zillow_MWS

October 13, 2017

1 Zillow prize data analysis report

```
In [1]: from datetime import datetime
        d = datetime.now().date()
        t = datetime.now().strftime('%H:%M:%S')
        print("This report was last updated on", d, "at", t)
```

```
This report was last updated on 2017-10-12 at 11:52:31
```

1.1 Introduction

The Zillow Prize is a Kaggle competition that aims to inspire data scientists around the world to improve the accuracy of the Zillow "Zestimate" statistical and machine learning models.

My goal is to compete for the Zillow prize and write up my results.

1.2 Methods

1.2.1 Data

The data were obtained from Kaggle website and consist of the following files: - `properties_2016.csv.zip` - `properties_2017.csv.zip` - `sample_submission.csv` - `train_2016_v2.csv.zip` - `train_2017.csv.zip` - `zillow_data_dictionary.xlsx` The `zillow_data_dictionary.xlsx` is a code book that explains the data. This data will be made available on figshare to provide an additional source if the Kaggle site data become unavailable.

1.2.2 Analysis

Data analysis was done in Jupyter Notebook (Pérez and Granger 2007)[**?**] Integrated Development Environment using the Python language (Pérez, Granger, and Hunter 2011)[**?**] and a number of software packages:

- NumPy (van der Walt, Colbert, and Varoquaux 2011)[**?**]
- pandas (McKinney 2010)[**?**]
- scikit-learn (Pedregosa et al. 2011)[**?**]

1.2.3 Visualization

The following packages were used to visualize the data:

- Matplotlib (Hunter 2007)[**?**]
- Seaborn (Waskom et al. 2014)[**?**]
- r-ggplot2
- r-cowplot

The use of `R` code and packages in a `Python` environment is possible through the use of the `Rpy2` package.

1.2.4 Prediction

Machine learning prediction was done using the following packages:

- scikit-learn (Pedregosa et al. 2011)[**?**]
- xgboost
- r-caret

1.2.5 Reproducibility

Reproducibility is extremely important in scientific research yet many examples of problematic studies exist in the literature (Couzin-Frankel 2010)[**?**].

The names and versions of each package used herein are listed in the accompanying `env.yml` file in the `config` folder. The computational environment used to analyze the data can be recreated using this `env.yml` file and the `conda` package and environment manager available as part of the Anaconda distribution of Python.

Additionally, details on how to setup a Docker image capable of running the analysis is included in the `README.md` file in the `config` folder.

The code in the form of a jupyter notebook (`01_zillow_MWS.ipynb`) or Python script (`01_zillow_MWS.py`), can also be run on the Kaggle website (this requires logging in with a username and password).

More information on the details of how this project was created and the computational environment was configured can be found in the accompanying `README.md` file.

This Python 3 environment comes with many helpful analytics libraries installed It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python (a modified version of this docker image will be made available as part of my project to ensure reproducibility). For example, here's several helpful packages to load in

1.3 Results

1.3.1 Import Libraries and Data

Input data files are available in the "../input/" directory.

Any results I write to the current directory are saved as output.

```
In [2]: import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
        import matplotlib.pyplot as plt # data visualization
        import datetime as dt
        import seaborn as sns
        import xgboost as xgb
        from sklearn.preprocessing import LabelEncoder
        from sklearn.ensemble import RandomForestRegressor
        from sklearn import ensemble
        %matplotlib inline
        ### Seaborn style
        sns.set_style("whitegrid")
```

```
/Users/marskar/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of the
model_selection module into which all the refactored classes and functions are moved.
Also note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

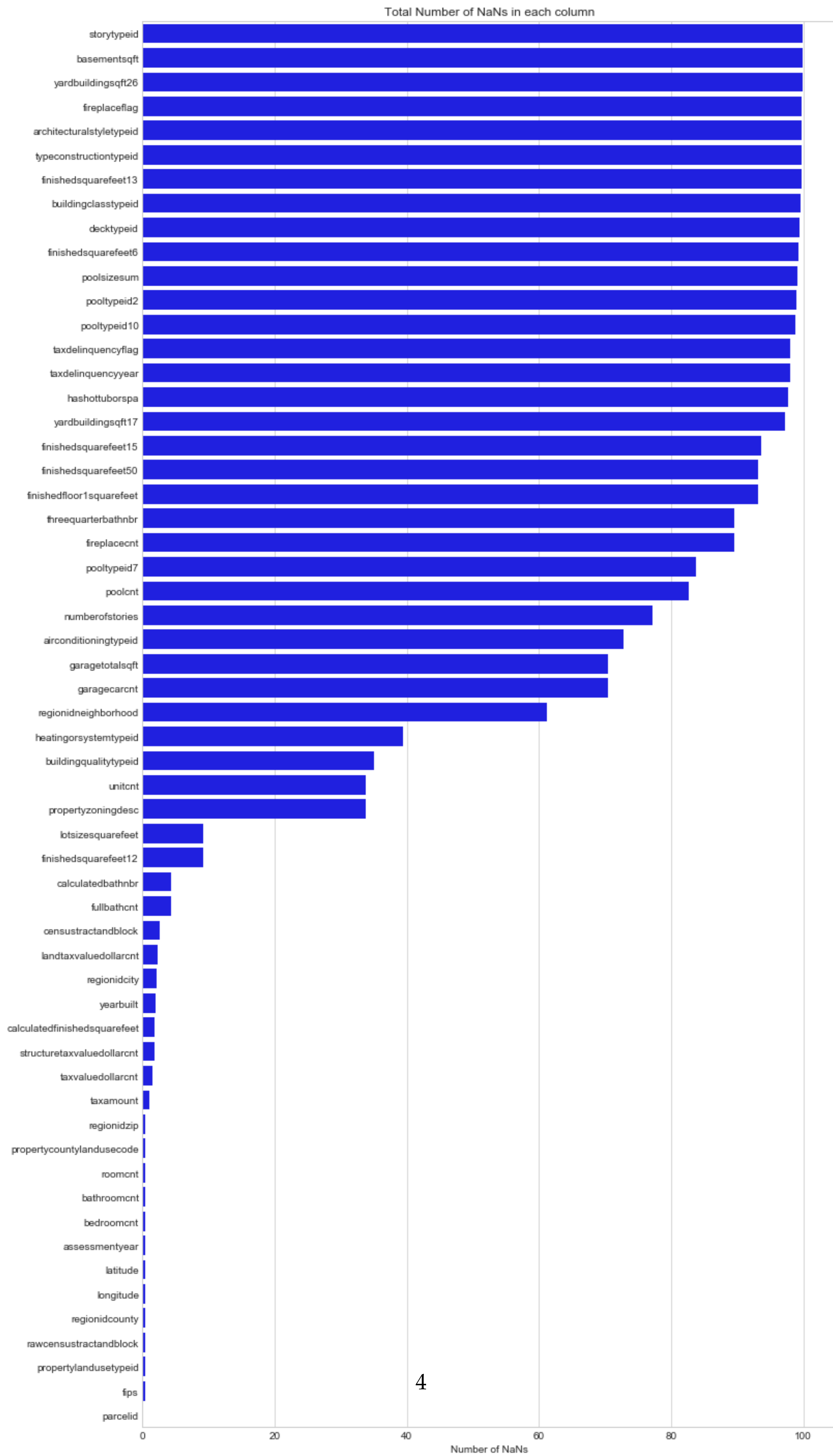
```
In [3]: prop = pd.read_csv("../input/properties_2016.csv")
        prop.shape
```

```
/Users/marskar/anaconda3/lib/python3.6/site-
packages/IPython/core/interactiveshell.py:2728: DtypeWarning: Columns (22,32,34,49,55)
have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
In [4]: ### ... check for NaNs
        nan = prop.isnull().sum()/len(prop)*100
        nan
```

```
In [5]: ### Plotting NaN counts
        nan_sorted = nan.sort_values(ascending=False).to_frame().reset_index()
        nan_sorted.columns = ['Column', 'Number of NaNs']
```

```
In [6]: fig, ax = plt.subplots(figsize=(12, 25))
        sns.barplot(x="Number of NaNs", y="Column", data=nan_sorted, color='Blue', ax=ax)
        ax.set(xlabel="Number of NaNs", ylabel="", title="Total Number of NaNs in each column")
        plt.show()
```


Total Number of NaNs in each column
storytypeid
basementsqft
yardbuildingsqft26
fireplaceflag
architecturalstyletypeid
typeconstructiontypeid
finishedsquarefeet13
buildingclasstypeid
decktypeid
finishedsquarefeet6
poolsizesum
pooltypeid2
pooltypeid10
taxdelinquencyflag
taxdelinquencyyear
hashottuborspa
yardbuildingsqft17
finishedsquarefeet15
finishedsquarefeet50
finishedfloor1squarefeet
threequarterbathnbr
fireplacecnt
pooltypeid7
poolcnt
numberofstories
airconditioningtypeid
garagetotalsqft
garagecarcnt
regionidneighborhood
heatingorsystemtypeid
buildingqualitytypeid
unitcnt
propertyzoningdesc
lotsizesquarefeet
finishedsquarefeet12
calculatedbathnbr
fullbathcnt
censustractandblock
landtaxvaluedollarcnt
regionidcity
yearbuilt
calculatedfinishedsquarefeet
structuretaxvaluedollarcnt
taxvaluedollarcnt
taxamount
regionidzip
propertycountylandusecode
roomcnt
bathroomcnt
bedroomcnt
assessmentyear
latitude
longitude
regionidcounty
rawcensustractandblock
propertylandusetypeid
fips
parcelid
0
20
40
60
80
100
Number of NaNs

There are several columns which have a very high proportion of missing values. It may be worth analysing these more closely.

Feature Importance by Random Forest

In [7]:
```
train = pd.read_csv("../input/train_2016_v2.csv", parse_dates=["transactiondate"])
train.shape
```

In [8]:
```
train['transaction_month'] = pd.DatetimeIndex(train['transactiondate']).month
train.sort_values('transaction_month', axis=0, ascending=True, inplace=True)
```

Feature Importance

In [9]:
```
#fill NaN values with -1 and encode object columns
for x in prop.columns:
    prop[x] = prop[x].fillna(-1)

#many more parcelids in properties file, merge with training file
train = pd.merge(train, prop, on='parcelid', how='left')
```

In [10]:
```
for c in train[['transactiondate', 'hashottuborspa', 'propertycountylandusecode',
 'propertyzoningdesc', 'fireplaceflag', 'taxdelinquencyflag']]:
    label = LabelEncoder()
    label.fit(list(train[c].values))
    train[c] = label.transform(list(train[c].values))

x_train = train.drop(['parcelid', 'logerror', 'transactiondate'], axis=1)
y_train = train['logerror']
```

In [11]:
```
rf = RandomForestRegressor(n_estimators=30, max_features=None)
rf.fit(x_train, y_train)
```

In [12]:
```
rf_importance = rf.feature_importances_
rf_importance_df = pd.DataFrame()
rf_importance_df['features'] = x_train.columns
rf_importance_df['importance'] = rf_importance
print(rf_importance_df.head())
```

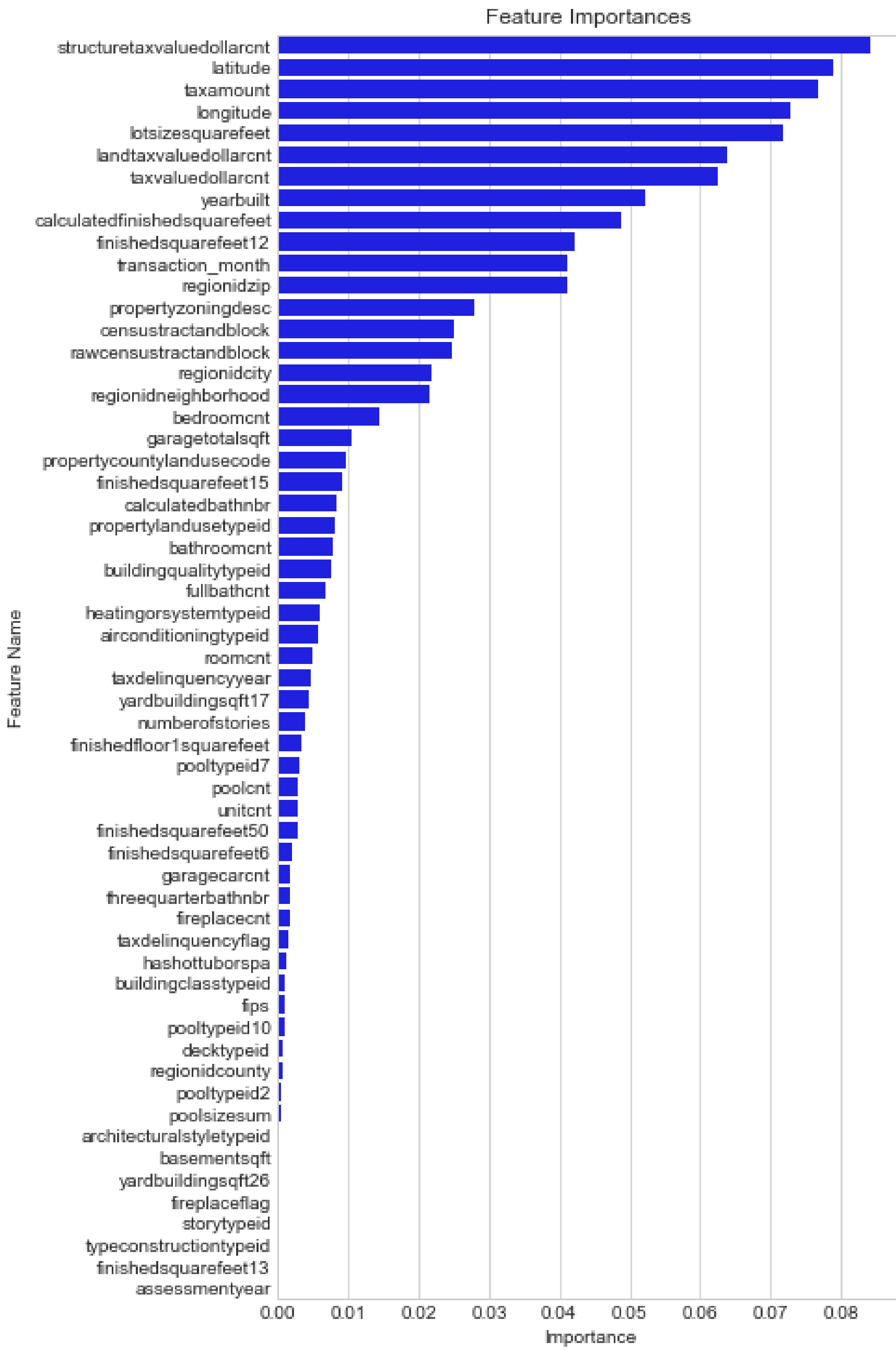
```
                  features  importance
0        transaction_month    0.041078
1    airconditioningtypeid    0.005809
2  architecturalstyletypeid    0.000262
3             basementsqft    0.000238
4              bathroomcnt    0.007725
```

In [13]:
```
rf_importance_df.sort_values('importance', axis=0, inplace=True, ascending=False)

print(rf_importance_df.head())
```

```
                      features  importance
50  structuretaxvaluedollarcnt    0.084278
24                    latitude    0.078829
54                   taxamount    0.076839
25                   longitude    0.072749
26           lotsizesquarefeet    0.071840
```

In [14]:
```
fig, ax = plt.subplots(figsize=(6, 12.5))
sns.barplot(x="importance", y="features", data=rf_importance_df, color='Blue', ax=ax)
ax.set(xlabel="Importance", ylabel="Feature Name", title="Feature Importances")
plt.show()
```


Feature Importances
structuretaxvaluedollarcnt
latitude
taxamount
longitude
lotsizesquarefeet
landtaxvaluedollarcnt
taxvaluedollarcnt
yearbuilt
calculatedfinishedsquarefeet
finishedsquarefeet12
transaction_month
regionidzip
propertyzoningdesc
censustractandblock
rawcensustractandblock
regionidcity
regionidneighborhood
bedroomcnt
garagetotalsqft
propertycountylandusecode
finishedsquarefeet15
calculatedbathnbr
propertylandusetypeid
bathroomcnt
buildingqualitytypeid
fullbathcnt
heatingorsystemtypeid
airconditioningtypeid
roomcnt
taxdelinquencyyear
yardbuildingsqft17
numberofstories
finishedfloor1squarefeet
pooltypeid7
poolcnt
unitcnt
finishedsquarefeet50
finishedsquarefeet6
garagecarcnt
threequarterbathnbr
fireplacecnt
taxdelinquencyflag
hashottuborspa
buildingclasstypeid
fips
pooltypeid10
decktypeid
regionidcounty
pooltypeid2
poolsizesum
architecturalstyletypeid
basementsqft
yardbuildingsqft26
fireplaceflag
storytypeid
typeconstructiontypeid
finishedsquarefeet13
assessmentyear
Feature Name
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
Importance

In [15]:
```
etr = ensemble.ExtraTreesRegressor(n_estimators=25, max_depth=30, max_features=0.3,
n_jobs=-1, random_state=0)
etr.fit(x_train, y_train)
```

In [16]:
```
etr_importance = etr.feature_importances_
etr_importance_df = pd.DataFrame()
etr_importance_df['features'] = x_train.columns
etr_importance_df['importance'] = etr_importance
print(etr_importance_df.head())
```

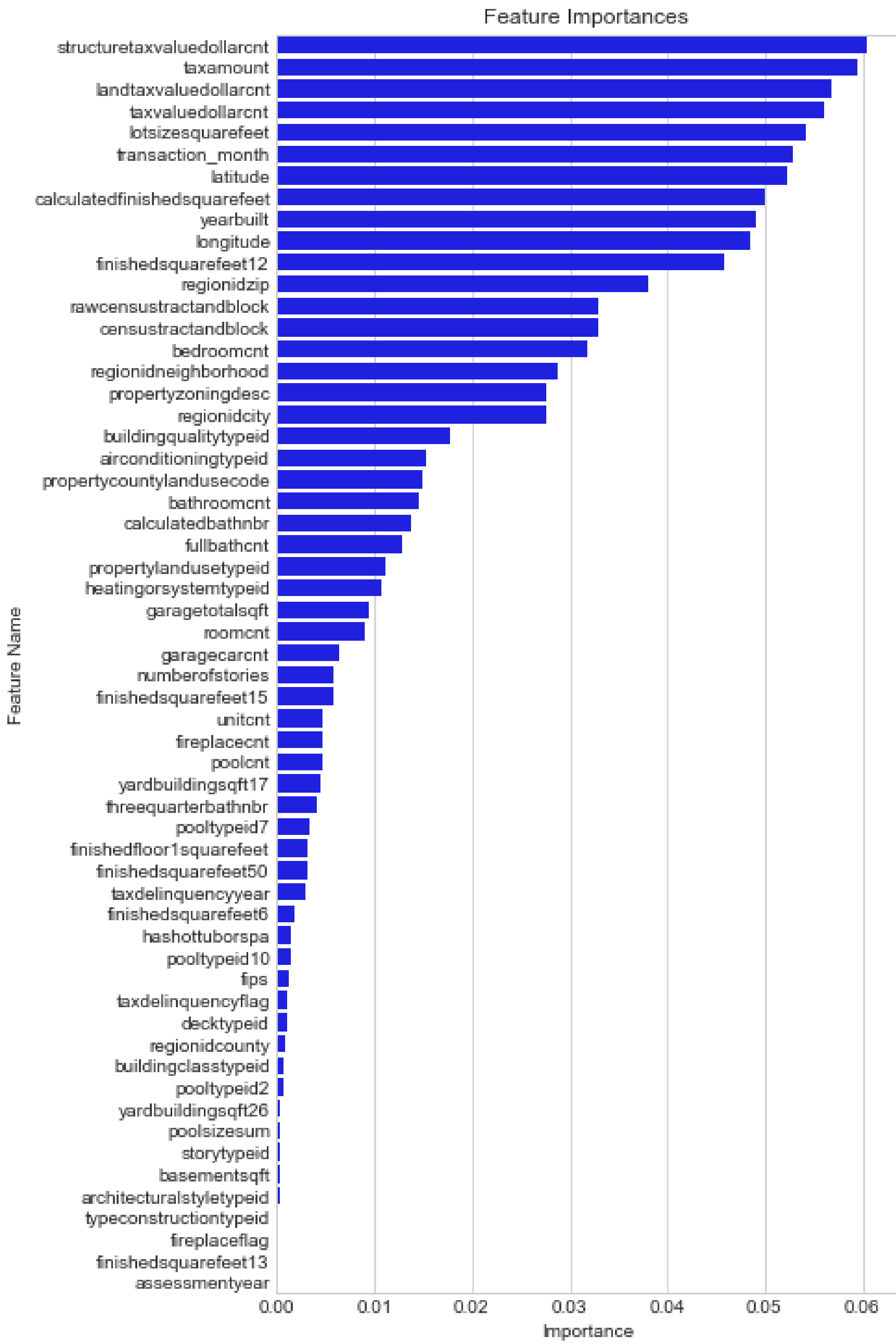
```
                 features  importance
0       transaction_month    0.052949
1    airconditioningtypeid    0.015291
2  architecturalstyletypeid    0.000204
3            basementsqft    0.000212
4             bathroomcnt    0.014486
```

In [17]:
```
etr_importance_df.sort_values('importance', axis=0, inplace=True, ascending=False)

print(etr_importance_df.head())
```

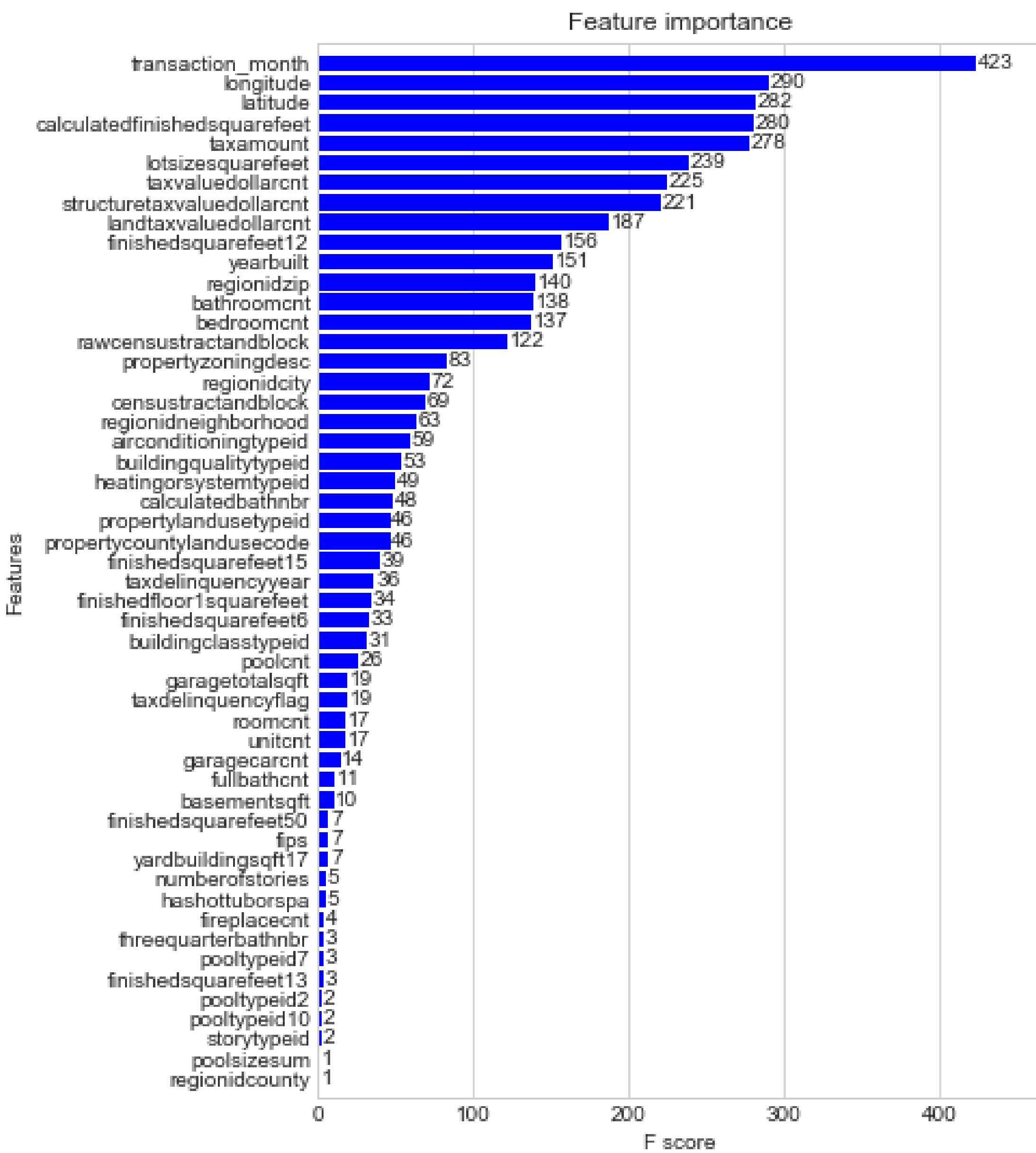
```
                     features  importance
50  structuretaxvaluedollarcnt    0.060509
54                   taxamount    0.059521
53       landtaxvaluedollarcnt    0.056897
51           taxvaluedollarcnt    0.056176
26           lotsizesquarefeet    0.054112
```

In [18]:
```
fig, ax = plt.subplots(figsize=(6, 12.5))
sns.barplot(x="importance", y="features", data=etr_importance_df, color='Blue', ax=ax)
ax.set(xlabel="Importance", ylabel="Feature Name", title="Feature Importances")
plt.show()
```


Feature Importances
structuretaxvaluedollarcnt
taxamount
landtaxvaluedollarcnt
taxvaluedollarcnt
lotsizesquarefeet
transaction_month
latitude
calculatedfinishedsquarefeet
yearbuilt
longitude
finishedsquarefeet12
regionidzip
rawcensustractandblock
censustractandblock
bedroomcnt
regionidneighborhood
propertyzoningdesc
regionidcity
buildingqualitytypeid
airconditioningtypeid
propertycountylandusecode
bathroomcnt
calculatedbathnbr
fullbathcnt
propertylandusetypeid
heatingorsystemtypeid
garagetotalsqft
roomcnt
garagecarcnt
numberofstories
finishedsquarefeet15
unitcnt
fireplacecnt
poolcnt
yardbuildingsqft17
threequarterbathnbr
pooltypeid7
finishedfloor1squarefeet
finishedsquarefeet50
taxdelinquencyyear
finishedsquarefeet6
hashottuborspa
pooltypeid10
fips
taxdelinquencyflag
decktypeid
regionidcounty
buildingclasstypeid
pooltypeid2
yardbuildingsqft26
poolsizesum
storytypeid
basementsqft
architecturalstyletypeid
typeconstructiontypeid
fireplaceflag
finishedsquarefeet13
assessmentyear
Feature Name
0.00
0.01
0.02
0.03
0.04
0.05
0.06
Importance

```
In [19]: xgb_params = {
             'eta': 0.05,
             'max_depth': 8,
             'subsample': 0.7,
             'colsample_bytree': 0.7,
             'objective': 'reg:linear',
             'silent': 1,
             'seed' : 0
         }
         dtrain = xgb.DMatrix(x_train, y_train, feature_names=x_train.columns.values)
         model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=50)
```

```
In [20]: # plot the important features #
         fig, ax = plt.subplots(figsize=(6,9))
         xgb.plot_importance(model, height=0.8, grid = False, color="blue", ax=ax)
         ax.xaxis.grid()
         plt.show()
```



```python
In [ ]: import matplotlib.pyplot as plt
        import numpy as np

        # Simple data to display in various forms
        x = np.linspace(0, 2 * np.pi, 400)
        y = np.sin(x ** 2)

        plt.close('all')

        # Just a figure and one subplot
        f, ax = plt.subplots()
        ax.plot(x, y)
        ax.set_title('Simple plot')

        # Two subplots, the axes array is 1-d
        f, axarr = plt.subplots(2, sharex=True)
        axarr[0].plot(x, y)
        axarr[0].set_title('Sharing X axis')
        axarr[1].scatter(x, y)

        # Two subplots, unpack the axes array immediately
        f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
        ax1.plot(x, y)
        ax1.set_title('Sharing Y axis')
        ax2.scatter(x, y)

        # Three subplots sharing both x/y axes
        f, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, sharey=True)
        ax1.plot(x, y)
        ax1.set_title('Sharing both axes')
        ax2.scatter(x, y)
        ax3.scatter(x, 2 * y ** 2 - 1, color='r')
        # Fine-tune figure; make subplots close to each other and hide x ticks for
        # all but bottom plot.
        f.subplots_adjust(hspace=0)
        plt.setp([a.get_xticklabels() for a in f.axes[:-1]], visible=False)

        # row and column sharing
        f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex='col', sharey='row')
        ax1.plot(x, y)
        ax1.set_title('Sharing x per column, y per row')
        ax2.scatter(x, y)
        ax3.scatter(x, 2 * y ** 2 - 1, color='r')
        ax4.plot(x, 2 * y ** 2 - 1, color='r')

        # Four axes, returned as a 2-d array
        f, axarr = plt.subplots(2, 2)
        axarr[0, 0].plot(x, y)
        axarr[0, 0].set_title('Axis [0,0]')
        axarr[0, 1].scatter(x, y)
        axarr[0, 1].set_title('Axis [0,1]')
        axarr[1, 0].plot(x, y ** 2)
        axarr[1, 0].set_title('Axis [1,0]')
        axarr[1, 1].scatter(x, y ** 2)
        axarr[1, 1].set_title('Axis [1,1]')
        # Fine-tune figure; hide x ticks for top plots and y ticks for right plots
        plt.setp([a.get_xticklabels() for a in axarr[0, :]], visible=False)
        plt.setp([a.get_yticklabels() for a in axarr[:, 1]], visible=False)

        # Four polar axes
        f, axarr = plt.subplots(2, 2, subplot_kw=dict(projection='polar'))
        axarr[0, 0].plot(x, y)
        axarr[0, 0].set_title('Axis [0,0]')
        axarr[0, 1].scatter(x, y)
        axarr[0, 1].set_title('Axis [0,1]')
```

```
axarr[1, 0].plot(x, y ** 2)
axarr[1, 0].set_title('Axis [1,0]')
axarr[1, 1].scatter(x, y ** 2)
axarr[1, 1].set_title('Axis [1,1]')
# Fine-tune figure; make subplots farther from each other.
f.subplots_adjust(hspace=0.3)

plt.show()
```

1.4 Conclusions

In Progress

1.5 Bibliography

Couzin-Frankel, J. 2010. "Cancer Research. As Questions Grow, Duke Halts Trials, Launches Investigation." Science 329 (5992): 614–15.

Hunter, J. D. 2007. "Matplotlib: A 2D Graphics Environment." Computing In Science & Engineering 9 (3): 90–95.

McKinney, W. 2010. "Data Structures for Statistical Computing in Python." In Proceedings of the 9th Python in Science Conference, edited by S. J. van der Walt and K. J. Millman. Austin, Texas.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." Journal of Machine Learning Research 12 (Oct): 2825–30.

Pérez, F., and B. E. Granger. 2007. "IPython: A System for Interactive Scientific Computing." Computing in Science & Engineering 9 (3): 21–29.

Pérez, F., B. E. Granger, and J. D. Hunter. 2011. "Python: An Ecosystem for Scientific Computing." Computing in Science & Engineering 13 (2): 13–21.

Van der Walt, S., S. C. Colbert, and G. Varoquaux. 2011. "The NumPy Array: A Structure for Efficient Numerical Computation." Computing in Science & Engineering 13 (2): 22–30.

Waskom, M, O Botvinnik, P Hobson, J Warmenhoven, JB Cole, Y Halchenko, J Vanderplas, et al. 2014. Seaborn: Statistical Data Visualization. Stanford, California.