Problem Statement

The real estate company has engaged your firm to build out a data product and provide your conclusions to help them understand which are the most profitable zip codes on the short-term rentals within New York region.

Author: Arpit Khandekar

```
# installing plotly and cufflinks libraries using pip command.
import sys
!{sys.executable} -m pip install cufflinks
!{sys.executable} -m pip install Plotly
```

...

```
# importing packages
from pandas import DataFrame, read_csv
import matplotlib.pyplot as plt
import seaborn as sns
from plotly import __version__
%matplotlib inline
import pandas as pd
# Plotly helps in building interactive visualization.
#Cufflinks directly binds plotly to Pandas dataframe.
import cufflinks as cf
from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot
# It saves the data visualization even if it crashes.
init_notebook_mode(connected=True)
cf.go_offline()
```

Importing Zillows data to forecast the values of properties in the year 2018

Zillow data which is considered as our cost data provides us an estimate of value for two-bedroom properties.

```
#import the zillows data for the year 2018 to forecast the value
file = r'F:\Arpit\Arpit_Stuff1\Full_Time_Prep\capital one\airbnb-zillow-data-challenge-master\Zip_Zhvi_2bedroom.csv'
# reading the csv file
Zillow_df = pd.read_csv(file)
# showing top elements
Zillow_df.head()
```

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...	2016-09	2016-10	2016-11	2016-12	2017-01
0	61639	10025	New York	NY	New York	New York	1	NaN	NaN	NaN	...	1374400	1364100	1366300	1354800.0	1327500
1	84654	60657	Chicago	IL	Chicago	Cook	2	167700.0	166400.0	166700.0	...	368600	370200	372300	375300.0	378700
2	61637	10023	New York	NY	New York	New York	3	NaN	NaN	NaN	...	1993500	1980700	1960900	1951300.0	1937800
3	84616	60614	Chicago	IL	Chicago	Cook	4	195800.0	193500.0	192600.0	...	398900	401200	403200	405700.0	408300
4	93144	79936	El Paso	TX	El Paso	El Paso	5	59100.0	60500.0	60900.0	...	82400	82300	82400	82300.0	82500

5 rows × 262 columns

```
#  to find the number of rows and columns
Zillow_df.shape
```

(8946, 262)

```python
# For Filtering purpose we are using State and Metro columns as City column is not cleaned
Zillow_df[(Zillow_df["State"] == "NY") & (Zillow_df["Metro"] == "New York")]['City']
```

```
0                     New York
2                     New York
13                    New York
14                    New York
20                    New York
31                    New York
51                    New York
67                    New York
70                    New York
108                   New York
189                   New York
211                    Yonkers
378                   New York
437                 Huntington
579                   New York
607           Town of Newburgh
621                   New York
642                 Middletown
667                   New York
763                   New York
783              Town of Islip
893                   New York
1078                 Patchogue
1179                Long Beach
1228      Town of Poughkeepsie
1322                    Ramapo
1408                Huntington
1427      Town of Poughkeepsie
1554                  New York
1743                  New York
                 ...
7244            Greenwood Lake
7277               Philipstown
7347                Washington
7527                     Dover
7563                Orangetown
7650            Highland Falls
7851           West Haverstraw
7854             East Fishkill
7884               Westhampton
7891                   Florida
7895                  Harriman
7912               Philipstown
7968                 Hyde Park
7975                     Dover
8014              Hamptonburgh
8040                    Tuxedo
8065                North East
8165        Cornwall on Hudson
8216       Town of Pine Plains
8243                  Stanford
8275           Town of Pawling
8360                    Ramapo
8401    Town of Shelter Island
8445                  Yorktown
8452                   Mt Hope
8475                  Deerpark
8501                    Tivoli
8506                  Stanford
8527           Cortlandt Manor
8749                  Deerpark
Name: City, Length: 156, dtype: object
```

```
#To check elements
Zillow_df.head()
```

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...	2016-09	2016-10	2016-11	2016-12	2017-01
0	61639	10025	New York	NY	New York	New York	1	NaN	NaN	NaN	...	1374400	1364100	1366300	1354800.0	1327500
1	84654	60657	Chicago	IL	Chicago	Cook	2	167700.0	166400.0	166700.0	...	368600	370200	372300	375300.0	378700
2	61637	10023	New York	NY	New York	New York	3	NaN	NaN	NaN	...	1993500	1980700	1960900	1951300.0	1937800
3	84616	60614	Chicago	IL	Chicago	Cook	4	195800.0	193500.0	192600.0	...	398900	401200	403200	405700.0	408300
4	93144	79936	El Paso	TX	El Paso	El Paso	5	59100.0	60500.0	60900.0	...	82400	82300	82400	82300.0	82500

5 rows × 262 columns

Zillow data preprocessing

#We are filtering the data for NY region and dropping the unnessecary coloumns.
We are pivoting the years coloumn for better data analysis.
Using splitting function in year coloumn as we just require year value for our analysis.
Changing the Region name coloumn to zipcode for better understanding.
Making sure that proper data type is used for year and zipcode coloumn.

```
# Filtering data for New York city and selecting columns
Zillow_df = Zillow_df[(Zillow_df["State"] == "NY") & (Zillow_df["Metro"] == "New York")].copy()
```

```
# To find and dropping Unnecessary columns
Zillow_df.drop(["RegionID","City","State","Metro","CountyName","SizeRank"],axis=1,inplace=True)
```

```
Zillow_df.head()
```

	RegionName	1996-04	1996-05	1996-06	1996-07	1996-08	1996-09	1996-10	1996-11	1996-12	...	2016-09	2016-10	2016-11	2016-12	2017-01	2017-02	2017-03	2017-
0	10025	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1374400	1364100	1366300	1354800.0	1327500	1317300	1333700	13521
2	10023	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1993500	1980700	1960900	1951300.0	1937800	1929800	1955000	20224
13	10128	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1526000	1523700	1527200	1541600.0	1557800	1582900	1598900	16461
14	10011	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2354000	2355500	2352200	2332100.0	2313300	2319600	2342100	23659
20	10003	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1932800	1930400	1937500	1935100.0	1915700	1916500	1965700	20453

5 rows × 256 columns

```python
# Using melting function which help in converting columns to rows
Zillow_df = pd.melt(Zillow_df, id_vars=['RegionName'], value_vars=list(Zillow_df.columns[6:])).copy()
```

```python
# To check how melting function has made changes on the data
Zillow_df.head()
```

	RegionName	variable	value
0	10025	1996-09	NaN
1	10023	1996-09	NaN
2	10128	1996-09	NaN
3	10011	1996-09	NaN
4	10003	1996-09	NaN

```python
#Using spilt function to split year and month as we focus on year for forecasting
Zillow_df[["year","month"]] = Zillow_df["variable"].str.split(pat="-", n=-1, expand=True).copy()
```

```python
# To see how the splitting has been done.
Zillow_df.head()
```

	RegionName	variable	value	year	month
0	10025	1996-09	NaN	1996	09
1	10023	1996-09	NaN	1996	09
2	10128	1996-09	NaN	1996	09
3	10011	1996-09	NaN	1996	09
4	10003	1996-09	NaN	1996	09

```python
# For analysis changing data type of year to int
Zillow_df["year"] = Zillow_df["year"].astype(int).copy()
```

```python
 # To use melting and splitting function we will drop the unneccessary coloumns
Zillow_df.drop(["variable","month"],axis=1,inplace=True)
```

```python
# To see how the data is seen after dropping coloumns
Zillow_df.head()
```

	RegionName	value	year
0	10025	NaN	1996
1	10023	NaN	1996
2	10128	NaN	1996
3	10011	NaN	1996
4	10003	NaN	1996

```python
# Renaming the Regionname coloumn to Zipcode
Zillow_df.rename(index=str, columns={"RegionName": "zipcode"},inplace=True)
```

```python
# Changing datatype of zipcode to string type
Zillow_df['zipcode']=Zillow_df['zipcode'].astype(str).copy()
```

```python
# To see how the how our final data lookslike
Zillow_df.head()
```

	zipcode	value	year
0	10025	NaN	1996
1	10023	NaN	1996
2	10128	NaN	1996
3	10011	NaN	1996
4	10003	NaN	1996

```python
# to check null values
Zillow_df.isnull().sum()
```

```
zipcode       0
value      4204
year          0
dtype: int64
```

Assumption used to build Machine learning models are as follow:

1. Since we see that the price of Apartments is increasing over the time in entire NY region. However, from 1996-2003, we have lots of Null values, therefore we need to handle those values or otherwise we will not take those values to build model.
2. In addition, we are not using data before the year 2012, we have two reason with us for not using this data the first reason is we have lot of null values for these time frame and second reason is we all know that there was recession during 2008-2011 the prices of all the real estates has declined. In case, if we use this data to build our model it could mislead our model in predicting correctly.
3. We will build various models which will help in prediction of values of properties in year 2018 in order to calculate profit percentage.

Building decision tree using the preprocessed data

To build the model selecting the data for certain time frame and replacing the null value with mean values for corresponding zipcode.

```
# Making Decision tree by considering some assumptions
DecisionTree_df = Zillow_df.copy()
```

```
# selecting the values between year 2012 and 2017
DecisionTree_df = DecisionTree_df[(DecisionTree_df["year"]>=2012) & (DecisionTree_df["year"]<=2017)].copy()
# finding unique year values from the data
DecisionTree_df['year'].unique()
```

```
array([2012, 2013, 2014, 2015, 2016, 2017], dtype=int64)
```

```
# Treating the null values by replacing those with the mean values for corresponding RegionName category.
# defining lambda fuction which can take number of arguments
DecisionTree_df['value'] = DecisionTree_df.groupby(["zipcode","year"]).value.transform(lambda x: x.fillna(x.mean()))
```

```
# To check null values
DecisionTree_df.isnull().sum()
```

```
zipcode    0
value      0
year       0
dtype: int64
```

```python
# To check the changes happened in data which will show there is no null values
DecisionTree_df.head()
```

	zipcode	value	year
28704	10025	904400.0	2012
28705	10023	1395200.0	2012
28706	10128	1057500.0	2012
28707	10011	1509600.0	2012
28708	10003	1348500.0	2012

```python
# Selecting the matrix of feature and dependent variables
DecisionTree_df = DecisionTree_df.loc[:,["zipcode","year","value"]].copy()
```

```python
X = DecisionTree_df.iloc[:, :-1].values
print(X)
```

```
[['10025' 2012]
 ['10023' 2012]
 ['10128' 2012]
 ...
 ['12581' 2017]
 ['10537' 2017]
 ['12729' 2017]]
```

```python
y = DecisionTree_df.iloc[:, 2].values
print(y)
```

```
[ 904400. 1395200. 1057500. ...  224900.  184200.  105800.]
```

Build the model using training and testing data

```python
DecisionTree_df.head()
```

	zipcode	year	value
28704	10025	2012	904400.0
28705	10023	2012	1395200.0
28706	10128	2012	1057500.0
28707	10011	2012	1509600.0
28708	10003	2012	1348500.0

```python
# for Implementing the model we will Split the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning:

This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and fun
ctions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will
be removed in 0.20.
```

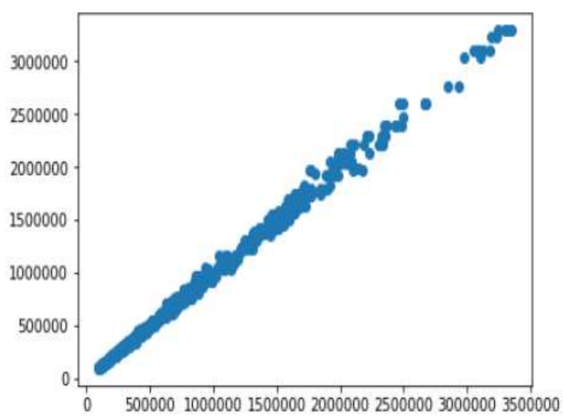
```python
# We will build Decision tree using Decision Tree Regressor function
from sklearn import tree
Dectree_model=tree.DecisionTreeRegressor()
Dectree_model.fit(X_train,y_train)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
          max_leaf_nodes=None, min_impurity_decrease=0.0,
          min_impurity_split=None, min_samples_leaf=1,
          min_samples_split=2, min_weight_fraction_leaf=0.0,
          presort=False, random_state=None, splitter='best')
```

Evaluating the performance on the test data set and depicting model performance on it by plotting graph.

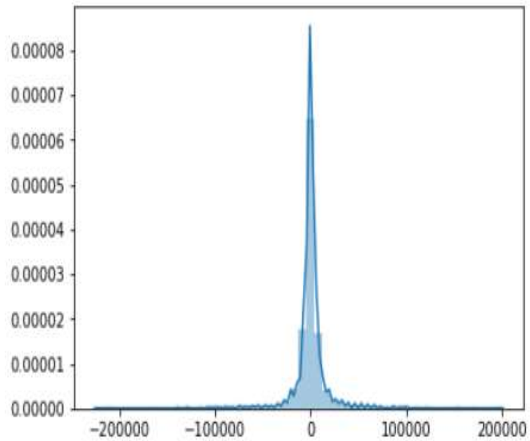
```python
# Evaluating performance on Test Data
predicitions=Dectree_model.predict(X_test)
Test_df=pd.DataFrame()
Test_df['A']=y_test
Test_df['B']=predicitions
tc=Test_df.corr()
#sns.heatmap(tc,annot=True,cmap='Spectral')
plt.scatter(y_test,predicitions,cmap='coolwarm')
# This plot shows that points on test data are not scattered and model is performing well on test data.
```

```
<matplotlib.collections.PathCollection at 0x18990c59f98>
```



```python
# This will show the error graph which is normaly distributed foster our model
sns.distplot(y_test-predicitions)
```

<matplotlib.axes._subplots.AxesSubplot at 0x18990ddacf8>



The error graph shows it is normally distributed which foster our model.

Now we will use our model for actual forecasting:

```python
# To see few elements of Tree
DecisionTree_df.head()
```

	zipcode	year	value
28704	10025	2012	904400.0
28705	10023	2012	1395200.0
28706	10128	2012	1057500.0
28707	10011	2012	1509600.0
28708	10003	2012	1348500.0

```python
# For prediction on actual data
DecisionTree_df = DecisionTree_df[DecisionTree_df["year"] == 2017].loc[:,["zipcode","year","value"]].copy()
DecisionTree_df["year"] = 2018
X_pred = DecisionTree_df.iloc[:, :-1].values

# We will Predict the results using our defined model
DecisionTree_df['predicted_value_2018'] = tree_model.predict(X_pred)

DecisionTree_df= DecisionTree_df.groupby("zipcode").mean()
DecisionTree_df.head()
```

zipcode	year	value	predicted_value_2018
10003	2018	2.016550e+06	2054475.0
10011	2018	2.373500e+06	2383250.0
10013	2018	3.224517e+06	3230550.0
10014	2018	2.473150e+06	2468100.0
10021	2018	1.717183e+06	1722920.0

Defining the column (hike in a year) which gives the difference between the predicted property value in year 2018 and original property value in the data set.

```
We found hike in one year which is nothing difference between pedicted property value in 2018 and  2017.
DecisionTree_df['hike_in_one_year'] = DecisionTree_df['predicted_value_2018'] - DecisionTree_df['value']
DecisionTree_df.head()
```

zipcode	year	value	predicted_value_2018	hike_in_one_year
10003	2018	2.016550e+06	2054475.0	37925.000000
10011	2018	2.373500e+06	2383250.0	9750.000000
10013	2018	3.224517e+06	3230550.0	6033.333333
10014	2018	2.473150e+06	2468100.0	-5050.000000
10021	2018	1.717183e+06	1722920.0	5736.666667

We can perform Model tuning is done with the help of different evaluation technique which is K-fold evaluation technique

```
# TO do Model Tunning we can apply k-FoldValidation
from sklearn.model_selection import cross_val_score
accuracy_tree=cross_val_score(estimator=tree_model,X=X_train,y=y_train,cv=10)
```

```
# 10 Accuracy that we are getting after dividing test data into 10 sets
accuracy_tree
```

```
array([0.99814825, 0.9975716 , 0.99814448, 0.99813278, 0.99261255,
       0.99768769, 0.99746718, 0.99844894, 0.99736798, 0.99724986])
```

Accuracy of model:

```
accuracy_tree.mean()
```

```
0.9972831311212185
```

```
# We found that bias variance trade off is very less. hecne model is trained really well
accuracy_tree.std()
```

```
0.001602335961363693
```

We have .99 accuracy mean and .0016 accuracy std for decision tree model.

Building Linear Regression model

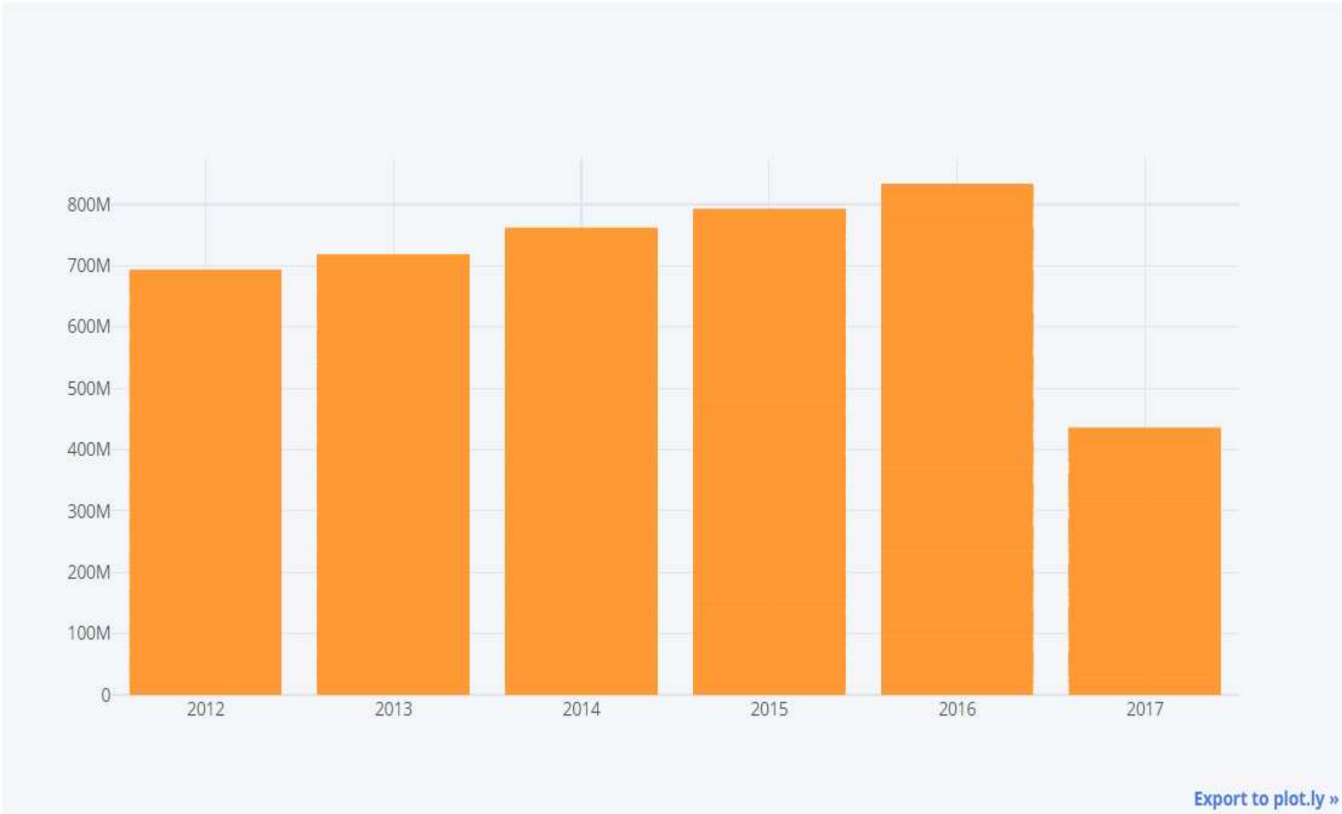
We are building the regression model using to predict the property values for 2018.
We are taking the data from 2012 to 2017, as there are many missing values in year 1996-2004.
Also, we are not taking the data during recession year that is from 2009-2011.

```
# We will build Linear Regression model to predict the values of 2018
LinearReg_df = Zillow_df.copy()
```

```
#To build a model We will take data from 2012 to 2017
LinearReg_df = LinearReg_df[(LinearReg_df["year"]>=2012) & (LinearReg_df["year"]<=2017)].copy()
LinearReg_df.head()
```

	zipcode	value	year
28704	10025	904400.0	2012
28705	10023	1395200.0	2012
28706	10128	1057500.0	2012
28707	10011	1509600.0	2012
28708	10003	1348500.0	2012

```
# Visualization of year v/s value in bar chart using linear model
LinearReg_df.iplot(kind='bar',x='year',y='value',colors='Orange')
```



Splitting the data into Training and testing data and implementing the Linear Regression model.

```python
# We will treat missing values by changing those values with the mean values for corresponding RegionName Category
LinearReg_df['value'] = LinearReg_df.groupby(["zipcode","year"]).value.transform(lambda x: x.fillna(x.mean()))

# In Linear Regression model we will Select Metrix of feature and dependent variable.
LinearReg_df = LinearReg_df.loc[:,["zipcode","year","value"]].copy()
X = LinearReg_df.iloc[:, :-1].values
y = LinearReg_df.iloc[:, 2].values
```

```python
# In Linear model there is a need to Encode the categorical data using LabelEncoder and OneHotEncoder
# Encoding the Independent Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
enc = LabelEncoder()
X[:,0] = enc.fit_transform(X[:,0])
onehotencoder = OneHotEncoder(categorical_features = [0])
X = onehotencoder.fit_transform(X).toarray().copy()
```

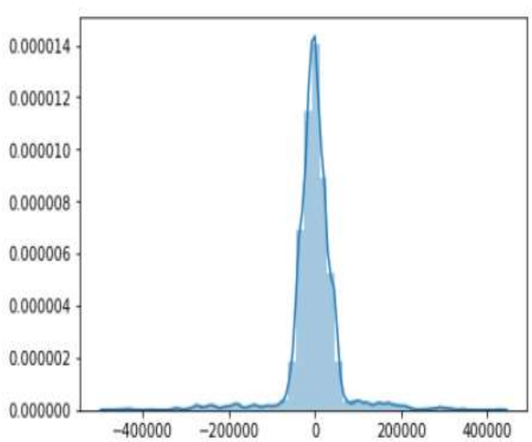
```python
# Splitting the dataset into the Training set and Test set to use in Linear model
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Accuracy on the model

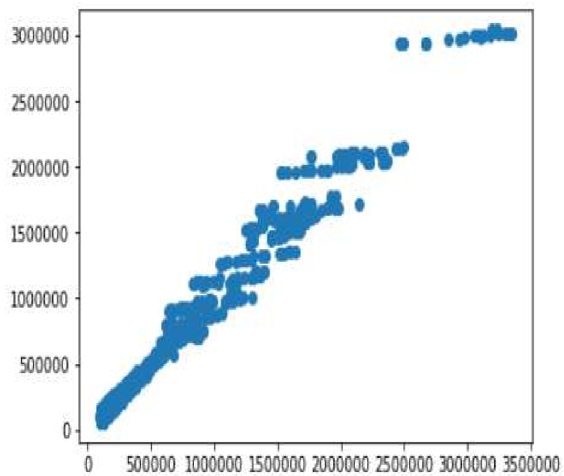
```python
# Finding the accuracy of Trained model by Predicting the Test set results
predictions=lm.predict(X_test)
sns.distplot(y_test-predictions)
# Graph shows that it is good model as error term is normally distributed
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x189904d0b38>
```



```
plt.scatter(y_test,predictions)
# Graphs shows that points are not too much scatterd hence by this model performed well on training set.
```

<matplotlib.collections.PathCollection at 0x18990552208>



<u>Above graph shows that model is performing well on training data set as it is not scattered.</u>

<u>Now we will use the Model for Actual forecasting.</u>

```
# Taking Actual data for prediction using Linear model
LinearReg_df= LinearReg_df[LinearReg_df["year"] == 2017].loc[:,["zipcode","year","value"]].copy()
LinearReg_df["year"] = 2018
X_pred = LinearReg_df.iloc[:, :-1].values
```

```
# Encoding categorical data w.hich can be used in Linear Regression model.
# Encoding the Independent Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
enc = LabelEncoder()
X_pred[:,0] = enc.fit_transform(X_pred[:,0])
onehotencoder = OneHotEncoder(categorical_features = [0])
X_pred = onehotencoder.fit_transform(X_pred).toarray()
```

```
# Predicting the results using Linear regression model lm
LinearReg_df["predicted_value_2018"] = lm.predict(X_pred)

#Hike in one year is varaible which help in finding the increase inv alue from 2017 to 2018
LinearReg_df["hike_in_one_year"] = LinearReg_df["predicted_value_2018"] - LinearReg_df["value"]
```

```
LinearReg_df = LinearReg_df.groupby("zipcode").mean()
# To check predictions using Linear Regression model.
LinearReg_df.head()
```

	year	value	predicted_value_2018	hike_in_one_year
zipcode				
10003	2018	2.016550e+06	1.795434e+06	-221115.653201
10011	2018	2.373500e+06	2.074999e+06	-298500.659907
10013	2018	3.224517e+06	3.062443e+06	-162073.761834
10014	2018	2.473150e+06	2.173671e+06	-299478.812302
10021	2018	1.717183e+06	1.538696e+06	-178487.593909

```python
# Plotting the graph between Zipcode and hike in one year values using iplot.
LinearReg_df.iplot(kind='bar',x='zipcode',y='hike_in_one_year',size=100,colors='red')
```

Model Tuning for better performance by implementing K-Fold evaluation technique.

```python
# Tuning the model by Applying k-FoldValidation
from sklearn.model_selection import cross_val_score
acc_linear=cross_val_score(estimator=lm,X=X_train,y=y_train,cv=10)
```

```python
# Accuracy mean value of Linear Regression model.
acc_linear.mean()
```

```
0.9799774618808268
```

```python
# Accuracy Std value of Linear Regression model.
acc_linear.std()
```

```
0.003072598016119257
```

We found that **Linear regression model** is having **Accuracy mean value as .97 and Accuracy Std value as 0.0030** which are **less than Decision tree model**. **<u>Hence, it is proved that Decision tree is the best model for this problem.</u>**

Importing Airbnb data

This data is considered as Revenue data and is the medium through which the investor plans to lease out their investment property.

```python
# Loading the Airbnb data
#import the Airbnb data for the year 2018 to forecast the value
file = r'F:\Arpit\Arpit_Stuff1\Full_Time_Prep\capital one\airbnb-zillow-data-challenge-master\listings.csv'
# reading the csv file
airbnb_df = pd.read_csv(file)
# showing top elements
airbnb_df.head()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2728: DtypeWarning:

Columns (43,88) have mixed types. Specify dtype option on import or set low_memory=False.
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2728: DtypeWarning:

Columns (43,88) have mixed types. Specify dtype option on import or set low_memory=False.
```

	id	listing_url	scrape_id	last_scraped	name	summary	space	description	experiences_offered	neighbo
0	7949480	https://www.airbnb.com/rooms/7949480	20170502132028	2017-05-03	City Island Sanctuary relaxing BR & Bath w Par...	Come relax on City Island in our quiet guest r...	On parle français et anglais, (lire Français c...	Come relax on City Island in our quiet guest r...	none	City sanctua
1	16042478	https://www.airbnb.com/rooms/16042478	20170502132028	2017-05-04	WATERFRONT STUDIO APARTMENT	My place is close to Sea Shore. You'll love my...	(URL HIDDEN)	My place is close to Sea Shore. You'll love my...	none	
2	1886820	https://www.airbnb.com/rooms/1886820	20170502132028	2017-05-04	Quaint City Island Community.	Quiet island boating town on Long Island Soun...	Master bed with queen bed, full bath and offi...	Quiet island boating town on Long Island Soun...	none	Small N town i
3	6627449	https://www.airbnb.com/rooms/6627449	20170502132028	2017-05-05	Large 1 BDRM in Great location	This ground floor apartment is light and airy ...	We are close to fishing, boating, biking, hors...	This ground floor apartment is light and airy ...	none	City and a hi
4	5557381	https://www.airbnb.com/rooms/5557381	20170502132028	2017-05-04	Quaint City Island Home	Located in an old sea-shanty town, our home ha...	You won't find a place so close to the city (N...	Located in an old sea-shanty town, our home ha...	none	City I two v

Data Preprocessing for Airbnb Data

As we seen that weekly, monthly price is having a lot of missing values. So, we will take price column to calculate revenue earned.
We are filtering the data based on NY region.
Also, we are focusing on analysis having room type as Entire house only. As other rooms are not interested in analyzing.
We are interested in analyzing only 2BHK properties thus we add filters accordingly.
Cleaning is done on Price columns by removing '$' symbol and ',' which can be used as calculation field.

```python
# Below are the Features which can be selected for analysis for Airbnb data
airbnb_df = airbnb_df[["state","zipcode","market","country","latitude","longitude","price","bedrooms","room_type"]].copy()
```

```python
# Filtering conditions as we need specific data from Airbnb data to solve our problem.
airbnb_df = airbnb_df[(airbnb_df["state"] == "NY") & (airbnb_df["market"] == "New York") & \
                    (airbnb_df["country"] == "United States") & (airbnb_df['room_type'] == 'Entire home/apt')\
                    & (airbnb_df['bedrooms'] == 2)][["zipcode","price"]].copy()
```

```
# Price columns is cleaned by removing unwanted symbols.
airbnb_df["price"] = airbnb_df.price.str.replace('$','').copy()
airbnb_df["price"] = airbnb_df.price.str.replace(',','').copy()
```

```
# For analysis changing data type of year to int
airbnb_df["price"] = airbnb_df["price"].astype(float).copy()
```

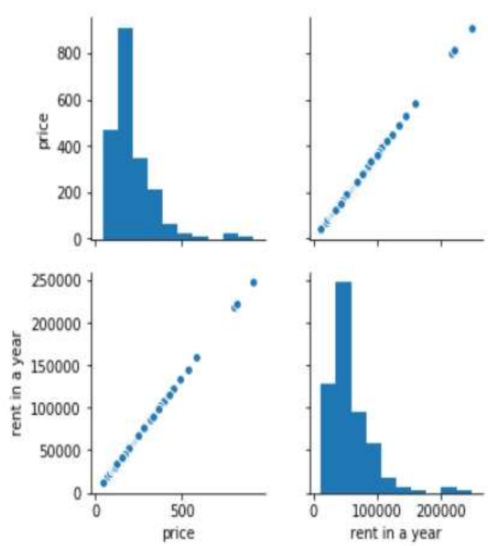
```
# Creating new field for rent value in a year by using the given condition 75 % occupancy(273 days) in a year.
airbnb_df["rent in a year"] = 273 * airbnb_df["price"]
```

```
# Finding average price per zipcode
airbnb_df = airbnb_df.groupby("zipcode").mean().copy()
airbnb_df.head()
```

	price	rent in a year
zipcode		
10001.0	247.800000	67649.400000
10002.0	249.000000	67977.000000
10003.0	327.523810	89414.000000
10010.0	301.407407	82284.222222
10011.0	372.466667	101683.400000

```
# Seaborn library gives you pairplot which is to visualize the relationship between two variables
sns.pairplot(airbnb_df)
```

```
<seaborn.axisgrid.PairGrid at 0x189a7e92e80>
```



Merging the data from the decision tree (chosen model) with Airbnb to do final analysis

```
# Concatinating the airbnb data from best model(decision tree) having best accuracy
result = pd.concat([DecisionTree_df, airbnb_df], axis=1, join='inner')
result
```

zipcode	year	value	predicted_value_2018	hike_in_one_year	price	rent in a year
10003	2018	2.016550e+06	2.054475e+06	37925.000000	326.466019	89125.223301
10011	2018	2.373500e+06	2.383250e+06	9750.000000	369.082353	100759.482353
10013	2018	3.224517e+06	3.230550e+06	6033.333333	393.693548	107478.338710
10014	2018	2.473150e+06	2.468100e+06	-5050.000000	332.505747	90774.068966
10021	2018	1.717183e+06	1.722920e+06	5736.666667	296.058824	80824.058824
10022	2018	1.884400e+06	1.884400e+06	0.000000	375.228571	102437.400000
10023	2018	2.013717e+06	1.988000e+06	-25716.666667	296.620690	80977.448276
10025	2018	1.358600e+06	1.391550e+06	32950.000000	293.140187	80027.271028
10028	2018	1.909750e+06	1.931480e+06	21730.000000	275.322581	75163.064516
10036	2018	1.729167e+06	1.731375e+06	2208.333333	445.666667	121667.000000
10128	2018	1.648883e+06	1.688467e+06	39583.333333	238.836735	65202.428571
10304	2018	3.124167e+05	3.106000e+05	-1816.666667	97.000000	26481.000000
10305	2018	4.070333e+05	4.078000e+05	766.666667	121.625000	33203.625000
10306	2018	3.400500e+05	3.254000e+05	-14650.000000	93.000000	25389.000000
10312	2018	3.455000e+05	3.449200e+05	-580.000000	215.000000	58695.000000
11201	2018	1.402783e+06	1.407100e+06	4316.666667	211.789474	57818.526316
11215	2018	1.050483e+06	1.048633e+06	-1850.000000	181.411765	49525.411765
11217	2018	1.247833e+06	1.244700e+06	-3133.333333	207.052632	56525.368421
11231	2018	1.196583e+06	1.193560e+06	-3023.333333	202.323529	55234.323529
11234	2018	4.719000e+05	4.708600e+05	-1040.000000	118.500000	32350.500000
11434	2018	3.716333e+05	3.681250e+05	-3508.333333	155.000000	42315.000000

Calculating Percentage profit based on the initial property cost.

Calculating the percentage profit which is earned relative to the initial investment value on a property.
Below is the formula which we will be using to calculate percentage profit.

Profit percentage = ("rent in a year" + "hike_in_one_year") / "initial cost of property"

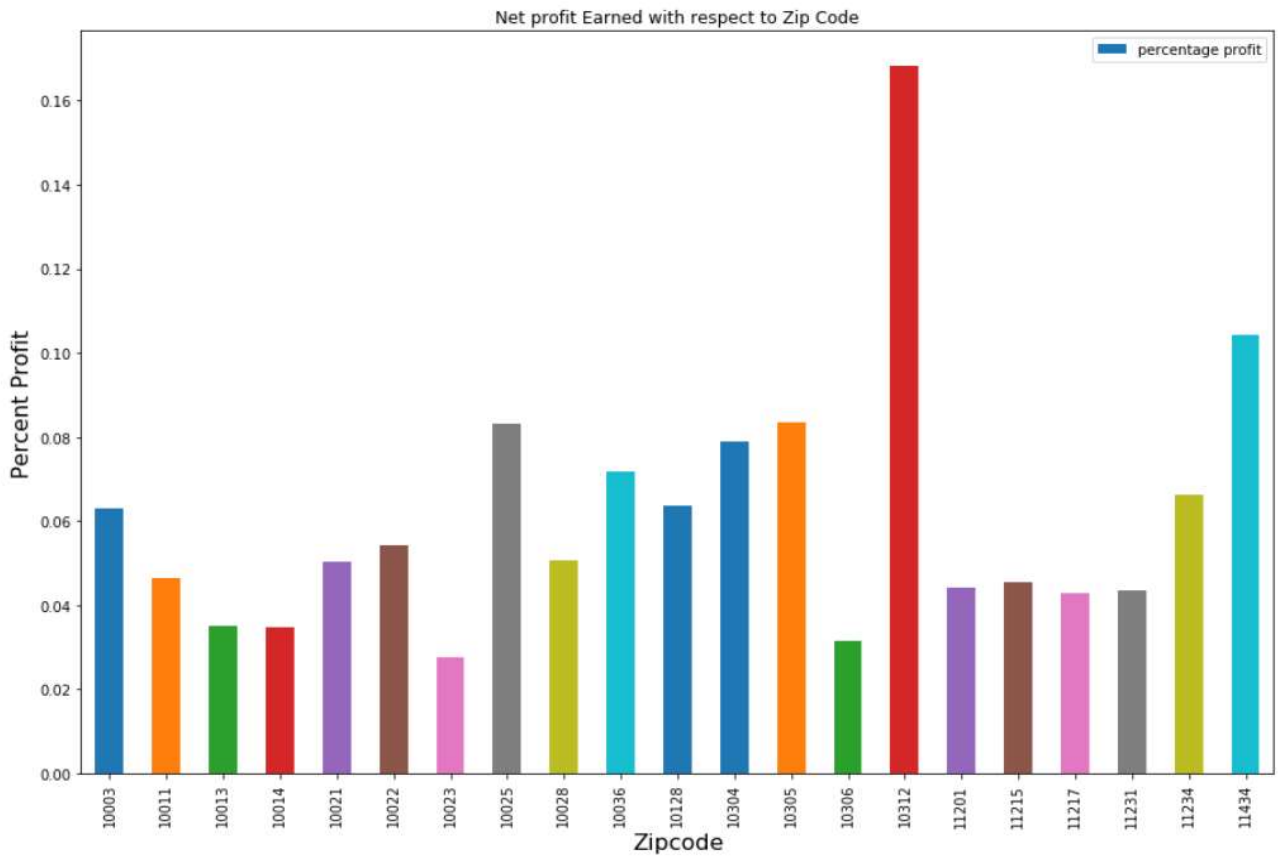
```python
# Calculating the percentage profit earned to the intial investment value on the property.

result["percentage profit"] = ( result["rent in a year"] + result["hike_in_one_year"] ) / ( result["value"])
result.reset_index()
```

	index	zipcode	year	value	predicted_value_2018	hike_in_one_year	price	rent in a year	percentage profit
0	0	10003	2018	2.016550e+06	2.054475e+06	37925.000000	326.466019	89125.223301	0.063004
1	1	10011	2018	2.373500e+06	2.383250e+06	9750.000000	369.082353	100759.482353	0.046560
2	2	10013	2018	3.224517e+06	3.230550e+06	6033.333333	393.693548	107478.338710	0.035203
3	3	10014	2018	2.473150e+06	2.468100e+06	-5050.000000	332.505747	90774.068966	0.034662
4	4	10021	2018	1.717183e+06	1.722920e+06	5736.666667	296.058824	80824.058824	0.050409
5	5	10022	2018	1.884400e+06	1.884400e+06	0.000000	375.228571	102437.400000	0.054361
6	6	10023	2018	2.013717e+06	1.988000e+06	-25716.666667	296.620690	80977.448276	0.027442
7	7	10025	2018	1.358600e+06	1.391550e+06	32950.000000	293.140187	80027.271028	0.083157
8	8	10028	2018	1.909750e+06	1.931480e+06	21730.000000	275.322581	75163.064516	0.050736
9	9	10036	2018	1.729167e+06	1.731375e+06	2208.333333	445.666667	121667.000000	0.071639
10	10	10128	2018	1.648883e+06	1.688467e+06	39583.333333	238.836735	65202.428571	0.063550
11	11	10304	2018	3.124167e+05	3.106000e+05	-1816.666667	97.000000	26481.000000	0.078947
12	12	10305	2018	4.070333e+05	4.078000e+05	766.666667	121.625000	33203.625000	0.083458
13	13	10306	2018	3.400500e+05	3.254000e+05	-14650.000000	93.000000	25389.000000	0.031581
14	14	10312	2018	3.455000e+05	3.449200e+05	-580.000000	215.000000	58695.000000	0.168205
15	15	11201	2018	1.402783e+06	1.407100e+06	4316.666667	211.789474	57818.526316	0.044294
16	16	11215	2018	1.050483e+06	1.048633e+06	-1850.000000	181.411765	49525.411765	0.045384
17	17	11217	2018	1.247833e+06	1.244700e+06	-3133.333333	207.052632	56525.368421	0.042788
18	18	11231	2018	1.196583e+06	1.193560e+06	-3023.333333	202.323529	55234.323529	0.043633
19	19	11234	2018	4.719000e+05	4.708600e+05	-1040.000000	118.500000	32350.500000	0.066350
20	20	11434	2018	3.716333e+05	3.681250e+05	-3508.333333	155.000000	42315.000000	0.104422

```python
# Resetting the index for result.
result = result.reset_index().copy()
```

```python
# Visualization done using plot based on Net profit earned with respect to Zipcode.
import numpy as np
NetProfit_df=result.plot(kind='bar',x='zipcode',y='percentage profit',figsize=(15, 10),\
          title= 'Net profit Earned with respect to Zip Code',legend=True)
NetProfit_df.set_xlabel('Zipcode',fontsize=16)
NetProfit_df.set_ylabel('Percent Profit',fontsize=16)
#By the below graph it is clearly proved us that zipcode 10312 followed by 11434 are best to invest as there return is the
#highest.
```

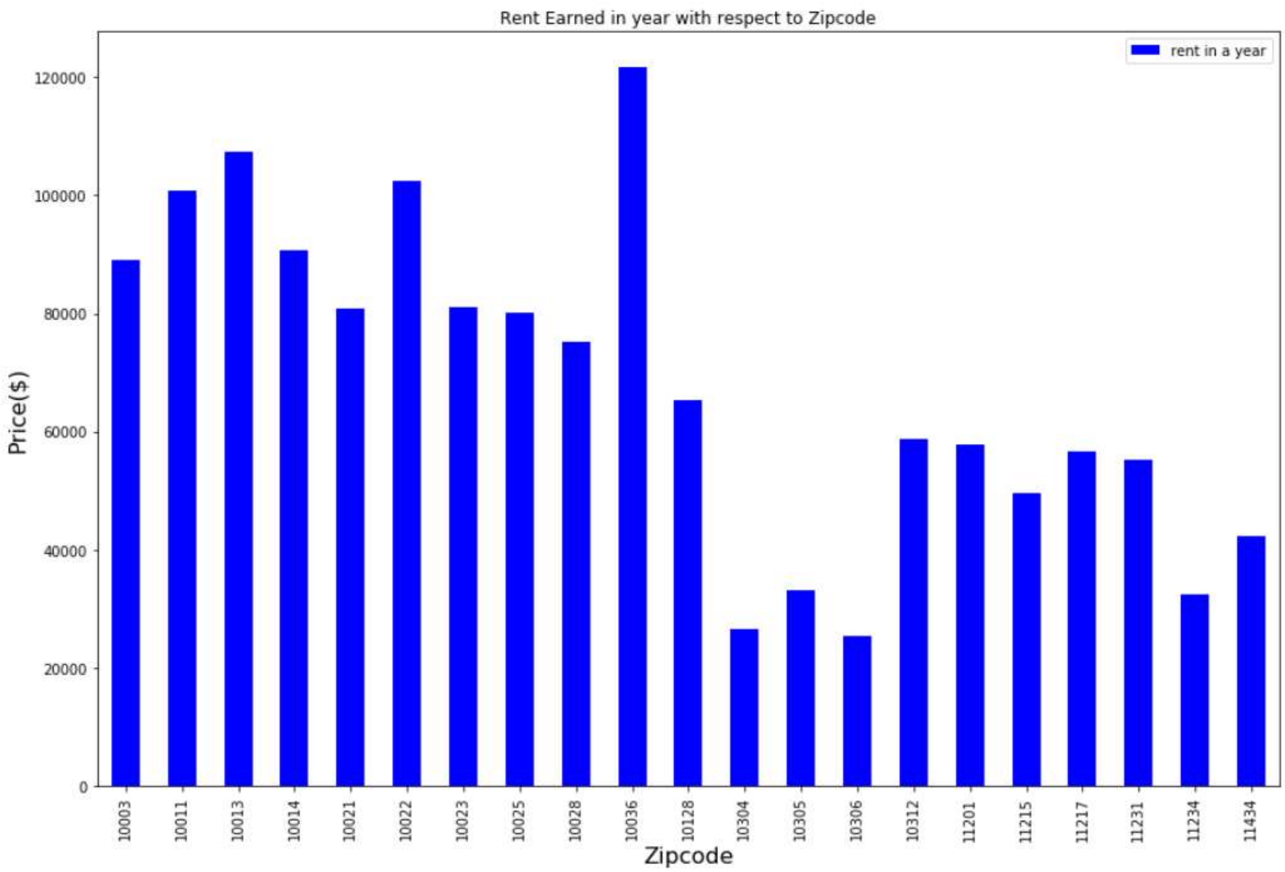
```
Text(0,0.5,'Percent Profit')
```



<u>Thus, it is clearly seen that zip code 10312 is having the highest percentage profit followed by 11434. These zip code are considered best to invest in as its cumulative return is the most.</u>

```python
# Visualization done using plot based on Rent in a year with respect to Zipcode.
Rent_df=result.plot(kind='bar',x='zipcode',y='rent in a year',\
            title='Rent Earned in year with respect to Zipcode',figsize=(15, 10),color='b')
Rent_df.set_xlabel('Zipcode',fontsize=16)
Rent_df.set_ylabel('Price($)',fontsize=16)
# By below graph we can find that which zipcode can produce most rent in a year.
```

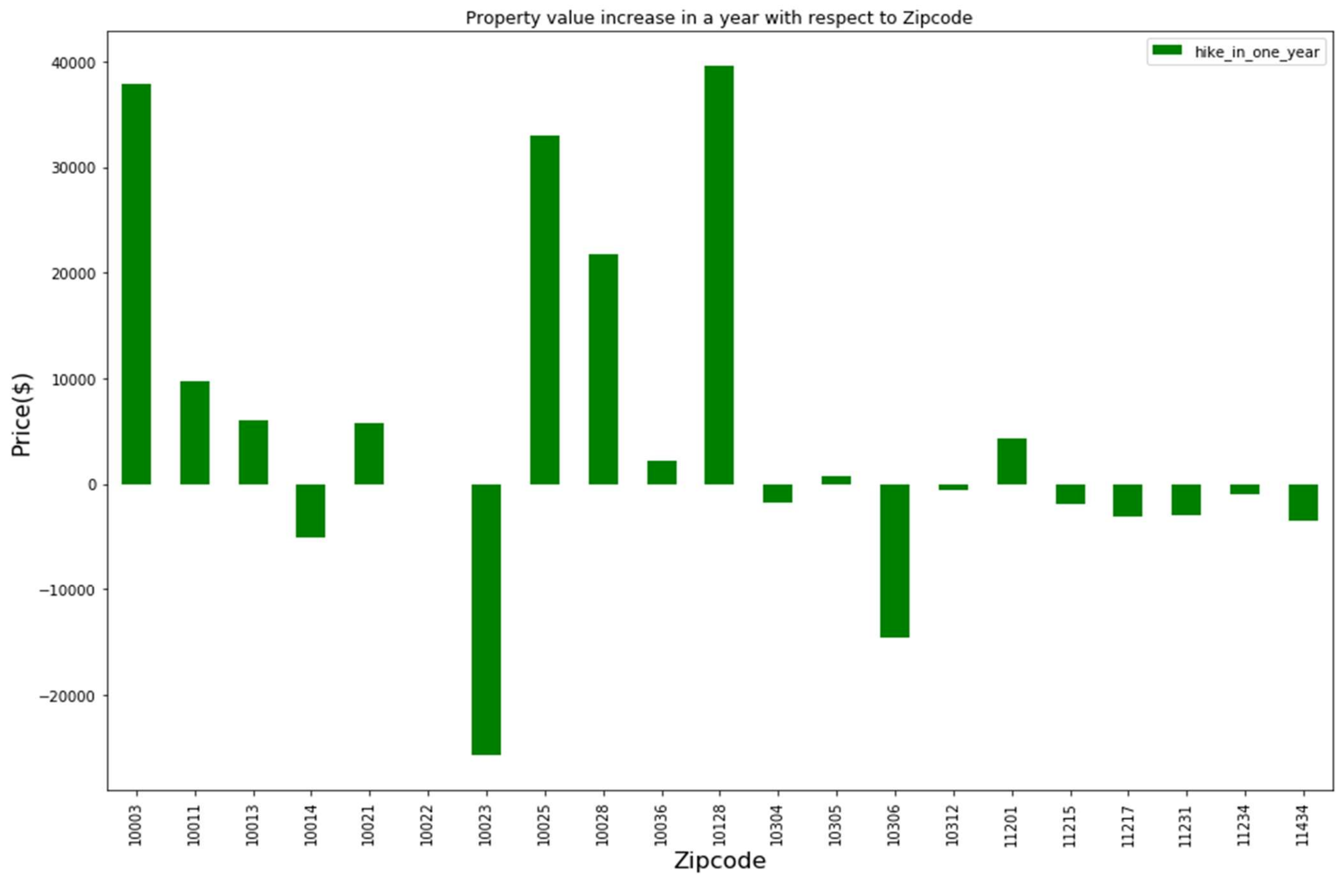
```
Text(0,0.5,'Price($)')
```



<u>**By above graph we can say that 10036 zipcode produces most rent in a year.**</u>

```
# Analysis of Rent hike happen in a year for different zipcodes
ProptyVal_df=result.plot(kind='bar',x='zipcode',y='hike_in_one_year',color='g',\
          figsize=(15, 10),title='Property value increase in a year with respect to Zipcode')
ProptyVal_df.set_xlabel('Zipcode',fontsize=16)
ProptyVal_df.set_ylabel('Price($)',fontsize=16)
# By below graph we can find which zipcode has most hike rent in a year with respect to zipcodes
```

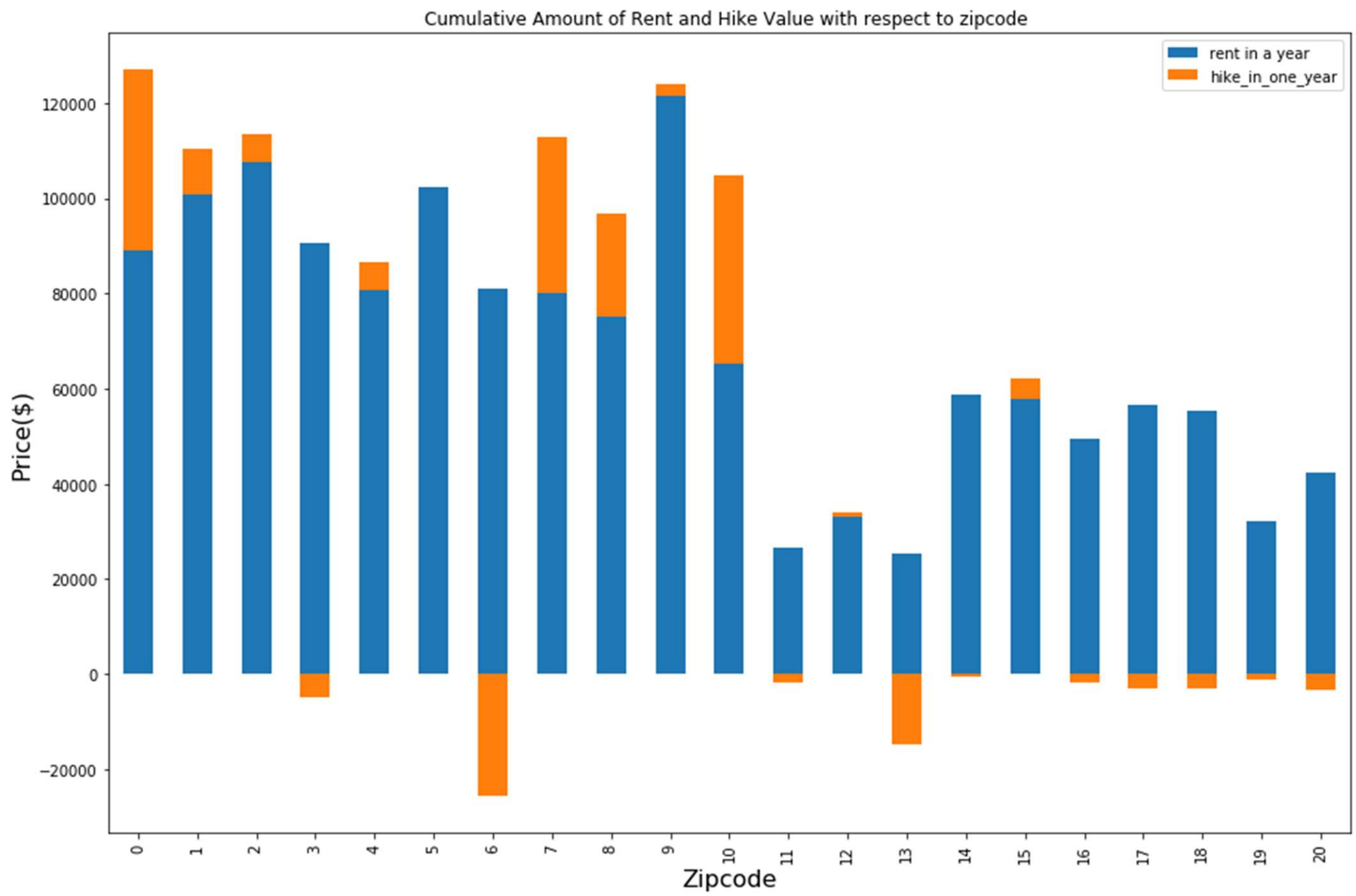
```
Text(0,0.5,'Price($)')
```



<u>With the help of above graph, we will be able to tell which zipcode has most hike rent in a year.</u>

```
# We can analyze the rent in a year and rent hike in a year with respect to zipcode
RentHike_df=result[['zipcode','rent in a year','hike_in_one_year']].plot(kind='bar',stacked=True,\
            figsize=(15, 10),title='Cumulative Amount of Rent and Hike Value with respect to zipcode ')
RentHike_df.set_xlabel('Zipcode',fontsize=16)
RentHike_df.set_ylabel('Price($)',fontsize=16)
```

```
Text(0,0.5,'Price($)')
```



<u>**Above graph helps us in analyzing changes in property value over a year from 2017 to 2018.**</u>

```
# To find the most profitable Zipcodes to invest we will visualize rent changes over a year

ProfitZip_df = result.sort_values('percentage profit',ascending=False)
ProfitZip_df[["zipcode","percentage profit"]]
# By below result it is proved that 10312 is the best profitable zipcode to invest in.
```

	zipcode	percentage profit
14	10312	0.168205
20	11434	0.104422
12	10305	0.083458
7	10025	0.083157
11	10304	0.078947
9	10036	0.071639
19	11234	0.066350
10	10128	0.063550
0	10003	0.063004
5	10022	0.054361
8	10028	0.050736
4	10021	0.050409
1	10011	0.046560
16	11215	0.045384
15	11201	0.044294
18	11231	0.043633
17	11217	0.042788
2	10013	0.035203
3	10014	0.034662
13	10306	0.031581
6	10023	0.027442

Final list of zipcodes in sorted descending order, 10312 zipcode is considered to be most profitable zipcode to invest in.