Nawen Gong, Xinyue Tao, Zichen Huang, Zheng Shan

MGTA 495 Big Data Tech and Business Applications

Final Project Report

March 6, 2019

Final Project Report

1. Overview

1.1 Project Introduction

This project is a group assignment of the MGTA 495 Big Data course. Data used in this analysis contains about 3M property information in the California area. This research aims to build a machine learning model, for house owners and buyers to get a suggested price range/category based on the location and other attributes of their property.

1.2 Dataset and Variables

The dataset includes 3M rows and 58 feature variables. Each row corresponds to a property, and includes the variables as described in *data_dict.xlsx*.

The housing market is influenced by both external and internal factors. Studies have shown that there are four key factors affecting the housing market and that will almost certainly impact the property price: location, updates and upgrades, neighborhood, and appraisal value. First, economists encapsulate “location” in something called “hedonic pricing” – for most homes, this translates to some key factors like quality of local schools, proximity to local employment opportunities and proximity to social, shopping and recreational centers. Second, for updates and upgrades, while some buyers actively seek out fixer-upper properties, most home buyers prefer a house that is move-in ready – and they are willing to pay a reasonable premium for that comfort. According to the National Association of Realtors, for example, upgraded kitchen and bathroom are among the most important upgrades cited by home buyers. Third, comparable properties that sold in your area also impact your own home’s market value. Appraisers and real estate agents usually look at recent sales of properties with similar features to use as a benchmark against your home’s potential price. Last but not least, the appraisal is the real estate industry’s formal process for pricing a property. Appraisers follow a structured process for evaluating the property by looking at recent comparable sales to establish a

benchmark price and then adjusting the price up or down according to the upgrades and improvements you have or have not made relative to the comparable properties. Almost everything can be included in the appraisal evaluation, including building quality, architectural style, heating system and construction type.

Based on our research purpose of exploring the influential factors on housing price as well as the previous study results, we first manually picked several relative attributes as the explanatory variables. Further, we checked the columns and removed the feature which has too many blank rows. The following table shows the final variables we choose in this analysis.

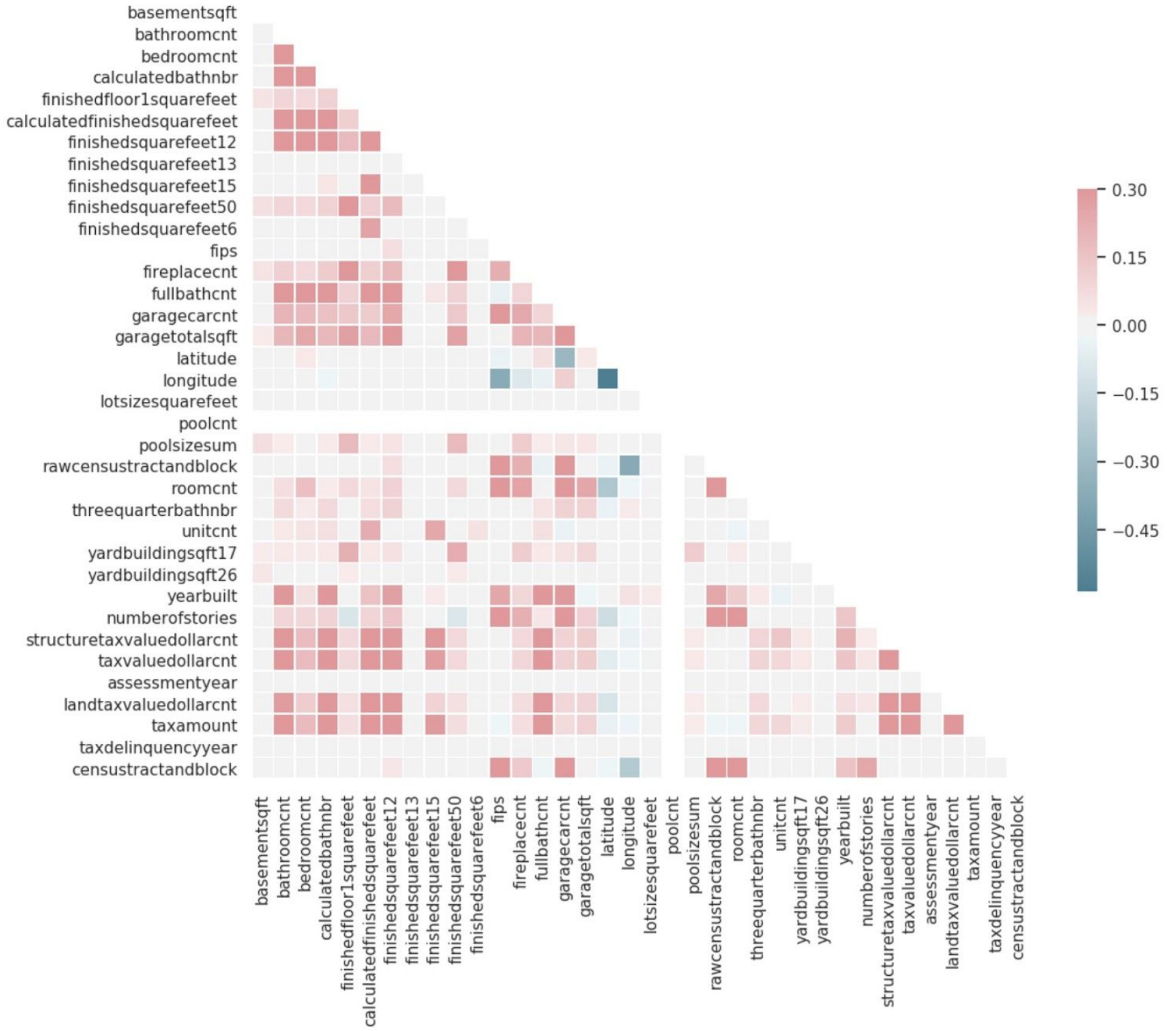
Table 1: Variable Description

Variable	Description	Type
taxvaluedollarcnt	The total tax assessed value of the parcel	Numeric (DV)
buildingqualitytypeid	Overall assessment of condition of the building from best (lowest) to worst (highest)	Character
propertylandusetypeid	Type of land use the property is zoned for	Character
regionidcounty	County in which the property is located	Character
bedroomcnt	Number of bedrooms in home	Numeric
calculatedbathnbr	Number of bathrooms in home including fractional bathroom	Numeric
calculatedfinishedsquarefeet	Calculated total finished living area of the home	Numeric

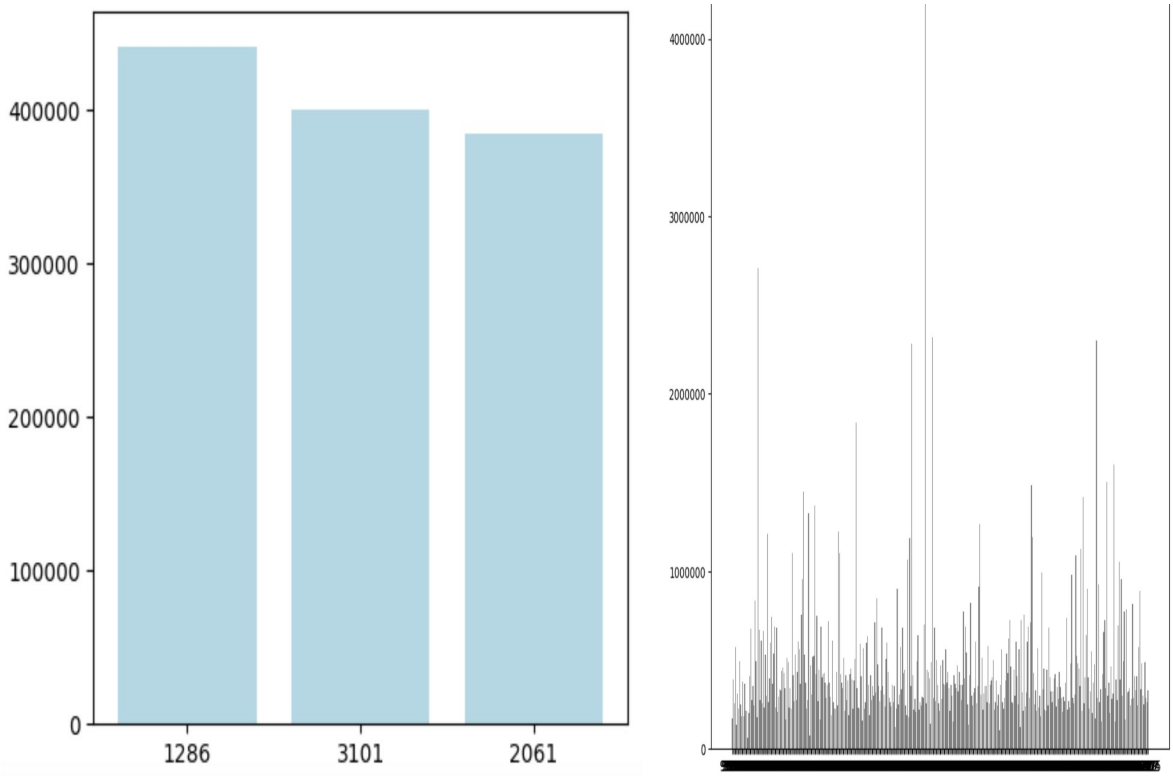
1.3 Exploratory Data Analysis

To make the variables that we chose based on research more reasonable, we did exploratory data analysis to detect the correlations among variables, as well as correlations between explanatory variables and response variable. The response variable should be the house price, in this case, is the total property tax assessed for that assessment year (*taxvaluedollarcnt*).

1.3.1 Correlation among variables



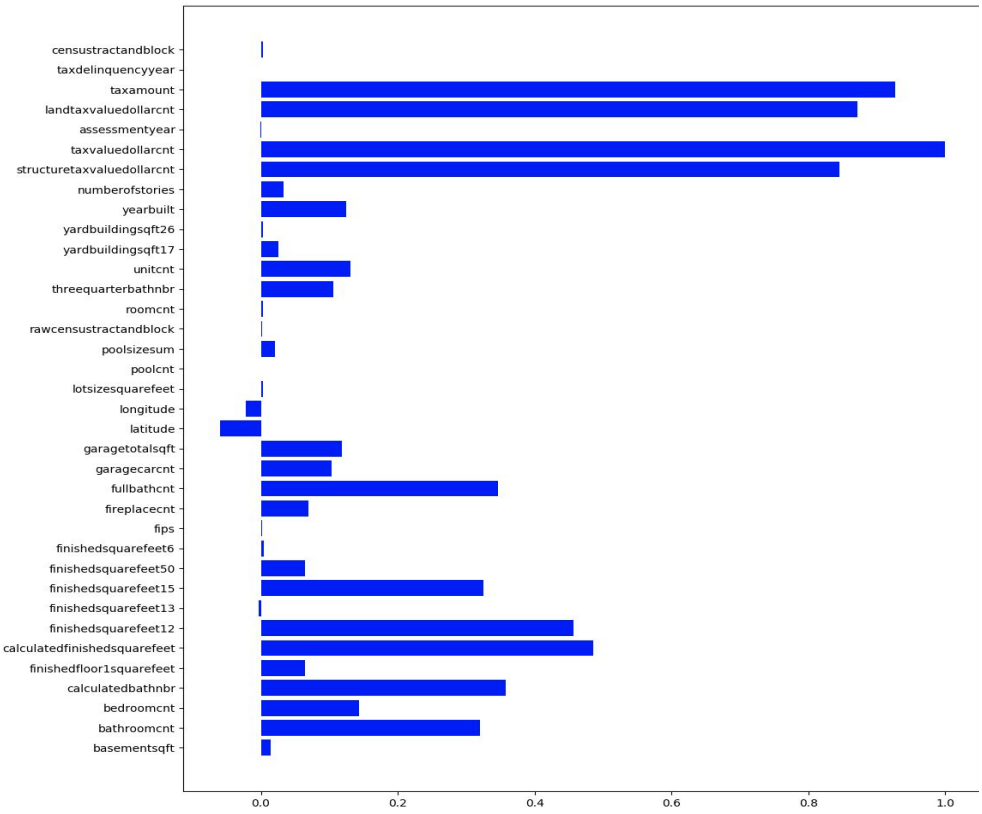
From the correlation plot, we have some interesting findings. First, the bathroom count of a house is correlated to the year built of house. The data shows that the newer the house, the more bathroom it has.



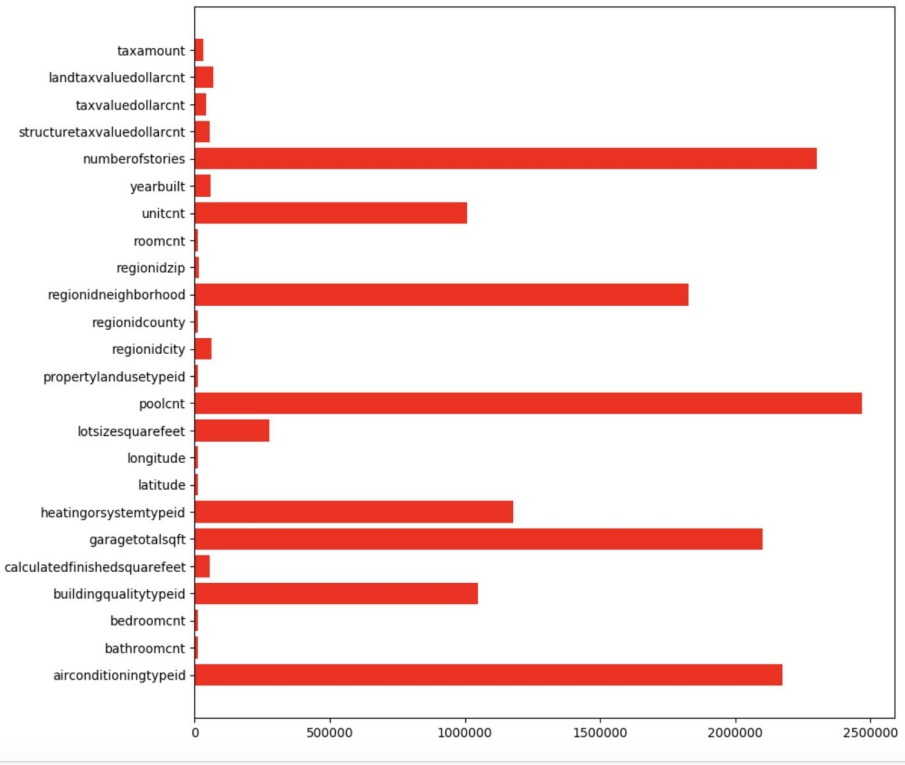
Another finding comes from the housing price plots by region-county and by zipcode respectively. We find that the fluctuation by region-county is slight, while it is strong in zipcode plot. Therefore, it is likely that house type may distribute equally across counties, which means we have similar amount of high-price house and lower-price house in each county.

1.3.2 Correlation with response variables

Plot to see how strong a variable is correlated to response variable. Then we picked only the numerical variables whose correlations with the response variable are bigger than 0.1. From this plot, we find that the bathroom number has a greater impact on taxvaluesollarcount than the bedroom number.



We also plotted to see how many observations can be left if we drop null values for each variable. We find that few values would be left for most of variables if we drop null values. More accurate selection will be done in the analytics part.



1.3.3 Summary Statistics

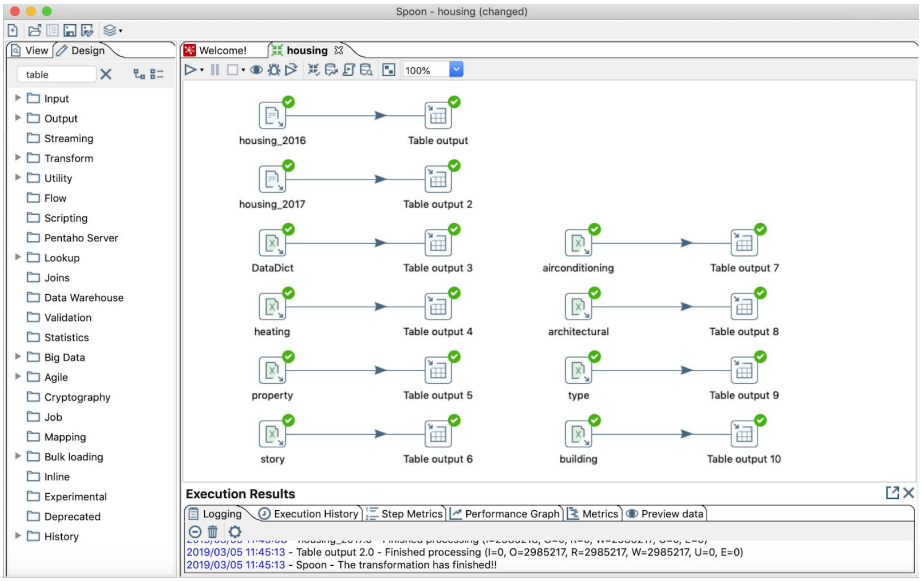
The summary statistic table shows the consistency of the dataset. The variation among variables is small. Moreover, one example is that the maximum number of bathroomcnt, bedroomcnt and fullbathcnt are all 20, which means this is a certain type of house, not abnormal data. So our dataset does not have obvious outliers.

	0	1	2	3	4
summary	count	mean	stddev	min	max
basementsqft	2971237	645.0003803129807	12.568447262315017	20	8516
bathroomcnt	2971237	2.2099579232488016	1.0768975679465302	0.0	20.0
bedroomcnt	2971237	3.090197113188884	1.2744726028341458	0.0	20.0
calculatedbathnbr	2971237	2.2992151384959736	0.9807692923823174	1.0	20.0
finishedfloor1squarefeet	2971237	1380.0462723101523	165.17575454847304	3	31303
calculatedfinishedsquarefeet	2971237	1827.098126206643	1806.6718730275406	1.0	952576.0
finishedsquarefeet12	2971237	1759.8576135124865	926.7876940332727	1	290345
finishedsquarefeet13	2971237	1178.0007666840445	18.11329867543705	120	2688
finishedsquarefeet15	2971237	2739.0041036780303	1380.3255435118613	112	820242
finishedsquarefeet50	2971237	1388.067254816765	173.43901387651167	3	31303
finishedsquarefeet6	2971237	2419.001721505218	662.1205051469076	117	952576
fips	2971237	6048.019906523781	20.218698916046854	6037	6111
fireplacecnt	2971237	1.0177363838697486	0.15824942732139913	1	9
fullbathcnt	2971237	2.234583104612658	0.9725778798175048	1	20
garagecarcnt	2971237	1.244724335352582	0.5021814074670394	0	25
garagetotalsqft	2971237	383.265228926538	133.74814615525753	0	7749
latitude	2971237	3.400143093969010 6E7	243346.46478450962	33324388	34819650
longitude	2971237	-1.1820182464171757E8	345181.5338952058	-119475780	-117554316
lotsizesquarefeet	2971237	22811.177774371165	322094.952454662	100.0	3.28263808E8

2. ETL

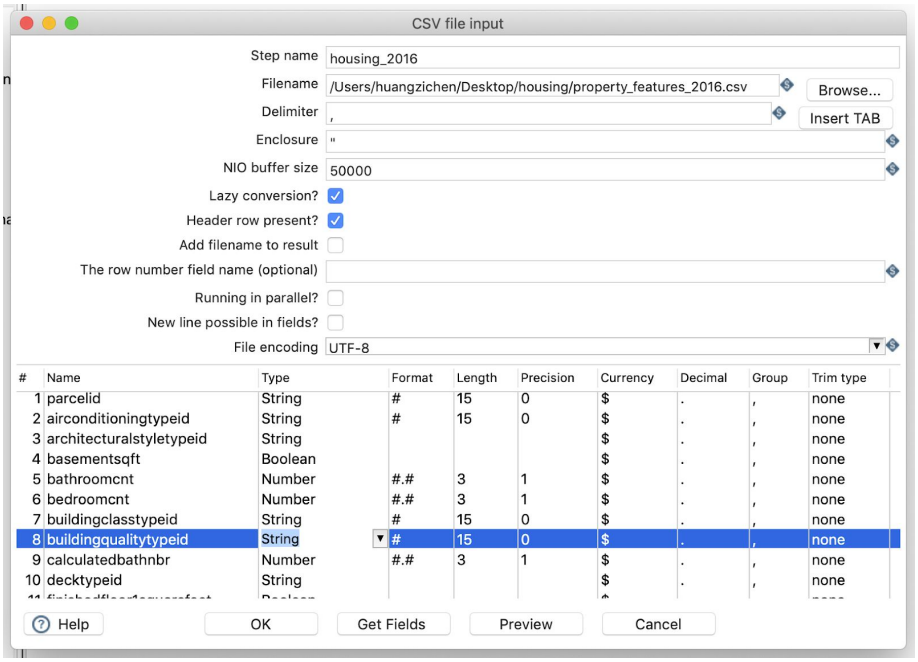
2.1 Import Data

In the ETL part, our main purpose was loading the housing data and data dictionary on our local Postgres database. For the *data_dict.xlsx*, this step is of vital to import and transform the dataset to readable and actionable tables for the following SQL and python analysis.



2.2 Transform variables type

To make the Databases/ Analytics task simpler, we also did some pre-process work in Pentaho Kettle. The first thing we did was transforming the character variables' type from integer to string. Then we preliminarily cleaned the data by removing spaces between characters in the column names and organizing output tables according to their directory names.



3. Database

3.1 Explore the data

After we group by region IDs, we found there are just 3 different counties which have ID number 3101, 1286 and 2061. We chose 3101 and 1286 as two regions to analyze in this part.

First, we compute the mean and variance of the total house price for every zipcode in each region.

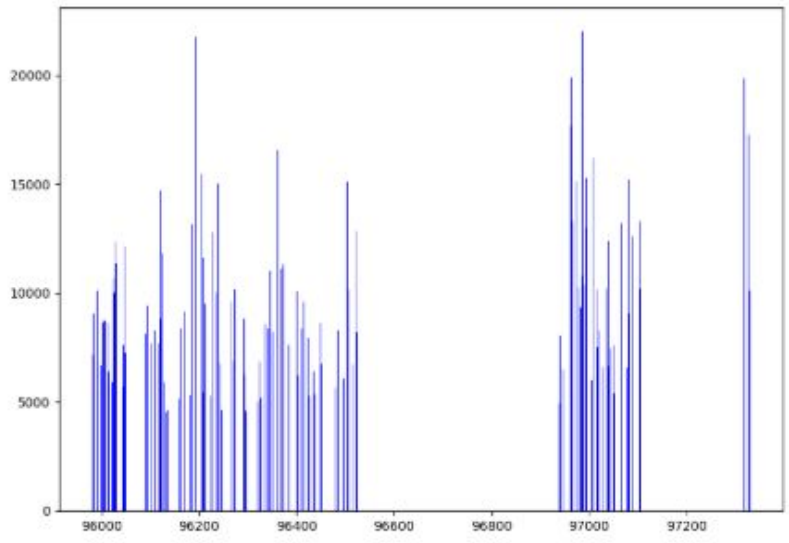
regionidcounty	zip	variance	mean
1286	97065	79889258493.77	426873.74
1286	97066	24263025138.88	248736.71
1286	97067	116449607594.02	430796.28
1286	97068	41834122605.76	343211.03
1286	97078	178004438643.82	546141.47
1286	97079	118040057125.76	598758.31
3101	95982	13095437954.86	185310.67
3101	95983	9652166689.56	173632.55
3101	95984	17100411048.66	206257.13
3101	95985	528646900422.82	684861.58
3101	95986	712354084840.51	708205.22
3101	95987	245418783202.2	363179.07

(shows only part of the table)

Also, we computed count number for every zipcode in each region and plot histogram to show the distribution.

regionidcounty	regionidzip	count
3101	96517	2950
3101	96522	4023
3101	96523	3302
3101	96524	2660
3101	96525	2201
3101	96531	3319
3101	96533	2110
3101	96859	1
1286	96939	1585
1286	96940	2002
1286	96941	2615
1286	96943	1450
1286	96946	1083
1286	96947	2055

(shows only part of the table)



3.2 Create two groups for the data and find indicator variables

According to the housing price, we created two groups, one has the higher price than the mean while the other has lower price.

For each group, we calculated the mean of every numeric variable and subtract them from corresponding columns between the two groups. Then we compared their difference in percentage. We chose the variable has the biggest difference as the indicator variable, which we found is "***calculatedfinishedsquarefeet"***.

Since this variable represents for total finished living area of the home, it does reflect the fluctuation of the housing price and is the most reasonable variable to indicate the housing price. So it should be the indicator variable to the housing price.

4. Analytics and Model Development

Due to the vast amount of null values in the datasets, we first impute both the categorical variables and numerical variables. We imputed the categorical variables using the mode and numerical using the mean. However, we discovered that for many such variables, the most common category is null. Therefore, we switched our method to use the second most common variable to impute these variables. We select only the numeric variables that are most strongly correlated to the response variable. There're 3

numeric features have been selected: “calculatedfinishedsquarefeet”, “calculatedbathnbr”, and “bedroomcnt”. For categorical features, we evaluate how much does the average price varies across the categorical variables, we use the following factor to compare such fluctuation:

stddev('avg(taxvaluedollarcnt)') / mean('avg(taxvaluedollarcnt)')

The selected categorical variables are: “Propertylandusetypeid”, “buildingqualitytypeid” and “regionidcounty”.

Next, we developed our model. First, we cut the price value using the number that most differentiated the amount of houses; then, we split the data into training and test set. Finally, we trained our classifier then tested it on the test set. The accuracy of our model is 82.3%. The detailed steps will be presented using our notebook.

In [5]:
```
print("The number of rows in house2016:  ", house2016.count())
```

```
The number of rows in house2016:   2985217
```

Next, we want to know how many rows remains if we drop Nulls for each of the variable

In [7]:
```
Count_row = {}
for col in house2016.columns:
    Count_row[col] = house2016.select(col).dropna(how='any').count()
```

In [8]:
```
#### Plot the number of not Null values in each variable
```

In [9]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 12))
plt.barh(*zip(*sorted(Count_row.items(), key=lambda x: x[1])), color =
'lightblue', align='center')

# plt.bar(range(len(Count_row)), Count_row.values(), align='center')
#plt.xticks(range(len(Count_row)), list(Count_row.keys()))
plt.title('Number of Valid Values')
plt.show()
```

```
<Figure size 1000x1200 with 1 Axes>
```

In [10]:
```
#### Let's create a list of all the 'id' type variable list
```

In [11]:
```
type_col =[ft for ft in house2016.columns if 'id' in ft]
type_col
```

Out[11]:
```
['parcelid',
 'airconditioningtypeid',
 'architecturalstyletypeid',
 'buildingclasstypeid',
 'buildingqualitytypeid',
 'decktypeid',
 'heatingorsystemtypeid',
 'pooltypeid10',
 'pooltypeid2',
 'pooltypeid7',
 'propertylandusetypeid',
 'regionidcity',
 'regionidcounty',
 'regionidneighborhood',
 'regionidzip',
 'storytypeid',
 'typeconstructiontypeid']
```

In [12]:
```
from pyspark.sql.types import StringType
```

Then change the id data type to string so that we can properky explore the dataset

In [13]:
```
# Change id type to string

House2016 = house2016.withColumn("airconditioningtypeid", house2016["airconditioningtypeid"].cast(StringType()))
House2016 = House2016.withColumn("buildingqualitytypeid", House2016["buildingqualitytypeid"].cast(StringType()))
House2016 = House2016.withColumn("heatingorsystemtypeid", House2016["heatingorsystemtypeid"].cast(StringType()))
House2016 = House2016.withColumn("propertylandusetypeid", House2016["propertylandusetypeid"].cast(StringType()))
House2016 = House2016.withColumn("regionidcity", House2016["regionidcity"].cast(StringType()))
House2016 = House2016.withColumn("regionidcounty", House2016["regionidcounty"].cast(StringType()))
House2016 = House2016.withColumn("regionidneighborhood", House2016["regionidneighborhood"].cast(StringType()))
House2016 = House2016.withColumn("regionidzip", House2016["regionidzip"].cast(StringType()))
House2016 = House2016.withColumn("parcelid", house2016["parcelid"].cast(StringType()))
House2016 = House2016.withColumn("architecturalstyletypeid", House2016["architecturalstyletypeid"].cast(StringType()))
House2016 = House2016.withColumn("buildingclasstypeid", House2016["buildingclasstypeid"].cast(StringType()))
House2016 = House2016.withColumn("decktypeid", House2016["decktypeid"].cast(StringType()))
House2016 = House2016.withColumn("pooltypeid10", House2016["pooltypeid10"].cast(StringType()))
House2016 = House2016.withColumn("pooltypeid2", House2016["pooltypeid2"].cast(StringType()))
House2016 = House2016.withColumn("pooltypeid7", House2016["pooltypeid7"].cast(StringType()))
House2016 = House2016.withColumn("typeconstructiontypeid", house2016["typeconstructiontypeid"].cast(StringType()))
House2016 = House2016.withColumn("storytypeid", House2016["storytypeid"].cast(StringType()))
```

In [14]:
```
House2016.printSchema()
```

```
root
 |-- parcelid: string (nullable = true)
 |-- airconditioningtypeid: string (nullable = true)
 |-- architecturalstyletypeid: string (nullable = true)
 |-- basementsqft: integer (nullable = true)
 |-- bathroomcnt: double (nullable = true)
 |-- bedroomcnt: double (nullable = true)
 |-- buildingclasstypeid: string (nullable = true)
 |-- buildingqualitytypeid: string (nullable = true)
 |-- calculatedbathnbr: double (nullable = true)
```

```
|-- decktypeid: string (nullable = true)
|-- finishedfloor1squarefeet: integer (nullable = true)
|-- calculatedfinishedsquarefeet: double (nullable = true)
|-- finishedsquarefeet12: integer (nullable = true)
|-- finishedsquarefeet13: integer (nullable = true)
|-- finishedsquarefeet15: integer (nullable = true)
|-- finishedsquarefeet50: integer (nullable = true)
|-- finishedsquarefeet6: integer (nullable = true)
|-- fips: integer (nullable = true)
|-- fireplacecnt: integer (nullable = true)
|-- fullbathcnt: integer (nullable = true)
|-- garagecarcnt: integer (nullable = true)
|-- garagetotalsqft: integer (nullable = true)
|-- hashottuborspa: boolean (nullable = true)
|-- heatingorsystemtypeid: string (nullable = true)
|-- latitude: integer (nullable = true)
|-- longitude: integer (nullable = true)
|-- lotsizesquarefeet: double (nullable = true)
|-- poolcnt: integer (nullable = true)
|-- poolsizesum: integer (nullable = true)
|-- pooltypeid10: string (nullable = true)
|-- pooltypeid2: string (nullable = true)
|-- pooltypeid7: string (nullable = true)
|-- propertycountylandusecode: string (nullable = true)
|-- propertylandusetypeid: string (nullable = true)
|-- propertyzoningdesc: string (nullable = true)
|-- rawcensustractandblock: double (nullable = true)
|-- regionidcity: string (nullable = true)
|-- regionidcounty: string (nullable = true)
|-- regionidneighborhood: string (nullable = true)
|-- regionidzip: string (nullable = true)
|-- roomcnt: double (nullable = true)
|-- storytypeid: string (nullable = true)
|-- threequarterbathnbr: integer (nullable = true)
|-- typeconstructiontypeid: string (nullable = true)
|-- unitcnt: integer (nullable = true)
|-- yardbuildingsqft17: integer (nullable = true)
|-- yardbuildingsqft26: integer (nullable = true)
|-- yearbuilt: double (nullable = true)
|-- numberofstories: integer (nullable = true)
|-- fireplaceflag: boolean (nullable = true)
|-- structuretaxvaluedollarcnt: double (nullable = true)
|-- taxvaluedollarcnt: double (nullable = true)
|-- assessmentyear: integer (nullable = true)
|-- landtaxvaluedollarcnt: double (nullable = true)
|-- taxamount: double (nullable = true)
|-- taxdelinquencyflag: string (nullable = true)
|-- taxdelinquencyyear: integer (nullable = true)
|-- censustractandblock: long (nullable = true)
```

Filter out all the null in zip codes

In [15]:
```
from pyspark.sql.functions import col, avg
# House2016.filter((f.col('taxvaluedollarcnt') < mean_txprice + 5*std_
txprice) & (f.col('taxvaluedollarcnt') > 0))
House2016 = house2016.where(col("regionidzip").isNotNull())

House2016.count()
```

Out[15]: 2971237

Filter out all the null in County codes

In [16]:
```
House2016 = House2016.where(col("regionidcounty").isNotNull())

House2016.count()
```

Out[16]: 2971237

Based on our backdround research and null value plot, we will first drop the following 'id' variables

In [17]:
```
drop = ["regionidcity", "regionidneighborhood", "parcelid", "pooltypei
d10", "pooltypeid2", "pooltypeid7"]
```

In [18]:
```
type_colDr = [c for c in type_col if c not in drop]
```

updated 'id' variables

In [19]:
```
type_colDr
```

Out[19]:
```
['airconditioningtypeid',
 'architecturalstyletypeid',
 'buildingclasstypeid',
 'buildingqualitytypeid',
 'decktypeid',
 'heatingorsystemtypeid',
 'propertylandusetypeid',
 'regionidcounty',
 'regionidzip',
 'storytypeid',
 'typeconstructiontypeid']
```

Imputaion for 'id' type variables: we'll use the most common (mode) category of each variable to fill na.

In [20]:
```
from pyspark.sql.functions import desc

type_dict={}
for i in type_colDr:
    type_dict[i] = House2016.groupBy(i).count().sort(desc("count")).co
llect()[1][0]
    # House2016.groupBy(i).count().sort(desc("count")).show(2)
```

In [21]:
```
type_dict
```

Out[21]:
```
{'airconditioningtypeid': 1,
 'architecturalstyletypeid': 7,
 'buildingclasstypeid': 4,
 'buildingqualitytypeid': None,
 'decktypeid': 66,
 'heatingorsystemtypeid': 2,
 'propertylandusetypeid': 266,
 'regionidcounty': 1286,
 'regionidzip': 96193,
 'storytypeid': 7,
 'typeconstructiontypeid': 6}
```

We then realized many 'id' variables have 'null' as the most frequent 'category', so we decided to use the 2nd most frequent to fill NA.

In [22]:
```
from pyspark.sql.functions import desc

for i in type_colDr:
    House2016.groupBy(i).count().sort(desc("count")).show(3)
    # House2016.groupBy(i).count().sort(desc("count")).show(2)
```

```
+--------------------+-------+
|airconditioningtypeid|  count|
+--------------------+-------+
|                null|2159981|
|                   1| 742110|
|                  13|  58448|
+--------------------+-------+
only showing top 3 rows

+-----------------------+-------+
|architecturalstyletypeid|  count|
+-----------------------+-------+
|                   null|2965176|
|                      7|   5251|
```

```
|                      8|    380|
+-----------------------+-------+
only showing top 3 rows

+-------------------+-------+
|buildingclasstypeid|  count|
+-------------------+-------+
|               null|2958610|
|                  4|   9263|
|                  3|   3161|
+-------------------+-------+
only showing top 3 rows

+---------------------+-------+
|buildingqualitytypeid|  count|
+---------------------+-------+
|                    7|1133234|
|                 null|1032754|
|                    4| 692159|
+---------------------+-------+
only showing top 3 rows

+----------+-------+
|decktypeid|  count|
+----------+-------+
|      null|2954180|
|        66|  17057|
+----------+-------+

+---------------------+-------+
|heatingorsystemtypeid|  count|
+---------------------+-------+
|                 null|1165462|
|                    2|1156510|
|                    7| 595198|
+---------------------+-------+
only showing top 3 rows

+---------------------+-------+
|propertylandusetypeid|  count|
+---------------------+-------+
|                  261|2146967|
|                  266| 480381|
|                  246| 114785|
+---------------------+-------+
only showing top 3 rows

+--------------+-------+
|regionidcounty|  count|
+--------------+-------+
|          3101|2008182|
|          1286| 740833|
|          2061| 222222|
```

```
+--------------+-------+

+-----------+-----+
|regionidzip|count|
+-----------+-----+
|      96987|22021|
|      96193|21759|
|      97118|20612|
+-----------+-----+
only showing top 3 rows

+-----------+-------+
|storytypeid|  count|
+-----------+-------+
|       null|2969619|
|          7|   1618|
+-----------+-------+

+----------------------+-------+
|typeconstructiontypeid|  count|
+----------------------+-------+
|                  null|2964490|
|                     6|   6670|
|                     4|     59|
+----------------------+-------+
only showing top 3 rows
```

Update the 2nd most count for some variables

In [23]:
```
type_dict.update({'regionidzip': '96987',
                  'regionidcounty': '3101',
                  'buildingqualitytypeid': '7',
                  'propertylandusetypeid': '261',
                  })
```

Impute

In [24]:
```
House2016 = House2016.na.fill(type_dict)
```

As we learned from our research that zip code could potentially influnce the tax assessment, and there could be many many zip codes.
Therefore, we will explore the zip code a bit at this point.

First, let's see how many distinct zip codes: 405

In [25]:
```
from pyspark.sql.functions import  mean , stddev, col

House2016.select('regionidzip').agg(f.countDistinct('regionidzip')).sh
ow()
```

```
+--------------------------+
|count(DISTINCT regionidzip)|
+--------------------------+
|                       405|
+--------------------------+
```

Then, we wondered what's the average number of houses included in each zip code, as well as the standard deveiation across the counts.

In [26]:
```
House2016.groupBy("regionidzip").count().sort('count').select(mean('co
unt'),stddev('count')).show()
```

```
+-----------------+------------------+
|       avg(count)|stddev_samp(count)|
+-----------------+------------------+
|7336.387654320988| 4511.408729334391|
+-----------------+------------------+
```

The std.dev is quite large, we should remove some outlier zips.

In [27]:
```
zip_count = House2016.groupBy("regionidzip").count().sort('count').sel
ect("regionidzip",'count')
```

We'll filter out thoes zip codes whose number of houses is less than one standard deveiation

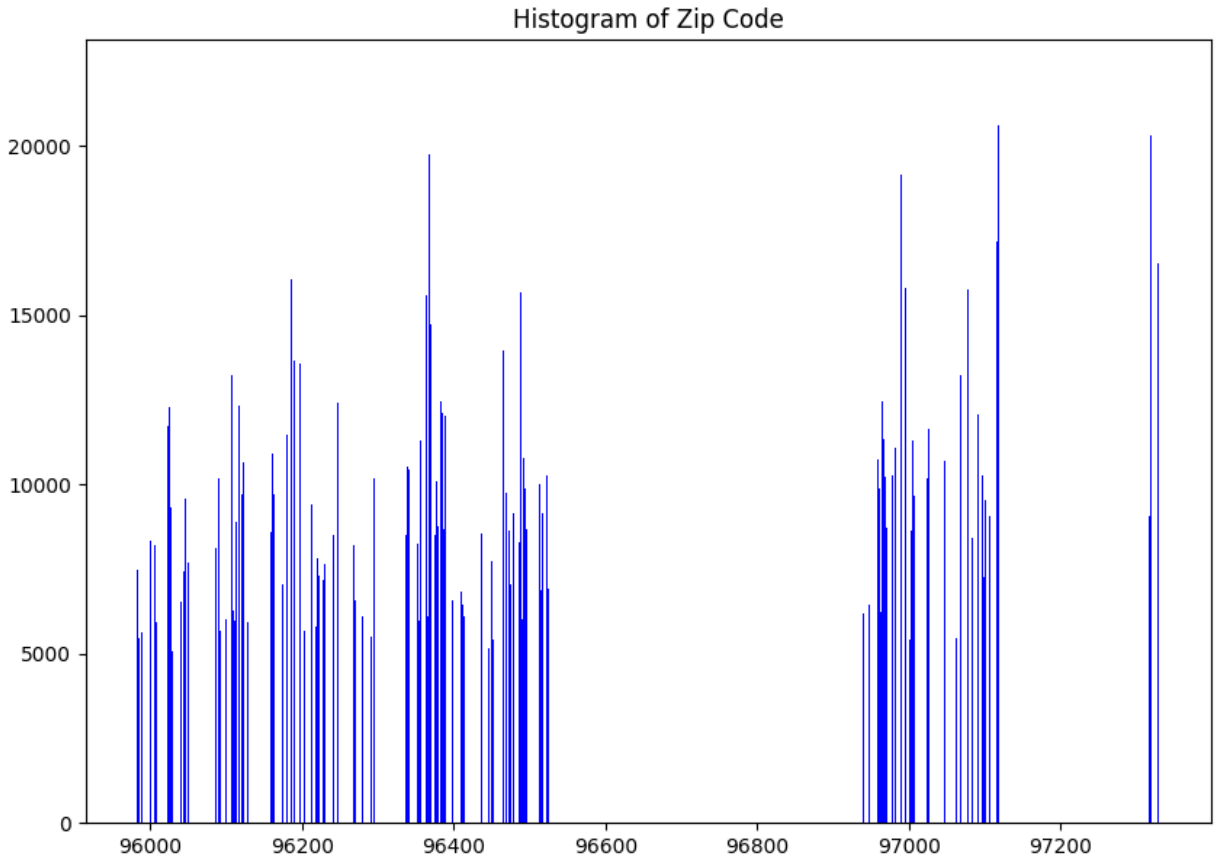
In [28]:
```
# Filter counted zip greater than mean-std and mean
zip0 = zip_count.filter(col('count') > 5000).select('regionidzip', 'co
unt').collect()
```

In [29]:
```
# zip and zip count for counted zip greater than mean-std
zipv = [row[1] for row in zip0]
zipz = [row[0] for row in zip0]
```

In [30]:
```
print("There're",  len(zipz), "unique zip codes left after the filtera
tion.")
```

```
There're 281 unique zip codes left after the filteration.
```

In [31]:
```
# Plot how strong a variable is correlated with response variable
import matplotlib.pyplot as plt
plt.rcdefaults()
#fig, ax = plt.subplots()
plt.figure(figsize=(10, 7))

#corr_df = pd.DataFrame(data = cor_value)
#plt.plot(cor_names, cor_value, label = cor_names)
plt.bar(zipz, zipv, align='center',
        color='blue', ecolor='black')
plt.title('Histogram of Zip Code')
plt.savefig('zip_plot.png')
plt.show()
```



Filter the data with zip code condition

In [32]:
```
zip_House = House2016.where(House2016.regionidzip.isin(zipz))
```

Now, get the numeric variables

In [33]:
```
# Get all numeric type VAR
numeric_features16 = [t[0] for t in zip_House.dtypes if t[1] != "boole
an" and t[1] != "string" and t[0] not in type_col]
numeric_features16 = [ft for ft in numeric_features16 if ft not in typ
e_col and ft not in drop]
numeric_features16
```

Out[33]:
```
['basementsqft',
 'bathroomcnt',
 'bedroomcnt',
 'calculatedbathnbr',
 'finishedfloor1squarefeet',
 'calculatedfinishedsquarefeet',
 'finishedsquarefeet12',
 'finishedsquarefeet13',
 'finishedsquarefeet15',
 'finishedsquarefeet50',
 'finishedsquarefeet6',
 'fips',
 'fireplacecnt',
 'fullbathcnt',
 'garagecarcnt',
 'garagetotalsqft',
 'latitude',
 'longitude',
 'lotsizesquarefeet',
 'poolcnt',
 'poolsizesum',
 'rawcensustractandblock',
 'roomcnt',
 'threequarterbathnbr',
 'unitcnt',
 'yardbuildingsqft17',
 'yardbuildingsqft26',
 'yearbuilt',
 'numberofstories',
 'structuretaxvaluedollarcnt',
 'taxvaluedollarcnt',
 'assessmentyear',
 'landtaxvaluedollarcnt',
 'taxamount',
 'taxdelinquencyyear',
 'censustractandblock']
```

Impute Numeric None values by mean

In [34]:
```
# Impute Numeric None values
meam_dict={}
for c in numeric_features16:
    m = zip_House.select(avg(zip_House[c])).collect()[0][0]
    #print(m)
    # meam_list.append(m)
    meam_dict[c] = m
    #House2016.select(numeric_features16[i]).na.fill(m).show(3)
```

In [35]:
```
meam_dict
```

Out[35]:
```
{'basementsqft': 647.895652173913,
 'bathroomcnt': 2.2074849468295192,
 'bedroomcnt': 3.1033735496419284,
 'calculatedbathnbr': 2.295578341765764,
 'finishedfloor1squarefeet': 1382.6442713455403,
 'calculatedfinishedsquarefeet': 1813.5716483779934,
 'finishedsquarefeet12': 1759.8635967383584,
 'finishedsquarefeet13': 1180.6752493933675,
 'finishedsquarefeet15': 2627.1679553990307,
 'finishedsquarefeet50': 1390.8713882081142,
 'finishedsquarefeet6': 2471.0858960176993,
 'fips': 6048.45293723126,
 'fireplacecnt': 1.1669700760387958,
 'fullbathcnt': 2.2396122968019294,
 'garagecarcnt': 1.8232444522069882,
 'garagetotalsqft': 383.15738678815046,
 'latitude': 34001813.918839194,
 'longitude': -118200430.76303254,
 'lotsizesquarefeet': 21964.877961956732,
 'poolcnt': 1.0,
 'poolsizesum': 521.1461153846154,
 'rawcensustractandblock': 60487614.93282979,
 'roomcnt': 1.5416538894761531,
 'threequarterbathnbr': 1.0088096801687882,
 'unitcnt': 1.1631564459640993,
 'yardbuildingsqft17': 320.6232412373297,
 'yardbuildingsqft26': 267.9068010075567,
 'yearbuilt': 1964.7416974365324,
 'numberofstories': 1.3981586213909682,
 'structuretaxvaluedollarcnt': 167257.9619801787,
 'taxvaluedollarcnt': 408774.5643702705,
 'assessmentyear': 2015.0000003726032,
 'landtaxvaluedollarcnt': 244391.25487808912,
 'taxamount': 5226.537456272089,
 'taxdelinquencyyear': 13.884521180485846,
 'censustractandblock': 60488476107277.54}
```

In [36]:
```
meam_dict.update({'yearbuilt':1965})
```

In [37]:
```
meam_dict
```

Out[37]:
```
{'basementsqft': 647.895652173913,
 'bathroomcnt': 2.2074849468295192,
 'bedroomcnt': 3.1033735496419284,
 'calculatedbathnbr': 2.295578341765764,
 'finishedfloor1squarefeet': 1382.6442713455403,
 'calculatedfinishedsquarefeet': 1813.5716483779934,
 'finishedsquarefeet12': 1759.8635967383584,
 'finishedsquarefeet13': 1180.6752493933675,
 'finishedsquarefeet15': 2627.1679553990307,
 'finishedsquarefeet50': 1390.8713882081142,
 'finishedsquarefeet6': 2471.0858960176993,
 'fips': 6048.45293723126,
 'fireplacecnt': 1.1669700760387958,
 'fullbathcnt': 2.2396122968019294,
 'garagecarcnt': 1.8232444522069882,
 'garagetotalsqft': 383.15738678815046,
 'latitude': 34001813.918839194,
 'longitude': -118200430.76303254,
 'lotsizesquarefeet': 21964.877961956732,
 'poolcnt': 1.0,
 'poolsizesum': 521.1461153846154,
 'rawcensustractandblock': 60487614.93282979,
 'roomcnt': 1.5416538894761531,
 'threequarterbathnbr': 1.0088096801687882,
 'unitcnt': 1.1631564459640993,
 'yardbuildingsqft17': 320.6232412373297,
 'yardbuildingsqft26': 267.9068010075567,
 'yearbuilt': 1965,
 'numberofstories': 1.3981586213909682,
 'structuretaxvaluedollarcnt': 167257.9619801787,
 'taxvaluedollarcnt': 408774.5643702705,
 'assessmentyear': 2015.0000003726032,
 'landtaxvaluedollarcnt': 244391.25487808912,
 'taxamount': 5226.537456272089,
 'taxdelinquencyyear': 13.884521180485846,
 'censustractandblock': 60488476107277.54}
```

In [38]:
```
House2016 = zip_House.na.fill(meam_dict)
```

In [39]:
```
drop_col = []
for col in house2016.columns:
    if col not in type_colDr and col not in numeric_features16:
        print(col)
        drop_col.append(col)
drop_col
```

```
parcelid
hashottuborspa
pooltypeid10
pooltypeid2
pooltypeid7
propertycountylandusecode
propertyzoningdesc
regionidcity
regionidneighborhood
fireplaceflag
taxdelinquencyflag
```

Out[39]:
```
['parcelid',
 'hashottuborspa',
 'pooltypeid10',
 'pooltypeid2',
 'pooltypeid7',
 'propertycountylandusecode',
 'propertyzoningdesc',
 'regionidcity',
 'regionidneighborhood',
 'fireplaceflag',
 'taxdelinquencyflag']
```

In [40]:
```
sel = type_colDr+numeric_features16
```

In [41]:
```
HouseDf = House2016.select(sel)
```

In [42]:
```
HouseDf.printSchema()
```

```
root
 |-- airconditioningtypeid: integer (nullable = false)
 |-- architecturalstyletypeid: integer (nullable = false)
 |-- buildingclasstypeid: integer (nullable = false)
 |-- buildingqualitytypeid: integer (nullable = true)
 |-- decktypeid: integer (nullable = false)
 |-- heatingorsystemtypeid: integer (nullable = false)
 |-- propertylandusetypeid: integer (nullable = true)
 |-- regionidcounty: integer (nullable = true)
 |-- regionidzip: integer (nullable = true)
 |-- storytypeid: integer (nullable = false)
 |-- typeconstructiontypeid: integer (nullable = false)
 |-- basementsqft: integer (nullable = true)
 |-- bathroomcnt: double (nullable = false)
 |-- bedroomcnt: double (nullable = false)
 |-- calculatedbathnbr: double (nullable = false)
 |-- finishedfloor1squarefeet: integer (nullable = true)
 |-- calculatedfinishedsquarefeet: double (nullable = false)
 |-- finishedsquarefeet12: integer (nullable = true)
 |-- finishedsquarefeet13: integer (nullable = true)
 |-- finishedsquarefeet15: integer (nullable = true)
 |-- finishedsquarefeet50: integer (nullable = true)
 |-- finishedsquarefeet6: integer (nullable = true)
 |-- fips: integer (nullable = true)
 |-- fireplacecnt: integer (nullable = true)
 |-- fullbathcnt: integer (nullable = true)
 |-- garagecarcnt: integer (nullable = true)
 |-- garagetotalsqft: integer (nullable = true)
 |-- latitude: integer (nullable = true)
 |-- longitude: integer (nullable = true)
 |-- lotsizesquarefeet: double (nullable = false)
 |-- poolcnt: integer (nullable = true)
 |-- poolsizesum: integer (nullable = true)
 |-- rawcensustractandblock: double (nullable = false)
 |-- roomcnt: double (nullable = false)
 |-- threequarterbathnbr: integer (nullable = true)
 |-- unitcnt: integer (nullable = true)
 |-- yardbuildingsqft17: integer (nullable = true)
 |-- yardbuildingsqft26: integer (nullable = true)
 |-- yearbuilt: double (nullable = false)
 |-- numberofstories: integer (nullable = true)
 |-- structuretaxvaluedollarcnt: double (nullable = false)
 |-- taxvaluedollarcnt: double (nullable = false)
 |-- assessmentyear: integer (nullable = true)
 |-- landtaxvaluedollarcnt: double (nullable = false)
 |-- taxamount: double (nullable = false)
 |-- taxdelinquencyyear: integer (nullable = true)
 |-- censustractandblock: long (nullable = true)
```

In [43]:
```
Count_row11 = {}
for col in HouseDf.columns:
    Count_row11[col] = HouseDf.select(col).dropna(how='any').count()
Count_row11
```

Out[43]:
```
{'airconditioningtypeid': 2683820,
 'architecturalstyletypeid': 2683820,
 'buildingclasstypeid': 2683820,
 'buildingqualitytypeid': 2683820,
 'decktypeid': 2683820,
 'heatingorsystemtypeid': 2683820,
 'propertylandusetypeid': 2683820,
 'regionidcounty': 2683820,
 'regionidzip': 2683820,
 'storytypeid': 2683820,
 'typeconstructiontypeid': 2683820,
 'basementsqft': 2683820,
 'bathroomcnt': 2683820,
 'bedroomcnt': 2683820,
 'calculatedbathnbr': 2683820,
 'finishedfloor1squarefeet': 2683820,
 'calculatedfinishedsquarefeet': 2683820,
 'finishedsquarefeet12': 2683820,
 'finishedsquarefeet13': 2683820,
 'finishedsquarefeet15': 2683820,
 'finishedsquarefeet50': 2683820,
 'finishedsquarefeet6': 2683820,
 'fips': 2683820,
 'fireplacecnt': 2683820,
 'fullbathcnt': 2683820,
 'garagecarcnt': 2683820,
 'garagetotalsqft': 2683820,
 'latitude': 2683820,
 'longitude': 2683820,
 'lotsizesquarefeet': 2683820,
 'poolcnt': 2683820,
 'poolsizesum': 2683820,
 'rawcensustractandblock': 2683820,
 'roomcnt': 2683820,
 'threequarterbathnbr': 2683820,
 'unitcnt': 2683820,
 'yardbuildingsqft17': 2683820,
 'yardbuildingsqft26': 2683820,
 'yearbuilt': 2683820,
 'numberofstories': 2683820,
 'structuretaxvaluedollarcnt': 2683820,
 'taxvaluedollarcnt': 2683820,
 'assessmentyear': 2683820,
 'landtaxvaluedollarcnt': 2683820,
 'taxamount': 2683820,
 'taxdelinquencyyear': 2683820,
 'censustractandblock': 2683820}
```

None values cleaned!!

In [44]:
```
HouseDf.na.drop().count() - HouseDf.count()
```

Out[44]: 0

Stats about all the numeric features

In [45]:
```
HouseDf.select(numeric_features16).describe().toPandas().transpose()
```

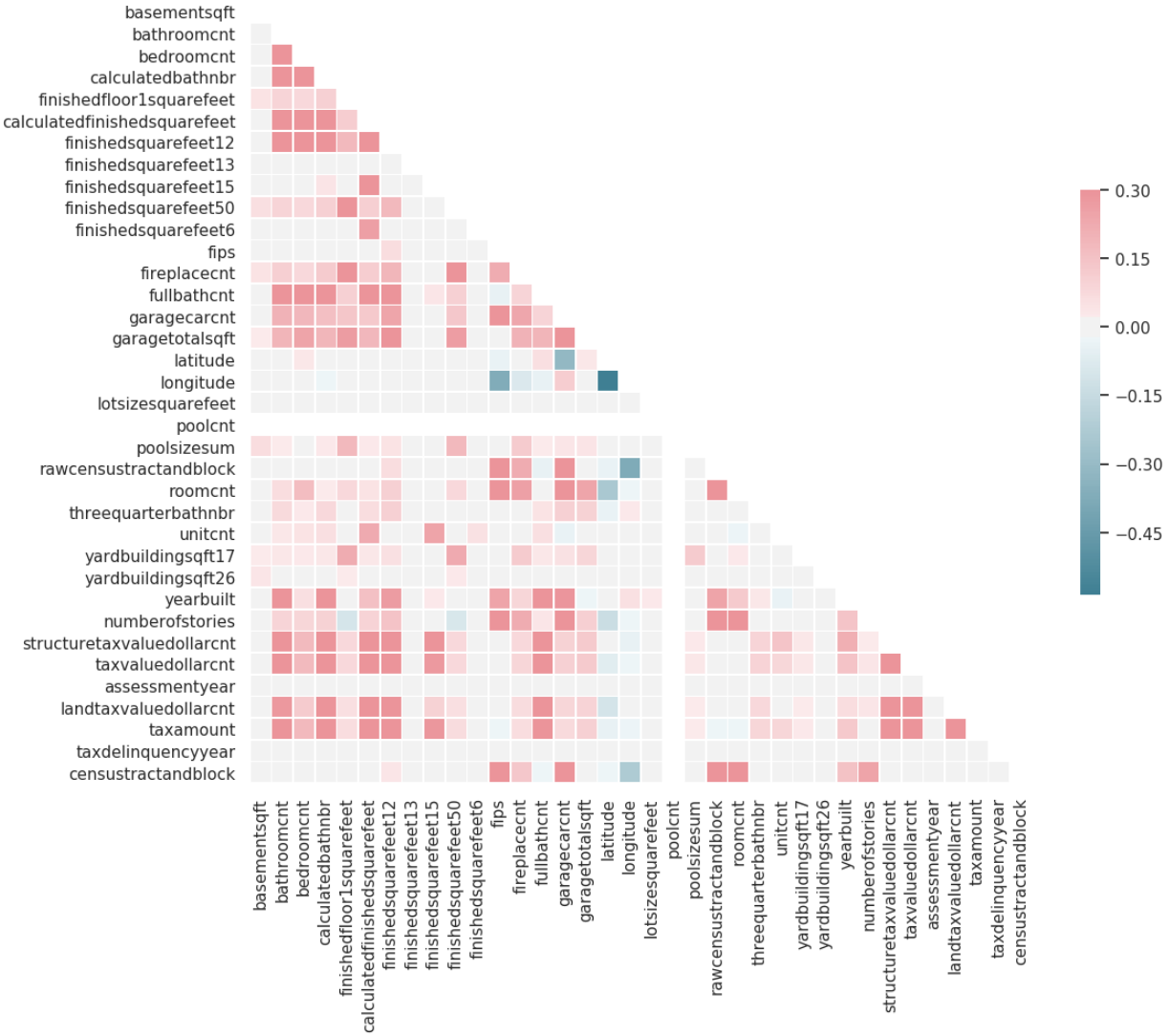
Out[45]:

	0	1	2	
summary	count	mean	stddev	
basementsqft	2683820	647.0004989157246	12.814841232135716	
bathroomcnt	2683820	2.2074849468295192	1.0595820701077143	
bedroomcnt	2683820	3.1033735496419284	1.2495052738662766	
calculatedbathnbr	2683820	2.2955783417660793	0.9631771799612371	
finishedfloor1squarefeet	2683820	1382.046134986698	168.0779321433231	
calculatedfinishedsquarefeet	2683820	1813.5716483778153	1453.3875858507695	
finishedsquarefeet12	2683820	1759.793229799316	904.985943029636	
finishedsquarefeet13	2683820	1180.0018663695778	18.635741856787227	
finishedsquarefeet15	2683820	2627.009631048282	1051.6662430264553	
finishedsquarefeet50	2683820	1390.062398372469	176.6699175082953	
finishedsquarefeet6	2683820	2471.000578652816	377.36319255476764	
fips	2683820	6048.45293723126	20.572908805040445	
fireplacecnt	2683820	1.0183518268736353	0.1609975065143112	
fullbathcnt	2683820	2.2303537495063006	0.9554811104360273	
garagecarcnt	2683820	1.252901461349867	0.5011858594659288	
garagetotalsqft	2683820	383.0483493676923	133.78019423731303	
latitude	2683820	3.4001813918839194E7	246535.7467099545	33
longitude	2683820	-1.1820043076303254E8	352885.2802397478	-119
lotsizesquarefeet	2683820	21964.877961967515	332408.6810954567	
poolcnt	2683820	1.0	0.0	
poolsizesum	2683820	521.0014155196698	16.153362429922844	
rawcensustractandblock	2683820	6.048761493282979E7	204205.7047654755	6.037101

roomcnt	2683820	1.5416538894761531	2.88692235473817
threequarterbathnbr	2683820	1.000939705345366	0.035758032959925
unitcnt	2683820	1.10689129673376	1.9007289621444663
yardbuildingsqft17	2683820	320.0177094589056	39.18890196385694
yardbuildingsqft26	2683820	267.0008048229762	10.450573435364143
yearbuilt	2683820	1964.7457817588363	22.625587937407406
numberofstories	2683820	1.0948278945681902	0.3118659350575201
structuretaxvaluedollarcnt	2683820	167257.9619801727	297284.8064783915
taxvaluedollarcnt	2683820	408774.56437025446	597044.5781729617
assessmentyear	2683820	2015.0000003726032	6.104123441142048E-4
landtaxvaluedollarcnt	2683820	244391.25487812248	390112.17994326307
taxamount	2683820	5226.537456271727	7493.774134178293
taxdelinquencyyear	2683820	13.016416153095214	0.3583754034029674
censustractandblock	2683820	6.048847610727156E13	3.340054228995646E11

Now we want to see how our features are correlated with each other

In [46]:
```
# Plot correlation among variables(numerical only)
from string import ascii_letters
import seaborn as sns
import numpy as np

sns.set(style="white")

# Compute the correlation matrix
corr = HouseDf[numeric_features16].toPandas().corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(12, 10))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.savefig('num_corPlot.png')
```

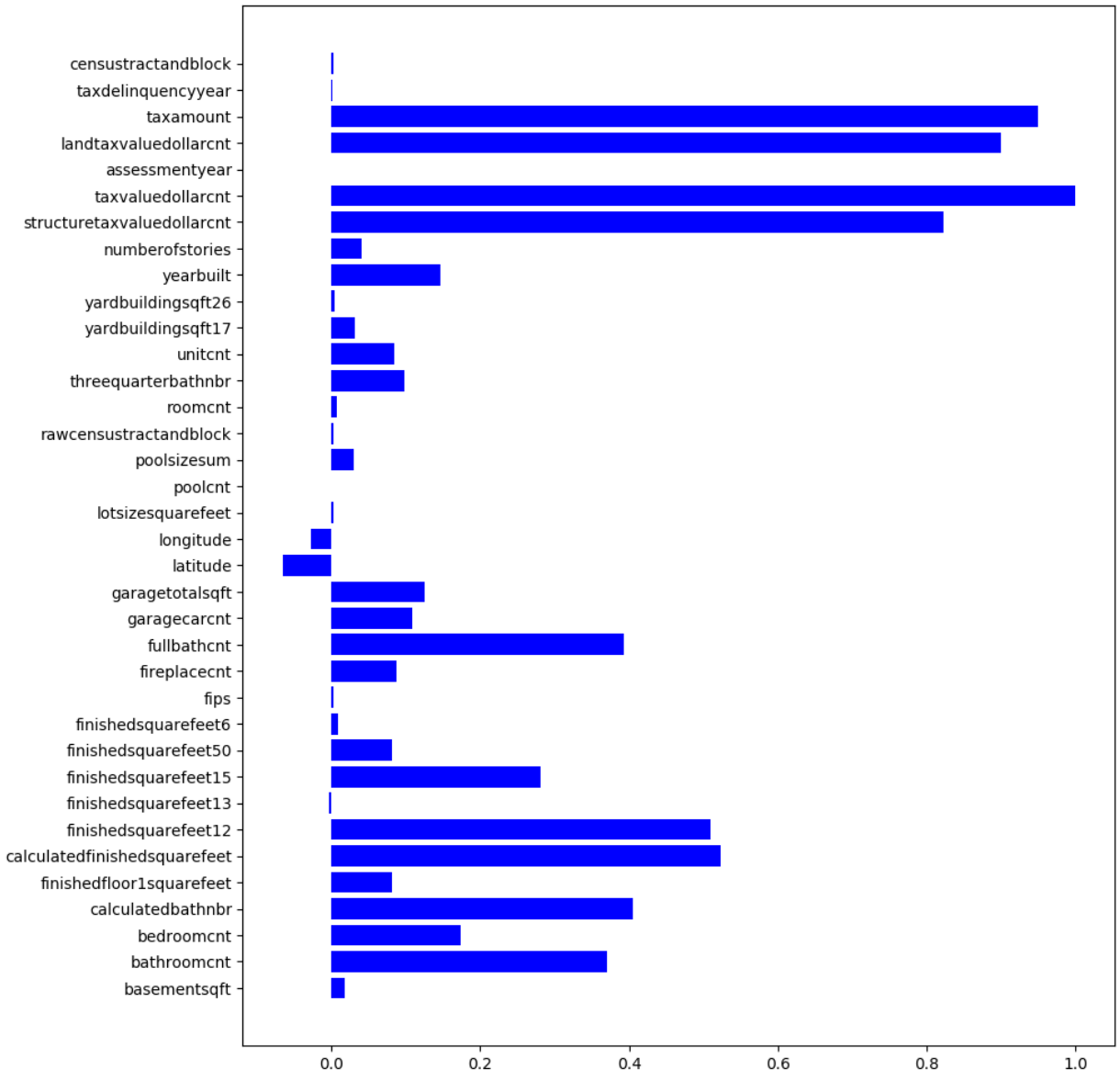


Also how features are correc\lated to our response variable - 'taxvaluedollarcnt'

In [47]:
```
pyspark.sql.DataFrame.dtypes
cor_value = []
cor_names = []
for col in numeric_features16:
    #if t != "boolean" and t != "string":
    print(col, '\t\t', HouseDf.stat.corr('taxvaluedollarcnt', col))
    cor_value.append(HouseDf.stat.corr('taxvaluedollarcnt', col))
    cor_names.append(col)
```

basementsqft 0.01805149766127698
bathroomcnt 0.3696070890490015
bedroomcnt 0.17415985844146975
calculatedbathnbr 0.405234089078775
finishedfloor1squarefeet 0.08081790858614783
calculatedfinishedsquarefeet 0.5237088708266852
finishedsquarefeet12 0.5096490121483217
finishedsquarefeet13 -0.0034907294069062415
finishedsquarefeet15 0.2809832165418459
finishedsquarefeet50 0.08182874588248186
finishedsquarefeet6 0.009191999699263376
fips 0.00329417662033179
fireplacecnt 0.08680705608196936
fullbathcnt 0.39328731202124795
garagecarcnt 0.10831886409214551
garagetotalsqft 0.12569333699417828
latitude -0.06582332567447154
longitude -0.027962727707293168
lotsizesquarefeet 0.003137086423457311
poolcnt nan
poolsizesum 0.030303740592498265
rawcensustractandblock 0.003020001319248574
roomcnt 0.007098337975657994
threequarterbathnbr 0.09774769309157294
unitcnt 0.08383112983743624
yardbuildingsqft17 0.03183789787427653
yardbuildingsqft26 0.004092497375239974
yearbuilt 0.14707704089710286
numberofstories 0.0402270684958037
structuretaxvaluedollarcnt 0.822481836723147
taxvaluedollarcnt 1.0
assessmentyear -0.00020295638990651982
landtaxvaluedollarcnt 0.8994730732323326
taxamount 0.950184732139935
taxdelinquencyyear 0.001455630677540741
censustractandblock 0.00290032708035248

In [48]:
```
# Plot how strong a variable is correlated with response variable
import matplotlib.pyplot as plt
plt.rcdefaults()
#fig, ax = plt.subplots()
plt.figure(figsize=(10, 12))

#corr_df = pd.DataFrame(data = cor_value)
#plt.plot(cor_names, cor_value, label = cor_names)
plt.barh(cor_names, cor_value, align='center',
        color='blue', ecolor='black')
plt.savefig('cor_plot.png')
plt.show()
```



In [49]:
```
from pyspark.sql.functions import udf, col, avg, mean, stddev

mean_txprice = HouseDf.select(avg(HouseDf["taxvaluedollarcnt"])).colle
ct()[0][0]
std_txprice = HouseDf.select(stddev(HouseDf["taxvaluedollarcnt"])).col
lect()[0][0]
```

In [50]:
```
print("Average price is: ", mean_txprice, "\nStandard deviation of hou
se price is: ", std_txprice)
```

```
Average price is:  408774.56437025446
Standard deviation of house price is:  597044.5781729617
```

The 10 lowest Average tax value by zip code

In [51]:
```
# the smallest 10 Average tax value by zip code
HouseDf.groupBy("regionidzip").avg('taxvaluedollarcnt').sort('avg(taxv
aluedollarcnt)').show(10)
```

```
+-----------+----------------------+
|regionidzip|avg(taxvaluedollarcnt)|
+-----------+----------------------+
|      97317|    133727.67283447614|
|      97328|    155953.39036243845|
|      96092|     162091.6904351132|
|      97318|    164811.92801706746|
|      96040|    173596.52163879096|
|      95983|     176777.4602219548|
|      97330|    182748.52527355772|
|      96091|    183221.73507672932|
|      95982|    188252.89804527425|
|      96090|    191119.61836800148|
+-----------+----------------------+
only showing top 10 rows
```

The 10 greatest Average tax value by zip code

In [52]:
```
# the greatest 10 Average tax value by zip code
HouseDf.groupBy("regionidzip").avg('taxvaluedollarcnt').sort(desc('avg
(taxvaluedollarcnt)')).show(10)
```

```
+----------+----------------------+
|regionidzip|avg(taxvaluedollarcnt)|
+----------+----------------------+
|      96086|    2708310.6582512315|
|      96116|    1840618.1586578002|
|      96120|    1488259.6317921209|
|      96030|    1449395.2823105108|
|      96969|    1213910.9391872424|
|      96121|    1104897.6433541386|
|      96117|    1102726.3427223242|
|      96940|    1091469.0793292485|
|      96978|     1053121.466249877|
|      96337|     993334.3118358863|
+----------+----------------------+
only showing top 10 rows
```

the top ten years-built in which most houses were built

In [53]:
```
# the top 10 years in which most houses were built
HouseDf.groupBy("yearbuilt").count().sort(desc('count')).show(10)
```

```
+---------+-----+
|yearbuilt|count|
+---------+-----+
|   1965.0|88724|
|   1955.0|82824|
|   1950.0|70706|
|   1954.0|66885|
|   1953.0|61811|
|   1964.0|61519|
|   1956.0|61455|
|   1951.0|52207|
|   1963.0|51406|
|   1952.0|49697|
+---------+-----+
only showing top 10 rows
```

the top ten years-built in which least number of houses were built

In [54]:
```
# the top 10 years in which least houses were built
HouseDf.groupBy("yearbuilt").count().sort('count').show(10)
```

```
+---------+-----+
|yearbuilt|count|
+---------+-----+
|   1828.0|    1|
|   1821.0|    1|
|   1853.0|    1|
|   1834.0|    1|
|   1833.0|    1|
|   1831.0|    1|
|   1810.0|    1|
|   1829.0|    1|
|   1878.0|    1|
|   1823.0|    1|
+---------+-----+
only showing top 10 rows
```

Now let's see the tax assessments pattern varies among counties

In [55]:
```
HouseDf = HouseDf.withColumn("regionidcounty", HouseDf["regionidcounty
"].cast(StringType()))
mean_county = HouseDf.groupBy("regionidcounty").avg('taxvaluedollarcnt
').toPandas().values
mean_by_county = [d[1] for d in mean_county]
```

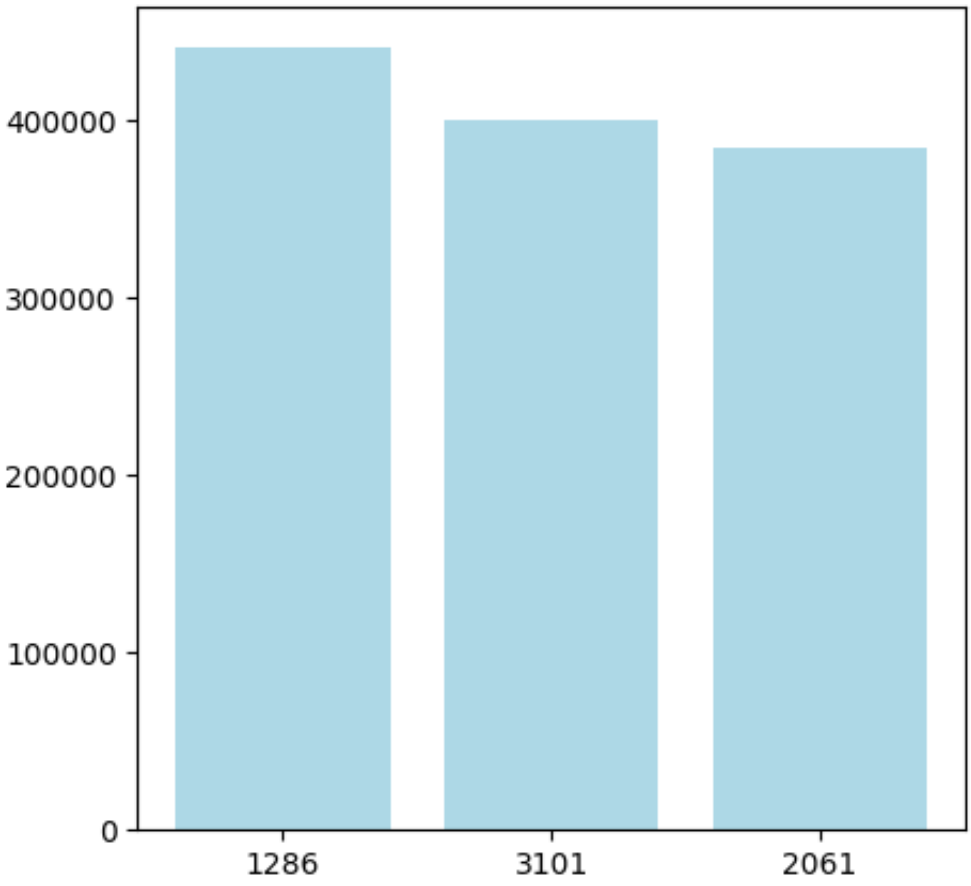
In [56]:
```
mean_zip = HouseDf.groupBy("regionidzip").avg('taxvaluedollarcnt').toP
andas().values
mean_by_zip = [d[1] for d in mean_zip]
```

In [57]:
```
countyid = [d[0] for d in mean_county]
zipid = [d[0] for d in mean_zip]
```

In [58]:
```
countyid, mean_by_county
```

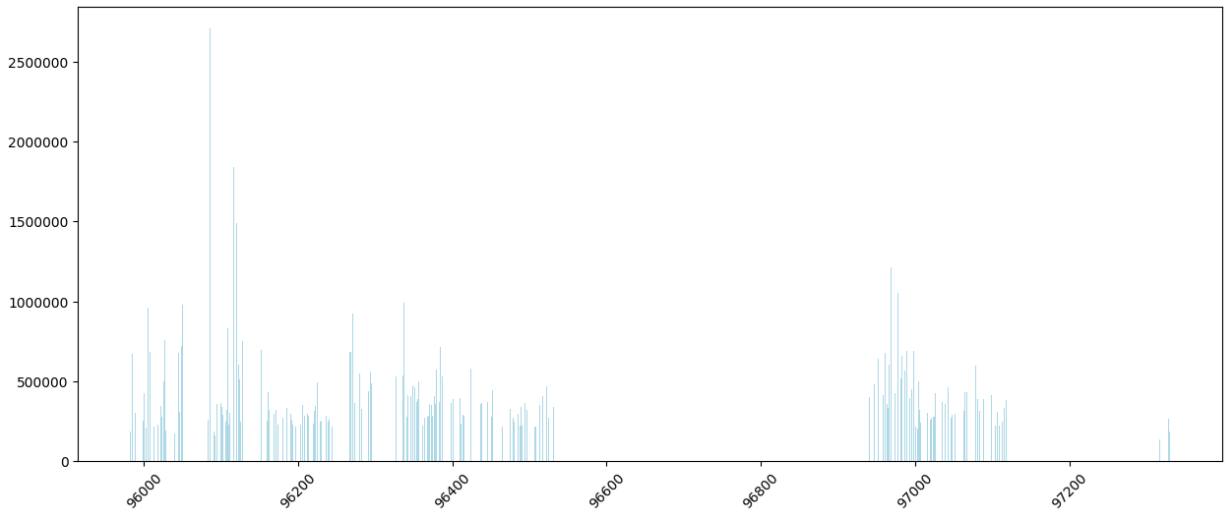
Out[58]:
```
(['1286', '3101', '2061'],
 [441123.89198351523, 399196.99533251673, 384381.5477141324])
```

In [59]:
```
import matplotlib.pyplot as plt
# plt.rcdefaults()
#fig, ax = plt.subplots()
plt.figure(figsize=(5, 5))

#corr_df = pd.DataFrame(data = cor_value)
#plt.plot(cor_names, cor_value, label = cor_names)
plt.bar(countyid, mean_by_county,
        color='lightblue')
#plt.savefig('m_byCounty_plot.png')
plt.show()
```



Tax assessments pattern varies among counties

In [60]:
```
# # Plot average price by Zipcode

import matplotlib.pyplot as plt
plt.rcdefaults()
#fig, ax = plt.subplots()
plt.figure(figsize=(15, 6))

#corr_df = pd.DataFrame(data = cor_value)
#plt.plot(cor_names, cor_value, label = cor_names)
plt.bar(zipid, mean_by_zip,align='center',
        color='lightblue', ecolor='grey')
#(zipid, mean_by_zip, align='center',color='grey', ecolor='black')
plt.xticks(rotation=45)
plt.savefig('meanZip_plot.png')
plt.show()
```



In [61]:
```
corr_num={}
for i in range(len(cor_names)):
    corr_num[cor_names[i]] = cor_value[i]
corr_num
```

Out[61]:

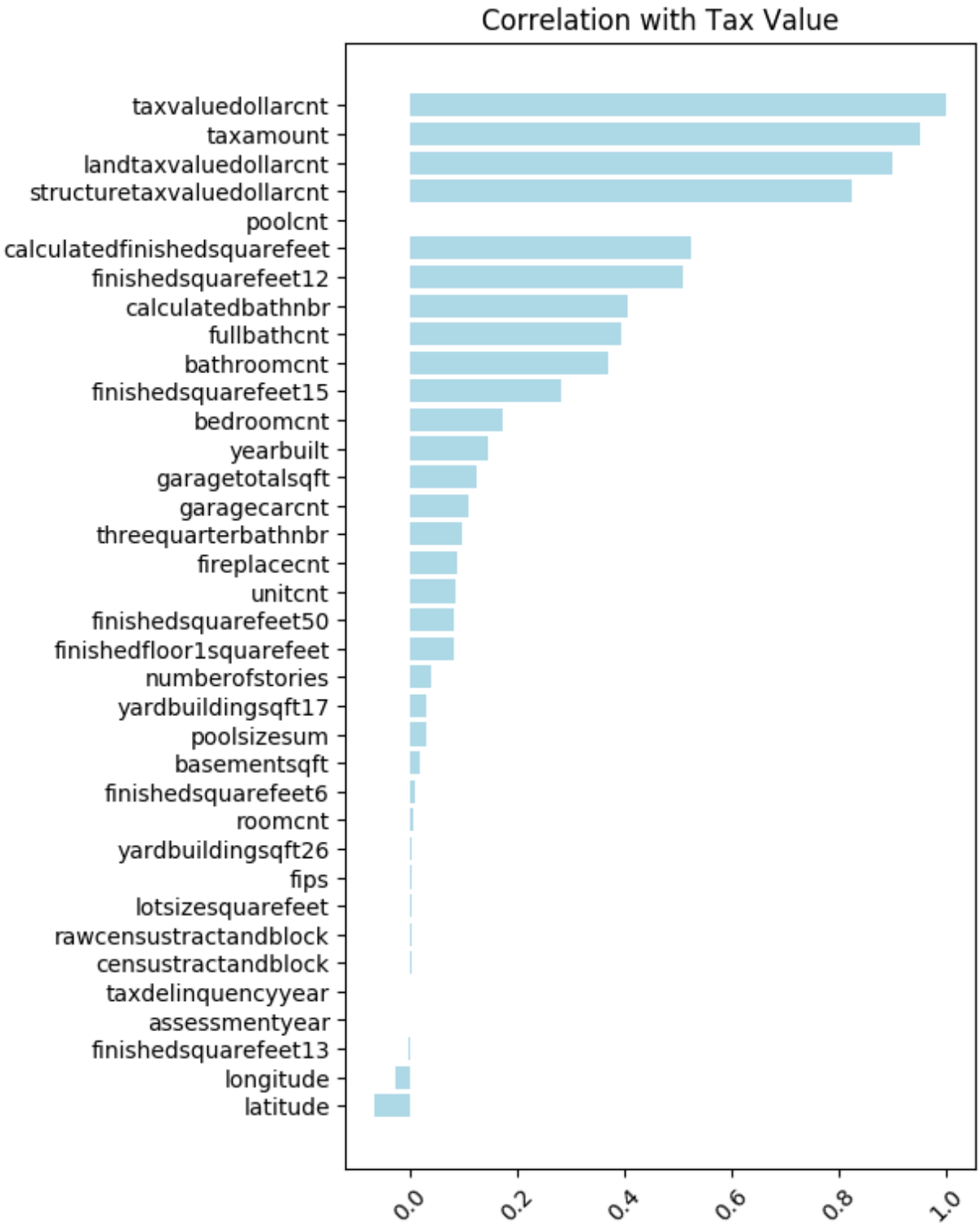
```
{'basementsqft': 0.018051497661276988,
 'bathroomcnt': 0.3696070890490015,
 'bedroomcnt': 0.17415985844146975,
 'calculatedbathnbr': 0.405234089078775,
 'finishedfloor1squarefeet': 0.08081790858614785,
 'calculatedfinishedsquarefeet': 0.5237088708266852,
 'finishedsquarefeet12': 0.5096490121483217,
 'finishedsquarefeet13': -0.003490729406906238,
 'finishedsquarefeet15': 0.2809832165418459,
 'finishedsquarefeet50': 0.08182874588248186,
 'finishedsquarefeet6': 0.009191999699263376,
 'fips': 0.003294176620331776,
 'fireplacecnt': 0.08680705608196934,
 'fullbathcnt': 0.39328731202124795,
 'garagecarcnt': 0.10831886409214547,
 'garagetotalsqft': 0.12569333699417828,
 'latitude': -0.06582332567447151,
 'longitude': -0.027962727707293158,
 'lotsizesquarefeet': 0.0031370864234573116,
 'poolcnt': nan,
 'poolsizesum': 0.030303740592498272,
 'rawcensustractandblock': 0.003020001319248574,
 'roomcnt': 0.007098337975657993,
 'threequarterbathnbr': 0.09774769309157294,
 'unitcnt': 0.08383112983743624,
 'yardbuildingsqft17': 0.03183789787427654,
 'yardbuildingsqft26': 0.004092497375239974,
 'yearbuilt': 0.14707704089710283,
 'numberofstories': 0.0402270684958037,
 'structuretaxvaluedollarcnt': 0.8224818367231469,
 'taxvaluedollarcnt': 1.0,
 'assessmentyear': -0.00020295638983164246,
 'landtaxvaluedollarcnt': 0.8994730732323326,
 'taxamount': 0.950184732139935,
 'taxdelinquencyyear': 0.0014556306775407406,
 'censustractandblock': 0.002900327080352474}
```


Visualize the correlation by corr values

In [62]:
```
sorted(corr_num.items(), key=lambda x: x[1])
```

Out[62]:
```
[('latitude', -0.06582332567447151),
 ('longitude', -0.027962727707293158),
 ('finishedsquarefeet13', -0.003490729406906238),
 ('assessmentyear', -0.00020295638983164246),
 ('taxdelinquencyyear', 0.0014556306775407406),
 ('censustractandblock', 0.002900327080352474),
 ('rawcensustractandblock', 0.003020001319248574),
 ('lotsizesquarefeet', 0.0031370864234573116),
 ('fips', 0.003294176620331776),
 ('yardbuildingsqft26', 0.004092497375239974),
 ('roomcnt', 0.007098337975657993),
 ('finishedsquarefeet6', 0.009191999699263376),
 ('basementsqft', 0.018051497661276988),
 ('poolsizesum', 0.030303740592498272),
 ('yardbuildingsqft17', 0.03183789787427654),
 ('numberofstories', 0.0402270684958037),
 ('finishedfloor1squarefeet', 0.08081790858614785),
 ('finishedsquarefeet50', 0.08182874588248186),
 ('unitcnt', 0.08383112983743624),
 ('fireplacecnt', 0.0868070560819693 4),
 ('threequarterbathnbr', 0.09774769309157294),
 ('garagecarcnt', 0.10831886409214547),
 ('garagetotalsqft', 0.12569333699417828),
 ('yearbuilt', 0.14707704089710283),
 ('bedroomcnt', 0.17415985844146975),
 ('finishedsquarefeet15', 0.2809832165418459),
 ('bathroomcnt', 0.3696070890490015),
 ('fullbathcnt', 0.39328731202124795),
 ('calculatedbathnbr', 0.405234089078775),
 ('finishedsquarefeet12', 0.5096490121483217),
 ('calculatedfinishedsquarefeet', 0.5237088708266852),
 ('poolcnt', nan),
 ('structuretaxvaluedollarcnt', 0.8224818367231469),
 ('landtaxvaluedollarcnt', 0.8994730732323326),
 ('taxamount', 0.950184732139935),
 ('taxvaluedollarcnt', 1.0)]
```

In [63]:
```
# Plot the correlation by corr values

plt.rcdefaults()
#fig, ax = plt.subplots()
plt.figure(figsize=(5, 9))
x, y = zip(*sorted(corr_num.items(), key=lambda x: x[1])) # unpack a l
ist of pairs into two tuples
plt.xticks(rotation=45)
plt.barh(x, y, align='center',
        color='lightblue', ecolor='grey')
plt.title('Correlation with Tax Value')
plt.savefig('correlation_num_plot.png')
plt.show()
```



get numeric features with highly correlated to response var

In [64]:
```
# sorted(corr_num, key=corr_num.get)
high_cor = [i[0] for i in sorted(corr_num.items(), key=lambda x: x[1])
][-10:-5]
high_cor
```

Out[64]:
```
['bathroomcnt',
 'fullbathcnt',
 'calculatedbathnbr',
 'finishedsquarefeet12',
 'calculatedfinishedsquarefeet']
```

Filter duplicated measurement and get the list of highly correlated num features

In [65]:
```
high_num = ['calculatedfinishedsquarefeet', 'calculatedbathnbr', 'bedr
oomcnt']#, 'unitcnt']
```

In [66]:
```
type_colDr
```

Out[66]:
```
['airconditioningtypeid',
 'architecturalstyletypeid',
 'buildingclasstypeid',
 'buildingqualitytypeid',
 'decktypeid',
 'heatingorsystemtypeid',
 'propertylandusetypeid',
 'regionidcounty',
 'regionidzip',
 'storytypeid',
 'typeconstructiontypeid']
```

To pick the categorical, we want to find thoes that have a strong variation within a categorical variable,

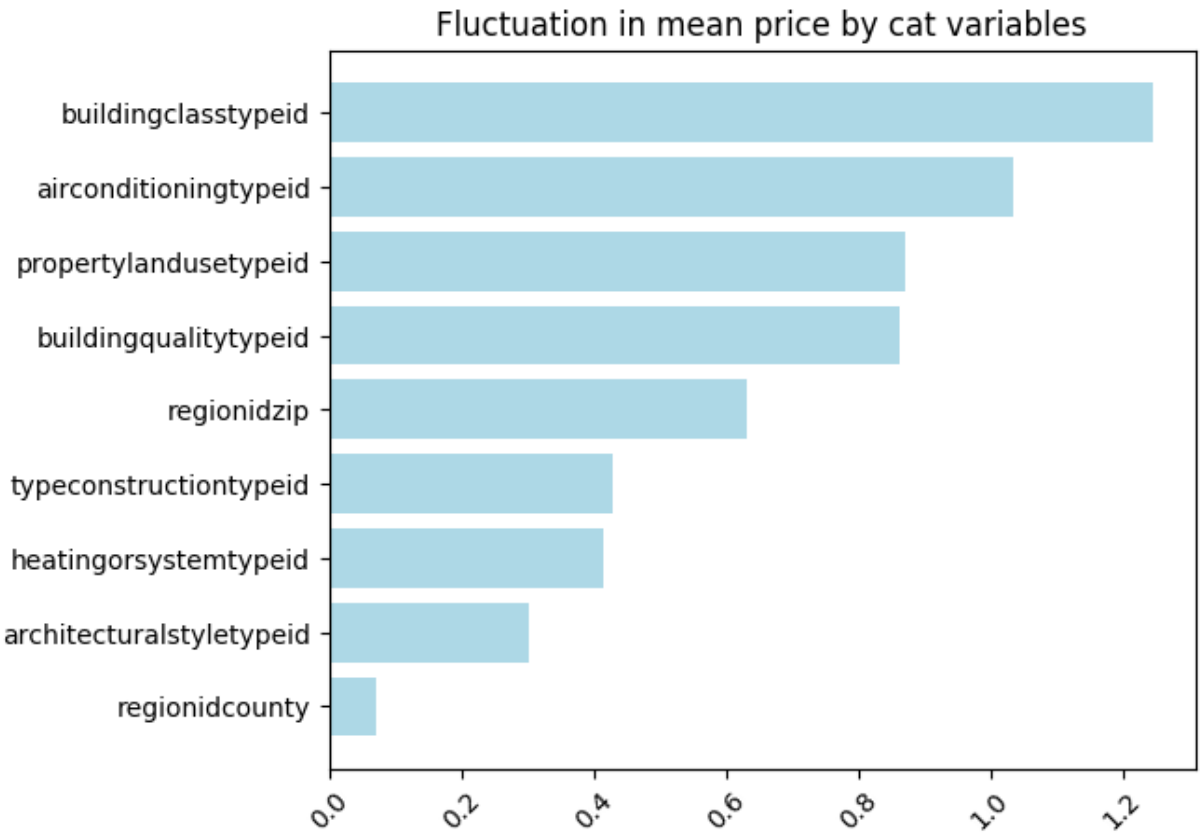
to measure this, we use the factor of standard deveiation over the mean

In [67]:
```
from pyspark.sql.functions import  mean , stddev, col

std_by_type={}

for id in type_colDr:
    std_by_type[id] = HouseDf.groupBy(id).avg('taxvaluedollarcnt').sel
ect(stddev('avg(taxvaluedollarcnt)')/mean('avg(taxvaluedollarcnt)')).c
ollect()[0][0]
```

In [68]:
```
sorted(std_by_type.items(), key=lambda x: x[1])
```

Out[68]:
```
[('regionidcounty', 0.07209310611902772),
 ('architecturalstyletypeid', 0.3023909763465844),
 ('heatingorsystemtypeid', 0.41442497566822073),
 ('typeconstructiontypeid', 0.43046811888551084),
 ('regionidzip', 0.6310315646012574),
 ('buildingqualitytypeid', 0.8638430455656158),
 ('propertylandusetypeid', 0.8718195887805529),
 ('airconditioningtypeid', 1.0352754092466445),
 ('buildingclasstypeid', 1.2470007828182506),
 ('decktypeid', nan),
 ('storytypeid', nan)]
```

In [69]:
```
# Plot the fluctuation in mean price by cat variables

plt.rcdefaults()
#fig, ax = plt.subplots()
plt.figure(figsize=(6, 5))
x, y = zip(*sorted(std_by_type.items(), key=lambda x: x[1])) # unpack
a list of pairs into two tuples
plt.xticks(rotation=45)
plt.barh(x, y, align='center',
        color='lightblue', ecolor='grey')
plt.title('Fluctuation in mean price by cat variables')
plt.savefig('fluc_cat_plot.png')
plt.show()
```



In [70]:
```
high_type = ['propertylandusetypeid', 'buildingqualitytypeid', 'region
idcounty'] #'airconditioningtypeid'
```

Now we combine the them into our final features to use

In [71]:
```
feat = high_num + high_type
feat
```

Out[71]:
```
['calculatedfinishedsquarefeet',
 'calculatedbathnbr',
 'bedroomcnt',
 'propertylandusetypeid',
 'buildingqualitytypeid',
 'regionidcounty']
```

In [72]:
```
HouseDf[feat+['taxvaluedollarcnt']].show(6)
```

```
+---------------------------+-----------------+----------+---------
------------+--------------------+-------------+-----------------+
|calculatedfinishedsquarefeet|calculatedbathnbr|bedroomcnt|propertyl
andusetypeid|buildingqualitytypeid|regionidcounty|taxvaluedollarcnt|
+---------------------------+-----------------+----------+---------
------------+--------------------+-------------+-----------------+
|          1813.5716483779934|2.295578341765764|       0.0|
269|                       7|           3101|               9.0|
|          1813.5716483779934|2.295578341765764|       0.0|
261|                       7|           3101|           27516.0|
|                     73026.0|2.295578341765764|       0.0|
47|                        7|           3101|         1413387.0|
|                        5068.0|2.295578341765764|       0.0|
47|                        7|           3101|         1156834.0|
|                        1776.0|2.295578341765764|       0.0|
31|                        7|           3101|          433491.0|
|                        2400.0|2.295578341765764|       0.0|
31|                        7|           3101|          283315.0|
+---------------------------+-----------------+----------+---------
------------+--------------------+-------------+-----------------+
only showing top 6 rows
```

In [73]:
```
HouseDf[feat+['taxvaluedollarcnt']].select('taxvaluedollarcnt').descri
be().toPandas().transpose()
```

Out[73]:

	0	1	2	3	4
summary	count	mean	stddev	min	max
taxvaluedollarcnt	2683820	408774.56437025446	597044.5781729617	1.0	1.49613482E8

In [74]:
```
from pyspark.sql.functions import udf, col, avg, mean, stddev
import pyspark.sql.functions as f
```

Remove outliers in 'taxvaluedollarcnt' using 3 std as upper limit

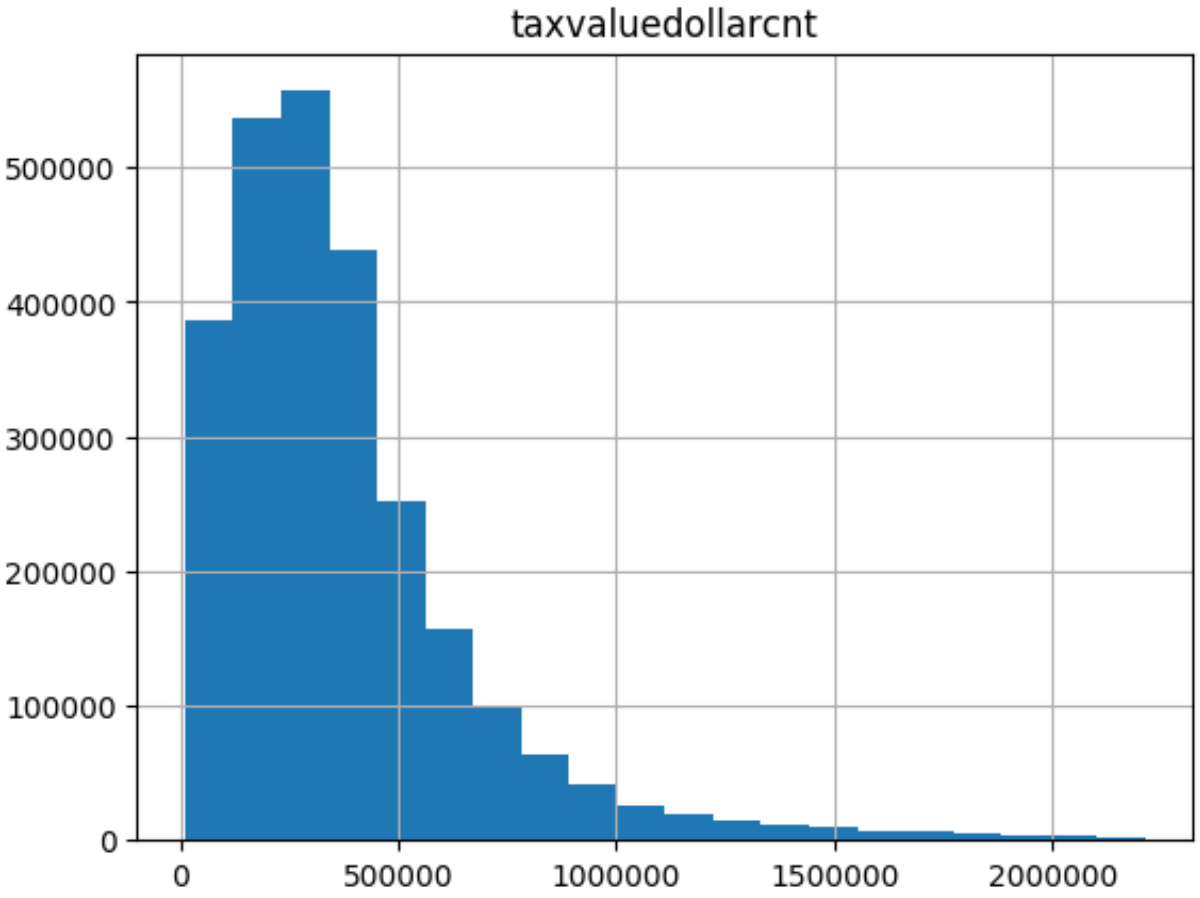
In [75]:
```
filter_HouseDf = HouseDf[feat+['taxvaluedollarcnt']].filter((f.col('taxvaluedollarcnt') < 411496 + 3*600599) & (f.col('taxvaluedollarcnt') > 10000))

print (" Number of rows remaining after removal:", '\t', filter_HouseDf.count(),
       "\n Number of outlier rows removed from DataFrame:", '\t', HouseDf.count() - filter_HouseDf.count(),
       "\n The upper limit above which is considered to be outliers:", 411496 + 3*600599)
```

```
 Number of rows remaining after removal:         2644905
 Number of outlier rows removed from DataFrame:          38915
 The upper limit above which is considered to be outliers: 2213293
```

Histogram of the cleaned 'taxvaluedollarcnt'

In [76]:
```
filter_HouseDf.select('taxvaluedollarcnt').toPandas().hist(bins=20)
```

Out[76]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f63c05df
e10>]],
      dtype=object)
```



Review of dropped variables

In [77]:
```
for col in house2016.columns:
    if col not in feat:
        print('removed column: ', col)
```

```
removed column:  parcelid
removed column:  airconditioningtypeid
removed column:  architecturalstyletypeid
removed column:  basementsqft
removed column:  bathroomcnt
removed column:  buildingclasstypeid
removed column:  decktypeid
removed column:  finishedfloor1squarefeet
removed column:  finishedsquarefeet12
removed column:  finishedsquarefeet13
removed column:  finishedsquarefeet15
removed column:  finishedsquarefeet50
removed column:  finishedsquarefeet6
removed column:  fips
removed column:  fireplacecnt
removed column:  fullbathcnt
removed column:  garagecarcnt
removed column:  garagetotalsqft
removed column:  hashottuborspa
removed column:  heatingorsystemtypeid
removed column:  latitude
removed column:  longitude
removed column:  lotsizesquarefeet
removed column:  poolcnt
removed column:  poolsizesum
removed column:  pooltypeid10
removed column:  pooltypeid2
removed column:  pooltypeid7
removed column:  propertycountylandusecode
removed column:  propertyzoningdesc
removed column:  rawcensustractandblock
removed column:  regionidcity
removed column:  regionidneighborhood
removed column:  regionidzip
removed column:  roomcnt
removed column:  storytypeid
removed column:  threequarterbathnbr
removed column:  typeconstructiontypeid
removed column:  unitcnt
removed column:  yardbuildingsqft17
removed column:  yardbuildingsqft26
removed column:  yearbuilt
removed column:  numberofstories
removed column:  fireplaceflag
removed column:  structuretaxvaluedollarcnt
removed column:  taxvaluedollarcnt
removed column:  assessmentyear
removed column:  landtaxvaluedollarcnt
removed column:  taxamount
removed column:  taxdelinquencyflag
removed column:  taxdelinquencyyear
removed column:  censustractandblock
```

In [78]:
```
feat
```

Out[78]:
```
['calculatedfinishedsquarefeet',
 'calculatedbathnbr',
 'bedroomcnt',
 'propertylandusetypeid',
 'buildingqualitytypeid',
 'regionidcounty']
```

In [79]:
```
featureCols = feat
```

In [80]:
```
featureCols
```

Out[80]:
```
['calculatedfinishedsquarefeet',
 'calculatedbathnbr',
 'bedroomcnt',
 'propertylandusetypeid',
 'buildingqualitytypeid',
 'regionidcounty']
```

In [81]:
```
labelCol = 'taxvaluedollarcnt'
House_mod = filter_HouseDf[featureCols + [labelCol]].dropna(how='any')
```

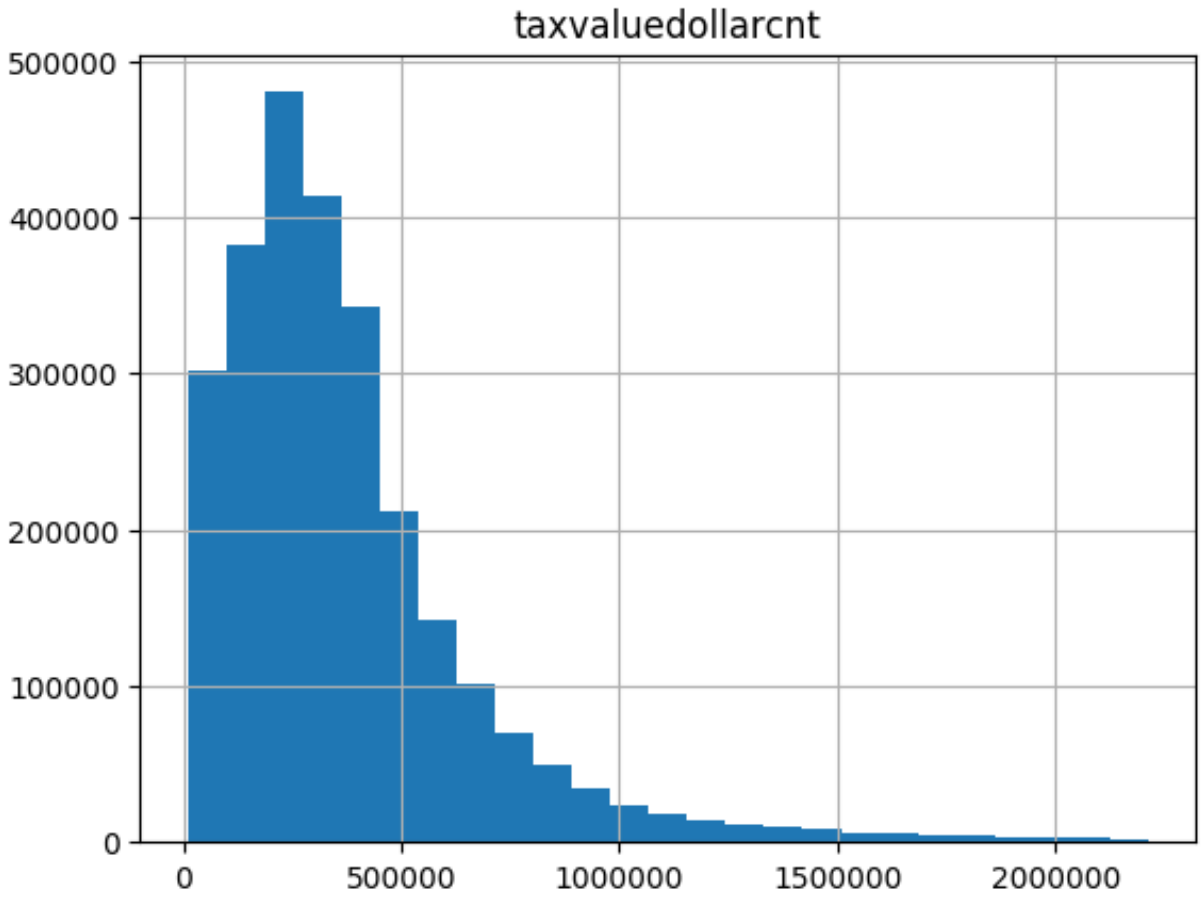
In [82]:
```
House_mod.printSchema()
```

```
root
 |-- calculatedfinishedsquarefeet: double (nullable = false)
 |-- calculatedbathnbr: double (nullable = false)
 |-- bedroomcnt: double (nullable = false)
 |-- propertylandusetypeid: integer (nullable = true)
 |-- buildingqualitytypeid: integer (nullable = true)
 |-- regionidcounty: string (nullable = true)
 |-- taxvaluedollarcnt: double (nullable = false)
```

To cut the tax assessment, we first re-plot the histogram of the response variable with 25 bins

In [83]:
```
House_mod.select('taxvaluedollarcnt').toPandas().hist(bins=25)
```

Out[83]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f62f2e9a
080>]],
      dtype=object)
```



We see the number of houses jumped quite a lot around 480000 to 500000, therefore, we decide to use 485000 as the cutting point

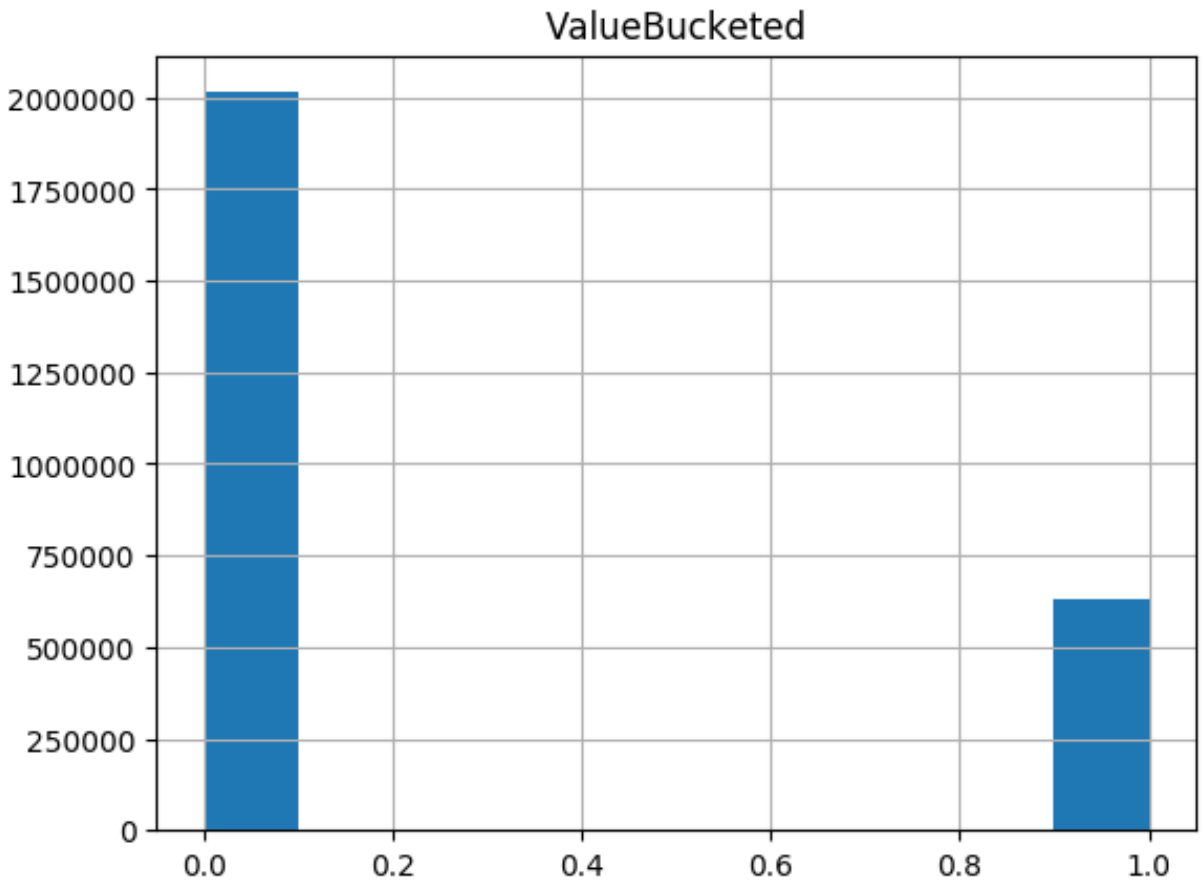
In [84]:
```
from pyspark.ml.feature import Bucketizer

# outlierUpperLim = mean_exp + 5*std_exp
# bucketSize = 3 # TODO
# upperLim = int(outlierUpperLim + bucketSize)
splits = [0, 485000, float("inf")]

bucketizer = Bucketizer(splits=splits, inputCol="taxvaluedollarcnt", o
utputCol="ValueBucketed")

# Transform original data into its bucket index.
bucketed_house = bucketizer.transform(House_mod)

# print("Bucketizer output with ", bucketSize, "buckets")
bucketed_house.select("ValueBucketed").show(5)
```

```
+-------------+
|ValueBucketed|
+-------------+
|          0.0|
|          1.0|
|          1.0|
|          0.0|
|          0.0|
+-------------+
only showing top 5 rows
```

Histogram of bucket

In [85]:
```
bucketed_house.select("ValueBucketed").toPandas().hist()
```

Out[85]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f632dfac
668>]],
      dtype=object)
```



In [86]:
```
bucketed_house.printSchema()
```

```
root
 |-- calculatedfinishedsquarefeet: double (nullable = false)
 |-- calculatedbathnbr: double (nullable = false)
 |-- bedroomcnt: double (nullable = false)
 |-- propertylandusetypeid: integer (nullable = true)
 |-- buildingqualitytypeid: integer (nullable = true)
 |-- regionidcounty: string (nullable = true)
 |-- taxvaluedollarcnt: double (nullable = false)
 |-- ValueBucketed: double (nullable = true)
```

In [87]:
```
high_type
```

Out[87]:
```
['propertylandusetypeid', 'buildingqualitytypeid', 'regionidcounty']
```

from pyspark.ml import Pipeline from pyspark.ml.classification import RandomForestClassifier from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer from pyspark.ml.evaluation import MulticlassClassificationEvaluator from pyspark.ml.feature import VectorAssembler from pyspark.ml.feature import VectorIndexer

assembler = VectorAssembler(inputCols=featureCols, outputCol="features")# handleInvalid="skip")
data_house = assembler.transform(bucketed_house2016)[['features', 'ValueBucketed']]

data_pr.show(5)

featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=5).fit(data_house)

In [88]:
```
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline

indexers = [StringIndexer(inputCol=c, outputCol=c+"_ind").fit(bucketed_house) for c in high_type]

pipeline = Pipeline(stages=indexers)
df_bucketed_house = pipeline.fit(bucketed_house).transform(bucketed_house)
```

In [89]:
```
ind_col = [col for col in df_bucketed_house.columns if "_ind" in col]
```

In [90]:
```
ind_col
```

Out[90]:
```
['propertylandusetypeid_ind',
 'buildingqualitytypeid_ind',
 'regionidcounty_ind']
```

In [91]:
```
featureCols[:3]
```

Out[91]:
```
['calculatedfinishedsquarefeet', 'calculatedbathnbr', 'bedroomcnt']
```

Final features in the model

In [92]:
```
ind_featureCols = featureCols[:3] + ind_col
ind_featureCols
```

Out[92]:
```
['calculatedfinishedsquarefeet',
 'calculatedbathnbr',
 'bedroomcnt',
 'propertylandusetypeid_ind',
 'buildingqualitytypeid_ind',
 'regionidcounty_ind']
```

In [93]:
```
new_house2016 = df_bucketed_house[ind_featureCols+['ValueBucketed']]
```

In [94]:
```
new_house2016.printSchema()
```

```
root
 |-- calculatedfinishedsquarefeet: double (nullable = false)
 |-- calculatedbathnbr: double (nullable = false)
 |-- bedroomcnt: double (nullable = false)
 |-- propertylandusetypeid_ind: double (nullable = false)
 |-- buildingqualitytypeid_ind: double (nullable = false)
 |-- regionidcounty_ind: double (nullable = false)
 |-- ValueBucketed: double (nullable = true)
```

In [95]:
```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import VectorIndexer

assembler = VectorAssembler(inputCols=ind_featureCols, outputCol="features")
```

In [96]:
```
model_house2016 = assembler.transform(new_house2016)[['features', 'ValueBucketed']]
```

train, test split

In [97]:
```
trainDf, testDf = model_house2016.randomSplit([0.75, 0.25], seed=12345
)
trainDf.show(5)
testDf.show(5)
```

```
+--------------------+-------------+
|            features|ValueBucketed|
+--------------------+-------------+
|(6,[0,1],[63.0,2....|          0.0|
|(6,[0,1],[78.0,1.0])|          0.0|
|(6,[0,1],[96.0,2....|          0.0|
|(6,[0,1],[99.0,2....|          0.0|
|(6,[0,1],[110.0,2...|          0.0|
+--------------------+-------------+
only showing top 5 rows

+--------------------+-------------+
|            features|ValueBucketed|
+--------------------+-------------+
|(6,[0,1],[100.0,2...|          0.0|
|(6,[0,1],[120.0,2...|          0.0|
|(6,[0,1],[132.0,2...|          0.0|
|(6,[0,1],[171.0,2...|          0.0|
|(6,[0,1],[173.0,2...|          0.0|
+--------------------+-------------+
only showing top 5 rows
```

Train using decision tree classifier

In [98]:
```
from pyspark.ml.classification import DecisionTreeClassifier

treeModel = DecisionTreeClassifier(labelCol='ValueBucketed',
                                   featuresCol='features',
                                   maxDepth=25)

# Fit the model
modelNoTuning = treeModel.fit(trainDf)
```

Test to see performance: the test and training accuracy is almost identical, we hence believe the model is robust to complete our classification task

In [99]:
```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

predDf_ts = modelNoTuning.transform(testDf)
predDf_tr = modelNoTuning.transform(trainDf)

evaluator = MulticlassClassificationEvaluator(
    labelCol="ValueBucketed", predictionCol="prediction", metricName="
accuracy")

accuracy_ts = evaluator.evaluate(predDf_ts)
accuracy_tr = evaluator.evaluate(predDf_tr)
print("Test Acc = %g " % (accuracy_ts))
print("Train Acc = %g " % (accuracy_tr))
```

```
Test Acc = 0.823976
Train Acc = 0.825503
```

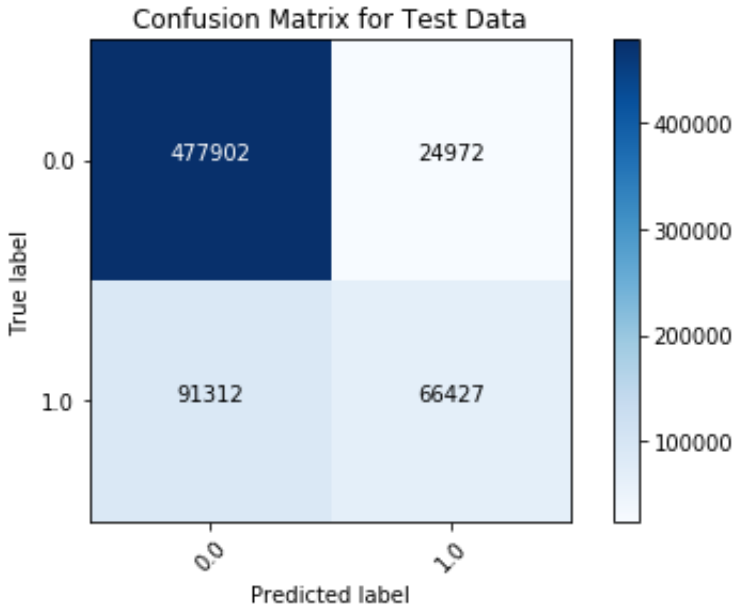
Confusion Matrix Evaluation

In [100]:
```
from sklearn.metrics import confusion_matrix
from plot_helper import plot_confusion_matrix

labelCol = 'ValueBucketed'
classes = predDf_ts.select(labelCol).groupBy(labelCol)\
                        .count().sort('count', ascending=False).toPand
as()
classLabels = classes[labelCol].values.tolist()


yTrue = predDf_ts.select('ValueBucketed').toPandas()
yPred = predDf_ts.select('prediction').toPandas()
cnfMatrix = confusion_matrix(yTrue, yPred)
cnfMatrix
```

Out[100]:
```
array([[477902,  24972],
       [ 91312,  66427]])
```

In [101]:
```
import matplotlib.pyplot as plt

%matplotlib inline

plt.figure()
plot_confusion_matrix(cnfMatrix, title="Confusion Matrix for Test Data
", classes=classLabels)
plt.show()
```

```
Confusion matrix, without normalization
[[477902  24972]
 [ 91312  66427]]
```



In [102]:
```
classLabels = [str(cl) for cl in classLabels]
classLabels
```

Out[102]:
```
['0.0', '1.0']
```

Report the mdoel performance

In [103]:
```
from sklearn.metrics import classification_report

# print (classification_report(yTrue, yPred, target_names=labelClasses
))
print (classification_report(yTrue, yPred, target_names=classLabels))
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89    502874
         1.0       0.73      0.42      0.53    157739

   micro avg       0.82      0.82      0.82    660613
   macro avg       0.78      0.69      0.71    660613
weighted avg       0.81      0.82      0.81    660613
```

In [ ]: