Model Build

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import Lasso, LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVR
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
pd.pandas.set_option('display.max_columns', None)
```

In [2]:
```
# chargement des datasets
X_train = pd.read_csv('Data/xtrain.csv')
X_test = pd.read_csv('Data/xtest.csv')

X_train.head()
```

Out[2]:

	id_mutation	id_parcelle	id_bien	date_mutation	adresse_nom_voie	nom_commune	valeur_fonciere	nature_mutation	code_
0	2017-1381514	95018000AV0057	95018000AV0057-95	2017-05-16	RUE DE ST QUENTIN	Argenteuil	12.354493	0.666667	
1	2017-131542	132098460A0288	132098460A0288-13	2017-04-07	RUE ANTOINE FORTUNE MARION	Marseille 9e Arrondissement	13.075272	0.666667	
2	2017-1162525	83038000AB0022	83038000AB0022-83	2017-05-22	SAINTE ANNE	Châteaudouble	11.652687	0.666667	
3	2019-173403	44109000NY0325	44109000NY0325-44	2019-03-29	RUE FELIX LEMOINE	Nantes	9.510445	0.666667	
4	2017-242501	22011000AB0237	22011000AB0237-22	2017-04-27	LE BOURG	Boqueho	8.006368	0.666667	

In [3]:
```
# récupération de la target
y_train = X_train['valeur_fonciere']
y_test = X_test['valeur_fonciere']
```

In [4]:
```
# chargement de la liste feature selection

features = pd.read_csv('Data/selected_features.csv', header=None)

features = [x for x in features[0]]

features = features + ['nombre_pieces_principales']

features
```

Out[4]:
```
['nature_mutation',
 'code_departement',
 'code_type_local',
 'type_local',
 'surface_reelle_bati',
 'surface_terrain',
 'nombre_pieces_principales']
```

In [5]:
```
# reduction du xtrain & xtest avec la feature selection

X_train = X_train[features]
X_test = X_test[features]
```

Regularised linear regression

In [6]:
```
lin_model = Lasso(alpha=0.005, random_state=123) # remember to set the random_state / seed
lin_model.fit(X_train, y_train)
```

Out[6]:
```
Lasso(alpha=0.005, random_state=123)
```

In [7]:

```
# evaluation du model

pred = lin_model.predict(X_train)
print('linear train mse: {}'.format(mean_squared_error(np.exp(y_train), np.exp(pred))))
print('linear train rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_train), np.exp(pred)))))
print('linear train r2 score: {}'.format(r2_score(np.exp(y_train), np.exp(pred))))
print()
pred = lin_model.predict(X_test)
print('linear test mse: {}'.format(mean_squared_error(np.exp(y_test), np.exp(pred))))
print('linear test rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_test), np.exp(pred)))))
print('linear train r2 score: {}'.format(r2_score(np.exp(y_test), np.exp(pred))))
print()
print('Average house price: ', np.exp(y_train).median())
```

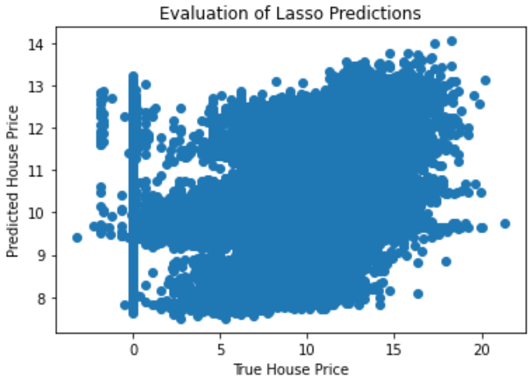
```
linear train mse: 3952014288849.5107
linear train rmse: 1987967.3762035207
linear train r2 score: 0.00041028706283052774

linear test mse: 8289962613220.356
linear test rmse: 2879229.5172876297
linear train r2 score: 0.00014883923373709695

Average house price:  120000.00000000028
```

In [8]:

```
# visualisation des résultats
plt.scatter(y_test, lin_model.predict(X_test))
plt.xlabel('True House Price')
plt.ylabel('Predicted House Price')
plt.title('Evaluation of Lasso Predictions')
```

Out[8]:

