Mikołaj Bigaj, Krystian Pińczak

First we import all of the needed libraries, which includes the tensorflow that we will use in predicting the sale price.

Importing needed libraries

[1]:
```
import tensorflow as tf
import keras
from keras import layers
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pathlib import Path
```

Then we set up the project which includes displaying all of the columns.

Setting up the project

]:
```
TEST_FILE = '/kaggle/input/house-prices-advanced-regression-techniques/test.csv'
TRAIN_FILE = '/kaggle/input/house-prices-advanced-regression-techniques/train.csv'
pd.set_option('display.max_columns', None)
```

]:
```
train_data = pd.read_csv(TRAIN_FILE)
MAX_ROWS = len(train_data.index)
MAX_COLS = len(train_data.columns)
train_data.head(10)
```

]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	Lan
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl

Now we start cleaning up the data. First we remove the ID column as it is redundant.

Clearing training data

```
print('Deleting Id column')
train_data = train_data.drop('Id', axis=1)
```

```
Deleting Id column
```

Now we remove the columns that have more than 40 % of null / none data, then we replace the null data in rows with 0 or empty string.

```
def delete_columns(dataset):
    print('Deleting columns...\n')

    for column in dataset.columns:
        nan_count = [nan for nan in dataset[column] if pd.isna(nan)]
        if len(nan_count) > 0.4 * MAX_ROWS:
            print(f'Deleting {column} column')
            dataset = dataset.drop(columns=column)

    return dataset

    print(f'\nDeleted a total of {MAX_COLS - len(dataset.columns)} columns')
```

```
def replace_nulls(row, train_data=train_data):
    null_count = 0
    for col in row.index:
        if pd.isna(row[col]):
            if pd.api.types.is_numeric_dtype(train_data[col]):
                row[col] = 0
            else:
                row[col] = ''
            null_count += 1
    return row, null_count

train_data = delete_columns(train_data)

print(f'Replacing nulls in {len(train_data)} rows...\n')

total_nulls_replaced = 0
for i in range(len(train_data)):
    train_data.iloc[i], nulls_replaced = replace_nulls(train_data.iloc[i])
    total_nulls_replaced += nulls_replaced

print(f'Total nulls replaced: {total_nulls_replaced}')
```

Which leads to removing 6 columns and replacing a total of 860 nulls.

```
Deleting columns...

Deleting Alley column
Deleting MasVnrType column
Deleting FireplaceQu column
Deleting PoolQC column
Deleting Fence column
Deleting MiscFeature column

Total nulls replaced: 860
```

Now we visualize the cleared up data to see any irregularities.

Data Visualization

```
train_data.describe()
```

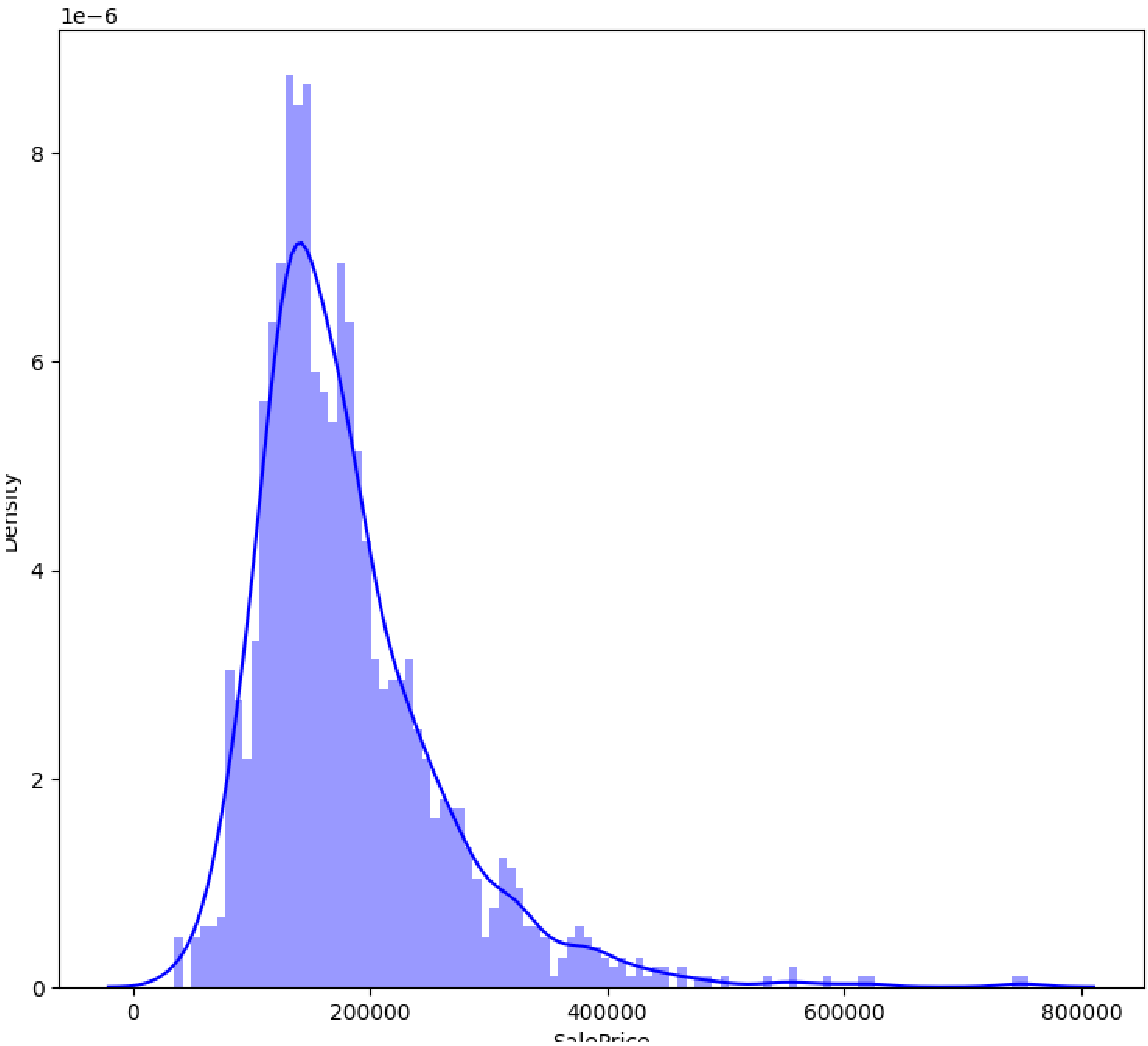
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	Mas
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	146
mean	56.897260	57.623288	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103
std	42.300571	34.664304	9981.264932	1.382997	1.112799	30.202904	20.645407	180
min	20.000000	0.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.0
25%	20.000000	42.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.0
50%	50.000000	63.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.0
75%	70.000000	79.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	164
max	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	160

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 74 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MSSubClass     1460 non-null   int64
 1   MSZoning       1460 non-null   object
 2   LotFrontage    1460 non-null   float64
 3   LotArea        1460 non-null   int64
 4   Street         1460 non-null   object
 5   LotShape       1460 non-null   object
 6   LandContour    1460 non-null   object
 7   Utilities      1460 non-null   object
 8   LotConfig      1460 non-null   object
 9   LandSlope      1460 non-null   object
 10  Neighborhood   1460 non-null   object
 11  Condition1     1460 non-null   object
 12  Condition2     1460 non-null   object
 13  BldgType       1460 non-null   object
 14  HouseStyle     1460 non-null   object
 15  OverallQual    1460 non-null   int64
 16  OverallCond    1460 non-null   int64
```

```
print(train_data['SalePrice'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(train_data['SalePrice'], color='b', bins=100, hist_kws={'alpha': 0.4})
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```



Now we prepare the cleared up data before giving it to train. First we transform it with one hot encoder.

```
def one_hot_encoder(dataset):
    columns = dataset.columns
    types = []
    for column in columns:
        row_list = dataset[column]
        if type(row_list[0]) != str:
            continue

        dict_map = dict()
        val = 0

        row_list = row_list.reset_index(drop=True)

        for value in row_list:
            if value in dict_map.keys():
                continue
            dict_map[value] = val
            val += 1

        for i in range(len(row_list)):
            row_list[i] = dict_map[row_list[i]]

        dataset[column] = row_list
    return dataset

train_data = one_hot_encoder(train_data)
train_data.head(10)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Nei
0	60	0	65.0	8450	0	0	0	0	0	0	0
1	20	0	80.0	9600	0	0	0	0	1	0	1
2	60	0	68.0	11250	0	1	0	0	0	0	0
3	70	0	60.0	9550	0	1	0	0	2	0	2
4	60	0	84.0	14260	0	1	0	0	1	0	3
5	50	0	85.0	14115	0	1	0	0	0	0	4
6	20	0	75.0	10084	0	0	0	0	0	0	5
7	60	0	0.0	10382	0	1	0	0	2	0	6
8	50	1	51.0	6120	0	0	0	0	0	0	7
9	190	0	50.0	7420	0	0	0	0	2	0	8

Then we split the data to training and testing based on given ratio between them (test_ratio).

```
def split_dataset(dataset, test_ratio=0.25):
  test_indices = np.random.rand(len(dataset)) < test_ratio
  return dataset[~test_indices], dataset[test_indices]

train_ds_pd, valid_ds_pd = split_dataset(train_data)
print(f"{len(train_ds_pd)} for training, {len(valid_ds_pd)} for testing")
```

```
1073 for training, 387 for testing
```

We also have this helper function to convert data into Tensorflow tensor applicable

```
def return_tensor(data):
    return np.asarray(data).astype(np.float32)
```

For training we use the Sequential model with Dense layers and reLu activation function for flattening unwanted negative results

Training model

```
model = keras.Sequential()
model.add(layers.Dense(768, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))

label = 'SalePrice'
```

```
print(train_ds_pd.shape, train_ds_pd[label].shape)
```

```
(1086, 74) (1086,)
```

```
x_label = train_ds_pd[label]
y_true = valid_ds_pd[label]

train_ds_pd = train_ds_pd.drop(label, axis=1)
valid_ds_pd = valid_ds_pd.drop(label, axis=1)
```

```
train_ds_pd = return_tensor(train_ds_pd)
valid_ds_pd = return_tensor(valid_ds_pd)
x_label = return_tensor(x_label)
y_true = return_tensor(y_true)
```

Here we make sure to compile the model with Adam optimizer and give it a learning rate that best suits the use case. We also want to extract the mean absolute error to see how far off our predictions are (in training)

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.008), loss='mean_absolute_error')
model.fit(train_ds_pd, x_label, epochs=200, steps_per_epoch=300, validation_split=0.1, shuffl
e=True)
```

```
Epoch 1/200
300/300 [==============================] - 2s 4ms/step - loss: 43909.4336 - val_loss: 4324
9.9453
Epoch 2/200
300/300 [==============================] - 1s 3ms/step - loss: 35542.8281 - val_loss: 2858
7.6309
Epoch 3/200
300/300 [==============================] - 1s 3ms/step - loss: 35171.3359 - val_loss: 3915
8.2656
Epoch 4/200
300/300 [==============================] - 1s 3ms/step - loss: 31889.1504 - val_loss: 2716
1.3828
Epoch 5/200
300/300 [==============================] - 1s 3ms/step - loss: 29893.9629 - val_loss: 2766
1.0078
Epoch 6/200
300/300 [==============================] - 1s 3ms/step - loss: 31074.9277 - val_loss: 3067
7.7324
Epoch 7/200
300/300 [==============================] - 1s 3ms/step - loss: 30106.0059 - val_loss: 2958
2.5293
Epoch 8/200
300/300 [==============================] - 1s 3ms/step - loss: 30027.0410 - val_loss: 3192
5.8008
Epoch 9/200
300/300 [==============================] - 1s 3ms/step - loss: 28520.9277 - val_loss: 2416
3.5293
Epoch 10/200
300/300 [==============================] - 1s 3ms/step - loss: 29037.2305 - val_loss: 2643
4.0040
```

Over here we evaluate the model based on test data previously split and make predictions

Evaluation

```
model.evaluate(valid_ds_pd, y_true)
```

```
12/12 [==============================] - 0s 2ms/step - loss: 22154.5156
```

```
22154.515625
```

```
predictions = model.predict(valid_ds_pd)
```

```
12/12 [==============================] - 0s 2ms/step
```

```
full_error = 0

for i in range(len(predictions)):
    full_error += abs(predictions[i] - y_true[i])

mean_error = full_error / len(predictions)
print(mean_error)
```

```
[22154.516]
```

The number here represents the average error of price prediction, which is around 22 154 $

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 768)               56832

 dense_1 (Dense)             (None, 256)               196864

 dense_2 (Dense)             (None, 32)                8224

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 261,953
Trainable params: 261,953
Non-trainable params: 0
_________________________________________________________________
```

Now we do the same steps for test data.
We apply column deletion, one_hot_encoding as well as null replacement and tensor conversion

Prediction ¶

```
test_data = pd.read_csv(TEST_FILE)
ids = test_data.pop('Id')

test_data = delete_columns(test_data)
test_data = one_hot_encoder(test_data)

for i in range(len(test_data)):
    test_data.iloc[i], nulls_replaced = replace_nulls(test_data.iloc[i])

test_data = return_tensor(test_data)

preds = model.predict(test_data)
output = pd.DataFrame({'Id': ids, 'SalePrice': preds.squeeze()})

output.head()
```

```
Deleting columns...

Deleting Alley column
Deleting MasVnrType column
Deleting FireplaceQu column
Deleting PoolQC column
Deleting Fence column
Deleting MiscFeature column


/tmp/ipykernel_20/1851572587.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  row[col] = 0

46/46 [==============================] - 0s 1ms/step
```

Lastly we export it to kaggle for submission.

```
kaggle_df = pd.read_csv('../input/house-prices-advanced-regression-techniques/sample_submissi
on.csv')
kaggle_df['SalePrice'] = model.predict(test_data)
kaggle_df.to_csv('/kaggle/working/submission.csv', index=False)
kaggle_df.head()
```

```
46/46 [==============================] - 0s 1ms/step
```

	Id	SalePrice
0	1461	145477.500000
1	1462	159610.390625
2	1463	187395.968750
3	1464	190421.796875
4	1465	168757.390625