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.

The techniques used in here have been inspired and adapted from the fast.ai machine learning course.

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 and previous examples of how much similar bulldozers have been sold
for?
```

2. Data

Looking at the dataset from Kaggle, 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:

- **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**).
- **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**).
- **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) between the actual and predicted auction prices.

NOTE: As with many regression evaluations metric, the goal will be to get this value as low as possible(minimize the error).

For example, the goal for this project will be to build a machine learning model which minimizes RMSLE

To see how well the model is doing, we'll calculate the RMSLE and then compare the results to others on the Kaggle leaderboard.

4. Features

Features are different parts of the data. During this step, we'll want to start finding out 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, this file is directly from the Kaggle competition page (account required) or view it on Google Sheets.

In [111…

```
# Import data analysis tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns

# Model
from sklearn.ensemble import RandomForestRegressor
```

Read the Data

In [3]:

```
# Import the training and validation set
df = pd.read_csv('data/bluebook-for-bulldozers/TrainAndValid.csv',
                 low_memory=False)
df.head()
```

Out[3]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	Usage
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	M

5 rows × 53 columns

In [4]:

```
# No 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  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 [5]:

```
# To check for missing null values
df.isna().sum()
```

Out[5]:

```
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 [6]:

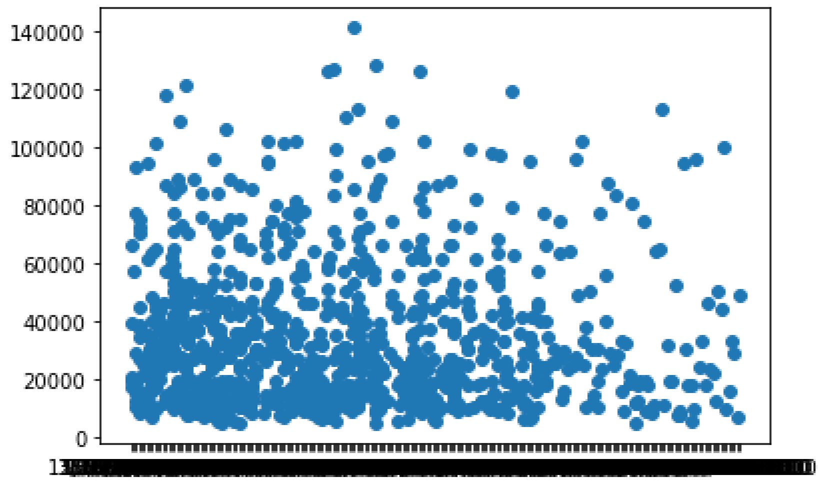
```
df.columns
```

Out[6]:

```
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')
```

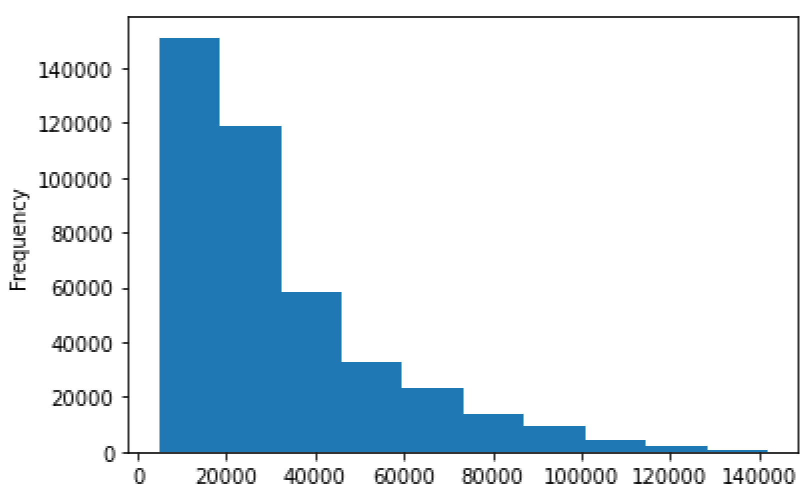
In [7]:
```
# Plot the first 1000 dataset
fig, ax = plt.subplots()
ax.scatter(df['saledate'][:1000], df['SalePrice'][:1000]); # (scatter plot: the x axis cor
```



In [8]:
```
df.saledate[:1000]
```

Out[8]:
```
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 [9]:
```
df.SalePrice.plot.hist();
```



Parsing dates

When working 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 `parse_dates` parameter

it's a good idea to make sure any date data is the format of a datetime object (a Python data type which encodes specific information about dates).

In [10]:
```
# Import data again but this time with parse dates
df = pd.read_csv('data/bluebook-for-bulldozers/TrainAndValid.csv',
                 low_memory = False,
                 parse_dates=['saledate'])
```

In [11]:
```
df.saledate.dtype
```

Out[11]:
```
dtype('<M8[ns]')
```

In [12]:
```
df.saledate[:1000]
```

Out[12]:
```
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]
```

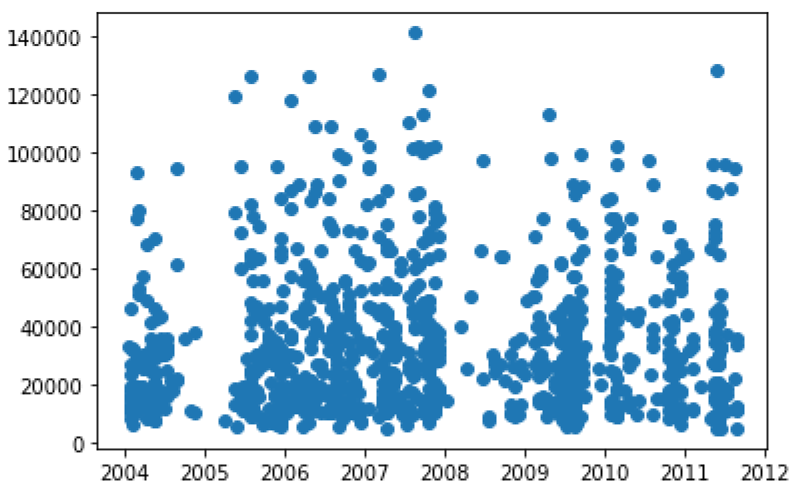
In [13]:
```
# With parse_dates... check dtype of "saledate"
df.info()
```

```
<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
```

In [14]:

```
# Plot the first 1000 dataset
fig, ax = plt.subplots()
ax.scatter(df['saledate'][:1000], df['SalePrice'][:1000]);
```



In [15]:

```
df.head()
```

Out[15]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	Usage
0	1139246	66000.0	999089	3157	121	3.0	2004	68.0	
1	1139248	57000.0	117657	77	121	3.0	1996	4640.0	

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	Usage
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	M

5 rows × 53 columns

In [16]:

```
df.head().T
```

Out[16]:

	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

	0	1	2	3	4
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

In [17]:

```
df.saledate.head(20)
```

Out[17]:

```
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 (when working with time series, it's a good idea to sort it by date) .

In [18]:

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

Out[18]:

```
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]
```

In [19]:

```
df.head(20)
```

Out[19]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter
205615	1646770	9500.0	1126363	8434	132	18.0	1974	NaN
274835	1821514	14000.0	1194089	10150	132	99.0	1980	NaN
141296	1505138	50000.0	1473654	4139	132	99.0	1978	NaN

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter
212552	1671174	16000.0	1327630	8591	132	99.0	1980	NaN
62755	1329056	22000.0	1336053	4089	132	99.0	1984	NaN
54653	1301884	23500.0	1182999	4123	132	99.0	1976	NaN
81383	1379228	31000.0	1082797	7620	132	99.0	1986	NaN
204924	1645390	11750.0	1527216	8202	132	99.0	1970	NaN
135376	1493279	63000.0	1363756	2759	132	99.0	1987	NaN
113390	1449549	13000.0	1289412	3356	132	99.0	1966	NaN
113394	1449555	10500.0	1102310	3356	132	99.0	1966	NaN
116419	1453775	20000.0	1514650	7008	132	99.0	1974	NaN
32138	1264985	20000.0	1204499	6788	132	99.0	1984	NaN
127610	1475641	23500.0	1194367	7277	132	99.0	1973	NaN
76171	1364654	14000.0	1270628	7289	132	99.0	1968	NaN
127000	1474844	11250.0	1279993	7257	132	99.0	1979	NaN
128130	1476264	29000.0	1245504	7277	132	99.0	1978	NaN
127626	1475662	22000.0	1242833	7277	132	99.0	1973	NaN
55455	1305337	17000.0	1517075	3356	132	99.0	1972	NaN
55454	1305336	17000.0	1236263	3356	132	99.0	1972	NaN

20 rows × 53 columns

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 [20]:
```
# 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 parse_dates=["saledate"], we can now access the different datetime attributes of the saledate column.

In [21]:

```
# Add datetime parameters for saledate column
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(Now we've enriched our DataFrame with date time features, we can
df_tmp.drop('saledate', axis=1, inplace=True)
```

In [22]:

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

Out[22]:

	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

	205615	274835	141296	212552	62755
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

	205615	274835	141296	212552	62755
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 [23]:

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

Out[23]:

```
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
Iowa               1336
Montana            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.

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

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

In [24]:
```
# Check for missing categories and different datatypes
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 [25]:

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

Out[25]:

```
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 [26]:

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

Out[26]:

	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

	205615	274835	141296	212552	62755
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

	205615	274835	141296	212552	62755
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 [27]:

```
pd.api.types.is_string_dtype(df_tmp['UsageBand'])
```

Out[27]:

```
True
```

In [28]:

```
# Find the columns which 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 [30]:

```
# 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 [31]:

```
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 [32]:

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

Out[32]:

```
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 [33]:

```
df_tmp.state.cat.codes
```

Out[33]:

```
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 [34]:

```
# Check missing data
df_tmp.isnull().sum()/len(df_tmp)
```

Out[34]:

```
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 [35]:

```
# Save preprocessed data
df_tmp.to_csv('data/bluebook-for-bulldozers/train_tmp.csv', index=False)
```

In [36]:

```
# Import preprocessed data
df_tmp = pd.read_csv('data/bluebook-for-bulldozers/train_tmp.csv', low_memory=False)
df_tmp.head().T
```

Out[36]:

	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

	0	1	2	3	4
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 [37]:

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

Out[37]:

```
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 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 [40]:

```
# Check for the columns that have numeric values
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 [41]:

```
# Check for which numeric columns have null(missing) 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 [42]:

```
# 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 us if the data was missing or 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())
```

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 [43]:

```
# 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 [44]:

```
# Check to see how many examples were missing
```

```
df_tmp.auctioneerID_is_missing.value_counts()
```

Out[44]:

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

In [45]:

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

Out[45]:

```
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 [46]:

```
# 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 [47]:

```
# 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)

    # Turn categories into numbers and add +1 because pandas encodes missing categ
    df_tmp[label] = pd.Categorical(content).codes + 1
```

In [48]:

```
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 [49]:

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

Out[49]:

```
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 [50]:

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

Out[50]:

	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 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.

In [51]:
```
len(df_tmp)
```

Out[51]:
```
412698
```

In [56]:
```
%%time

# Instantiate model
model = RandomForestRegressor(n_jobs=-1,
                              random_state=42)

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

```
Wall time: 8min 47s
```

Out[56]:
```
RandomForestRegressor(n_jobs=-1, random_state=42)
```

In [57]:
```
# Score the model
model.score(df_tmp.drop('SalePrice', axis = 1), df_tmp.SalePrice)
```

Out[57]:
```
0.9875468079970562
```

The metric above is not reliable.

Splitting data into train/validation sets

In [58]:
```
df_tmp.head()
```

Out[58]:

	SalesID	SalePrice	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	Usage
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

In [59]:

```
df_tmp.saleYear
```

Out[59]:

```
0         1989
1         1989
2         1989
3         1989
4         1989
          ...
412693    2012
412694    2012
412695    2012
412696    2012
412697    2012
Name: saleYear, Length: 412698, dtype: int64
```

In [60]:

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

Out[60]:

```
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 [61]:

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

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

Out[61]:

```
(11573, 401125)
```

In [62]:

```
# Split the data into X and 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[62]:
```
((401125, 102), (401125,), (11573, 102), (11573,))
```

In [63]:
```
y_train
```

Out[63]:
```
0          9500.0
1         14000.0
2         50000.0
3         16000.0
4         22000.0
           ...
401120    29000.0
401121    11000.0
401122    11000.0
401123    18000.0
401124    13500.0
Name: SalePrice, Length: 401125, dtype: float64
```

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.

In [67]:
```
# Create evaluation function (the competition uses Root Mean Square Log Error RMSLE)
from sklearn.metrics import mean_squared_log_error, mean_absolute_error

def rmsle(y_test, y_preds):
    """
    calculates root mean squared log error between predictions
    and true labels
    """
    return np.sqrt(mean_squared_log_error(y_test, y_preds))

# Create a 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)

Retraining 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 your 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 [65]:

```
# This takes far too long for experimenting...

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

In [70]:

```
len(X_train)
```

Out[70]:

```
401125
```

Depending on 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 see's using the max_samples parameter.

In [71]:

```
# Change max samples in RandomForestRegressor
model = RandomForestRegressor(n_jobs=-1,
                              random_state=42,
                              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 [72]:

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

```
Wall time: 25.3 s
```

Out[72]:

```
RandomForestRegressor(max_samples=10000, n_jobs=-1, random_state=42)
```

In [73]:

```
show_scores(model)
```

Out[73]:

```
{'Training MAE': 5561.2988092240585,
 'Valid MAE': 7177.26365505919,
 'Training RMSLE': 0.257745378256977,
 'Valid RMSLE': 0.29362638671089003,
 'Training R^2': 0.8606658995199189,
 'Valid R^2': 0.8320374995090507}
```

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.

In [74]:
```
%%time
from sklearn.model_selection import RandomizedSearchCV

# Different RandomForestRegressor 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]}

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

# Fit the RandomizedSearchCV model
rs_model.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Wall time: 12min
```

Out[74]:
```
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, 11,
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, 70,
80, 90])},
                   verbose=True)
```

In [75]:
```
# Find the best parameters from the RandomizedSearch
rs_model.best_params_
```

Out[75]:
```
{'n_estimators': 80,
 'min_samples_split': 16,
 'min_samples_leaf': 7,
 'max_samples': 10000,
 'max_features': 0.5,
 'max_depth': None}
```

In [76]:
```
# Evaluate the RandomizedSearch model
show_scores(rs_model)
```

Out[76]:
```
{'Training MAE': 6220.152642892111,
 'Valid MAE': 7523.711301067172,
 'Training RMSLE': 0.2816262592419515,
 'Valid RMSLE': 0.30412742636021506,
 'Training R^2': 0.8276368097596425,
 'Valid R^2': 0.8153446700929028}
```

Train a model with the best parameters

In [80]:
```
%%time
```

```python
# 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,
                                    random_state = 42)
# Fit the ideal model
ideal_model.fit(X_train, y_train)
```

```
Wall time: 3min 45s
```

Out[80]:
```
RandomForestRegressor(max_features=0.5, min_samples_split=14, n_estimators=90,
                      n_jobs=-1, random_state=42)
```

In [81]:
```python
# scores for ideal model (trained on all the data)
show_scores(ideal_model)
```

Out[81]:
```
{'Training MAE': 2929.921568433354,
 'Valid MAE': 5926.828732131214,
 'Training RMSLE': 0.14357234549234102,
 'Valid RMSLE': 0.24477262723855048,
 'Training R^2': 0.9596218418812977,
 'Valid R^2': 0.8829608391304742}
```

In [82]:
```python
# scores for rs_model (only trained on ~10,000 samples)
show_scores(rs_model)
```

Out[82]:
```
{'Training MAE': 6220.152642892111,
 'Valid MAE': 7523.711301067172,
 'Training RMSLE': 0.2816262592419515,
 'Valid RMSLE': 0.30412742636021506,
 'Training R^2': 0.8276368097596425,
 'Valid R^2': 0.8153446700929028}
```

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 [90]:
```python
# Import the test data
df_test = pd.read_csv('data/bluebook-for-bulldozers/Test.csv',
                      parse_dates=['saledate'])
df_test
```

Out[90]:

	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

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand
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
...	...	...	...	...	...	...	...	...
12452	6643171	2558317	21450	149	2	2008	NaN	NaN
12453	6643173	2558332	21434	149	2	2005	NaN	NaN
12454	6643184	2558342	21437	149	2	1000	NaN	NaN
12455	6643186	2558343	21437	149	2	2006	NaN	NaN
12456	6643196	2558346	21446	149	2	2008	NaN	NaN

12457 rows × 52 columns

In [91]:

```
df_test.head()
```

Out[91]:

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

5 rows × 52 columns

Preprocessing the data(getting the test data in the same format as our training 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 [92]:
```
def preprocess_data(df):
    """
    performs transformations on df and returns transformed 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():

                # Add a binary column which tells us if the data was missing or not
                df[label+"_is_missing"] = pd.isnull(content)

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

        # Turn categorical missing 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
```

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

In [93]:
```
# Process the test data
df_test = preprocess_data(df_test)
df_test.head()
```

Out[93]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiM
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 [94]:
```
X_train.head()
```

Out[94]:

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiM
0	1646770	1126363	8434	132	18.0	1974	0.0	0	
1	1821514	1194089	10150	132	99.0	1980	0.0	0	

	SalesID	MachineID	ModelID	datasource	auctioneerID	YearMade	MachineHoursCurrentMeter	UsageBand	fiM
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 [95]:

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

Out[95]:

```
{'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 [96]:

```
# Match test dataset columns to training dataset
df_test['auctioneerID_is_missing'] = False
df_test
```

Out[96]:

	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
...	...	...	...	...	...	...	...	...
12452	6643171	2558317	21450	149	2	2008	3525.0	0
12453	6643173	2558332	21434	149	2	2005	3525.0	0
12454	6643184	2558342	21437	149	2	1000	3525.0	0
12455	6643186	2558343	21437	149	2	2006	3525.0	0
12456	6643196	2558346	21446	149	2	2008	3525.0	0

12457 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 [97]:

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

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.

In [98]:

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

Out[98]:

	SalesID	SalesPrice
0	1227829	19789.714023
1	1227844	19565.299439
2	1227847	48800.694515
3	1227848	61935.439690
4	1227863	42767.772640
...	...	...
12452	6643171	42265.421622
12453	6643173	14782.962987
12454	6643184	15604.288740
12455	6643186	19025.395893
12456	6643196	29334.359121

12457 rows × 2 columns

In [99]:

```
# Export to CSV
df_preds.to_csv('data/bluebook-for-bulldozers/predictions.csv', index = False)
```

Feature Importance

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

In [113…

```
# Find feature importance of our best model
ideal_model.feature_importances_
```

Out[113…

```
array([3.36536104e-02, 2.06759219e-02, 4.12493898e-02, 1.65309161e-03,
       3.37161218e-03, 2.06931736e-01, 3.23666071e-03, 1.03610744e-03,
       4.53494594e-02, 4.41122835e-02, 6.01968492e-02, 4.71102363e-03,
       1.59697733e-02, 1.50896732e-01, 4.57018149e-02, 5.95545126e-03,
       2.22480465e-03, 3.33266898e-03, 2.95703302e-03, 6.73951018e-02,
       6.65158539e-04, 3.98094940e-05, 9.84704968e-04, 2.85986069e-04,
       1.12813990e-03, 2.47446180e-05, 9.54863612e-04, 8.81712425e-03,
       1.92386650e-03, 2.12645204e-03, 3.62305282e-03, 3.44945894e-03,
       3.27317013e-03, 1.77639716e-03, 1.14087894e-03, 5.80942535e-03,
       8.12450169e-04, 1.31354263e-02, 1.39777375e-03, 3.28177035e-03,
       1.28211228e-03, 8.69687238e-04, 2.05251738e-03, 5.97963087e-04,
       5.41678527e-04, 3.54317048e-04, 3.88821556e-04, 2.41962294e-03,
       8.59586252e-04, 2.92501513e-04, 2.15302259e-04, 7.33725591e-02,
       3.79418026e-03, 5.68832526e-03, 2.90982570e-03, 9.82376367e-03,
       2.59268029e-04, 1.74795886e-03, 3.23057317e-04, 0.00000000e+00,
       0.00000000e+00, 2.70655122e-03, 1.26563042e-03, 3.88600447e-03,
       3.21688076e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.02259045e-04, 1.08902429e-05, 3.10665050e-04,
       3.99741175e-06, 1.06257032e-04, 3.58165848e-06, 3.05858606e-04,
       1.69755228e-05, 1.39801322e-03, 1.86677952e-03, 3.62449442e-03,
```

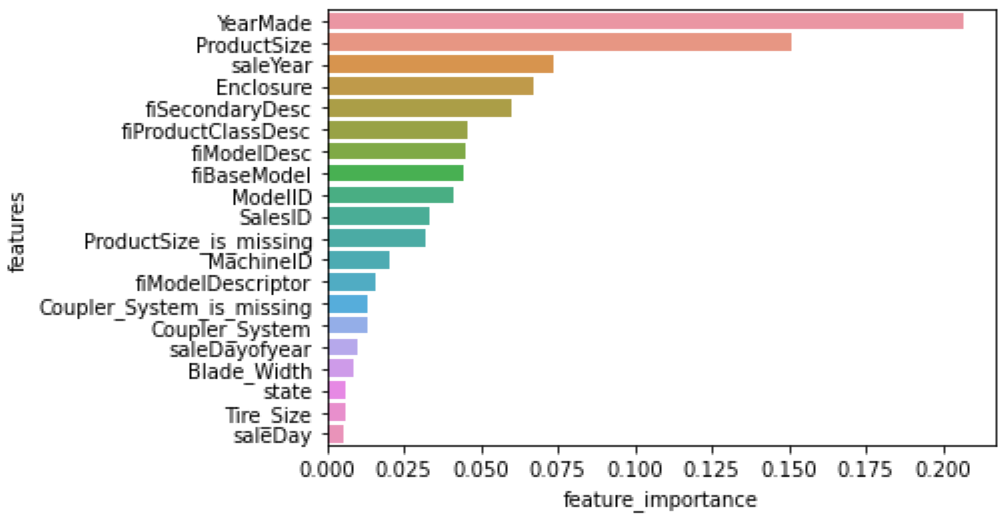
```
       2.26585416e-04, 3.61935932e-03, 9.31853135e-04, 1.33423840e-03,
       1.34288116e-03, 1.12092114e-03, 2.71203804e-03, 2.71750836e-04,
       1.34054763e-02, 1.76499457e-03, 1.11672051e-03, 4.27793962e-05,
       1.84939697e-04, 4.09157804e-05, 8.90131799e-05, 4.49277953e-05,
       4.21445103e-05, 2.97559133e-04, 1.86343248e-04, 1.60013050e-04,
       1.02015171e-04, 1.30937225e-04])
```

In [115…

```python
# 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 [116…

```python
plot_features(X_train.columns, ideal_model.feature_importances_)
```



In [117…

```python
sum(ideal_model.feature_importances_)
```

Out[117…

```
1.0
```

In [105…

```python
df.ProductSize.isna().sum()
```

Out[105…

```
216605
```

In [106…

```python
df.ProductSize.value_counts()
```

Out[106…

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

In [107…

```python
df.Turbocharged.value_counts()
```

Out[107…

```
None or Unspecified    77111
```

```
Yes                        3985
Name: Turbocharged, dtype: int64
```

In [108…

```python
df.Thumb.value_counts()
```

Out[108…

```
None or Unspecified    85074
Manual                  9678
Hydraulic               7580
Name: Thumb, dtype: int64
```