```
Text(0.5, 1.0, 'Evaluation of Lasso Predictions')
```



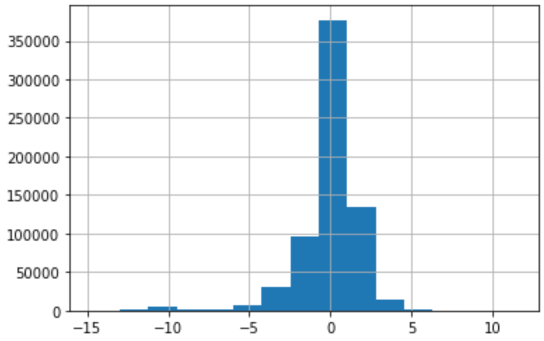
We can see that our model is doing a pretty good job at estimating house prices.

In [9]:

```
# distribution des erreurs

errors = y_test - lin_model.predict(X_test)
errors.hist(bins=15)
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13e3e3d5f60>
```



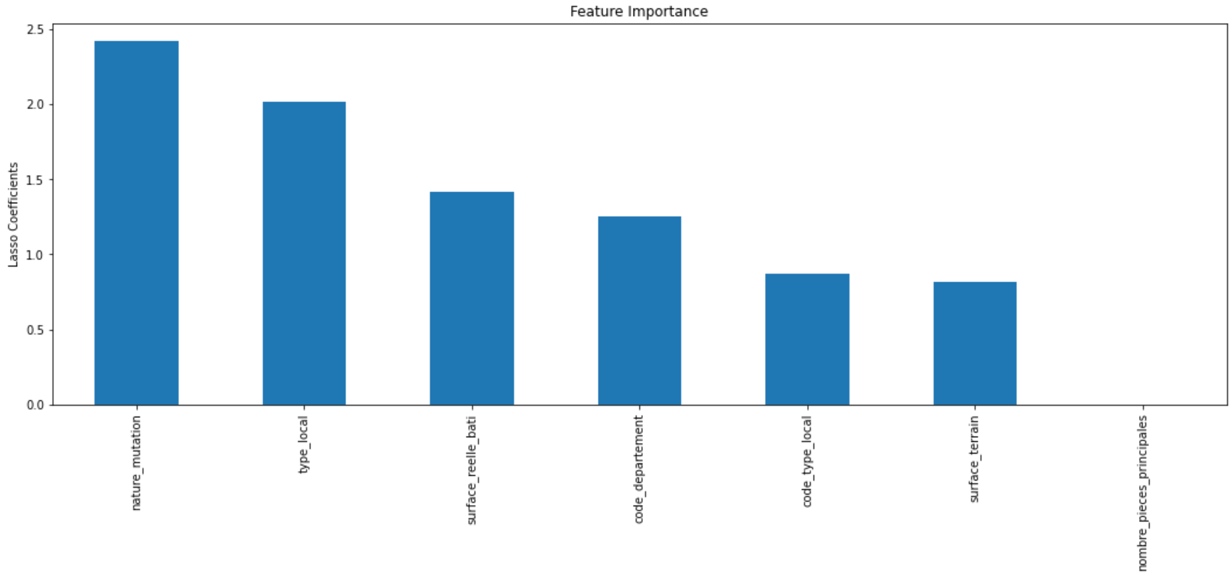
Feature importance

In [10]:

```
importance = pd.Series(np.abs(lin_model.coef_.ravel()))
importance.index = features
importance.sort_values(inplace=True, ascending=False)
importance.plot.bar(figsize=(18,6))
plt.ylabel('Lasso Coefficients')
plt.title('Feature Importance')
```

Out[10]:

```
Text(0.5, 1.0, 'Feature Importance')
```



Linear Regression

In [11]:

```
linreg_model = LinearRegression()
linreg_model.fit(X_train, y_train)
```

Out[11]:

```
LinearRegression()
```

In [12]:

