In [28]:
```python
# Import libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_boston
from rfpimp import *
from sklearn.inspection import permutation_importance

import matplotlib.pyplot as plt
import rfpimp
boston = load_boston()
X = boston.data
y = boston.target
features = boston.feature_names
df = pd.DataFrame(data=X,columns=features)
```

In [29]:
```python
df.head()
```

Out[29]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90

In [30]:
```python
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.33, r
```

In [34]:
```python
X_train
```

Out[34]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
478	10.23300	0.0	18.10	0.0	0.614	6.185	96.7	2.1705	24.0	666.0	20.2	379
26	0.67191	0.0	8.14	0.0	0.538	5.813	90.3	4.6820	4.0	307.0	21.0	376.
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.
492	0.11132	0.0	27.74	0.0	0.609	5.983	83.5	2.1099	4.0	711.0	20.1	396.
108	0.12802	0.0	8.56	0.0	0.520	6.474	97.1	2.4329	5.0	384.0	20.9	395.
...	...	...	...	...	...	...	...	...	...	...	...	
106	0.17120	0.0	8.56	0.0	0.520	5.836	91.9	2.2110	5.0	384.0	20.9	395.
270	0.29916	20.0	6.96	0.0	0.464	5.856	42.1	4.4290	3.0	223.0	18.6	388.
348	0.01501	80.0	2.01	0.0	0.435	6.635	29.7	8.3440	4.0	280.0	17.0	390.
435	11.16040	0.0	18.10	0.0	0.740	6.629	94.6	2.1247	24.0	666.0	20.2	109.
102	0.22876	0.0	8.56	0.0	0.520	6.405	85.4	2.7147	5.0	384.0	20.9	70.

339 rows × 13 columns

In [40]:

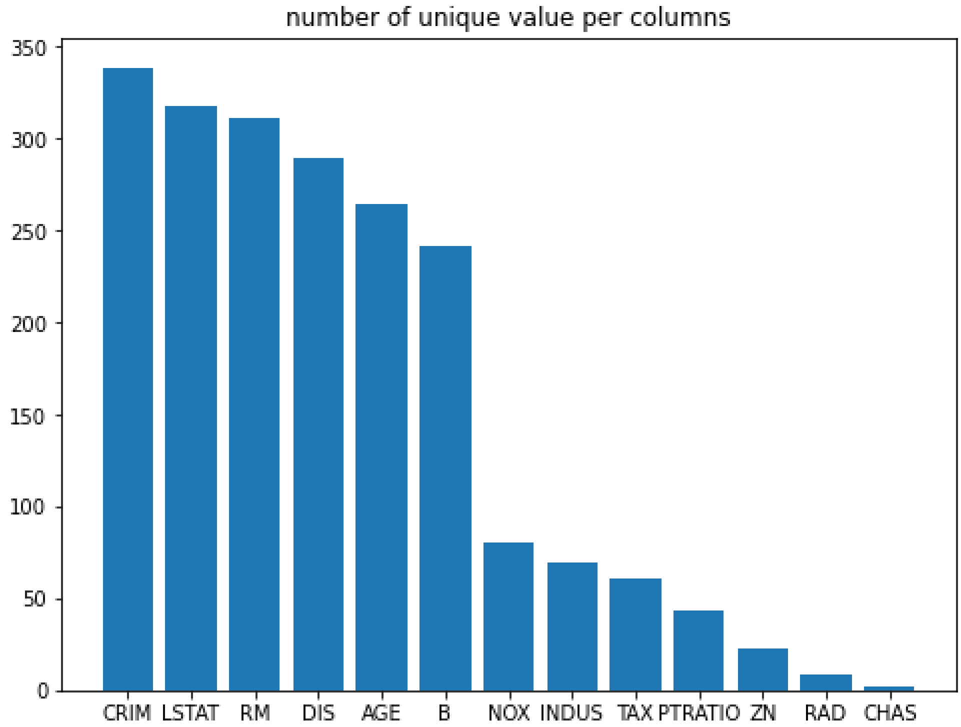
```
X_train.describe()
```

Out[40]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	A
count	339.000000	339.000000	339.000000	339.000000	339.000000	339.000000	339.000
mean	3.351324	11.716814	11.261858	0.076696	0.557498	6.327324	68.940
std	7.689661	22.981007	6.968227	0.266502	0.117683	0.720720	27.951
min	0.009060	0.000000	1.210000	0.000000	0.385000	3.863000	2.900
25%	0.082100	0.000000	5.130000	0.000000	0.448000	5.890000	45.650
50%	0.259150	0.000000	9.900000	0.000000	0.538000	6.229000	78.100
75%	3.397665	20.000000	18.100000	0.000000	0.631000	6.705500	93.900
max	88.976200	95.000000	27.740000	1.000000	0.871000	8.780000	100.000

In [77]:

```
count_list = []
for col in X_train.columns:
    uni_count = len(X_train[col].unique())
    count_list.append(uni_count)

sort_count = np.array(count_list).argsort()[::-1]
fig, ax = plt.subplots(figsize=(8,6))
plt.bar(X_train.columns[sort_count], np.array(count_list)[sort_count])
plt.title('number of unique value per columns')
plt.show()
```



CRIM, LSTAT, RM, DIS has the most unique values

Feature importance using linear coefficients

In [36]:
```
# first let's rescale our data
scaler = StandardScaler()
x_train_sc = scaler.fit_transform(X_train)

lmbda = .1

lm = Lasso(alpha=lmbda, tol=.1)
lm.fit(x_train_sc, y_train)

# extract the coefficients
importance = abs(lm.coef_)

importance.shape
```

Out[36]:
```
(13,)
```

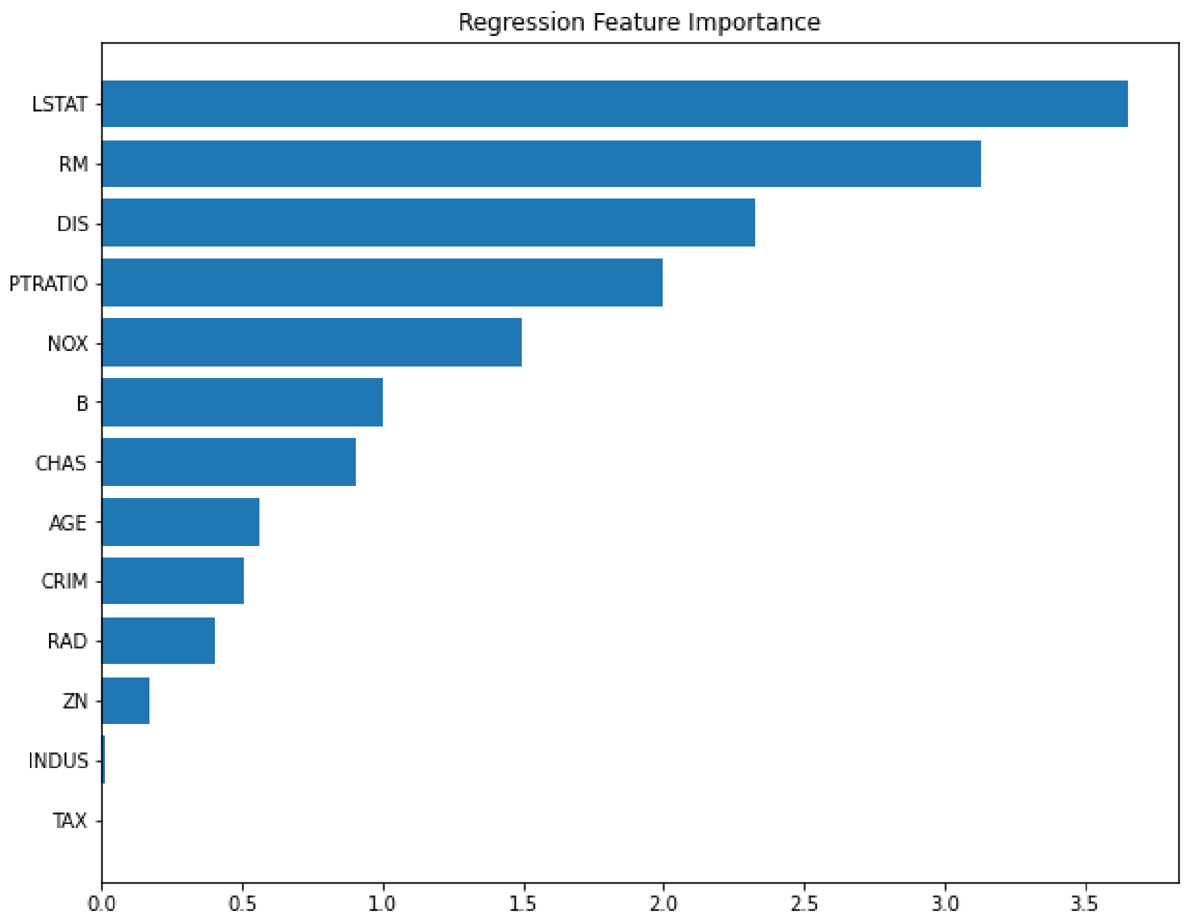
In [37]:
```
feat_sort = importance.argsort()

feat_imp = pd.DataFrame({"features":X_train.columns[feat_sort], "importance"
feat_imp['features'] = pd.Categorical(feat_imp['features'], categories=feat_

feat_imp
```

