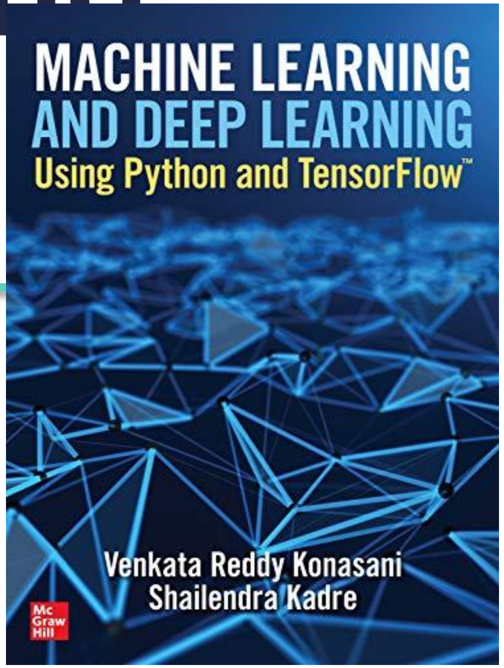
statinfer

Feature Engineering Tips and Tricks

statinfer.com

Chapter 5 in the book



Contents

- Feature Engineering Introduction
- Handling geo location variable
- Handling Date Time variables
- Handling Exponential Variables
- One-hot encoding

Look at this sales data.

Date	Sales
26-01-2019	15390
27-01-2019	15161
28-01-2019	13639
29-01-2019	13761
30-01-2019	13153
31-01-2019	13525
01-02-2019	9601
02-02-2019	11507
03-02-2019	11528
04-02-2019	9267
05-02-2019	9809
06-02-2019	9081
07-02-2019	9436
08-02-2019	9520
09-02-2019	11051
10-02-2019	11367
11-02-2019	9750

Date	Sales
12-02-2019	9882
13-02-2019	9286
14-02-2019	9123
15-02-2019	9725
16-02-2019	11588
17-02-2019	11620
18-02-2019	9702
19-02-2019	9009
20-02-2019	9601
21-02-2019	9148
22-02-2019	9961
23-02-2019	11061
24-02-2019	11400
25-02-2019	9232
26-02-2019	13061
27-02-2019	13442
28-02-2019	13846

- Do you find any patterns?
- Can the model find the patterns with this data?

Create "Day of the Month" column

Date	Day	Sales
26-01-2019	26	15390
27-01-2019	27	15161
28-01-2019	28	13639
29-01-2019	29	13761
30-01-2019	30	13153
31-01-2019	31	13525
01-02-2019	1	9601
02-02-2019	2	11507
03-02-2019	3	11528
04-02-2019	4	9267
05-02-2019	5	9809
06-02-2019	6	9081
07-02-2019	7	9436
08-02-2019	8	9520
09-02-2019	9	11051
10-02-2019	10	11367
11-02-2019	11	9750

Date	Day	Sales
12-02-2019	12	9882
13-02-2019	13	9286
14-02-2019	14	9123
15-02-2019	15	9725
16-02-2019	16	11588
17-02-2019	17	11620
18-02-2019	18	9702
19-02-2019	19	9009
20-02-2019	20	9601
21-02-2019	21	9148
22-02-2019	22	9961
23-02-2019	23	11061
24-02-2019	24	11400
25-02-2019	25	9232
26-02-2019	26	13061
27-02-2019	27	13442
28-02-2019	28	13846

- Create a new column, Day of month
- Do you see any patterns now?

Create "Day of the Week" column

Date	Day	WeekDay	Sales
26-01-2019	26	Saturday	15390
27-01-2019	27	Sunday	15161
28-01-2019	28	Monday	13639
29-01-2019	29	Tuesday	13761
30-01-2019	30	Wednesday	13153
31-01-2019	31	Thursday	13525
01-02-2019	1	Friday	9601
02-02-2019	2	Saturday	11507
03-02-2019	3	Sunday	11528
04-02-2019	4	Monday	9267
05-02-2019	5	Tuesday	9809
06-02-2019	6	Wednesday	9081
07-02-2019	7	Thursday	9436
08-02-2019	8	Friday	9520
09-02-2019	9	Saturday	11051
10-02-2019	10	Sunday	11367
11-02-2019	11	Monday	9750

Date	Day	WeekDay	Sales
12-02-2019	12	Tuesday	9882
13-02-2019	13	Wednesday	9286
14-02-2019	14	Thursday	9123
15-02-2019	15	Friday	9725
16-02-2019	16	Saturday	11588
17-02-2019	17	Sunday	11620
18-02-2019	18	Monday	9702
19-02-2019	19	Tuesday	9009
20-02-2019	20	Wednesday	9601
21-02-2019	21	Thursday	9148
22-02-2019	22	Friday	9961
23-02-2019	23	Saturday	11061
24-02-2019	24	Sunday	11400
25-02-2019	25	Monday	9232
26-02-2019	26	Tuesday	13061
27-02-2019	27	Wednesday	13442
28-02-2019	28	Thursday	13846

- Create a new column, Day of week
- Do you see any patterns now?

Feature Engineering Introduction

- Sometimes the information is hidden in the predictor columns
- We have to manually extract the hidden patterns and create new columns.
- Machine learning algorithm will be able to easily find the patterns from these new columns and improve their accuracy.
- This process of creating new columns is known as feature engineering.

Variables types to consider for feature engineering

- Date Variable
  - Create weekday, month, date, year, quarter, half-year, special-day indicator.
- Time Variable
  - Create hour, minute, working hour, midnight, peak hour etc.,
- Longitude and Latitude variables
  - Create city, state, country, providence etc.,
- Categorical Variables
  - Perform one hot encoding

Case-Study: House price prediction in King county

- Import King county house price dataset
- https://www.kaggle.com/harlfoxem/housesalesprediction
- Perform basic data exploration tasks
- Consider all the numerical columns and build a regression model.
- Call this first model as Model1 and take a note of R-squared and MAPE.

Data Importing

```
import pandas as pd
house_price_data=pd.read_csv("https://raw.githubus
```

```
house_price_data.info()
```

```
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  int64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
```

Predictor Columns

```
house_price_data.columns.values
```

```
array(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated',
       'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype=object)
```

```
pred_cols=house_price_data.columns.values[3:]
print(pred_cols)
```

```
['bedrooms' 'bathrooms' 'sqft_living' 'sqft_lot' 'floors' 'waterfront'
 'view' 'condition' 'grade' 'sqft_above' 'sqft_basement' 'yr_built'
 'yr_renovated' 'zipcode' 'lat' 'long' 'sqft_living15' 'sqft_lot15']
```

Train and Test data

```
X = house_price_data[pred_cols]
y = house_price_data['price']

from sklearn  import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y ,train_size=0.8,

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(17290, 18)
(17290,)
(4323, 18)
(4323,)
```

Basic Model

```
from  sklearn.linear_model import LinearRegression
model1 = LinearRegression()
model1.fit(X_train, y_train)
```

```
#Rsquared value on train and test data
from sklearn import metrics
y_pred_train=model1.predict(X_train)
print("Train RSquared", metrics.r2_score(y_train,y_pred_train))

y_pred_test=model1.predict(X_test)
print("Test RSquared",metrics.r2_score(y_test,y_pred_test))

import numpy as np
#MAPE
print("MAPE on Train data : ", round(np.mean(np.abs(y_train - y_pred_train)/y_train),2))
print("MAPE on Test data : ", round(np.mean(np.abs(y_test - y_pred_test)/y_test),2))
```

```
Train RSquared 0.7004310823997755
Test RSquared 0.6964362880041284
MAPE on Train data :  0.26
MAPE on Test data :  0.26
```

Handling Date variables

Handling Date variables

- Create weekday, month, date, year, quarter, half-year
- We can create age variable using date. For example, create customer account age using customer account creation date
- We can create indicator variables for special date, year end, festivals
- we can include a season index. For example, if sales are high in winter, then we can have winter_index as a new column.

Code - Handling Date Variables

```
date_vars = ['date', 'yr_built', 'yr_renovated']
house_price_dates=house_price_data[date_vars]
house_price_dates.head()
```

	date	yr_built	yr_renovated
0	20141013T000000	1910	1987
1	20140611T000000	1940	2001
2	20140919T000000	2001	0
3	20140804T000000	2001	0
4	20150413T000000	2009	0

Code - Handling Date Variables

```
house_price_dates['sale_year'] = np.int64([d[0:4] for d in house_price_dates["date"]])
house_price_dates['sale_month'] = np.int64([d[4:6] for d in house_price_dates["date"]])
house_price_dates['day_sold'] = np.int64([d[6:8] for d in house_price_dates["date"]])
house_price_dates['age_of_house'] = house_price_dates['sale_year'] -
 house_price_dates['yr_built’]
house_price_dates['Ind_renovated'] = house_price_dates['yr_renovated']>0
house_price_dates.head()
```

date	yr_built	yr_renovated	sale_year	sale_month	day_sold	age_of_house	Ind_renovated
20141013T000000	1910	1987	2014	10	13	104	True
20140611T000000	1940	2001	2014	6	11	74	True
20140919T000000	2001	0	2014	9	19	13	False
20140804T000000	2001	0	2014	8	4	13	False
20150413T000000	2009	0	2015	4	13	6	False

Code - Model2 with Date columns

```
#Rsquared Calculation on Train data
from sklearn import metrics
y_pred_train=model2.predict(X_train)
print("Train data R-Squared : ", metrics.r2_score(y_train,y_pred_train))

#Rsquared Calculation on test data
y_pred_test=model2.predict(X_test)
print("Test data R-Squared : " , metrics.r2_score(y_test,y_pred_test))
```

```
Train data R-Squared :  0.7031095708216482
Test data R-Squared :  0.6984175384447083
```

```
#MAPE
print("MAPE on Train data : ", round(np.mean(np.abs(y_train - y_pred_train)/y_train),2))
print("MAPE on Test data : ", round(np.mean(np.abs(y_test - y_pred_test)/y_test),2))
```

```
MAPE on Train data :  0.26
MAPE on Test data :  0.26
```

Handling Geo location(Lat-Long) Variables

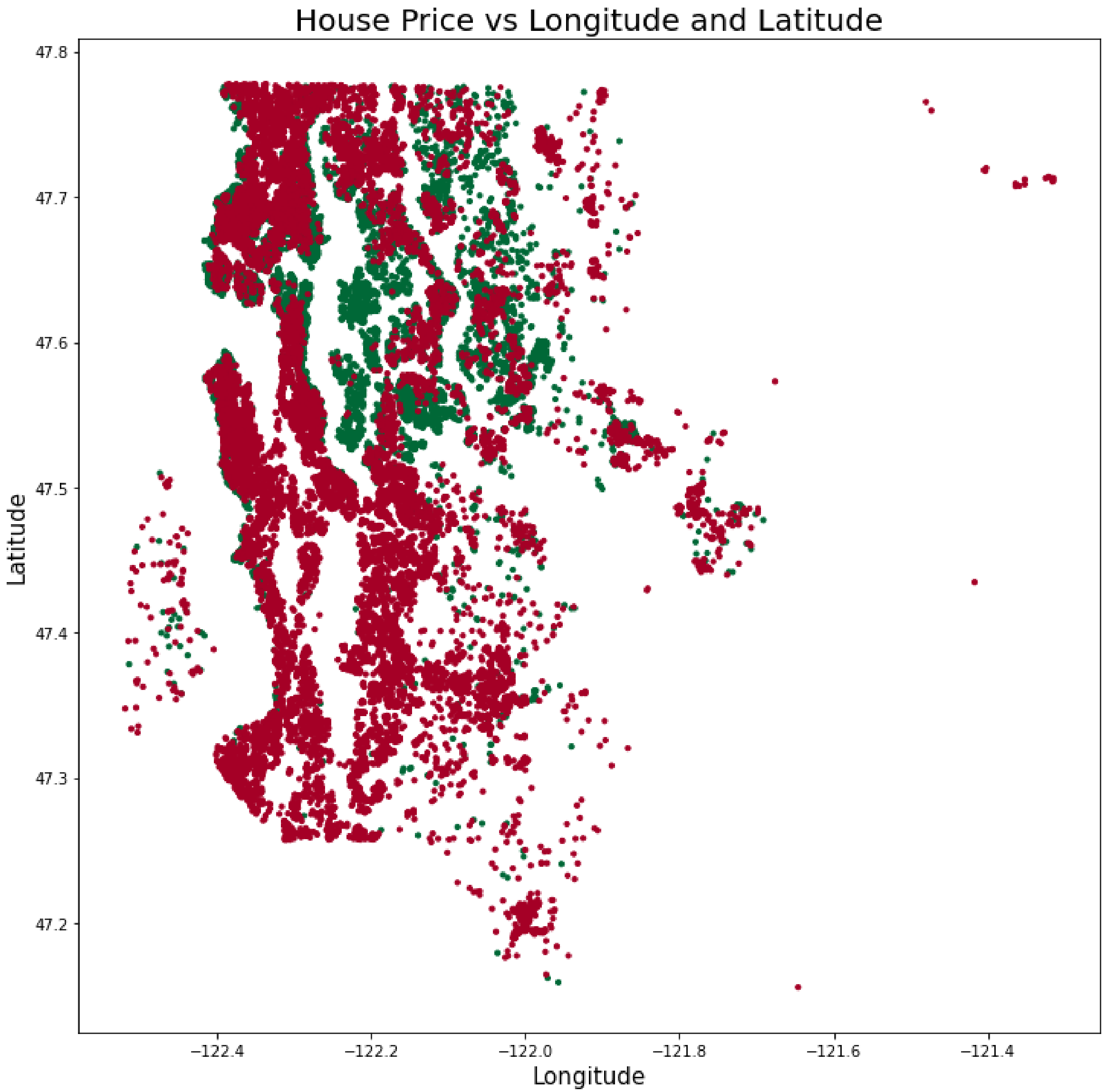
Handling Geo location(Lat-Long) Variables

- Create city, state, country, providence etc.,
- Calculate distances
- Identify special places and create indicators - For example sales is high in a city

Code - Handling Geo location(Lat-Long) Variables

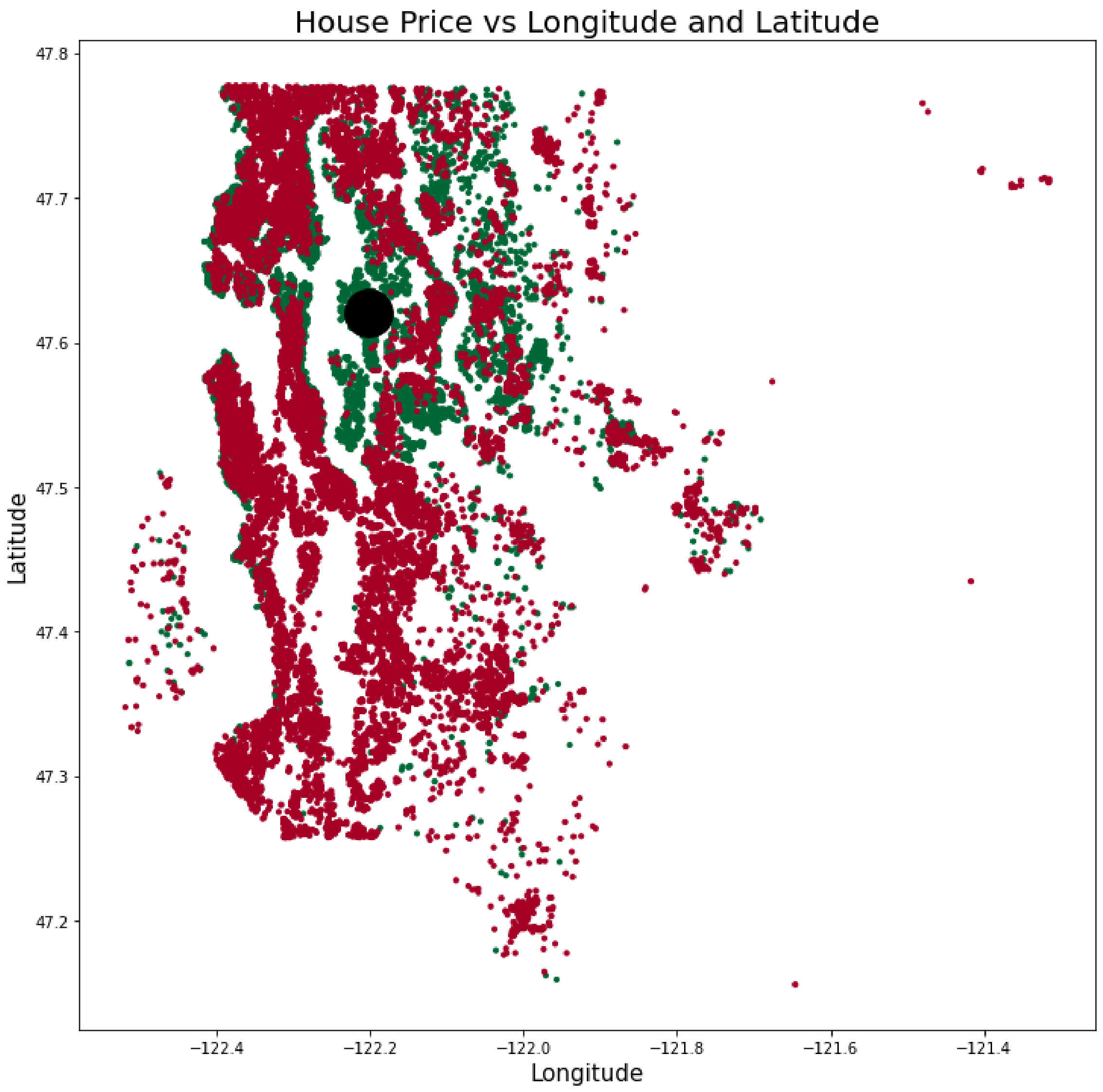
```
###' House Price versus Longitude and Latitude'
bubble_col= house_price_data["price"] > house_price_data["price"].quantile(0.7)

import matplotlib.pyplot as plt
plt.figure(figsize=(12,12))
plt.scatter(house_price_data["long"],house_price_data["lat"], c=bubble_col,cmap="RdYlGn",s=10)
plt.title('House Price vs Longitude and Latitude', fontsize=20)
plt.xlabel('Longitude', fontsize=15)
plt.ylabel('Latitude', fontsize=15)
plt.show()
```

Code - Handling Geo location(Lat-Long) Variables



Code - Handling Geo location(Lat-Long) Variables

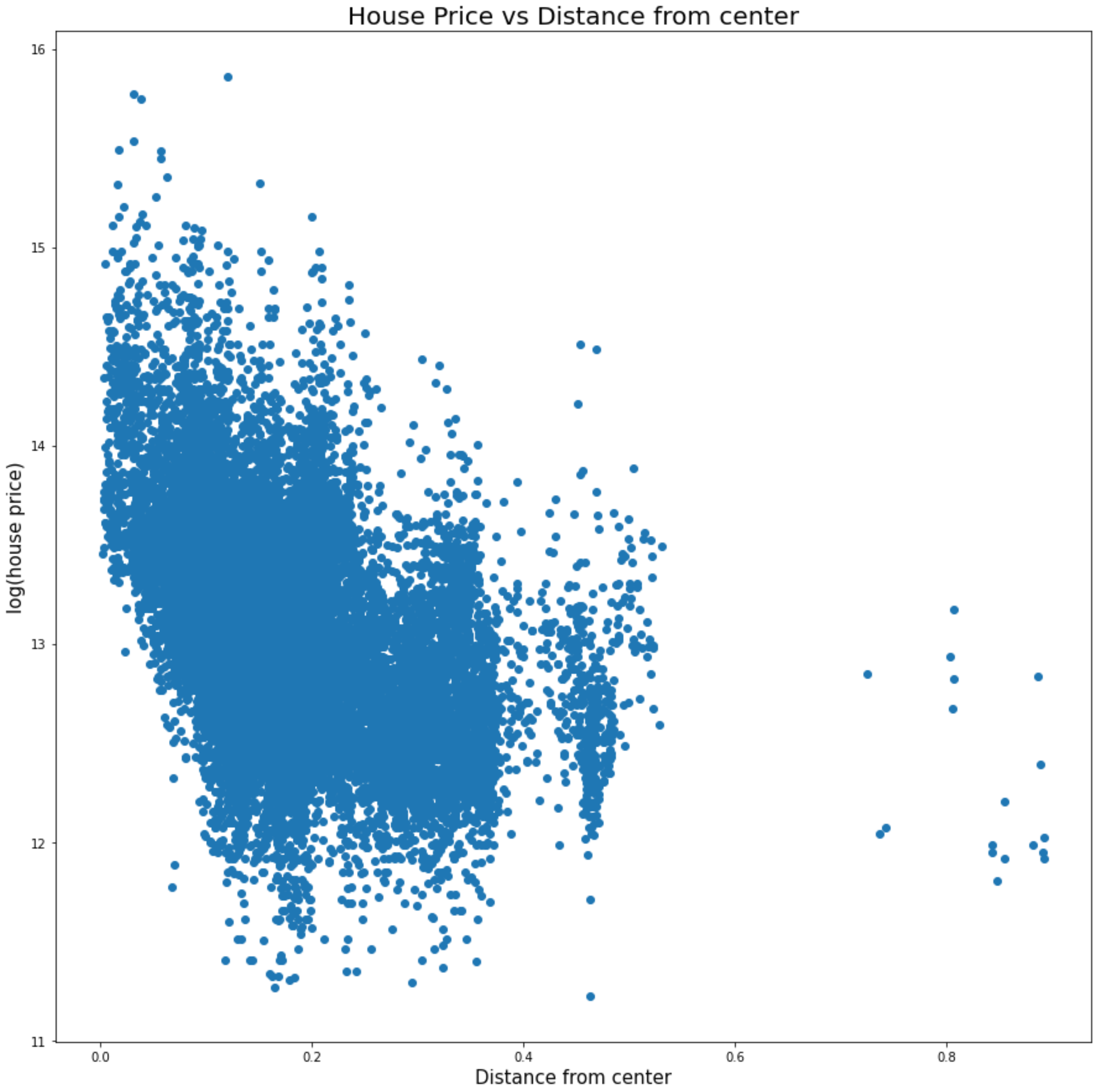
```
high_long_mean=house_price_data["long"][bubble_col].mean()


high_lat_mean=house_price_data["lat"][bubble_col].mean()
```



Code - Handling Geo location(Lat-Long) Variables

```
##Distance from high priced houses center to every house
house_price_data["High_cen_distance"]=np.sqrt((house_price_data["long"] - high_long_mean) ** 2

plt.figure(figsize=(15,15))
plt.scatter(house_price_data["High_cen_distance"],np.log(house_price_data["price"]))
plt.title('House Price vs Distance from center', fontsize=20)
plt.xlabel('Distance from center', fontsize=15)
plt.ylabel('log(house price)', fontsize=15)
```

Code - Handling Geo location(Lat-Long) Variables



Code - Handling Geo location(Lat-Long) Variables

```
#Rsquared Calculation on Train data
from sklearn import metrics
y_pred_train=model3.predict(X_train)
print("Train data R-Squared : ", metrics.r2_score(y_train,y_pred_train))

#Rsquared Calculation on test data
y_pred_test=model3.predict(X_test)
print("Test data R-Squared : " , metrics.r2_score(y_test,y_pred_test))
#MAPE
print("MAPE on Train data : ", round(np.mean(np.abs(y_train - y_pred_train)/y_train),2))
print("MAPE on Test data : ", round(np.mean(np.abs(y_test - y_pred_test)/y_test),2))
```

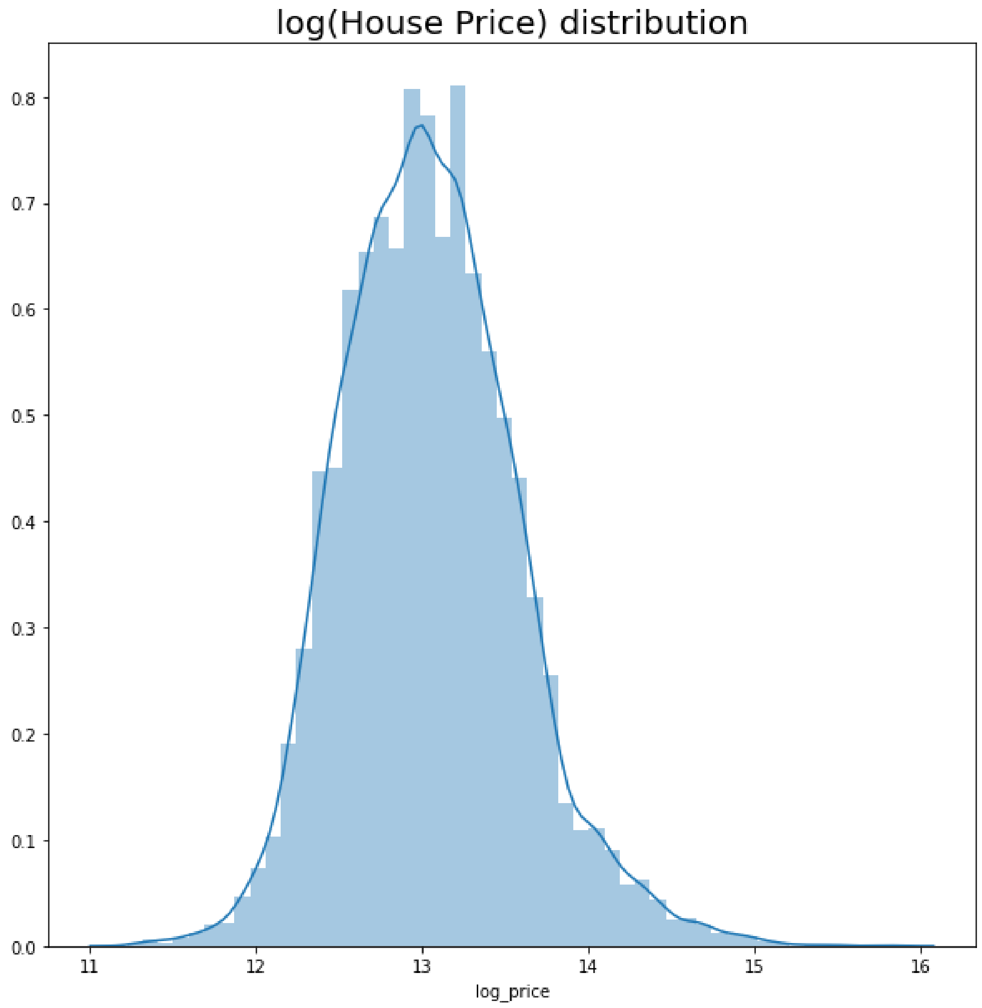
```
Train data R-Squared :  0.7148088489464941
Test data R-Squared :  0.7090610925034878
MAPE on Train data :  0.26
MAPE on Test data :  0.26
```

Transformations

Transformations

- Log transformation
- Square
- Square root
- Inverse transformation
- Some transformations work well when the data has exponential values

Code - Transformations



Code - Transformations

```
X = house_price_data[['bedrooms', 'bathrooms', 'sqft_living',

y = house_price_data['log_price']

from sklearn  import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_

import sklearn
model4 = sklearn.linear_model.LinearRegression()
model4.fit(X_train, y_train)
```

Code - Transformations

```python
#Rsquared Calculation on Train data
from sklearn import metrics
y_pred_train=model4.predict(X_train)
print("Train data R-Squared : ", metrics.r2_score(y_train,y_pred_train))

#Rsquared Calculation on test data
y_pred_test=model4.predict(X_test)
print("Test data R-Squared : " , metrics.r2_score(y_test,y_pred_test))

#MAPE
print("MAPE on Train data : ", round(np.mean(np.abs(y_train - y_pred_train)/y_train),4))
print("MAPE on Test data : ", round(np.mean(np.abs(y_test - y_pred_test)/y_test),4))
```

```
Train data R-Squared :  0.7718064095694626
Test data R-Squared :  0.7644327349746437
MAPE on Train data :  0.015
MAPE on Test data :  0.015
```

One hot encoding

One hot encoding

- Non-numerical categorical variables need to be one-hot encoded
- Some coded numerical variables like country code, zip code, product code also need to be one-hot encoded.

Code - One hot encoding

```python
# get dummy variables
one_hot_data = pd.get_dummies(house_price_data['zipcode'])
#Try all ['view', 'condition', 'grade','zipcode']
print("one_hot_data \n", one_hot_data.sample(10))
```

```
one_hot_data
       98001  98002  98003  98004  98005  ...  98177  98178  98188  98198  98199
21445      0      0      0      0      0  ...      0      0      0      0      0
670        0      0      0      0      0  ...      0      0      0      0      0
9361       0      0      0      0      0  ...      0      0      0      0      0
18079      0      0      0      0      0  ...      0      0      0      0      0
12997      0      0      0      0      0  ...      0      0      0      0      0
9033       0      0      0      0      0  ...      0      0      0      0      0
3472       0      0      0      0      1  ...      0      0      0      0      0
8572       0      0      0      0      0  ...      0      0      0      0      0
11267      0      0      0      0      0  ...      0      0      0      0      0
9814       0      0      0      0      0  ...      0      0      0      0      0
```

Code - One hot encoding

```
# Concatenate dummy columns with main dataframe
house_price_with_dummy = pd.concat([house_price_data, one_hot_data],axis=1)
house_price_with_dummy.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
0	6762700020	20141013T000000	7700000	6	8.00	12050	27600	2.5	
1	9808700762	20140611T000000	7062500	5	4.50	10040	37325	2.0	
2	9208900037	20140919T000000	6885000	6	7.75	9890	31374	2.0	
3	2470100110	20140804T000000	5570000	5	5.75	9200	35069	2.0	
4	8907500070	20150413T000000	5350000	5	5.00	8000	23985	2.0	

5 rows × 94 columns

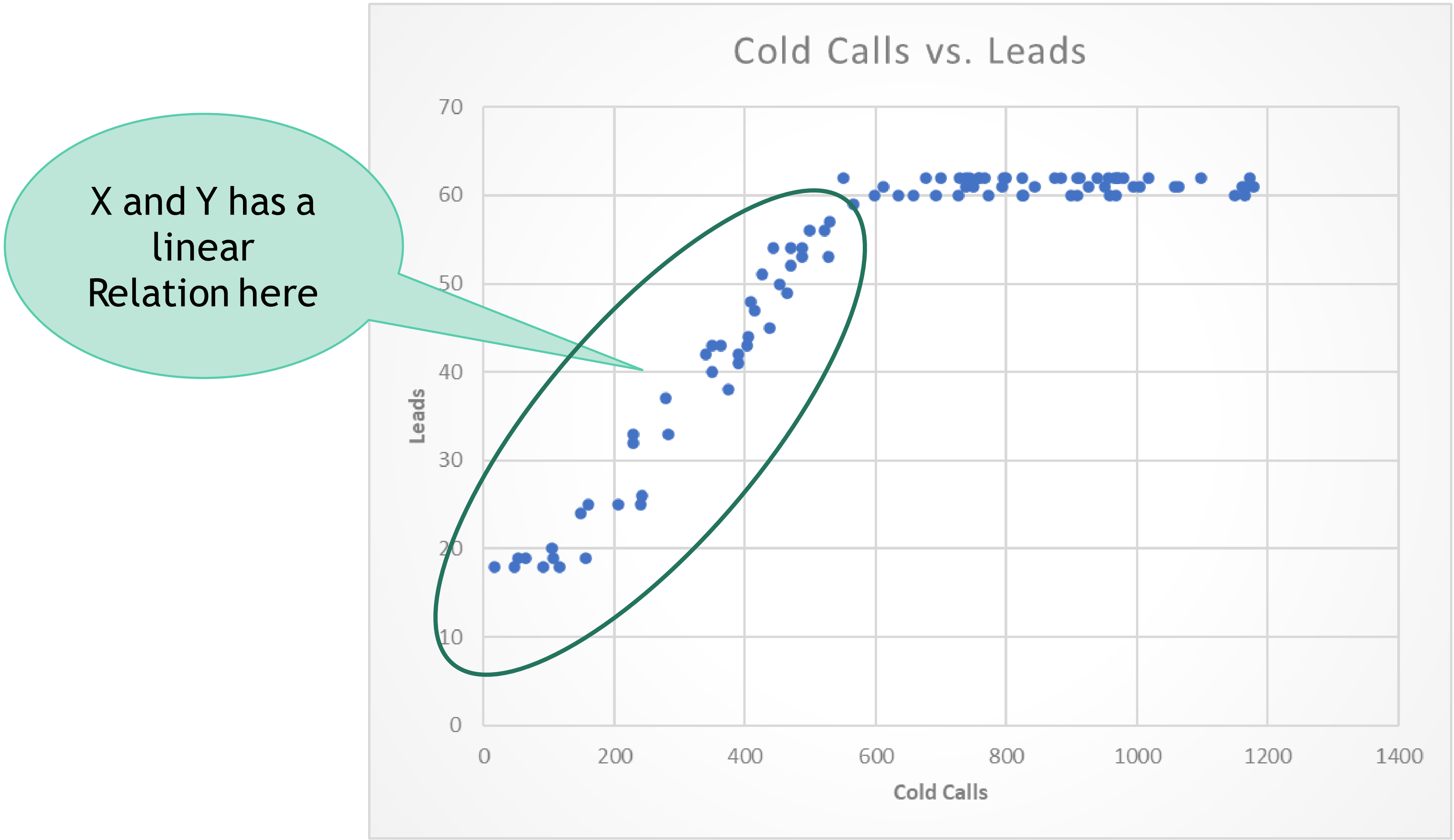
Code - One hot encoding

```
#Rsquared Calculation on Train data
from sklearn import metrics
y_pred_train=model5.predict(X_train)
print("Train data R-Squared : ", metrics.r2_score(y_train,y_pred_train))

#Rsquared Calculation on test data
y_pred_test=model5.predict(X_test)
print("Test data R-Squared : " , metrics.r2_score(y_test,y_pred_test))

#MAPE
print("MAPE on Train data : ", round(np.mean(np.abs(y_train - y_pred_train)/y_train),4))
print("MAPE on Test data : ", round(np.mean(np.abs(y_test - y_pred_test)/y_test),4))
```

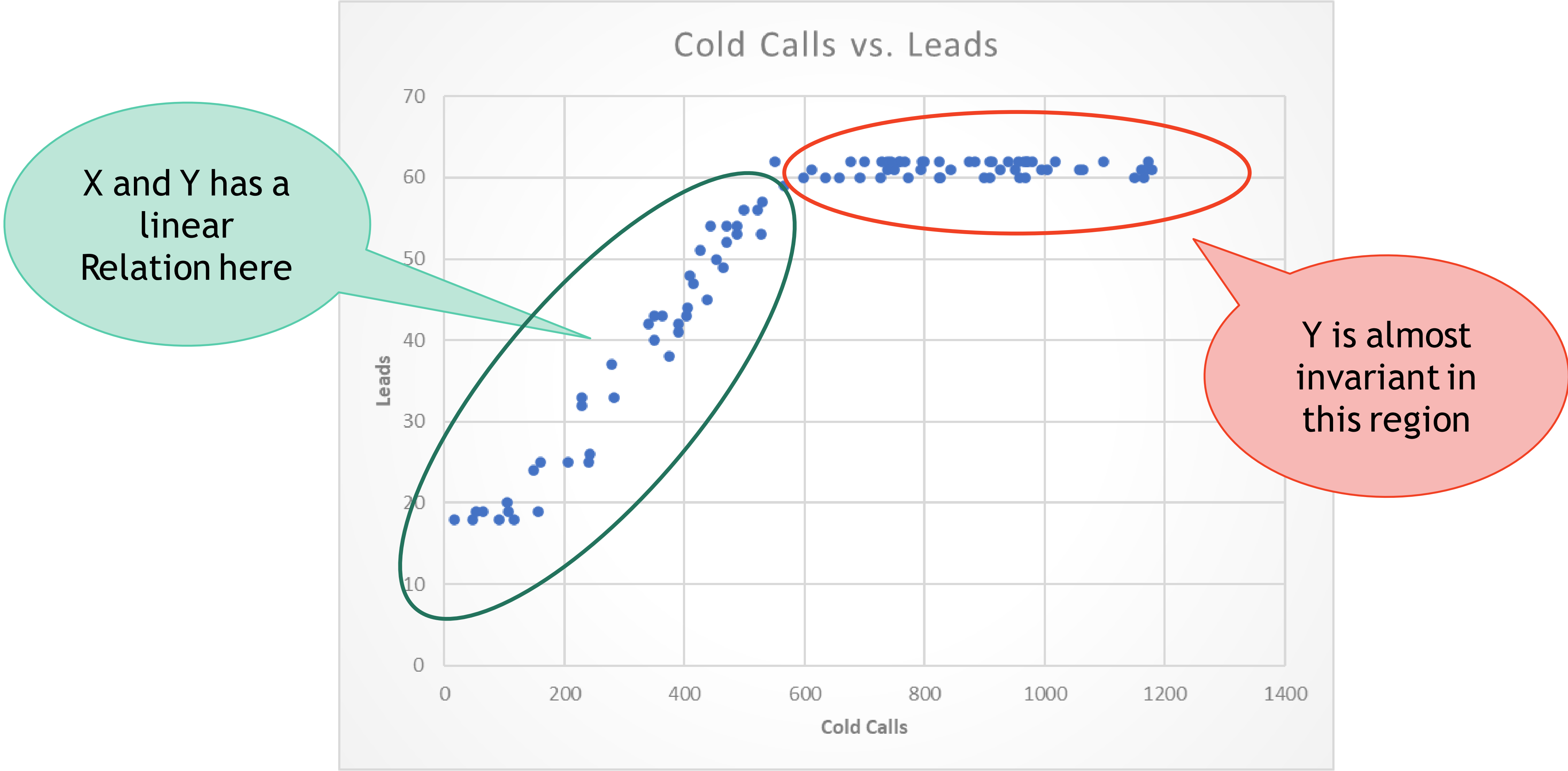
```
Train data R-Squared :  0.809193412413927
Test data R-Squared :  0.8030493346117882
MAPE on Train data :  0.1966
MAPE on Test data :  0.2011
```

Binning

Continuous Variable with Inconsistent relation with Target



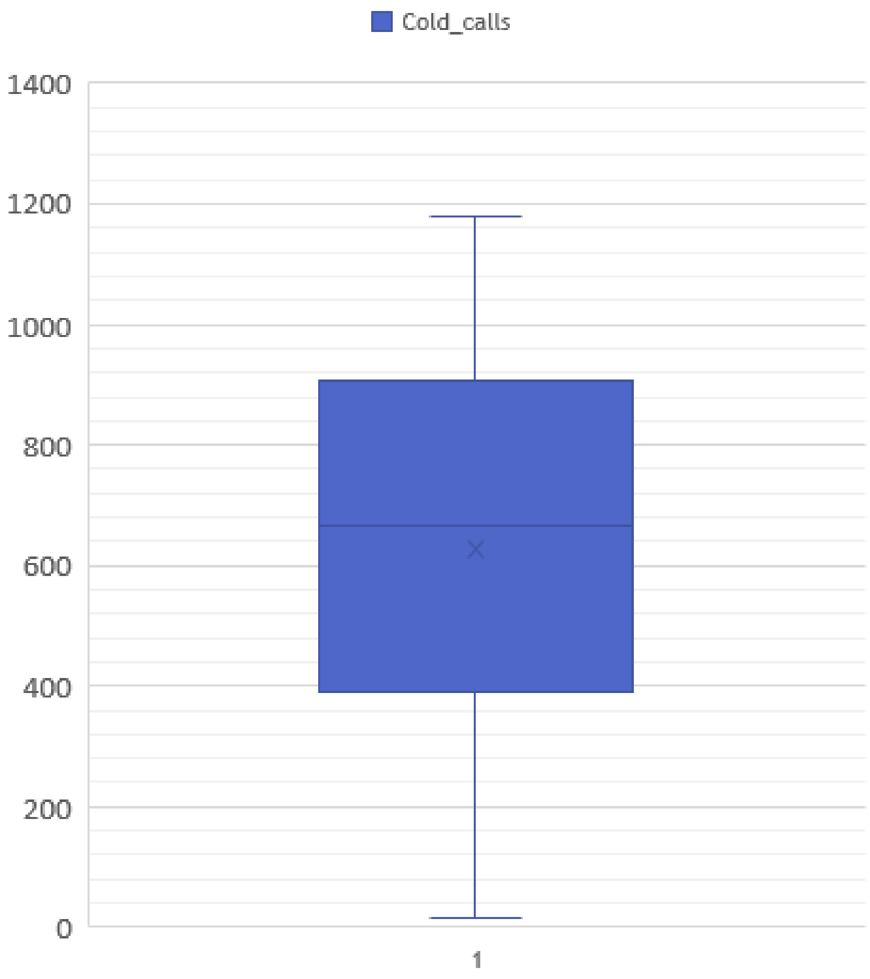
Continuous Variable with Inconsistent relation with Target



Binning or Discretising

- How to extract inconstant patterns ?
- If the patters are observed in isolated intervals we can use binning to capture them
- Divide the feature column to 5 or 10 bins. Perform one-hot encoding.
- By creating bins we can capture all the intermediate isolated patterns.

Binning or Discretising



Bin Range	Data percent
[16-157]	10%
[161-349]	10%
[349-426]	10%
[438-522]	10%
[528-692]	10%
[700-766]	10%
[772-883]	10%
[898-966]	10%
[968-1098]	10%
[1149-1179]	10%

Code : Binning

```
house_price_with_dummy['bins'] = pd.qcut(house_price_with_dummy["sqft_living"], q=10)
house_price_with_dummy['bins'].value_counts(sort=False)
```

```
bins_one_hot = pd.get_dummies(house_price_with_dummy['bins'])
data_with_bins_dummy = pd.concat([house_price_with_dummy, bins_one_hot],axis=1)
bins_cols=list(bins_one_hot.columns)

all_pred_cols=prev_cols+encoded_cols+bins_cols
```

Code : Binning

```
X = data_with_bins_dummy[all_pred_cols]
y = data_with_bins_dummy['price']

from sklearn  import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_spl

import sklearn
model6 = sklearn.linear_model.LinearRegression()
model6.fit(X_train, y_train)
```

Conclusion

- The tips and tricks discussed here in this session may not work in all the scenarios.
- Feature engineering requires creativity and business knowledge.
- We need to study the underline data and business thoroughly to create new features.
- We need to be careful with overfitting while performing feature engineering.

statinfer

Thank you