```
predlr = linreg_model.predict(X_train)
print('linear regression train mse: {}'.format(mean_squared_error(np.exp(y_train), np.exp(predlr))))
print('linear regression train rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_train), np.exp(predlr)))))
print('linear regression train r2 score: {}'.format(r2_score(np.exp(y_train), np.exp(predlr))))
print()
predlr = linreg_model.predict(X_test)
print('linear regression test mse: {}'.format(mean_squared_error(np.exp(y_test), np.exp(predlr))))
print('linear regression test rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_test), np.exp(predlr)))))
print('linear regression train r2 score: {}'.format(r2_score(np.exp(y_test), np.exp(predlr))))
print()
print('Average house price: ', np.exp(y_train).median())
```

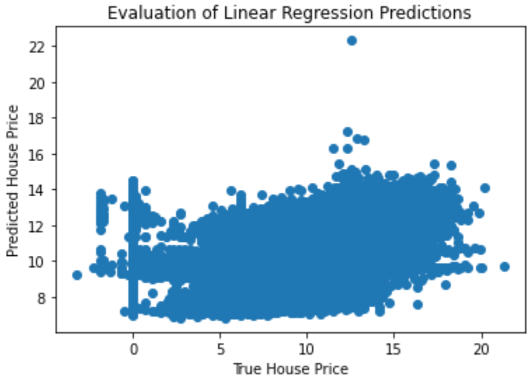
```
linear regression train mse: 4088423032683.79
linear regression train rmse: 2021984.9239506684
linear regression train r2 score: -0.03409180911534948

linear regression test mse: 44196069506402.2
linear regression test rmse: 6648012.447822447
linear regression train r2 score: -4.330481385623116

Average house price:  120000.00000000028
```

In [13]:

```
plt.scatter(y_test, linreg_model.predict(X_test))
plt.xlabel('True House Price')
plt.ylabel('Predicted House Price')
plt.title('Evaluation of Linear Regression Predictions')
```

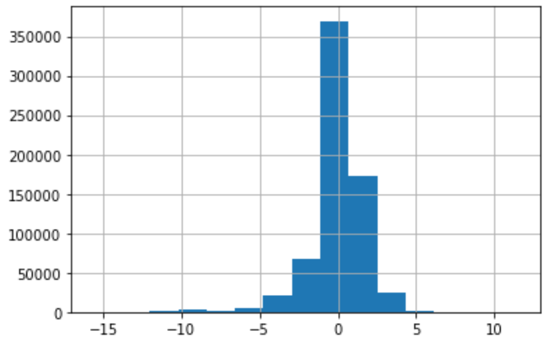
Out[13]:

```
Text(0.5, 1.0, 'Evaluation of Linear Regression Predictions')
```



In [14]:
```
errors = y_test - linreg_model.predict(X_test)
errors.hist(bins=15)
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x13e3c577128>
```



Ridge

In [15]:
```
ridge_model = Ridge()
ridge_model.fit(X_train, y_train)
```

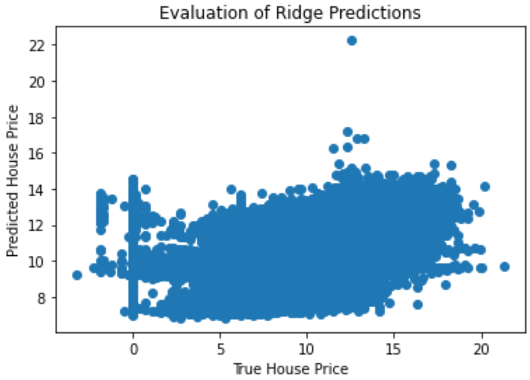
Out[15]:
```
Ridge()
```

In [16]:
```
predR = ridge_model.predict(X_train)
print('ridge train mse: {}'.format(mean_squared_error(np.exp(y_train), np.exp(predR))))
print('ridge train rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_train), np.exp(predR)))))
print('ridge train r2 score: {}'.format(r2_score(np.exp(y_train), np.exp(predR))))
print()
predR = ridge_model.predict(X_test)
print('ridge test mse: {}'.format(mean_squared_error(np.exp(y_test), np.exp(predR))))
print('ridge test rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_test), np.exp(predR)))))
print('ridge train r2 score: {}'.format(r2_score(np.exp(y_test), np.exp(predR))))
print()
print('Average house price: ', np.exp(y_train).median())
```

```
ridge train mse: 4073351328631.8604
ridge train rmse: 2018254.5252350755
ridge train r2 score: -0.03027969730966862

ridge test mse: 39701144688342.27
ridge test rmse: 6300884.436993133
ridge train r2 score: -3.788349170246521

Average house price:  120000.00000000028
```

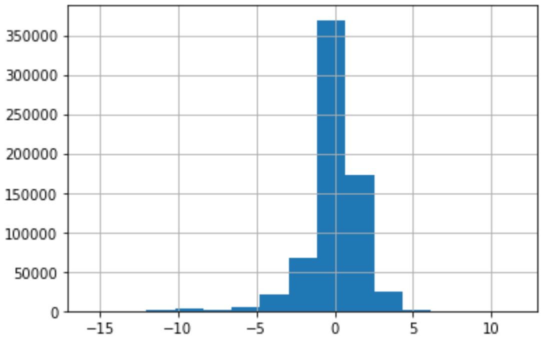
In [17]:
```
plt.scatter(y_test, ridge_model.predict(X_test))
plt.xlabel('True House Price')
plt.ylabel('Predicted House Price')
plt.title('Evaluation of Ridge Predictions')
```

Out[17]:
```
Text(0.5, 1.0, 'Evaluation of Ridge Predictions')
```



In [18]:
```
errors = y_test - ridge_model.predict(X_test)
errors.hist(bins=15)
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x13e3b3b9d30>
```



Decision Tree Regressor

In [19]:
```
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
```

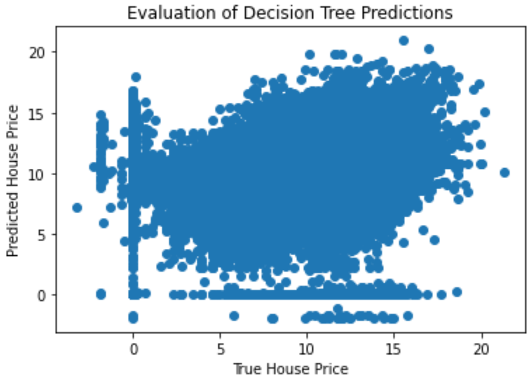
Out[19]:
```
DecisionTreeRegressor()
```

In [20]:
```
preddt = dt_model.predict(X_train)
print('decision tree train mse: {}'.format(mean_squared_error(np.exp(y_train), np.exp(preddt))))
print('decision tree train rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_train), np.exp(preddt)))))
print('decision tree train r2 score: {}'.format(r2_score(np.exp(y_train), np.exp(preddt))))
print()
preddt = dt_model.predict(X_test)
print('decision tree test mse: {}'.format(mean_squared_error(np.exp(y_test), np.exp(preddt))))
print('decision tree test rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_test), np.exp(preddt)))))
print('decision tree train r2 score: {}'.format(r2_score(np.exp(y_test), np.exp(preddt))))
print()
print('Average house price: ', np.exp(y_train).median())
```

```
decision tree train mse: 1328018266535.8845
decision tree train rmse: 1152396.7487527395
decision tree train r2 score: 0.6641020753474125

decision tree test mse: 12855740852031.67
decision tree test rmse: 3585490.3224010617
decision tree train r2 score: -0.5505289967068832

Average house price:  120000.00000000028
```

In [21]:
```
plt.scatter(y_test, dt_model.predict(X_test))
plt.xlabel('True House Price')
plt.ylabel('Predicted House Price')
plt.title('Evaluation of Decision Tree Predictions')
```

Out[21]:
```
Text(0.5, 1.0, 'Evaluation of Decision Tree Predictions')
```

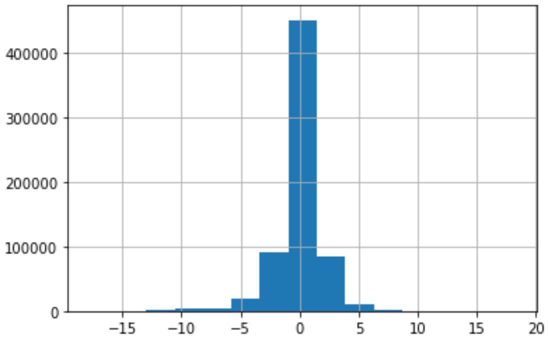


In [22]:

```
errors = y_test - dt_model.predict(X_test)
errors.hist(bins=15)
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13e3e0d12e8>
```



XGBoost

In [24]:

```
xgb_model = xgb.XGBRegressor()
xgb_model.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\IPython\core\formatters.py in __call__(self, obj,
 include, exclude)
    968 
    969             if method is not None:
--> 970                 return method(include=include, exclude=exclude)
    971             return None
    972         else:

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\base.py in _repr_mimebundle_(self, **kwarg
s)
    461     def _repr_mimebundle_(self, **kwargs):
    462         """Mime bundle used by jupyter kernels to display estimator"""
--> 463         output = {"text/plain": repr(self)}
    464         if get_config()["display"] == 'diagram':
    465             output["text/html"] = estimator_html_repr(self)

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\base.py in __repr__(self, N_CHAR_MAX)
    277             n_max_elements_to_show=N_MAX_ELEMENTS_TO_SHOW)
    278 
--> 279         repr_ = pp.pformat(self)
    280 
    281         # Use bruteforce ellipsis when there are a lot of non-blank characters

~\AppData\Local\Programs\Python\Python36\lib\pprint.py in pformat(self, object)
    142     def pformat(self, object):
    143         sio = _StringIO()
--> 144         self._format(object, sio, 0, 0, {}, 0)
    145         return sio.getvalue()
    146 

~\AppData\Local\Programs\Python\Python36\lib\pprint.py in _format(self, object, stream, indent, allowance, contex
t, level)
    159             self._readable = False
    160             return
--> 161         rep = self._repr(object, context, level)
    162         max_width = self._width - indent - allowance
    163         if len(rep) > max_width:

~\AppData\Local\Programs\Python\Python36\lib\pprint.py in _repr(self, object, context, level)
    391     def _repr(self, object, context, level):
    392         repr, readable, recursive = self.format(object, context.copy(),
--> 393                                                 self._depth, level)
    394         if not readable:
    395             self._readable = False

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\utils\_pprint.py in format(self, object, c
ontext, maxlevels, level)
    168     def format(self, object, context, maxlevels, level):
    169         return _safe_repr(object, context, maxlevels, level,
--> 170                           changed_only=self._changed_only)
    171 
    172     def _pprint_estimator(self, object, stream, indent, allowance, context,

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\utils\_pprint.py in _safe_repr(object, con
text, maxlevels, level, changed_only)
    412         recursive = False
    413         if changed_only:
--> 414             params = _changed_params(object)
    415         else:
    416             params = object.get_params(deep=False)

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\utils\_pprint.py in _changed_params(estima
tor)
     96     init_params = {name: param.default for name, param in init_params.items()}
     97     for k, v in params.items():
---> 98         if (repr(v) != repr(init_params[k]) and
     99                 not (is_scalar_nan(init_params[k]) and is_scalar_nan(v))):
    100             filtered_params[k] = v

KeyError: 'base_score'
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\IPython\core\formatters.py in __call__(self, obj)
    700                 type_pprinters=self.type_printers,
    701                 deferred_pprinters=self.deferred_printers)
--> 702             printer.pretty(obj)
    703             printer.flush()
    704             return stream.getvalue()

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\IPython\lib\pretty.py in pretty(self, obj)
    392                         if cls is not object \
    393                                 and callable(cls.__dict__.get('__repr__')):
--> 394                             return _repr_pprint(obj, self, cycle)
    395 
    396             return _default_pprint(obj, self, cycle)

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\IPython\lib\pretty.py in _repr_pprint(obj, p, cycl
e)
    698     """A pprint that just redirects to the normal repr function."""
    699     # Find newlines and replace them with p.break_()
--> 700     output = repr(obj)
    701     lines = output.splitlines()
    702     with p.group():

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\base.py in __repr__(self, N_CHAR_MAX)
    277             n_max_elements_to_show=N_MAX_ELEMENTS_TO_SHOW)
    278 
--> 279         repr_ = pp.pformat(self)
    280 
    281         # Use bruteforce ellipsis when there are a lot of non-blank characters

~\AppData\Local\Programs\Python\Python36\lib\pprint.py in pformat(self, object)
    142     def pformat(self, object):
    143         sio = _StringIO()
--> 144         self._format(object, sio, 0, 0, {}, 0)
    145         return sio.getvalue()
    146 

~\AppData\Local\Programs\Python\Python36\lib\pprint.py in _format(self, object, stream, indent, allowance, contex
t, level)
    159             self._readable = False
    160             return
--> 161         rep = self._repr(object, context, level)
    162         max_width = self._width - indent - allowance
    163         if len(rep) > max_width:

~\AppData\Local\Programs\Python\Python36\lib\pprint.py in _repr(self, object, context, level)
    391     def _repr(self, object, context, level):
    392         repr, readable, recursive = self.format(object, context.copy(),
--> 393                                                 self._depth, level)
    394         if not readable:
    395             self._readable = False

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\utils\_pprint.py in format(self, object, c
ontext, maxlevels, level)
    168     def format(self, object, context, maxlevels, level):
    169         return _safe_repr(object, context, maxlevels, level,
--> 170                           changed_only=self._changed_only)
    171 
    172     def _pprint_estimator(self, object, stream, indent, allowance, context,

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\utils\_pprint.py in _safe_repr(object, con
text, maxlevels, level, changed_only)
    412         recursive = False
    413         if changed_only:
--> 414             params = _changed_params(object)
    415         else:
    416             params = object.get_params(deep=False)

c:\users\amand\desktop\projetecolev2\projetv2\lib\site-packages\sklearn\utils\_pprint.py in _changed_params(estima
tor)
     96     init_params = {name: param.default for name, param in init_params.items()}
     97     for k, v in params.items():
---> 98         if (repr(v) != repr(init_params[k]) and
     99                 not (is_scalar_nan(init_params[k]) and is_scalar_nan(v))):
    100             filtered_params[k] = v

KeyError: 'base_score'
```

In [25]:

```
predxgb = xgb_model.predict(X_train)
print('xgboost train mse: {}'.format(mean_squared_error(np.exp(y_train), np.exp(predxgb))))
print('xgboost train rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_train), np.exp(predxgb)))))
print('xgboost train r2 score: {}'.format(r2_score(np.exp(y_train), np.exp(predxgb))))
print()
predxgb = xgb_model.predict(X_test)
print('xgboost test mse: {}'.format(mean_squared_error(np.exp(y_test), np.exp(predxgb))))
print('xgboost test rmse: {}'.format(sqrt(mean_squared_error(np.exp(y_test), np.exp(predxgb)))))
print('xgboost train r2 score: {}'.format(r2_score(np.exp(y_test), np.exp(predxgb))))
print()
print('Average house price: ', np.exp(y_train).median())
```

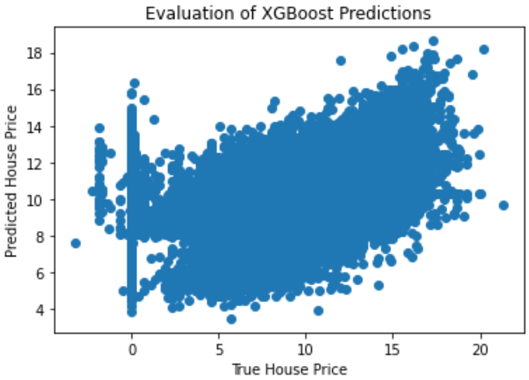
```
xgboost train mse: 3625338870624.5947
xgboost train rmse: 1904032.2661721348
xgboost train r2 score: 0.08303685763176794

xgboost test mse: 8095664586021.379
xgboost test rmse: 2845288.1376095074
xgboost train r2 score: 0.023583095465446013

Average house price:  120000.00000000028
```

In [26]:

```
plt.scatter(y_test, xgb_model.predict(X_test))
plt.xlabel('True House Price')
plt.ylabel('Predicted House Price')
plt.title('Evaluation of XGBoost Predictions')
```

Out[26]:

```
Text(0.5, 1.0, 'Evaluation of XGBoost Predictions')
```

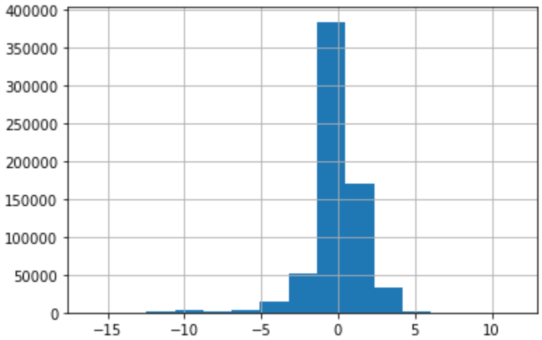


In [27]:

```
errors = y_test - xgb_model.predict(X_test)
errors.hist(bins=15)
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x13e065c72e8>
```



In [ ]: