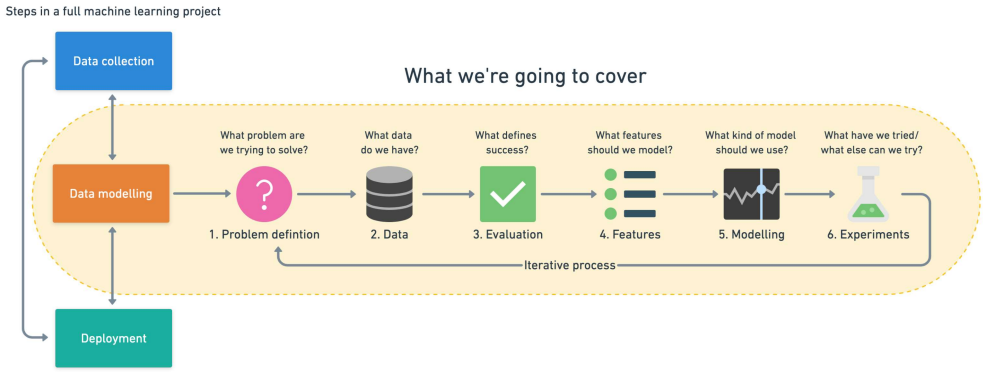
# 🚜 Predicting the Sale Price of Bulldozers using Machine Learning

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

The data and evaluation metric we'll be using (root mean square log error or RMSLE) is from the Kaggle Bluebook for Bulldozers competition.

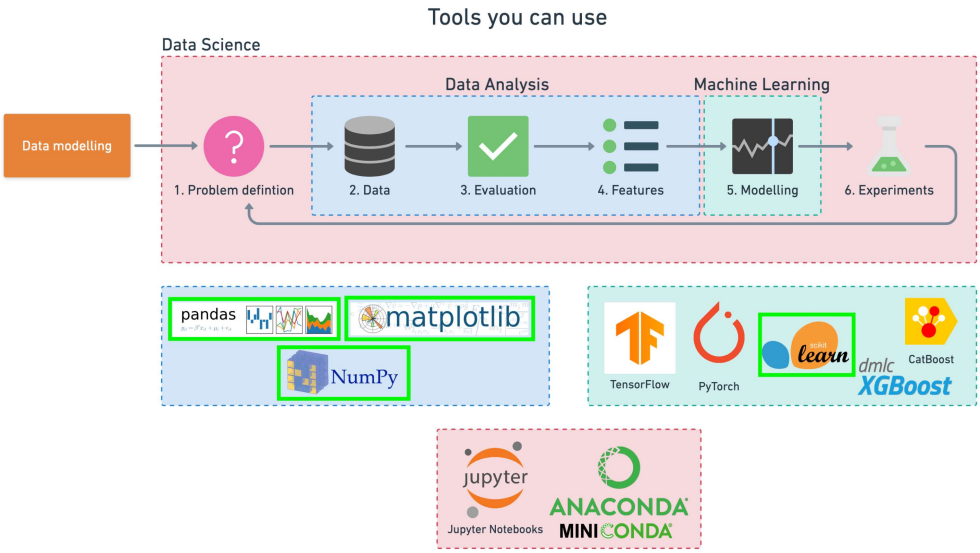
## What we'll end up with

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




6 Step Machine Learning Modelling Framework (read more)

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.




Tools which can be used for each step of the machine learning modelling process.

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

> 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, we can see 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). 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.

# 4. Features

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 (account required) or view it on Google Sheets.

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 [ ]:

```
# Import data analysis tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:

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

In [ ]:

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

Out[ ]:

| | 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** | 11/16/2006 0:00 | 3/26/2004 0:00 | 2/26/2004 0:00 | 5/19/2011 0:00 | 7/23/2009 0:00 |
| **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 |

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **Forks** | None or Unspecified | None or Unspecified | None or Unspecified | NaN | None or Unspecified |
| **Pad_Type** | NaN | NaN | NaN | NaN | NaN |
| **Ride_Control** | None or Unspecified | None or Unspecified | NaN | NaN | NaN |
| **Stick** | NaN | NaN | NaN | NaN | NaN |
| **Transmission** | NaN | NaN | NaN | NaN | NaN |
| **Turbocharged** | NaN | NaN | NaN | NaN | NaN |
| **Blade_Extension** | NaN | NaN | NaN | NaN | NaN |
| **Blade_Width** | NaN | NaN | NaN | NaN | NaN |
| **Enclosure_Type** | NaN | NaN | NaN | NaN | NaN |
| **Engine_Horsepower** | NaN | NaN | NaN | NaN | NaN |
| **Hydraulics** | 2 Valve | 2 Valve | Auxiliary | 2 Valve | Auxiliary |
| **Pushblock** | NaN | NaN | NaN | NaN | NaN |
| **Ripper** | NaN | NaN | NaN | NaN | NaN |
| **Scarifier** | NaN | NaN | NaN | NaN | NaN |
| **Tip_Control** | NaN | NaN | NaN | NaN | NaN |
| **Tire_Size** | None or Unspecified | 23.5 | NaN | NaN | NaN |
| **Coupler** | None or Unspecified | None or Unspecified | None or Unspecified | None or Unspecified | None or Unspecified |
| **Coupler_System** | NaN | NaN | None or Unspecified | NaN | None or Unspecified |
| **Grouser_Tracks** | NaN | NaN | None or Unspecified | NaN | None or Unspecified |
| **Hydraulics_Flow** | NaN | NaN | Standard | NaN | Standard |
| **Track_Type** | NaN | NaN | NaN | NaN | NaN |
| **Undercarriage_Pad_Width** | NaN | NaN | NaN | NaN | NaN |
| **Stick_Length** | NaN | NaN | NaN | NaN | NaN |
| **Thumb** | NaN | NaN | NaN | NaN | NaN |
| **Pattern_Changer** | NaN | NaN | NaN | NaN | NaN |
| **Grouser_Type** | NaN | NaN | NaN | NaN | NaN |
| **Backhoe_Mounting** | NaN | NaN | NaN | NaN | NaN |
| **Blade_Type** | NaN | NaN | NaN | NaN | NaN |
| **Travel_Controls** | NaN | NaN | NaN | NaN | NaN |
| **Differential_Type** | Standard | Standard | NaN | NaN | NaN |
| **Steering_Controls** | Conventional | Conventional | NaN | NaN | NaN |

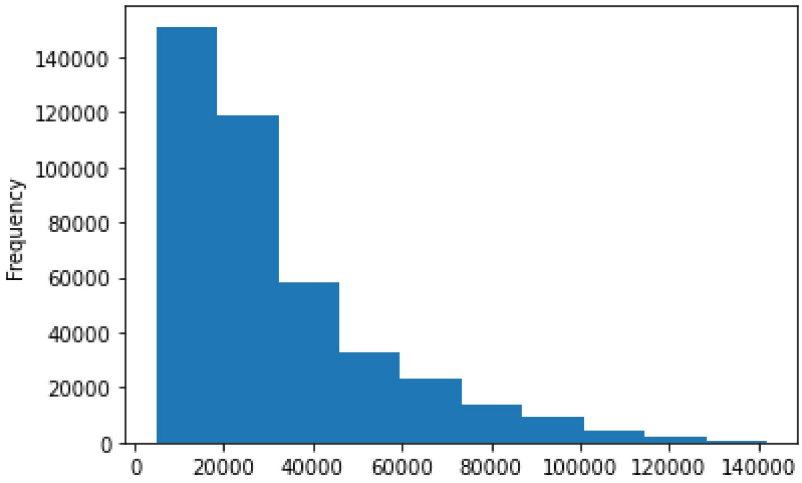
In [ ]:

```
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
```

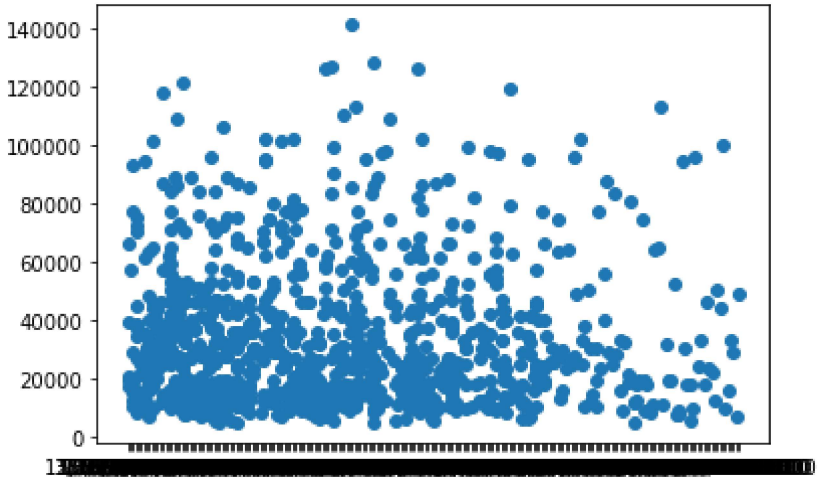
In [ ]:

```
df.SalePrice.plot.hist();
```




In [ ]:

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




we can't see clearly so it's better to use datetime

it's a good idea to make sure any date data is the format of a datetime object

In [ ]:

```
df = pd.read_csv("data/bluebook-for-bulldozers/TrainAndValid.csv",
                 low_memory=False,
                 parse_dates=["saledate"])
```

In [ ]:

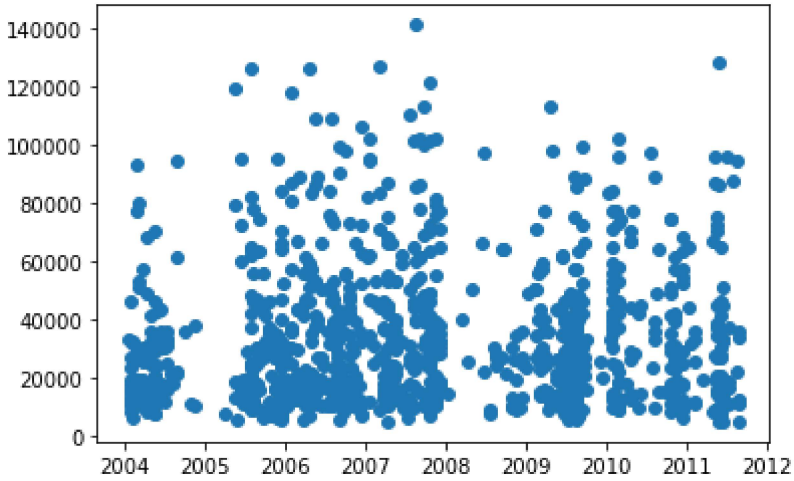
```
df.saledate.dtype
```

Out[ ]:

```
dtype('<M8[ns]')
```

In [ ]:

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




In [ ]:

```
df.saledate.head(10)
```

Out[ ]:

```
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
Name: saledate, dtype: datetime64[ns]
```

we can see that the data is not ordered based on date, so we'll do so

## Sort DataFrame by saledate

In [ ]:

```
# Sort DataFrame in date order
df.sort_values(by=["saledate"], inplace=True, ascending=True)
df.saledate.head(10)
```

Out[ ]:

```
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
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 [ ]:

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

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 of the `saledate` column.

In [ ]:
```
# 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)
```

In [ ]:
```
df_tmp.head().T[-5:]
```

Out[ ]:

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

Out[ ]:
```
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
```

In [ ]:

```
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 [ ]:

```python
# Check for missing values
df_tmp.isna().sum().sort_values(ascending=False)
```

Out[ ]:

```
Blade_Width                 386715
Engine_Horsepower           386715
Tip_Control                 386715
Pushblock                   386715
Blade_Extension             386715
Enclosure_Type              386715
Scarifier                   386704
Hydraulics_Flow             367823
Grouser_Tracks              367823
Coupler_System              367724
fiModelSeries               354031
Steering_Controls           341176
Differential_Type           341134
UsageBand                   339028
fiModelDescriptor           337882
Backhoe_Mounting            331986
Stick                       331602
Turbocharged                331602
Pad_Type                    331602
Blade_Type                  330823
Travel_Controls             330821
Tire_Size                   315060
Track_Type                  310505
Grouser_Type                310505
Stick_Length                310437
Pattern_Changer             310437
Thumb                       310366
Undercarriage_Pad_Width     309782
Ripper                      305753
Drive_System                305611
```

```
MachineHoursCurrentMeter    265194
Ride_Control                259970
Transmission                224691
ProductSize                 216605
Forks                       214983
Coupler                     192019
fiSecondaryDesc             140727
Hydraulics                   82565
auctioneerID                 20136
Enclosure                      334
saleYear                         0
saleMonth                        0
saleDay                          0
saleDayofweek                    0
SalesID                          0
SalePrice                        0
ProductGroupDesc                 0
ProductGroup                     0
state                            0
fiProductClassDesc               0
fiBaseModel                      0
fiModelDesc                      0
YearMade                         0
datasource                       0
ModelID                          0
MachineID                        0
saleDayofyear                    0
dtype: int64
```

## Encoding categorical features

as we have large data we'll use pandas types API which allows us to interact and manipulate the types of data.

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

In [ ]:

```
# Select the object (string) columns
mask1 = df_tmp.dtypes == object
categorical_cols = df_tmp.columns[mask1]
categorical_cols
```

Out[ ]:

```
Index(['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'],
      dtype='object')
```

In [ ]:

```
for label, content in df_tmp[categorical_cols].items():
    df_tmp[label] = content.astype("category").cat.as_ordered()
```

In [ ]:

```
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 [ ]:
```
# Let's see example of one of the category features
df_tmp.UsageBand.cat.categories
```

Out[ ]:
```
Index(['High', 'Low', 'Medium'], dtype='object')
```

In [ ]:
```
df_tmp.UsageBand.cat.codes
```

Out[ ]:
```
205615   -1
274835   -1
141296   -1
212552   -1
62755    -1
         ..
410879   -1
412476   -1
411927   -1
407124   -1
409203   -1
Length: 412698, dtype: int8
```

`-1` means that it's a missing value

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

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

Out[ ]:

| | SalesID | SalePrice | MachineID | ModelID | datasource | auctioneerID | YearMade | MachineHou |
|---|---|---|---|---|---|---|---|---|
| **205615** | 1646770 | 9500.0 | 1126363 | 8434 | 132 | 18.0 | 1974 | |
| **274835** | 1821514 | 14000.0 | 1194089 | 10150 | 132 | 99.0 | 1980 | |
| **141296** | 1505138 | 50000.0 | 1473654 | 4139 | 132 | 99.0 | 1978 | |
| **212552** | 1671174 | 16000.0 | 1327630 | 8591 | 132 | 99.0 | 1980 | |
| **62755** | 1329056 | 22000.0 | 1336053 | 4089 | 132 | 99.0 | 1984 | |

5 rows × 57 columns

## Save Processed Data

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

# Fill missing values

We know two things:

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

## Filling numerical values first

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

In [ ]:
```
numerical_columns = df_tmp.columns[df_tmp.dtypes!= 'category']
numerical_columns
```

Out[ ]:
```
Index(['SalesID', 'SalePrice', 'MachineID', 'ModelID', 'datasource',
       'auctioneerID', 'YearMade', 'MachineHoursCurrentMeter', 'saleYear',
       'saleMonth', 'saleDay', 'saleDayofweek', 'saleDayofyear'],
      dtype='object')
```

In [ ]:
```
#or
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 [ ]:
```
df_tmp[numerical_columns].isna().sum()
```

Out[ ]:
```
SalesID                         0
SalePrice                       0
MachineID                       0
ModelID                         0
datasource                      0
auctioneerID                20136
YearMade                        0
MachineHoursCurrentMeter    265194
saleYear                        0
saleMonth                       0
saleDay                         0
saleDayofweek                   0
saleDayofyear                   0
dtype: int64
```

In [ ]:
```
for label, content in df_tmp[numerical_columns].items():
    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 th
            df_tmp[label] = content.fillna(content.median())
```

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

Out[ ]:

| | SalesID | SalePrice | MachineID | ModelID | datasource | auctioneerID | YearMade | MachineHou |
|---|---|---|---|---|---|---|---|---|

| | SalesID | SalePrice | MachineID | ModelID | datasource | auctioneerID | YearMade | MachineHou |
|---|---|---|---|---|---|---|---|---|
| **205615** | 1646770 | 9500.0 | 1126363 | 8434 | 132 | 18.0 | 1974 | |
| **274835** | 1821514 | 14000.0 | 1194089 | 10150 | 132 | 99.0 | 1980 | |
| **141296** | 1505138 | 50000.0 | 1473654 | 4139 | 132 | 99.0 | 1978 | |
| **212552** | 1671174 | 16000.0 | 1327630 | 8591 | 132 | 99.0 | 1980 | |
| **62755** | 1329056 | 22000.0 | 1336053 | 4089 | 132 | 99.0 | 1984 | |

5 rows × 59 columns

In [ ]:

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

Out[ ]:

```
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 [ ]:

```
categorical_cols
```

Out[ ]:

```
Index(['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'],
      dtype='object')
```

In [ ]:

```
# Turn categorical variables into numbers
for label, content in df_tmp[categorical_cols].items():
    # 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 [ ]:

```
print(f'we now have: {len(df_tmp.columns)} columns')
```

```
we now have: 103 columns
```

In [ ]:

```
#check if there is any missing values
df_tmp.isna().sum().all()
```

Out[ ]: False

# Modeling

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

we'll use RandomForestRegressor.(base_model)

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [ ]:
```
%%time
# Instantiate model
model = RandomForestRegressor(n_jobs=-1)

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

```
Wall time: 4min 57s
```

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

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

Out[ ]:
```
0.9876014881702748
```

## Splitting data into train/valid sets

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

as we're 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 [ ]:
```
# 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[ ]:
```
(11573, 401125)
```

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

## Building an evaluation function

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

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

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

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

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

Making calculations on ~400,000 rows may take a while...

To save time, Let's alter the number of samples each `n_estimator` in the `RandomForestRegressor` see's using the `max_samples` parameter.

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

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

```
Wall time: 12.3 s
RandomForestRegressor(max_samples=10000, n_jobs=-1)
```

Out[ ]:

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

Out[ ]:
```
{'Training MAE': 5567.416010021816,
 'Valid MAE': 7148.687202972435,
 'Training RMSLE': 0.25779205377702047,
 'Valid RMSLE': 0.29352479036732365,
 'Training R^2': 0.8602438962964619,
 'Valid R^2': 0.8334095654776342}
```

## 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 [ ]:
```
from sklearn.model_selection import RandomizedSearchCV

rf_grid = {"n_estimators":np.arange(10, 100, 10),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 20, 2),
           "min_samples_leaf": np.arange(1, 20, 2),
           "max_features": [0.5, 1, "sqrt", "auto"],
           "max_samples": [10000]}

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

In [ ]:
```
%%time
rs_model.fit(X_train, y_train)
```

Out[ ]:
```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Wall time: 8min 33s
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, 12, 14, 16, 18]),
                                        'n_estimators': array([10, 20, 30, 40, 50, 6
0, 70, 80, 90])},
                   verbose=True)
```

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

Out[ ]:
```
{'n_estimators': 30,
 'min_samples_split': 2,
 'min_samples_leaf': 17,
 'max_samples': 10000,
```

```
 'max_features': 'auto',
 'max_depth': 10}
```

In [ ]:

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

Out[ ]:

```
{'Training MAE': 6810.890533745758,
 'Valid MAE': 8255.726486014262,
 'Training RMSLE': 0.3028907324206923,
 'Valid RMSLE': 0.3290276112473395,
 'Training R^2': 0.7968353926972964,
 'Valid R^2': 0.7746272883200173}
```

we found no improvements to the bese_model so we will tune the parameters with `n_iter = 100`

In [ ]:

```
rf_grid_2 = {"n_estimators":np.arange(10, 100, 10),
            "max_depth": [None, 3, 5],
            "min_samples_split": np.arange(2, 20, 2),
            "min_samples_leaf": np.arange(1, 10, 2),
            "max_features": [0.5, "auto"],
            "max_samples": [10000]}

rs_model_2 = RandomizedSearchCV(RandomForestRegressor(),
                                param_distributions=rf_grid_2,
                                n_iter=100,
                                cv=5,
                                verbose=True)
```

In [ ]:

```
%%time
rs_model_2.fit(X_train, y_train)
```

Out[ ]:

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Wall time: 1h 13min 7s
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=100,
                   param_distributions={'max_depth': [None, 3, 5],
                                        'max_features': [0.5, 'auto'],
                                        'max_samples': [10000],
                                        'min_samples_leaf': array([1, 3, 5, 7, 9]),
                                        'min_samples_split': array([ 2,  4,  6,  8,
10, 12, 14, 16, 18]),
                                        'n_estimators': array([10, 20, 30, 40, 50, 6
0, 70, 80, 90])},
                   verbose=True)
```

In [ ]:

```
# Find the best parameters from the RandomizedSearch
rs_model_2.best_params_
```

Out[ ]:

```
{'n_estimators': 60,
 'min_samples_split': 4,
 'min_samples_leaf': 1,
 'max_samples': 10000,
 'max_features': 'auto',
 'max_depth': None}
```

In [ ]:

```
# Evaluate the RandomizedSearch model
show_scores(rs_model_2)
```

Out[ ]:

```
{'Training MAE': 5613.149214424711,
 'Valid MAE': 7223.159360268633,
```

```
 'Training RMSLE': 0.25931690066020846,
 'Valid RMSLE': 0.29399781229016125,
 'Training R^2': 0.8580031303499047,
 'Valid R^2': 0.8294632862397979}
```

In [ ]:

```
import pickle
# Save an existing model to file
pickle.dump((rs_model_2), open("./saved_model/RF_model_1.pkl", "wb"))
```

we w'll use the parameters from RandomizedSearchCV and use it to fit the whole training data

In [ ]:

```
%%time
# Most ideal hyperparameters
ideal_model = RandomForestRegressor(n_estimators=60,
                                    min_samples_leaf=1,
                                    min_samples_split=4,
                                    max_features='auto',
                                    n_jobs=-1,
                                    max_samples=None)

ideal_model.fit(X_train, y_train)
```

```
Wall time: 2min 55s
```

Out[ ]:

```
RandomForestRegressor(min_samples_split=4, n_estimators=60, n_jobs=-1)
```

In [ ]:

```
show_scores(ideal_model)
```

Out[ ]:

```
{'Training MAE': 1769.9029223634943,
 'Valid MAE': 6138.284271974273,
 'Training RMSLE': 0.09446287944652551,
 'Valid RMSLE': 0.2553797471343369,
 'Training R^2': 0.9837841228642226,
 'Valid R^2': 0.8716694904202225}
```

In [ ]:

```
# saving the ideal model
pickle.dump((ideal_model), open("./saved_model/ideal_model_1.pkl", "wb"))
```

# Feature Importance

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?

Doing this for our `RandomForestRegressor` leads us to find the `feature_importances_` attribute.

In [ ]:

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

Out[ ]:

```
array([4.96082753e-02, 1.61159456e-02, 3.26864183e-02, 3.62418896e-04,
       4.29095732e-03, 2.62196548e-01, 3.10656468e-03, 1.05088871e-03,
       1.98516908e-02, 7.67434074e-02, 5.24853576e-02, 4.29844589e-03,
```

```
       3.33736752e-03, 2.28514673e-01, 3.81799892e-02, 8.61902319e-03,
       1.74188651e-04, 1.68110100e-04, 1.64821230e-03, 9.83925513e-03,
       3.95673229e-04, 3.45323714e-05, 5.27610307e-04, 1.66243369e-04,
       1.51601842e-03, 2.43651338e-05, 5.63891479e-04, 1.05490131e-02,
       4.15575905e-04, 4.68409989e-05, 6.36311073e-03, 1.99915053e-03,
       2.21275936e-03, 1.18758248e-03, 2.22471341e-04, 1.68781395e-03,
       9.30921422e-04, 1.56060351e-02, 7.29864184e-06, 4.26378464e-06,
       1.81877528e-04, 1.06000403e-03, 1.25391484e-03, 7.28497112e-04,
       2.36085755e-04, 4.47308214e-04, 2.45208422e-04, 1.65377411e-03,
       9.58638203e-04, 1.74939520e-04, 1.76213785e-04, 7.61584689e-02,
       3.02565984e-03, 8.88521966e-03, 4.38001955e-03, 1.44110663e-02,
       2.72195789e-04, 4.52571428e-04, 2.14708704e-04, 0.00000000e+00,
       0.00000000e+00, 8.35753467e-04, 5.36310724e-04, 8.45145384e-04,
       2.38650956e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 7.61665535e-06, 6.04339166e-06, 6.70476816e-05,
       2.68606913e-06, 6.51124759e-05, 2.38066265e-06, 9.21844308e-05,
       3.79418208e-06, 5.08294663e-04, 2.61903995e-04, 5.23917195e-04,
       4.11635479e-05, 2.35373835e-03, 7.00352222e-06, 5.93618618e-05,
       2.82622510e-03, 5.36695638e-06, 4.34810084e-04, 1.41256231e-04,
       1.45266480e-02, 2.27310380e-06, 3.72312674e-06, 3.06264847e-05,
       3.54436432e-05, 3.34782798e-05, 2.85339730e-05, 2.76420481e-05,
       2.24693657e-05, 2.30486445e-04, 1.92428105e-04, 9.00572367e-05,
       5.24136713e-05, 5.48688676e-05])
```

In [ ]:

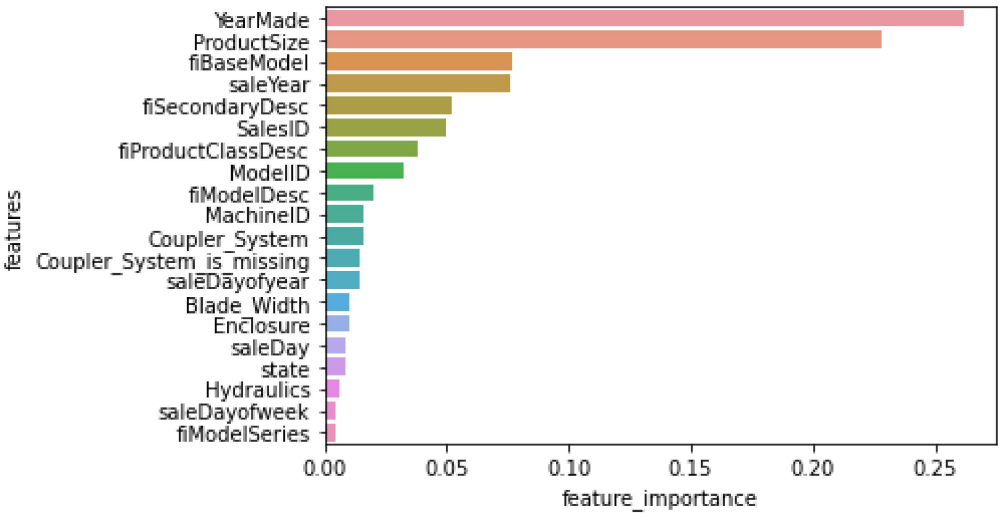
```
import seaborn as sns

# 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 [ ]:

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




# Make predictions on test data

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

In [ ]:
```
df_test = pd.read_csv("data/bluebook-for-bulldozers/Test.csv",
                      parse_dates=["saledate"])
df_test.head()
```

Out[ ]:

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

5 rows × 52 columns

## Preprocessing the test data

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.

In [ ]:
```
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
```

In [ ]:
```
df_test = preprocess_data(df_test)
```

```
df_test.head()
```

Out[ ]:

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

5 rows × 101 columns

In [ ]:

```
X_train.shape,df_test.shape
```

Out[ ]:

```
((401125, 102), (12457, 101))
```

We've found that 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 [ ]:

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

Out[ ]:

```
{'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, then we reorder the test_data's columns

In [ ]:

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

Out[ ]:

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

5 rows × 102 columns

In [ ]:

```
columns_list = X_train.columns.to_list()
columns_list.index('auctioneerID_is_missing'), df_test.columns.to_list().index('auct
```

Out[ ]:
```
(56, 101)
```

In [ ]:
```
df_test = df_test.reindex(columns = columns_list)
```

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

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

Out[ ]:
```
array([21406.09126984, 19001.45502646, 47950.69444444, ...,
       11807.61904762, 18511.66666667, 29803.18783069])
```

In [ ]:
```
# 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[ ]:

| | SalesID | SalePrice |
|---|---|---|
| **0** | 1227829 | 21406.091270 |
| **1** | 1227844 | 19001.455026 |
| **2** | 1227847 | 47950.694444 |
| **3** | 1227848 | 87623.670635 |
| **4** | 1227863 | 52444.236111 |
| **...** | ... | ... |
| **12452** | 6643171 | 46354.583333 |
| **12453** | 6643173 | 10824.781746 |
| **12454** | 6643184 | 11807.619048 |
| **12455** | 6643186 | 18511.666667 |
| **12456** | 6643196 | 29803.187831 |

12457 rows × 2 columns

In [ ]:
```
# Export to csv...
df_preds.to_csv("data/predictions.csv",index=False)
```