Task1

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
```

In [2]:
```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import validation_curve
from sklearn.model_selection import KFold
```

1.1

In [3]:
```
credit = fetch_openml('credit-g', as_frame = True)
df=credit.data
df['class'] = credit.target
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
checking_status           1000 non-null category
duration                  1000 non-null float64
credit_history            1000 non-null category
purpose                   1000 non-null category
credit_amount             1000 non-null float64
savings_status            1000 non-null category
employment                1000 non-null category
installment_commitment    1000 non-null float64
personal_status           1000 non-null category
other_parties             1000 non-null category
residence_since           1000 non-null float64
property_magnitude        1000 non-null category
age                       1000 non-null float64
other_payment_plans       1000 non-null category
housing                   1000 non-null category
existing_credits          1000 non-null float64
job                       1000 non-null category
num_dependents            1000 non-null float64
own_telephone             1000 non-null category
foreign_worker            1000 non-null category
class                     1000 non-null category
dtypes: category(14), float64(7)
memory usage: 70.8 KB
```

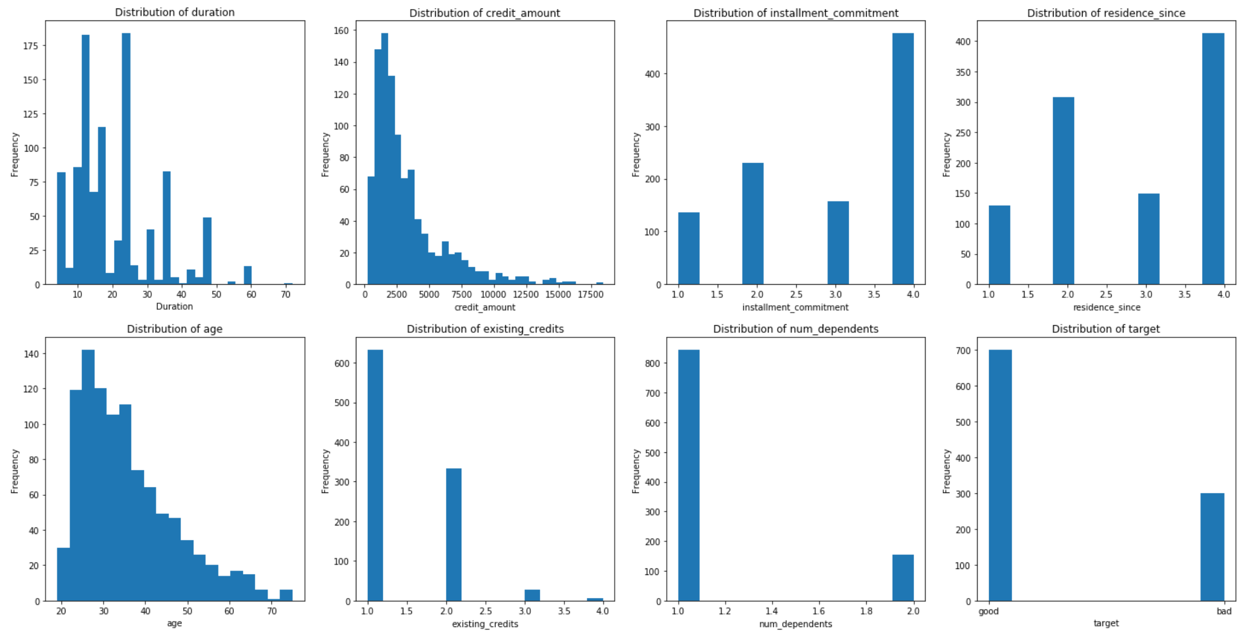
Continuous data:\ duration, credit_amount, installment_commitment, residence_since, age, existing_credits, num_dependents

Categorical data:\ checking_status, credit_history, purpose, savings_status, employment, personal_status, other_parties, property_magnitude, other_payment_plans, housing, job, own_telephone, foreign_worker,

1.2

In [4]:
```
fig, ax = plt.subplots(2,4, figsize=(25,12.5));
ax[0,0].hist(df['duration'], bins='auto');
ax[0,0].set_title('Distribution of duration')
ax[0,0].set_xlabel('Duration')
ax[0,0].set_ylabel('Frequency')

ax[0,1].hist(df['credit_amount'], bins='auto');
ax[0,1].set_title('Distribution of credit_amount')
ax[0,1].set_xlabel('credit_amount')
ax[0,1].set_ylabel('Frequency')

ax[0,2].hist(df['installment_commitment'], bins='auto');
ax[0,2].set_title('Distribution of installment_commitment')
ax[0,2].set_xlabel('installment_commitment')
ax[0,2].set_ylabel('Frequency')

ax[0,3].hist(df['residence_since'], bins='auto');
ax[0,3].set_title('Distribution of residence_since')
ax[0,3].set_xlabel('residence_since')
ax[0,3].set_ylabel('Frequency')

ax[1,0].hist(df['age'], bins='auto');
ax[1,0].set_title('Distribution of age')
ax[1,0].set_xlabel('age')
ax[1,0].set_ylabel('Frequency')

ax[1,1].hist(df['existing_credits'], bins='auto');
ax[1,1].set_title('Distribution of existing_credits')
ax[1,1].set_xlabel('existing_credits')
ax[1,1].set_ylabel('Frequency')

ax[1,2].hist(df['num_dependents'], bins='auto');
ax[1,2].set_title('Distribution of num_dependents')
ax[1,2].set_xlabel('num_dependents')
ax[1,2].set_ylabel('Frequency')

ax[1,3].hist(df['class']);
ax[1,3].set_title('Distribution of target')
ax[1,3].set_xlabel('target')
ax[1,3].set_ylabel('Frequency')
```

Out[4]: Text(0, 0.5, 'Frequency')



1.3

In [5]:
```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
X_trainval, X_test, y_trainval, y_test = train_test_split(X,y, random_
state = 123)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainv
al, random_state = 123)

X_train_cont = X_train.loc[:,X.columns[X.dtypes != 'category']]
X_val_cont = X_val.loc[:,X.columns[X.dtypes != 'category']]
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_cont)
X_val_scaled = scaler.transform(X_val_cont)

X_train.loc[:,X.columns[X.dtypes != 'category']] = X_train_scaled
X_val.loc[:,X.columns[X.dtypes != 'category']] = X_val_scaled

X_train = pd.get_dummies(X_train)
X_val = pd.get_dummies(X_val)

lr = LogisticRegression().fit(X_train, y_train)
lr.score(X_val, y_val) #evaluate an initial Logistic Regression model
w/an traning/validation split
```

Out[5]: 0.7446808510638298

1.4

In [6]:
```
categorical = df.iloc[:,:-1].dtypes != float
preprocess_scaled = make_column_transformer(
    (StandardScaler(), ~categorical),
    (OneHotEncoder(),categorical))
preprocess_notscaled = make_column_transformer(
    (OneHotEncoder(),categorical))
```

In [7]:
```
#pipe = make_pipeline(StandardScaler(),LogisticRegression())
#pipe_fit(X_train, y_train)
#pipe_score(X_test,y_tesgt)
```

In [8]:
```
#CVscore=[]
#Logistic Regression
#Non Scaled
lr_p_n = make_pipeline(preprocess_notscaled, LogisticRegression())
lr_n_scores = np.mean(cross_val_score(lr_p_n,df.iloc[:,:-1],df.iloc[:,
-1], cv=10))

#Scaled
lr_p_y = make_pipeline(preprocess_scaled, LogisticRegression())
lr_y_scores = np.mean(cross_val_score(lr_p_y,df.iloc[:,:-1],df.iloc[:,
-1], cv=10))

#Linear Support Vector Machine
#Non Scaled
svc_p_n = make_pipeline(preprocess_notscaled, LinearSVC(max_iter=5000)
)
svc_n_scores = np.mean(cross_val_score(svc_p_n,df.iloc[:,:-1],df.iloc[
:,-1], cv=10))
#Scaled
svc_p_y = make_pipeline(preprocess_scaled, LinearSVC(max_iter=5000))
svc_y_scores = np.mean(cross_val_score(svc_p_y,df.iloc[:,:-1],df.iloc[
:,-1], cv=10))

#KNN
#Non Scaled
knn_p_n = make_pipeline(preprocess_notscaled, KNeighborsClassifier())
knn_n_scores = np.mean(cross_val_score(knn_p_n,df.iloc[:,:-1],df.iloc[
:,-1], cv=10))

#Scaled
knn_p_y = make_pipeline(preprocess_scaled, KNeighborsClassifier())
knn_y_scores = np.mean(cross_val_score(knn_p_y,df.iloc[:,:-1],df.iloc[
:,-1], cv=10))

print('Cross Validation score for Logistic Regression: without scaling
is {s1:.3f}, the score with scaling is {s2:.3f}.'
      .format(s1=lr_n_scores, s2=lr_y_scores))
print('Cross Validation score for Linear Support Vector Machine: witho
ut scaling is {s1:.3f}, the score with scaling is {s2:.3f}.'
      .format(s1=svc_n_scores, s2=svc_y_scores))
print('Cross Validation score for K Nearest Neighbors: without scaling
is {s1:.3f}, the score with scaling is {s2:.3f}.'
      .format(s1=knn_n_scores, s2=knn_y_scores))
```

```
Cross Validation score for Logistic Regression: without scaling is 0
.741, the score with scaling is 0.750.
Cross Validation score for Linear Support Vector Machine: without sc
aling is 0.739, the score with scaling is 0.752.
Cross Validation score for K Nearest Neighbors: without scaling is 0
.714, the score with scaling is 0.722.
```

Overall, the scaled data provides higher cross validation score. Logistic Regression and Linear Support Vectore Machine performs well for both scaled and unscaled data. KNN is a bit worse. However, KNN also increases in its CV score. The modification changes the result because after scaling, the classification process will be less affected by scales. Therefore, scaling features with StandardScaler can increase accuracies.

1.5

In [9]:
```
#Logistic Regression
lr_pipe = make_pipeline(preprocess_scaled, LogisticRegression())
param = np.logspace(-4,4,9)
lr_param = {'logisticregression__C': param}
gridlr = GridSearchCV(lr_pipe, lr_param, cv=10, return_train_score=Tru
e)
gridlr.fit(X_trainval, y_trainval)
print("best parameters: ", gridlr.best_params_)
print("best_mean cv score: ", gridlr.best_score_)

#Linear Support Vector Machine
svm_pipe = make_pipeline(preprocess_scaled, LinearSVC())
svm_param = {'linearsvc__C': np.logspace(-4,4,9)}
gridsvm = GridSearchCV(svm_pipe, svm_param, cv=10, return_train_score=
True)
gridsvm.fit(X_trainval, y_trainval)
print("best parameters: ", gridsvm.best_params_)
print("best_mean cv score: ", gridsvm.best_score_)

##KNN
knn_pipe = make_pipeline(preprocess_scaled, KNeighborsClassifier())
knn_param = {'kneighborsclassifier__n_neighbors': np.arange(1, 15, 2)}
gridknn = GridSearchCV(knn_pipe, knn_param, cv=10, return_train_score=
True)
gridknn.fit(X_trainval, y_trainval)
print("best parameters: ", gridknn.best_params_)
print("best_mean cv score: ", gridknn.best_score_)
```

```
best parameters:  {'logisticregression__C': 0.1}
best_mean cv score:  0.7533333333333334
best parameters:  {'linearsvc__C': 0.01}
best_mean cv score:  0.752
best parameters:  {'kneighborsclassifier__n_neighbors': 13}
best_mean cv score:  0.7506666666666668
```

In [10]:
```
print("LR: test-set score: {:.3f}".format(gridlr.score(X_test, y_test)
))
print("SVM: test-set score: {:.3f}".format(gridsvm.score(X_test, y_tes
t)))
print("KNN: test-set score: {:.3f}".format(gridknn.score(X_test, y_tes
t)))
```

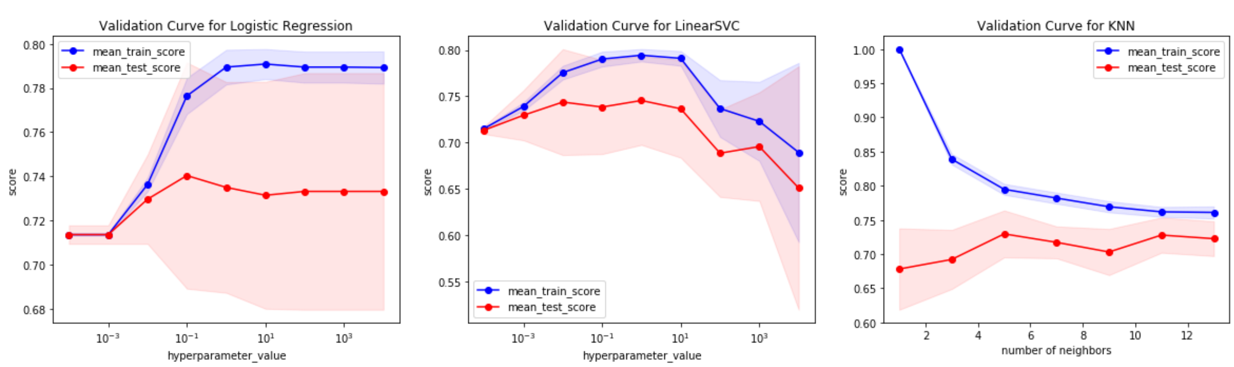
```
LR: test-set score: 0.732
SVM: test-set score: 0.724
KNN: test-set score: 0.716
```

The best model is logistic regression model with hyperparameter = 0.1 with best mean cv score 0.753. The test set score is 0.732

In [11]:
```
#plot
figg, axx = plt.subplots(1,3, figsize=(20,5))
##Logistic Regression
train_scores_lr,test_scores_lr = validation_curve(LogisticRegression()
,X_train, y_train,
                                                 'C',param,cv=10,scor
ing='accuracy')
mean_train_scores_lr = np.mean(train_scores_lr,1)
std_train_scores_lr = np.std(train_scores_lr,1)
mean_test_scores_lr = np.mean(test_scores_lr,1)
std_test_scores_lr = np.std(test_scores_lr,1)

axx[0].semilogx(param, mean_train_scores_lr,"o-",
                color="b", label="mean_train_score")
axx[0].semilogx(param, mean_test_scores_lr,"o-",
                color="r", label="mean_test_score")
axx[0].fill_between(param,mean_train_scores_lr-std_train_scores_lr,
                 mean_train_scores_lr+std_train_scores_lr,color='b',al
pha=0.1)
axx[0].fill_between(param,mean_test_scores_lr-std_test_scores_lr,
                 mean_test_scores_lr+std_test_scores_lr,color='r',alph
a=0.1)
axx[0].set_xlabel('hyperparameter_value') ;
axx[0].set_ylabel('score');
axx[0].set_title('Validation Curve for Logistic Regression');
axx[0].legend(loc='best')

##SVC
train_scores_svc,test_scores_svc = validation_curve(LinearSVC(),X_trai
n, y_train,
                                                 'C',param,cv=10,scor
ing='accuracy')
mean_train_scores_svc = np.mean(train_scores_svc,1)
std_train_scores_svc = np.std(train_scores_svc,1)
mean_test_scores_svc = np.mean(test_scores_svc,1)
std_test_scores_svc = np.std(test_scores_svc,1)

axx[1].semilogx(param, mean_train_scores_svc,"o-",
                color="b", label="mean_train_score")
axx[1].semilogx(param, mean_test_scores_svc,"o-",
                color="r", label="mean_test_score")
axx[1].fill_between(param,mean_train_scores_svc-std_train_scores_svc,
                 mean_train_scores_svc+std_train_scores_svc,color='b',
alpha=0.1)
axx[1].fill_between(param,mean_test_scores_svc-std_test_scores_svc,
```

```
                 mean_test_scores_svc+std_test_scores_svc,color='r',al
pha=0.1)
axx[1].set_xlabel('hyperparameter_value') ;
axx[1].set_ylabel('score');
axx[1].set_title('Validation Curve for LinearSVC');
axx[1].legend(loc='best')

##KNN
train_scores_knn,test_scores_knn = validation_curve(KNeighborsClassifi
er(),
                                                    X_train, y_train,'
n_neighbors',np.arange(1, 15, 2),
                                                    cv=10,scoring='acc
uracy')
mean_train_scores_knn = np.mean(train_scores_knn,1)
std_train_scores_knn = np.std(train_scores_knn,1)
mean_test_scores_knn = np.mean(test_scores_knn,1)
std_test_scores_knn = np.std(test_scores_knn,1)

axx[2].plot(np.arange(1, 15, 2), mean_train_scores_knn,"o-",
                color="b", label="mean_train_score")
axx[2].plot(np.arange(1, 15, 2), mean_test_scores_knn,"o-",
                color="r", label="mean_test_score")
axx[2].fill_between(np.arange(1, 15, 2),mean_train_scores_knn-std_trai
n_scores_knn,
                 mean_train_scores_knn+std_train_scores_knn,color='b',
alpha=0.1)
axx[2].fill_between(np.arange(1, 15, 2),mean_test_scores_knn-std_test_
scores_knn,
                 mean_test_scores_knn+std_test_scores_knn,color='r',al
pha=0.1)
axx[2].set_xlabel('number of neighbors') ;
axx[2].set_ylabel('score');
axx[2].set_title('Validation Curve for KNN');
axx[2].legend(loc='best')
```

Out[11]: <matplotlib.legend.Legend at 0x1a18d9abd0>



1.6

In [12]:
```
#Change stratified Kfold to Kfold shuffling.
kfold = KFold(n_splits = 10, shuffle = True, random_state=123)

kfold_cv_lr = GridSearchCV(lr_p_y, param_grid = lr_param,
                           cv = kfold, return_train_score = True)
kfold_cv_lr.fit(X_trainval, y_trainval)
print("The best parameter for Logistic Regression is: ", kfold_cv_lr.b
est_params_)
kfold_cv_svc = GridSearchCV(svc_p_y, param_grid = svm_param,
                           cv = kfold, return_train_score = True)
kfold_cv_svc.fit(X_trainval, y_trainval)
print("The best parameter for Linear SVC is: ", kfold_cv_svc.best_para
ms_)


kfold_cv_knn = GridSearchCV(knn_p_y, param_grid = knn_param,
                           cv = kfold, return_train_score = True)
kfold_cv_knn.fit(X_trainval, y_trainval)
print("The best parameter for KNN is: ", kfold_cv_knn.best_params_)
```

```
The best parameter for Logistic Regression is:  {'logisticregression
__C': 0.1}
The best parameter for Linear SVC is:  {'linearsvc__C': 0.01}
The best parameter for KNN is:  {'kneighborsclassifier__n_neighbors'
: 13}
```

After changing cross validation strategy from 'stratified k-fold' to 'kfold' with shuffling, all parameters stays same.

In [13]:
```
#Change Random States
randstate = np.arange(0,100,23)
##LR
for i in range(len(randstate)):
    kfold_cv1 = GridSearchCV(lr_p_y, param_grid = lr_param,
                           cv = KFold(n_splits = 10, shuffle = True, r
andom_state=randstate[i]),
                            return_train_score = True)
    kfold_cv1.fit(X_trainval, y_trainval)
    print('The best parameter for logistic regression is:', kfold_cv1.
best_params_,
         'when random state =', randstate[i])

#SVC
for i in range(len(randstate)):
    kfold_cv1 = GridSearchCV(svc_p_y, param_grid = svm_param,
                           cv = KFold(n_splits = 10, shuffle = True, r
andom_state=randstate[i]),
                            return_train_score = True)
    kfold_cv1.fit(X_trainval, y_trainval)
    print('The best parameter for Linear SVC is:', kfold_cv1.best_para
ms_,
         'when random state =', randstate[i])

#KNN
for i in range(len(randstate)):
    kfold_cv1 = GridSearchCV(knn_p_y, param_grid = knn_param,
                           cv = KFold(n_splits = 10, shuffle = True, r
andom_state=randstate[i]),
                            return_train_score = True)
    kfold_cv1.fit(X_trainval, y_trainval)
    print('The best parameter for KNN is:', kfold_cv1.best_params_,
         'when random state =', randstate[i])
```

```
The best parameter for logistic regression is: {'logisticregression_
_C': 0.1} when random state = 0
The best parameter for logistic regression is: {'logisticregression_
_C': 1.0} when random state = 23
The best parameter for logistic regression is: {'logisticregression_
_C': 0.1} when random state = 46
The best parameter for logistic regression is: {'logisticregression_
_C': 0.1} when random state = 69
The best parameter for logistic regression is: {'logisticregression_
_C': 0.1} when random state = 92
The best parameter for Linear SVC is: {'linearsvc__C': 0.1} when ran
dom state = 0
The best parameter for Linear SVC is: {'linearsvc__C': 0.1} when ran
dom state = 23
The best parameter for Linear SVC is: {'linearsvc__C': 0.01} when ra
ndom state = 46
The best parameter for Linear SVC is: {'linearsvc__C': 1.0} when ran
dom state = 69
The best parameter for Linear SVC is: {'linearsvc__C': 0.01} when ra
ndom state = 92
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
13} when random state = 0
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
5} when random state = 23
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
13} when random state = 46
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
13} when random state = 69
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
13} when random state = 92
```

If we change random seed of the shuffling, best parameters do have difference in different random states for all three models.

In [14]:

```
#Change random state when spliting data into train and test sets.
#LR
for j in range(len(randstate)):
    X_trainval1, X_test1, y_trainval1, y_test1 = train_test_split(X, y
, random_state = randstate[j])
    s_cv = GridSearchCV(lr_p_y, param_grid = lr_param,
                           cv = KFold(n_splits = 10, shuffle = True, r
andom_state=123),
                        return_train_score = True)
    s_cv.fit(X_trainval1, y_trainval1)
    print('The best parameter for logistic regression is:', s_cv.best_
params_,
         'when random state for spliting dataset =', randstate[j])
```

```
The best parameter for logistic regression is: {'logisticregression_
_C': 0.1} when random state for spliting dataset = 0
The best parameter for logistic regression is: {'logisticregression_
_C': 0.1} when random state for spliting dataset = 23
The best parameter for logistic regression is: {'logisticregression_
_C': 100.0} when random state for spliting dataset = 46
The best parameter for logistic regression is: {'logisticregression_
_C': 1.0} when random state for spliting dataset = 69
The best parameter for logistic regression is: {'logisticregression_
_C': 1.0} when random state for spliting dataset = 92
```

In [15]:

```
#SVC
for j in range(len(randstate)):
    X_trainval1, X_test1, y_trainval1, y_test1 = train_test_split(X, y
, random_state = randstate[j])
    s_cv = GridSearchCV(svc_p_y, param_grid = svm_param,
                           cv = KFold(n_splits = 10, shuffle = True, r
andom_state=123),
                            return_train_score = True)
    s_cv.fit(X_trainval1, y_trainval1)
    print('The best parameter for LinearSVC is:', s_cv.best_params_,
         'when random state for spliting dataset =', randstate[j])
```

```
The best parameter for LinearSVC is: {'linearsvc__C': 0.1} when rand
om state for spliting dataset = 0
The best parameter for LinearSVC is: {'linearsvc__C': 0.01} when ran
dom state for spliting dataset = 23
The best parameter for LinearSVC is: {'linearsvc__C': 0.1} when rand
om state for spliting dataset = 46
The best parameter for LinearSVC is: {'linearsvc__C': 100.0} when ra
ndom state for spliting dataset = 69
The best parameter for LinearSVC is: {'linearsvc__C': 0.1} when rand
om state for spliting dataset = 92
```

In [16]:
```
#KNN
for j in range(len(randstate)):
    X_trainval1, X_test1, y_trainval1, y_test1 = train_test_split(X, y
, random_state = randstate[j])
    s_cv = GridSearchCV(knn_p_y, param_grid = knn_param,
                              cv = KFold(n_splits = 10, shuffle = True, r
andom_state=123),
                              return_train_score = True)
    s_cv.fit(X_trainval1, y_trainval1)
    print('The best parameter for KNN is:', s_cv.best_params_,
         'when random state for spliting dataset =', randstate[j])
```

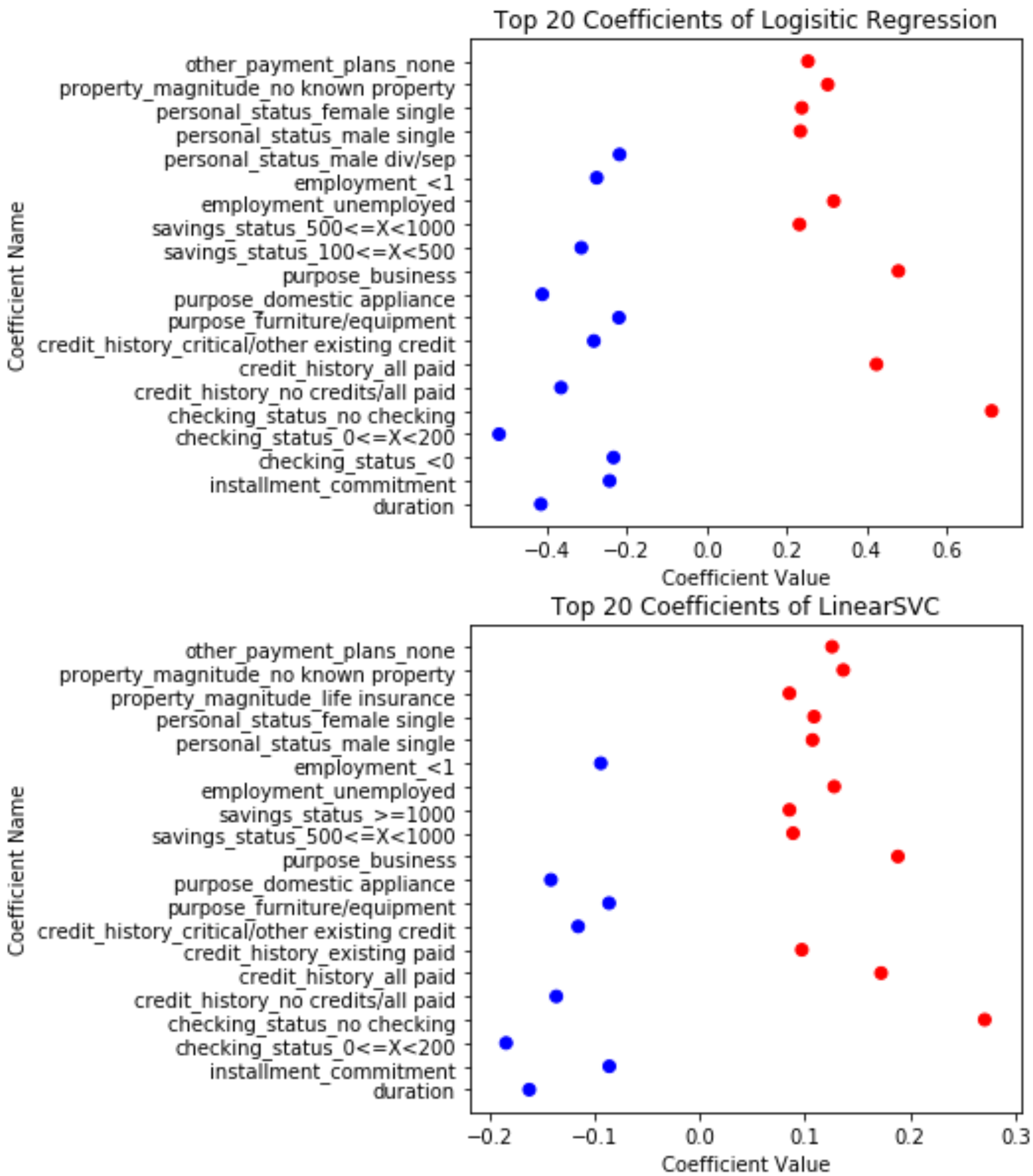
```
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
5} when random state for spliting dataset = 0
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
3} when random state for spliting dataset = 23
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
7} when random state for spliting dataset = 46
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
13} when random state for spliting dataset = 69
The best parameter for KNN is: {'kneighborsclassifier__n_neighbors':
7} when random state for spliting dataset = 92
```

If we change the reandom state of the split into training and test data, parameters would have chance to change for all methods.

1.7

In [17]:
```
fig, ax = plt.subplots(2, 1, figsize = (5, 10))
columns = pd.get_dummies(X).columns
coef_lr = gridlr.best_estimator_[1].coef_[0]
idx_lr = np.sort(np.abs(coef_lr).argsort()[-20:][::-1])
ax[0].scatter(coef_lr[idx_lr],columns[idx_lr], c = np.sign(coef_lr[idx
_lr]), cmap='bwr')
ax[0].set_title('Top 20 Coefficients of Logisitic Regression')
ax[0].set_xlabel('Coefficient Value')
ax[0].set_ylabel('Coefficient Name')

coef_svc = gridsvm.best_estimator_[1].coef_[0]
idx_svc = np.sort(np.abs(coef_svc).argsort()[-20:][::-1])
ax[1].scatter(coef_svc[idx_svc],columns[idx_svc], c = np.sign(coef_svc
[idx_svc]), cmap='bwr')
ax[1].set_title('Top 20 Coefficients of LinearSVC')
ax[1].set_xlabel('Coefficient Value')
ax[1].set_ylabel('Coefficient Name')
```

```
Out[17]: Text(0, 0.5, 'Coefficient Name')
```




```
In [ ]:
```

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
```

In [3]:
```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import validation_curve
from sklearn.model_selection import KFold

from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
```

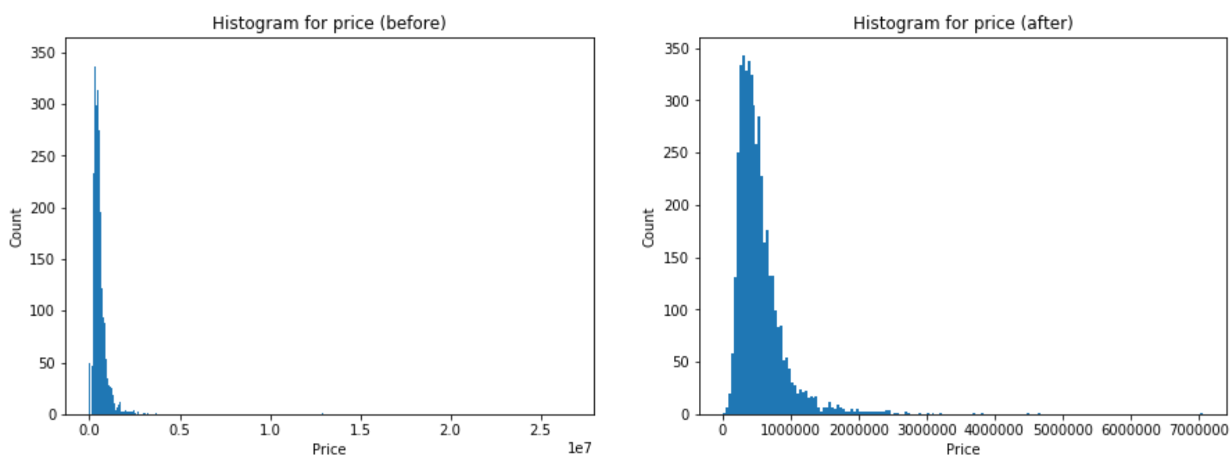
2.1

For target 'price', I will remove 4 outliers of 'price' and 0s, which contain no information.

In [4]:
```
df_temp = pd.read_csv('data.csv')
df_temp = df_temp.drop(['date'], axis=1)
#remove outliers of 'price' and 0s, which contain no information.
df = df_temp.loc[(df_temp['price'] != 0) & (df_temp['price'] < 1000000
0), :]
X = df.iloc[:,1:]
y = pd.DataFrame(df.iloc[:,0])
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4549 entries, 0 to 4599
Data columns (total 17 columns):
price            4549 non-null float64
bedrooms         4549 non-null float64
bathrooms        4549 non-null float64
sqft_living      4549 non-null int64
sqft_lot         4549 non-null int64
floors           4549 non-null float64
waterfront       4549 non-null int64
view             4549 non-null int64
condition        4549 non-null int64
sqft_above       4549 non-null int64
sqft_basement    4549 non-null int64
yr_built         4549 non-null int64
yr_renovated     4549 non-null int64
street           4549 non-null object
city             4549 non-null object
statezip         4549 non-null object
country          4549 non-null object
dtypes: float64(4), int64(9), object(4)
memory usage: 639.7+ KB
None
```

In [5]:
```
df_temp = pd.read_csv('data.csv')
df_temp = df_temp.drop(['date'], axis=1)

df = df_temp.loc[(df_temp['price'] != 0) & (df_temp['price'] < 1000000
0), :]
X = df.iloc[:,1:]
y = pd.DataFrame(df.iloc[:,0])
fig1, ax1 = plt.subplots(1, 2, figsize = (15,5))
ax1[0].hist(df_temp['price'], bins='auto');
ax1[0].set_title('Histogram for price (before)')
ax1[0].set_xlabel('Price')
ax1[0].set_ylabel('Count')
ax1[1].hist(y['price'], bins='auto');
ax1[1].set_title('Histogram for price (after)')
ax1[1].set_xlabel('Price')
ax1[1].set_ylabel('Count');
```



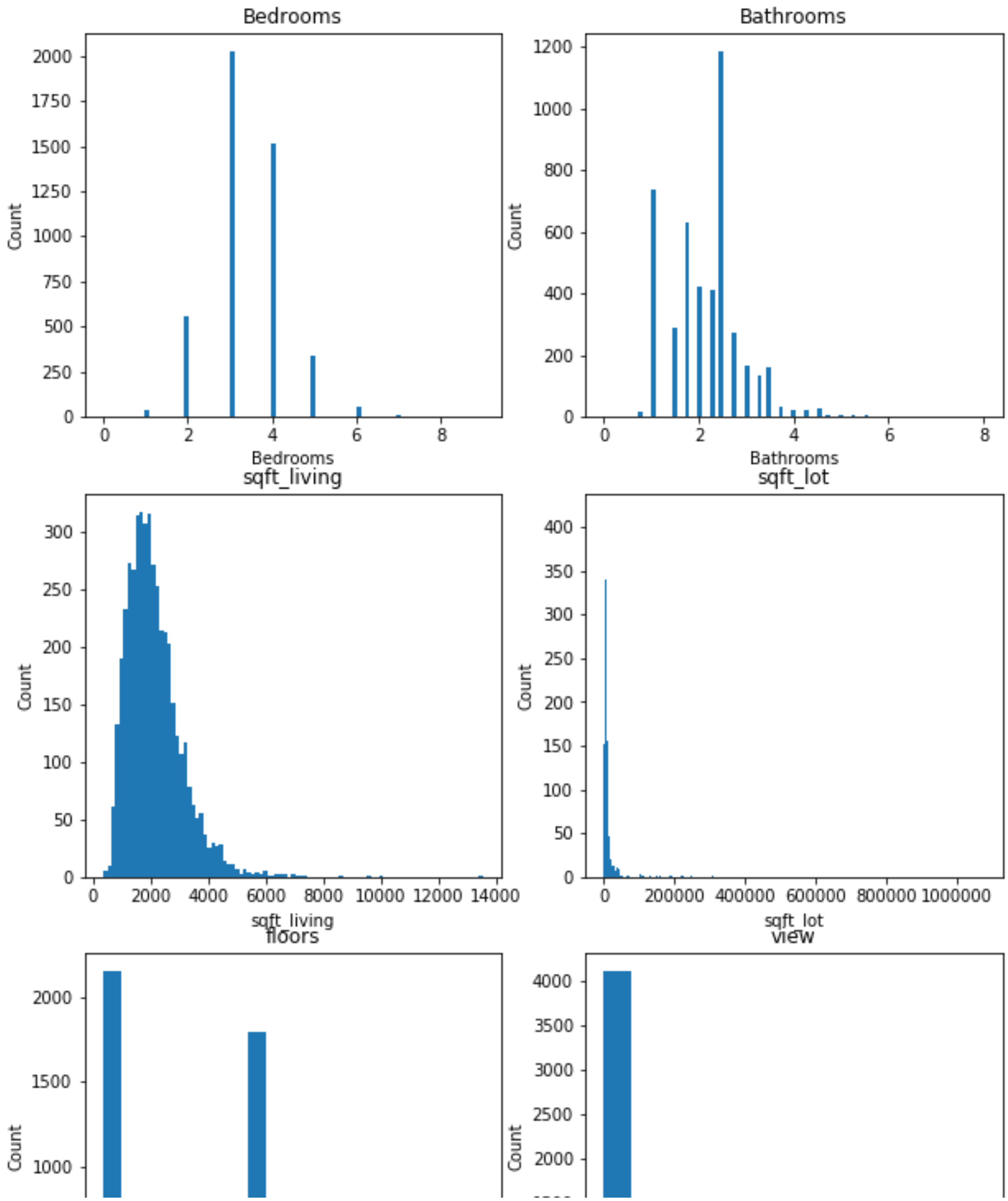
In [6]:
```
print('Continuous variables: ', list(df.columns[df.dtypes != 'object']
))
print('Categorical variables: ', list(df.columns[df.dtypes == 'object'
]))
```

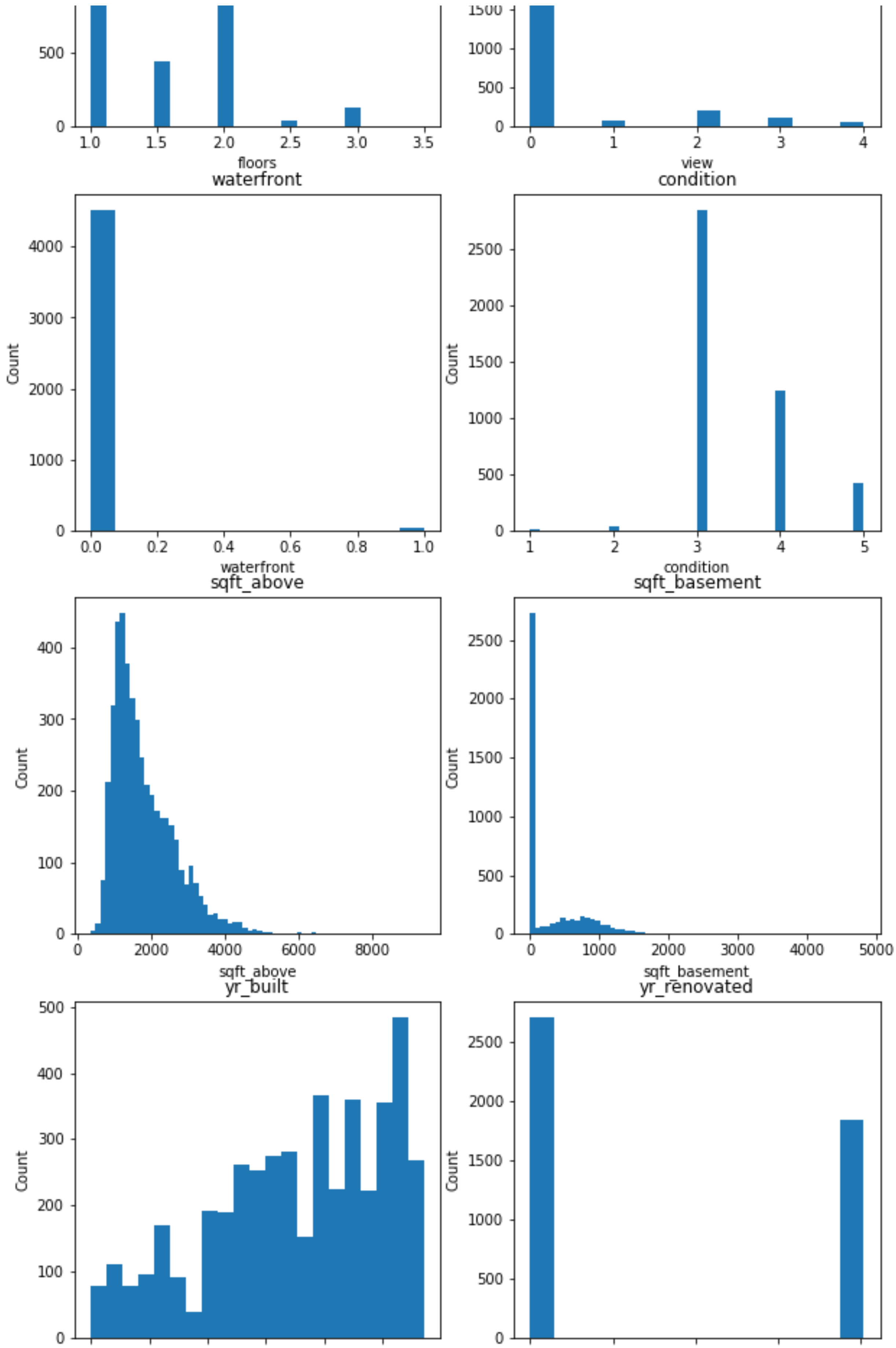
```
Continuous variables:  ['price', 'bedrooms', 'bathrooms', 'sqft_livi
ng', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'sqft_
above', 'sqft_basement', 'yr_built', 'yr_renovated']
Categorical variables:  ['street', 'city', 'statezip', 'country']
```

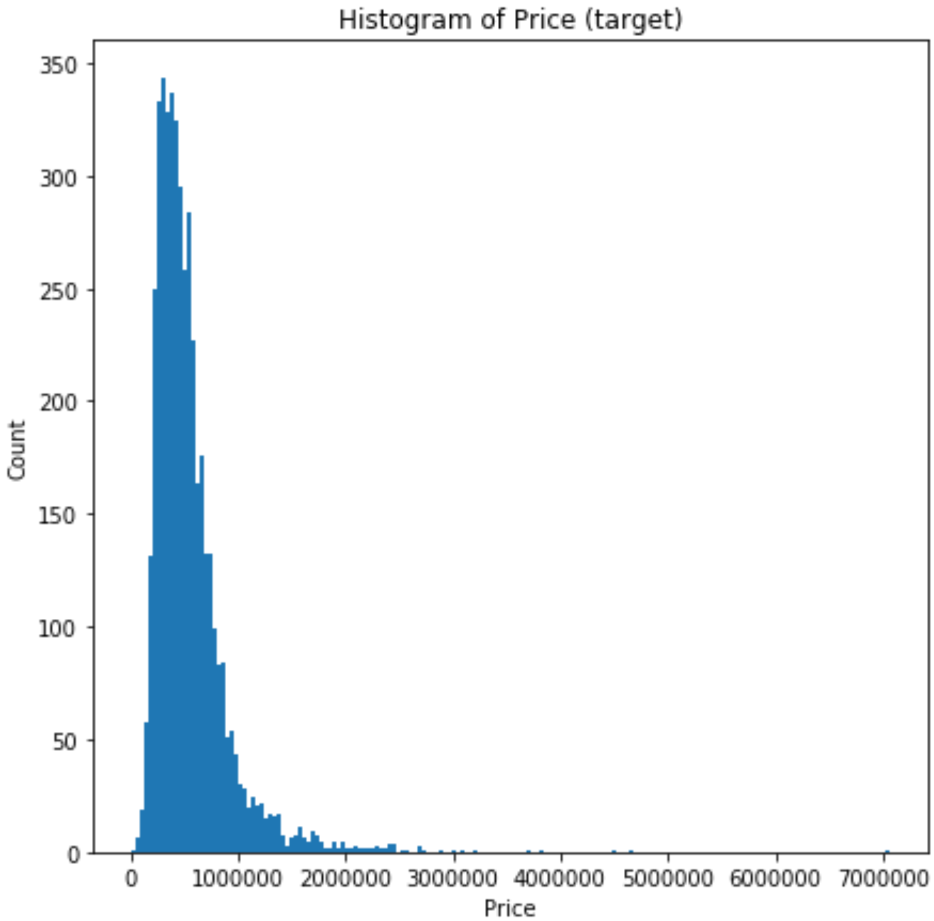
In [7]:
```
fig, ax = plt.subplots(6, 2, figsize = (10, 30))

ax[0,0].hist(X['bedrooms'], bins = 'auto')
ax[0,0].set_title("Bedrooms")
ax[0,0].set_xlabel('Bedrooms')
ax[0,0].set_ylabel('Count')

ax[0,1].hist(X['bathrooms'], bins = 'auto')
```

```
ax[0,1].set_title("Bathrooms")
ax[0,1].set_xlabel('Bathrooms')
ax[0,1].set_ylabel('Count')


ax[1,0].hist(X['sqft_living'], bins = 'auto')
ax[1,0].set_title("sqft_living")
ax[1,0].set_xlabel('sqft_living')
ax[1,0].set_ylabel('Count')

ax[1,1].hist(X['sqft_lot'], bins = 'auto')
ax[1,1].set_title("sqft_lot")
ax[1,1].set_xlabel('sqft_lot')
ax[1,1].set_ylabel('Count')

ax[2,0].hist(X['floors'], bins = 'auto')
ax[2,0].set_title("floors")
ax[2,0].set_xlabel('floors')
ax[2,0].set_ylabel('Count')

ax[2,1].hist(X['view'], bins = 'auto')
ax[2,1].set_title("view")
ax[2,1].set_xlabel('view')
ax[2,1].set_ylabel('Count')

ax[3,0].hist(X['waterfront'], bins = 'auto')
ax[3,0].set_title("waterfront")
ax[3,0].set_xlabel('waterfront')
ax[3,0].set_ylabel('Count')

ax[3,1].hist(X['condition'], bins = 'auto')
ax[3,1].set_title("condition")
ax[3,1].set_xlabel('condition')
ax[3,1].set_ylabel('Count')

ax[4,0].hist(X['sqft_above'], bins = 'auto')
ax[4,0].set_title("sqft_above")
ax[4,0].set_xlabel('sqft_above')
ax[4,0].set_ylabel('Count')

ax[4,1].hist(X['sqft_basement'], bins = 'auto')
ax[4,1].set_title("sqft_basement")
ax[4,1].set_xlabel('sqft_basement')
ax[4,1].set_ylabel('Count')

ax[5,0].hist(X['yr_built'], bins = 'auto')
ax[5,0].set_title("yr_built")
ax[5,0].set_xlabel('yr_built')
ax[5,0].set_ylabel('Count')
```

```
ax[5,1].hist(X['yr_renovated'], bins = 'auto')
ax[5,1].set_title("yr_renovated")
ax[5,1].set_xlabel('yr_renovated')
ax[5,1].set_ylabel('Count');

figa, axa = plt.subplots(1, 1, figsize = (7, 7))
axa.hist(y['price'], bins = 'auto')
axa.set_title("Histogram of Price (target)");
axa.set_xlabel('Price')
axa.set_ylabel('Count');
```



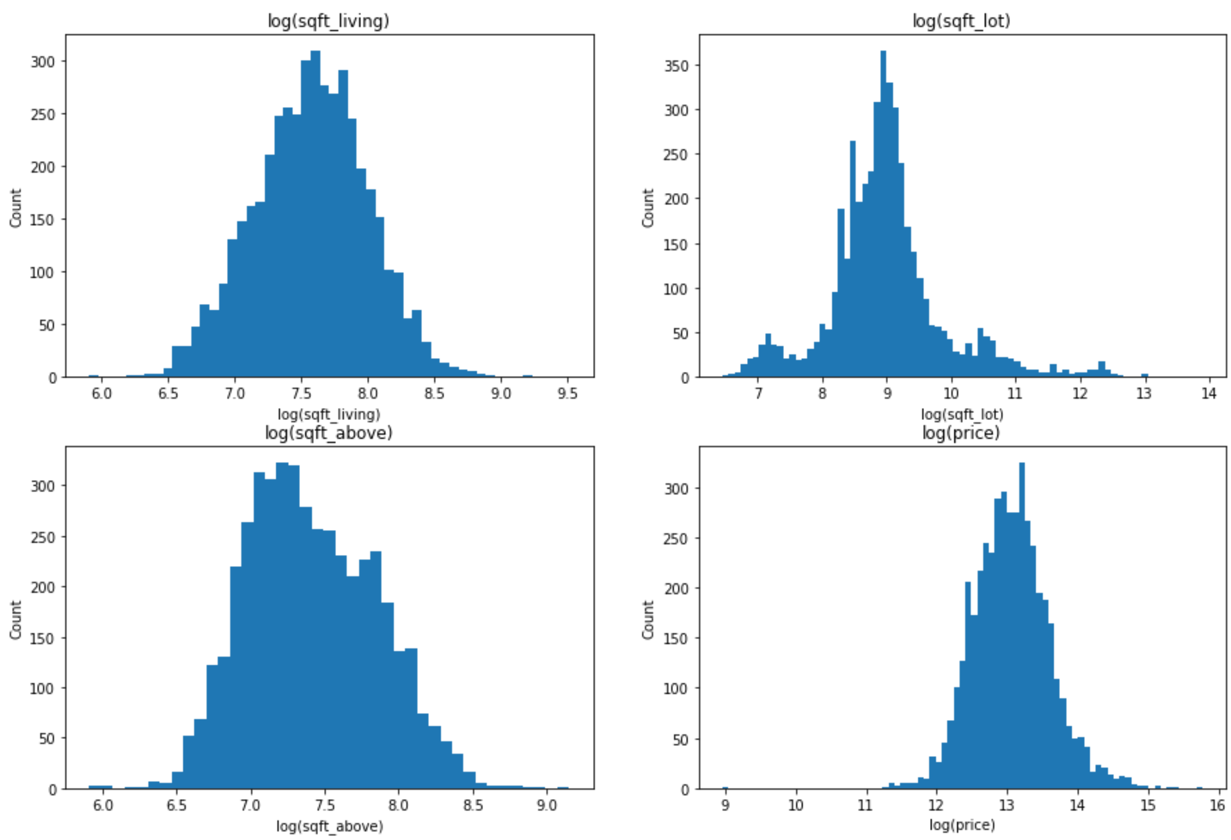

floors
view
waterfront
condition
sqft_above
sqft_basement
yr_built
yr_renovated
Count



'sqrt_living', 'sqrt_lot','sqrt_above' and 'price' are heavily right-skewed. I will take log of these values to deal with the skewness. After taking log, the histogram of these columns look like below:

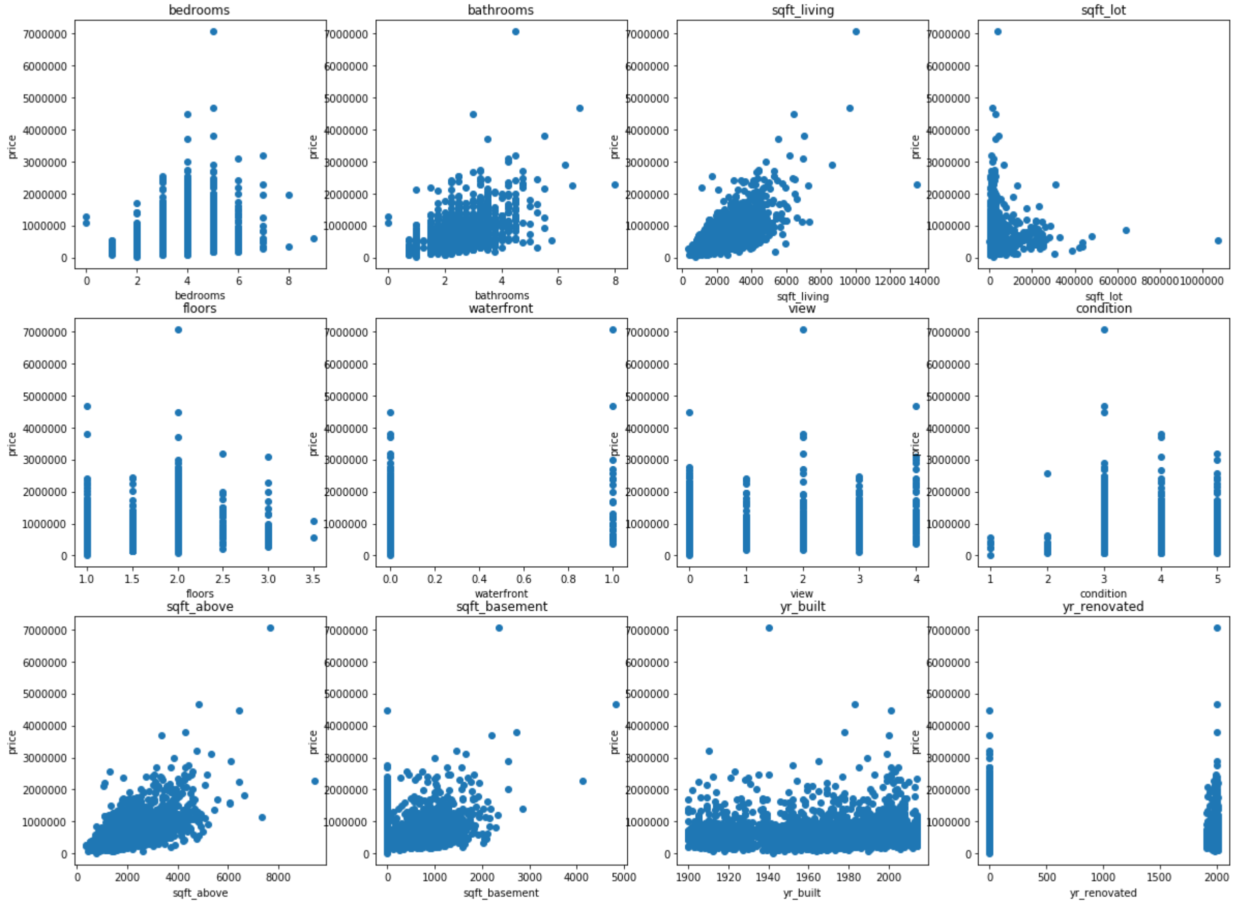
In [9]:
```
fig2, ax2 = plt.subplots(2, 2, figsize = (15, 10))
ax2[0,0].hist(np.log(X['sqft_living']), bins = 'auto')
ax2[0,0].set_title("log(sqft_living)")
ax2[0,0].set_xlabel('log(sqft_living)')
ax2[0,0].set_ylabel('Count')

ax2[0,1].hist(np.log(X['sqft_lot']), bins = 'auto')
ax2[0,1].set_title("log(sqft_lot)")
ax2[0,1].set_xlabel('log(sqft_lot)')
ax2[0,1].set_ylabel('Count')

ax2[1,0].hist(np.log(X['sqft_above']), bins = 'auto')
ax2[1,0].set_title("log(sqft_above)")
ax2[1,0].set_xlabel('log(sqft_above)')
ax2[1,0].set_ylabel('Count')

ax2[1,1].hist(np.log(y['price']), bins = 'auto');
ax2[1,1].set_title("log(price)")
ax2[1,1].set_xlabel('log(price)')
ax2[1,1].set_ylabel('Count')
```

Out[9]:
```
Text(0, 0.5, 'Count')
```



2.3

In [10]:
```
fig3, ax3 = plt.subplots(3, 4, figsize = (20, 15))
cont_var = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
            'floors', 'waterfront', 'view', 'condition',
            'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated']

for i in range(12):
    if i<=3:
        ax3[0,i].scatter(X[cont_var[i]], y['price']);
        ax3[0,i].set_title(cont_var[i])
        ax3[0,i].set_xlabel(cont_var[i])
        ax3[0,i].set_ylabel('price')
    elif 4<=i<=7:
        ax3[1,i-4].scatter(X[cont_var[i]], y['price']);
        ax3[1,i-4].set_title(cont_var[i])
        ax3[1,i-4].set_xlabel(cont_var[i])
        ax3[1,i-4].set_ylabel('price')
    else:
        ax3[2,i-8].scatter(X[cont_var[i]], y['price']);
        ax3[2,i-8].set_title(cont_var[i])
        ax3[2,i-8].set_xlabel(cont_var[i])
        ax3[2,i-8].set_ylabel('price')
```



2.4

I will drop columns 'country' and 'street'. Because 'country' only has 'USA', which is not informative; 'street' has too many categories, which is also not so useful in modeling.

In [11]:
```
X = X.drop(columns=['country', 'street'])
cat_var = ['city', 'statezip']
categorical = X.dtypes == object
continuous = X.dtypes != object
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state
= 123)
```

In [13]:
```
pipe_cate = make_pipeline(SimpleImputer(strategy='constant', fill_valu
e='NA'),
                          OneHotEncoder(handle_unknown='ignore'))

pipe_cont = make_pipeline(StandardScaler(),
                          SimpleImputer())
```

In [14]:
```
pre_notscaled = make_column_transformer((pipe_cate, categorical),
                                        (SimpleImputer(), ~categorical)
)

pre_scaled = make_column_transformer((pipe_cont, ~categorical),
                                     (pipe_cate, categorical))

preprocessor = {'scaled':pre_scaled, 'not scaled': pre_notscaled}
```

In [15]:
```
regressor = {'OLS': LinearRegression(),
        'Ridge':Ridge(),
        'Lasso':Lasso(),
        'ElasticNet':ElasticNet()}

for regression_name, regression in regressor.items():
    for preprocess_name, preprocess in preprocessor.items():
        pipe = make_pipeline(preprocess, regression)
        mean_cv_score = np.mean(cross_val_score(pipe, X_train, y_train
)).round(5)
        print('The mean cross validation score of {} {} is {}'
              .format(preprocess_name, regression_name,mean_cv_score))
```

```
The mean cross validation score of scaled OLS is 0.75218
The mean cross validation score of not scaled OLS is 0.75146
The mean cross validation score of scaled Ridge is 0.75308
The mean cross validation score of not scaled Ridge is 0.50727
The mean cross validation score of scaled Lasso is 0.75223
The mean cross validation score of not scaled Lasso is 0.75223
The mean cross validation score of scaled ElasticNet is 0.57225
The mean cross validation score of not scaled ElasticNet is 0.57121
```

By comparing mean cross validation score of scaled and non scaled data, we observe that scaling does improve cv score for most of regressions, though some improvements are minor. Note that Lasso does not have any improvement by scaling data.

We will use scaled data for following questions.

2.5

In [16]:
```
##Ridge
param_r = {'ridge__alpha': np.logspace(-4, 4, 9)}
pipe_r =  make_pipeline(pre_scaled, Ridge())
grid_r = GridSearchCV(pipe_r, param_r, cv = 10,
                      return_train_score = True)
grid_r.fit(X_train, y_train)
print("Ridge: best parameters: ", grid_r.best_params_)
print("Ridge: best mean cross-validation score: ", grid_r.best_score_)
```

```
Ridge: best parameters:  {'ridge__alpha': 1.0}
Ridge: best mean cross-validation score:  0.7569730622905126
```

In [17]:
```
##Lasso
param_l = {'lasso__alpha':np.logspace(-3, 3, 7)}
pipe_l =  make_pipeline(pre_scaled, Lasso())
grid_l = GridSearchCV(pipe_l, param_l, cv = 10,
                      return_train_score = True)
grid_l.fit(X_train, y_train)
print("Lasso: best parameters: ", grid_l.best_params_)
print("Lasso: best mean cross-validation score: ", grid_l.best_score_)
```

```
Lasso: best parameters:  {'lasso__alpha': 100.0}
Lasso: best mean cross-validation score:  0.7568187591885047
```

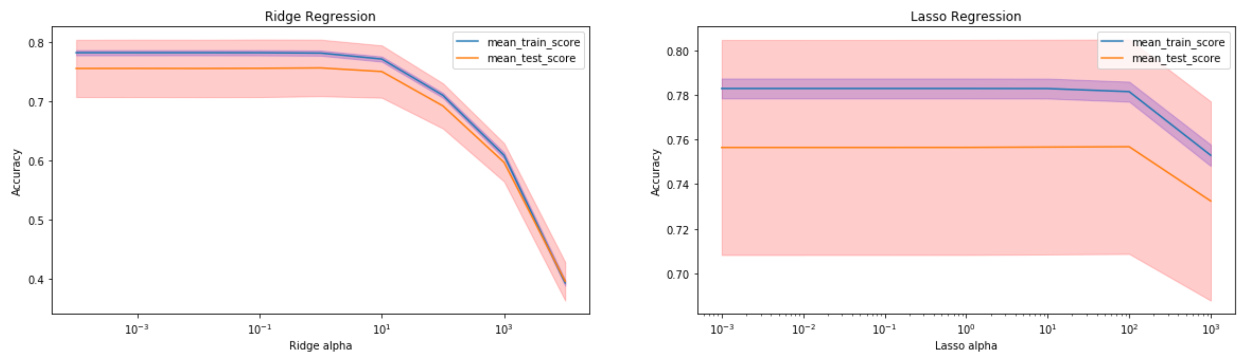
In [18]:
```
##ElasticNet
param_e = {'elasticnet__alpha':np.logspace(-3, 3, 7),
           'elasticnet__l1_ratio':np.arange(0, 1, 0.2)}
pipe_e =  make_pipeline(pre_scaled, ElasticNet())
grid_e = GridSearchCV(pipe_e, param_e, cv = 10,
                      return_train_score = True)
grid_e.fit(X_train, y_train)
print("ElasticNet: best parameters: ", grid_e.best_params_)
print("ElasticNet: best mean cross-validation score: ", grid_e.best_sc
ore_)
```

```
ElasticNet: best parameters:  {'elasticnet__alpha': 0.001, 'elasticn
et__l1_ratio': 0.4}
ElasticNet: best mean cross-validation score:  0.7574611047740805
```

In [19]:
```
#plot
figg, axx = plt.subplots(1,2, figsize=(20,5))
##Ridge
mean_train_scores_r = grid_r.cv_results_['mean_train_score']
std_train_scores_r  = grid_r.cv_results_['std_train_score']
```

```
mean_test_scores_r  = grid_r.cv_results_['mean_test_score']
std_test_scores_r   = grid_r.cv_results_['std_test_score']




axx[0].semilogx(np.logspace(-4, 4, 9), mean_train_scores_r, label = "m
ean_train_score");

axx[0].fill_between(np.logspace(-4, 4, 9),
                    mean_train_scores_r - std_train_scores_r,
                    mean_train_scores_r + std_train_scores_r,
                    alpha=0.2, color='b');

axx[0].semilogx(np.logspace(-4, 4, 9), mean_test_scores_r , label = "m
ean_test_score");

axx[0].fill_between(np.logspace(-4, 4, 9),
                       mean_test_scores_r - std_test_scores_r,
                       mean_test_scores_r + std_test_scores_r,
                       alpha=0.2, color='r');
axx[0].legend();
axx[0].set_xlabel('Ridge alpha');
axx[0].set_ylabel('Accuracy');
axx[0].set_title('Ridge Regression');



# LASSO plot
mean_train_scores_l = grid_l.cv_results_['mean_train_score']
std_train_scores_l  = grid_l.cv_results_['std_train_score']
mean_test_scores_l  = grid_l.cv_results_['mean_test_score']
std_test_scores_l   = grid_l.cv_results_['std_test_score']

axx[1].semilogx(np.logspace(-3, 3, 7), mean_train_scores_l, label = "m
ean_train_score");

axx[1].fill_between(np.logspace(-3, 3, 7),
                    mean_train_scores_l - std_train_scores_l,
                    mean_train_scores_l + std_train_scores_l,
                    alpha=0.2, color='b');

axx[1].semilogx(np.logspace(-3, 3, 7), mean_test_scores_l , label = "m
ean_test_score");

axx[1].fill_between(np.logspace(-3, 3, 7),
                       mean_test_scores_l - std_test_scores_l,
                       mean_test_scores_l + std_test_scores_l,
                       alpha=0.2, color='r');
axx[1].legend();
```

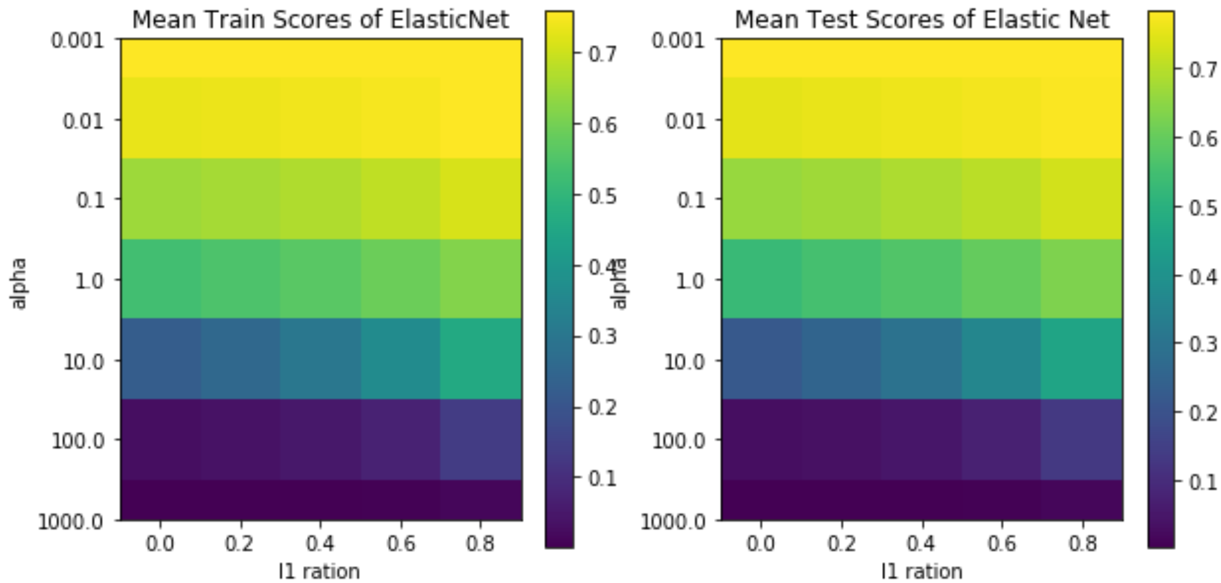
```
axx[1].set_xlabel('Lasso alpha');
axx[1].set_ylabel('Accuracy');
axx[1].set_title('Lasso Regression');
```



In [20]:

```
##ElasticNet
df_e=pd.DataFrame(grid_e.cv_results_).loc[:,['param_elasticnet__alpha'
,'param_elasticnet__l1_ratio',
         'mean_test_score','mean_train_score']]
fige, axe = plt.subplots(1,2,figsize=(10,5))
fige.colorbar(axe[0].imshow(np.array(df_e.iloc[:,2]).reshape(7,5)),
              ax=axe[0])
axe[0].set_xticks(np.arange(5))
axe[0].set_xticklabels(np.round(np.arange(0, 1, 0.2),1))
axe[0].set_yticks(np.arange(7))
axe[0].set_yticklabels(np.logspace(-3, 3, 7))
axe[0].set_title('Mean Train Scores of ElasticNet')
axe[0].set_xlabel('l1 ration')
axe[0].set_ylabel('alpha')

fige.colorbar(axe[1].imshow(np.array(df_e.iloc[:,-1]).reshape(7,5)),
              ax=axe[1])
axe[1].set_xticks(np.arange(5))
axe[1].set_xticklabels(np.round(np.arange(0, 1, 0.2),1))
axe[1].set_yticks(np.arange(7))
axe[1].set_yticklabels(np.logspace(-3, 3, 7))
axe[1].set_title('Mean Test Scores of Elastic Net')
axe[1].set_xlabel('l1 ration')
axe[1].set_ylabel('alpha')
```

Out[20]: Text(0, 0.5, 'alpha')

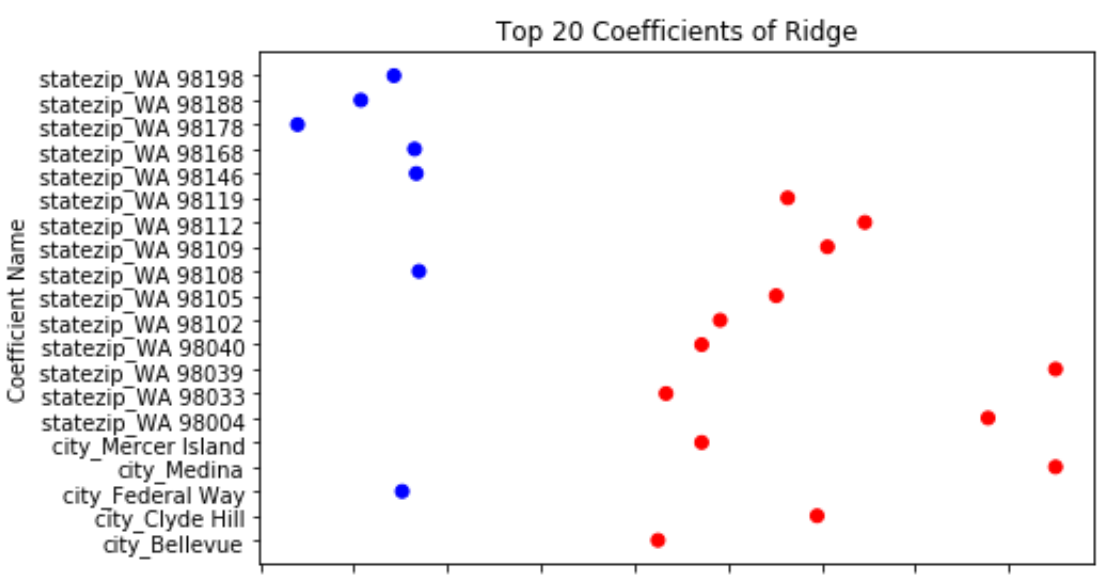


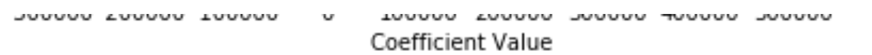
2.6

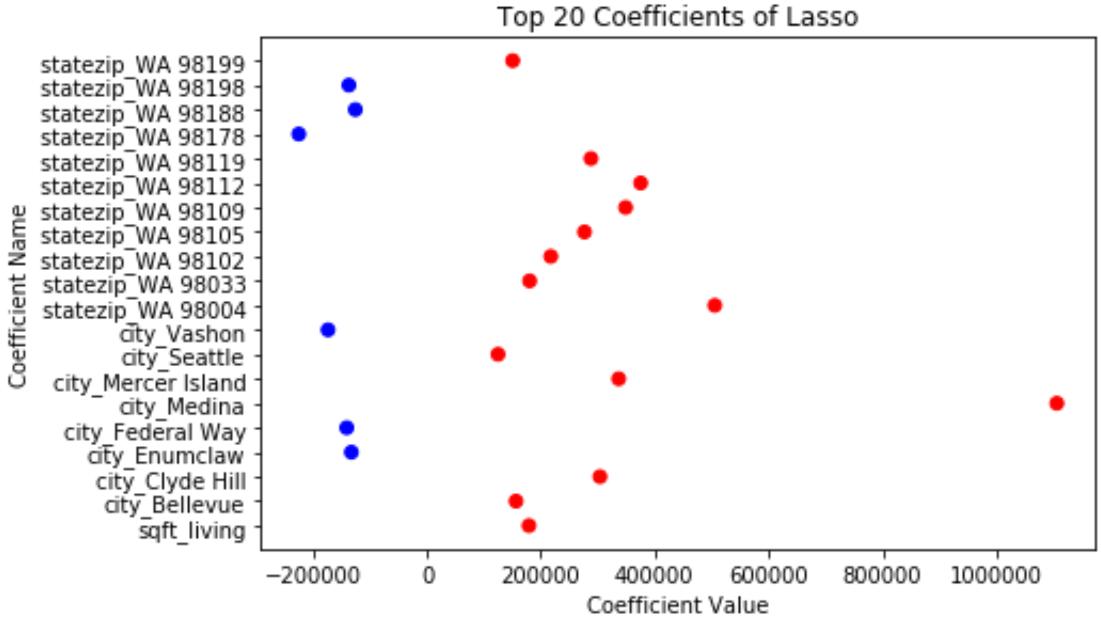
In [92]:

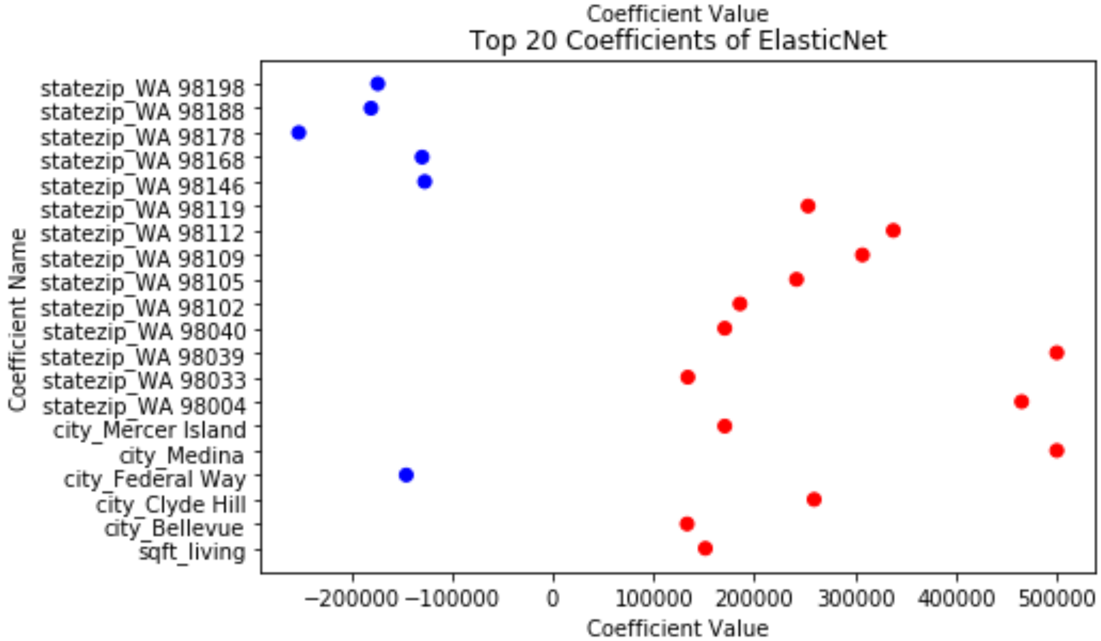
```
fig26, ax26 = plt.subplots(3, 1, figsize = (7, 15))
columns = pd.get_dummies(X).columns
#Ridge
coef_r = grid_r.best_estimator_[1].coef_[0]
idx_r = np.sort(np.abs(coef_r).argsort()[-20:][::-1])
ax26[0].scatter(coef_r[idx_r],columns[idx_r], c = np.sign(coef_r[idx_r
]), cmap='bwr')
ax26[0].set_title('Top 20 Coefficients of Ridge')
ax26[0].set_xlabel('Coefficient Value')
ax26[0].set_ylabel('Coefficient Name')


#Lasso
coef_l = grid_l.best_estimator_[1].coef_
idx_l = np.sort(np.abs(coef_l).argsort()[-20:][::-1])
ax26[1].scatter(coef_l[idx_l],columns[idx_l], c = np.sign(coef_l[idx_l
]), cmap='bwr')
ax26[1].set_title('Top 20 Coefficients of Lasso')
ax26[1].set_xlabel('Coefficient Value')
ax26[1].set_ylabel('Coefficient Name')


#ElasticNet
coef_e = grid_e.best_estimator_[1].coef_
idx_e = np.sort(np.abs(coef_e).argsort()[-20:][::-1])
ax26[2].scatter(coef_e[idx_e],columns[idx_e], c = np.sign(coef_e[idx_e
]), cmap='bwr')
ax26[2].set_title('Top 20 Coefficients of ElasticNet')
ax26[2].set_xlabel('Coefficient Value')
ax26[2].set_ylabel('Coefficient Name')
```

Out[92]: Text(0, 0.5, 'Coefficient Name')




Coefficient Value


Top 20 Coefficients of Lasso
statezip_WA 98199
statezip_WA 98198
statezip_WA 98188
statezip_WA 98178
statezip_WA 98119
statezip_WA 98112
statezip_WA 98109
statezip_WA 98105
statezip_WA 98102
statezip_WA 98033
statezip_WA 98004
city_Vashon
city_Seattle
city_Mercer Island
city_Medina
city_Federal Way
city_Enumclaw
city_Clyde Hill
city_Bellevue
sqft_living
Coefficient Name
−200000
0
200000
400000
600000
800000
1000000
Coefficient Value


Top 20 Coefficients of ElasticNet
statezip_WA 98198
statezip_WA 98188
statezip_WA 98178
statezip_WA 98168
statezip_WA 98146
statezip_WA 98119
statezip_WA 98112
statezip_WA 98109
statezip_WA 98105
statezip_WA 98102
statezip_WA 98040
statezip_WA 98039
statezip_WA 98033
statezip_WA 98004
city_Mercer Island
city_Medina
city_Federal Way
city_Clyde Hill
city_Bellevue
sqft_living
Coefficient Name
−200000
−100000
0
100000
200000
300000
400000
500000
Coefficient Value

In [ ]: