估比較

訓練模型	R-square (越接近1越好)	RMSLE (越接近0越好)
Linear Regression	0.897	0.189
SVR(支援向量回歸)	0.94	0.15
深度學習預測模型	0.9434	0.10(training) / 0.13656(testing)

- 發現我們設計深度學習模型於預測房價獲得最好之成效
- 未來將持續朝資料欲處理集超參數優化等方向延伸，並運用至其他房地產資料，以落實所學

THANKS