🚜 Predicting the Sale Price of Bulldozers using Machine Learning

In this notebook, we're going to go through an example machine learning project with the goal of predicting the sale price of bulldozers.

1. Problem defition

> How well can we predict the future sale price of a bulldozer, given its characteristics and previous examples of how much similar bulldozers have been sold for?

2. Data

The data is downloaded from the Kaggle Bluebook for Bulldozers competition: https://www.kaggle.com/c/bluebook-for-bulldozers/data There are 3 main datasets:

- Train.csv is the training set, which contains data through the end of 2011.
- Valid.csv is the validation set, which contains data from January 1, 2012 - April 30, 2012 You make predictions on this set throughout the majority of the competition. Your score on this set is used to create the public leaderboard.
- Test.csv is the test set, which won't be released until the last week of the competition. It contains data from May 1, 2012 - November 2012. Your score on the test set determines your final rank for the competition.

3. Evaluation

The evaluation metric for this competition is the RMSLE (root mean squared log error) between the actual and predicted auction prices.

For more on the evaluation of this project check: https://www.kaggle.com/c/bluebook-for-bulldozers/overview/evaluation

Note: The goal for most regression evaluation metrics is to minimize the error. For example, our goal for this project will be to build a machine learning model which minimises RMSLE.

4. Features

Kaggle provides a data dictionary detailing all of the features of the dataset. You can view this data dictionary on Google Sheets: https://docs.google.com/spreadsheets/d/18ly-bLR8sbDJLITkWG7ozKm8l3RyieQ2Fpgix-beSYI/edit?usp=sharing

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
```

In [139…

```
# Import training and validation sets
# The argument "low_memory=False" is used to indicate that the entire
#file should be read into memory at once, rather than attempting to read it in smaller c
df = pd.read_csv(r"C:\Users\sonjo\TrainAndValid.csv",
                 low_memory=False)
df.head()
```

Out[139]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	Us
0	1139246	66000.0	999089	3157	121	3.0	2004	68.0	
1	1139248	57000.0	117657	77	121	3.0	1996	4640.0	
2	1139249	10000.0	434808	7009	121	3.0	2001	2838.0	
3	1139251	38500.0	1026470	332	121	3.0	2001	3486.0	
4	1139253	11000.0	1057373	17311	121	3.0	2007	722.0	

5 rows × 53 columns

In [140…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412698 entries, 0 to 412697
Data columns (total 53 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   SalesID                   412698 non-null  int64  
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64  
 3   ModelID                   412698 non-null  int64  
 4   datasource                412698 non-null  int64  
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64  
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   object 
 9   saledate                  412698 non-null  object 
 10  fiModelDesc               412698 non-null  object 
 11  fiBaseModel               412698 non-null  object 
 12  fiSecondaryDesc           271971 non-null  object 
 13  fiModelSeries             58667 non-null   object 
 14  fiModelDescriptor         74816 non-null   object 
 15  ProductSize               196093 non-null  object 
 16  fiProductClassDesc        412698 non-null  object 
 17  state                     412698 non-null  object 
 18  ProductGroup              412698 non-null  object 
 19  ProductGroupDesc          412698 non-null  object 
 20  Drive_System              107087 non-null  object 
 21  Enclosure                 412364 non-null  object 
 22  Forks                     197715 non-null  object 
 23  Pad_Type                  81096 non-null   object 
 24  Ride_Control              152728 non-null  object 
 25  Stick                     81096 non-null   object 
 26  Transmission              188007 non-null  object 
 27  Turbocharged              81096 non-null   object 
 28  Blade_Extension           25983 non-null   object 
 29  Blade_Width               25983 non-null   object 
 30  Enclosure_Type            25983 non-null   object 
 31  Engine_Horsepower         25983 non-null   object 
```

```
 32  Hydraulics                330133 non-null  object
 33  Pushblock                 25983 non-null   object
 34  Ripper                    106945 non-null  object
 35  Scarifier                 25994 non-null   object
 36  Tip_Control               25983 non-null   object
 37  Tire_Size                 97638 non-null   object
 38  Coupler                   220679 non-null  object
 39  Coupler_System            44974 non-null   object
 40  Grouser_Tracks            44875 non-null   object
 41  Hydraulics_Flow           44875 non-null   object
 42  Track_Type                102193 non-null  object
 43  Undercarriage_Pad_Width   102916 non-null  object
 44  Stick_Length              102261 non-null  object
 45  Thumb                     102332 non-null  object
 46  Pattern_Changer           102261 non-null  object
 47  Grouser_Type              102193 non-null  object
 48  Backhoe_Mounting          80712 non-null   object
 49  Blade_Type                81875 non-null   object
 50  Travel_Controls           81877 non-null   object
 51  Differential_Type         71564 non-null   object
 52  Steering_Controls         71522 non-null   object
dtypes: float64(3), int64(5), object(45)
memory usage: 166.9+ MB
```

In [141…

```
df.isna().sum()
```

Out[141]:

```
SalesID                          0
SalePrice                        0
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                 20136
YearMade                         0
MachineHoursCurrentMeter    265194
UsageBand                   339028
saledate                         0
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
```

```
Grouser_Tracks             367823
Hydraulics_Flow            367823
Track_Type                 310505
Undercarriage_Pad_Width    309782
Stick_Length               310437
Thumb                      310366
Pattern_Changer            310437
Grouser_Type               310505
Backhoe_Mounting           331986
Blade_Type                 330823
Travel_Controls            330821
Differential_Type          341134
Steering_Controls          341176
dtype: int64
```

In [142…

```
df.columns
```

Out[142]:

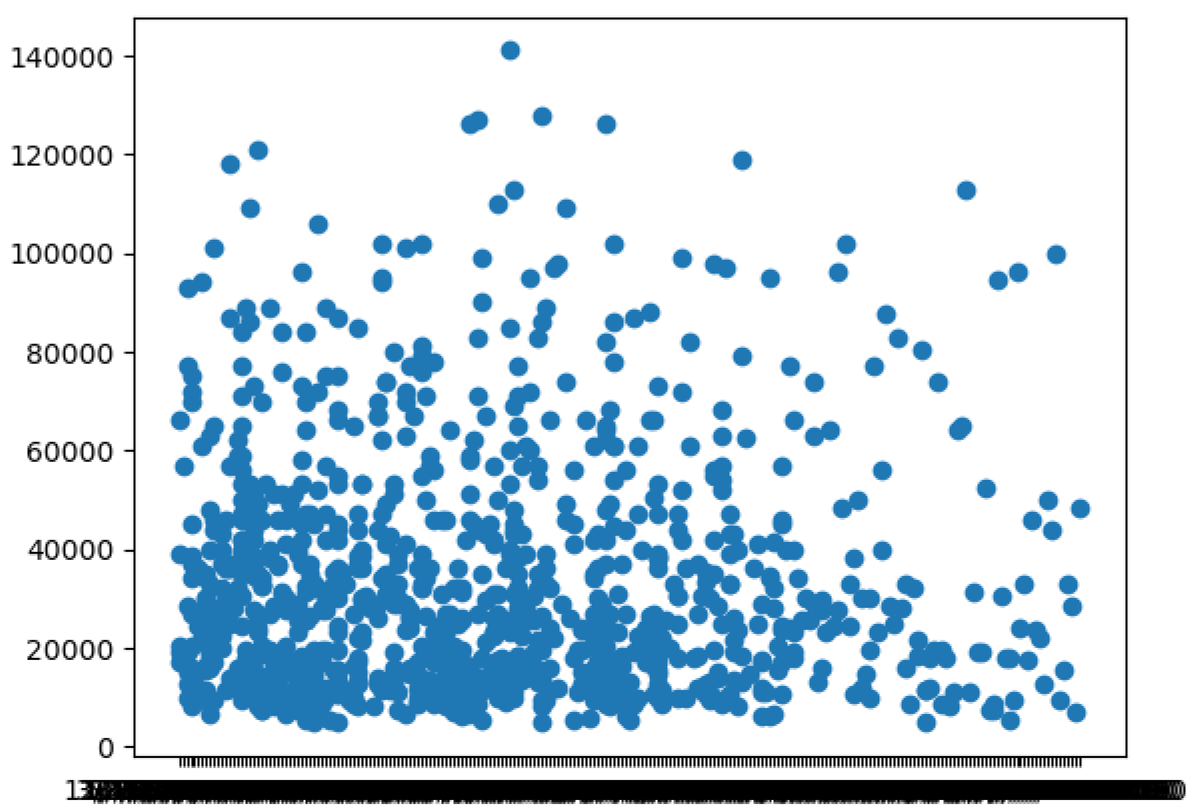
```
Index(['SalesID', 'SalePrice', 'MachineID', 'ModelID', 'datasource',
       'auctioneerID', 'YearMade', 'MachineHoursCurrentMeter', 'UsageBand',
       'saledate', 'fiModelDesc', 'fiBaseModel', 'fiSecondaryDesc',
       'fiModelSeries', 'fiModelDescriptor', 'ProductSize',
       'fiProductClassDesc', 'state', 'ProductGroup', 'ProductGroupDesc',
       'Drive_System', 'Enclosure', 'Forks', 'Pad_Type', 'Ride_Control',
       'Stick', 'Transmission', 'Turbocharged', 'Blade_Extension',
       'Blade_Width', 'Enclosure_Type', 'Engine_Horsepower', 'Hydraulics',
       'Pushblock', 'Ripper', 'Scarifier', 'Tip_Control', 'Tire_Size',
       'Coupler', 'Coupler_System', 'Grouser_Tracks', 'Hydraulics_Flow',
       'Track_Type', 'Undercarriage_Pad_Width', 'Stick_Length', 'Thumb',
       'Pattern_Changer', 'Grouser_Type', 'Backhoe_Mounting', 'Blade_Type',
       'Travel_Controls', 'Differential_Type', 'Steering_Controls'],
      dtype='object')
```

In [143…

```
fig, ax = plt.subplots()
ax.scatter(df["saledate"][:1000], df["SalePrice"][:1000])
```

Out[143]:

```
<matplotlib.collections.PathCollection at 0x1f5854b6d60>
```



In [144…

```
df.saledate[:1000]
```

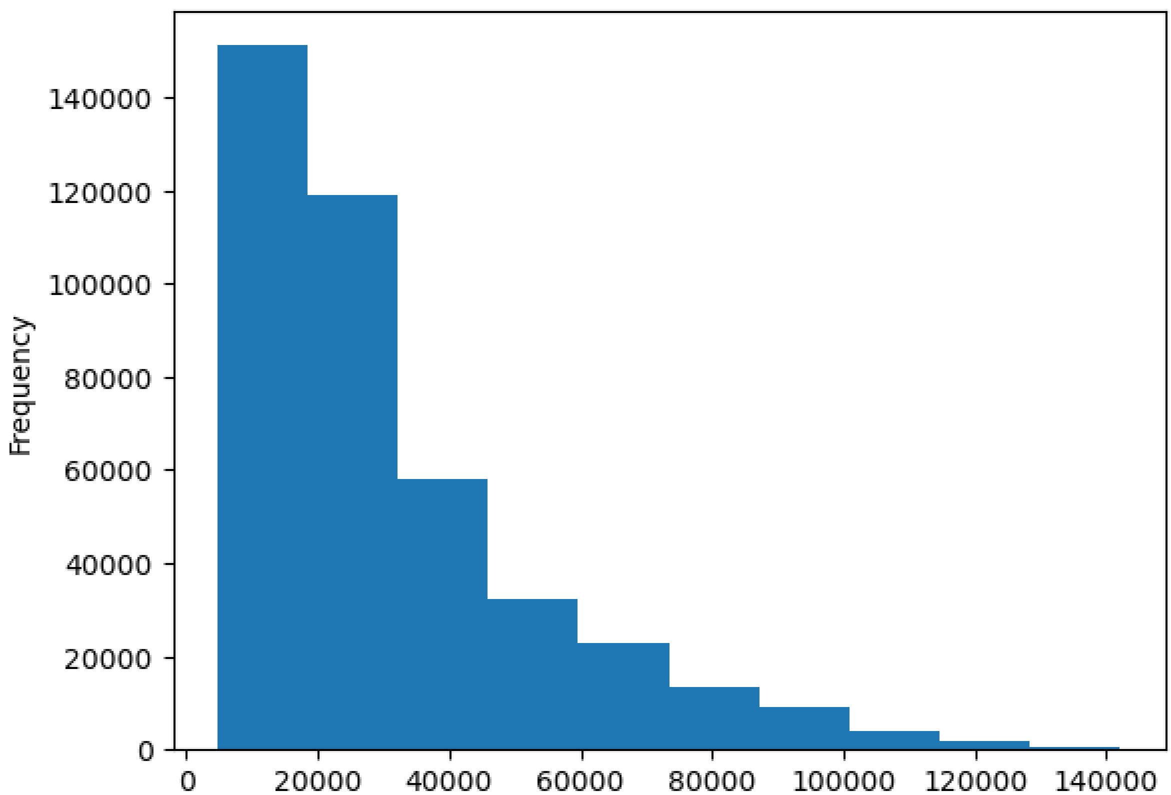
```
Out[144]: 0      11/16/2006 0:00
          1       3/26/2004 0:00
          2       2/26/2004 0:00
          3       5/19/2011 0:00
          4       7/23/2009 0:00
                       ...
          995     7/16/2009 0:00
          996     6/14/2007 0:00
          997     9/22/2005 0:00
          998     7/28/2005 0:00
          999     6/16/2011 0:00
          Name: saledate, Length: 1000, dtype: object
```

```
In [145…  df.saledate.dtype
```

```
Out[145]: dtype('O')
```

```
In [146…  df.SalePrice.plot.hist()
```

```
Out[146]: <AxesSubplot:ylabel='Frequency'>
```



Parsing dates

When we work with time series data, we want to enrich the time & date component as much as possible.

We can do that by telling pandas which of our columns has dates in it using the parse_dates parameter.

```
In [147…  # Import data again but this time parse dates
          df = pd.read_csv(r"C:\Users\sonjo\TrainAndValid.csv",
                           low_memory=False,
                           parse_dates=["saledate"])
```

```
In [148…  df.saledate.dtype
```

```
Out[148]: dtype('<M8[ns]')
```

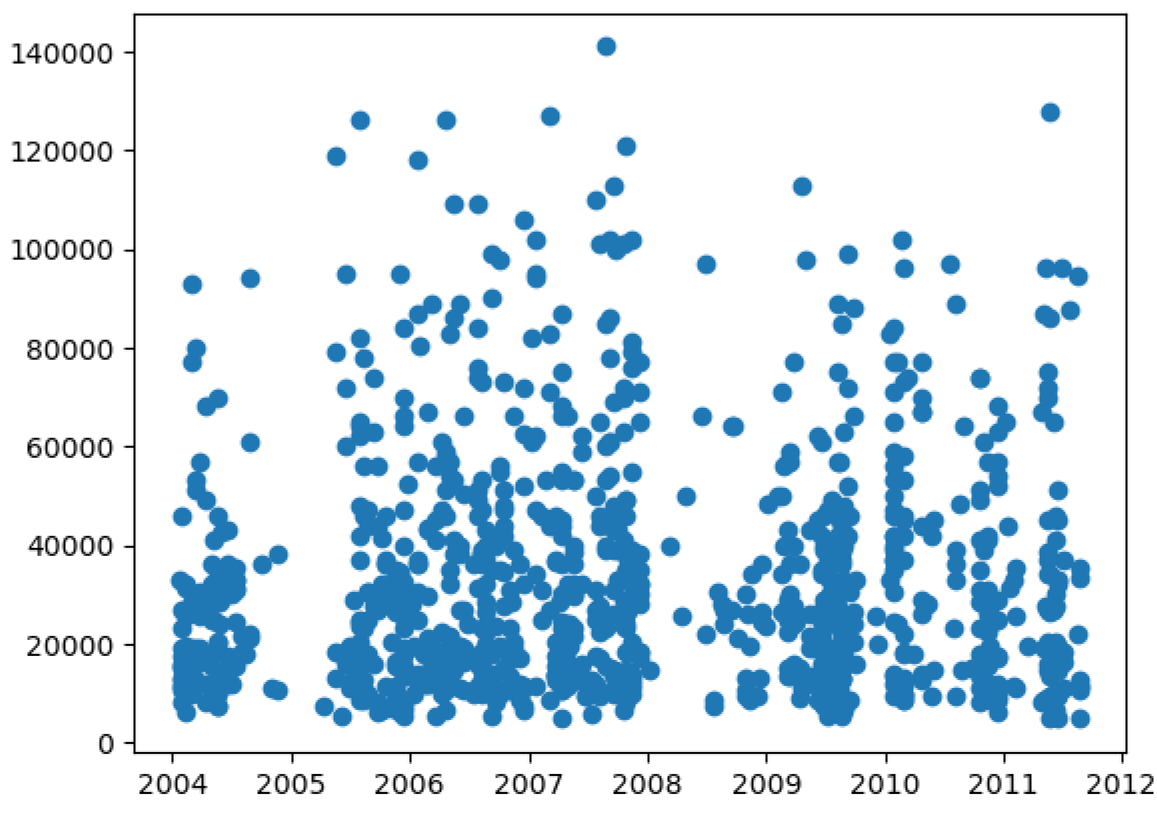
dtype('<M8[ns]') is a NumPy data type object that represents a date or datetime object with nanosecond precision. The < indicates that the byte order is little-endian, and M8[ns] stands for "datetime64[ns]", which means that the data type represents a datetime object with nanosecond precision.

In [149…
```
df.saledate[:1000]
```

Out[149]:
```
0      2006-11-16
1      2004-03-26
2      2004-02-26
3      2011-05-19
4      2009-07-23
          ...
995    2009-07-16
996    2007-06-14
997    2005-09-22
998    2005-07-28
999    2011-06-16
Name: saledate, Length: 1000, dtype: datetime64[ns]
```

In [150…
```
fig, ax = plt.subplots()
ax.scatter(df["saledate"][:1000], df["SalePrice"][:1000])
```

Out[150]:
```
<matplotlib.collections.PathCollection at 0x1f586271340>
```



In [151…
```
df.head()
```

Out[151]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	Us
0	1139246	66000.0	999089	3157	121	3.0	2004	68.0	
1	1139248	57000.0	117657	77	121	3.0	1996	4640.0	
2	1139249	10000.0	434808	7009	121	3.0	2001	2838.0	
3	1139251	38500.0	1026470	332	121	3.0	2001	3486.0	

4	1139253	11000.0	1057373	17311	121	3.0	2007	722.0

5 rows × 53 columns

In [152…:
```
df.saledate.head(20)
```

Out[152]:
```
0     2006-11-16
1     2004-03-26
2     2004-02-26
3     2011-05-19
4     2009-07-23
5     2008-12-18
6     2004-08-26
7     2005-11-17
8     2009-08-27
9     2007-08-09
10    2008-08-21
11    2006-08-24
12    2005-10-20
13    2006-01-26
14    2006-01-03
15    2006-11-16
16    2007-06-14
17    2010-01-28
18    2006-03-09
19    2005-11-17
Name: saledate, dtype: datetime64[ns]
```

In [153…:
```
df.head().T
```

Out[153]:

	0	**1**	**2**	**3**	**4**
SalesID	1139246	1139248	1139249	1139251	1139253
SalePrice	66000.0	57000.0	10000.0	38500.0	11000.0
MachineID	999089	117657	434808	1026470	1057373
ModelID	3157	77	7009	332	17311
datasource	121	121	121	121	121
auctioneerID	3.0	3.0	3.0	3.0	3.0
YearMade	2004	1996	2001	2001	2007
MachineHoursCurrentMeter	68.0	4640.0	2838.0	3486.0	722.0
UsageBand	Low	Low	High	High	Medium
saledate	2006-11-16 00:00:00	2004-03-26 00:00:00	2004-02-26 00:00:00	2011-05-19 00:00:00	2009-07-23 00:00:00
fiModelDesc	521D	950FII	226	PC120-6E	S175
fiBaseModel	521	950	226	PC120	S175
fiSecondaryDesc	D	F	NaN	NaN	NaN
fiModelSeries	NaN	II	NaN	-6E	NaN
fiModelDescriptor	NaN	NaN	NaN	NaN	NaN
ProductSize	NaN	Medium	NaN	Small	NaN
fiProductClassDesc	Wheel Loader -	Wheel Loader -	Skid Steer Loader	Hydraulic	Skid Steer Loader

	110.0 to 120.0 Horsepower	150.0 to 175.0 Horsepower	- 1351.0 to 1601.0 Lb Operat...	Excavator, Track - 12.0 to 14.0 Metr...	- 1601.0 to 1751.0 Lb Operat...
state	Alabama	North Carolina	New York	Texas	New York
ProductGroup	WL	WL	SSL	TEX	SSL
ProductGroupDesc	Wheel Loader	Wheel Loader	Skid Steer Loaders	Track Excavators	Skid Steer Loaders
Drive_System	NaN	NaN	NaN	NaN	NaN
Enclosure	EROPS w AC	EROPS w AC	OROPS	EROPS w AC	EROPS
Forks	None or Unspecified	None or Unspecified	None or Unspecified	NaN	None or Unspecified
Pad_Type	NaN	NaN	NaN	NaN	NaN
Ride_Control	None or Unspecified	None or Unspecified	NaN	NaN	NaN
Stick	NaN	NaN	NaN	NaN	NaN
Transmission	NaN	NaN	NaN	NaN	NaN
Turbocharged	NaN	NaN	NaN	NaN	NaN
Blade_Extension	NaN	NaN	NaN	NaN	NaN
Blade_Width	NaN	NaN	NaN	NaN	NaN
Enclosure_Type	NaN	NaN	NaN	NaN	NaN
Engine_Horsepower	NaN	NaN	NaN	NaN	NaN
Hydraulics	2 Valve	2 Valve	Auxiliary	2 Valve	Auxiliary
Pushblock	NaN	NaN	NaN	NaN	NaN
Ripper	NaN	NaN	NaN	NaN	NaN
Scarifier	NaN	NaN	NaN	NaN	NaN
Tip_Control	NaN	NaN	NaN	NaN	NaN
Tire_Size	None or Unspecified	23.5	NaN	NaN	NaN
Coupler	None or Unspecified	None or Unspecified	None or Unspecified	None or Unspecified	None or Unspecified
Coupler_System	NaN	NaN	None or Unspecified	NaN	None or Unspecified
Grouser_Tracks	NaN	NaN	None or Unspecified	NaN	None or Unspecified
Hydraulics_Flow	NaN	NaN	Standard	NaN	Standard
Track_Type	NaN	NaN	NaN	NaN	NaN
Undercarriage_Pad_Width	NaN	NaN	NaN	NaN	NaN
Stick_Length	NaN	NaN	NaN	NaN	NaN
Thumb	NaN	NaN	NaN	NaN	NaN
Pattern_Changer	NaN	NaN	NaN	NaN	NaN
Grouser_Type	NaN	NaN	NaN	NaN	NaN

Backhoe_Mounting	NaN	NaN	NaN	NaN	NaN
Blade_Type	NaN	NaN	NaN	NaN	NaN
Travel_Controls	NaN	NaN	NaN	NaN	NaN
Differential_Type	Standard	Standard	NaN	NaN	NaN
Steering_Controls	Conventional	Conventional	NaN	NaN	NaN

Sort DataFrame by saledate

When working with time series data, it's a good idea to sort it by date.

In [154…

```
# Sort DataFrame in date order
df.sort_values(by=["saledate"], inplace=True, ascending=True)
df.saledate.head(20)
```

Out[154]:

```
205615   1989-01-17
274835   1989-01-31
141296   1989-01-31
212552   1989-01-31
62755    1989-01-31
54653    1989-01-31
81383    1989-01-31
204924   1989-01-31
135376   1989-01-31
113390   1989-01-31
113394   1989-01-31
116419   1989-01-31
32138    1989-01-31
127610   1989-01-31
76171    1989-01-31
127000   1989-01-31
128130   1989-01-31
127626   1989-01-31
55455    1989-01-31
55454    1989-01-31
Name: saledate, dtype: datetime64[ns]
```

Make a copy of the original DataFrame

We make a copy of the original dataframe so when we manipulate the copy, we've still got our original data.

In [155…

```
# Make a copy of the original DataFrame to perform edits on
df_tmp = df.copy()
```

Add datetime parameters for saledate column

In [156…

```
df_tmp["saleYear"] = df_tmp.saledate.dt.year
df_tmp["saleMonth"] = df_tmp.saledate.dt.month
df_tmp["saleDay"] = df_tmp.saledate.dt.day
df_tmp["saleDayofWeek"] = df_tmp.saledate.dt.dayofweek
df_tmp["saleDayofYear"] = df_tmp.saledate.dt.dayofyear
```

In [157…

```
df_tmp.head().T
```

Out[157]:

	205615	274835	141296	212552	62755

SalesID	1646770	1821514	1505138	1671174	1329056
SalePrice	9500.0	14000.0	50000.0	16000.0	22000.0
MachineID	1126363	1194089	1473654	1327630	1336053
ModelID	8434	10150	4139	8591	4089
datasource	132	132	132	132	132
auctioneerID	18.0	99.0	99.0	99.0	99.0
YearMade	1974	1980	1978	1980	1984
MachineHoursCurrentMeter	NaN	NaN	NaN	NaN	NaN
UsageBand	NaN	NaN	NaN	NaN	NaN
saledate	1989-01-17 00:00:00	1989-01-31 00:00:00	1989-01-31 00:00:00	1989-01-31 00:00:00	1989-01-31 00:00:00
fiModelDesc	TD20	A66	D7G	A62	D3B
fiBaseModel	TD20	A66	D7	A62	D3
fiSecondaryDesc	NaN	NaN	G	NaN	B
fiModelSeries	NaN	NaN	NaN	NaN	NaN
fiModelDescriptor	NaN	NaN	NaN	NaN	NaN
ProductSize	Medium	NaN	Large	NaN	NaN
fiProductClassDesc	Track Type Tractor, Dozer - 105.0 to 130.0 Hor...	Wheel Loader - 120.0 to 135.0 Horsepower	Track Type Tractor, Dozer - 190.0 to 260.0 Hor...	Wheel Loader - Unidentified	Track Type Tractor, Dozer - 20.0 to 75.0 Horse...
state	Texas	Florida	Florida	Florida	Florida
ProductGroup	TTT	WL	TTT	WL	TTT
ProductGroupDesc	Track Type Tractors	Wheel Loader	Track Type Tractors	Wheel Loader	Track Type Tractors
Drive_System	NaN	NaN	NaN	NaN	NaN
Enclosure	OROPS	OROPS	OROPS	EROPS	OROPS
Forks	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Pad_Type	NaN	NaN	NaN	NaN	NaN
Ride_Control	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Stick	NaN	NaN	NaN	NaN	NaN
Transmission	Direct Drive	NaN	Standard	NaN	Standard
Turbocharged	NaN	NaN	NaN	NaN	NaN
Blade_Extension	NaN	NaN	NaN	NaN	NaN
Blade_Width	NaN	NaN	NaN	NaN	NaN
Enclosure_Type	NaN	NaN	NaN	NaN	NaN
Engine_Horsepower	NaN	NaN	NaN	NaN	NaN
Hydraulics	2 Valve	2 Valve	2 Valve	2 Valve	2 Valve
Pushblock	NaN	NaN	NaN	NaN	NaN

Ripper	None or Unspecified	NaN	None or Unspecified	NaN	None or Unspecified
Scarifier	NaN	NaN	NaN	NaN	NaN
Tip_Control	NaN	NaN	NaN	NaN	NaN
Tire_Size	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Coupler	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Coupler_System	NaN	NaN	NaN	NaN	NaN
Grouser_Tracks	NaN	NaN	NaN	NaN	NaN
Hydraulics_Flow	NaN	NaN	NaN	NaN	NaN
Track_Type	NaN	NaN	NaN	NaN	NaN
Undercarriage_Pad_Width	NaN	NaN	NaN	NaN	NaN
Stick_Length	NaN	NaN	NaN	NaN	NaN
Thumb	NaN	NaN	NaN	NaN	NaN
Pattern_Changer	NaN	NaN	NaN	NaN	NaN
Grouser_Type	NaN	NaN	NaN	NaN	NaN
Backhoe_Mounting	None or Unspecified	NaN	None or Unspecified	NaN	None or Unspecified
Blade_Type	Straight	NaN	Straight	NaN	PAT
Travel_Controls	None or Unspecified	NaN	None or Unspecified	NaN	Lever
Differential_Type	NaN	Standard	NaN	Standard	NaN
Steering_Controls	NaN	Conventional	NaN	Conventional	NaN
saleYear	1989	1989	1989	1989	1989
saleMonth	1	1	1	1	1
saleDay	17	31	31	31	31
saleDayofWeek	1	1	1	1	1
saleDayofYear	17	31	31	31	31

In [158…

```
# Now we've enriched our DataFrame with date time features, we can remove 'saledate'
df_tmp.drop("saledate", axis=1, inplace=True)
```

In [159…

```
# Check the values of different columns
df_tmp.state.value_counts()
```

Out[159]:

```
Florida           67320
Texas             53110
California        29761
Washington        16222
Georgia           14633
Maryland          13322
Mississippi       13240
Ohio              12369
Illinois          11540
Colorado          11529
```

```
New Jersey          11156
North Carolina      10636
Tennessee           10298
Alabama             10292
Pennsylvania        10234
South Carolina       9951
Arizona              9364
New York             8639
Connecticut          8276
Minnesota            7885
Missouri             7178
Nevada               6932
Louisiana            6627
Kentucky             5351
Maine                5096
Indiana              4124
Arkansas             3933
New Mexico           3631
Utah                 3046
Unspecified          2801
Wisconsin            2745
New Hampshire        2738
Virginia             2353
Idaho                2025
Oregon               1911
Michigan             1831
Wyoming              1672
Montana              1336
Iowa                 1336
Oklahoma             1326
Nebraska              866
West Virginia         840
Kansas                667
Delaware              510
North Dakota          480
Alaska                430
Massachusetts         347
Vermont               300
South Dakota          244
Hawaii                118
Rhode Island           83
Puerto Rico            42
Washington DC           2
Name: state, dtype: int64
```

5. Modelling

We've explored our dataset a little as well as enriched it with some datetime attributes, now let's try to model.

Why model so early?

We know the evaluation metric we're heading towards. We could spend more time doing exploratory data analysis (EDA), finding more out about the data ourselves but what we'll do instead is use a machine learning model to help us do EDA.

Remember, one of the biggest goals of starting any new machine learning project is reducing the time between experiments.

Following the Scikit-Learn machine learning map, we find a RandomForestRegressor() might be a good candidate.

In [160…

```
# This won't work since we've got missing numbers and categories.
#The n_jobs parameter is set to -1, which means that the model will use all available CP
#This can speed up the training process for large datasets.

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_jobs=-1)
model.fit(df_tmp.drop("SalePrice", axis=1), df_tmp.SalePrice)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1928\1906908024.py in <module>
      6
      7 model = RandomForestRegressor(n_jobs=-1)
----> 8 model.fit(df_tmp.drop("SalePrice", axis=1), df_tmp.SalePrice)

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in fit(self, X, y, sample_weig
ht)
    325         if issparse(y):
    326             raise ValueError("sparse multilabel-indicator for y is not supporte
d.")
--> 327         X, y = self._validate_data(
    328             X, y, multi_output=True, accept_sparse="csc", dtype=DTYPE
    329         )

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, valid
ate_separately, **check_params)
    579                 y = check_array(y, **check_y_params)
    580             else:
--> 581                 X, y = check_X_y(X, y, **check_params)
    582             out = X, y
    583

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_spar
se, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, mult
i_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_s
parse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, e
nsure_min_samples, ensure_min_features, estimator)
    744                     array = array.astype(dtype, casting="unsafe", copy=False)
    745                 else:
--> 746                     array = np.asarray(array, order=order, dtype=dtype)
    747             except ComplexWarning as complex_warning:
    748                 raise ValueError(

~\anaconda3\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   2062
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065
   2066     def __array_wrap__(

ValueError: could not convert string to float: 'Low'
```

In [161…

```
df_tmp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 412698 entries, 205615 to 409203
Data columns (total 57 columns):
```

```
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   SalesID                   412698 non-null  int64
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64
 3   ModelID                   412698 non-null  int64
 4   datasource                412698 non-null  int64
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   object
 9   fiModelDesc               412698 non-null  object
 10  fiBaseModel               412698 non-null  object
 11  fiSecondaryDesc           271971 non-null  object
 12  fiModelSeries             58667 non-null   object
 13  fiModelDescriptor         74816 non-null   object
 14  ProductSize               196093 non-null  object
 15  fiProductClassDesc        412698 non-null  object
 16  state                     412698 non-null  object
 17  ProductGroup              412698 non-null  object
 18  ProductGroupDesc          412698 non-null  object
 19  Drive_System              107087 non-null  object
 20  Enclosure                 412364 non-null  object
 21  Forks                     197715 non-null  object
 22  Pad_Type                  81096 non-null   object
 23  Ride_Control              152728 non-null  object
 24  Stick                     81096 non-null   object
 25  Transmission              188007 non-null  object
 26  Turbocharged              81096 non-null   object
 27  Blade_Extension           25983 non-null   object
 28  Blade_Width               25983 non-null   object
 29  Enclosure_Type            25983 non-null   object
 30  Engine_Horsepower         25983 non-null   object
 31  Hydraulics                330133 non-null  object
 32  Pushblock                 25983 non-null   object
 33  Ripper                    106945 non-null  object
 34  Scarifier                 25994 non-null   object
 35  Tip_Control               25983 non-null   object
 36  Tire_Size                 97638 non-null   object
 37  Coupler                   220679 non-null  object
 38  Coupler_System            44974 non-null   object
 39  Grouser_Tracks            44875 non-null   object
 40  Hydraulics_Flow           44875 non-null   object
 41  Track_Type                102193 non-null  object
 42  Undercarriage_Pad_Width   102916 non-null  object
 43  Stick_Length              102261 non-null  object
 44  Thumb                     102332 non-null  object
 45  Pattern_Changer           102261 non-null  object
 46  Grouser_Type              102193 non-null  object
 47  Backhoe_Mounting          80712 non-null   object
 48  Blade_Type                81875 non-null   object
 49  Travel_Controls           81877 non-null   object
 50  Differential_Type         71564 non-null   object
 51  Steering_Controls         71522 non-null   object
 52  saleYear                  412698 non-null  int64
 53  saleMonth                 412698 non-null  int64
 54  saleDay                   412698 non-null  int64
 55  saleDayofWeek             412698 non-null  int64
 56  saleDayofYear             412698 non-null  int64
dtypes: float64(3), int64(10), object(44)
memory usage: 182.6+ MB
```

In [162…

```
df_tmp.isna().sum()
```

Out[162]:

```
SalesID                          0
SalePrice                        0
```

```
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                 20136
YearMade                         0
MachineHoursCurrentMeter    265194
UsageBand                   339028
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
Grouser_Tracks              367823
Hydraulics_Flow             367823
Track_Type                  310505
Undercarriage_Pad_Width     309782
Stick_Length                310437
Thumb                       310366
Pattern_Changer             310437
Grouser_Type                310505
Backhoe_Mounting            331986
Blade_Type                  330823
Travel_Controls             330821
Differential_Type           341134
Steering_Controls           341176
saleYear                         0
saleMonth                        0
saleDay                          0
saleDayofWeek                    0
saleDayofYear                    0
dtype: int64
```

Convert strings to categories

One way to help turn all of our data into numbers is to convert the columns with the string datatype into a category datatype.

To do this we can use the pandas types API which allows us to interact and manipulate the types of data.

In [163…

```
df_tmp.head().T
```

Out[163]:

	205615	274835	141296	212552	62755
SalesID	1646770	1821514	1505138	1671174	1329056
SalePrice	9500.0	14000.0	50000.0	16000.0	22000.0
MachineID	1126363	1194089	1473654	1327630	1336053
ModelID	8434	10150	4139	8591	4089
datasource	132	132	132	132	132
auctioneerID	18.0	99.0	99.0	99.0	99.0
YearMade	1974	1980	1978	1980	1984
MachineHoursCurrentMeter	NaN	NaN	NaN	NaN	NaN
UsageBand	NaN	NaN	NaN	NaN	NaN
fiModelDesc	TD20	A66	D7G	A62	D3B
fiBaseModel	TD20	A66	D7	A62	D3
fiSecondaryDesc	NaN	NaN	G	NaN	B
fiModelSeries	NaN	NaN	NaN	NaN	NaN
fiModelDescriptor	NaN	NaN	NaN	NaN	NaN
ProductSize	Medium	NaN	Large	NaN	NaN
fiProductClassDesc	Track Type Tractor, Dozer - 105.0 to 130.0 Hor...	Wheel Loader - 120.0 to 135.0 Horsepower	Track Type Tractor, Dozer - 190.0 to 260.0 Hor...	Wheel Loader - Unidentified	Track Type Tractor, Dozer - 20.0 to 75.0 Horse...
state	Texas	Florida	Florida	Florida	Florida
ProductGroup	TTT	WL	TTT	WL	TTT
ProductGroupDesc	Track Type Tractors	Wheel Loader	Track Type Tractors	Wheel Loader	Track Type Tractors
Drive_System	NaN	NaN	NaN	NaN	NaN
Enclosure	OROPS	OROPS	OROPS	EROPS	OROPS
Forks	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Pad_Type	NaN	NaN	NaN	NaN	NaN
Ride_Control	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Stick	NaN	NaN	NaN	NaN	NaN
Transmission	Direct Drive	NaN	Standard	NaN	Standard
Turbocharged	NaN	NaN	NaN	NaN	NaN
Blade_Extension	NaN	NaN	NaN	NaN	NaN
Blade_Width	NaN	NaN	NaN	NaN	NaN
Enclosure_Type	NaN	NaN	NaN	NaN	NaN
Engine_Horsepower	NaN	NaN	NaN	NaN	NaN
Hydraulics	2 Valve	2 Valve	2 Valve	2 Valve	2 Valve

Pushblock	NaN	NaN	NaN	NaN	NaN
Ripper	None or Unspecified	NaN	None or Unspecified	NaN	None or Unspecified
Scarifier	NaN	NaN	NaN	NaN	NaN
Tip_Control	NaN	NaN	NaN	NaN	NaN
Tire_Size	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Coupler	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Coupler_System	NaN	NaN	NaN	NaN	NaN
Grouser_Tracks	NaN	NaN	NaN	NaN	NaN
Hydraulics_Flow	NaN	NaN	NaN	NaN	NaN
Track_Type	NaN	NaN	NaN	NaN	NaN
Undercarriage_Pad_Width	NaN	NaN	NaN	NaN	NaN
Stick_Length	NaN	NaN	NaN	NaN	NaN
Thumb	NaN	NaN	NaN	NaN	NaN
Pattern_Changer	NaN	NaN	NaN	NaN	NaN
Grouser_Type	NaN	NaN	NaN	NaN	NaN
Backhoe_Mounting	None or Unspecified	NaN	None or Unspecified	NaN	None or Unspecified
Blade_Type	Straight	NaN	Straight	NaN	PAT
Travel_Controls	None or Unspecified	NaN	None or Unspecified	NaN	Lever
Differential_Type	NaN	Standard	NaN	Standard	NaN
Steering_Controls	NaN	Conventional	NaN	Conventional	NaN
saleYear	1989	1989	1989	1989	1989
saleMonth	1	1	1	1	1
saleDay	17	31	31	31	31
saleDayofWeek	1	1	1	1	1
saleDayofYear	17	31	31	31	31

In [164…

```
# pd.api.types.is_string_dtype(df_tmp["UsageBand"]) returns a boolean value, True if the
pd.api.types.is_string_dtype(df_tmp["UsageBand"])
```

Out[164]:

```
True
```

In [165…

```
# These columns contain strings
for label, content in df_tmp.items():
    if pd.api.types.is_string_dtype(content):
        print(label)
```

```
UsageBand
fiModelDesc
fiBaseModel
fiSecondaryDesc
fiModelSeries
```

```
fiModelDescriptor
ProductSize
fiProductClassDesc
state
ProductGroup
ProductGroupDesc
Drive_System
Enclosure
Forks
Pad_Type
Ride_Control
Stick
Transmission
Turbocharged
Blade_Extension
Blade_Width
Enclosure_Type
Engine_Horsepower
Hydraulics
Pushblock
Ripper
Scarifier
Tip_Control
Tire_Size
Coupler
Coupler_System
Grouser_Tracks
Hydraulics_Flow
Track_Type
Undercarriage_Pad_Width
Stick_Length
Thumb
Pattern_Changer
Grouser_Type
Backhoe_Mounting
Blade_Type
Travel_Controls
Differential_Type
Steering_Controls
```

In [166…

```python
# If you're wondering what df.items() does, let's use a dictionary as an example
random_dict = {"key1": "hello",
               "key2": "world!"}

for key, value in random_dict.items():
    print(f"This is a key: {key}")
    print(f"This is a value: {value}")
```

```
This is a key: key1
This is a value: hello
This is a key: key2
This is a value: world!
```

In [167…

```python
# This will turn all of the string values into category values
for label, content in df_tmp.items():
    if pd.api.types.is_string_dtype(content):
        df_tmp[label] = content.astype("category").cat.as_ordered()
```

In [168…

```python
df_tmp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 412698 entries, 205615 to 409203
Data columns (total 57 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   SalesID                   412698 non-null  int64
```

```
 1   SalePrice                 412698 non-null  float64
 2   MachineID                 412698 non-null  int64
 3   ModelID                   412698 non-null  int64
 4   datasource                412698 non-null  int64
 5   auctioneerID              392562 non-null  float64
 6   YearMade                  412698 non-null  int64
 7   MachineHoursCurrentMeter  147504 non-null  float64
 8   UsageBand                 73670 non-null   category
 9   fiModelDesc               412698 non-null  category
 10  fiBaseModel               412698 non-null  category
 11  fiSecondaryDesc           271971 non-null  category
 12  fiModelSeries             58667 non-null   category
 13  fiModelDescriptor         74816 non-null   category
 14  ProductSize               196093 non-null  category
 15  fiProductClassDesc        412698 non-null  category
 16  state                     412698 non-null  category
 17  ProductGroup              412698 non-null  category
 18  ProductGroupDesc          412698 non-null  category
 19  Drive_System              107087 non-null  category
 20  Enclosure                 412364 non-null  category
 21  Forks                     197715 non-null  category
 22  Pad_Type                  81096 non-null   category
 23  Ride_Control              152728 non-null  category
 24  Stick                     81096 non-null   category
 25  Transmission              188007 non-null  category
 26  Turbocharged              81096 non-null   category
 27  Blade_Extension           25983 non-null   category
 28  Blade_Width               25983 non-null   category
 29  Enclosure_Type            25983 non-null   category
 30  Engine_Horsepower         25983 non-null   category
 31  Hydraulics                330133 non-null  category
 32  Pushblock                 25983 non-null   category
 33  Ripper                    106945 non-null  category
 34  Scarifier                 25994 non-null   category
 35  Tip_Control               25983 non-null   category
 36  Tire_Size                 97638 non-null   category
 37  Coupler                   220679 non-null  category
 38  Coupler_System            44974 non-null   category
 39  Grouser_Tracks            44875 non-null   category
 40  Hydraulics_Flow           44875 non-null   category
 41  Track_Type                102193 non-null  category
 42  Undercarriage_Pad_Width   102916 non-null  category
 43  Stick_Length              102261 non-null  category
 44  Thumb                     102332 non-null  category
 45  Pattern_Changer           102261 non-null  category
 46  Grouser_Type              102193 non-null  category
 47  Backhoe_Mounting          80712 non-null   category
 48  Blade_Type                81875 non-null   category
 49  Travel_Controls           81877 non-null   category
 50  Differential_Type         71564 non-null   category
 51  Steering_Controls         71522 non-null   category
 52  saleYear                  412698 non-null  int64
 53  saleMonth                 412698 non-null  int64
 54  saleDay                   412698 non-null  int64
 55  saleDayofWeek             412698 non-null  int64
 56  saleDayofYear             412698 non-null  int64
dtypes: category(44), float64(3), int64(10)
memory usage: 63.2 MB
```

In [169…

```
df_tmp.state.cat.categories
```

Out[169]:

```
Index(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',
       'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
       'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana',
       'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada',
```

```
       'New Hampshire', 'New Jersey', 'New Mexico', 'New York',
       'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon',
       'Pennsylvania', 'Puerto Rico', 'Rhode Island', 'South Carolina',
       'South Dakota', 'Tennessee', 'Texas', 'Unspecified', 'Utah', 'Vermont',
       'Virginia', 'Washington', 'Washington DC', 'West Virginia', 'Wisconsin',
       'Wyoming'],
      dtype='object')
```

In [170…

```python
# The cat.codes attribute returns an integer code for each unique category in the catego
df_tmp.state.cat.codes
```

Out[170]:

```
205615    43
274835     8
141296     8
212552     8
62755      8
          ..
410879     4
412476     4
411927     4
407124     4
409203     4
Length: 412698, dtype: int8
```

In [171…

```python
df_tmp.isnull().sum()/len(df_tmp)*100
```

Out[171]:

```
SalesID                      0.000000
SalePrice                    0.000000
MachineID                    0.000000
ModelID                      0.000000
datasource                   0.000000
auctioneerID                 4.879113
YearMade                     0.000000
MachineHoursCurrentMeter    64.258610
UsageBand                   82.149174
fiModelDesc                  0.000000
fiBaseModel                  0.000000
fiSecondaryDesc             34.099269
fiModelSeries               85.784520
fiModelDescriptor           81.871490
ProductSize                 52.485110
fiProductClassDesc           0.000000
state                        0.000000
ProductGroup                 0.000000
ProductGroupDesc             0.000000
Drive_System                74.051970
Enclosure                    0.080931
Forks                       52.092087
Pad_Type                    80.349796
Ride_Control                62.992794
Stick                       80.349796
Transmission                54.444412
Turbocharged                80.349796
Blade_Extension             93.704113
Blade_Width                 93.704113
Enclosure_Type              93.704113
Engine_Horsepower           93.704113
Hydraulics                  20.006155
Pushblock                   93.704113
Ripper                      74.086378
Scarifier                   93.701448
Tip_Control                 93.704113
Tire_Size                   76.341538
Coupler                     46.527727
Coupler_System              89.102443
Grouser_Tracks              89.126431
```

```
Hydraulics_Flow             89.126431
Track_Type                  75.237825
Undercarriage_Pad_Width     75.062637
Stick_Length                75.221348
Thumb                       75.204144
Pattern_Changer             75.221348
Grouser_Type                75.237825
Backhoe_Mounting            80.442842
Blade_Type                  80.161038
Travel_Controls             80.160553
Differential_Type           82.659475
Steering_Controls           82.669652
saleYear                     0.000000
saleMonth                    0.000000
saleDay                      0.000000
saleDayofWeek                0.000000
saleDayofYear                0.000000
dtype: float64
```

Save Processed Data

In [172…

```
# Save preprocessed data
df_tmp.to_csv("train_tmp.csv",
              index=False)
```

In [173…

```
# Import preprocessed data
df_tmp = pd.read_csv("train_tmp.csv",
                     low_memory=False)
df_tmp.head().T
```

Out[173]:

	0	1	2	3	4
SalesID	1646770	1821514	1505138	1671174	1329056
SalePrice	9500.0	14000.0	50000.0	16000.0	22000.0
MachineID	1126363	1194089	1473654	1327630	1336053
ModelID	8434	10150	4139	8591	4089
datasource	132	132	132	132	132
auctioneerID	18.0	99.0	99.0	99.0	99.0
YearMade	1974	1980	1978	1980	1984
MachineHoursCurrentMeter	NaN	NaN	NaN	NaN	NaN
UsageBand	NaN	NaN	NaN	NaN	NaN
fiModelDesc	TD20	A66	D7G	A62	D3B
fiBaseModel	TD20	A66	D7	A62	D3
fiSecondaryDesc	NaN	NaN	G	NaN	B
fiModelSeries	NaN	NaN	NaN	NaN	NaN
fiModelDescriptor	NaN	NaN	NaN	NaN	NaN
ProductSize	Medium	NaN	Large	NaN	NaN
fiProductClassDesc	Track Type Tractor, Dozer - 105.0 to 130.0 Hor...	Wheel Loader - 120.0 to 135.0 Horsepower	Track Type Tractor, Dozer - 190.0 to 260.0 Hor...	Wheel Loader - Unidentified	Track Type Tractor, Dozer - 20.0 to 75.0 Horse...
state	Texas	Florida	Florida	Florida	Florida

ProductGroup	TTT	WL	TTT	WL	TTT
ProductGroupDesc	Track Type Tractors	Wheel Loader	Track Type Tractors	Wheel Loader	Track Type Tractors
Drive_System	NaN	NaN	NaN	NaN	NaN
Enclosure	OROPS	OROPS	OROPS	EROPS	OROPS
Forks	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Pad_Type	NaN	NaN	NaN	NaN	NaN
Ride_Control	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Stick	NaN	NaN	NaN	NaN	NaN
Transmission	Direct Drive	NaN	Standard	NaN	Standard
Turbocharged	NaN	NaN	NaN	NaN	NaN
Blade_Extension	NaN	NaN	NaN	NaN	NaN
Blade_Width	NaN	NaN	NaN	NaN	NaN
Enclosure_Type	NaN	NaN	NaN	NaN	NaN
Engine_Horsepower	NaN	NaN	NaN	NaN	NaN
Hydraulics	2 Valve	2 Valve	2 Valve	2 Valve	2 Valve
Pushblock	NaN	NaN	NaN	NaN	NaN
Ripper	None or Unspecified	NaN	None or Unspecified	NaN	None or Unspecified
Scarifier	NaN	NaN	NaN	NaN	NaN
Tip_Control	NaN	NaN	NaN	NaN	NaN
Tire_Size	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Coupler	NaN	None or Unspecified	NaN	None or Unspecified	NaN
Coupler_System	NaN	NaN	NaN	NaN	NaN
Grouser_Tracks	NaN	NaN	NaN	NaN	NaN
Hydraulics_Flow	NaN	NaN	NaN	NaN	NaN
Track_Type	NaN	NaN	NaN	NaN	NaN
Undercarriage_Pad_Width	NaN	NaN	NaN	NaN	NaN
Stick_Length	NaN	NaN	NaN	NaN	NaN
Thumb	NaN	NaN	NaN	NaN	NaN
Pattern_Changer	NaN	NaN	NaN	NaN	NaN
Grouser_Type	NaN	NaN	NaN	NaN	NaN
Backhoe_Mounting	None or Unspecified	NaN	None or Unspecified	NaN	None or Unspecified
Blade_Type	Straight	NaN	Straight	NaN	PAT
Travel_Controls	None or Unspecified	NaN	None or Unspecified	NaN	Lever

Differential_Type	NaN	Standard	NaN	Standard	NaN
Steering_Controls	NaN	Conventional	NaN	Conventional	NaN
saleYear	1989	1989	1989	1989	1989
saleMonth	1	1	1	1	1
saleDay	17	31	31	31	31
saleDayofWeek	1	1	1	1	1
saleDayofYear	17	31	31	31	31

Excellent, our processed DataFrame has the columns we added to it but it's still missing values.

In [174…

```
# Check missing values
df_tmp.isna().sum()
```

Out[174]:

```
SalesID                          0
SalePrice                        0
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                 20136
YearMade                         0
MachineHoursCurrentMeter    265194
UsageBand                   339028
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
Grouser_Tracks              367823
Hydraulics_Flow             367823
Track_Type                  310505
Undercarriage_Pad_Width     309782
Stick_Length                310437
Thumb                       310366
Pattern_Changer             310437
```

```
Grouser_Type              310505
Backhoe_Mounting          331986
Blade_Type                330823
Travel_Controls           330821
Differential_Type         341134
Steering_Controls         341176
saleYear                       0
saleMonth                      0
saleDay                        0
saleDayofWeek                  0
saleDayofYear                  0
dtype: int64
```

Fill missing values

From our experience with machine learning models. We know two things:

1. All of our data has to be numerical
2. There can't be any missing values

And as we've seen using df_tmp.isna().sum() our data still has plenty of missing values.

Let's fill them.

Filling numerical values first

We're going to fill any column with missing values with the median of that column.

In [175…

```
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
SalesID
SalePrice
MachineID
ModelID
datasource
auctioneerID
YearMade
MachineHoursCurrentMeter
saleYear
saleMonth
saleDay
saleDayofWeek
saleDayofYear
```

In [176…

```
# Check for which numeric columns have null values
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

```
auctioneerID
MachineHoursCurrentMeter
```

In [177…

```
# Fill numeric rows with the median
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            # Add a binary column which tells if the data was missing our not
            df_tmp[label+"_is_missing"] = pd.isnull(content)
```

```
            # Fill missing numeric values with median since it's more robust than the me
            df_tmp[label] = content.fillna(content.median())
```

Why add a binary column indicating whether the data was missing or not?

We can easily fill all of the missing numeric values in our dataset with the median. However, a numeric value may be missing for a reason. In other words, absence of evidence may be evidence of absence. Adding a binary column which indicates whether the value was missing or not helps to retain this information.

In [178…

```
# Check if there's any null values
for label, content in df_tmp.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

In [179…

```
# Check to see how many examples were missing
df_tmp.auctioneerID_is_missing.value_counts()
```

Out[179]:

```
False    392562
True      20136
Name: auctioneerID_is_missing, dtype: int64
```

In [180…

```
df_tmp.isna().sum()
```

Out[180]:

```
SalesID                          0
SalePrice                        0
MachineID                        0
ModelID                          0
datasource                       0
auctioneerID                     0
YearMade                         0
MachineHoursCurrentMeter         0
UsageBand                   339028
fiModelDesc                      0
fiBaseModel                      0
fiSecondaryDesc             140727
fiModelSeries               354031
fiModelDescriptor           337882
ProductSize                 216605
fiProductClassDesc               0
state                            0
ProductGroup                     0
ProductGroupDesc                 0
Drive_System                305611
Enclosure                      334
Forks                       214983
Pad_Type                    331602
Ride_Control                259970
Stick                       331602
Transmission                224691
Turbocharged                331602
Blade_Extension             386715
Blade_Width                 386715
Enclosure_Type              386715
Engine_Horsepower           386715
Hydraulics                   82565
Pushblock                   386715
Ripper                      305753
Scarifier                   386704
Tip_Control                 386715
Tire_Size                   315060
Coupler                     192019
Coupler_System              367724
Grouser_Tracks              367823
```

```
Hydraulics_Flow                          367823
Track_Type                               310505
Undercarriage_Pad_Width                  309782
Stick_Length                             310437
Thumb                                    310366
Pattern_Changer                          310437
Grouser_Type                             310505
Backhoe_Mounting                         331986
Blade_Type                               330823
Travel_Controls                          330821
Differential_Type                        341134
Steering_Controls                        341176
saleYear                                      0
saleMonth                                     0
saleDay                                       0
saleDayofWeek                                 0
saleDayofYear                                 0
auctioneerID_is_missing                       0
MachineHoursCurrentMeter_is_missing           0
dtype: int64
```

Filling and turning categorical variables to numbers

Now we've filled the numeric values, we'll do the same with the categorical values at the same time as turning them into numbers.

In [181…

```python
# Check columns which *aren't* numeric
for label, content in df_tmp.items():
    if not pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
UsageBand
fiModelDesc
fiBaseModel
fiSecondaryDesc
fiModelSeries
fiModelDescriptor
ProductSize
fiProductClassDesc
state
ProductGroup
ProductGroupDesc
Drive_System
Enclosure
Forks
Pad_Type
Ride_Control
Stick
Transmission
Turbocharged
Blade_Extension
Blade_Width
Enclosure_Type
Engine_Horsepower
Hydraulics
Pushblock
Ripper
Scarifier
Tip_Control
Tire_Size
Coupler
Coupler_System
Grouser_Tracks
```

```
Hydraulics_Flow
Track_Type
Undercarriage_Pad_Width
Stick_Length
Thumb
Pattern_Changer
Grouser_Type
Backhoe_Mounting
Blade_Type
Travel_Controls
Differential_Type
Steering_Controls
```

In [182…

```python
# Turn categorical variables into numbers
for label, content in df_tmp.items():
    # Check columns which *aren't* numeric
    if not pd.api.types.is_numeric_dtype(content):
        # Add binary column to inidicate whether sample had missing value
        df_tmp[label+"_is_missing"] = pd.isnull(content)
        # We add the +1 because pandas encodes missing categories as -1
        df_tmp[label] = pd.Categorical(content).codes+1
```

In [183…

```python
df_tmp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412698 entries, 0 to 412697
Columns: 103 entries, SalesID to Steering_Controls_is_missing
dtypes: bool(46), float64(3), int16(4), int64(10), int8(40)
memory usage: 77.9 MB
```

In [184…

```python
df_tmp.isna().sum()
```

Out[184]:

```
SalesID                         0
SalePrice                       0
MachineID                       0
ModelID                         0
datasource                      0
                               ..
Backhoe_Mounting_is_missing     0
Blade_Type_is_missing           0
Travel_Controls_is_missing      0
Differential_Type_is_missing    0
Steering_Controls_is_missing    0
Length: 103, dtype: int64
```

In [185…

```python
df_tmp.head().T
```

Out[185]:

	0	1	2	3	4
SalesID	1646770	1821514	1505138	1671174	1329056
SalePrice	9500.0	14000.0	50000.0	16000.0	22000.0
MachineID	1126363	1194089	1473654	1327630	1336053
ModelID	8434	10150	4139	8591	4089
datasource	132	132	132	132	132
...	...	...	...	...	...
Backhoe_Mounting_is_missing	False	True	False	True	False
Blade_Type_is_missing	False	True	False	True	False
Travel_Controls_is_missing	False	True	False	True	False
Differential_Type_is_missing	True	False	True	False	True

Steering_Controls_is_missing	True	False	True	False	True

103 rows × 5 columns

Now all of our data is numeric and there are no missing values, we should be able to build a machine learning model!

Let's reinstantiate our trusty RandomForestRegressor.

This will take a few minutes which is too long for interacting with it. So what we'll do is create a subset of rows to work with.

Code Description

%%time is a Jupyter notebook magic command that measures the execution time of a code cell.

The rest of the code instantiates a RandomForestRegressor model and fits it to the training data stored in the df_tmp DataFrame.

The RandomForestRegressor is a type of ensemble learning model that uses multiple decision trees to make predictions and then averages the results to obtain a more accurate prediction. The n_jobs parameter is set to -1 to use all available CPU cores for parallel processing, which can significantly speed up the training process.

df_tmp.drop("SalePrice", axis=1) selects all columns in the df_tmp DataFrame except for the SalePrice column, which is the target variable we are trying to predict. This serves as the input features for the model.

df_tmp.SalePrice selects the SalePrice column as the target variable for the model to predict.

In [186…

```
%%time
# Instantiate model
model = RandomForestRegressor(n_jobs=-1)

# Fit the model
model.fit(df_tmp.drop("SalePrice", axis=1), df_tmp.SalePrice)
```

```
Wall time: 7min 1s
```

Out[186]:

```
RandomForestRegressor(n_jobs=-1)
```

In [187…

```
# Score the model
model.score(df_tmp.drop("SalePrice", axis=1), df_tmp.SalePrice)
```

Out[187]:

```
0.9875605451507269
```

Question: Why is this metric not reliable?

Splitting data into train/valid sets

In [188…

```
df_tmp.head()
```

Out[188]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	Us
0	1646770	9500.0	1126363	8434	132	18.0	1974	0.0	
1	1821514	14000.0	1194089	10150	132	99.0	1980	0.0	

2	1505138	50000.0	1473654	4139	132	99.0	1978	0.0
3	1671174	16000.0	1327630	8591	132	99.0	1980	0.0
4	1329056	22000.0	1336053	4089	132	99.0	1984	0.0

5 rows × 103 columns

According to the Kaggle data page, the validation set and test set are split according to dates.

This makes sense since we're working on a time series problem.

E.g. using past events to try and predict future events.

Knowing this, randomly splitting our data into train and test sets using something like train_test_split() wouldn't work.

Instead, we split our data into training, validation and test sets using the date each sample occured.

In our case:

- Training = all samples up until 2011
- Valid = all samples form January 1, 2012 - April 30, 2012
- Test = all samples from May 1, 2012 - November 2012

For more on making good training, validation and test sets, check out the post How (and why) to create a good validation set by Rachel Thomas.

In [189…

```
df_tmp.saleYear.value_counts()
```

Out[189]:

```
2009    43849
2008    39767
2011    35197
2010    33390
2007    32208
2006    21685
2005    20463
2004    19879
2001    17594
2000    17415
2002    17246
2003    15254
1998    13046
1999    12793
2012    11573
1997     9785
1996     8829
1995     8530
1994     7929
1993     6303
1992     5519
1991     5109
1989     4806
1990     4529
Name: saleYear, dtype: int64
```

In [190…

```
# Split data into training and validation
df_val = df_tmp[df_tmp.saleYear == 2012]
df_train = df_tmp[df_tmp.saleYear != 2012]

len(df_val), len(df_train)
```

Out[190]: (11573, 401125)

In [191…
```
# Split data into X & y
X_train, y_train = df_train.drop("SalePrice", axis=1), df_train.SalePrice
X_valid, y_valid = df_val.drop("SalePrice", axis=1), df_val.SalePrice

X_train.shape, y_train.shape, X_valid.shape, y_valid.shape
```

Out[191]: ((401125, 102), (401125,), (11573, 102), (11573,))

Building an evaluation function According to Kaggle for the Bluebook for Bulldozers competition, the evaluation function they use is root mean squared log error (RMSLE).

RMSLE = generally you don't care as much if you're off by $10 as much as you'd care if you were off by 10%, you care more about ratios rather than differences. MAE (mean absolute error) is more about exact differences.

It's important to understand the evaluation metric you're going for.

Since Scikit-Learn doesn't have a function built-in for RMSLE, we'll create our own.

We can do this by taking the square root of Scikit-Learn's mean_squared_log_error (MSLE). MSLE is the same as taking the log of mean squared error (MSE).

We'll also calculate the MAE and R^2 for fun.

In [192…
```
# Create evaluation function (the competition uses Root Mean Square Log Error)
from sklearn.metrics import mean_squared_log_error, mean_absolute_error

def rmsle(y_test, y_preds):
    return np.sqrt(mean_squared_log_error(y_test, y_preds))

# Create function to evaluate our model
def show_scores(model):
    train_preds = model.predict(X_train)
    val_preds = model.predict(X_valid)
    scores = {"Training MAE": mean_absolute_error(y_train, train_preds),
              "Valid MAE": mean_absolute_error(y_valid, val_preds),
              "Training RMSLE": rmsle(y_train, train_preds),
              "Valid RMSLE": rmsle(y_valid, val_preds),
              "Training R^2": model.score(X_train, y_train),
              "Valid R^2": model.score(X_valid, y_valid)}
    return scores
```

Testing our model on a subset (to tune the hyperparameters)

Retraing an entire model would take far too long to continuing experimenting as fast as we want to.

So what we'll do is take a sample of the training set and tune the hyperparameters on that before training a larger model.

If you're experiments are taking longer than 10-seconds (give or take how long you have to wait), you should be trying to speed things up. You can speed things up by sampling less data or using a faster computer.

In [58]:
```
# This takes too long...
```

```
# %%time
# # Retrain a model on training data
# model.fit(X_train, y_train)
# show_scores(model)
```

In [193…

```
len(X_train)
```

Out[193]:

```
401125
```

Depending on your computer (mine is a MacBook Pro), making calculations on ~400,000 rows may take a while...

Let's alter the number of samples each n_estimator in the RandomForestRegressor see's using the max_samples parameter...

In [194…

```
# Change max samples in RandomForestRegressor
model = RandomForestRegressor(n_jobs=-1,
                              max_samples=10000)
```

Setting max_samples to 10000 means every n_estimator (default 100) in our RandomForestRegressor will only see 10000 random samples from our DataFrame instead of the entire 400,000.

In other words, we'll be looking at 40x less samples which means we'll get faster computation speeds but we should expect our results to worsen (simple the model has less samples to learn patterns from).ipynb_checkpoints/

In [195…

```
%%time
# Cutting down the max number of samples each tree can see improves training time
model.fit(X_train, y_train)
```

```
Wall time: 16.8 s
```

Out[195]:

```
RandomForestRegressor(max_samples=10000, n_jobs=-1)
```

In [196…

```
show_scores(model)
```

Out[196]:

```
{'Training MAE': 5552.526340866313,
 'Valid MAE': 7135.9560839885935,
 'Training RMSLE': 0.2572767692418251,
 'Valid RMSLE': 0.2946830042117064,
 'Training R^2': 0.8609013949328514,
 'Valid R^2': 0.8342288053647234}
```

Beautiful, that took far less time than the model with all the data.

With this, let's try tune some hyperparameters.

Hyperparameter tuning with RandomizedSearchCV

You can increase n_iter to try more combinations of hyperparameters but in our case, we'll try 20 and see where it gets us.

Remember, we're trying to reduce the amount of time it takes between experiments.

In [198…

```
%%time
from sklearn.model_selection import RandomizedSearchCV

# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 100, 10),
```

```
            "max_depth": [None, 3, 5, 10],
            "min_samples_split": np.arange(2, 20, 2),
            "min_samples_leaf": np.arange(1, 20, 2),
            "max_features": [0.5, 1, "sqrt", "auto"],
            "max_samples": [10000]}

rs_model = RandomizedSearchCV(RandomForestRegressor(),
                              param_distributions=rf_grid,
                              n_iter=20,
                              cv=5,
                              verbose=True)

rs_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Wall time: 14min 14s
```

Out[198]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=20,
                   param_distributions={'max_depth': [None, 3, 5, 10],
                                        'max_features': [0.5, 1, 'sqrt',
                                                         'auto'],
                                        'max_samples': [10000],
                                        'min_samples_leaf': array([ 1,  3,  5,  7,  9, 1
1, 13, 15, 17, 19]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10,
12, 14, 16, 18]),
                                        'n_estimators': array([10, 20, 30, 40, 50, 60, 7
0, 80, 90])},
                   verbose=True)
```

In [199…

```
# Find the best parameters from the RandomizedSearch
rs_model.best_params_
```

Out[199]:

```
{'n_estimators': 70,
 'min_samples_split': 18,
 'min_samples_leaf': 3,
 'max_samples': 10000,
 'max_features': 0.5,
 'max_depth': None}
```

In [200…

```
# Evaluate the RandomizedSearch model
show_scores(rs_model)
```

Out[200]:

```
{'Training MAE': 6116.451338163107,
 'Valid MAE': 7427.91364357921,
 'Training RMSLE': 0.2776696647991228,
 'Valid RMSLE': 0.302743845714793346,
 'Training R^2': 0.8338763447445233,
 'Valid R^2': 0.8212781169149082}
```

Train a model with the best parameters

In a model I prepared earlier, I tried 100 different combinations of hyperparameters (setting n_iter to 100 in RandomizedSearchCV) and found the best results came from the ones you see below.

Note: This kind of search on my computer (n_iter = 100) took ~2-hours. So it's kind of a set and come back later experiment.

We'll instantiate a new model with these discovered hyperparameters and reset the max_samples back to its original value.

In [201…

```
%%time
# Most ideal hyperparameters
ideal_model = RandomForestRegressor(n_estimators=90,
```

```
                                      min_samples_leaf=1,
                                      min_samples_split=14,
                                      max_features=0.5,
                                      n_jobs=-1,
                                      max_samples=None)
ideal_model.fit(X_train, y_train)
```

```
Wall time: 2min 42s
```

Out[201]:

```
RandomForestRegressor(max_features=0.5, min_samples_split=14, n_estimators=90,
                      n_jobs=-1)
```

In [202…

```
show_scores(ideal_model)
```

Out[202]:

```
{'Training MAE': 2925.9264947540223,
 'Valid MAE': 5940.501347177965,
 'Training RMSLE': 0.1433531196332131,
 'Valid RMSLE': 0.24544398682210017,
 'Training R^2': 0.9597276615058973,
 'Valid R^2': 0.8826529856184702}
```

With these new hyperparameters as well as using all the samples, we can see an improvement to our models performance.

You can make a faster model by altering some of the hyperparameters. Particularly by lowering n_estimators since each increase in n_estimators is basically building another small model.

However, lowering of n_estimators or altering of other hyperparameters may lead to poorer results.

In [203…

```
%%time
# Faster model
fast_model = RandomForestRegressor(n_estimators=40,
                                   min_samples_leaf=3,
                                   max_features=0.5,
                                   n_jobs=-1)
fast_model.fit(X_train, y_train)
```

```
Wall time: 1min 20s
```

Out[203]:

```
RandomForestRegressor(max_features=0.5, min_samples_leaf=3, n_estimators=40,
                      n_jobs=-1)
```

In [204…

```
show_scores(fast_model)
```

Out[204]:

```
{'Training MAE': 2547.850661221548,
 'Valid MAE': 5919.78833325197,
 'Training RMSLE': 0.1295503907957013,
 'Valid RMSLE': 0.24415185334487172,
 'Training R^2': 0.9671263907938321,
 'Valid R^2': 0.8812113414300267}
```

Make predictions on test data

Now we've got a trained model, it's time to make predictions on the test data.

Remember what we've done.

Our model is trained on data prior to 2011. However, the test data is from May 1 2012 to November 2012.

So what we're doing is trying to use the patterns our model has learned in the training data to predict the sale price of a Bulldozer with characteristics it's never seen before but are assumed to be similar to that of those in the training data.

In [206…

```
df_test  = pd.read_csv(r"C:/Users/sonjo/Test.csv",
```

```
                    parse_dates=["saledate"])
df_test .head()
```

Out[206]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand
0	1227829	1006309	3168	121	3	1999	3688.0	Low
1	1227844	1022817	7271	121	3	1000	28555.0	High
2	1227847	1031560	22805	121	3	2004	6038.0	Medium
3	1227848	56204	1269	121	3	2006	8940.0	High
4	1227863	1053887	22312	121	3	2005	2286.0	Low

5 rows × 52 columns

In [207…

```
# Let's see how the model goes predicting on the test data
model.predict(df_test)
```

```
C:\Users\sonjo\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The featu
re names should match those that were passed during fit. Starting version 1.2, an error
will be raised.
Feature names unseen at fit time:
- saledate
Feature names seen at fit time, yet now missing:
- Backhoe_Mounting_is_missing
- Blade_Extension_is_missing
- Blade_Type_is_missing
- Blade_Width_is_missing
- Coupler_System_is_missing
- ...

  warnings.warn(message, FutureWarning)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1928\1593019188.py in <module>
      1 # Let's see how the model goes predicting on the test data
----> 2 model.predict(df_test)

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    969         check_is_fitted(self)
    970         # Check data
--> 971         X = self._validate_X_predict(X)
    972 
    973         # Assign chunk of trees to jobs

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self,
 X)
    577         Validate X whenever one tries to predict, apply, predict_proba."""
    578         check_is_fitted(self)
--> 579         X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False
)
    580         if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.i
ntc):
    581             raise ValueError("No support for np.int64 index based sparse matrice
s")

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, valid
ate_separately, **check_params)
```

```
    564                 raise ValueError("Validation should be done on X, y or both.")
    565         elif not no_val_X and no_val_y:
--> 566             X = check_array(X, **check_params)
    567             out = X
    568         elif no_val_X and not no_val_y:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_s
parse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, e
nsure_min_samples, ensure_min_features, estimator)
    744                     array = array.astype(dtype, casting="unsafe", copy=False)
    745                 else:
--> 746                     array = np.asarray(array, order=order, dtype=dtype)
    747             except ComplexWarning as complex_warning:
    748                 raise ValueError(

~\anaconda3\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   2062
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065
   2066     def __array_wrap__(

ValueError: could not convert string to float: 'Low'
```

Ahhh... the test data isn't in the same format of our other data, so we have to fix it. Let's create a function to preprocess our data.

Preprocessing the data

Our model has been trained on data formatted in the same way as the training data.

This means in order to make predictions on the test data, we need to take the same steps we used to preprocess the training data to preprocess the test data.

Remember: Whatever you do to the training data, you have to do to the test data.

Let's create a function for doing so (by copying the preprocessing steps we used above).

In [208…

```python
def preprocess_data(df):
    # Add datetime parameters for saledate
    df["saleYear"] = df.saledate.dt.year
    df["saleMonth"] = df.saledate.dt.month
    df["saleDay"] = df.saledate.dt.day
    df["saleDayofweek"] = df.saledate.dt.dayofweek
    df["saleDayofyear"] = df.saledate.dt.dayofyear

    # Drop original saledate
    df.drop("saledate", axis=1, inplace=True)

    # Fill numeric rows with the median
    for label, content in df.items():
        if pd.api.types.is_numeric_dtype(content):
            if pd.isnull(content).sum():
                df[label+"_is_missing"] = pd.isnull(content)
                df[label] = content.fillna(content.median())

        # Turn categorical variables into numbers
        if not pd.api.types.is_numeric_dtype(content):
            df[label+"_is_missing"] = pd.isnull(content)
            # We add the +1 because pandas encodes missing categories as -1
            df[label] = pd.Categorical(content).codes+1
```

```python
    return df
```

Question: Where would this function break?

Hint: What if the test data had different missing values to the training data?

Now we've got a function for preprocessing data, let's preprocess the test dataset into the same format as our training dataset

In [209…

```python
df_test = preprocess_data(df_test)
df_test.head()
```

Out[209]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand
0	1227829	1006309	3168	121	3	1999	3688.0	2
1	1227844	1022817	7271	121	3	1000	28555.0	1
2	1227847	1031560	22805	121	3	2004	6038.0	3
3	1227848	56204	1269	121	3	2006	8940.0	1
4	1227863	1053887	22312	121	3	2005	2286.0	2

5 rows × 101 columns

In [210…

```python
X_train.head()
```

Out[210]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand
0	1646770	1126363	8434	132	18.0	1974	0.0	0
1	1821514	1194089	10150	132	99.0	1980	0.0	0
2	1505138	1473654	4139	132	99.0	1978	0.0	0
3	1671174	1327630	8591	132	99.0	1980	0.0	0
4	1329056	1336053	4089	132	99.0	1984	0.0	0

5 rows × 102 columns

In [211…

```python
# Make predictions on the test dataset using the best model
test_preds = ideal_model.predict(df_test)
```

```
C:\Users\sonjo\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The featu
re names should match those that were passed during fit. Starting version 1.2, an error
will be raised.
Feature names unseen at fit time:
- saleDayofweek
- saleDayofyear
Feature names seen at fit time, yet now missing:
- auctioneerID_is_missing
- saleDayofWeek
- saleDayofYear

  warnings.warn(message, FutureWarning)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1928\2502794175.py in <module>
      1 # Make predictions on the test dataset using the best model
----> 2 test_preds = ideal_model.predict(df_test)
```

```
~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    969         check_is_fitted(self)
    970         # Check data
--> 971         X = self._validate_X_predict(X)
    972 
    973         # Assign chunk of trees to jobs

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self,
 X)
    577         Validate X whenever one tries to predict, apply, predict_proba."""
    578         check_is_fitted(self)
--> 579         X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False
)
    580         if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.i
ntc):
    581             raise ValueError("No support for np.int64 index based sparse matrice
s")

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, valid
ate_separately, **check_params)
    583 
    584         if not no_val_X and check_params.get("ensure_2d", True):
--> 585             self._check_n_features(X, reset=reset)
    586 
    587         return out

~\anaconda3\lib\site-packages\sklearn\base.py in _check_n_features(self, X, reset)
    398 
    399         if n_features != self.n_features_in_:
--> 400             raise ValueError(
    401                 f"X has {n_features} features, but {self.__class__.__name__} "
    402                 f"is expecting {self.n_features_in_} features as input."

ValueError: X has 101 features, but RandomForestRegressor is expecting 102 features as i
nput.
```

We've found an error and it's because our test dataset (after preprocessing) has 101 columns where as, our training dataset (X_train) has 102 columns (after preprocessing).

Let's find the difference.

In [212…

```
# We can find how the columns differ using sets
set(X_train.columns) - set(df_test.columns)
```

Out[212]:

```
{'auctioneerID_is_missing', 'saleDayofWeek', 'saleDayofYear'}
```

In [213…

```
# Match test dataset columns to training dataset
df_test["auctioneerID_is_missing"] = False
df_test.head()
```

Out[213]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand
0	1227829	1006309	3168	121	3	1999	3688.0	2
1	1227844	1022817	7271	121	3	1000	28555.0	1
2	1227847	1031560	22805	121	3	2004	6038.0	3
3	1227848	56204	1269	121	3	2006	8940.0	1
4	1227863	1053887	22312	121	3	2005	2286.0	2

5 rows × 102 columns

In [214…

```
# Make predictions on the test dataset using the best model
test_preds = ideal_model.predict(df_test)
```

```
C:\Users\sonjo\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The featu
re names should match those that were passed during fit. Starting version 1.2, an error
will be raised.
Feature names unseen at fit time:
- saleDayofweek
- saleDayofyear
Feature names seen at fit time, yet now missing:
- saleDayofWeek
- saleDayofYear

  warnings.warn(message, FutureWarning)
```

When looking at the Kaggle submission requirements, we see that if we wanted to make a submission, the data is required to be in a certain format. Namely, a DataFrame containing the SalesID and the predicted SalePrice of the bulldozer.

Let's make it.

In [215…

```
# Create DataFrame compatible with Kaggle submission requirements
df_preds = pd.DataFrame()
df_preds["SalesID"] = df_test["SalesID"]
df_preds["SalePrice"] = test_preds
df_preds
```

Out[215]:

	SalesID	SalePrice
0	1227829	20474.427188
1	1227844	21409.272038
2	1227847	49314.376689
3	1227848	66308.216505
4	1227863	48969.189905
...	...	...
12452	6643171	51239.373931
12453	6643173	15170.601740
12454	6643184	14772.719480
12455	6643186	20342.673047
12456	6643196	31011.109441

12457 rows × 2 columns

In [217…

```
# Export to csv...
df_preds.to_csv("predictions.csv",
                index=False)
```

Feature Importance

Since we've built a model which is able to make predictions. The people you share these predictions with (or yourself) might be curious of what parts of the data led to these predictions.

This is where feature importance comes in. Feature importance seeks to figure out which different attributes of the data were most important when it comes to predicting the target variable.

In our case, after our model learned the patterns in the data, which bulldozer sale attributes were most important for predicting its overall sale price?

Beware: the default feature importances for random forests can lead to non-ideal results.

To find which features were most important of a machine learning model, a good idea is to search something like "[MODEL NAME] feature importance".

Doing this for our RandomForestRegressor leads us to find the feature*importances* attribute.

Let's check it out.

In [218…
```
# Find feature importance of our best model
ideal_model.feature_importances_
```

Out[218]:
```
array([3.33059583e-02, 1.86012486e-02, 4.26604746e-02, 1.72410831e-03,
       3.32742039e-03, 2.09496441e-01, 3.14870478e-03, 1.04894605e-03,
       4.43854639e-02, 5.03592327e-02, 6.01954248e-02, 4.73869658e-03,
       1.66661703e-02, 1.63645838e-01, 4.02686692e-02, 5.92921304e-03,
       2.42252033e-03, 2.04658077e-03, 4.27299917e-03, 5.40149080e-02,
       7.30273504e-04, 4.62761766e-04, 1.81169916e-03, 1.89333363e-04,
       1.28194032e-03, 2.19796959e-05, 2.25865210e-04, 8.49347651e-03,
       1.22099604e-03, 3.43569763e-04, 4.94504826e-03, 4.77199439e-03,
       3.59409297e-03, 2.95569161e-03, 2.02838327e-03, 1.04187705e-02,
       1.07038527e-03, 1.10485627e-02, 7.37763846e-04, 3.05109954e-03,
       9.30255297e-04, 9.97927442e-04, 1.66790171e-03, 5.89510499e-04,
       4.78723010e-04, 3.54633838e-04, 2.82284597e-04, 1.92681035e-03,
       1.05894747e-03, 2.38661828e-04, 3.22805079e-04, 7.31226889e-02,
       3.83190457e-03, 5.66883005e-03, 2.87567136e-03, 9.86715478e-03,
       2.54512255e-04, 1.52511031e-03, 3.19901714e-04, 0.00000000e+00,
       0.00000000e+00, 2.67380717e-03, 1.46011421e-03, 6.78133718e-03,
       2.75845500e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.80990886e-04, 4.85278688e-06, 1.16546916e-03,
       9.89520187e-06, 1.27908417e-04, 4.84521769e-05, 2.70909559e-04,
       6.65082386e-06, 3.77775290e-04, 2.88629303e-05, 8.06433137e-05,
       3.88405199e-04, 1.98587846e-03, 3.55413572e-03, 8.22130468e-04,
       6.71625973e-04, 1.99483534e-03, 2.50375066e-03, 2.20848095e-04,
       1.46857603e-02, 1.89013889e-03, 1.08873332e-03, 7.89842320e-05,
       1.70511862e-04, 4.43022395e-05, 7.41197385e-05, 7.26812915e-05,
       5.52668767e-05, 4.27567007e-04, 1.18077442e-04, 1.40524306e-04,
       7.48145052e-05, 1.57748220e-04])
```

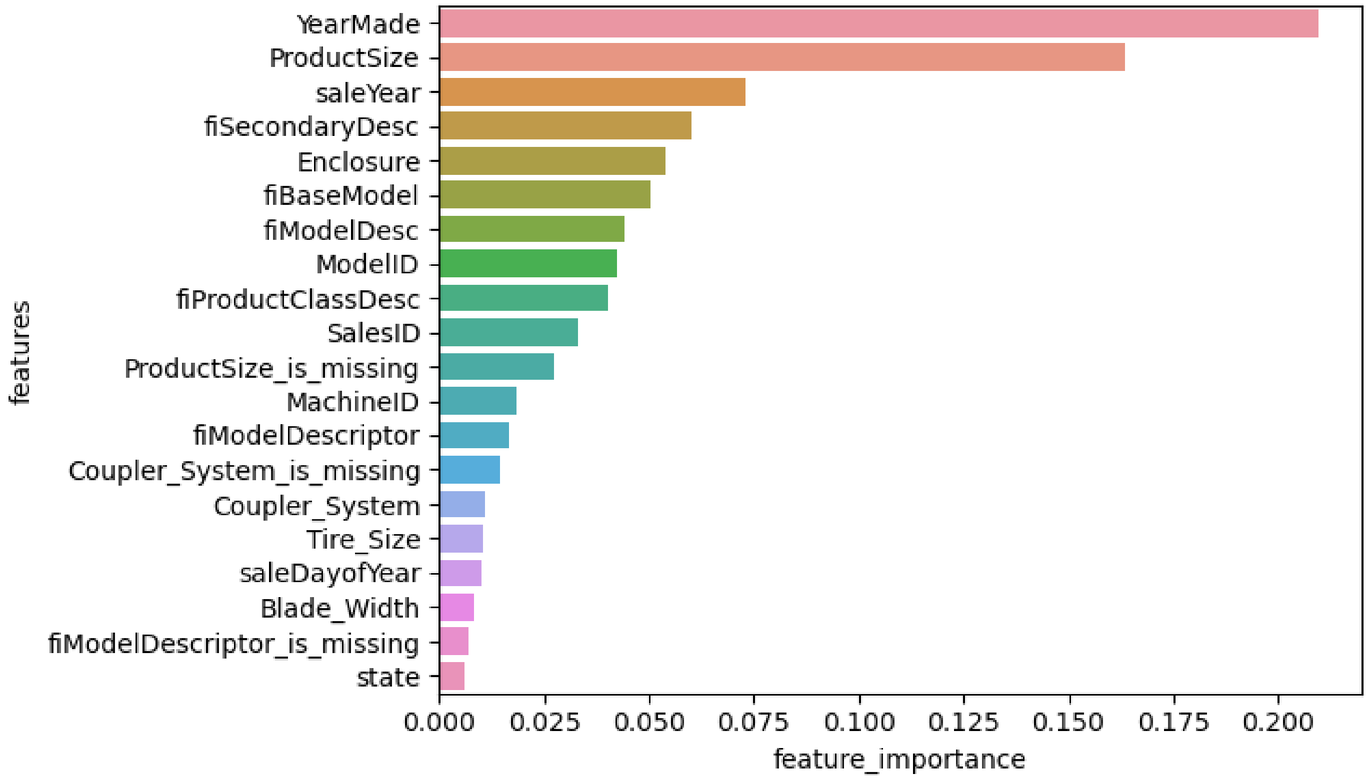
In [219…
```
# Install Seaborn package in current environment (if you don't have it)
# import sys
# !conda install --yes --prefix {sys.prefix} seaborn
```

In [221…
```
import seaborn as sns

# Helper function for plotting feature importance
def plot_features(columns, importances, n=20):
    df = (pd.DataFrame({"features": columns,
                        "feature_importance": importances})
          .sort_values("feature_importance", ascending=False)
          .reset_index(drop=True))

    sns.barplot(x="feature_importance",
                y="features",
                data=df[:n],
                orient="h")
```

In [222…
```
plot_features(X_train.columns, ideal_model.feature_importances_)
```



In [223…
```
sum(ideal_model.feature_importances_)
```

Out[223]:
```
0.9999999999999997
```

In [224…
```
df.ProductSize.isna().sum()
```

Out[224]:
```
216605
```

In [225…
```
df.ProductSize.value_counts()
```

Out[225]:
```
Medium            64342
Large / Medium    51297
Small             27057
Mini              25721
Large             21396
Compact            6280
Name: ProductSize, dtype: int64
```

In [226…
```
df.Turbocharged.value_counts()
```

Out[226]:
```
None or Unspecified    77111
Yes                     3985
Name: Turbocharged, dtype: int64
```

In [227…
```
df.Thumb.value_counts()
```

Out[227]:
```
None or Unspecified    85074
Manual                  9678
Hydraulic               7580
Name: Thumb, dtype: int64
```

In [228…
```
pip install nbconvert[webpdf]
```

```
Requirement already satisfied: nbconvert[webpdf] in c:\users\sonjo\anaconda3\lib\site-pa
ckages (6.4.4)Note: you may need to restart the kernel to use updated packages.
```

```
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\sonjo\anaconda3\lib\site
-packages (from nbconvert[webpdf]) (1.5.0)
Requirement already satisfied: jupyterlab-pygments in c:\users\sonjo\anaconda3\lib\site-
packages (from nbconvert[webpdf]) (0.1.2)
Requirement already satisfied: testpath in c:\users\sonjo\anaconda3\lib\site-packages (f
rom nbconvert[webpdf]) (0.6.0)
Requirement already satisfied: traitlets>=5.0 in c:\users\sonjo\anaconda3\lib\site-packa
ges (from nbconvert[webpdf]) (5.1.1)
Requirement already satisfied: entrypoints>=0.2.2 in c:\users\sonjo\anaconda3\lib\site-p
ackages (from nbconvert[webpdf]) (0.4)
Requirement already satisfied: bleach in c:\users\sonjo\anaconda3\lib\site-packages (fro
m nbconvert[webpdf]) (4.1.0)
Requirement already satisfied: defusedxml in c:\users\sonjo\anaconda3\lib\site-packages
(from nbconvert[webpdf]) (0.7.1)
Requirement already satisfied: pygments>=2.4.1 in c:\users\sonjo\anaconda3\lib\site-pack
ages (from nbconvert[webpdf]) (2.14.0)
Requirement already satisfied: mistune<2,>=0.8.1 in c:\users\sonjo\anaconda3\lib\site-pa
ckages (from nbconvert[webpdf]) (0.8.4)
Requirement already satisfied: beautifulsoup4 in c:\users\sonjo\anaconda3\lib\site-packa
ges (from nbconvert[webpdf]) (4.11.1)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in c:\users\sonjo\anaconda3\lib\si
te-packages (from nbconvert[webpdf]) (0.5.13)
Requirement already satisfied: jinja2>=2.4 in c:\users\sonjo\anaconda3\lib\site-packages
(from nbconvert[webpdf]) (2.11.3)
Requirement already satisfied: nbformat>=4.4 in c:\users\sonjo\anaconda3\lib\site-packag
es (from nbconvert[webpdf]) (5.5.0)
Requirement already satisfied: jupyter-core in c:\users\sonjo\anaconda3\lib\site-package
s (from nbconvert[webpdf]) (4.11.1)
Collecting pyppeteer<1.1,>=1
  Using cached pyppeteer-1.0.2-py3-none-any.whl (83 kB)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\sonjo\anaconda3\lib\site-pac
kages (from jinja2>=2.4->nbconvert[webpdf]) (2.0.1)
Requirement already satisfied: jupyter-client>=6.1.5 in c:\users\sonjo\anaconda3\lib\sit
e-packages (from nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (7.3.4)
Requirement already satisfied: nest-asyncio in c:\users\sonjo\anaconda3\lib\site-package
s (from nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (1.5.5)
Requirement already satisfied: jsonschema>=2.6 in c:\users\sonjo\anaconda3\lib\site-pack
ages (from nbformat>=4.4->nbconvert[webpdf]) (4.16.0)
Requirement already satisfied: fastjsonschema in c:\users\sonjo\anaconda3\lib\site-packa
ges (from nbformat>=4.4->nbconvert[webpdf]) (2.16.2)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in c:\users\sonjo\anaconda3\lib\site-
packages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (4.64.1)
Requirement already satisfied: importlib-metadata>=1.4 in c:\users\sonjo\anaconda3\lib\s
ite-packages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (4.11.3)
Requirement already satisfied: certifi>=2021 in c:\users\sonjo\anaconda3\lib\site-packag
es (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (2022.9.14)
Collecting pyee<9.0.0,>=8.1.0
  Using cached pyee-8.2.2-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in c:\users\sonjo\anaconda3\lib\si
te-packages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (1.26.11)
Collecting websockets<11.0,>=10.0
  Downloading websockets-10.4-cp39-cp39-win_amd64.whl (101 kB)
     -------------------------------------- 101.4/101.4 kB 1.9 MB/s eta 0:00:00
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in c:\users\sonjo\anaconda3\lib\sit
e-packages (from pyppeteer<1.1,>=1->nbconvert[webpdf]) (1.4.4)
Requirement already satisfied: soupsieve>1.2 in c:\users\sonjo\anaconda3\lib\site-packag
es (from beautifulsoup4->nbconvert[webpdf]) (2.3.1)
Requirement already satisfied: webencodings in c:\users\sonjo\anaconda3\lib\site-package
s (from bleach->nbconvert[webpdf]) (0.5.1)
Requirement already satisfied: packaging in c:\users\sonjo\anaconda3\lib\site-packages
(from bleach->nbconvert[webpdf]) (21.3)
Requirement already satisfied: six>=1.9.0 in c:\users\sonjo\anaconda3\lib\site-packages
(from bleach->nbconvert[webpdf]) (1.16.0)
Requirement already satisfied: pywin32>=1.0 in c:\users\sonjo\anaconda3\lib\site-package
s (from jupyter-core->nbconvert[webpdf]) (302)
Requirement already satisfied: zipp>=0.5 in c:\users\sonjo\anaconda3\lib\site-packages
```

```
(from importlib-metadata>=1.4->pyppeteer<1.1,>=1->nbconvert[webpdf]) (3.8.0)
Requirement already satisfied: attrs>=17.4.0 in c:\users\sonjo\anaconda3\lib\site-packag
es (from jsonschema>=2.6->nbformat>=4.4->nbconvert[webpdf]) (21.4.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in c:\users
\sonjo\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=4.4->nbconvert[webpd
f]) (0.18.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\sonjo\anaconda3\lib\si
te-packages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (2.
8.2)
Requirement already satisfied: pyzmq>=23.0 in c:\users\sonjo\anaconda3\lib\site-packages
(from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (23.2.0)
Requirement already satisfied: tornado>=6.0 in c:\users\sonjo\anaconda3\lib\site-package
s (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[webpdf]) (6.1)
Requirement already satisfied: colorama in c:\users\sonjo\anaconda3\lib\site-packages (f
rom tqdm<5.0.0,>=4.42.1->pyppeteer<1.1,>=1->nbconvert[webpdf]) (0.4.5)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\sonjo\anaconda3\lib
\site-packages (from packaging->bleach->nbconvert[webpdf]) (3.0.9)
Installing collected packages: pyee, websockets, pyppeteer
Successfully installed pyee-8.2.2 pyppeteer-1.0.2 websockets-10.4
```

In [1]:

```
pip install nbconvert[webpdf] --allow-chromium-download
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
Usage:
  C:\Users\sonjo\anaconda3\python.exe -m pip install [options] <requirement specifier>
[package-index-options] ...
  C:\Users\sonjo\anaconda3\python.exe -m pip install [options] -r <requirements file> [p
ackage-index-options] ...
  C:\Users\sonjo\anaconda3\python.exe -m pip install [options] [-e] <vcs project url>
...
  C:\Users\sonjo\anaconda3\python.exe -m pip install [options] [-e] <local project path>
...
  C:\Users\sonjo\anaconda3\python.exe -m pip install [options] <archive url/path> ...

no such option: --allow-chromium-download
```

In [ ]: