🚜 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.

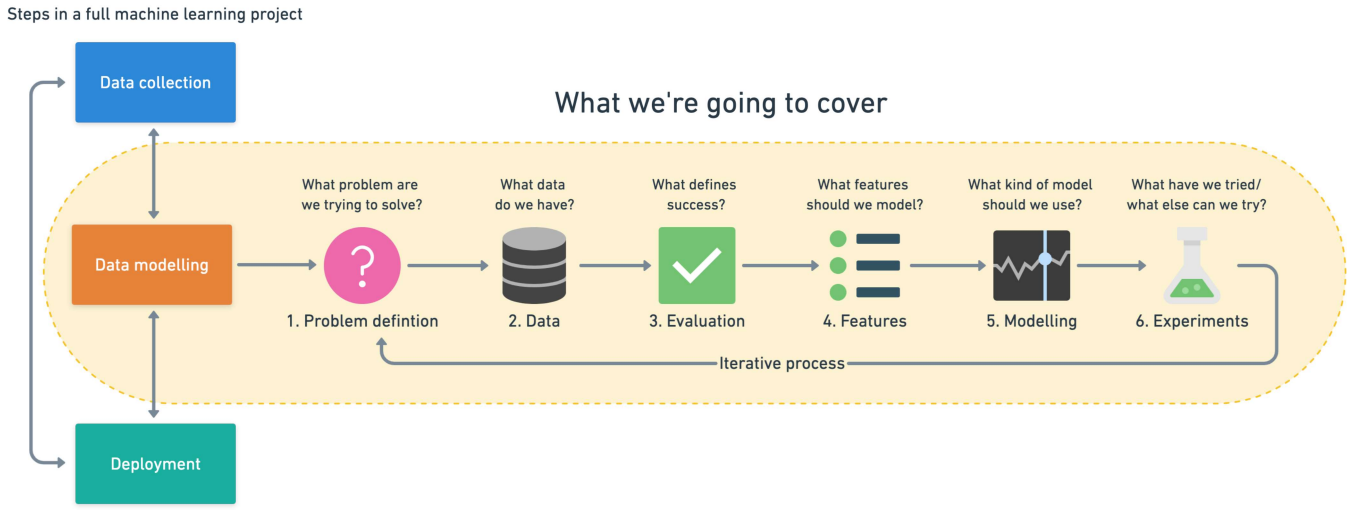
Since we're trying to predict a number, this kind of problem is known as a **regression problem**.

The data and evaluation metric we'll be using (root mean square log error or RMSLE) is from the [Kaggle Bluebook for Bulldozers competition (https://www.kaggle.com/c/bluebook-for-bulldozers/overview)](https://www.kaggle.com/c/bluebook-for-bulldozers/overview).

The techniques used in here have been inspired and adapted from [the fast.ai machine learning course (https://course18.fast.ai/ml)](https://course18.fast.ai/ml).

What we'll end up with

Since we already have a dataset, we'll approach the problem with the following machine learning modelling framework.



To work through these topics, we'll use pandas, Matplotlib and NumPy for data anaylsis, as well as, Scikit-Learn for machine learning and modelling tasks.

We'll work through each step and by the end of the notebook, we'll have a trained machine learning model which predicts the sale price of a bulldozer given different characteristics about it.

1. Problem Definition

For this dataset, the problem we're trying to solve, or better, the question we're trying to answer is,

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

2. Data

Looking at the dataset from Kaggle (https://www.kaggle.com/c/bluebook-for-bulldozers/data), you can you it's a time series problem. This means there's a time attribute to dataset.

In this case, it's historical sales data of bulldozers. Including things like, model type, size, sale date and more.

There are 3 datasets:

1. **Train.csv** - Historical bulldozer sales examples up to 2011 (close to 400,000 examples with 50+ different attributes, including `SalePrice` which is the **target variable**).
2. **Valid.csv** - Historical bulldozer sales examples from January 1 2012 to April 30 2012 (close to 12,000 examples with the same attributes as **Train.csv**).
3. **Test.csv** - Historical bulldozer sales examples from May 1 2012 to November 2012 (close to 12,000 examples but missing the `SalePrice` attribute, as this is what we'll be trying to predict).

3. Evaluation

For this problem, Kaggle has set the evaluation metric to being root mean squared log error (RMSLE) (https://www.kaggle.com/c/bluebook-for-bulldozers/overview/evaluation). As with many regression evaluations, the goal will be to get this value as low as possible.

To see how well our model is doing, we'll calculate the RMSLE and then compare our results to others on the Kaggle leaderboard (https://www.kaggle.com/c/bluebook-for-bulldozers/leaderboard).

4. Features

Features are different parts of the data. During this step, you'll want to start finding out what you can about the data.

One of the most common ways to do this, is to create a **data dictionary**.

For this dataset, Kaggle provide a data dictionary which contains information about what each attribute of the dataset means. You can download this file directly from the Kaggle competition page (https://www.kaggle.com/c/bluebook-for-bulldozers/download/BnI6RAHA0enbg0UfAvGA%2Fversions%2FwBG4f35Q8mAbfkzwCeZn%2Ffiles%2FData% (account required) or view it on Google Sheets.

With all of this being known, let's get started!

First, we'll import the dataset and start exploring. Since we know the evaluation metric we're trying to minimise, our first goal will be building a baseline model and seeing how it stacks up against the competition.

Importing the data and preparing it for modelling

In [1]:
```
# Import data analysis tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

Now we've got our tools for data analysis ready, we can import the data and start to explore it.

For this project, we've downloaded the data from Kaggle (https://www.kaggle.com/c/bluebook-for-bulldozers/data) and stored it under the file path `"../data/"` .

In [2]:
```
# Import the training and validation set
df = pd.read_csv("data/TrainAndValid.csv", low_memory=False)
```

In [3]:
```
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 
```

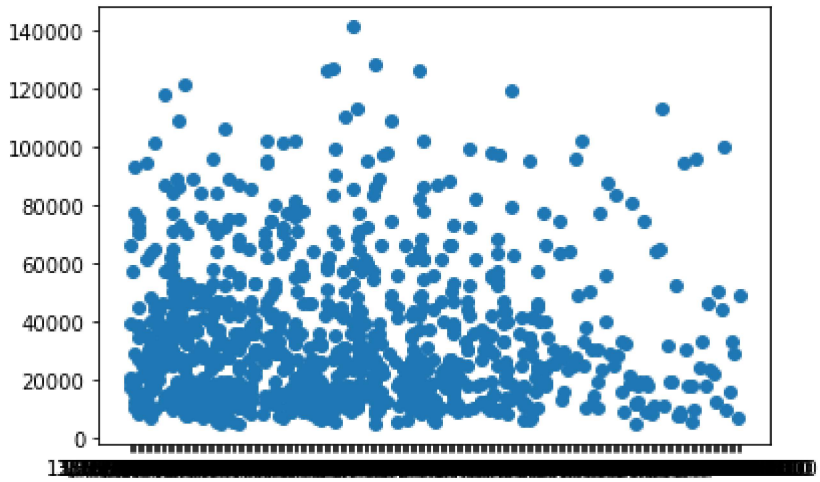
```
dtypes: float64(3), int64(5), object(45)
memory usage: 166.9+ MB
```

In [4]:
```
df.isna().sum()
```

Out[4]:
```
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
```

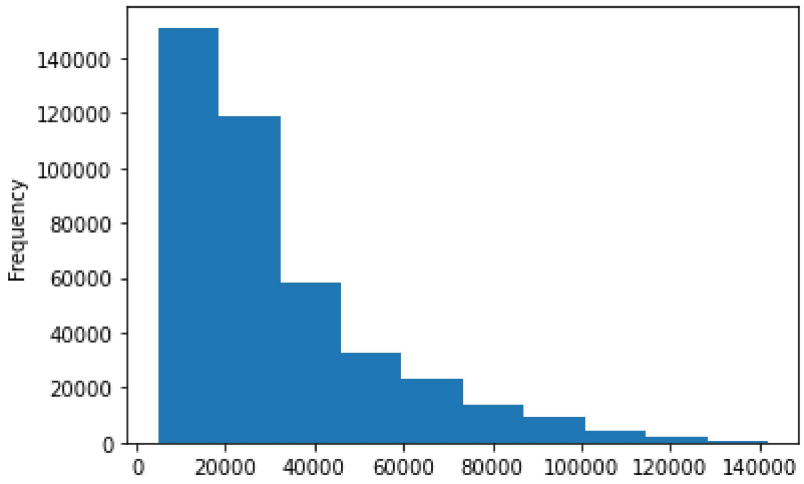
In [5]:
```
fig, ax = plt.subplots()
