Denver-Homes-Rea...

Denver Home Real Estate Compare -Anish Pohane

FINISHED

Took 0 sec. Last updated by anonymous at September 12 2018, 9:00:16 AM.

I. Project Report

FINISHED

Took 1 sec. Last updated by anonymous at September 12 2018, 8:56:34 AM.

FINISHED

The initial goals of this project is to analyze the data of the real estate market from a particular city or county. In order to collect all the data, there is an open data catalog for all the parcels in Denver, https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-parcels (https://www.denvergov.org/opendata/dataset/city-and-county-of-denver-parcels). Data pulled from a csv file that contained all of the property types, owner information, prices, and other data of all zip codes in Denver.

Took 0 sec. Last updated by anonymous at September 12 2018, 8:58:18 AM.

READY

```
import org.apache.spark.sql.SparkSession
import spark.implicits._

val spark = SparkSession
  .builder()
  .appName("Intermediate Data")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
```

```
import org.apache.spark.sql.SparkSession
import spark.implicits._
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@21a53b6f
```

READY

```
val denverTab=spark.read
    .option("header", "true")
    .option("inferSchema", "true")
    .csv("C:/Users/parcelData/parcels.csv")
```

```
denverTab: org.apache.spark.sql.DataFrame = [PIN: int, SCHEDNUM: string ... 59 more fields]
```

FINISHED

Downloaded the csv file of the parcels, stored it locally in a folder, and began working on the data set. Used a Spark session to read the csv and store it into a DataFrame.

Took 0 sec. Last updated by anonymous at September 12 2018, 8:59:27 AM.

```
denverTab.printSchema()
```

READY

```
root
 |-- PIN: integer (nullable = true)
 |-- SCHEDNUM: string (nullable = true)
 |-- MAPNUM: string (nullable = true)
 |-- BLKNUM: string (nullable = true)
 |-- PARCELNUM: string (nullable = true)
 |-- APPENDAGE: string (nullable = true)
 |-- PARCEL_SOURCE: string (nullable = true)
 |-- SYSTEM_START_DATE: timestamp (nullable = true)
 |-- OWNER_NAME: string (nullable = true)
 |-- OWNER_ADDRESS_LINE1: string (nullable = true)
 |-- OWNER_ADDRESS_LINE2: string (nullable = true)
 |-- OWNER_ADDR_NBR_PREFIX: string (nullable = true)
 |-- OWNER_ADDR_NBR: string (nullable = true)
 |-- OWNER_ADDR_NBR_SUFFIX: string (nullable = true)
 |-- OWNER_STR_NAME_PRE_MOD: string (nullable = true)
 |-- OWNER_STR_NAME_PRE_DIR: string (nullable = true)
 |-- OWNER_STR_NAME_PRE_TYPE: string (nullable = true)
```

```
val denverTab_with_RPSq= denverTab.withColumn("RPSQ", $"TOTAL_VALUE"/$"IMP_AREA")
```

READY

```
denverTab_with_RPSq: org.apache.spark.sql.DataFrame = [PIN: int, SCHEDNUM: string ... 60 more fields]
```

```
denverTab_with_RPSq.createOrReplaceTempView("denverview_with_RPSq")
```

READY

Table 1

READY

```
%sql select * from denverview_with_RPSq LIMIT 20
```

READY

PIN	SCHEDNUM	MAPNUM	BLKNUM	PARCELNUM	APPENDAGE	PARCEL_SC
164177451	0532101079000	05321	01	079	000	null
164177469	0532101080000	05321	01	080	000	null
164177426	0529416061000	05294	16	061	000	null
164178147	0224300078000	02243	00	078	000	null
164177388	0529416057000	05294	16	057	000	null
164178139	0224300077000	02243	00	077	000	null

164176098	0219318060000	02193	18	060	000	null

READY

This first table just display the first 20 enteries from the csv file with the added RPSq column so that one can see what the data looks like and how the calculation turns out for the additional column.

READY

```
denverTab_with_RPSq.select(
    col("SITUS_ZIP"),
    col("SALE_PRICE"),
    col("TOTAL_VALUE"),
    col("RPSq")
    ).show(truncate=false)
```

```
+----------+----------+-----------+------------------+
|SITUS_ZIP |SALE_PRICE|TOTAL_VALUE|RPSq              |
+----------+----------+-----------+------------------+
|80219-6036|null      |253600     |169.51871657754012|
|80219-6036|null      |131200     |109.6989966555184 |
|null      |null      |199700     |null              |
|80205-3538|null      |1655500    |null              |
|null      |null      |256800     |214.71571906354515|
|80205     |null      |100        |null              |
|80212-2251|null      |197400     |84.93975903614458 |
|null      |240000    |25600      |null              |
|null      |550000    |1500       |null              |
|80222-7206|null      |453200     |214.99051233396585|
|80211-2636|null      |777200     |326.417471650567  |
|80218-1427|null      |387100     |218.45372460496614|
|80247-1515|84000     |121700     |142.33918128654972|
|80224-3411|null      |261300     |216.48715824357913|
|80220-2364|113000    |196700     |211.96120689655172|
```

READY

```
denverTab_with_RPSq.createOrReplaceTempView("denverview_RPSq")
```

READY

```
%md
<br>
#### In order to properly analyze the data and accurately represent the price ranges of the different
RPSq which was TOTAL_VALUE/IMP_AREA, to the data set. The extra column was added to better understand
column, Created a view called denverview_RPSq.
```

In order to properly analyze the data and accurately represent the price ranges of the different properties in Denver, I performed some further data preprocessing by adding an extra column, RPSq which was TOTAL_VALUE/IMP_AREA, to the data set. The extra column

was added to better understand the rate per square foot for each property type. As seen above, after adding the extra column, I created a view called denverview_RPSq.

Table 2

READY

```
%sql select SITUS_ZIP, D_CLASS_CN, count(*)
from denverView_RPSq where SITUS_ZIP like '80212%'
group by SITUS_ZIP, D_CLASS_CN
LIMIT 20
```

READY

SITUS_ZIP	D_CLASS_CN
80212-1966	SINGLE FAMILY
80212-2558	SINGLE FAMILY
80212-1147	ROWHOUSE
80212-1345	APT W/2 UNITS
80212-1962	SINGLE FAMILY
80212-2945	SINGLE FAMILY
80212-2149	SINGLE FAMILY
80212-2424	RESTAURANT
80212-1916	RETAIL, SINGLE

The table above displays the count of property types grouped by specific zip codes which start with '80212'.

READY

Table 3

READY

```
%sql select D_CLASS_CN from denverview_with_RPSq
```

READY

D_CLASS_CN
SINGLE FAMILY
SINGLE FAMILY
VCNT LAND - RES RATIO
VCNT LAND

SINGLE FAMILY
VCNT LAND
ROWHOUSE
VCNT LAND I 0 ZONE

```
Output exceeds 102400. Truncated.
```

This table just specifies the different property types given in the CSV file. READY

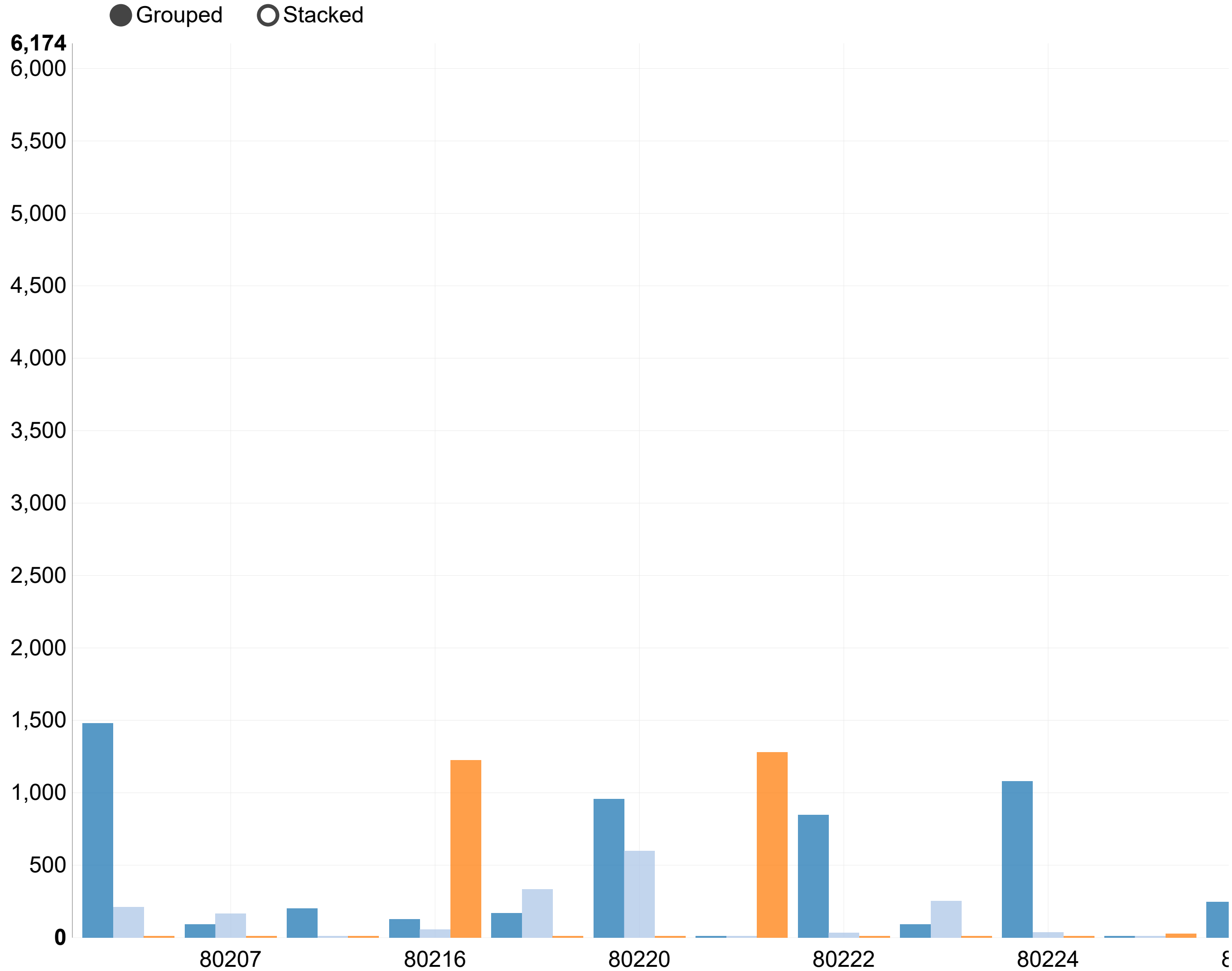
Graph 1

READY

```
%sql select substr(SITUS_ZIP, 1, 5), substr(D_CLASS_CN, 1, 5), count(*)
from denverview_RPSq
where (substr(D_CLASS_CN, 1, 5)='SINGL' OR
substr(D_CLASS_CN, 1, 3)='APT' OR
substr(D_CLASS_CN, 1, 5)='CONDO' OR
substr(D_CLASS_CN, 1, 5)='ROWHO') AND
substr(SITUS_ZIP, 1, 2)!='CO' AND
substring(SITUS_ZIP, 1, 5) IS NOT NULL AND
SALE_PRICE IS NOT NULL AND
TOTAL_VALUE IS NOT NULL
group by substr(SITUS_ZIP, 1, 5), substr(D_CLASS_CN, 1, 5)
having max(TOTAL_VALUE)<800000 and count(*)>25
order by 1
```

READY

settings



READY

This first graph displays the counts of each residential property types, 'SINGLE FAMILY', 'CONDOMINIUM', 'ROWHOUSE', 'APT', per zip codes in Denver that were recorded in the CSV file. This data is more interesting for potential homeowners so they can see which zip codes have more residential housing available. The data was filtered to remove the outliers and the null values so that the tables/graphs could accurately display the typical trends.

Graph 2

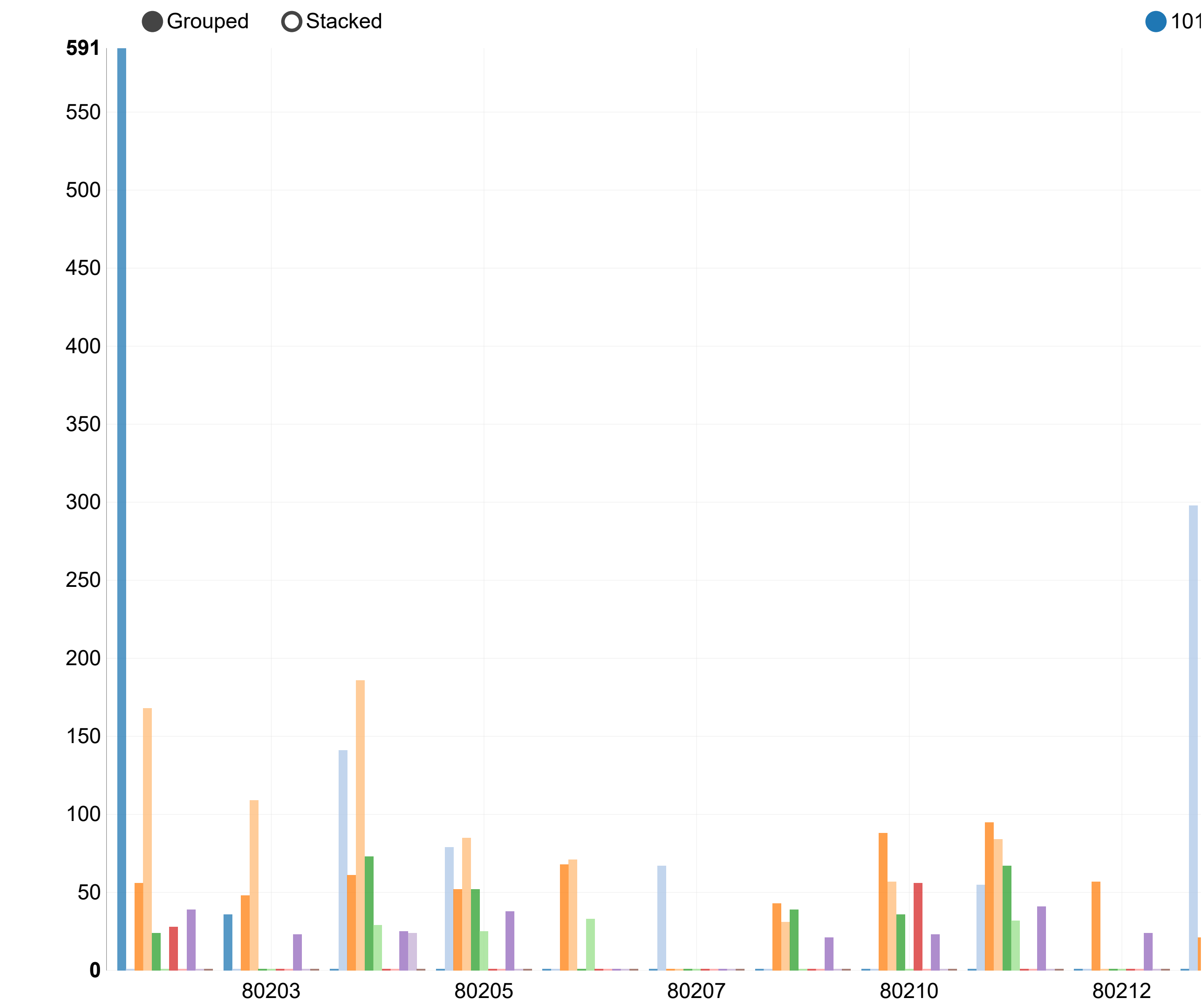
READY

```
%sql select substr(SITUS_ZIP, 1, 5),substr(D_CLASS_CN,1,3),count(*)
from denverview_RPSq
where ((substr(D_CLASS_CN,1,3)!='SIN' AND
substr(D_CLASS_CN,1,3)!='ROW' AND
substr(D_CLASS_CN,1,3)!='CON' AND
substr(D_CLASS_CN,1,3)!='APT')) AND
substr(SITUS_ZIP, 1, 5)!='CO' AND
substring(SITUS_ZIP, 1, 5) is NOT NULL AND
```

READY

```
SALE_PRICE is NOT NULL AND
TOTAL_VALUE is NOT NULL AND
substring(SITUS_ZIP, 1, 1) ='8'
group by substring(SITUS_ZIP, 1, 5),substr(D_CLASS_CN,1,3)
```

settings



This graph is the same as above but more useful for property buyers as they can see the amount and what kind of non-residential property types per zip code there are. READY

Graph 3

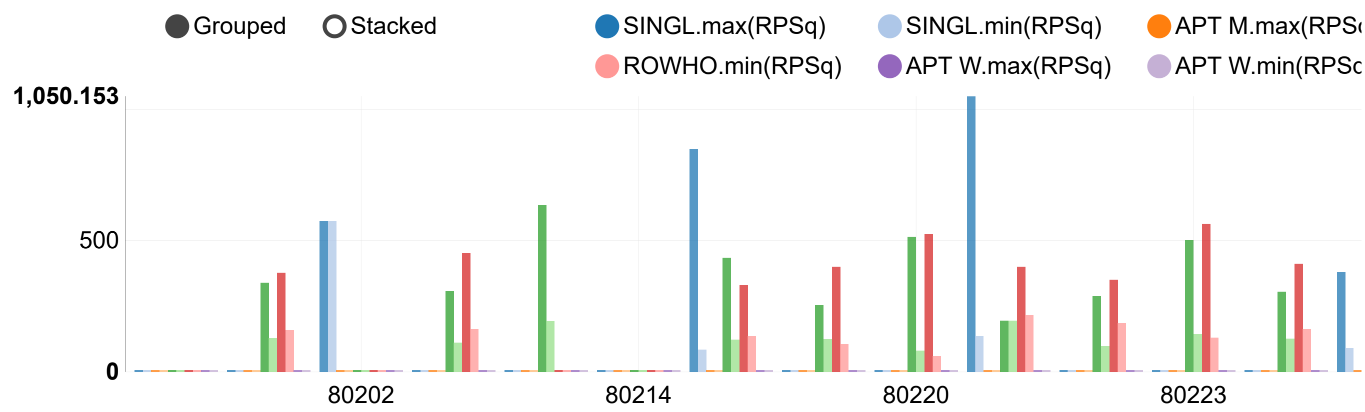
READY

```
%sql select substr(SITUS_ZIP, 1, 5), substr(D_CLASS_CN, 1, 5), max(RPSq), min(RPSq)
from denverview_RPSq where (substr(D_CLASS_CN, 1, 5)= 'SINGL' OR
substr(D_CLASS_CN, 1, 3)='APT' OR
substr(D_CLASS_CN, 1, 5)='CONDO'OR
substr(D_CLASS_CN, 1, 5)='ROWHO') AND
```

READY

```
substr(SITUS_ZIP, 1, 2)!='CO'AND
substr (SITUS_ZIP, 1, 5) IS NOT NULL AND
SALE_PRICE IS NOT NULL AND
TOTAL_VALUE IS NOT NULL
group by substr(SITUS_ZIP, 1, 5), substr(D_CLASS_CN, 1, 5)
having max(TOTAL_VALUE)<800000
```

settings



READY

The third and final graph for the Intermediate Report displays the minimum and maximum rate per square foot of each residential property types per zip code. This is really useful for potential buyers or real estate agents so that they can get a feel for the range of values that the properties have per zip code.

READY

For the intermediate milestone, I just wanted to get a basic feel of the data and highlight the differences between the types of properties in Denver and how much of each type there is for each of the zip codes. This allowed me to get a better understanding of the property prices per residential and non-residential property types per zip code.

READY

II. Machine Learning Experiments and Further Analysis

READY

For the final report, I wanted to conduct an in-depth analysis to view trends that would help me understand the real-estate market of Denver. In the next couple of graph and tables, I further analyze trends to find any major correlations between:

-RPSq and IMP_AREA

-SALE_YEAR and SALE_PRICE

-SALE_YEAR and RPSq

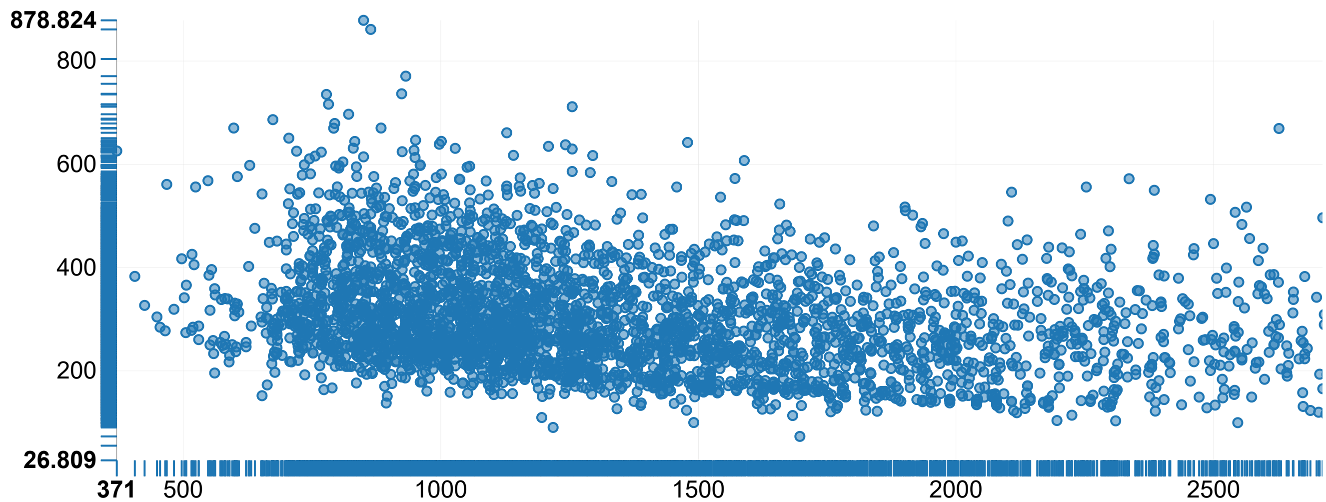
Graph 4

READY

```
%sql select IMP_AREA, RPSq,  substr(D_CLASS_CN, 1, 3)
from denverview_RPSq
where substr(D_CLASS_CN,1,3)='SIN' AND
RPSq is not null AND
IMP_AREA is not null AND
RPSq>0 AND
RPSq<1000 AND
IMP_AREA> 0 AND
IMP_AREA<5000
```

READY

settings



Output exceeds 102400. Truncated.

```
import org.apache.spark.sql.SparkSession
import spark.implicits._

val spark = SparkSession
  .builder()
  .appName("Intermediate Data")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
```

READY

```
import org.apache.spark.sql.SparkSession
import spark.implicits._
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@21a53b6f
```

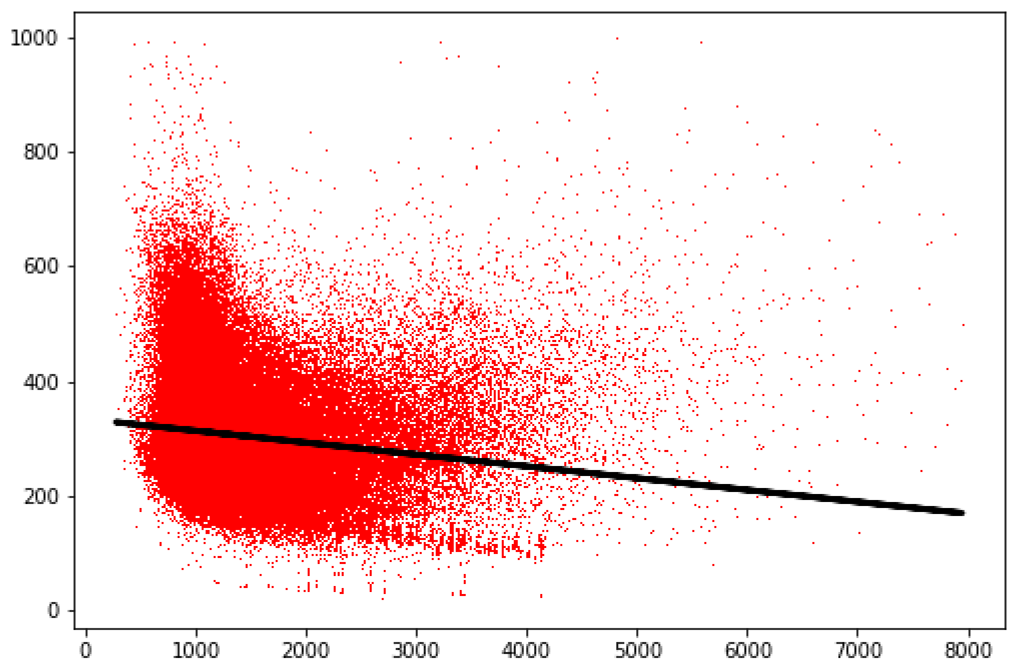
Graph 5

READY

READY

```
%pyspark

import matplotlib.pyplot as plt
import scipy
from sklearn import datasets, linear_model
res=spark.sql("select IMP_AREA, RPSq,  substr(D_CLASS_CN, 1, 3) from denverview_RPSq where substr(D_CL
    <8000")
res=res.toPandas()
x=res.IMP_AREA.values
y=res.RPSq.values
x=x.reshape(-1,1)
y=y.reshape(-1,1)
regr=linear_model.LinearRegression()
regr.fit(x,y)
plt.plot(x,y,'r,')
plt.plot(x, regr.predict(x), color='black', linewidth=3)
plt.axis()
plt.show()
```



READY

The two graphs above display the same data points, RPSq per IMP_AREA, for single family homes in all of Denver. The y-axis is all of the RPSq values and the x-axis is all of the IMP_AREA values. I wanted to see if the area got larger, would it have any effect on how much the land is worth per square foot. This type of information is very useful for property builders if they want to build single family homes in Denver and see what price would fit in their budget to build new properties. In the second graph, I wanted to get a better understanding of what the trend between the RPSq and the IMP_AREA is. Since the plot is very scattered and sparse on the edges, I wanted to display the trend of the market currently on single family homes in Denver. In order to do that, I used sklearn liner_model.LinearRegression(). This function helps fit the current dataset and take the x and y values to make a linear prediction of how the trend is going. For now, I didn't want to use linear regression to predict any values, but I wanted to use it to get a general idea of how the trend is going and if the market for single family houses is affordable. In the second graph, the linear regression is plotted as a black line. From that black line, one can tell that the trend is going downwards. So, as the IMP_AREA increases, the RPSq decreases. It seems like it would be more affordable to get a property that has greater area.

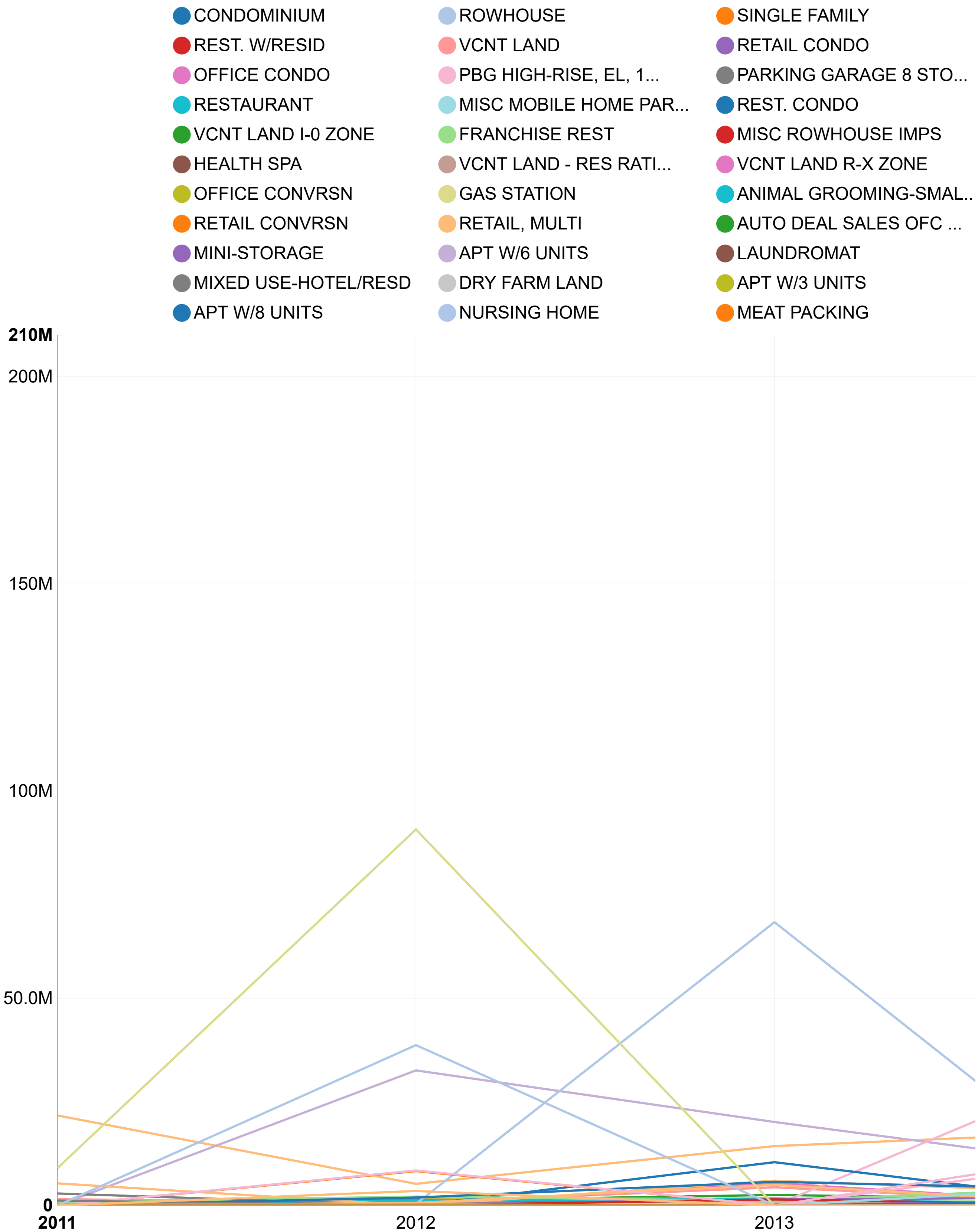
Graph 6

READY

```
%sql select SALE_YEAR, SALE_PRICE, D_CLASS_CN
from denverview_RPSq
where SALE_YEAR is not null
and SALE_PRICE is not null
and SALE_PRICE!=0
and SALE_PRICE>100
and SALE_YEAR>2010
```

READY

settings




```
Output exceeds 102400. Truncated.
```

READY

Another thing that I wanted to accomplish through this project is to view the yearly trends on the sale price of different property types, which I accomplished through the graph above. It seems as if for 2016, not enough data was accumulated, but the trends in property sale prices is clearly visible in 2015. You can tell which properties' sale prices are on the rise and which properties' sale prices are decreasing. This graph is very useful if property owners want to know if their properties can be sold in the market based of the average sale price.

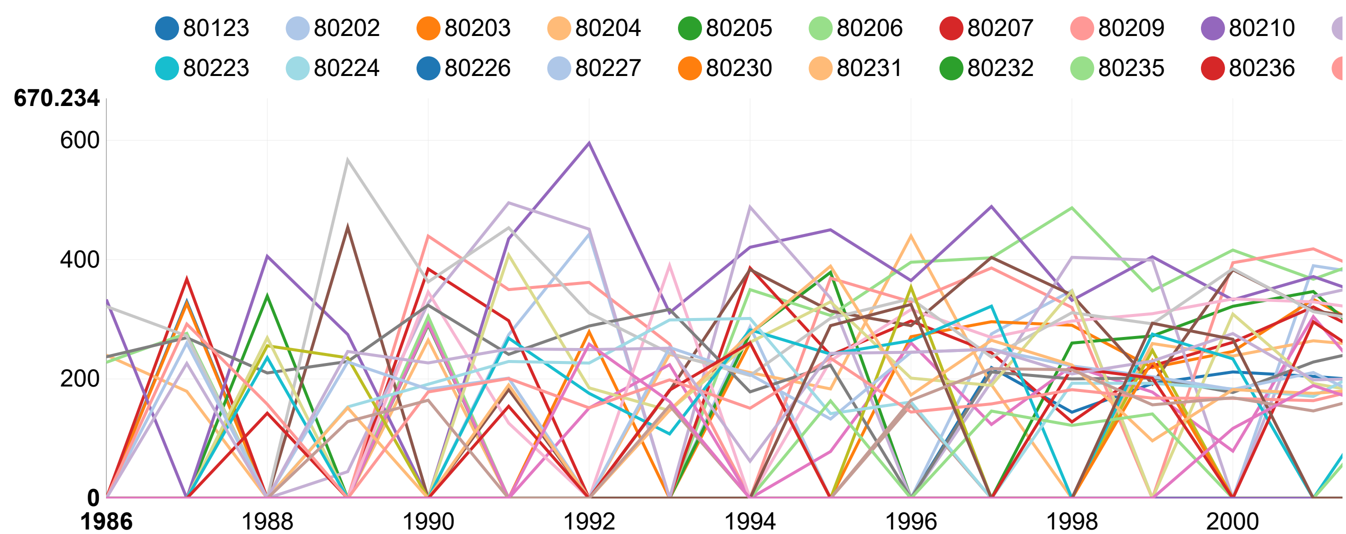
Graph 7

READY

```
%sql select SALE_YEAR, RPSq, substr(SITUS_ZIP, 1, 5) from denverview_RPSq where RPSq is not null AND S
```

READY

settings




```
Output exceeds 102400. Truncated.
```

READY

In the graph above, I wanted to see if there were any trends between the RPSq and years for each zip code. From the graph, it seems like there is no clear trend that is happening overall, but per zip code, you can see if the rate per square foot (RPSq) is increasing or decreasing per year. The two graphs above are just to get a general idea of what the sale price and rpsq trends are like throughout the year and throughout different zip codes. It is useful for buyers and sellers to know when is a good time and where is a good place to sell/buy property.

Table 4

READY

```
%sql select distinct(substr(SITUS_ZIP, 1, 5)) from denverview_RPSq
```

READY

substring(SITUS_ZIP, 1, 5)
80218
80249
80224
80227
80290
80299
80210
80220
80238

Graph 8

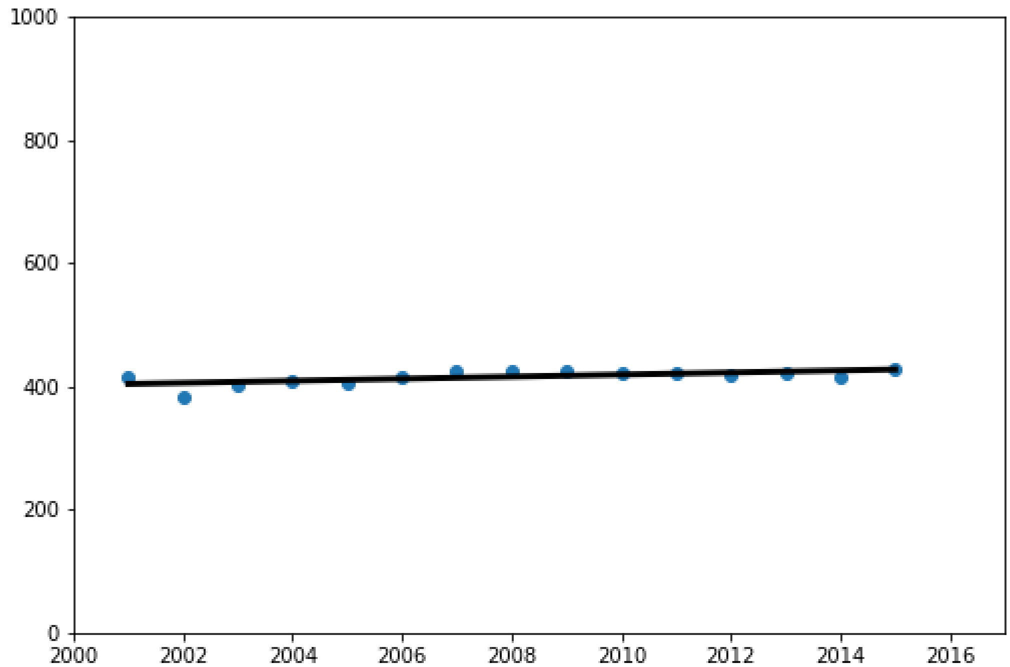
READY

READY

```
%pyspark
import matplotlib.pyplot as plt
import scipy
from sklearn import datasets, linear_model
res=spark.sql("select SALE_YEAR, avg(RPSq) RPSq, substr(SITUS_ZIP, 1, 5), count(*) from denverview_RPS
    AND substr(SITUS_ZIP, 1, 5)=80209 Group by SALE_YEAR, substr(SITUS_ZIP, 1, 5) order by SALE_YEAR "
res=res.toPandas()
print(res)
x=res.SALE_YEAR.values
y=res.RPSq.values
x=x.reshape(len(x),1)
y=y.reshape(len(y),1)
regr=linear_model.LinearRegression()
regr.fit(x,y)
pre= regr.predict(2017)
plt.scatter(x,y)
plt.plot(x, regr.predict(x), color='black', linewidth=3)
plt.axis([2000, 2017, 0, 1000])
plt.show()
```

```
   SALE_YEAR        RPSq substring(SITUS_ZIP, 1, 5)  count(1)
0       2001  416.156082                      80209       157
1       2002  383.478114                      80209       220
2       2003  403.759297                      80209       274
3       2004  408.986885                      80209       289
4       2005  406.331633                      80209       280
5       2006  416.886402                      80209       326
6       2007  424.770435                      80209       291
7       2008  425.322874                      80209       320
8       2009  425.344218                      80209       373
9       2010  422.208650                      80209       401
10      2011  423.664091                      80209       425
11      2012  418.142977                      80209       644
```

```
12        2013  422.588712                    80209        847
13        2014  416.787836                    80209        787
14        2015  427.175366                    80209        886
```



READY

The graph above conducts another linear regression machine learning algorithm on RPSq but this time it is on a particular zip cade as opposed to on a particular property type. In SQL statement, I have specified a particular zip code, 80209, that I want to know the average RPSq for each year. If you wanted to view the linear regression on a different zip code, one could change the zip code in the SQL statement to another zip code from the table (Table 4) above this graph. One could see the trends on RPSq throughout the years and see which zip code has the most affordable prices or which zip code would bring in the most sale revenue.

Graph 9

READY

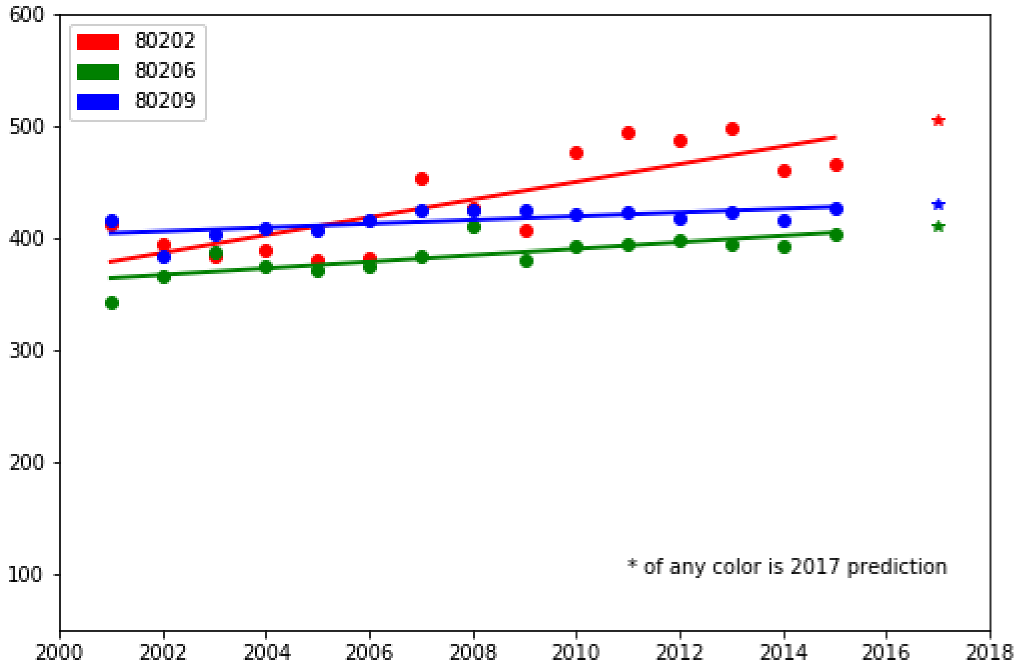
READY

```
%pyspark
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import scipy
from sklearn import datasets, linear_model
res=spark.sql("select SALE_YEAR, avg(RPSq) RPSq, substr(SITUS_ZIP, 1, 5) ZIP, count(*) countval from d
    is not null AND substr(SITUS_ZIP, 1, 5) in (select substr(SITUS_ZIP, 1, 5) from denverview_RPSq wh
    order by avg(RPSq) desc limit 4) Group by SALE_YEAR, substr(SITUS_ZIP, 1, 5) order by substr(SITUS
res2=res.filter(res["countval"]>50).toPandas()
plot1=res2[0:15]
plot2=res2[15:30]
plot3=res2[30:45]
plot1x=plot1.SALE_YEAR.values
```

```
plot1y=plot1.RPSq.values
zip1=plot1.ZIP.values[0]
plot1x=plot1x.reshape(len(plot1x),1)
plot1y=plot1y.reshape(len(plot1y),1)
regr=linear_model.LinearRegression()
regr.fit(plot1x,plot1y)
pre= regr.predict(2017)
print(plot1)
print ("2017 Prediction: ")
print(pre)
plt.plot(plot1x,plot1y, 'ro')
plt.plot(plot1x, regr.predict(plot1x), color='red', linewidth=2)
plt.plot(2017, pre, 'r*')
###
plot2x=plot2.SALE_YEAR.values
plot2y=plot2.RPSq.values
plot2x=plot2x.reshape(len(plot2x),1)
plot2y=plot2y.reshape(len(plot2y),1)
zip2=plot2.ZIP.values[0]
regr=linear_model.LinearRegression()
regr.fit(plot2x,plot2y)
pre= regr.predict(2017)
print(plot2)
print ("2017 Prediction: ")
print(pre)
plt.plot(plot2x,plot2y, 'go')
plt.plot(plot2x, regr.predict(plot2x), color='green', linewidth=2)
plt.plot(2017, pre, 'g*')
###
plot3x=plot3.SALE_YEAR.values
plot3y=plot3.RPSq.values
zip3=plot3.ZIP.values[0]
plot3x=plot3x.reshape(len(plot3x),1)
plot3y=plot3y.reshape(len(plot3y),1)
regr=linear_model.LinearRegression()
regr.fit(plot3x,plot3y)
pre= regr.predict(2017)
print(plot3)
print ("2017 Prediction: ")
print(pre)
plt.plot(plot3x,plot3y, 'bo')
plt.plot(plot3x, regr.predict(plot3x), color='blue', linewidth=2)
plt.plot(2017, pre, 'b*')
###
plt.axis([2000, 2018, 50, 600])
label1 = mpatches.Patch(color='red', label=zip1)
label2 = mpatches.Patch(color='green', label=zip2)
label3 = mpatches.Patch(color='blue', label=zip3)
first_legend = plt.legend(handles=[label1, label2, label3])
ax = plt.gca().add_artist(first_legend)
plt.text(2011, 100, "* of any color is 2017 prediction")
plt.show()
```

```
    SALE_YEAR        RPSq    ZIP  countval
0        2001  413.174263  80202        76
1        2002  394.328256  80202       119
2        2003  383.584910  80202       131
3        2004  389.884821  80202       105
4        2005  380.210405  80202       226
5        2006  382.327198  80202       161
6        2007  453.432611  80202       311
7        2008  425.984155  80202       173
8        2009  406.907576  80202       121
9        2010  476.715824  80202       272
10       2011  495.326280  80202       259
11       2012  488.326061  80202       422
```

```
12        2013  498.235728  80202       485
13        2014  460.655847  80202       503
14        2015  465.431688  80202       523
2017 Prediction:
```



READY

This last graph accomplishes the major project goal that I had. In this graph, I plot the average yearly RPSq values for the top 3 zip codes. Furthermore, I go onto predicting the RPSq for those three zip codes for the year 2017. The way I go about accomplishing this is by taking all the RPSq values for the year 2015, since there seems to be a lot of missing values for 2016, and picking the three zip codes that have the highest RPSq values. Then I run a linear regression on each of the zip codes. Based off of the graph above, 80202, 80209, and 80214 were the three zip codes with the highest RPSq values in 2015. I plot the (RPSq, SALE_YEAR) for those three zip codes. Then taking the result of the linear regression, I made a prediction for the RPSq value for those three zip codes for the year 2017. I then plotted the predicted value as a * in their respective zip code colors. So, the red * represents the 2017 prediction for the zip code 80202 and so on. This graph put the linear regression into use to make a prediction instead of looking a trend.

IV. Related Work

READY

READY

-Pagourtzi, Elli, et al. “Real estate appraisal: a review of valuation methods.” Journal of Property Investment & Finance21.4 (2003): 383-401. (http://trentglobal.com/assets/Real_estate%5B1%5D.pdf (http://trentglobal.com/assets/Real_estate%5B1%5D.pdf))

This article highlights the different ways machine learning and linear regression methods could be applicable to the real estate market. The article also further highlights which valuation methods would be useful and what datasets are interesting to property owners and buyers. Several estimation models for stepwise regression and other methods are given so one can see how data science and machine learning can be applied to real estate properties.

III and V. Evalute Results and Conclusion Section

READY

READY

Throughout the final report, I evaluate all of my results through graphs, tables, and explain it in words after each diagram. My methods were pretty accurate in accomplishing what I wanted to do in the initial project proposal. In my initial project proposal, there were some major points I wanted to cover which I accomplished in this Final Project submission. The first thing I wanted to cover was filtering the data by property types and zip codes which I accomplished during the intermediate report (Graph 1 and Graph 2). I wanted to also look at the average, max, and min market prices which I looked at the min and max during the intermediate report (Graph 3) and the average during the final report (Graph 7). Another main thing I wanted to look at were the yearly price trends and the property market price trends within the last 1 to 5 years (Graph 7, Graph 8, Graph 9). I wanted to look at the neighborhoods/zip codes that were on the rise (Graph 7, Graph 8). Finally, I wanted to answer questions like what are the market price trends for the next year (Graph 9) and which properties would be worth investing in based off of the trends(Graph 6). I mostly accomplished my goals through SQL and Spark statements, but all the machine learning I wanted to do was accomplished by using sklearn linear_model.LinearRegression() methods. Overall, everything I proposed was accomplished, but there are a couple things I would do if I had more time. I wanted to make sure there were better visualizations and that the data was user-interactive. So if someone wanted to view the data, they would be able to look at the graphs with a drop-down menu where they could change things like zip code or property types or they could look at a heat-map of Denver to look see which zip codes have the higher prices. I would basically make the graphs and table more interactive and user-friendly.

VI. Acknowledgements

READY

READY

Mostly all of the visualizations and graphs/tables are done through spark, SQL, and python commands. However for all the machine learning linear regression methods I used scikit-learn which is machine learning in python. It is built using NumPy, SciPy, and matplotlib. From this tool, I used sklearn.linear_model.LinearRegression() and the fit() and predict() methods that would fit a linear regression line on the scatter plot and would be used to predict already existing x values or future x values, like how I used it to predict the RPSq for 2017.

scikit-learn: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

```
%md
```

READY