Out[37]:

	features	importance
0	TAX	0.000000
1	INDUS	0.013135
2	ZN	0.167361
3	RAD	0.403679
4	CRIM	0.505789
5	AGE	0.557549
6	CHAS	0.903945
7	B	1.001507
8	NOX	1.492046
9	PTRATIO	1.999247
10	DIS	2.324220
11	RM	3.133981
12	LSTAT	3.652480

In [38]:

```
# plot the feature importance
fig, ax = plt.subplots(figsize=(10,8))
plt.title("Regression Feature Importance")
plt.barh(feat_imp['features'], feat_imp['importance'])
plt.show()
```

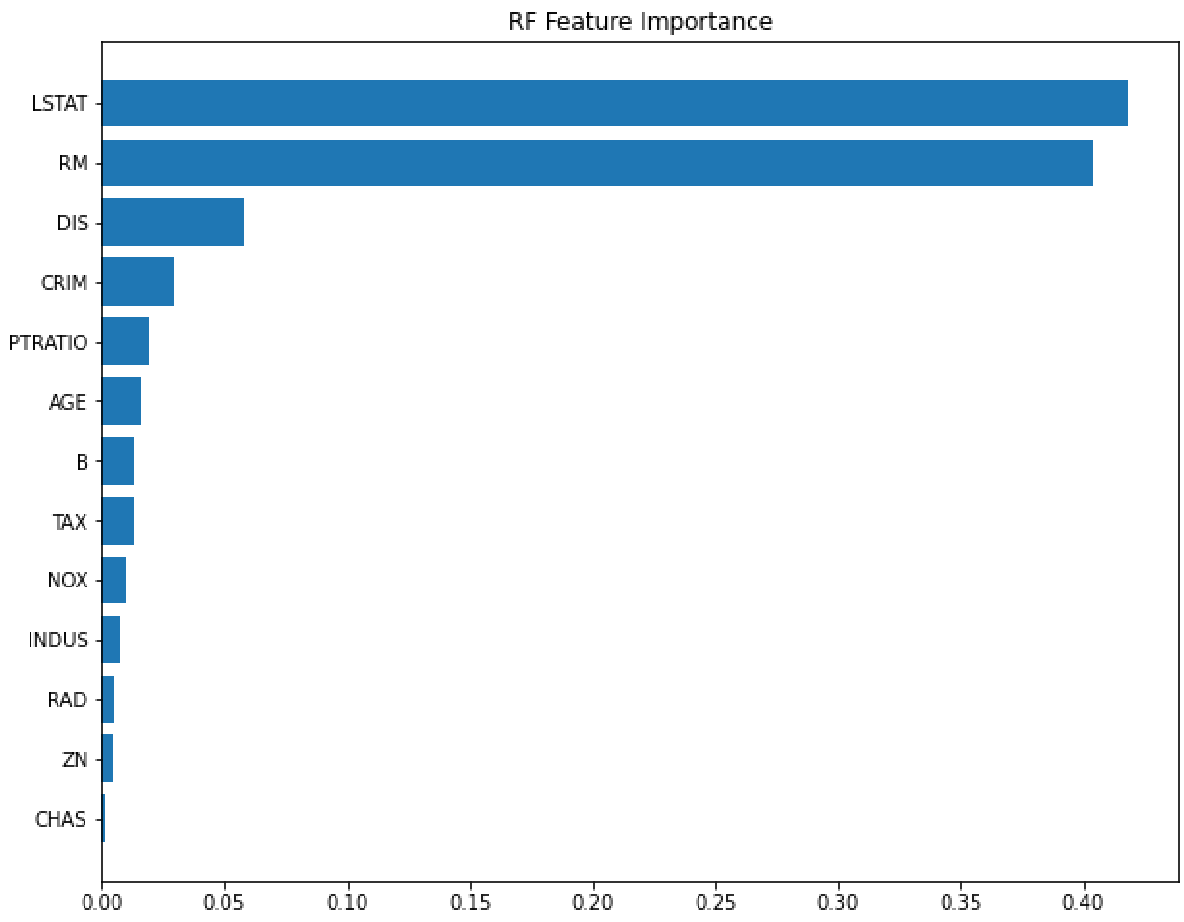


Random Forest feature importance using impurity

In [105…

```
# create a random forest regressor object
rfc = RandomForestRegressor(n_estimators=100, random_state=42)

# train the model on your data
rfc.fit(X_train, y_train)

# get feature importances
importance = rfc.feature_importances_

feat_sort = importance.argsort()
feat_imp = pd.DataFrame({"features":X_train.columns[feat_sort], "importance"
feat_imp['features'] = pd.Categorical(feat_imp['features'], categories=feat_
print(feat_imp)
# plot the feature importance
fig, ax = plt.subplots(figsize=(10,8))
plt.title("RF Feature Importance")
plt.barh(feat_imp['features'], feat_imp['importance'])
plt.show()
```

```
   features  importance
0      CHAS    0.001078
1        ZN    0.004622
2       RAD    0.004951
3     INDUS    0.007564
4       NOX    0.009763
5       TAX    0.012990
6         B    0.013339
7       AGE    0.016033
8   PTRATIO    0.019683
9      CRIM    0.029733
10      DIS    0.058104
11       RM    0.404017
12    LSTAT    0.418126
```



Problem with impurity feature importance

While Gini impurity is a widely used impurity measure, it has some limitations and potential problems:

- Biased towards multi-class classification: Gini impurity performs well in multi-class classification problems as compared to binary classification problems. In binary classification, the impurity measure that works better is entropy. This is because the Gini impurity does not consider the difference between the probabilities of different classes, and hence, it may not work well in situations where the class distribution is not uniform.
- Tendency towards selecting attributes with more distinct values: Gini impurity tends to favor splitting on attributes with many distinct values. This is because such attributes can create splits that lead to smaller subsets of examples, which can lead to a lower impurity measure. However, such attributes may not be the most informative ones, and this can lead to overfitting of the model.
- Sensitivity to class imbalance: Gini impurity is sensitive to class imbalance, meaning that if the class distribution is highly skewed, the impurity measure may not work well. In such cases, other impurity measures, such as information gain or gain ratio, may be more appropriate.
- Local minima: Gini impurity can lead to local minima, which means that the decision tree may not be optimal. This can happen if the algorithm chooses to split on an attribute that leads to a lower impurity measure at the current node but does not lead to the best overall classification accuracy.

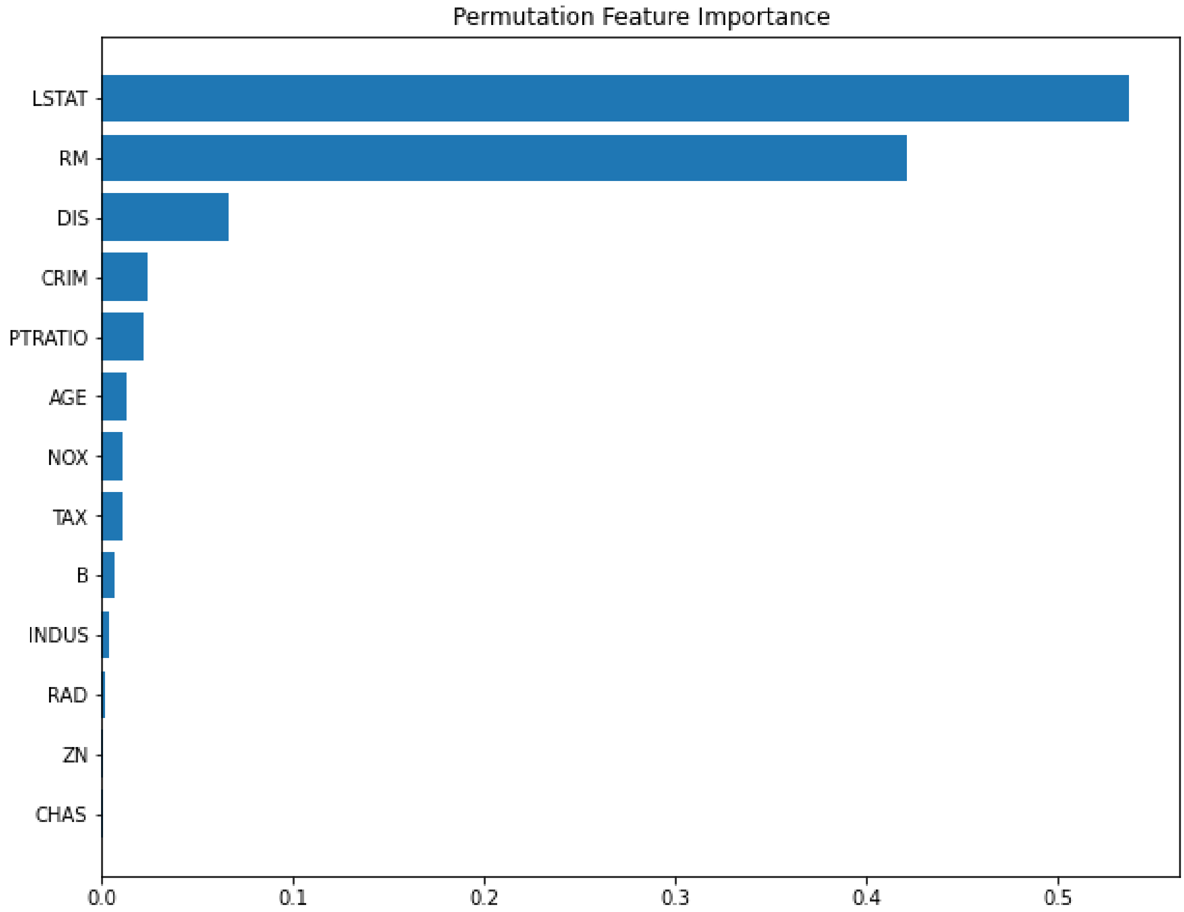
To avoid the above problems, we try to explore other methods of getting feature importance

Feature importance using permutation importance

In [104…

```
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestRegressor

# load data and split into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.33, r

# train a random forest regressor on the training set
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# compute permutation importance on the validation set
result = permutation_importance(rf, X_test, y_test, n_repeats=10, random_sta

# get feature importance scores and their standard deviation
importances = result.importances_mean
std = result.importances_std

# sort feature importance
feat_sort = importances.argsort()
feat_imp = pd.DataFrame({"features":X_train.columns[feat_sort], "importance"
feat_imp['features'] = pd.Categorical(feat_imp['features'], categories=feat_

print(feat_imp)
# plot the feature importance scores with error bars
fig, ax = plt.subplots(figsize=(10,8))
plt.title("Permutation Feature Importance")
plt.barh(feat_imp['features'], feat_imp['importance'])
plt.show()
```

```
   features  importance
0      CHAS    0.000164
1        ZN    0.000471
2       RAD    0.001188
3     INDUS    0.003634
4         B    0.007110
5       TAX    0.010512
6       NOX    0.011172
7       AGE    0.013020
8   PTRATIO    0.021871
9      CRIM    0.023349
10      DIS    0.066079
11       RM    0.421620
12    LSTAT    0.537335
```



Drop column feature importance

we can use drop-columns importance to get more accurate feature importance.

In [106…

```
def dropcol_importances(rf, X_train, y_train):
    rf_ = clone(rf)
    rf_.random_state = 999
    rf_.fit(X_train, y_train)
    baseline = rf_.oob_score_
    imp = []
    for col in X_train.columns:
        X = X_train.drop(col, axis=1)
        rf_ = clone(rf)
        rf_.random_state = 999
        rf_.fit(X, y_train)
        o = rf_.oob_score_
        imp.append(baseline - o)
    imp = np.array(imp)
    I = pd.DataFrame(
            data={'Feature':X_train.columns,
                  'Importance':imp})
    I = I.set_index('Feature')
    I = I.sort_values('Importance', ascending=True)
    return I
```

In [109…

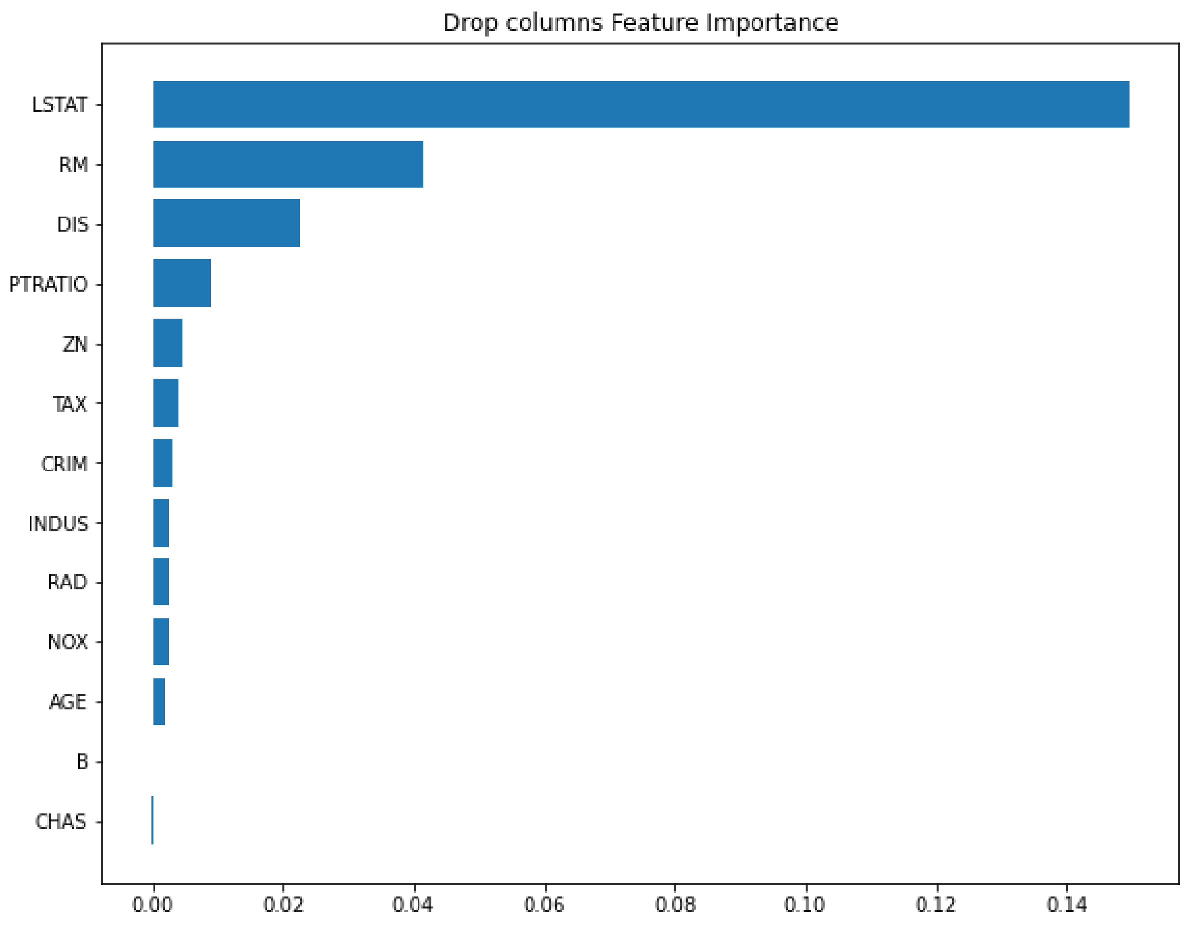
```
# train a random forest regressor on the training set
rf = RandomForestRegressor(n_estimators=100, random_state=42, oob_score=True
rf.fit(X_train, y_train)
drop_imp = dropcol_importances(rf, X_train, y_train)
drop_imp
```

Out[109]:

Feature	Importance
CHAS	-0.000236
B	-0.000022
AGE	0.001992
NOX	0.002455
RAD	0.002468
INDUS	0.002488
CRIM	0.003176
TAX	0.003986
ZN	0.004649
PTRATIO	0.008928
DIS	0.022452
RM	0.041468
LSTAT	0.149666

In [114…

```
fig, ax = plt.subplots(figsize=(10,8))
plt.title("Drop columns Feature Importance")
plt.barh(drop_imp.index, drop_imp['Importance'])
plt.show()
```



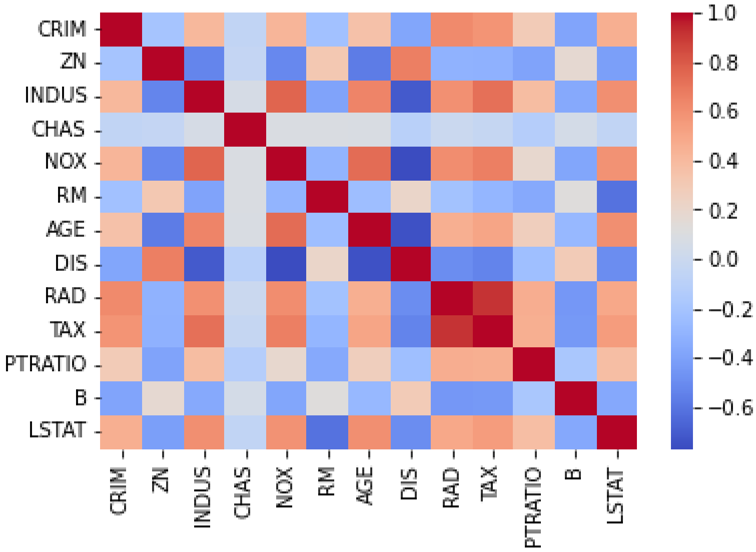
Dealing with collinear features

What if some of our features are collinear? i.e. one features changes might affect the other

In [115…

```
import seaborn as sns

# compute correlation matrix
corr_matrix = df.corr()

# visualize correlation matrix as a heatmap
sns.heatmap(corr_matrix, cmap="coolwarm")
```

Out[115]: <AxesSubplot:>



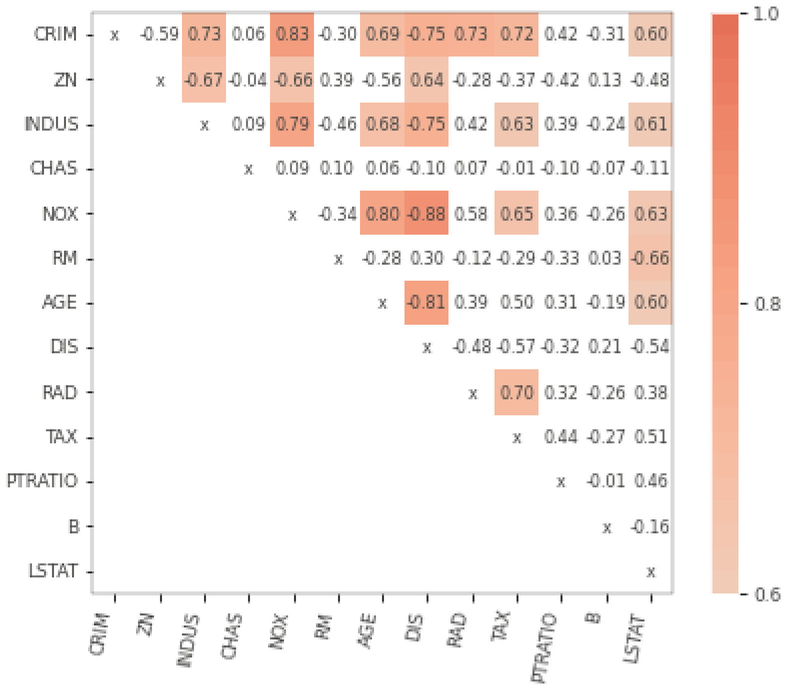
In [122…

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# compute VIF for each feature
# X = df.drop("target", axis=1) # assume "target" is the target variable
vif = pd.DataFrame()
vif["feature"] = df.columns
vif["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.shap

# print VIF values
print(vif)
```

```
    feature        VIF
0      CRIM   2.100373
1        ZN   2.844013
2     INDUS  14.485758
3      CHAS   1.152952
4       NOX  73.894947
5        RM  77.948283
6       AGE  21.386850
7       DIS  14.699652
8       RAD  15.167725
9       TAX  61.227274
10  PTRATIO  85.029547
11        B  20.104943
12    LSTAT  11.102025
```

In [123…

```python
from rfpimp import plot_corr_heatmap
viz = plot_corr_heatmap(X_train, figsize=(7,5))
viz.view()
```



There are some collinear feature like NOX, TAX, PTRATIO and so on

For example, NOX and DIS has a correlation of -0.88, which means on average, the increase of 1 unit of NOX result in decrease of DIS by 88 units. Let try to plot the feature importance by group

Automatic feature selection algorithm

Here is one of the method from SKlearn : Feature ranking with recursive feature elimination.

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

In [127…

```python
from sklearn.feature_selection import RFE

# create a logistic regression model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# create an RFE object to select the top 5 features
rfe = RFE(model, n_features_to_select=5, verbose=1, step=1)

# fit the RFE object to the data
rfe.fit(X_train, y_train)
```

```
# get the selected features
selected_features = X_train.columns[rfe.support_]
```

```
Fitting estimator with 13 features.
Fitting estimator with 12 features.
Fitting estimator with 11 features.
Fitting estimator with 10 features.
Fitting estimator with 9 features.
Fitting estimator with 8 features.
Fitting estimator with 7 features.
Fitting estimator with 6 features.
```

In [129…

```
selected_features
```

Out[129]:

```
Index(['CRIM', 'RM', 'DIS', 'PTRATIO', 'LSTAT'], dtype='object')
```

We select 'CRIM', 'RM', 'DIS', 'PTRATIO', 'LSTAT' as our top 5 features. The selection result is the same as the top 5 features ranking above

In [135…

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from featimp import *

# Load dataset and split into training, validation, and test sets
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, te

# Initialize the set of selected features and their validation error
selected_features = []
best_error = float('inf')

# Iterate through all features and select the one with the lowest validation
for i in range(X_train.shape[1]):
    candidate_features = selected_features + [i]
    X_train_subset = X_train[:, candidate_features]
    X_val_subset = X_val[:, candidate_features]
    model = LinearRegression().fit(X_train_subset, y_train)
    val_error = calc_validation_error(X_val_subset, y_val, model)
    if val_error < best_error:
        best_error = val_error
        selected_features = candidate_features

# Train a final model using the selected features on the training and valida
X_trainval_subset = X_trainval[:, selected_features]
model = LinearRegression().fit(X_trainval_subset, y_trainval)

# Evaluate the final model on the test set
X_test_subset = X_test[:, selected_features]
test_error = calc_validation_error(X_test_subset, y_test, model)
print("Selected features:", selected_features)
print("Test error:", test_error)
```

```
Selected features: [0, 1, 2, 3, 5, 6, 7, 10, 12]
Test error: 25.244798638656956
```

In [142…

```
df.columns[selected_features]
```

Out[142]:

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'RM', 'AGE', 'DIS', 'PTRATIO', 'LSTA
T'], dtype='object')
```