Table of Contents

Load Data

In [2]:
```
%matplotlib inline
from featimp import *
import shap
import warnings
warnings.filterwarnings("ignore")
```

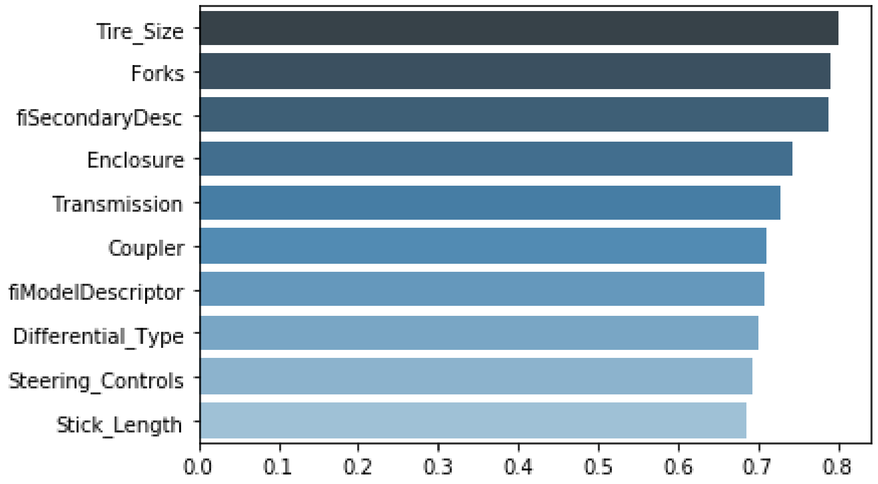
In [3]:
```
# data loading and cleaning
X_origin, y_origin = clean_bulldozer(path="data/bulldozer-train.feather"
)
# sample for a smaller dataset
X = X_origin.sample(frac=0.1)
y = y_origin[X.index]
print('X size : {}'.format(X.shape))
# delete bool and object type, so that we can calculate spearman matrix
X_num = drop_bool_object(X)
print('X_num size : {}'.format(X_num.shape))
```

```
X size : (38912, 53)
X_num size : (38912, 49)
```

PCA Analysis

- PCA Analysis is a direct way to get the feature importance
- PCA gives us the rank of features, not the exact value; while we can scale the importance values from 0 to 0.8 to present it.
- The basic idea is to find the most ,important direction(like pc1), and the most important features connected to it.
  - Firstly, we get explained_variance*ratio* of each pc direction; that's the weight for pc1, pc2...
  - Then, from the components_, we can see the features linked with each direction in pc1, pc2...

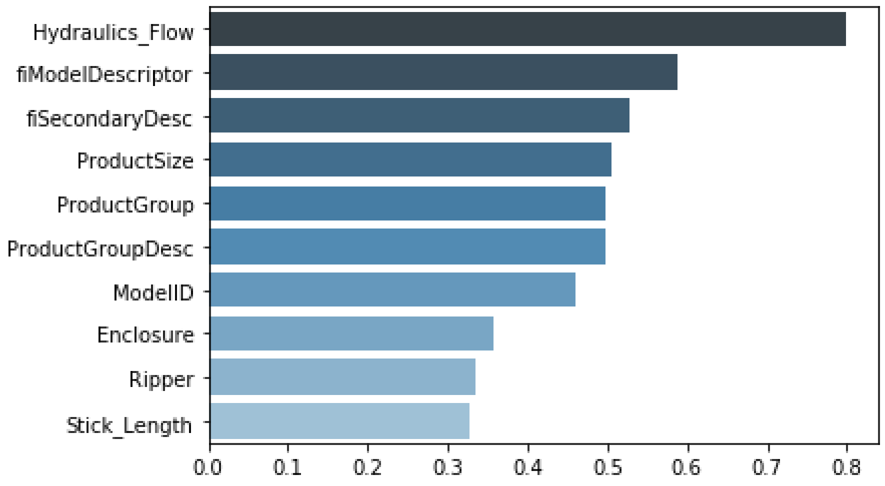
In [4]:
```
I_pca = pca_importance(X_num)
plot_imp(scale(abs(I_pca)), X_num,top_n=10)
```



Spearman Correlation : a direct way to get feature importance

- The simple method is to use spearman correlation directly as importance
- Two points to be considered here :
    - We should sort importance with abs(I) instead of I, since negative value can also show strong correlation.
    - Problem from codependant columns. To solve this problem, we use mRMR.

In [5]:
```
rho, pval = spearmanr(X_num,y)
# rho n*n, the last row stands for other's relationship with y
I_spearman = rho[-1][:-1]
plot_imp(scale(abs(I_spearman)), X_num,top_n=10)
```

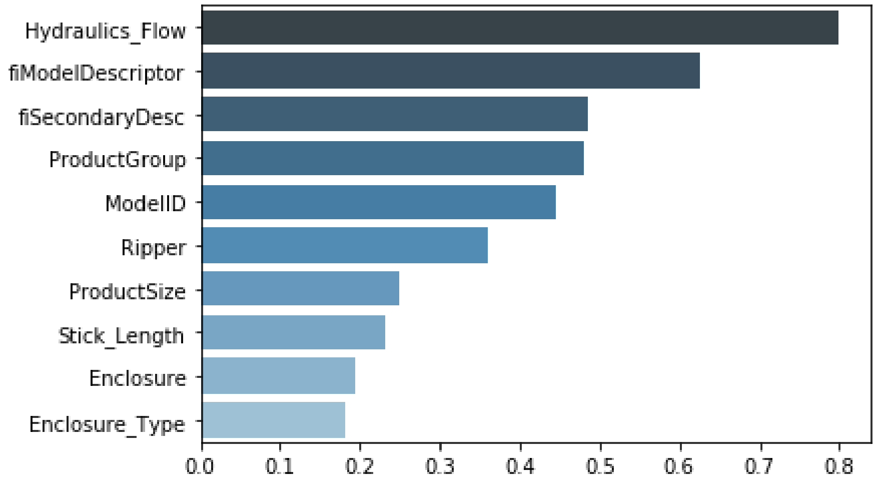


mRMR: deal with codependant features

Implement mRMR

- Method: minimal-redundancy-maximal-relevance.
- Steps:
    - put the 1st element that is highly related with y into selected_feature_space S
    - from the rest features, everytime we put a new element k into S, by making sure its "I(Xk, y) - mean(I(Xk, Xj) for j in selected_feature_space S)" is highest
    - until we pick enough num_features in total S

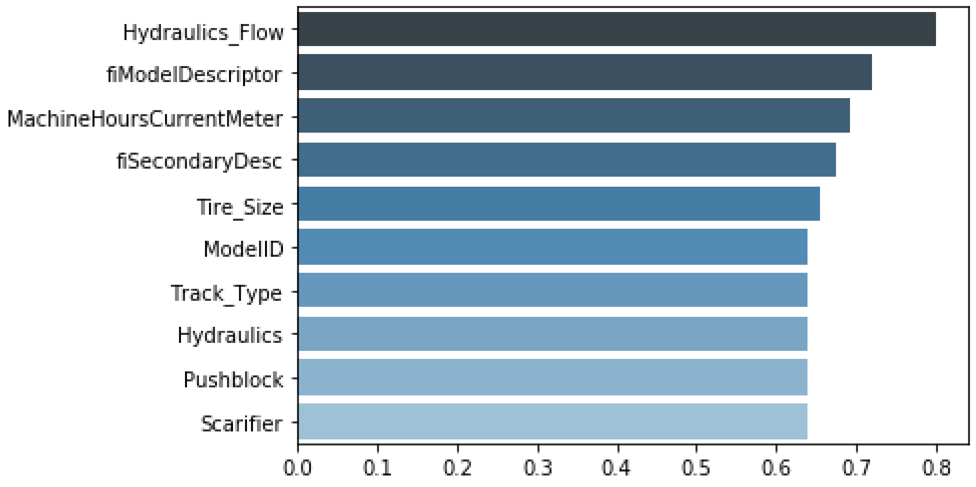
In [6]:
```
features_mRMR = mRMR_S(X_num,y,num_features=10)
I_mRMR = final_S_I(X_num,y, features_mRMR)
I_mRMR = new_I(I_mRMR,features_mRMR,X_num)
plot_imp(scale(I_mRMR), X_num)
```



Use hierarchy to get high-correlated subsets

- The basic idea is : only pick 1 feature from a correlated subset.
- We can use hierarchy to get several subsets firstly, in which the features are highly correlated with each-other
- For each subset, we pick up a single feature hat has highest "I(Xk, y) - mean(I(Xk, Xj) for j in subset S)"

In [7]:
```
feature_groups = feature_hierarchy(X_num, 2)
# depth=2 in hierarchy, result in 14 subsets
k_all, rho_all = k_rho(X_num, y,feature_groups)
I_hierarchy = rho_all
features_hierarchy = k_all
I_hierarchy = new_I(I_hierarchy,features_hierarchy,X_num)
plot_imp(scale(I_hierarchy), X_num, top_n=10)
```



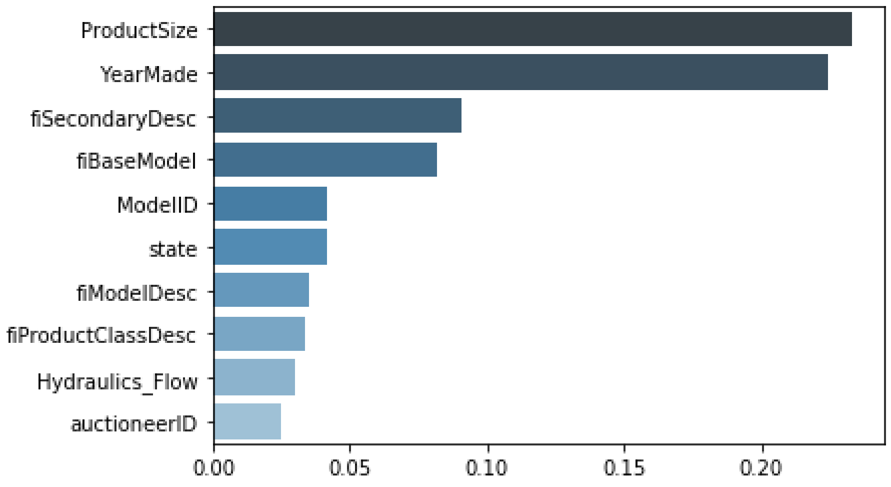
Basic: sklearn's importance

- This is directly from sklearn package, and it runs very quickly.

In [8]:
```
rf = RandomForestRegressor(oob_score = True, n_jobs=-1)
rf.fit(X,y)
```

Out[8]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nod
es=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.
0,
                      n_estimators=100, n_jobs=-1, oob_score=True,
                      random_state=None, verbose=0, warm_start=False)
```

In [9]:
```
I_sklearn = rf.feature_importances_
plot_imp(I_sklearn, X,10)
```

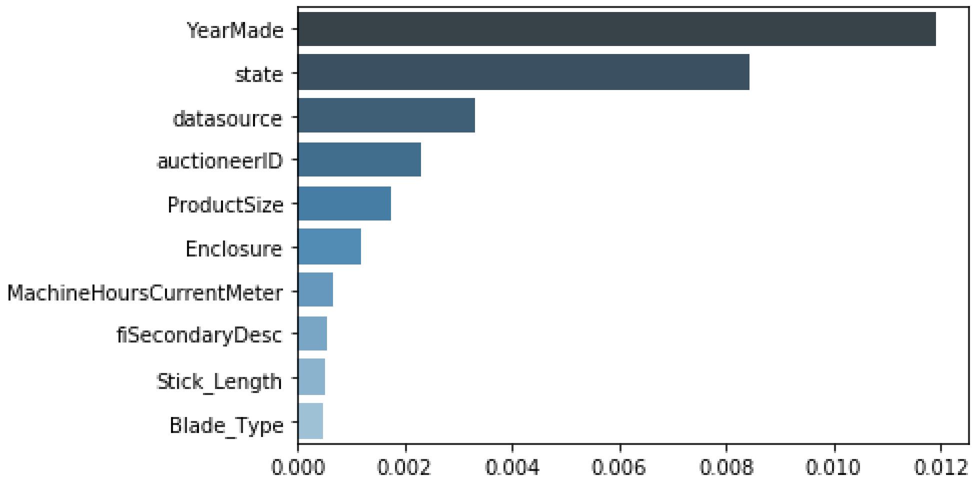


Drop-column importance

Implementation for drop-column, with r2_score as metric for regression

- Drop-column: we drop 1 column at a time, and see how the metric changed.
- Drop-column is very slow, since it train rf model many times
- Drop-column has a potential bias towards correlated predictive variables
- r2_score is only for training data, not as good as oob score.

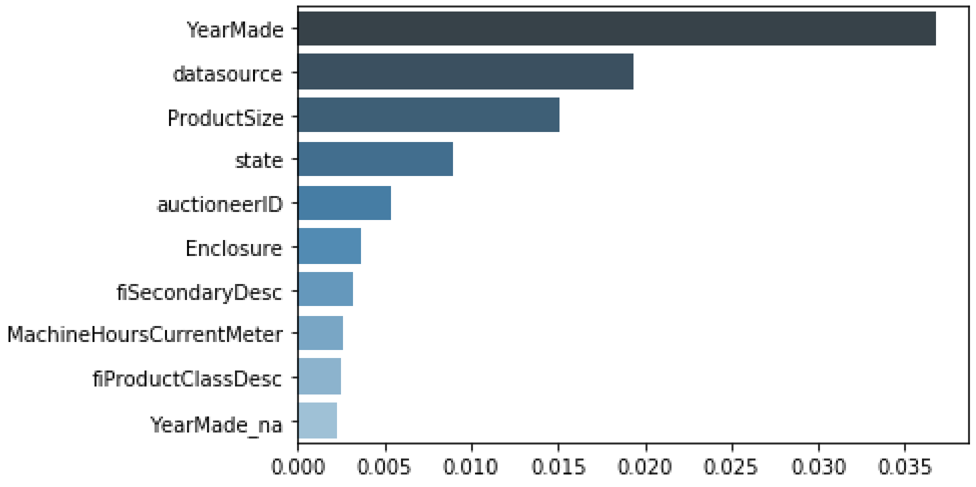
In [10]:
```
I_dropcol_r2 = dropcol_importances(rf, r2_score, X, y)
plot_imp(I_dropcol_r2, X, 10)
```



Rfpimp implementation for drop-column, with oob as metrics

- By using oob score, it's more reasonable than just using r2_score, which is only for training data.
- Citation: [https://github.com/parrt/random-forest-importances (https://github.com/parrt/random-forest-importances)](https://github.com/parrt/random-forest-importances)

In [11]:
```
I_dropcol_oob = dropcol_importances_oob(rf, X, y)
# use out-of-bag samples to estimate the R^2 on unseen data
plot_imp(I_dropcol_oob, X, 10)
```

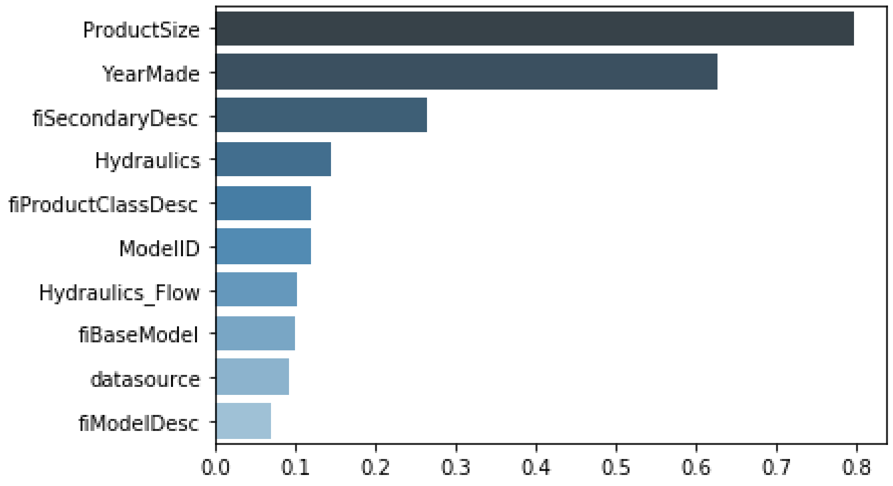


Permutation importance

Implementation for permutation, use r2_score as metric for regression

- Permutation: we shuffle 1 feature at a time, and see how the metric changed.
- Use r2_score, it's quicker than oob_regression_r2_score, but not as accurate as oob_score
- If regressor, use metric = r2_score ; if classifier, use metric = accuracy_score

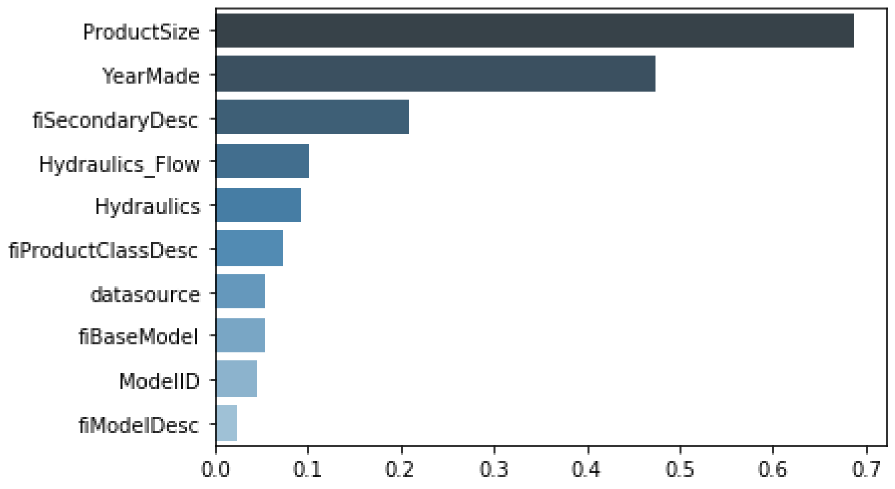
In [12]:
```
I_permutation_r2 = permutation_importances(rf, r2_score, X, y )
plot_imp(I_permutation_r2, X)
```



Rfpimp implementation for permutation, with oob as metrics

- This is the method in rfpimp package
- If regressor, use oob_regression_r2_score; if classifier, use oob_classifier_accuracy
- Citation: https://github.com/parrt/random-forest-importances/blob/master/notebooks/permutation-importances-classifier.ipynb (https://github.com/parrt/random-forest-importances/blob/master/notebooks/permutation-importances-classifier.ipynb)

```
In [13]: I_permutation_oob = permutation_importances_oob(rf, oob_regression_r2_sc
         ore, X, y )
         plot_imp(I_permutation_oob, X)
```



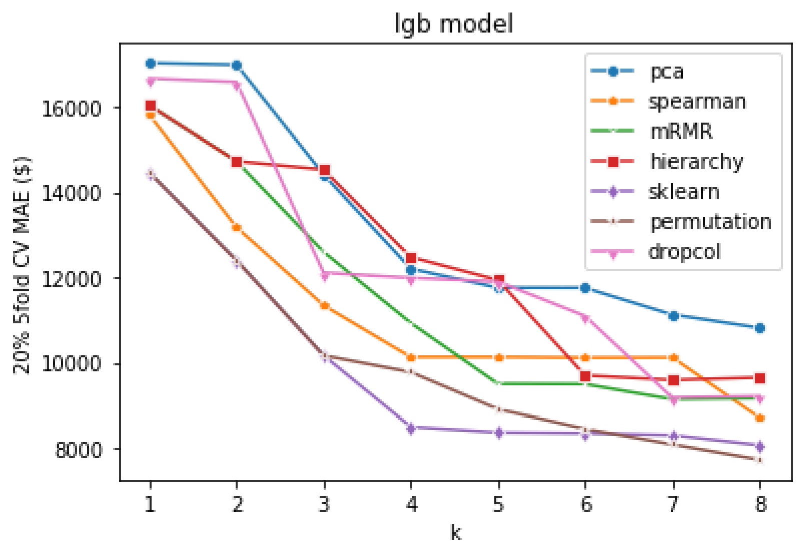
Summarize: comparing Strategies

- Top 1,2,...8 features selected from various methods are used in the training X.
- Implemented light gradient boosting method to train this dataset, calculate the 20% 5fold CV MAE.
- Results as follows:
  - For lgb boosting model, permutation is best.
  - For OLS model, mRMR is best, while teh second-best is still permutation .
  - Permutation is relatively a good method, in terms of its speed and results; it gave result not too far away from shap.
  - We can see that features determined by sklearn rf model exported well for lbg, while it didn't export well for OLS.

Test feature importances with lgb

- permutation method behaves well for lgb

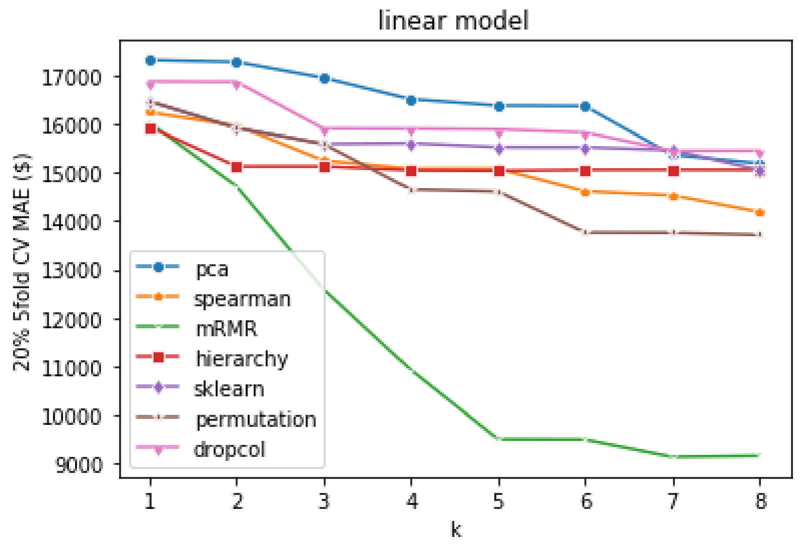
```
In [14]: plot_lgb(8,X,y,I_pca,I_spearman,I_mRMR,I_hierarchy,I_sklearn,I_permutati
         on_oob,I_dropcol_oob)
```



Test feature importances with OLS

- mRMR method behaves well for OLS

```
In [15]: plot_linear(8,X,y,I_pca,I_spearman,I_mRMR,I_hierarchy,I_sklearn,I_permut
         ation_oob,I_dropcol_oob)
```



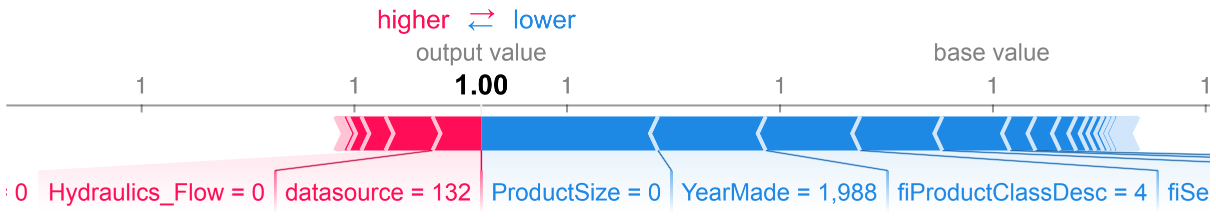
Compare with shap

- Shap gives us complicated feature importance
- Here, shap says that the top2 important features are: productsize and yearmade; the same as our result from permutation.

In [16]:
```
# load JS visualization code to notebook
shap.initjs()
# shap analysis for lgb model
model,validation_err = lgb_model(X,y)
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X, y, tree_limit=100)
```

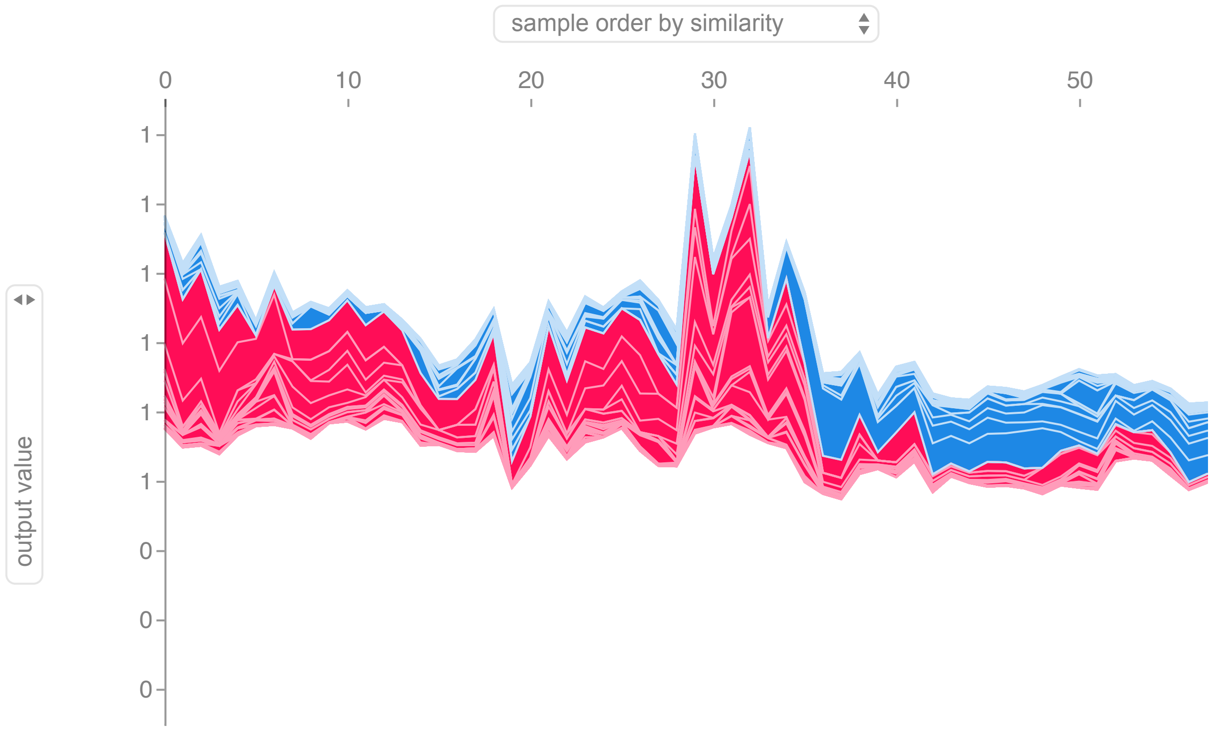
In [17]:
```
# visualize one prediction's explanation
i = 2
shap.force_plot(explainer.expected_value, shap_values[i,:], X.iloc[i,:],
link="logit")
```

Out[17]:

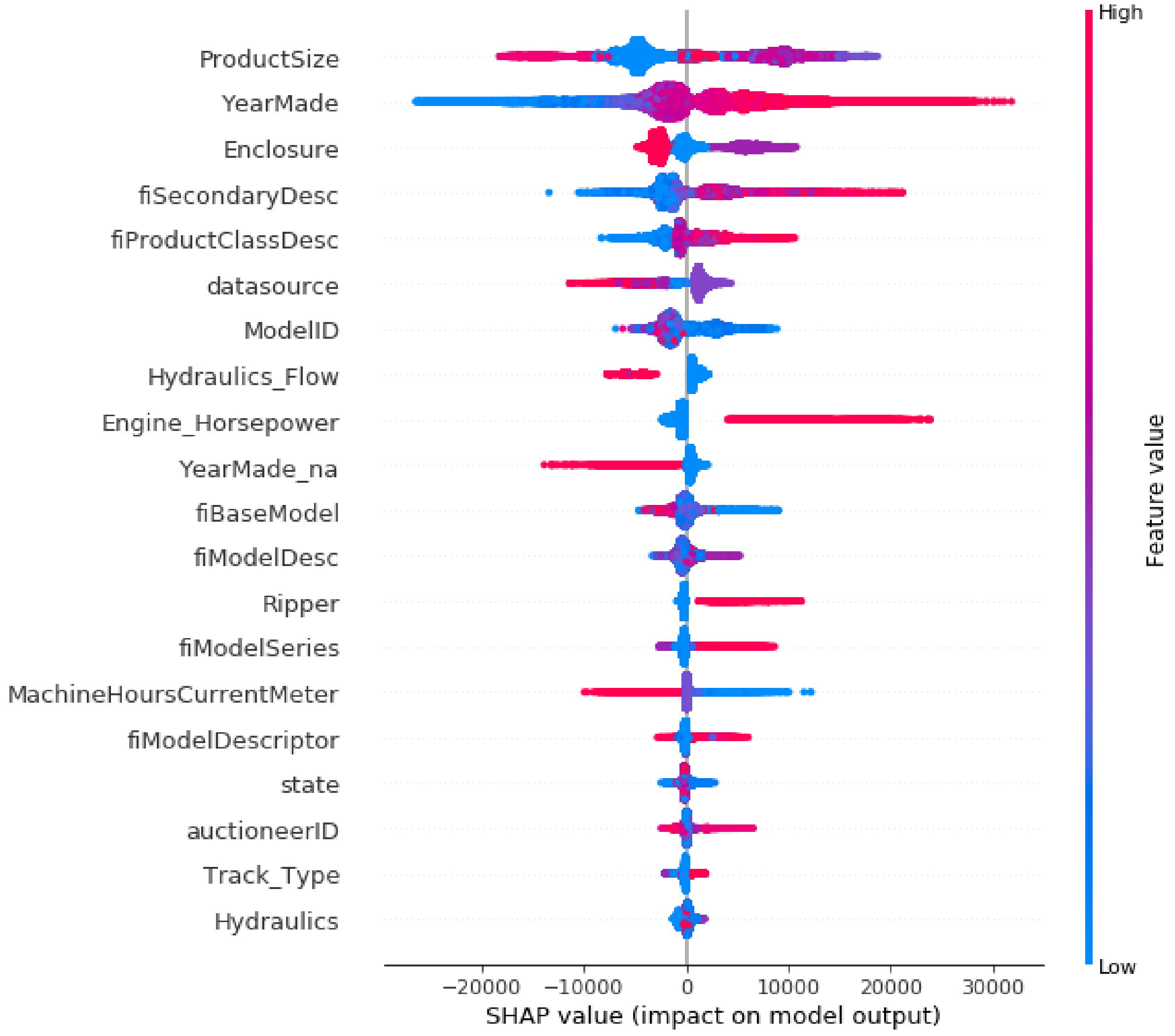


```
In [18]: # visualize the first 100 prediction's explanation
         shap.force_plot(explainer.expected_value, shap_values[:100,:], X.iloc[:1
         00, :], link="logit")
```

Out[18]:



```
In [19]: shap.summary_plot(shap_values, X)
```

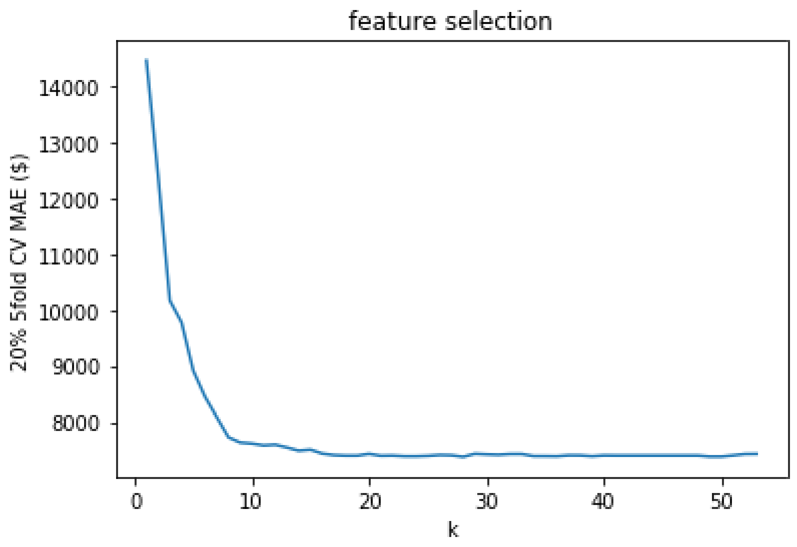


Automatic feature selection algorithm

- I choose 5fold MAE as the validation metric, and used permutation_oob to get the feature importance, since permutation behaves best in the previous analysis.
- I calculated the val_MAE by using only the top 1,2,3... features.
- Here below is how the val metric changed with feature numbers
- We can find from the below graph: the val_MAE stopped to drop when k > 12.
- So, the final chosen features are the top 12 features.

In [20]:
```
lgb_5fold_val_err = mae_I(X,y,sort_I(I_permutation_oob),X.shape[1])
sns.lineplot(np.arange(1,X.shape[1]+1),lgb_5fold_val_err)
plt.ylabel('20% 5fold CV MAE ($)')
plt.xlabel('k')
plt.title('feature selection')
```

Out[20]:
```
Text(0.5, 1.0, 'feature selection')
```



In [21]:
```
print('Automatic selected features are : {}'.format(
    [X.columns[i] for i in [sort_I(I_permutation_oob)[i][0] for i in ran
ge(12)]]))
```

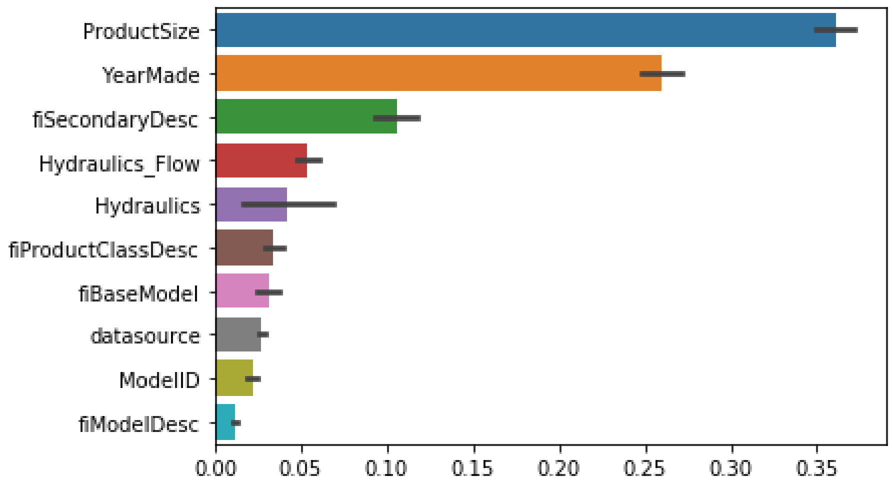
```
Automatic selected features are : ['ProductSize', 'YearMade', 'fiSecond
aryDesc', 'Hydraulics_Flow', 'Hydraulics', 'fiProductClassDesc', 'datas
ource', 'fiBaseModel', 'ModelID', 'fiModelDesc', 'YearMade_na', 'Enclos
ure']
```

Variance and empirical p-values for feature importances

Variance of feature importance

- Get permutation feature importance with boostraped data for 20 times, so that we can analyze the mean and standard deviation for each feature importance value.
- Here below is the feature importance with scale bar(mean-std, mean+std)
- We can see that:
    - the feature importance rank for top3 is quite reliable
    - the rank for the top 4~10 features is not really reliabel, considering the scale bar

```
In [33]: I_20 = I_boostrap_permutation(rf,20,X,y)
         normalized_I_20 = normalize(np.array([i for i in I_20]), axis=1, norm='l
         1')
         sorted_I_20 = sort_mean_I(re_order(normalized_I_20))[:10]
         features = [[X.columns[i]]*20 for i in [item[0] for item in sorted_I_20
         ]]
         df_I_20 = pd.DataFrame({'features':flatten(features),'I':flatten([item[1
         ] for item in sorted_I_20])})
         plot_std(df_I_20)
```

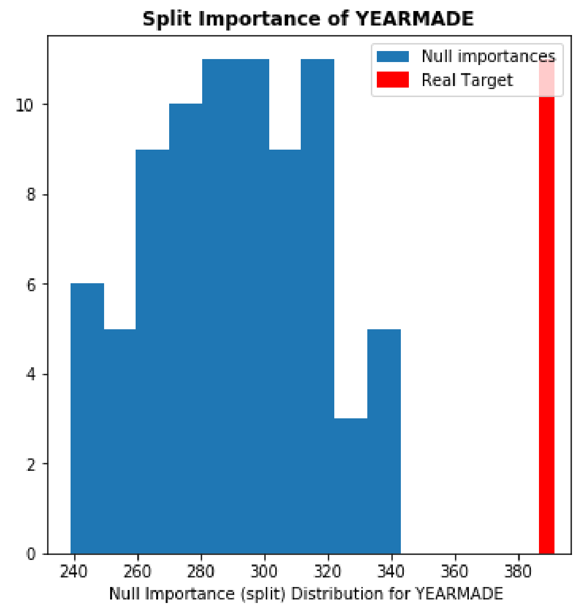


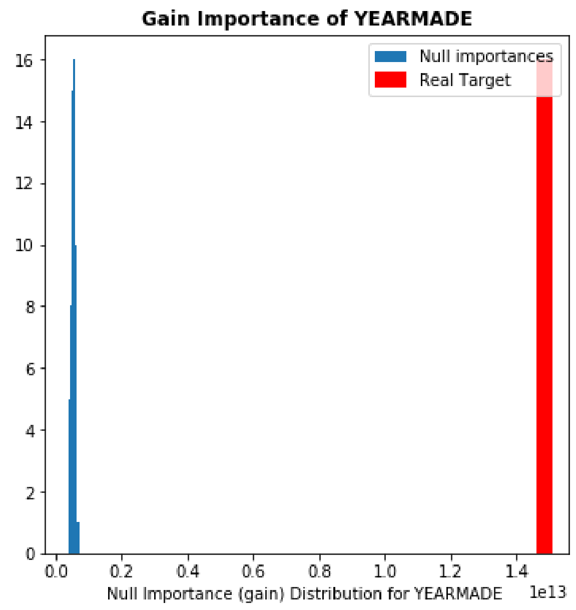
Empirical p-values

- Null hypothesis: this feature is not significant; Alternative hypothesis: this feature is significant.
- Firstly, with current y, get benchmark feature importance (red in below graph)
- Run lgb model 80 times with shuffled y (blue in below graph)
- For each feature, we can see the probability(p-value) that shuffled importance is bigger than benchmark.
- Example with Feature = 'YearMade':
  - in this graph, blue pillars are always at the right side of red pillar.
  - shuffled importance is bigger than benchmark.
  - p-value is big, we can accept the alternative hypothesis.
  - conclusion: Yearmade is important.
- Example with Feature = 'state':
  - in this graph, blue pillars are always at the left side of red pillar.
  - shuffled importance is lower than benchmark.
  - p-value is small, we can accept the null hypothesis.
  - conclusion: state is not important.

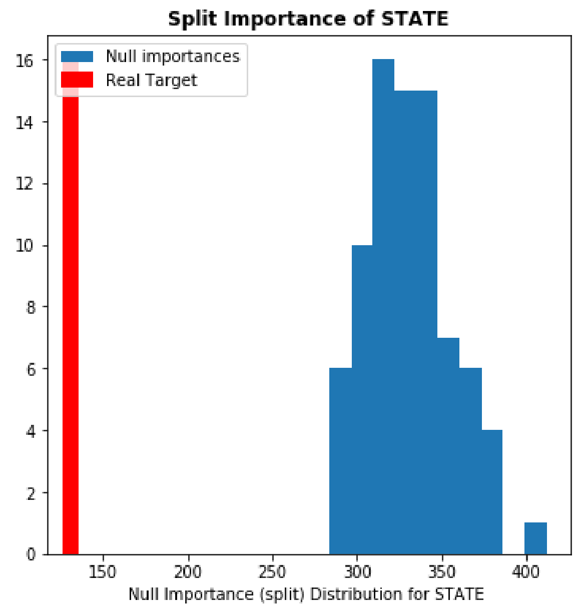
```
In [26]: actual_imp_df = lgb_feature_importances(X,y, shuffle=False)
         null_imp_df = pd.DataFrame()
         for i in range(80):
             null_imp_df = pd.concat([null_imp_df, lgb_feature_importances(X,y, s
         huffle=True)], axis=0)
```

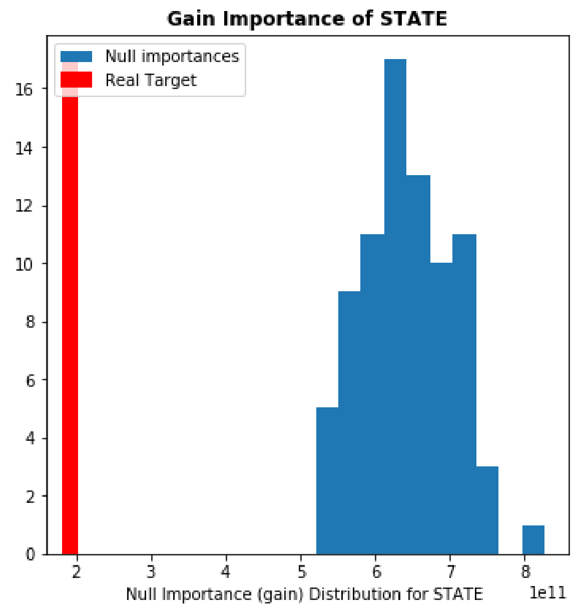
In [27]:
```
display_distributions(actual_imp_df_=actual_imp_df, null_imp_df_=null_im
p_df, feature_='YearMade')
```





In [28]:
```
display_distributions(actual_imp_df_=actual_imp_df, null_imp_df_=null_im
p_df, feature_='state')
```





Summary

- I used the following methods to get feature importance:
    - direct methods: PCA, Spearman correlation, mRMR, hierarchy
        - these methods only give the rank of features, instead of real valid score of feature importance
    - model-based methods: drop-column, permutation
        - permutation did a better work than drop-column
- I compared the feature importance ananlysis results from the above methods:
    - by comparing with shap result
    - by fitting the data with lgb, and comparing the 5CrossValidation MAE when using top 1,2...8 features
    - by fitting the data with OLS, and comparing the 5CrossValidation MAE when using top 1,2...8 features
    - in this way, the automatic feature selection algorithm is implemented
- I also calculated the variance and empirical p-values, to check how reliable is the calculated feature importance.