ax.scatter(df["saledate"][:1000], df["SalePrice"][:1000])
# as saledate type is object graph is messy
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x2d6c1484460>
```



In [6]:
```
df.SalePrice.plot.hist()
```

Out[6]:
```
<AxesSubplot:ylabel='Frequency'>
```



Parsing dates

When working with time series data, it's a good idea to make sure any date data is the format of a [datetime object (https://docs.python.org/3/library/datetime.html)](https://docs.python.org/3/library/datetime.html) (a Python data type which encodes specific information about dates).

In [7]:
```
df = pd.read_csv("data/TrainAndValid.csv",
                 low_memory=False,
                 parse_dates=["saledate"])
```

In [8]:
```
# With parse_dates... check dtype of "saledate"
df.info()
```

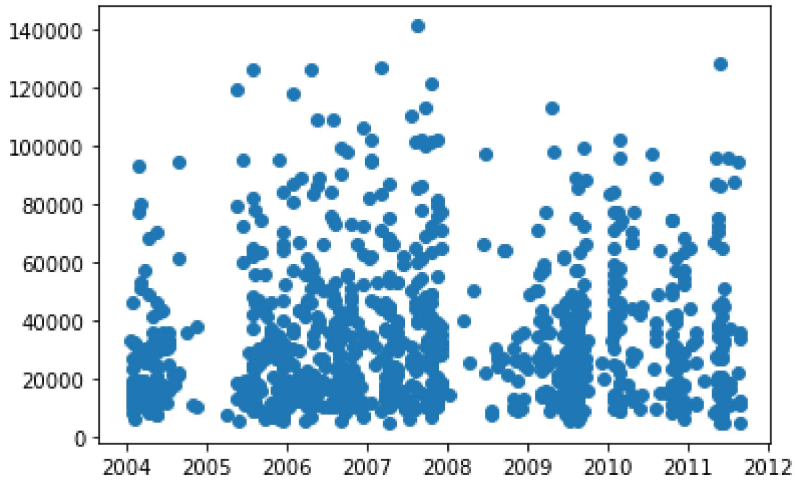
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 412698 entries, 0 to 412697
Data columns (total 53 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   SalesID                   412698 non-null  int64
 1   SalePrice                 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  datetime64[ns]
 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: datetime64[ns](1), float64(3), int64(5), object(44)
memory usage: 166.9+ MB
```

- Now that the saledate datatype is changed graph is more organized

In [9]:
```
fig, ax = plt.subplots()
ax.scatter(df["saledate"][:1000], df["SalePrice"][:1000])
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x2d6c1ef45e0>
```



In [10]:
```
df.head().T
```

Out[10]:

	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 - 110.0 to 120.0 Horsepower	Wheel Loader - 150.0 to 175.0 Horsepower	Skid Steer Loader - 1351.0 to 1601.0 Lb Operat...	Hydraulic Excavator, Track - 12.0 to 14.0 Metr...	Skid Steer Loader - 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

	0	1	2	3	4
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

In [11]:
```
df.saledate.head(20)
```

Out[11]:
```
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]
```

Sort DataFrame by saledate

As we're working on a time series problem and trying to predict future examples given past examples, it makes sense to sort our data by date.

In [12]:
```
# Sort DataFrame in date order
df.sort_values(by=["saledate"], inplace=True, ascending=True)
df.saledate.head(20)
```

Out[12]:
```
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

Since we're going to be manipulating the data, we'll make a copy of the original DataFrame and perform our changes there.

This will keep the original DataFrame in tact if we need it again.

In [13]:
```
# Make a copy of the original DataFrame to perform edits on
df_tmp = df.copy()
```

Add datetime parameters for saledate column

Why?

So we can enrich our dataset with as much information as possible.

Because we imported the data using `read_csv()` and we asked pandas to parse the dates using `parase_dates=["saledate"]`, we can now access the [different datetime attributes (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DatetimeIndex.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DatetimeIndex.html) of the `saledate` column.

In [14]:
```
# Add datetime parameters for saledate
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

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

We could add more of these style of columns, such as, whether it was the start or end of a quarter but these will do for now.

In [15]:
```
df_tmp.head().T
```

Out[15]:

	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

	205615	274835	141296	212552	62755
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 [16]:
```
# Check the different values of different columns
df_tmp.state.value_counts()
```

Out[16]:
```
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 (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html), we find a RandomForestRegressor() (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn-ensemble-randomforestregressor) might be a good candidate.

In [17]:
```
# This won't work since we've got missing numbers and categories
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_jobs=-1)
# n_jobs = -1 means its going to use as many cores on the pc as possible
model.fit(df_tmp.drop("SalePrice", axis=1), df_tmp.SalePrice)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [17], in <cell line: 6>()
      4 model = RandomForestRegressor(n_jobs=-1)
      5 # n_jobs = -1 means its going to use as many cores on the pc as possible
----> 6 model.fit(df_tmp.drop("SalePrice", axis=1), df_tmp.SalePrice)

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\ensemble\_forest.py:327, in BaseForest.fit(self, X, y, sample_weight)
    325 if issparse(y):
    326     raise ValueError("sparse multilabel-indicator for y is not supported.")
--> 327 X, y = self._validate_data(
    328     X, y, multi_output=True, accept_sparse="csc", dtype=DTYPE
    329 )
    330 if sample_weight is not None:
    331     sample_weight = _check_sample_weight(sample_weight, X)

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\base.py:581, in BaseEstimator._validate_data(self, X, y, reset, validate_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
    584 if not no_val_X and check_params.get("ensure_2d", True):

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\utils\validation.py:964, in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    961 if y is None:
    962     raise ValueError("y cannot be None")
--> 964 X = check_array(
    965     X,
    966     accept_sparse=accept_sparse,
    967     accept_large_sparse=accept_large_sparse,
    968     dtype=dtype,
    969     order=order,
    970     copy=copy,
    971     force_all_finite=force_all_finite,
    972     ensure_2d=ensure_2d,
    973     allow_nd=allow_nd,
    974     ensure_min_samples=ensure_min_samples,
    975     ensure_min_features=ensure_min_features,
    976     estimator=estimator,
    977 )
    979 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric)
    981 check_consistent_length(X, y)

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\utils\validation.py:746, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_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(
    749         "Complex data not supported\n{}\n".format(array)
    750     ) from complex_warning

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\pandas\core\generic.p
y:2064, in NDFrame.__array__(self, dtype)
   2063 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064     return np.asarray(self._values, dtype=dtype)

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

In [18]:
```
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
```

All data is not in numeric format

In [19]:
```
# Check for missing values
df_tmp.isna().sum()
```

Out[19]:
```
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 (https://pandas.pydata.org/pandas-docs/stable/reference/general_utility_functions.html#data-types-related-functionality) which allows us to interact and manipulate the types of data.

In [20]:
```
df_tmp.head().T
```

Out[20]:

	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

	205615	274835	141296	212552	62755
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 [21]:
```
pd.api.types.is_string_dtype(df_tmp["UsageBand"])
```

Out[21]:
```
True
```

In [22]:
```
# These columns containing strings
for label, content in df_tmp.items():
    if pd.api.types.is_string_dtype(content):
        print(label)
# label is column name and content is data in that column
```

```
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 [23]:
```
# 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 [24]:
```
# 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()
# content.astype("category").cat.as_ordered() is convering the content to type category as
# as_ordered means they will be ordered and assign a numerical value to each category
```

In [25]:
```
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 [26]:
```
df_tmp.state.cat.categories
```

Out[26]:
```
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 [27]:
```
df_tmp.state.cat.codes
# numeric values of categories
```

Out[27]:
```
205615    43
274835     8
141296     8
212552     8
62755      8
          ..
410879     4
412476     4
411927     4
407124     4
409203     4
Length: 412698, dtype: int8
```

All of our data is categorical and thus we can now turn the categories into numbers, however it's still missing values.

In [28]:
```
df_tmp.isnull().sum()/len(df_tmp)
```

Out[28]:

SalesID	0.000000
SalePrice	0.000000
MachineID	0.000000
ModelID	0.000000
datasource	0.000000
auctioneerID	0.048791
YearMade	0.000000
MachineHoursCurrentMeter	0.642586
UsageBand	0.821492
fiModelDesc	0.000000
fiBaseModel	0.000000
fiSecondaryDesc	0.340993
fiModelSeries	0.857845
fiModelDescriptor	0.818715
ProductSize	0.524851
fiProductClassDesc	0.000000
state	0.000000
ProductGroup	0.000000
ProductGroupDesc	0.000000
Drive_System	0.740520
Enclosure	0.000809
Forks	0.520921
Pad_Type	0.803498
Ride_Control	0.629928
Stick	0.803498
Transmission	0.544444
Turbocharged	0.803498
Blade_Extension	0.937041
Blade_Width	0.937041
Enclosure_Type	0.937041
Engine_Horsepower	0.937041
Hydraulics	0.200062
Pushblock	0.937041
Ripper	0.740864
Scarifier	0.937014
Tip_Control	0.937041
Tire_Size	0.763415
Coupler	0.465277
Coupler_System	0.891024
Grouser_Tracks	0.891264
Hydraulics_Flow	0.891264
Track_Type	0.752378
Undercarriage_Pad_Width	0.750626
Stick_Length	0.752213
Thumb	0.752041
Pattern_Changer	0.752213
Grouser_Type	0.752378
Backhoe_Mounting	0.804428
Blade_Type	0.801610
Travel_Controls	0.801606
Differential_Type	0.826595
Steering_Controls	0.826697
saleYear	0.000000
saleMonth	0.000000
saleDay	0.000000
saleDayofweek	0.000000
saleDayofyear	0.000000

dtype: float64

In the format it's in, it's still good to be worked with, let's save it to file and reimport it so we can continue on.

Save Processed Data

In [29]:
```
# Save preprocessed data
df_tmp.to_csv("data/train_tmp.csv",
              index=False)
```

In [30]:
```
# Import preprocessed data
df_tmp = pd.read_csv("data/train_tmp.csv",
                     low_memory=False)
df_tmp.head().T
```

Out[30]:

	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

	0	1	2	3	4
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 [31]:
```
# Check missing values
df_tmp.isna().sum()
```

Out[31]:
```
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 [32]:
```
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 [33]:
```
# 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 [34]:
```
# 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 mean
            df_tmp[label] = content.fillna(content.median())
```

In [35]:
```
# Check to see how many examples were missing
df_tmp.auctioneerID_is_missing.value_counts()
```

Out[35]:
```
False    392562
True      20136
Name: auctioneerID_is_missing, 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 [36]:
```
# 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 [37]:
```
# Turn categorical variables into numbers and fill missing
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 [38]:
```
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 [39]:
```
df_tmp.isna().sum()
```

Out[39]:
```
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 [40]:
```
df_tmp.head().T
```

Out[40]:

	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 (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html).

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.

In [41]:
```
%%time
# Calculates how much time this perticular cell takes to run

# Instantiate model
model = RandomForestRegressor(n_jobs=-1)

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

```
CPU times: total: 22min 36s
Wall time: 2min 10s
```

Out[41]:
```
RandomForestRegressor(n_jobs=-1)
```

In [42]:
```
# Score the model
model.score(df_tmp.drop("SalePrice", axis=1), df_tmp.SalePrice)
```

Out[42]:
```
0.9875826947456512
```

Question: Why is this metric not reliable? We have evaluated it on the exact same data and not a validation set

Splitting data into train/valid sets

In [43]:
```
df_tmp.head()
```

Out[43]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	U
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 (https://www.kaggle.com/c/bluebook-for-bulldozers/data), 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 (https://www.fast.ai/2017/11/13/validation-sets/)](https://www.fast.ai/2017/11/13/validation-sets/) by Rachel Thomas.

In [44]:
```
df_tmp.saleYear.value_counts()
```

Out[44]:
```
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 [45]:
```
# 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[45]:
```
(11573, 401125)
```

In [46]:
```
# 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[46]:
```
((401125, 102), (401125,), (11573, 102), (11573,))
```

Building an evaluation function

According to Kaggle for the Bluebook for Bulldozers competition, [the evaluation function (https://www.kaggle.com/c/bluebook-for-bulldozers/overview/evaluation)](https://www.kaggle.com/c/bluebook-for-bulldozers/overview/evaluation) 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 (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_log_error.html#sklearn.metrics.mean_squar (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 [47]:
```
# Create evaluation function (the competition uses Root Mean Square Log Error)
from sklearn.metrics import mean_squared_log_error, mean_absolute_error, r2_score

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": r2_score(y_train, train_preds),
              "Valid R^2": r2_score(y_valid, val_preds)}
    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 [48]:
```
# This takes too long...

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

In [49]:
```
len(X_train)
```

Out[49]:
```
401125
```

Depending your computer, making calculations on ~400,000 rows may take a while...

Let's alter the number of samples each `n_estimator` in the `RandomForestRegressor` (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html) see's using the `max_samples` parameter.

In [50]:
```
# 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).

In [51]:
```
%%time
# Cutting down the max number of samples each tree can see improves training time
model.fit(X_train, y_train)
```

```
CPU times: total: 43.5 s
Wall time: 5.74 s
```

Out[51]:
```
RandomForestRegressor(max_samples=10000, n_jobs=-1)
```

In [52]:
```
show_scores(model)
```

Out[52]:
```
{'Training MAE': 5560.213845010907,
 'Valid MAE': 7149.241240819148,
 'Training RMSLE': 0.25760537127844607,
 'Valid RMSLE': 0.29288010073917653,
 'Training R^2': 0.8606646245011151,
 'Valid R^2': 0.8338198984362071}
```

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 [53]:
```
%%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(n_jobs = -1),
                              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
CPU times: total: 4min 12s
Wall time: 5min
```

Out[53]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(n_jobs=-1), 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, 1
2, 14, 16, 18]),
                                        'n_estimators': array([10, 20, 30, 40, 50, 60, 7
0, 80, 90])},
                   verbose=True)
```

In [54]:
```
show_scores(rs_model)
```

Out[54]:
```
{'Training MAE': 6524.040847503526,
 'Valid MAE': 8074.54028081687,
 'Training RMSLE': 0.29245547886034223,
 'Valid RMSLE': 0.3214474993853766,
 'Training R^2': 0.807965904954355,
 'Valid R^2': 0.7777503893500135}
```

Train a model with the best parameters

After trying 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 [55]:
```
%%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)
```

```
CPU times: total: 9min 53s
Wall time: 58.2 s
```

Out[55]:
```
RandomForestRegressor(max_features=0.5, min_samples_split=14, n_estimators=90,
                      n_jobs=-1)
```

In [56]:
```
show_scores(ideal_model)
```

Out[56]:
```
{'Training MAE': 2928.3350332837163,
 'Valid MAE': 5908.198864516647,
 'Training RMSLE': 0.1433564598491311,
 'Valid RMSLE': 0.24474366771324324,
 'Training R^2': 0.9596341292494392,
 'Valid R^2': 0.8835024979133405}
```

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.

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 [57]:
```
df_test = pd.read_csv("data/Test.csv",
                      parse_dates=["saledate"])
df_test.head()
```

Out[57]:

	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

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 [58]:
```
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

    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 [59]:
```
df_test = preprocess_data(df_test)
df_test.head()
```

Out[59]:

	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 [60]:
```
X_train.head()
```

Out[60]:

	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 [61]:
```
# Make predictions on the test dataset using the best model
test_preds = ideal_model.predict(df_test)
```

```
C:\Users\sonar\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\base.p
y:493: FutureWarning: The feature names should match those that were passed during fit. S
tarting version 1.2, an error will be raised.
Feature names seen at fit time, yet now missing:
- auctioneerID_is_missing

  warnings.warn(message, FutureWarning)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [61], in <cell line: 2>()
      1 # Make predictions on the test dataset using the best model
----> 2 test_preds = ideal_model.predict(df_test)

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\ensemble\_fore
st.py:971, in ForestRegressor.predict(self, X)
    969 check_is_fitted(self)
    970 # Check data
--> 971 X = self._validate_X_predict(X)
    973 # Assign chunk of trees to jobs
    974 n_jobs, _, _ = _partition_estimators(self.n_estimators, self.n_jobs)

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\ensemble\_fore
st.py:579, in BaseForest._validate_X_predict(self, X)
    576 """
    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.intc):
    581     raise ValueError("No support for np.int64 index based sparse matrices")

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\base.py:585, i
n BaseEstimator._validate_data(self, X, y, reset, validate_separately, **check_params)
    582     out = X, y
    584 if not no_val_X and check_params.get("ensure_2d", True):
--> 585     self._check_n_features(X, reset=reset)
    587 return out

File ~\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\base.py:400, i
n BaseEstimator._check_n_features(self, X, reset)
    397     return
    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."
    403     )

ValueError: X has 101 features, but RandomForestRegressor is expecting 102 features as in
put.
```

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 [62]:
```
# We can find how the columns differ using sets
set(X_train.columns) - set(df_test.columns)
```

Out[62]:
```
{'auctioneerID_is_missing'}
```

In this case, it's because the test dataset wasn't missing any `auctioneerID` fields.

To fix it, we'll add a column to the test dataset called `auctioneerID_is_missing` and fill it with `False`, since none of the `auctioneerID` fields are missing in the test dataset.

In [63]:
```
# Match test dataset columns to training dataset
df_test["auctioneerID_is_missing"] = False
df_test.head()
```

Out[63]:

	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

Now the test dataset matches the training dataset, we should be able to make predictions on it using our trained model.

In [64]:
```
# Make predictions on the test dataset using the best model
test_preds = ideal_model.predict(df_test)
```

```
C:\Users\sonar\Desktop\ML\bulldozer-price-prediction\env\lib\site-packages\sklearn\base.p
y:493: FutureWarning: The feature names should match those that were passed during fit. S
tarting version 1.2, an error will be raised.
Feature names must be in the same order as they were in fit.

  warnings.warn(message, FutureWarning)
```

When looking at the [Kaggle submission requirements (https://www.kaggle.com/c/bluebook-for-bulldozers/overview/evaluation)](https://www.kaggle.com/c/bluebook-for-bulldozers/overview/evaluation), 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 [65]:
```
# 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[65]:

	SalesID	SalePrice
0	1227829	22919.108854
1	1227844	21239.509854
2	1227847	47049.480912
3	1227848	60140.512036
4	1227863	43899.692392
...	...	...
12452	6643171	42356.477407
12453	6643173	15837.029346
12454	6643184	15130.982434
12455	6643186	17215.902496
12456	6643196	30230.558346

12457 rows × 2 columns

In [66]:
```
# Export to csv...
#df_preds.to_csv("../data/bluebook-for-bulldozers/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 [67]:
```
# Find feature importance of our best model
ideal_model.feature_importances_
```

Out[67]:
```
array([3.38750390e-02, 1.70389626e-02, 4.05365689e-02, 1.73728637e-03,
       3.31727823e-03, 2.03447931e-01, 3.16399848e-03, 9.83476314e-04,
       4.43690713e-02, 5.15824964e-02, 6.58687401e-02, 4.55615942e-03,
       1.46873494e-02, 1.51504283e-01, 4.46492598e-02, 5.95317213e-03,
       3.23270455e-03, 2.28481827e-03, 3.34323640e-03, 5.98333973e-02,
       4.47677407e-04, 1.67362985e-04, 8.26044247e-04, 2.00859974e-04,
       9.45280781e-04, 2.77074193e-05, 2.31982845e-04, 6.91386095e-03,
       6.21198130e-04, 1.00655538e-03, 3.34252541e-03, 1.52302518e-03,
       2.91420587e-03, 6.17545803e-04, 3.27197892e-03, 9.29481042e-03,
       7.97472550e-04, 1.29976456e-02, 2.06161539e-03, 1.29629887e-03,
       1.44020065e-03, 9.81092123e-04, 2.09658332e-03, 6.08679406e-04,
       6.25771056e-04, 3.45487161e-04, 4.60941053e-04, 2.32304514e-03,
       7.72977417e-04, 3.42058090e-04, 6.10738889e-04, 7.26899811e-02,
       3.80127616e-03, 5.67725103e-03, 2.91090639e-03, 9.83071533e-03,
       2.48633222e-04, 1.71221159e-03, 3.20186808e-04, 0.00000000e+00,
       0.00000000e+00, 3.34019805e-03, 1.46259399e-03, 1.16929906e-02,
       3.08799416e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 9.10962870e-05, 4.05674481e-06, 2.00021396e-04,
       5.28584314e-06, 1.53964748e-04, 8.00386811e-06, 3.42288783e-04,
       1.39503428e-05, 1.07009768e-03, 2.24531614e-04, 9.61667194e-05,
       1.78296756e-03, 2.90206724e-03, 5.12337165e-03, 9.65582808e-04,
       2.02616232e-03, 1.82122923e-03, 3.15526187e-03, 2.81137525e-04,
       1.25392481e-02, 3.59027846e-03, 1.21379319e-03, 7.49539833e-05,
       1.07859781e-04, 9.07869417e-05, 7.56754281e-05, 7.91148991e-05,
       3.04551273e-04, 4.14113635e-04, 1.74896982e-04, 1.17912669e-04,
       1.49354408e-04, 1.54873223e-04])
```

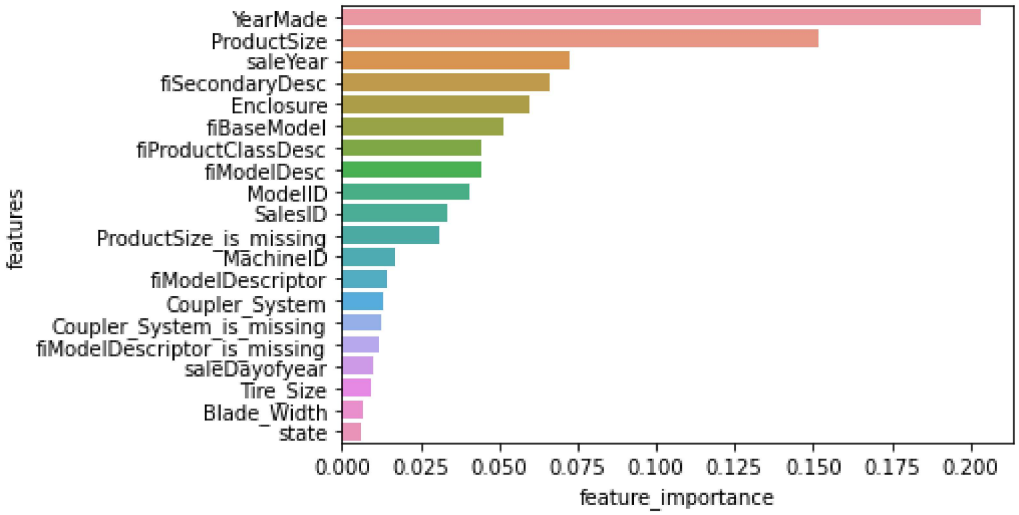
In [68]:
```
# Install Seaborn package in current environment (if you don't have it)
# import sys
# !conda install --yes --prefix {sys.prefix} seaborn
```

In [69]:
```
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 [70]:
```
plot_features(X_train.columns, ideal_model.feature_importances_)
```



In [71]:
```
sum(ideal_model.feature_importances_)
```

Out[71]:
```
1.0000000000000004
```

In [ ]:
```

```