

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
```

```
|████████████████████| 68 kB 5.6 MB/s
|████████████████████| 9.7 MB 69.1 MB/s
Building wheel for uuid (setup.py) ... done
```

```
import jovian
```

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

```
[jovian] Updating notebook "krupatel2807/python-sklearn-assignment" on
https://jovian.ai
```

```
[jovian] Committed successfully! https://jovian.ai/krupatel2807/python-sklearn-assignment
```

```
'https://jovian.ai/krupatel2807/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 download the dataset.

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

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

```
('house-prices.zip', <http.client.HTTPMessage at 0x7fe226592340>)
```

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)
```

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

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(train_csv_path)
```

```
prices_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Insid

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig
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
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 = prices_df.shape[0]
```

```
n_cols = prices_df.shape[1]
```

```
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.

Let's save our work before continuing.

```
import jovian
```

```
jovian.commit()
```

[jovian] Updating notebook "krupatel2807/python-sklearn-assignment" on

<https://jovian.ai>

[jovian] Committed successfully! <https://jovian.ai/krupatel2807/python-sklearn-assignment>

'<https://jovian.ai/krupatel2807/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.

```
prices_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside
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	Inside
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
...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Inside
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Inside

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig
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 = prices_df.columns[1:-1]
```

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

```
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']
```

```
len(input_cols)
```

79

```
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.


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

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

```
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	
...	
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 "krupatel2807/python-sklearn-assignment" on <https://jovian.ai>

[jovian] Committed successfully! <https://jovian.ai/krupatel2807/python-sklearn-assignment>

'<https://jovian.ai/krupatel2807/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	GarageYrBltd	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: Create 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 "krupatel2807/python-sklearn-assignment" on <https://jovian.ai>
[jovian] Committed successfully! <https://jovian.ai/krupatel2807/python-sklearn-assignment>
'<https://jovian.ai/krupatel2807/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			col1	col2	col3	col4	col5	
0	2	5.0	3.0	6	NaN	mean()		0	2.0	5.0	3.0	6.0	7.0
1	9	NaN	9.0	0	7.0			1	9.0	11.0	9.0	0.0	7.0
2	19	17.0	NaN	9	NaN			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 columns
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()
```

Scale Numerical Values

The numeric columns in our dataset have varying ranges.

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

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])
```

```
# 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']]
```

Let's save our work before continuing.

```
jovian.commit()
```


Encode Categorical Columns

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

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

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.

Index	Categorical column			
1	Cat A			
2	Cat B			
3	Cat C			



Index	Cat A	Cat B	Cat C
1	1	0	0
2	0	1	0
3	0	0	1

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](#).

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

1. Create the encoder

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

2. Fit the encoder to the categorical columns

```
encoder.fit(inputs_df[categorical_cols])
```

3. Generate column names for each category

```
encoded_cols = list(encoder.get_feature_names(categorical_cols))  
len(encoded_cols)
```

4. Transform and add new one-hot category columns

```
inputs_df[encoded_cols] = encoder.transform(inputs_df[categorical_cols])
```

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

```
inputs_df
```

Let's save our work before continuing.

```
jovian.commit()
```

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[numeric_columns], targets,
                                                                           test_size=0.25,
                                                                           random_state=42)
```

```
train_inputs
```

NameError Traceback (most recent call last)

<ipython-input-99-5c3085c3e44f> in <module>

----> 1 train_inputs

NameError: name 'train_inputs' is not defined

```
train_targets
```

```
val_inputs
```

```
val_targets
```

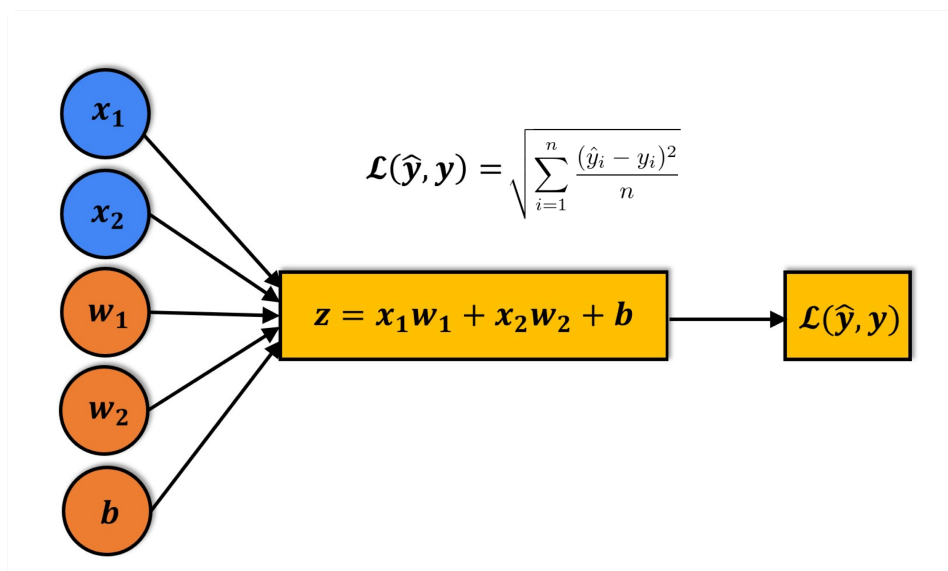
Let's save our work before continuing.

```
jovian.commit()
```

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:



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)
```

`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()
```

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
```

```
-----
ImportError                                Traceback (most recent call last)
<ipython-input-97-534151e31f77> in <module>
----> 1 from sklearn.metrics import mean_squared_error

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/__init__.py in <module>
    18 from ._ranking import top_k_accuracy_score
    19
---> 20 from ._classification import accuracy_score
    21 from ._classification import balanced_accuracy_score
    22 from ._classification import class_likelihood_ratios

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py in <module>
    41 from ..utils.validation import _num_samples
    42 from ..utils.sparsefuncs import count_nonzero
---> 43 from ..utils._param_validation import validate_params
    44 from ..exceptions import UndefinedMetricWarning
    45

/usr/local/lib/python3.8/dist-packages/sklearn/utils/_param_validation.py in <module>
    15 from scipy.sparse import csr_matrix
    16
---> 17 from .validation import _is_arraylike_not_scalar
    18
    19

ImportError: cannot import name '_is_arraylike_not_scalar' from
'sklearn.utils.validation' (/usr/local/lib/python3.8/dist-
packages/sklearn/utils/validation.py)
```

NOTE: If your import is failing due to a missing package, you can manually install dependencies using either `!pip` or `!apt`.

To view examples of installing some common dependencies, click the "Open Examples" button below.

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

```
NameError                                Traceback (most recent call last)
<ipython-input-98-699b347cac26> in <module>
----> 1 train_preds = model.predict(train_inputs)
```

NameError: name 'model' is not defined

```
train_preds
```

```
train_rmse = mean_squared_error(train_targets, train_preds, squared=False)
```

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

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

```
val_preds
```

```
val_rmse = mean_squared_error(val_targets, val_preds, squared=False)
```

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

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	65545.193876
132	RoofMatl_WdShngl	55227.753606
278	PoolQC_nan	42795.899250
86	Neighborhood_StoneBr	40698.278299
260	GarageQual_Ex	38461.559810
...
217	Heating_OthW	-19693.222461
241	Functional_Sev	-22687.663108
277	PoolQC_Gd	-89227.282728
102	Condition2_PosN	-89376.833635
125	RoofMatl_ClyTile	-145623.242279

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': 'Lv1', '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', 'BsmtFin
'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)
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-96-eb2f8253e8b4> in <module>  
----> 1 predicted_price = predict_input(sample_input)  
  
<ipython-input-94-09cdc347837d> in predict_input(single_input)  
      1 def predict_input(single_input):  
      2     input_df = pd.DataFrame([single_input])  
----> 3     input_df[numeric_cols] = imputer.transform(input_df[numeric_cols])  
      4     input_df[numeric_cols] = scaler.transform(input_df[numeric_cols])  
      5     input_df[encoded_cols] =  
encoder.transform(input_df[categorical_cols].values)
```

NameError: name 'imputer' is not defined

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

The predicted sale price of the house is \$-35981.22701685922

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  
}
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-93-cf3b3dc1770f> in <module>  
      1 house_price_predictor = {  
----> 2     'model': model,  
      3     'imputer': imputer,  
      4     'scaler': scaler,  
      5     'encoder': encoder,
```

NameError: name 'model' is not defined

```
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] Detected Colab notebook...

[jovian] `jovian.commit()` is no longer required on Google Colab. If you ran this notebook from Jovian,

then just save this file in Colab using `Ctrl+S/Cmd+S` and it will be updated on Jovian. Also, you can also delete this cell, it's no longer necessary.

Make Submission

To make a submission, just execute the following cell:

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

[jovian] Detected Colab notebook...

[jovian] `jovian.commit()` is no longer required on Google Colab. If you ran this notebook from Jovian,

then just save this file in Colab using `Ctrl+S/Cmd+S` and it will be updated on Jovian. Also, you can also delete this cell, it's no longer necessary.

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 evaluated.

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