House Price Prediction with Linear Regression

In this assignment, you're going to predict the price of a house using information like its location, area, no. of rooms etc. You'll use the dataset from the House Prices - Advanced Regression Techniques competition on Kaggle. We'll follow a step-by-step process to train our model:

1. Download and explore the data
2. Prepare the dataset for training
3. Train a linear regression model
4. Make predictions and evaluate the model

As you go through this notebook, you will find a **???** in certain places. Your job is to replace the **???** with appropriate code or values, to ensure that the notebook runs properly end-to-end and your machine learning model is trained properly without errors.

Guidelines

1. Make sure to run all the code cells in order. Otherwise, you may get errors like `NameError` for undefined variables.
2. Do not change variable names, delete cells, or disturb other existing code. It may cause problems during evaluation.
3. In some cases, you may need to add some code cells or new statements before or after the line of code containing the **???**.
4. Since you'll be using a temporary online service for code execution, save your work by running `jovian.commit` at regular intervals.
5. Review the "Evaluation Criteria" for the assignment carefully and make sure your submission meets all the criteria.
6. Questions marked **(Optional)** will not be considered for evaluation and can be skipped. They are for your learning.
7. It's okay to ask for help & discuss ideas on the community forum, but please don't post full working code, to give everyone an opportunity to solve the assignment on their own.

Important Links:

- Make a submission here: https://jovian.ai/learn/machine-learning-with-python-zero-to-gbms/assignment/assignment-1-train-your-first-ml-model
- Ask questions, discuss ideas and get help here: https://jovian.ai/forum/c/zero-to-gbms/gbms-assignment-1/100
- Review the following notebooks:

- https://jovian.ai/aakashns/python-sklearn-linear-regression
- https://jovian.ai/aakashns/python-sklearn-logistic-regression

How to Run the Code and Save Your Work

Option 1: Running using free online resources (1-click, recommended): The easiest way to start executing the code is to click the **Run** button at the top of this page and select **Run on Binder**. This will set up a cloud-based Jupyter notebook server and allow you to modify/execute the code.

Option 2: Running on your computer locally: To run the code on your computer locally, you'll need to set up Python, download the notebook and install the required libraries. Click the **Run** button at the top of this page, select the **Run Locally** option, and follow the instructions.

Saving your work: You can save a snapshot of the assignment to your Jovian profile, so that you can access it later and continue your work. Keep saving your work by running `jovian.commit` from time to time.

```
!pip install jovian scikit-learn --upgrade --quiet
```

```
import jovian
```

```
jovian.commit(project='python-sklearn-assignment', privacy='secret')
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Let's begin by installing the required libraries:

```
!pip install numpy pandas matplotlib seaborn plotly opendatasets jovian --quiet
```

Step 1 - Download and Explore the Data

The dataset is available as a ZIP file at the following url:

```
dataset_url = 'https://github.com/JovianML/opendatasets/raw/master/data/house-prices-ac
```

We'll use the `urlretrieve` function from the module `urllib.request` to dowload the dataset.

```
from urllib.request import urlretrieve
```

```
urlretrieve(dataset_url, 'house-prices.zip')
```

```
('house-prices.zip', <http.client.HTTPMessage at 0x7f4626c183d0>)
```

The file `housing-prices.zip` has been downloaded. Let's unzip it using the [`zipfile`](#) module.

```
from zipfile import ZipFile
```

```
with ZipFile('house-prices.zip') as f:
    f.extractall(path='house-prices')
```

The dataset is extracted to the folder `house-prices`. Let's view the contents of the folder using the [`os`](#) module.

```
import os
```

```
data_dir = 'house-prices'
```

```
os.listdir(data_dir)
```

```
['data_description.txt', 'sample_submission.csv', 'test.csv', 'train.csv']
```

Use the "File" > "Open" menu option to browse the contents of each file. You can also check out the [dataset description](#) on Kaggle to learn more.

We'll use the data in the file `train.csv` for training our model. We can load the for processing using the [Pandas](#) library.

```
import pandas as pd
pd.options.display.max_columns = 200
pd.options.display.max_rows = 200
```

```
train_csv_path = data_dir + '/train.csv'
train_csv_path
```

```
'house-prices/train.csv'
```

> **QUESTION 1**: Load the data from the file `train.csv` into a Pandas data frame.

```
prices_df = pd.read_csv('house-prices/train.csv')
```

```
prices_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfi
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Insid
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Insid
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corne

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfi
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR
...	...	...	...	...	...	...	...	...	...	...	.
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Insid
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Insid
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	Insid
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	Insid
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	Insid

1460 rows × 81 columns

Let's explore the columns and data types within the dataset.

```
prices_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
```

```
24  Exterior2nd    1460 non-null   object
25  MasVnrType     1452 non-null   object
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
60  GarageFinish   1379 non-null   object
61  GarageCars     1460 non-null   int64
62  GarageArea     1460 non-null   int64
63  GarageQual     1379 non-null   object
64  GarageCond     1379 non-null   object
65  PavedDrive     1460 non-null   object
66  WoodDeckSF     1460 non-null   int64
```

```
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

> **QUESTION 2**: How many rows and columns does the dataset contain?

```
n_rows = len(prices_df)
n_rows
```

```
1460
```

```
n_cols = len(list(prices_df.columns))
n_cols
```

```
81
```

```
print('The dataset contains {} rows and {} columns.'.format(n_rows, n_cols))
```

```
The dataset contains 1460 rows and 81 columns.
```

> **(OPTIONAL) QUESTION**: Before training the model, you may want to explore and visualize data from the various columns within the dataset, and study their relationship with the price of the house (using scatter plot and correlations). Create some graphs and summarize your insights using the empty cells below.

```
prices_df.LotArea.corr(prices_df.SalePrice)
```

```
0.2638433538714056
```

```

```

```

```

Let's save our work before continuing.

```python
import jovian
```

```python
jovian.commit()
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Step 2 - Prepare the Dataset for Training

Before we can train the model, we need to prepare the dataset. Here are the steps we'll follow:

1. Identify the input and target column(s) for training the model.
2. Identify numeric and categorical input columns.
3. Impute (fill) missing values in numeric columns
4. Scale values in numeric columns to a $(0, 1)$ range.
5. Encode categorical data into one-hot vectors.
6. Split the dataset into training and validation sets.

Identify Inputs and Targets

While the dataset contains 81 columns, not all of them are useful for modeling. Note the following:

- The first column `Id` is a unique ID for each house and isn't useful for training the model.
- The last column `SalePrice` contains the value we need to predict i.e. it's the target column.
- Data from all the other columns (except the first and the last column) can be used as inputs to the model.

```python
prices_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfi
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Insid
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Insid
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corn
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR
...	...	...	...	...	...	...	...	...	...	...	.

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfi
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Insid
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Insid
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	Insid
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	Insid
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	Insid

1460 rows × 81 columns

> **QUESTION 3**: Create a list `input_cols` of column names containing data that can be used as input to train the model, and identify the target column as the variable `target_col` .

```
# Identify the input columns (a list of column names)
input_cols = list(prices_df.columns [1:-1])
input_cols
```

```
['MSSubClass',
 'MSZoning',
 'LotFrontage',
 'LotArea',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'OverallQual',
 'OverallCond',
 'YearBuilt',
 'YearRemodAdd',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'MasVnrArea',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinSF1',
 'BsmtFinType2',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 '1stFlrSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'GrLivArea',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath',
 'BedroomAbvGr',
 'KitchenAbvGr',
 'KitchenQual',
 'TotRmsAbvGrd',
 'Functional',
 'Fireplaces',
 'FireplaceQu',
 'GarageType',
 'GarageYrBlt',
 'GarageFinish',
 'GarageCars',
 'GarageArea',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'PoolArea',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'MiscVal',
 'MoSold',
 'YrSold',
 'SaleType',
 'SaleCondition']
```

```python
# Identify the name of the target column (a single string, not a list)
target_col = 'SalePrice'
```

```python
print(list(input_cols))
```

```
['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'Alley', 'LotShape',
'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1',
'Condition2', 'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt',
'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2',
'BsmtUnfSF', 'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical',
'1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish',
'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF',
'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType', 'SaleCondition']
```

```python
len(input_cols)
```

```
79
```

```python
print(target_col)
```

```
SalePrice
```

Make sure that the `Id` and `SalePrice` columns are not included in `input_cols`.

Now that we've identified the input and target columns, we can separate input & target data.

```python
inputs_df = prices_df[input_cols].copy()
```

```python
targets = prices_df[target_col]
```

```python
inputs_df
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	Lan
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	
...	...	...	...	...	...	...	...	...	...	...	

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	Lan
1455	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Inside	
1456	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Inside	
1457	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	Inside	
1458	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	Inside	
1459	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	Inside	

1460 rows × 79 columns

```
targets
```

```
0       208500
1       181500
2       223500
3       140000
4       250000
         ...
1455    175000
1456    210000
1457    266500
1458    142125
1459    147500
Name: SalePrice, Length: 1460, dtype: int64
```

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Identify Numeric and Categorical Data

The next step in data preparation is to identify numeric and categorical columns. We can do this by looking at the data type of each column.

```
prices_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Id              1460 non-null   int64
```

```
1   MSSubClass     1460 non-null   int64
2   MSZoning       1460 non-null   object
3   LotFrontage    1201 non-null   float64
4   LotArea        1460 non-null   int64
5   Street         1460 non-null   object
6   Alley          91 non-null     object
7   LotShape       1460 non-null   object
8   LandContour    1460 non-null   object
9   Utilities      1460 non-null   object
10  LotConfig      1460 non-null   object
11  LandSlope      1460 non-null   object
12  Neighborhood   1460 non-null   object
13  Condition1     1460 non-null   object
14  Condition2     1460 non-null   object
15  BldgType       1460 non-null   object
16  HouseStyle     1460 non-null   object
17  OverallQual    1460 non-null   int64
18  OverallCond    1460 non-null   int64
19  YearBuilt      1460 non-null   int64
20  YearRemodAdd   1460 non-null   int64
21  RoofStyle      1460 non-null   object
22  RoofMatl       1460 non-null   object
23  Exterior1st    1460 non-null   object
24  Exterior2nd    1460 non-null   object
25  MasVnrType     1452 non-null   object
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
```

```
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

> **QUESTION 4**: Crate two lists `numeric_cols` and `categorical_cols` containing names of numeric and categorical input columns within the dataframe respectively. Numeric columns have data types `int64` and `float64`, whereas categorical columns have the data type `object`.

> *Hint*: See this StackOverflow question.

```
import numpy as np
```

```
numeric_cols = inputs_df.select_dtypes(include=['int64', 'float64']).columns.tolist()
```

```
categorical_cols =  inputs_df.select_dtypes(include=['object']).columns.tolist()
```

```
print(list(numeric_cols))
```

```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
'1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold']
```

```
print(list(categorical_cols))
```

```
['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual',
'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature', 'SaleType', 'SaleCondition']
```

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Impute Numerical Data

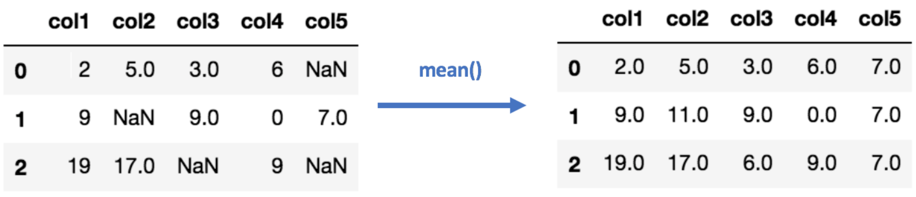
Some of the numeric columns in our dataset contain missing values (nan).

```
missing_counts = inputs_df[numeric_cols].isna().sum().sort_values(ascending=False)
missing_counts[missing_counts > 0]
```

```
LotFrontage     259
```

```
GarageYrBlt      81
MasVnrArea        8
dtype: int64
```

Machine learning models can't work with missing data. The process of filling missing values is called imputation.



	col1	col2	col3	col4	col5
0	2	5.0	3.0	6	NaN
1	9	NaN	9.0	0	7.0
2	19	17.0	NaN	9	NaN

	col1	col2	col3	col4	col5
0	2.0	5.0	3.0	6.0	7.0
1	9.0	11.0	9.0	0.0	7.0
2	19.0	17.0	6.0	9.0	7.0

There are several techniques for imputation, but we'll use the most basic one: replacing missing values with the average value in the column using the `SimpleImputer` class from `sklearn.impute`.

```
from sklearn.impute import SimpleImputer
```

> **QUESTION 5**: Impute (fill) missing values in the numeric columns of `inputs_df` using a `SimpleImputer`.
>
> *Hint*: See this notebook.

```
# 1. Create the imputer
imputer = SimpleImputer(strategy = 'mean')
```

```
# 2. Fit the imputer to the numeric colums
imputer.fit(inputs_df[numeric_cols])
```

```
SimpleImputer()
```

```
# 3. Transform and replace the numeric columns
inputs_df[numeric_cols] = imputer.transform(inputs_df[numeric_cols])
```

After imputation, none of the numeric columns should contain any missing values.

```
missing_counts = inputs_df[numeric_cols].isna().sum().sort_values(ascending=False)
missing_counts[missing_counts > 0] # should be an empty list
```

```
Series([], dtype: int64)
```

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
```

```
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Scale Numerical Values

The numeric columns in our dataset have varying ranges.

```
inputs_df[numeric_cols].describe().loc[['min', 'max']]
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin
min	20.0	21.0	1300.0	1.0	1.0	1872.0	1950.0	0.0	
max	190.0	313.0	215245.0	10.0	9.0	2010.0	2010.0	1600.0	56

A good practice is to scale numeric features to a small range of values e.g. $(0, 1)$. Scaling numeric features ensures that no particular feature has a disproportionate impact on the model's loss. Optimization algorithms also work better in practice with smaller numbers.

> **QUESTION 6**: Scale numeric values to the $(0, 1)$ range using `MinMaxScaler` from `sklearn.preprocessing`.
>
> *Hint*: See this notebook.

```
from sklearn.preprocessing import MinMaxScaler
```

```
# Create the scaler
scaler = MinMaxScaler()
```

```
# Fit the scaler to the numeric columns
scaler.fit(inputs_df[numeric_cols])
```

```
MinMaxScaler()
```

```
# Transform and replace the numeric columns
inputs_df[numeric_cols] = scaler.transform(inputs_df[numeric_cols])
```

After scaling, the ranges of all numeric columns should be $(0, 1)$.

```
inputs_df[numeric_cols].describe().loc[['min', 'max']]
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinS
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
max	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Encode Categorical Columns

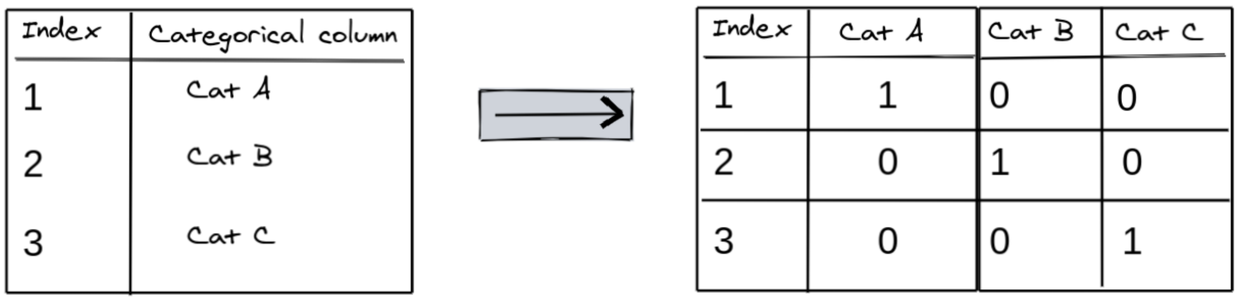
Our dataset contains several categorical columns, each with a different number of categories.

```
inputs_df[categorical_cols].nunique().sort_values(ascending=False)
```

```
Neighborhood     25
Exterior2nd      16
Exterior1st      15
SaleType          9
Condition1        9
Condition2        8
HouseStyle        8
RoofMatl          8
Functional        7
BsmtFinType2      6
Heating           6
RoofStyle         6
SaleCondition     6
BsmtFinType1      6
GarageType        6
Foundation        6
Electrical        5
FireplaceQu       5
HeatingQC         5
GarageQual        5
GarageCond        5
MSZoning          5
LotConfig         5
ExterCond         5
BldgType          5
BsmtExposure      4
MiscFeature       4
Fence             4
LotShape          4
LandContour       4
BsmtCond          4
KitchenQual       4
MasVnrType        4
ExterQual         4
```

```
BsmtQual        4
LandSlope       3
GarageFinish    3
PavedDrive      3
PoolQC          3
Utilities       2
CentralAir      2
Street          2
Alley           2
dtype: int64
```

Since machine learning models can only be trained with numeric data, we need to convert categorical data to numbers. A common technique is to use one-hot encoding for categorical columns.



One hot encoding involves adding a new binary (0/1) column for each unique category of a categorical column.

> **QUESTION 7**: Encode categorical columns in the dataset as one-hot vectors using `OneHotEncoder` from `sklearn.preprocessing`. Add a new binary (0/1) column for each category
>
> *Hint*: See this notebook.

```python
from sklearn.preprocessing import OneHotEncoder
```

```python
# 1. Create the encoder
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
```

```python
# 2. Fit the encoder to the categorical colums
encoder.fit(inputs_df[categorical_cols])
```

```
OneHotEncoder(handle_unknown='ignore', sparse=False)
```

```python
# 3. Generate column names for each category
encoded_cols = list(encoder.get_feature_names(categorical_cols))
len(encoded_cols)
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning:
Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and
```

```
will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

```
268
```

```
# 4. Transform and add new one-hot category columns
inputs_df[encoded_cols] = encoder.transform(inputs_df[categorical_cols])
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: PerformanceWarning:
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert`
many times, which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

The new one-hot category columns should now be added to `inputs_df` .

```
inputs_df
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	La
0	0.235294	RL	0.150685	0.033420	Pave	NaN	Reg	Lvl	AllPub	Inside	
1	0.000000	RL	0.202055	0.038795	Pave	NaN	Reg	Lvl	AllPub	FR2	
2	0.235294	RL	0.160959	0.046507	Pave	NaN	IR1	Lvl	AllPub	Inside	
3	0.294118	RL	0.133562	0.038561	Pave	NaN	IR1	Lvl	AllPub	Corner	
4	0.235294	RL	0.215753	0.060576	Pave	NaN	IR1	Lvl	AllPub	FR2	
...	...	...	...	...	...	...	...	...	...	...	
1455	0.235294	RL	0.140411	0.030929	Pave	NaN	Reg	Lvl	AllPub	Inside	
1456	0.000000	RL	0.219178	0.055505	Pave	NaN	Reg	Lvl	AllPub	Inside	
1457	0.294118	RL	0.154110	0.036187	Pave	NaN	Reg	Lvl	AllPub	Inside	
1458	0.000000	RL	0.160959	0.039342	Pave	NaN	Reg	Lvl	AllPub	Inside	
1459	0.000000	RL	0.184932	0.040370	Pave	NaN	Reg	Lvl	AllPub	Inside	

1460 rows × 347 columns

Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Training and Validation Set

Finally, let's split the dataset into a training and validation set. We'll use a randomly select 25% subset of the data for validation. Also, we'll use just the numeric and encoded columns, since the inputs to our model must be numbers.

```
from sklearn.model_selection import train_test_split
```

```
train_inputs, val_inputs, train_targets, val_targets = train_test_split(inputs_df[numer
                                                                        targets,
                                                                        test_size=0.25,
                                                                        random_state=42
```

```
train_inputs
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtF
1023	0.588235	0.075342	0.008797	0.666667	0.500	0.963768	0.933333	0.008750	0.0C
810	0.000000	0.195205	0.041319	0.555556	0.625	0.739130	0.816667	0.061875	0.11
1384	0.176471	0.133562	0.036271	0.555556	0.500	0.485507	0.000000	0.000000	0.03
626	0.000000	0.167979	0.051611	0.444444	0.500	0.637681	0.466667	0.000000	0.0C
813	0.000000	0.184932	0.039496	0.555556	0.625	0.623188	0.133333	0.151875	0.1C
...	...	...	...	...	...	...	...	...	
1095	0.000000	0.195205	0.037472	0.555556	0.500	0.971014	0.933333	0.000000	0.0C
1130	0.176471	0.150685	0.030400	0.333333	0.250	0.405797	0.000000	0.000000	0.11
1294	0.000000	0.133562	0.032120	0.444444	0.750	0.601449	0.666667	0.000000	0.02
860	0.176471	0.116438	0.029643	0.666667	0.875	0.333333	0.800000	0.000000	0.0C
1126	0.588235	0.109589	0.011143	0.666667	0.500	0.978261	0.950000	0.081250	0.0C

1095 rows × 304 columns

```
train_targets
```

```
1023    191000
810     181000
1384    105000
626     139900
813     157900
         ...
1095    176432
1130    135000
1294    115000
860     189950
1126    174000
Name: SalePrice, Length: 1095, dtype: int64
```

```
val_inputs
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtF
892	0.000000	0.167808	0.033252	0.555556	0.875	0.659420	0.883333	0.000000	0.11
1105	0.235294	0.263699	0.051209	0.777778	0.500	0.884058	0.750000	0.226250	0.18
413	0.058824	0.119863	0.035804	0.444444	0.625	0.398551	0.000000	0.000000	0.00
522	0.176471	0.099315	0.017294	0.555556	0.750	0.543478	0.000000	0.000000	0.07
1036	0.000000	0.232877	0.054210	0.888889	0.500	0.978261	0.966667	0.043750	0.18
...	...	...	...	...	...	...	...	...	
988	0.235294	0.167979	0.050228	0.555556	0.625	0.753623	0.433333	0.186250	0.02
243	0.823529	0.184932	0.044226	0.555556	0.625	0.782609	0.500000	0.000000	0.00
1342	0.235294	0.167979	0.037743	0.777778	0.500	0.942029	0.866667	0.093125	0.00
1057	0.235294	0.167979	0.133955	0.666667	0.625	0.884058	0.733333	0.000000	0.10
1418	0.000000	0.171233	0.036944	0.444444	0.500	0.659420	0.216667	0.000000	0.00

365 rows × 304 columns

```
val_targets
```

```
892     154500
1105    325000
413     115000
522     159000
1036    315500
         ...  
988     195000
243     120000
1342    228500
1057    248000
1418    124000
Name: SalePrice, Length: 365, dtype: int64
```

Let's save our work before continuing.

```
jovian.commit()
```

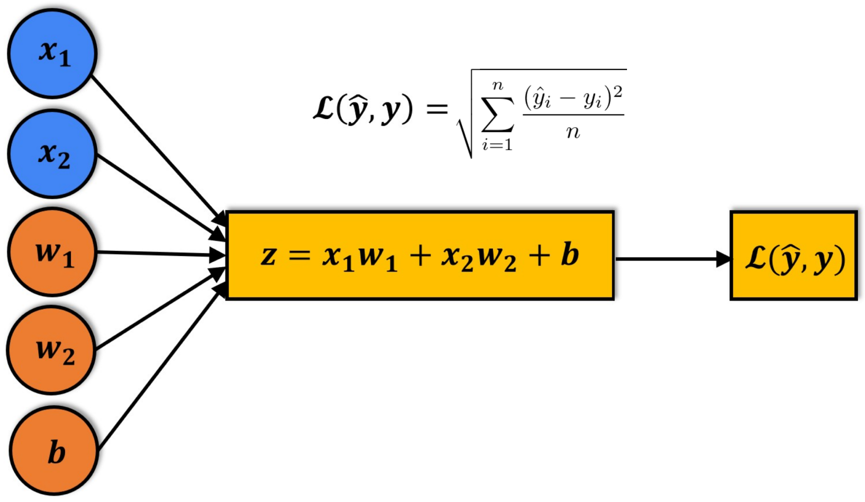
```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Step 3 - Train a Linear Regression Model

We're now ready to train the model. Linear regression is a commonly used technique for solving regression problems. In a linear regression model, the target is modeled as a linear combination (or weighted sum) of input features. The predictions from the model are evaluated using a loss function like the Root Mean Squared Error (RMSE).

Here's a visual summary of how a linear regression model is structured:


x_1
x_2
w_1
w_2
b
$\mathcal{L}(\hat{y}, y) = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$
$z = x_1 w_1 + x_2 w_2 + b$
$\mathcal{L}(\hat{y}, y)$

However, linear regression doesn't generalize very well when we have a large number of input columns with co-linearity i.e. when the values one column are highly correlated with values in other column(s). This is because it tries to fit the training data perfectly.

Instead, we'll use Ridge Regression, a variant of linear regression that uses a technique called L2 regularization to introduce another loss term that forces the model to generalize better. Learn more about ridge regression here: https://www.youtube.com/watch?v=Q81RR3yKn30

> **QUESTION 8**: Create and train a linear regression model using the `Ridge` class from `sklearn.linear_model`.

```
from sklearn.linear_model import Ridge
```

```
# Create the model
model = Ridge()
```

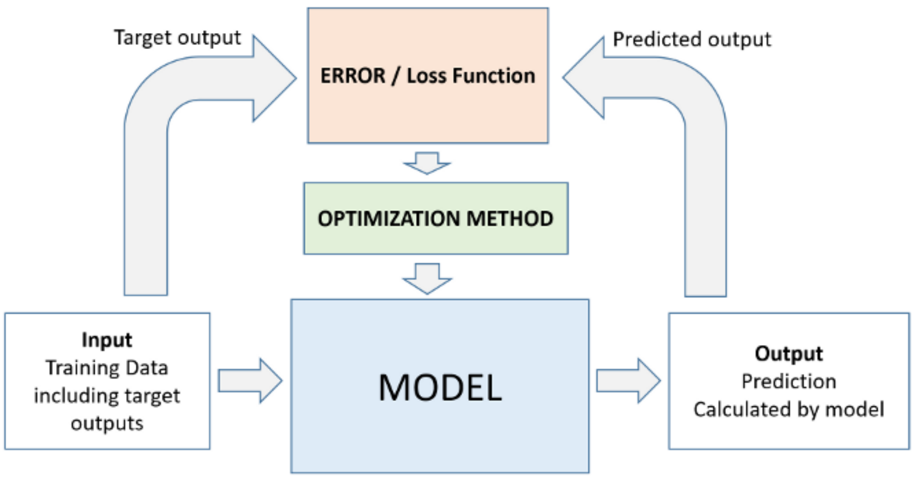
```
# Fit the model using inputs and targets
model.fit(train_inputs,train_targets)
```

```
Ridge()
```

`model.fit` uses the following strategy for training the model (source):

1. We initialize a model with random parameters (weights & biases).
2. We pass some inputs into the model to obtain predictions.
3. We compare the model's predictions with the actual targets using the loss function.
4. We use an optimization technique (like least squares, gradient descent etc.) to reduce the loss by adjusting the weights & biases of the model

5. We repeat steps 1 to 4 till the predictions from the model are good enough.



Let's save our work before continuing.

```
jovian.commit()
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on
https://jovian.ai
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-
assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Step 4 - Make Predictions and Evaluate Your Model

The model is now trained, and we can use it to generate predictions for the training and validation inputs. We can evaluate the model's performance using the RMSE (root mean squared error) loss function.

> **QUESTION 9**: Generate predictions and compute the RMSE loss for the training and validation sets.
>
> *Hint*: Use the `mean_squared_error` with the argument `squared=False` to compute RMSE loss.

```
from sklearn.metrics import mean_squared_error
```

```
train_preds = model.predict(train_inputs)
```

```
train_preds
```

```
array([172549.49239604, 176648.3841514 , 104461.18939205, ...,
       121549.23101908, 173504.31921626, 190778.41334452])
```

```
def rmse(targets, predictions):
    return np.sqrt(np.mean(np.square(targets - predictions)))
train_rmse = rmse(train_targets,train_preds)
```

```python
print('The RMSE loss for the training set is $ {}.'.format(train_rmse))
```

```
The RMSE loss for the training set is $ 21877.850450615537.
```

```python
val_preds = model.predict(val_inputs)
```

```python
val_preds
```

```
array([157673.50925059, 345745.87352179,  87613.27623036, 188833.72115116,
       338661.35543453,  63336.82182647, 248302.26661826, 148819.87161788,
        57119.99507449, 145872.68840307, 145100.99812253, 107298.29146324,
        98564.57311208, 225130.82499085, 172008.58322822, 131593.88782295,
       187592.61766972, 122661.63356877, 128586.23398817, 211939.300775  ,
       161320.55505759, 202909.99572299, 179658.90516251, 127274.09425341,
       201947.54169482, 141226.78402659, 201719.30695948, 102527.60973202,
       171451.3095456 , 212056.7308324 , 138930.82966494, 274828.76850522,
       233517.93452593, 108467.96304335, 248211.70373113, 145473.52369456,
       128041.00086456, 203601.10154157, 313641.53054834, 111163.55150571,
       134686.14358912, 225819.97523186,  97466.9691439 , 358759.18620223,
       136099.06173434, 145505.26662872, 100270.95219149, 138007.10547327,
       420684.90146174, 132960.9120818 , 118956.81942242, 255796.47324499,
        97964.96159595, 265661.19684948, 170449.49388262, 227268.42624662,
       206482.83224897, 178042.77857039, 134127.40000499,  91669.15451687,
        43787.38900189, 173223.8933693 , 316933.22499508, 256920.52817056,
       312250.68270351, 191780.31460864,  94220.28574208, 296604.56228788,
       117519.94599469, 176728.02884646, 129174.63844253, 117345.18244627,
       107702.04381479,  53360.83569561, 443315.95328461, 192660.98185041,
       300837.43363927, 367318.38476689, 149378.35844635, 117132.88448742,
       114256.01747004,  46434.43714028, 113617.48485558,  87195.4236113 ,
       163572.82002558, 140931.72621903, 254271.73535653, 211206.41953729,
       133017.73943949, 198209.23757938, 141626.69640064, 153607.69715563,
       171045.43632549, 270826.24301519, 116123.47783821, 193429.41322547,
       206730.81086883, 169654.70083924, 205500.42620025, 273084.92037488,
       145343.11758515, 207974.25168876, 269046.26884667, 147482.67871071,
       193016.0396471 , 193196.61800595, 147586.00397779, 298907.09354587,
       156859.52530761, 212860.70624188,  67038.50494827, 142493.90821743,
       147936.93703722, 135754.58122491, 202017.04523815, 130668.13741662,
        98164.20881016, 122296.17526403, 109376.05268123, 275235.31092977,
        96698.53208074, 134479.44821045, 182529.25743736, 183466.42730462,
       193408.55073875, 132847.57469644, 231565.87163444,  93234.12219544,
       147585.57196038, 193892.08004408, 186618.96685156, 331023.07533607,
       192652.06446298, 133799.70116641,  37971.46666017, 380984.61292508,
       347269.48061746, 147864.07610958, 228835.31411499, 554096.19431202,
       344006.48167601, 129697.79068106, 182491.30526219, 163525.5985915 ,
       120284.75050748, 107044.24406145, 239250.85561799, 192232.15991826,
       118199.82464732,  49920.41051051, 133842.1109494 , 128473.64589757,
       284292.87112169, 157396.27658182,  72491.78069519, 113012.96927176,
       135415.76878583, 154524.41755489,  94297.65639696, 134462.46793768,
       218436.65418721,  89554.43115538, 291458.33937309, 157126.7663051 ,
       112585.3798747 , 131280.77423482, 265331.1118754 , 334426.72417586,
       432162.77628621, 219956.35503457, 380906.85281654,  97537.44376028,
       116065.04331055, 147497.1992827 , 309032.16150658, 103673.64893082,
       123643.35028812, 219660.80533399, 133507.40950386, 158930.40785033,
       204386.28989677, 129012.19408285, 125745.03486332, 148714.09159668,
       232727.49929833, 105200.88161324, 257074.64504664, 216902.16508506,
       201712.97487137,  74231.89741742, 118891.96328707,  95905.78760187,
       161962.15347356,  75874.05378004, 196287.18819029, 200329.70840148,
       193620.25893871,  91753.69648904, 204199.62957758, 138890.08432347,
       267853.51333496, 221351.87350929, 107216.35033522, 304279.09243966,
       196353.39818795, 115025.36252037, 225245.14669293, 135461.08470117,
       134529.69783176, 102562.81634883, 214661.15857377, 157080.64180881,
       120147.24572196, 172973.24122754, 210008.183993  , 245740.05226185,
       212743.40993304, 134470.42971054, 141266.60441067, 117494.27868964,
       135644.40981439, 213338.60707176, 200739.88609453,  85906.95103593,
       228551.73435027, 137166.55987252,  77819.06749765,  92766.53910173,
       158307.10488227, 107163.4855985 ,  95100.72321353, 167694.91222237,
       110275.02924978, 119557.81219626, 228449.02975767, 141273.00398395,
       204304.01625792, 158551.05561211, 241989.51545112, 108315.56501908,
       110209.14229607, 257256.74398752, 223604.57107024, 469757.86891228,
       201639.85460518, 131210.05298424, 131008.48750419, 167863.28594426,
       145226.28314766, 100548.31404872, 173419.45736124, 164550.79981183,
       152121.86772536, 112610.84748793, 143628.57450355, 133974.36117532,
       100113.31585233, 130461.27845474, 180056.82346423, 237360.66536291,
       289011.45395851, 191818.8583351 , 135000.00382996, 232389.25441958,
       349153.71264355, 230613.12077487, 161979.76071856, 131754.36648084,
       111947.98282762, 189236.19189096, 398710.72127732, 234203.59927127,
       227252.71379545,  74852.38533671,  79501.81481469, 139940.43496391,
       130936.13163812, 306727.07758841, 259339.94100294, 121649.15560954,
       204839.96558154, 104337.19348058, 201319.50806873, 106488.4436549 ,
       301079.77319103, 181839.91449816, 203922.30400843, 142029.55462626,
       295059.33379676, 189109.76218371,  82292.68040212, 119970.64997634,
       137039.78780518, 172015.45148679,  93800.17706018, 174576.71244515,
       177606.46984527, 147115.42421691, 186265.51290699, 120673.54334129,
       182640.88732315, 234602.02595246, 126513.79537803, 140978.8829015 ,
       173187.56515908, 199484.35961338, 147453.15022641, 212754.47232352,
       234095.53505326, 103762.69666598, 125995.03388321, 175177.24522076,
       106561.23881585, 194925.04286021, 135521.71400327, 183488.59054237,
       188752.97385706, 171372.82323242, 304731.93147149,  57852.31807702,
       234455.5828873 , 145570.01654985, 115771.17779775,  84669.76607412,
       196028.52939132, 149619.86568234, 149415.59331953, 220004.19604427,
       158940.33252994,  91257.22670321, 155757.06883378, 136028.61828342,
       139509.0316083 , 219014.27965184, 158546.09906568, 112430.70068928,
       174349.17567299,  82198.69300757,  74800.74900319, 216580.92105659,
       188910.47543913, 135983.36151189, 135529.93344352, 177809.72807452,
       232299.79966589, 364136.06909293, 374399.92539393, 114647.65641632,
       226665.82868701, 129294.4477863 , 233387.51977745, 351458.02929839,
       294948.70290983, 187441.58626751, 226488.40103526, 142790.17369647,
       114643.94982322,  87717.30078644, 218112.89015028, 341357.86232945,
       185656.5204631 , 107352.57080924, 217861.2091599 , 258291.49924279,
       127231.48524982])
```

```
val_rmse = rmse(val_targets,val_preds)
```

```
print('The RMSE loss for the validation set is $ {}.'.format(val_rmse))
```

```
The RMSE loss for the validation set is $ 29009.302517872122.
```

Feature Importance

Let's look at the weights assigned to different columns, to figure out which columns in the dataset are the most important.

> **QUESTION 10**: Identify the weights (or coefficients) assigned to for different features by the model.
>
> *Hint:* Read the docs.

```
weights = model.coef_
```

Let's create a dataframe to view the weight assigned to each column.

```
weights_df = pd.DataFrame({
    'columns': train_inputs.columns,
    'weight': weights
}).sort_values('weight', ascending=False)
```

```
weights_df
```

	columns	weight
275	PoolQC_Ex	77483.612175
15	GrLivArea	74388.503609
13	2ndFlrSF	62433.996447
3	OverallQual	62055.123269
12	1stFlrSF	60726.614752
...	...	...
266	GarageCond_Ex	-21028.480031
21	KitchenAbvGr	-26474.353422
277	PoolQC_Gd	-72681.458501
102	Condition2_PosN	-84704.714012
125	RoofMatl_ClyTile	-139512.027823

304 rows × 2 columns

Can you tell which columns have the greatest impact on the price of the house?

Making Predictions

The model can be used to make predictions on new inputs using the following helper function:

```
def predict_input(single_input):
    input_df = pd.DataFrame([single_input])
    input_df[numeric_cols] = imputer.transform(input_df[numeric_cols])
    input_df[numeric_cols] = scaler.transform(input_df[numeric_cols])
    input_df[encoded_cols] = encoder.transform(input_df[categorical_cols].values)
    X_input = input_df[numeric_cols + encoded_cols]
    return model.predict(X_input)[0]
```

```
sample_input = { 'MSSubClass': 20, 'MSZoning': 'RL', 'LotFrontage': 77.0, 'LotArea': 93
 'Street': 'Pave', 'Alley': None, 'LotShape': 'IR1', 'LandContour': 'Lvl', 'Utilities':
 'LotConfig': 'Inside', 'LandSlope': 'Gtl', 'Neighborhood': 'NAmes', 'Condition1': 'Nor
 'BldgType': '1Fam', 'HouseStyle': '1Story', 'OverallQual': 4, 'OverallCond': 5, 'YearB
 'YearRemodAdd': 1959, 'RoofStyle': 'Gable', 'RoofMatl': 'CompShg', 'Exterior1st': 'Ply
 'Exterior2nd': 'Plywood', 'MasVnrType': 'None','MasVnrArea': 0.0,'ExterQual': 'TA','Ex
 'Foundation': 'CBlock','BsmtQual': 'TA','BsmtCond': 'TA','BsmtExposure': 'No','BsmtFir
 'BsmtFinSF1': 569,'BsmtFinType2': 'Unf','BsmtFinSF2': 0,'BsmtUnfSF': 381,
 'TotalBsmtSF': 950,'Heating': 'GasA','HeatingQC': 'Fa','CentralAir': 'Y','Electrical':
 '2ndFlrSF': 0, 'LowQualFinSF': 0, 'GrLivArea': 1225, 'BsmtFullBath': 1, 'BsmtHalfBath'
 'HalfBath': 1, 'BedroomAbvGr': 3, 'KitchenAbvGr': 1,'KitchenQual': 'TA','TotRmsAbvGrd'
 'Fireplaces': 0,'FireplaceQu': np.nan,'GarageType': np.nan,'GarageYrBlt': np.nan,'Gara
 'GarageArea': 0,'GarageQual': np.nan,'GarageCond': np.nan,'PavedDrive': 'Y', 'WoodDeck
 'EnclosedPorch': 0,'3SsnPorch': 0, 'ScreenPorch': 0, 'PoolArea': 0, 'PoolQC': np.nan,
 'MiscVal': 400, 'MoSold': 1, 'YrSold': 2010, 'SaleType': 'WD', 'SaleCondition': 'Norma
```

```
predicted_price = predict_input(sample_input)
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/base.py:445: UserWarning: X does not
have valid feature names, but OneHotEncoder was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: PerformanceWarning:
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert`
many times, which has poor performance.  Consider joining all columns at once using
pd.concat(axis=1) instead.  To get a de-fragmented frame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

```
print('The predicted sale price of the house is ${}'.format(predicted_price))
```

```
The predicted sale price of the house is $123884.48449778199
```

Change the values in `sample_input` above and observe the effects on the predicted price.

Saving the model

Let's save the model (along with other useful objects) to disk, so that we use it for making predictions without retraining.

```
import joblib
```

```
house_price_predictor = {
    'model': model,
    'imputer': imputer,
    'scaler': scaler,
    'encoder': encoder,
    'input_cols': input_cols,
    'target_col': target_col,
    'numeric_cols': numeric_cols,
    'categorical_cols': categorical_cols,
    'encoded_cols': encoded_cols
}
```

```
joblib.dump(house_price_predictor, 'house_price_predictor.joblib')
```

```
['house_price_predictor.joblib']
```

Congratulations on training and evaluating your first machine learning model using `scikit-learn` ! Let's save our work before continuing. We'll include the saved model as an output.

```
jovian.commit(outputs=['house_price_predictor.joblib'])
```

```
[jovian] Updating notebook "btech60309-19/python-sklearn-assignment" on https://jovian.ai
[jovian] Uploading additional outputs...
[jovian] Committed successfully! https://jovian.ai/btech60309-19/python-sklearn-assignment
```

```
'https://jovian.ai/btech60309-19/python-sklearn-assignment'
```

Make Submission

To make a submission, just execute the following cell:

```
jovian.submit('zerotogbms-a1')
```

You can also submit your Jovian notebook link on the assignment page: https://jovian.ai/learn/machine-learning-with-python-zero-to-gbms/assignment/assignment-1-train-your-first-ml-model

Make sure to review the evaluation criteria carefully. You can make any number of submissions, and only your final submission will be evalauted.

Ask questions, discuss ideas and get help here: https://jovian.ai/forum/c/zero-to-gbms/gbms-assignment-1/100