AIRBNB- MONTREAL PRICE PREDICTION ANALYSIS

Ruta patel (20755706), Nitheesha Reddy Penta Reddy (20811504)

University of Waterloo

Abstract— Airbnb is a multinational online market company hosting affordable short term rentals in various countries**.** It is an online platform for hosts to offer their homes to guests for short term lodging [1]. Guests can make bookings according to their preferences and budget. Airbnb hosts over 6M listings, in 100K cities over 191+ countries [2]. Since, there are various features to choose from while booking a listing, predicting the price from them is an interesting data analysis task.
In this report, we focus on data from Airbnb Montreal, Canada, where we predict the price of a listing based on 10 feature variables. We have performed 5 supervised learning algorithms: Linear regression, Random forest, KNN, Decision tree and Support vector regressor (SVR). Prior to that exploratory data analysis (EDA), data cleaning and splitting the data into training and testing sets has been done. All the models are evaluated and results are discussed.

Key words: Data modeling, EDA, linear regression, random forest, decision tree, SVR, KNN, Cross validation

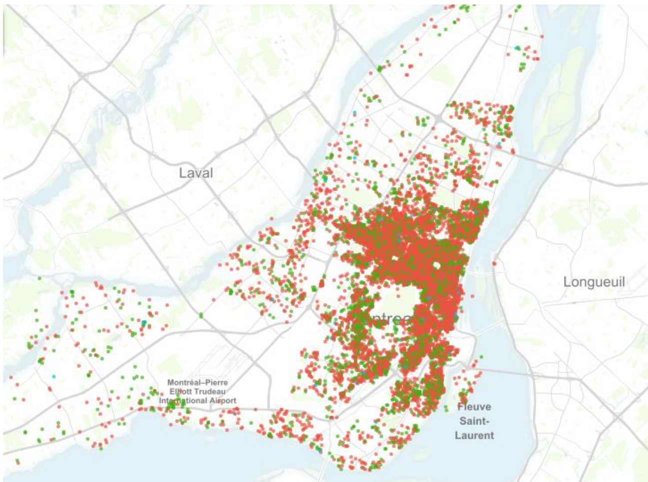
I. INTRODUCTION

Airbnb is a continuously growing company with thousands of hosting and millions of customers. This is a convenient source of income for hosts who have empty houses and for guests looking for affordable stay in otherwise expensive cities such as New York or Toronto. Millions of options to choose from with a wide range of services, sometimes it is difficult to estimate the optimal price for a given listing. As the users continue to increase from both the demand and supply end, it is essential that new hosts price their property reasonably. Therefore, it is important to understand the price dynamics based on various factors for the both the hosts and guests.
The price range varies for each listing depending upon its location, amenities it offer, the housing style, the décor and luxury of the house, seasonality and many more. Taking into consideration all the features, sometimes it is difficult to decide on a price for a particular house. Also, as the listings increases day by day, one does not want to overcharge or undercharge in particular.
Hence, it is difficult to answer the question, how much a host should demand and how much a guest should pay?

In this project we seek to analyze the same. We are using the Airbnb Montreal data, which is a city in Canada and a popular tourist attraction. The dataset size is 21,104 with 106 predictive attributes out of which 2 are non – predictive and 1 target variable price.
This document comprises of concise process of the work done following with brief analysis of the results and conclusions.

A quick glance of the dataset:

1. Currently, there are 21,104 listings present ranging from $10 to $12,960 per night.
2. There are 14,332 unique hosts, distributed in 32 neighborhoods across the city.
3. There are 106 features to predict the price variable from.



[3]

Flow chart of the work plan:

1. Selecting 10 features out of 104.
2. Visually analyzing the data through EDA.
3. Looking for missing data and replacing/eliminating them.
4. Log transformation of target variable price to eliminate the skewness and get a normalized distribution.
5. Finding outliers and eliminating them.
6. Finding statistical relationship between features and target variable price.
7. Generating correlation matrix for exploratory variables and price to come up with a hypotheses.
8. Encoding of categorical variable(neighborhood, room type and property type)
9. Splitting data into training and testing sets.
10. Building regression models.
11. Evaluating the models using metrics.
12. Cross validation

Flow chart of the report:

I. Introduction
II. Literature review
III. About the dataset

IV. Exploratory data analysis
V. Data cleaning and preprocessing
VI. Evaluation metrics
VII. Training and Testing data
VIII. Building models
IX. Results
X. Conclusion

II. LITERATURE REVIEW

There have been few Airbnb price predictions done for various cities such as New York, Boston, Toronto and more[4, 5,6,7,8,9]. However, we could not find a single project done on Montreal data before. Therefore, we do not know what to expect from our data. This is an advantage for us as we don't have any already done work or analysis; our predictions will be fair and unbiased in terms of what others have done. As our data, has been updated recently, we will be analyzing all the recently added listings and features too.
Most of the analysis projects have done predictions using Linear regression, Lasso and Ridge regression[4,8,9] but in our analysis we are adding Decision tree, Random forest and SVR algorithms instead of Lasso and Ridge to improve upon the results.
Additionally, it was found that almost all the analysis includes 70% of the feature variables.[4,5,7,8,9] In our analysis, we focus on only those we feel are much more influential.

III. ABOUT THE DATASET

The dataset for this project was collected from the Inside Airbnb website [4] which is the open source non commercial set of tools and data for public viewing and research. The data available on this website is scrapped from information available from the Airbnb websites of multiple countries. As mentioned, we are using the Montreal data which has been recently updated in July, 2019. Hence, our dataset is very fresh. The dataset is multivariate with 104 features, out of which 24 are of type float, 21 of type integer and 61 of type object.

It is not logical to take all the variables into consideration and many of them are not even of predictive nature (scrape_id, summary etc). Therefore, we selected the below mentioned features for our analysis as our first estimation was that these variables will correlate and affect the price maximum.

1. **Id**: identity of each listing (numeric)(non predictive)
2. **neighbourhood_cleansed** : names of neighbouthood (categorical)
3. **property_type:** type of property (Apartment,condo..etc)(categorical)
4. **room_type:** type of room (private room, shared etc)(categorical)
5. **accommodates:** number of people that can accommodate in a house (numeric)
6. **bathrooms:** number of bathrooms (numeric)
7. **bedrooms:** number of bedrooms (numeric)
8. **beds:** number of beds (numeric)
9. **availability_365:** availability of the rental (numeric)
10. **number_of_reviews:** total count of the number of reviews for the rental (numeric)
11. **review_scores_rating:** average review score rating of the rental (numeric)
12. **price:** price of a listing per day, target **variable**(numeric)

IV. EXPLORATORY DATA ANALYSIS

For initial understanding of the data, it is very important to perform the EDA for each predictive variable to see the distributions and visually check if there are some outliers present or not. The price variable was of type 'string' due to the '$' sign. In order to include it in the EDA and other analysis, we converted it into type integer.
We generated histogram plots for each variable to see their distribution.
"Due to large size of graph generated, it is not posted here."
Observations:

1) None of the feature variables have a normal distribution. Almost all the features are skewed towards the left.
2) Most of the properties can accommodate 2, 3, 4 and 6 people.
3) Most of the guests gives higher ratings.
4) There is not much variation in price for different neighbourhoods.

One important observation we made is that the target variable price was heavily skewed towards the right, the obvious reason being that most of the listings are relatively priced lower with few increasing towards the right.

V. DATA CLEANING AND PREPROCESSING

The dataset comprises of 21,104 instances and in order to build a model that fits the data more accurately we did the following data cleaning:

1) Checking if there are missing or Null values and eliminating them.

```
dataset.isnull().sum()
```

```
id                            0
neighbourhood_cleansed        0
property_type                 0
room_type                     0
accommodates                  0
bathrooms                    30
bedrooms                      4
beds                         27
price                         2
availability_365              0
number_of_reviews             0
review_scores_rating       4725
dtype: int64
```

As seen, there are some missing values in our data. Since, there are only small values present in 'bathrooms' ,'bedrooms', 'beds' and 'price' we dropped these values. For the 'review score rating', we replaced the missing values with the mean.

2) Normalizing the target variable

We performed log transformation on our 'price' variable to get a normalized distribution.

Fig 1: Histogram of pricing before performing normalization

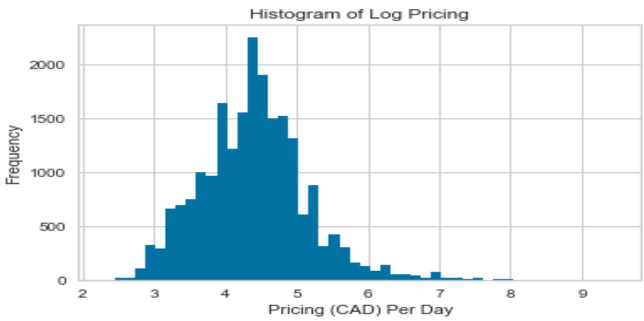


Fig 2: Histogram of pricing after performing the normalization

3) Fixing outliers

After performing EDA, we do not find any outliers in our data, however in the 'price' out of 21,104 instances we have just one listing priced at $12,960 per night, which might be true assuming that the property is exceptionally luxurious or have valuable possession such as art work or artifacts.

The outliers are eliminated using the Z-Score. It is used to find the correlation between the mean and standard deviation of the given data points. In our project, we used the formula: sum (mean) x 2(standard deviation) as our threshold value. Any data point that is outside this threshold value is removed.

After removing the outliers, the size of our data is 20501.

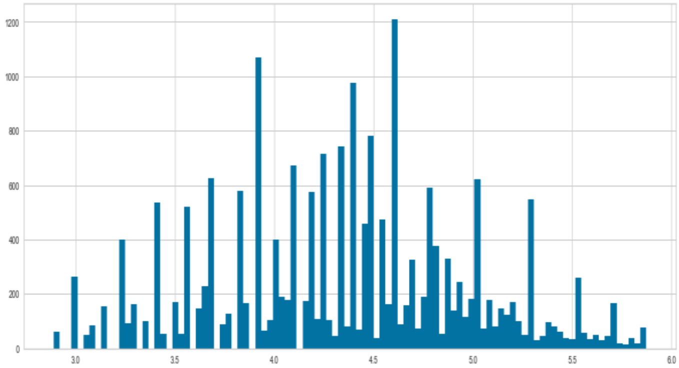


Fig 3: Price log graph before removing outliers

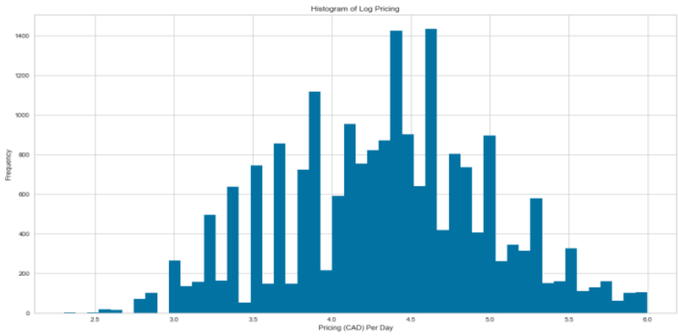


Fig 4: Price log graph after removing outliers

4) Encoding of categorical variables

From the features we selected for analysis, 3 variables are of type categorical. Since, we need numeric data for regression we encoded them into numeric using dummy variables.

Pandas dummy variable works in a way that for each category, a new column is added into that dataset with 0's and 1's. 1's indicating that particular category is present in a row whereas it will be 0 for the rest.

After doing the encoding our dataset set columns changed from12 to 77.

5) Correlation

This is very important step through which we gather necessary information to form the most suitable and accurate model. To apply a regression model, it is necessary to check which predictor variable has the most influence on the outcome variable. Hence, to get accurate results, we will check the correlation between each of the predictor variables and the outcome variable to see which one affects the price the most. This is done using box plots and correlation tests.

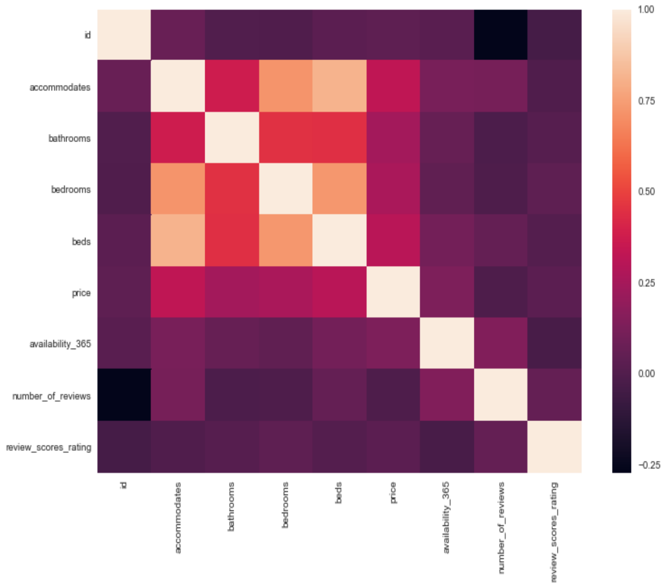


Fig 5: Correlation Matrix for numeric variables

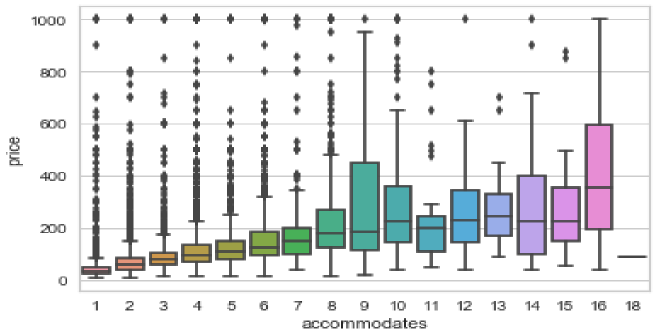


Fig 6: Box-plot of price Vs accommodates

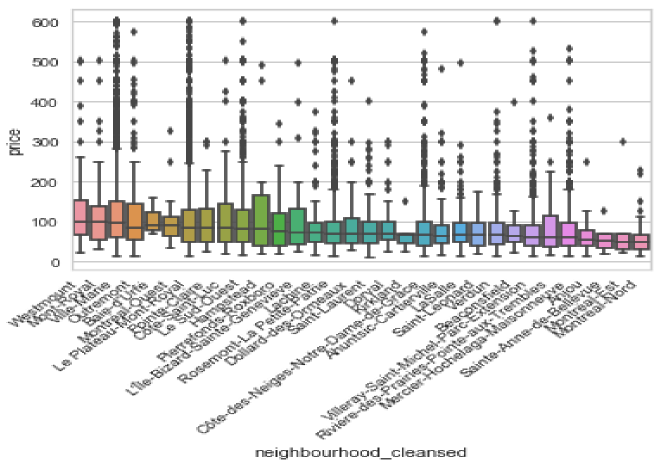


Fig 7: Box-plot of price Vs neighbourhood_cleansed

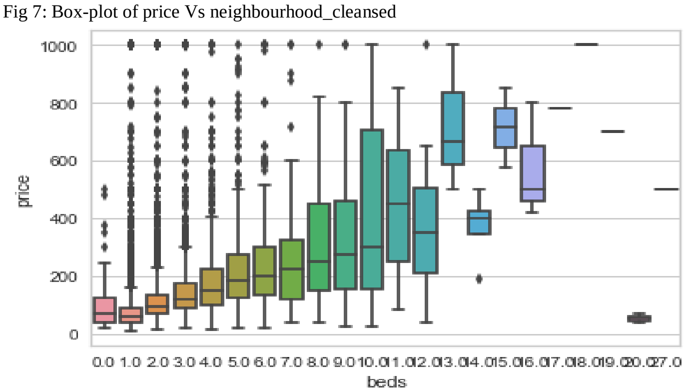


Fig 8: Box-plot of price Vs beds

In order to form a hypothesis for our project, we make predictions based on the plots and correlation values generated against the feature variables and target variables.
It can be seen from the box plots, that the price increases with accommodates and beds variables. Also, after plotting correlation matrix for all the variables after doing encoding, the room_type(entire home/apt) correlates the maximum with price
Hence, hypothesis can be generated as:
H1: The 'room_type' (entire apartment), 'accommodates' and 'beds' variables contribute the price of a listing in a very significant way.

VI. EVALUATION METRICS

The evaluation of all the models which we perform is based on the parameters Root Mean Square Error(RMSE), Mean Absolute Error(MAE), Mean Square Error(MSE), Variance.

a) Root Mean Square Error: It is used to measure the differences between the values predicted by a model and the and the observed values. It is calculated using the formula

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

b) Mean Absolute Error: It is the average of the absolute difference between the measured value and the true value. It is calculated using the formula:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |x_i - x|$$

c) Mean Square Error: It is the average squared difference between the observed values and the values predicted by the model. It is calculate using the formula:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

The lesser the Mean square error the better is the performance of the model.

d) Coefficient of Determination, R2: It tells us how well a model has fitted the data. It determines how close the predicted data is to the regression line. The greater the value of R2, the accurate a model has performed. Our aim is to get this value as high as possible.

VII. TRAINING AND TESTING THE DATA

We split our dataset in a [70: 30] ratio. This means that the entire dataset is divided into 2 parts where 70% of the data is assigned to the training set and 30% to the testing set. The

models are built around this divided set, and are later evaluated according to their performance.
Furthermore, we performed cross validation to test the accuracy and decide which model is suited best for our data.
Cross validation is done by splitting the data into 10 folds and testing the accuracy of it. With each iteration, the model is tested on k-1 folds and tested on the last remaining fold.

Our training and testing dataset size is as follows:

Print (X_train.shape) = (14350, 74)
Print (y_train.shape) = (14350,)
Print (X_test.shape) = (6151, 74)
Print (y_test.shape) = (6151,)

VIII. BUILDING MODEL

ANALYSIS 1: It is done for testing the hypothesis.
Linear regression and Random forest algorithms have used by taking into consideration only the 4 above mentioned variables. The results are mentioned in the section IX and conclusions in section X.

ANALYSIS 2: It is done taking into account all the 10 feature variables.
We have implemented the following 5 algorithms in order to achieve the best results.
For building the models, we have used **"yellowbrick regression visualizer"**, which is an open source python project to extend the **scikit-learn API.** We have used it for better visual analysis and model selection. We have also implemented the algorithms using **Sklearn** library for being confident about the results produced by the visualizer. Evaluation metrics results are also generated using sklearn. metrics.

PREDICTION ERROR PLOT: As name suggests, these plots shows the actual target values against the predicted values generated by the model. We diagnose our model by comparing the 45 degree line with the generated regression line.

RESIDUAL PLOTS: it shows the error against the predicted values. The green data points are testing set and blue for training set.

1. The **horizontal line indicates zero error**. If most of the values are **below the horizontal line** (actual – predicated) than it indicates that the **model predicted higher values than the actual values** and vice versa for positive error values.
2. A model linear regression **model is appropriate** for the data if the **residuals are randomly distributed** around the middle line. Also, the **histogram** should be **normally distributed around zero.**

Head (5) of the actual and predicted outputs is also included.

1) LINEAR REGRESSION

It is the most popular algorithm in supervised learning. It performs regression to find the relation between the variables and predicting the target. It predicts a variable (y) based on single or multiple independent variables(x)
We performed linear regression and the generated the following results.

```
MAE: 0.3471407114152832
MSE: 0.20201163734592703
RMSE: 0.4494570472758515
Coefficient of Determination: 0.5376198234970
468
```

```
Actual  Predicted
0    5.855072    4.913738
1    4.553877    4.653816
2    4.094345    3.907751
3    3.401197    3.740622
4    4.276666    4.380818
```

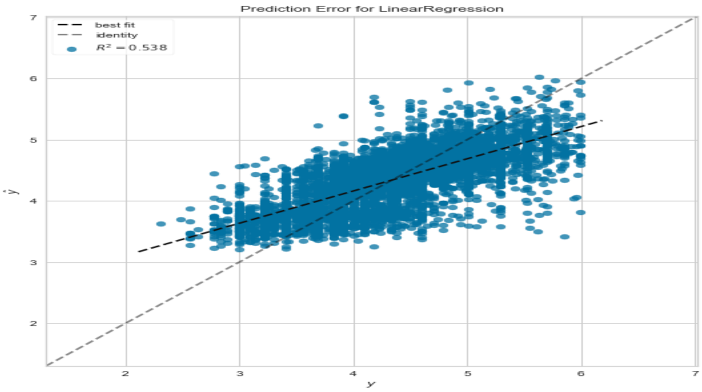


Fig 9: Prediction Error for Linear Regression

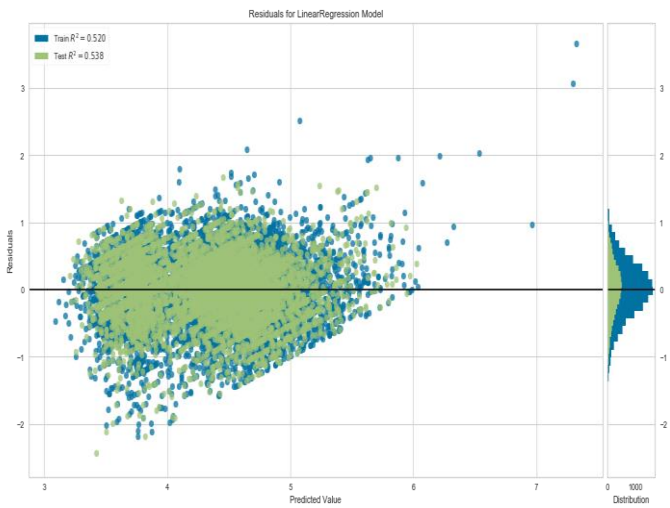


Fig 10: Residuals for Linear Regression Model

2) SUPPORT VECTOR MACHINE – REGRESSION

SVR is different from simple regression in a sense that , in simple regression we minimize the error rate whereas in SVR we try to fit it within a particular threshold. The model produced by SVR depends on the training data subset by ignoring all the data close to the model prediction. Basically, the idea is to maximize the margin and minimize the error.

The results produced are mentioned below, with predicted and actual output.

Mean Absolute Error: 0.38604173169912465
Mean Squared Error: 0.2443872659829381
Root Mean Squared Error: 0.49435540452486015
Coefficient of Determination: 0.44062714076831666

	Actual	Predicted
0	5.855072	4.870401
1	4.553877	4.173498
2	4.094345	3.747557
3	3.401197	3.755990
4	4.276666	4.282114

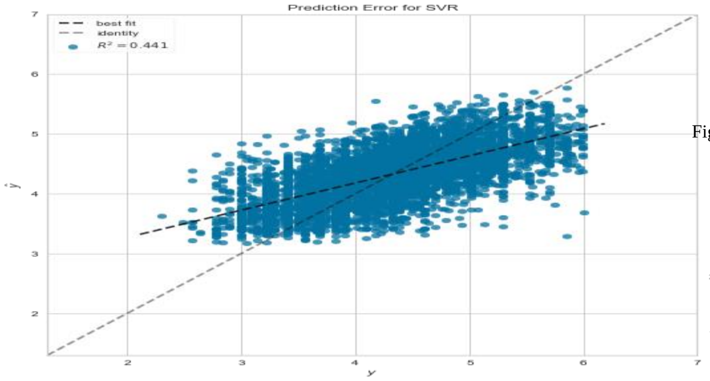


Fig 11: Prediction Error for SVR

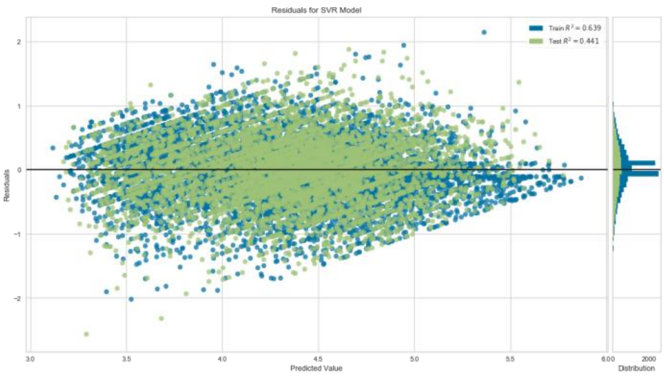


Fig 12: Residuals for SVR

3) DECISION TREE REGRESSION

Here, regression model is built in the form of a tree. Decision trees can handle both numerical and categorical data. The predicted response is calculated by the mean response of the training observations belonging to the same node. At the end, a tree is constructed having all the decision and leaf nodes which represents the decision for the target variable.

Mean Absolute Error: 0.4421603620569738
Mean Squared Error: 0.3402545361699368
Root Mean Squared Error: 0.5833134116150055
Coefficient of Determination: 0.2211985677796503

	Actual	Predicted
0	5.855072	5.796058
1	4.553877	4.418841
2	4.094345	3.448852
3	3.401197	4.077537
4	4.276666	4.325088

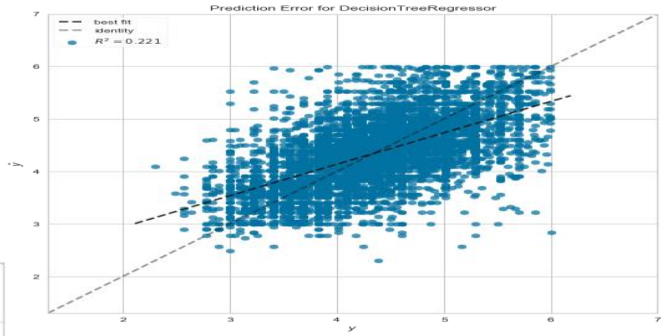


Fig 13: Prediction Error for Decision Tree Regression

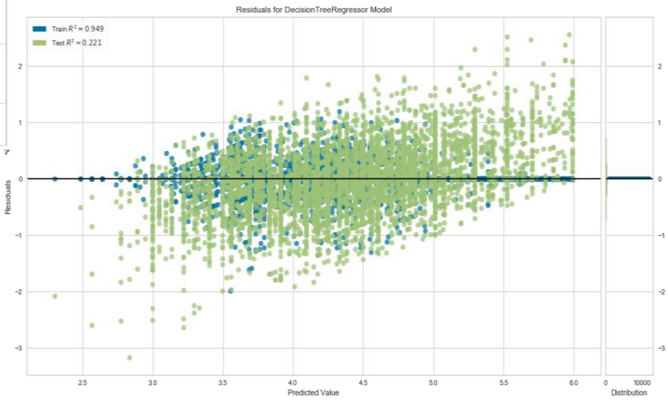


Fig 14: Residuals for Decision Tree Regression

4) K NEAREST NEIGHBORS

KNN predicts the outcome variable based on the distance measurement functions. The algorithm uses the similar features to predict the values of all the new data points. For continuous variable we use the Euclidean or Manhattan distance calculation techniques and select the optimized value of K. For our data set we have calculated K as 10 and obtained the following results.

Mean Absolute Error: 0.42617254192126436
Mean Squared Error: 0.2923055070894922
Root Mean Squared Error: 0.5406528526600893
Coefficient of Determination: 0.330948089246059

```
      Actual  Predicted
0   5.855072   4.873812
1   4.553877   4.285223
2   4.094345   3.529846
3   3.401197   3.964794
4   4.276666   4.313525
```

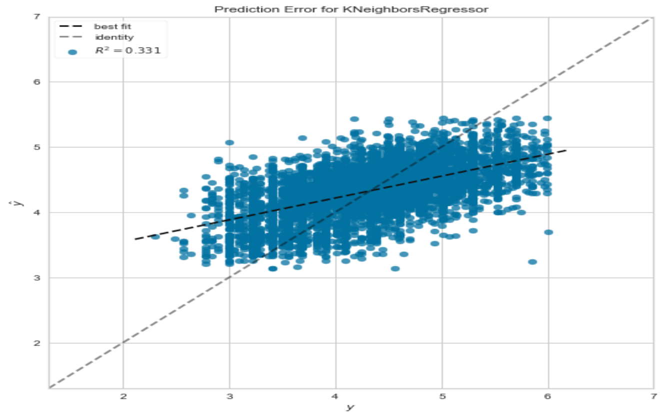


Fig 15: Prediction Error for KNN

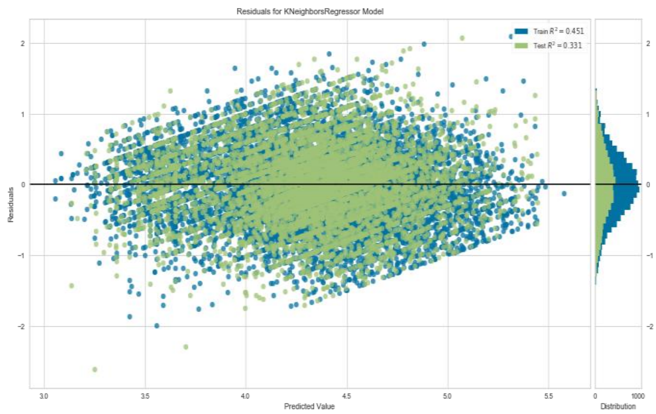


Fig 16: Residuals for KNN

5) RANDOM FOREST

Random forest regression model as the name itself suggests it creates a forest with the number of decision trees. The more the number of trees in the forest the better the algorithm performs with respect to the prediction. The model searches for the best feature among the subset of features instead of searching for the features while splitting the node. After applying the model to the dataset the following results are obtained as:

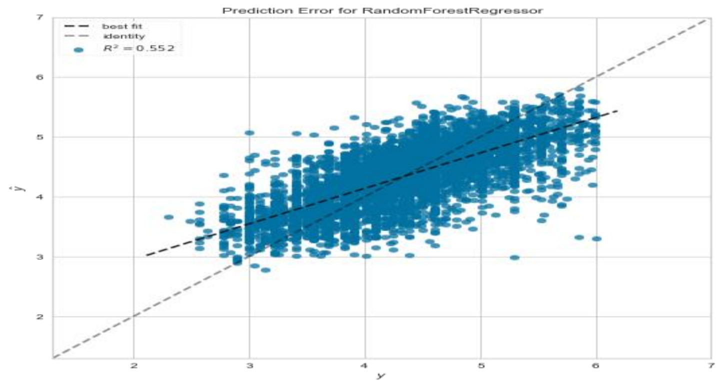


Fig 17: Prediction Error for Random Forest Regression

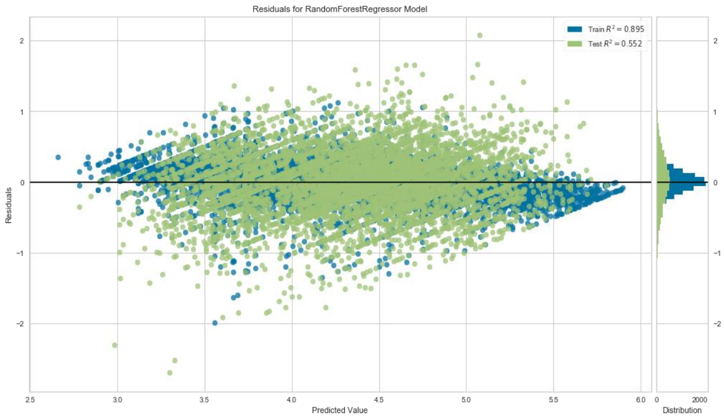


Fig 18: Residuals for Random Forest Regression

```
MAE is  0.33864798192778517
Coefficient of Determination:: 0.552399050795
2058
Root Mean Squared Error: 0.44221563205419245
Mean Squared Error: 0.19555466523308893
      Actual  Predicted
0   5.855072   5.500489
1   4.553877   4.538150
2   4.094345   3.765531
3   3.401197   3.749079
4   4.276666   4.325088
```

IX. RESULTS

ANALYSIS 1:

For the cross validation, since our feature variables are just 4, we have used k = 15. The results for testing the hypothesis are as follows:

1) LINEAR REGRESSION:

```
MAE: 0.37925473947695626
MSE: 0.23711599410554182
RMSE: 0.48694557612277556
Coefficient of Determination: 0.4572702016248
2657
0.4572702016248266
      Actual  Predicted
0   5.855072   4.389727
1   4.553877   4.380922
2   4.094345   3.748783
3   3.401197   3.748783
4   4.276666   4.389727
```

cross validation scores:

```
Accuracy of every fold in cross validation:
[0.37918829 0.50532231 0.45802735 0.51200261
0.42054653 0.44833803 0.43437954 0.37143517
0.49497771 0.44590961 0.31317018 0.41823587 0.51037608
0.38472555 0.40128497]

Mean of the validation score: 0.4331946543503
15
MSE of every fold in cross validation: [0.217
938 0.18768102 0.20794958 0.24828108 0.24099
608 0.24263276 0.23084179 0.29096942 0.257520
28 0.22983808 0.27993703 0.23295176 0.2162557
3 0.24549467 0.22522864]
MEan of MSE: 0.23655811469875593
```

2) RANDOM FOREST

```
MAE is  0.3759661171116998
variance: 0.4622530578709334
Root Mean Squared Error: 0.48470507381875266
Mean Squared Error: 0.23493900858564248
      Actual  Predicted
0   5.855072   4.396290
1   4.553877   4.303880
2   4.094345   3.748059
3   3.401197   3.748059
4   4.276666   4.396290
```

Cross validation scores:

```
Accuracy of every fold in cross validation:
0.3772647 0.50825255 0.46290356 0.52691106 .4
3964146 0.46216401 0.44624371 0.40154996 0.50
91735  0.45856015 0.30534585 0.42943197 0.532
80541 0.42354479 0.39510341]
Mean of the validation score: 0.4452597393303
206
MSE of every fold in cross validation: [0.212
45004 0.18656929 0.20607863 0.24069602 0.2330
5446 0.2365518 0.22599977 0.27702896 0.250281
58 0.22459061 0.28312606 0.22846859 0.206349
0.23000578 0.22755406]
MEan of MSE: 0.23125365639291853
```

ANALYSIS 2: Done using all the 10 variables

The results of all the algorithms are mentioned below. Model evaluation is done using evaluation metrics mentioned in section 3.

	MSE
Linear Regression	0.202012
Decision Tree	0.343124
KNN	0.292306
SVM	0.244387
Random Forest Regressor	0.195555

Fig 19: Mean Square Error Results of all the models

	R2 score
Linear Regression	0.537620
Decision Tree	0.223208
KNN	0.330948
SVM	0.440627
Random Forest Regressor	0.552399

Fig 20: R2 results of all the models

After calculating all the metrics, it is evident that Random forest and linear regression model worked the best for our dataset and produced least mean squared error. Out of all, random forest has the highest R2 value followed by linear regression and SVM. Decision tree algorithm performed the worst out of all. Also, as seen from its prediction and residual plots [fig, fig], the algorithm gave a good R2 score on the training dataset but performed poorly on the testing dataset.

Furthermore, the algorithms are evaluated using the cross validation score. As our dataset is large, after number of iterations, k = 8 gives highest accuracy for all.

1) LINEAR REGRESSION

```
Accuracy of every fold in cross validation:
[0.49622187 0.54451644 0.49740895 0.50407984
0.57116561 0.44638317 0.52863207 0.4512711]
Mean of the validation score:
0.5049598821910983
MSE of every fold in cross validation:
[0.18292566 0.21558531 0.2202283 0.21859613
0.20756591]
Mean of MSE: 0.2089802617765057
```

2) RANDOM FOREST

```
Accuracy of every fold in cross validation:
0.49622187 0.54451644 0.49740895 0.50407984 0
.57116561 0.44638317 0.52863207 0.4512711]
Mean of the validation score: 0.5049598821910
983
MSE of every fold in cross validation: [0.182
92566 0.21558531 0.2202283 0.21859613 0.20756
591]
Mean of MSE: 0.2089802617765057
```

3) SVR

```
Accuracy of every fold in cross validation: [
0.37642864 0.50832369 0.42242273 0.41391611 0
.45595343 0.3736427 0.46118163 0.38469612]
Mean of the validation score: 0.4245706301445
0824
MSE of every fold in cross validation: [0.225
14604 0.21650186 0.24738111 0.25414071 0.2684
1165 0.25217662 0.2309688 0.23980827]
```

Mean of MSE: 0.24181688230041232

4) KNN

Accuracy of every fold in cross validation: [0.29122544 0.35743532 0.32920785 0.2722369 .25023406]
Mean of the validation score: 0.30006791528529214
MSE of every fold in cross validation: [0.26842353 0.2933317 0.31272406 0.2996555 0.30543718]
Mean of MSE: 0.29591439589132634

5) DECISION TREE

Accuracy of every fold in cross validation: [0.11881243 0.26581176 0.20538147 0.15972736 0.23813867 0.05930223 0.11035235 0.07741884]
Mean of the validation score: 0.13501342888564677
MSE of every fold in cross validation: [0.3139465 0.34776161 0.35687068 0.37307067 0.41146924]
Mean of MSE: 0.3606237398205672

Again, from the validation score we verified the results. We split the data into 8 and tested the accuracy. Random forest and linear regression gives similar results. All the models accuracy has reduced by some percentage but again, decision tree algorithm performed much poorly.
In conclusion, we consider the validation score generated by the random forest algorithm using cross validation.

X. CONCLUSION

In this project, we implemented 5 algorithms to predict the price for the Airbnb listings of Montreal city based on the multiple factors. We started with 104 variables out of which we selected 10 features expected to affect the price the most. After data exploration and data cleaning, we generated the correlation matrix and found that out of all, variables accommodates, room_type_entire apartment, bedrooms and beds showed the highest correlation with the price.
Hence, we generated the hypothesis that "The '**room_type' (entire apartment), 'accommodates' and 'beds'** variables contribute the price of a listing in a very significant way. "

1) To test our hypothesis we performed linear regression and Random forest algorithm on the selected 4 variables and we found out that all the 4 variables have an influence on price **proving our Hypothesis 1**. Together they account for **44% of variability** in our data. Of course, more the number of people that can be accommodated, bigger the house would be and so will be the price. If seen logically, entire apartment > more accommodation > more number of bedrooms and beds > higher the price.
However, a large 56% of price is decided by other factors, therefore for analysis 2 we consider all the 10 variables to check the amount of variance they contribute towards the price. We predicted the price using 5 algorithms (Linear regression, Random forest, KNN, SVR and Decision tree) out of which linear regression and Random forest fits our data the best.
However, the results were not exactly as expected.
2) All the 10 predictor variables has an influence of only 50% on the price out of which 44% variance is due to 'accommodates, 'beds', 'bedrooms' and 'room_type_entire apart' only. That means the other 6 predictor variables contributes just 6% towards the price.
3) Also, it was unexpected that **"neighbourhood" and "property_type"** would not have any significant contribution towards the price given that in some of the property types where castles. Similarly, neighborhoods such as '**downtown Montreal', Le Plateau Mount royal', 'the village**' etc does not have higher price considering they are famous tourist spots [10].
4) In contrast, all the neighbourhoods have similar pricing range which can be understood knowing the fact the in general Montreal is an expensive city given its heritage and architectural icons [11].
5) A large 50% of the influence remains unexplained. One of the reasons can be the overfitting of our model as concluded from the cross validation scores.
6) To further improve the predictions, more variables such as amenities, housing rule, seasonality etc. can be included. Since, the variables amenities and housing rules contains lots of text, we didn't include them in our analysis for simplicity and getting a crisp analysis.
Also, the problem can be turned into classification to improve the accuracy.

Business Insights
In a nutshell, in our analysis we predicted 50% (half) of the factors that affects the pricing. Using Random forest regression, we fit a model that predicted a MAE of $33.86 for all the listings and $19.55 on single listing. This can give a better understanding to the hosts and customers about the correct price to be charged and paid. Also, this analysis can be used by Airbnb in giving better suggestions to the new hosts. It can be used by hosts to better price their property according to the features present in their property and make a fine profit. They can gain confidence in demanding the price that they deemed fit.
Additionally, it can be used to avoid paying unfair amount that is sometimes charged by the hosts. This analysis gives a crisp idea about what important features to look for before booking and paying for your next stay.

REFERENCES

1. Sraders, Anne (September 24, 2018). "How Does Airbnb Work for Hosts and Travelers?" TheStreet, Inc. Retrieved April 12, 2019.

2 Airbnb Fast facts, < https://press.airbnb.com/fast-facts/ >

3 Leaflet, @openstreetmap, date: November 12, 2018 <http://insideairbnb.com/montreal/?neighbourhood=&filterEntireHomes=false&filterHighlyAvailable=false&filterRecentReviews=false&filterMultiListings=false>

4) Samuel klam, Airbnb pricing prediction, date: 22nd January, 2018, <https://airbnb-pricing-prediction.herokuapp.com/ >

5) Philip Mohun, Making models – Airbnb price predictions, date : 27th February, 2018 <https://medium.com/datadriveninvestor/making-models-airbnb-price-prediction-data-analysis-15b9af87c9d8>

6) Laura Lewis, Predicting Airbnb prices with machine learning and deep learning, date: 22nd May, 2019 < https://towardsdatascience.com/predicting-airbnb-prices-with-machine-learning-and-deep-learning-f46d44afb8a6>

7) Dmytro Lakubovskyi, Digging into Airbnb data : reviews, sentiments, superhosts, price predictions, date: 1st October, 2018 < https://towardsdatascience.com/digging-into-airbnb-data-reviews-sentiments-superhosts-and-prices-prediction-part1-6c80ccb26c6a>

8) Dima, Date: 2nd October, 2018 <https://github.com/Dima806/Airbnb_project?source=post_page--------------------------->

9) FaisalAl-Tameemi, Airbnb listings data- Toronto date: 7th November, 2018 < https://medium.com/datadriveninvestor/airbnb-listings-analysis-in-toronto-october-2018-2a5358bae007>

10) Michael D'Alimonte, 10 things that makes Montreal the most unique city in Canada, date: 5 years ago < https://www.mtlblog.com/lifestyle/things-that-make-montreal-the-most-unique-city-in-canada >

11) Kimberly Chen, A quick travel guide to Montreal neighbourhoods, date: 19th February, 2016 <https://matadornetwork.com/trips/quick-travel-guide-montreal-neighbourhoods/>

12) 12 Inside Airbnb data, < http://insideairbnb.com/get-the-data.html>

In [163]:

```
#1.Importing Required libraries

#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import pylab
import scipy.stats as stats
%matplotlib inline

# models for regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

#Evaluation metrics
from sklearn import metrics
from sklearn.metrics import accuracy_score, mean_absolute_error,r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from yellowbrick.regressor import ResidualsPlot
from yellowbrick.regressor import PredictionError
```

In [164]:

```
#2. Reading the csv
dataset = pd.read_csv('C:\\Users\\Pravinaben\\Desktop\\MON\\listings.csv')
```

```
C:\Users\Pravinaben\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3049:
DtypeWarning: Columns (61,62) have mixed types. Specify dtype option on import or set
low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [165]:

```
#3.1 Taking all the necessary Features
columns = ['id',
    'price',
    'property_type',
    'room_type',
    'accommodates',
    'bedrooms',
    'beds',
    'neighbourhood_cleansed',
    'number_of_reviews',
    'review_scores_rating',
    'availability_365',
    'bathrooms'
]
dataset = pd.read_csv('C:\\Users\\Pravinaben\\Desktop\\MON\\listings.csv', usecols=columns)
```

In [166]:

```
#3.2 Basic Analysis of dataset

#order of pandas dataframe: Around 21k instances , 11 (independent variables) and 1 ( target varia
ble)

dataset.shape
```

Out[166]:

```
(21104, 12)
```

In [167]:

```
#3.3 first few instances of dataset
dataset.head(5)
```

Out[167]:

	id	neighbourhood_cleansed	property_type	room_type	accommodates	bathrooms	bedrooms	beds	price	availability_365
0	2078	Le Plateau-Mont-Royal	House	Private room	2	1.0	1.0	1.0	$39.00	193
1	2843	Le Sud-Ouest	Serviced apartment	Private room	2	1.0	1.0	1.0	$30.00	232
2	14584	Le Plateau-Mont-Royal	Loft	Entire home/apt	4	1.0	1.0	1.0	$175.00	322
3	29059	Ville-Marie	Apartment	Entire home/apt	4	1.0	1.0	2.0	$94.00	292
4	29061	Ville-Marie	House	Entire home/apt	5	1.0	2.0	3.0	$145.00	334

In [168]:

```
#3.4 Types of data in the dataframe

dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21104 entries, 0 to 21103
Data columns (total 12 columns):
id                        21104 non-null int64
neighbourhood_cleansed    21104 non-null object
property_type             21104 non-null object
room_type                 21104 non-null object
accommodates              21104 non-null int64
bathrooms                 21074 non-null float64
bedrooms                  21100 non-null float64
beds                      21077 non-null float64
price                     21104 non-null object
availability_365          21104 non-null int64
number_of_reviews         21104 non-null int64
review_scores_rating      16379 non-null float64
dtypes: float64(4), int64(4), object(4)
memory usage: 1.9+ MB
```

In [169]:

```
#3.5 Distribution of data

dataset.describe()
```

Out[169]:

	id	accommodates	bathrooms	bedrooms	beds	availability_365	number_of_reviews	review_scores_ra
count	2.110400e+04	21104.000000	21074.000000	21100.000000	21077.000000	21104.000000	21104.000000	16379.000
mean	2.114740e+07	3.579795	1.146745	1.427630	1.869004	102.961192	20.612301	93.541
std	1.049004e+07	2.405151	0.500057	1.037121	1.452110	125.035151	41.218796	9.085
min	2.078000e+03	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	20.000
25%	1.323859e+07	2.000000	1.000000	1.000000	1.000000	0.000000	1.000000	91.000
50%	2.196378e+07	3.000000	1.000000	1.000000	1.000000	38.000000	5.000000	96.000
75%	3.028807e+07	4.000000	1.000000	2.000000	2.000000	193.000000	20.000000	100.000
max	3.667266e+07	18.000000	20.000000	20.000000	50.000000	365.000000	610.000000	100.000

In [170]:

```python
#3.6 No of listings
dataset.groupby(by='neighbourhood_cleansed').count()[['id']].sort_values(by='id', ascending=False)
.head(10)
```

Out[170]:

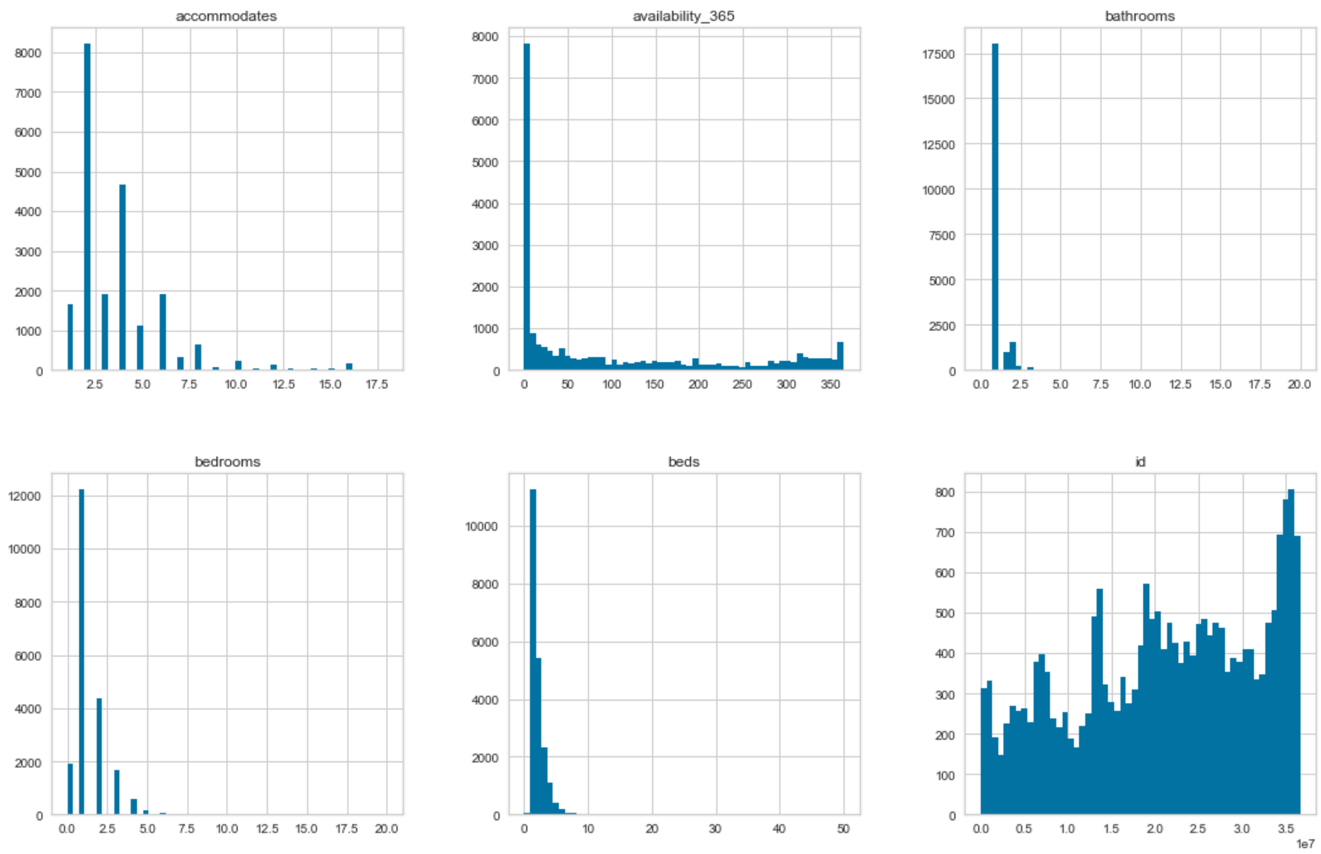
neighbourhood_cleansed	id
Le Plateau-Mont-Royal	6167
Ville-Marie	5529
Rosemont-La Petite-Patrie	1998
Côte-des-Neiges-Notre-Dame-de-Grâce	1432
Le Sud-Ouest	1293
Villeray-Saint-Michel-Parc-Extension	1146
Mercier-Hochelaga-Maisonneuve	1039
Verdun	510
Ahuntsic-Cartierville	341
Outremont	292

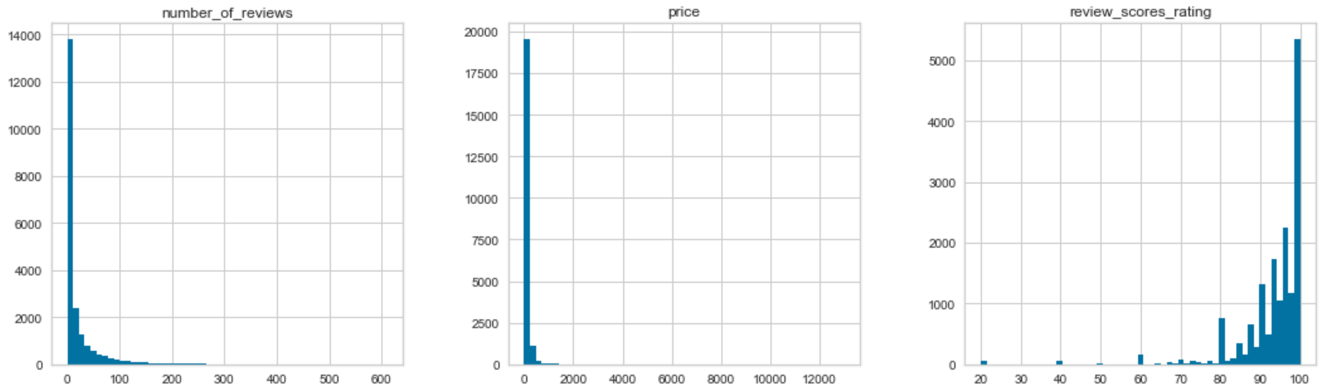
In [171]:

```python
# 4. Exploratory Data Analysis

#4.1 Histogram plot of data

# convert the price format
dataset['price'] = (dataset['price'].str.replace(r'[^-+\d.]', '').astype(float))

dataset.hist(bins=55, figsize= (18,18))
plt.show
```

Out[171]:

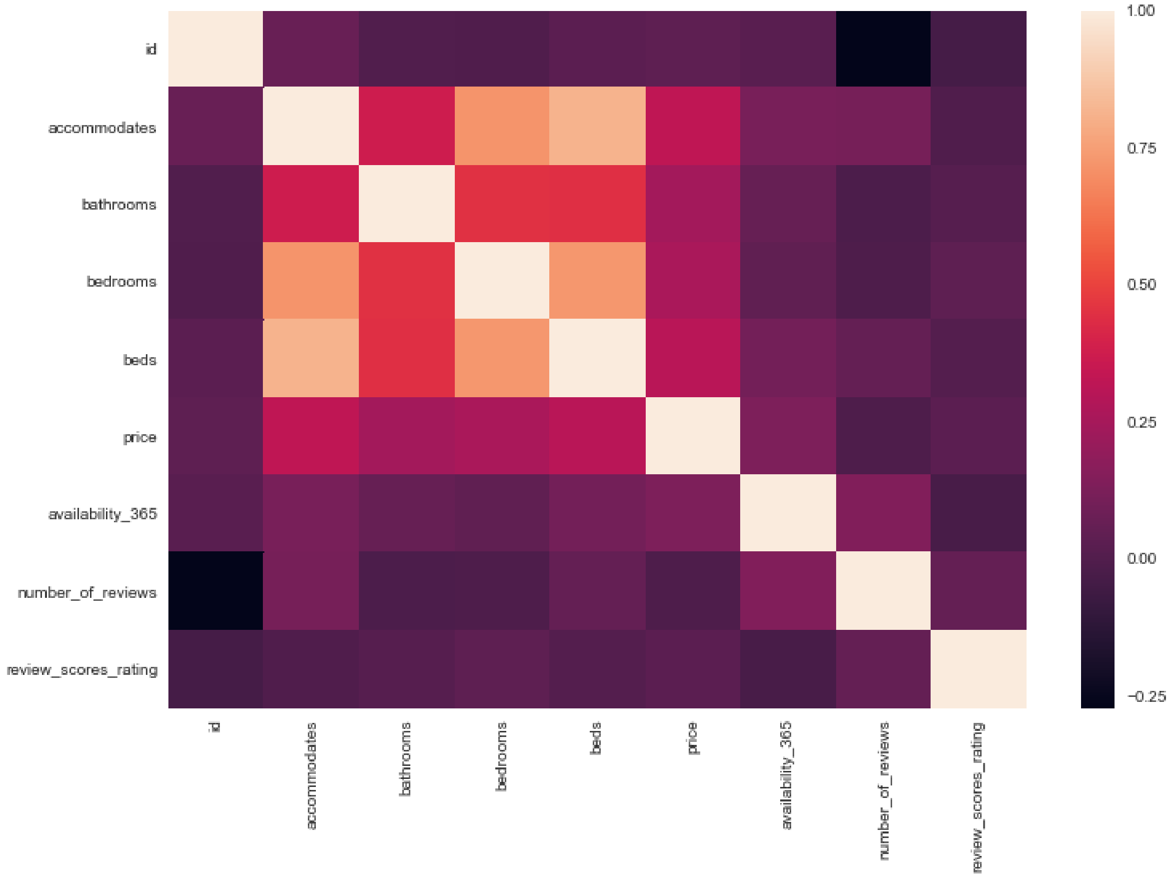
```
<function matplotlib.pyplot.show(*args, **kw)>
```





In [172]:

```
#4.2 Identifing correlation
plt.figure(figsize=(12,8))
sns.heatmap(dataset.corr());
plt.show()
```



In [72]:

```
#4.3 Price variable analysis

dataset['price'] = dataset['price'][(dataset['price'] != 0)]
dataset['price'].max()
```

Out[72]:

12960.0

In [73]:

```
# Minimum Price
dataset['price'].min()
```

Out[73]:

```
10.0
```

In [16]:

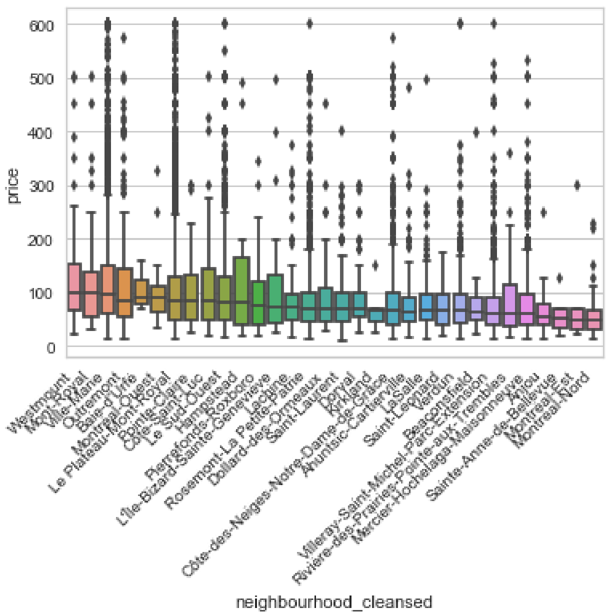
```python
# Mean Price
dataset['price'].mean()
```

Out[16]:

```
117.247180362051
```

In [17]:

```python
#4.4 Comparing the neighbourhood_cleansed vs price

sort_price = dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)]\
                    .groupby('neighbourhood_cleansed')['price']\
                    .median()\
                    .sort_values(ascending=False)\
                    .index
sns.boxplot(y='price', x='neighbourhood_cleansed', data=dataset.loc[(dataset.price <= 1000) & (data
set.price > 0)],
            order=sort_price)
ax = plt.gca()
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
plt.show();
```



In [162]:

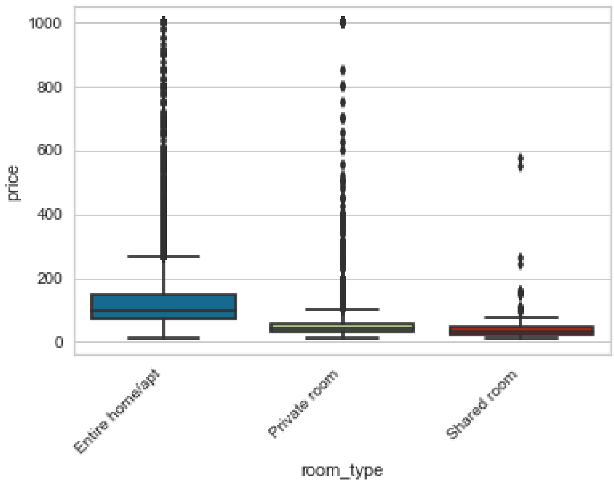
```python
#4.5 Comparing the property_type vs price
sort_price = dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)]\
                    .groupby('property_type')['price']\
                    .median()\
                    .sort_values(ascending=False)\
                    .index
sns.boxplot(y='price', x='property_type', data=dataset.loc[(dataset.price <= 1000) & (dataset.price
> 0)], order=sort_price)
ax = plt.gca()
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
plt.show();
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-162-7d146d78932e> in <module>
      1 #4.5 Comparing the property_type vs price
----> 2 sort_price = dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)]\
      3                     .groupby('property_type')['price']\
```

```
      4                        .median() \
      5                        .sort_values(ascending=False)\

~\Anaconda3\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)
   5065             if self._info_axis._can_hold_identifiers_and_holds_name(name):
   5066                 return self[name]
-> 5067             return object.__getattribute__(self, name)
   5068 
   5069     def __setattr__(self, name, value):

AttributeError: 'DataFrame' object has no attribute 'price'
```

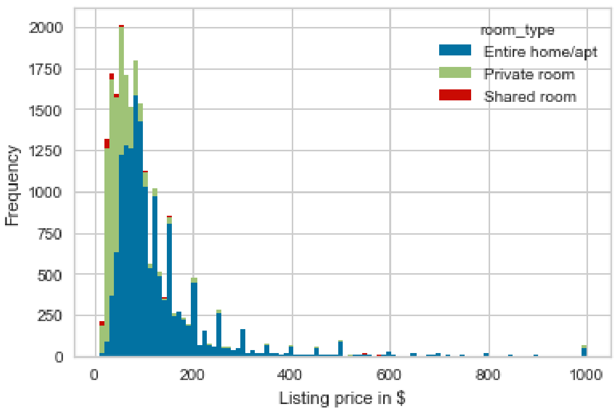
In [19]:

```
#4.6 Comparing the room_type vs price
sort_price = dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)]\
                    .groupby('room_type')['price']\
                    .median()\
                    .sort_values(ascending=False)\
                    .index
sns.boxplot(y='price', x='room_type', data=dataset.loc[(dataset.price <= 1000) & (dataset.price > 0
)], order=sort_price)
ax = plt.gca()
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
plt.show();
```



In [20]:

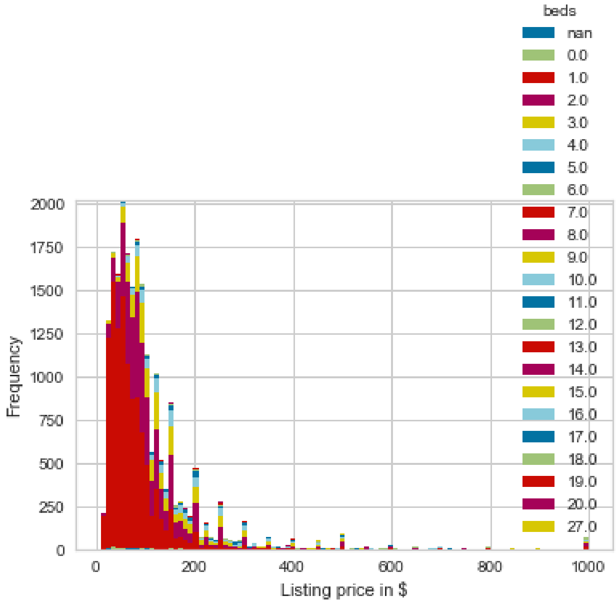
```
#4.6.1 Comparing the room_type vs price
dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)].pivot(columns = 'room_type', values = 'p
rice').plot.hist(stacked = True, bins=100)
plt.xlabel('Listing price in $');
```



In [21]:

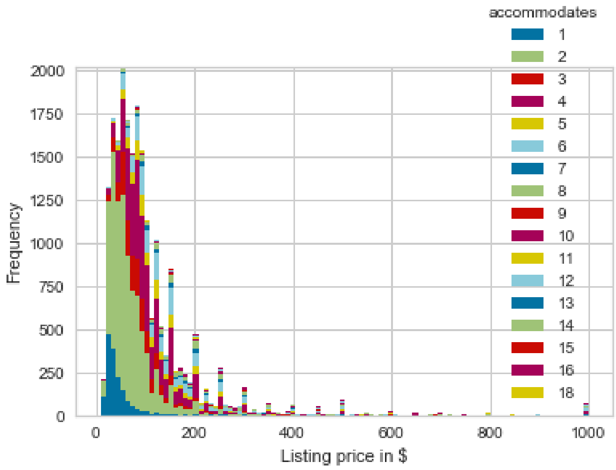
```
#4.7 Comparing the beds vs price
dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)].pivot(columns = 'beds',values = 'price')
.plot.hist(stacked = True,bins=100)
```

```
.plot.hist(stacked = True,bins=100)
plt.xlabel('Listing price in $');
```



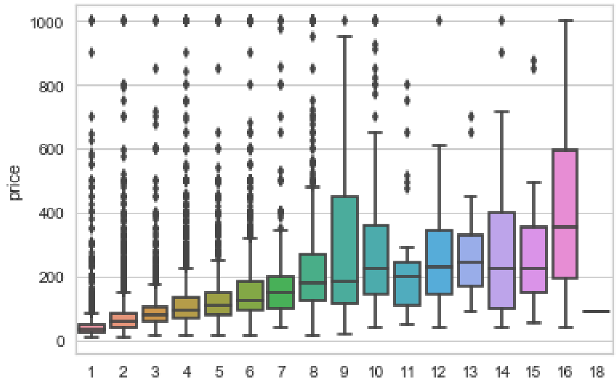
In [22]:

```
#4.8 Comparing the accommodates vs price
dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)].pivot(columns = 'accommodates',values =
'price').plot.hist(stacked = True,bins=100)
plt.xlabel('Listing price in $');
```



In [23]:

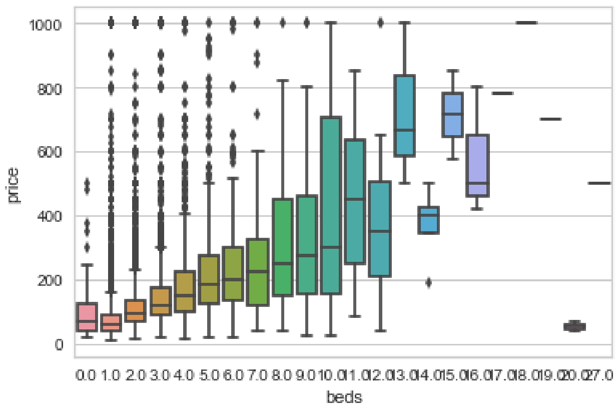
```
#4.8.1 Comparing the accommodates vs price
sns.boxplot(y='price', x='accommodates', data = dataset.loc[(dataset.price <= 1000) & (dataset.pric
e > 0)])
plt.show();
```



accommodates

In [24]:

```
#4.9 Comparing the beds vs price
sns.boxplot(y='price', x='beds', data = dataset.loc[(dataset.price <= 1000) & (dataset.price > 0)])
plt.show();
```

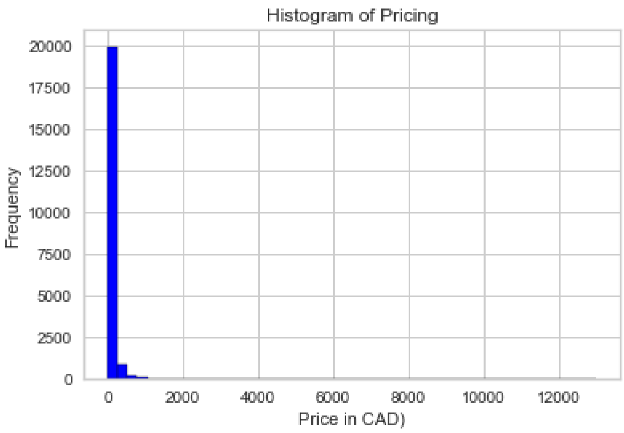


In [74]:

```
#5 Data Cleaning

#5.1 Data analysis and cleaning for Price variable

# histogram distribution of price
plt.hist(dataset['price'], color = 'blue', edgecolor = 'black',
         bins = int(50))

sns.distplot(dataset['price'], hist=True, kde=False,
             bins=int(50), color = 'blue',
             hist_kws={'edgecolor':'black'})

plt.title('Histogram of Pricing')
plt.xlabel('Price in CAD)')
plt.ylabel('Frequency')

print("Skewness: %f" % dataset['price'].skew())
print("Kurtosis: %f" % dataset['price'].kurt())
```

```
C:\Users\Pravinaben\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824: RuntimeWarning:
invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\Pravinaben\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825: RuntimeWarning:
invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

```
Skewness: 22.556308
Kurtosis: 903.892000
```

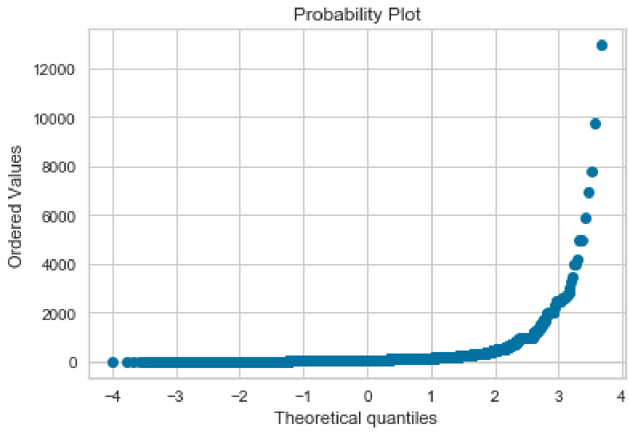


In [27]:

```
#Probability plot of price
stats.probplot(dataset['price'], plot=plt)
```

Out[27]:

```
((array([-3.99138831, -3.77587696, -3.65797805, ...,  3.65797805,
          3.77587696,  3.99138831]),
  array([1.000e+01, 1.000e+01, 1.200e+01, ..., 1.296e+04,       nan,
               nan])),
 (nan, nan, nan))
```



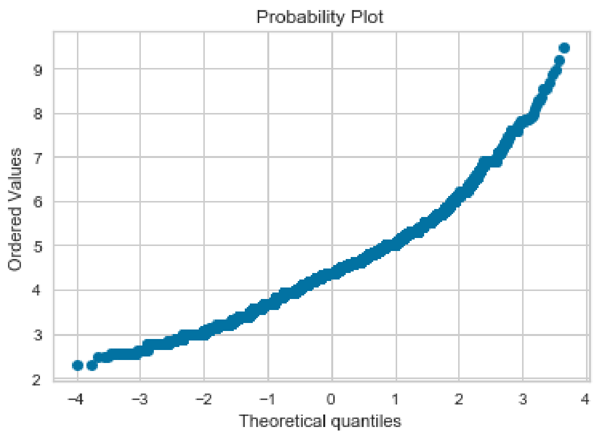
In [77]:

```
# Applying natural log to the 'price'
dataset['price_log'] = dataset['price'].apply(lambda x: math.log(x))

# Removing the price column since we have added the price_log column
dataset = dataset.drop('price', axis = 1)
```

In [78]:

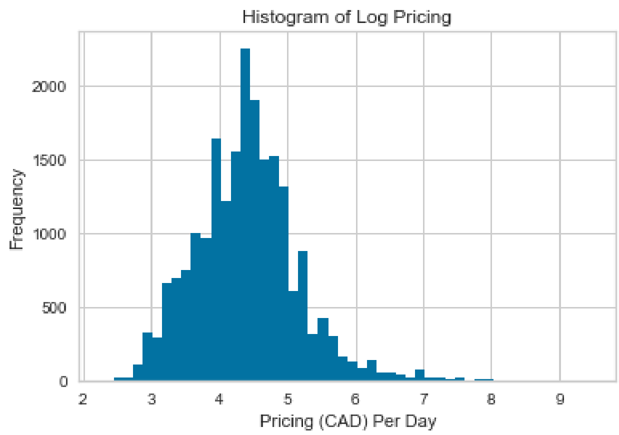
```
# Checking the distribution for price_log
stats.probplot(dataset['price_log'], dist="norm", plot=pylab)
pylab.show()
```

```
C:\Users\Pravinaben\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:877:
RuntimeWarning: invalid value encountered in greater
  return (self.a < x) & (x < self.b)
C:\Users\Pravinaben\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:877:
RuntimeWarning: invalid value encountered in less
  return (self.a < x) & (x < self.b)
C:\Users\Pravinaben\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:1831:
RuntimeWarning: invalid value encountered in less_equal
  cond2 = cond0 & (x <= self.a)
```



In [30]:

```
#visualize distribution of price_log (target variable)
plt.hist(dataset['price_log'], bins=50)
plt.title("Histogram of Log Pricing")
plt.xlabel("Pricing (CAD) Per Day")
plt.ylabel("Frequency")
plt.show()

print("Skewness: %f" % dataset['price_log'].skew())
print("Kurtosis: %f" % dataset['price_log'].kurt())
```




```
Skewness: 0.652819
Kurtosis: 1.683841
```

In [79]:

```
#5.2 Checking for null values

dataset.isnull().sum()
```

Out[79]:

```
id                           0
neighbourhood_cleansed       0
property_type                0
room_type                    0
accommodates                 0
bathrooms                   30
bedrooms                     4
beds                        27
availability_365             0
number_of_reviews            0
review_scores_rating      4725
price_log                    2
dtype: int64
```

In [80]:

```
#5.3 Dropping the values of bathrooms bedrooms and beds since there are very few missing values
dataset = dataset.dropna(how='any', subset=['bedrooms','bathrooms','beds','price_log'])
```

In [81]:

```
#5.4 Replacing review score rating missing values with mean
dataset['review_scores_rating'].fillna((dataset['review_scores_rating'].mean()), inplace=True)
```

In [82]:

```
#5.5 Rechecking for null values
dataset.isnull().sum()
```

Out[82]:

```
id                        0
neighbourhood_cleansed    0
property_type             0
room_type                 0
accommodates              0
bathrooms                 0
bedrooms                  0
beds                      0
availability_365          0
number_of_reviews         0
review_scores_rating      0
price_log                 0
dtype: int64
```

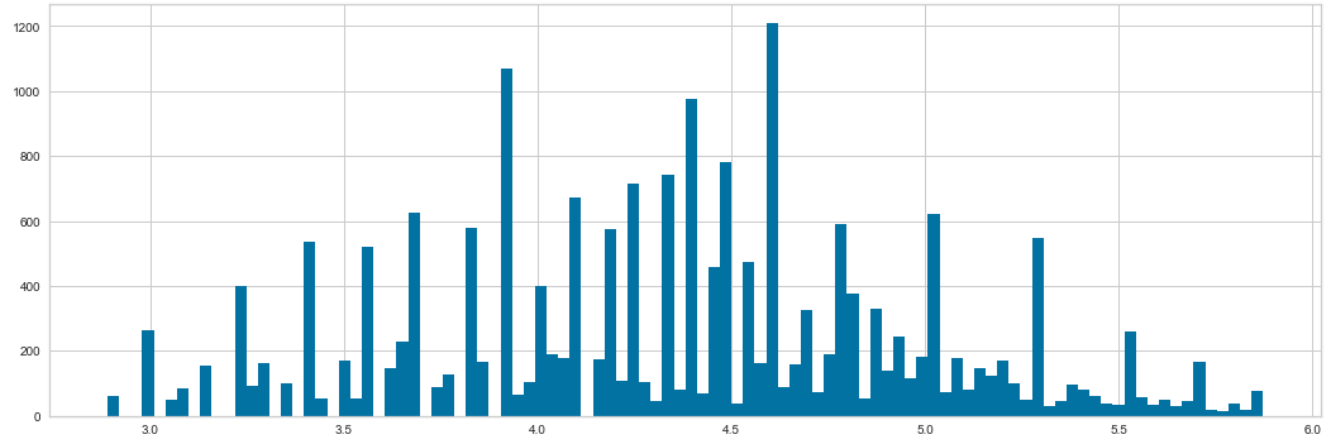
In [83]:

```
#5.6 Encoding the categorical variables
encoding = dataset.copy()
encoding = pd.get_dummies(encoding, columns=['room_type', 'neighbourhood_cleansed', 'property_type'
])

print(encoding.head())
print ('Number of Columns:', len(encoding.columns))

# move target predictor 'price' to the end of the dataframe
cols = list(encoding.columns.values)
idx = cols.index('price_log')
rearrange_cols = cols[:idx] + cols[idx+1:] + [cols[idx]]
dataset = encoding[rearrange_cols]
```

```
      id  accommodates  bathrooms  bedrooms  beds  availability_365  \
0   2078             2        1.0       1.0   1.0               193
1   2843             2        1.0       1.0   1.0               232
2  14584             4        1.0       1.0   1.0               322
3  29059             4        1.0       1.0   2.0               292
4  29061             5        1.0       2.0   3.0               334

   number_of_reviews  review_scores_rating  price_log  \
0                245                  93.0   3.663562
1                125                  88.0   3.401197
2                157                  98.0   5.164786
3                293                  93.0   4.543295
4                 49                  92.0   4.976734

   room_type_Entire home/apt  ...  property_type_Houseboat  \
0                          0  ...                        0
1                          0  ...                        0
2                          1  ...                        0
3                          1  ...                        0
4                          1  ...                        0

   property_type_Loft  property_type_Nature lodge  property_type_Other  \
0                   0                           0                    0
1                   0                           0                    0
2                   1                           0                    0
3                   0                           0                    0
4                   0                           0                    0

   property_type_Resort  property_type_Serviced apartment  property_type_Tent  \
0                     0                                 0                   0
1                     0                                 1                   0
2                     0                                 0                   0
3                     0                                 0                   0
4                     0                                 0                   0

   property_type_Tiny house  property_type_Townhouse  property_type_Villa
0                         0                        0                    0
1                         0                        0                    0
2                         0                        0                    0
3                         0                        0                    0
4                         0                        0                    0

[5 rows x 76 columns]
Number of Columns: 76
```

In [84]:

```
#5.7 Fixing outliers
# Outlier : we define outlier as values which are two standard deviations away from the mean.
# Using Z-Score method

def reject_outliers(price_log):
    m = np.median(dataset['price_log'])
    sd = np.std(dataset['price_log'])
    filtered= [a for a in (dataset['price_log']) if (m - 2 * sd < a < m + 2 * sd)]
    return filtered

figure_size = plt.rcParams["figure.figsize"]
figure_size[0] =18.0
figure_size[1] = 6.0

filtered = reject_outliers('price_log')
plt.hist(filtered, 100)
figure_size[0]=17.0
figure_size[1]=9.0
plt.show()

data_price_log = pd.DataFrame(filtered)
data_price_log.shape
```



Out[84]:

```
(20219, 1)
```

In [85]:

```
#distribution of price_log variable
dataset['price_log'].describe()
```

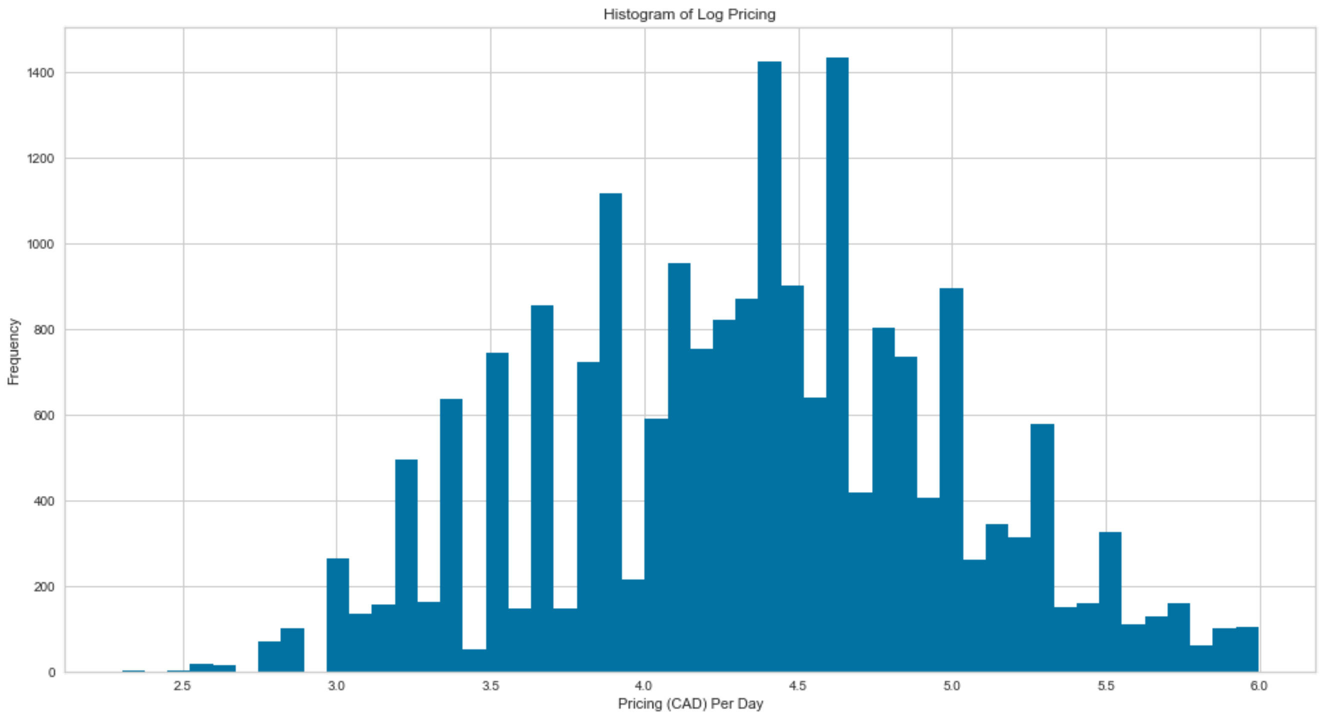
Out[85]:

```
count    21052.000000
mean         4.403857
std          0.746461
min          2.302585
25%          3.912023
50%          4.382027
75%          4.828314
max          9.469623
Name: price_log, dtype: float64
```

In [86]:

```
#5.8 Removing Outliers

#Outlier for price_log column would be : mean + 2 * sd = 4.4 + (2 x 0.7) = 6 above

newdataset = dataset[dataset['price_log']<6]

plt.hist(newdataset['price_log'], bins=50)
```

```
plt.title("Histogram of Log Pricing")
plt.xlabel("Pricing (CAD) Per Day")
plt.ylabel("Frequency")
plt.show()

newdataset.shape
```



Out[86]:

```
(20501, 76)
```

In [87]:

```
#5.9 Correlation of variables with target variable:

corr_mat = newdataset.corr()
corr_mat['price_log'].sort_values(ascending=False)
```

Out[87]:

```
price_log                                        1.000000
room_type_Entire home/apt                        0.582242
accommodates                                     0.536265
beds                                             0.456942
bedrooms                                         0.413456
neighbourhood_cleansed_Ville-Marie               0.163347
availability_365                                 0.152498
bathrooms                                        0.133105
property_type_Condominium                        0.107136
property_type_Loft                               0.100125
property_type_Serviced apartment                 0.098102
number_of_reviews                                0.080634
neighbourhood_cleansed_Le Plateau-Mont-Royal     0.044822
property_type_Townhouse                          0.043580
property_type_Aparthotel                         0.041239
id                                               0.040225
review_scores_rating                             0.039055
neighbourhood_cleansed_Westmount                 0.023843
property_type_Cottage                            0.021505
property_type_Castle                             0.020095
property_type_Boutique hotel                     0.018453
property_type_Farm stay                          0.017575
property_type_Hotel                              0.015669
neighbourhood_cleansed_Le Sud-Ouest              0.015252
property_type_Nature lodge                       0.015096
neighbourhood_cleansed_Outremont                 0.013320
property_type_Earth house                        0.010821
property_type_House                              0.008828
```

```
neighbourhood_cleansed_Mont-Royal                                    0.008658
neighbourhood_cleansed_Côte-Saint-Luc                                0.006461
                                                                       ...
neighbourhood_cleansed_Beaconsfield                                 -0.005507
neighbourhood_cleansed_Kirkland                                     -0.005992
neighbourhood_cleansed_Montréal-Est                                 -0.006306
neighbourhood_cleansed_Hampstead                                    -0.007023
neighbourhood_cleansed_Pierrefonds-Roxboro                          -0.007240
neighbourhood_cleansed_Lachine                                      -0.007323
neighbourhood_cleansed_Sainte-Anne-de-Bellevue                      -0.012189
property_type_Campsite                                              -0.014407
neighbourhood_cleansed_Saint-Laurent                                -0.017482
property_type_Bed and breakfast                                     -0.018051
neighbourhood_cleansed_Saint-Léonard                                -0.020160
neighbourhood_cleansed_LaSalle                                      -0.020603
property_type_Guest suite                                           -0.021628
neighbourhood_cleansed_Rivière-des-Prairies-Pointe-aux-Trembles     -0.022165
property_type_Other                                                 -0.023377
neighbourhood_cleansed_Anjou                                        -0.028449
property_type_Guesthouse                                            -0.028782
property_type_Bungalow                                              -0.034353
neighbourhood_cleansed_Ahuntsic-Cartierville                        -0.038979
neighbourhood_cleansed_Montréal-Nord                                -0.041525
neighbourhood_cleansed_Verdun                                       -0.041778
neighbourhood_cleansed_Rosemont-La Petite-Patrie                    -0.054018
property_type_Hostel                                                -0.061698
neighbourhood_cleansed_Côte-des-Neiges-Notre-Dame-de-Grâce          -0.080063
neighbourhood_cleansed_Mercier-Hochelaga-Maisonneuve                -0.085260
neighbourhood_cleansed_Villeray-Saint-Michel-Parc-Extension         -0.097080
room_type_Shared room                                               -0.117794
property_type_Apartment                                             -0.144734
room_type_Private room                                              -0.563361
property_type_Cave                                                        NaN
Name: price_log, Length: 76, dtype: float64
```

In [88]:

```
newdataset.drop('id', axis = 1)
```

Out[88]:

	accommodates	bathrooms	bedrooms	beds	availability_365	number_of_reviews	review_scores_rating	room_type_Entire home/apt	roon
0	2	1.0	1.0	1.0	193	245	93.000000	0	
1	2	1.0	1.0	1.0	232	125	88.000000	0	
2	4	1.0	1.0	1.0	322	157	98.000000	1	
3	4	1.0	1.0	2.0	292	293	93.000000	1	
4	5	1.0	2.0	3.0	334	49	92.000000	1	
5	2	1.0	1.0	1.0	244	131	89.000000	0	
6	5	1.0	2.0	3.0	252	4	93.000000	1	
7	2	1.0	1.0	1.0	239	34	99.000000	1	
8	1	1.0	3.0	1.0	333	12	87.000000	0	
9	4	1.0	2.0	3.0	259	351	93.000000	1	
11	7	1.0	4.0	4.0	365	19	89.000000	0	
12	4	1.0	1.0	1.0	171	81	97.000000	1	
13	2	1.0	2.0	2.0	262	42	98.000000	1	
14	4	1.0	2.0	2.0	142	165	86.000000	1	
15	6	3.0	3.0	3.0	301	37	97.000000	1	
16	2	1.0	1.0	1.0	58	52	97.000000	0	
17	6	1.5	2.0	2.0	221	19	98.000000	1	
18	3	1.0	1.0	1.0	84	67	98.000000	0	
20	3	1.0	1.0	1.0	224	99	97.000000	1	
21	2	1.0	1.0	1.0	3	260	93.000000	1	
22	3	1.0	1.0	1.0	207	207	91.000000	1	

	accommodates	bathrooms	bedrooms	beds	availability_365	number_of_reviews	review_scores_rating	room_type_Entire home/apt	roon
22	3	1.0	1.0	1.0	207	207	91.000000	1	
23	4	1.0	1.0	1.0	185	135	90.000000	1	
24	4	1.0	1.0	1.0	315	235	93.000000	1	
25	3	1.0	1.0	1.0	298	191	95.000000	1	
27	2	1.0	1.0	1.0	16	15	91.000000	1	
28	4	1.0	1.0	1.0	192	5	85.000000	1	
29	4	1.0	1.0	1.0	128	169	84.000000	1	
30	8	2.5	4.0	4.0	0	30	96.000000	1	
31	2	1.0	1.0	1.0	0	181	97.000000	0	
32	2	1.0	1.0	1.0	274	75	97.000000	1	
...	...	...	...	...	...	...	...	...	
21073	2	1.5	1.0	1.0	318	0	93.537615	0	
21074	2	1.5	1.0	1.0	358	0	93.537615	0	
21075	5	1.0	2.0	3.0	42	0	93.537615	1	
21076	2	1.0	1.0	1.0	92	0	93.537615	1	
21077	2	1.0	1.0	1.0	174	0	93.537615	0	
21078	8	1.0	2.0	5.0	75	0	93.537615	1	
21079	2	1.5	1.0	1.0	171	0	93.537615	0	
21080	3	1.0	1.0	2.0	48	0	93.537615	0	
21082	6	1.0	2.0	3.0	353	0	93.537615	1	
21083	4	0.0	0.0	1.0	1	0	93.537615	1	
21084	3	1.0	1.0	1.0	19	0	93.537615	1	
21085	2	1.0	1.0	1.0	0	0	93.537615	1	
21086	2	1.0	1.0	1.0	28	0	93.537615	0	
21087	4	1.0	2.0	2.0	17	0	93.537615	1	
21088	2	1.0	1.0	1.0	15	0	93.537615	1	
21089	6	1.0	2.0	2.0	37	0	93.537615	1	
21090	2	1.0	1.0	1.0	8	0	93.537615	0	
21091	2	1.0	1.0	1.0	5	0	93.537615	0	
21092	6	1.5	2.0	3.0	47	0	93.537615	1	
21093	4	1.0	2.0	2.0	4	0	93.537615	1	
21094	4	1.0	2.0	3.0	66	0	93.537615	1	
21095	5	1.0	3.0	4.0	24	0	93.537615	1	
21096	2	1.0	1.0	1.0	31	0	93.537615	0	
21097	2	3.5	2.0	2.0	251	0	93.537615	1	
21098	6	1.0	2.0	3.0	231	0	93.537615	1	
21099	5	1.0	0.0	2.0	80	0	93.537615	1	
21100	2	1.0	1.0	2.0	90	0	93.537615	0	
21101	2	1.0	1.0	1.0	5	0	93.537615	1	
21102	3	1.0	1.0	1.0	167	0	93.537615	1	
21103	2	1.0	1.0	1.0	125	0	93.537615	0	

20501 rows × 75 columns

In [177]:

```
#6 Modelling
#Training and Testing data

X = newdataset.iloc[: , 1:-1].values
y = newdataset.iloc[: , -1].values
```

In [178]:

```
#30 percent of the data is allocated for testing
# random state is set not to introduce sampling bias

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

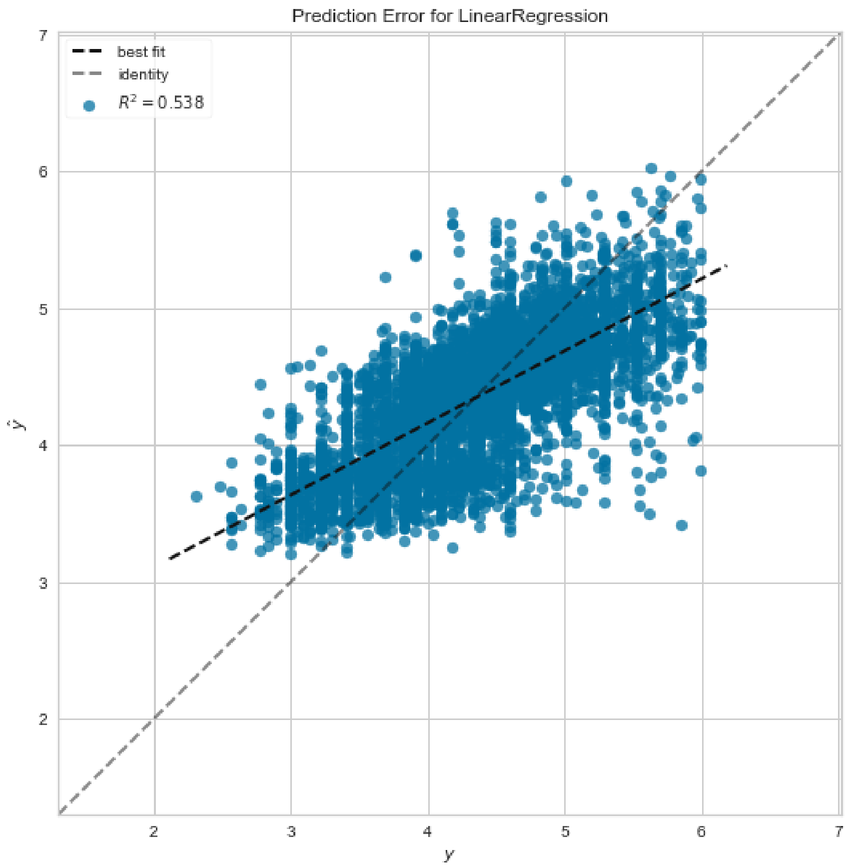
In [179]:

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

```
(14350, 74)
(14350,)
(6151, 74)
(6151,)
```

In [103]:

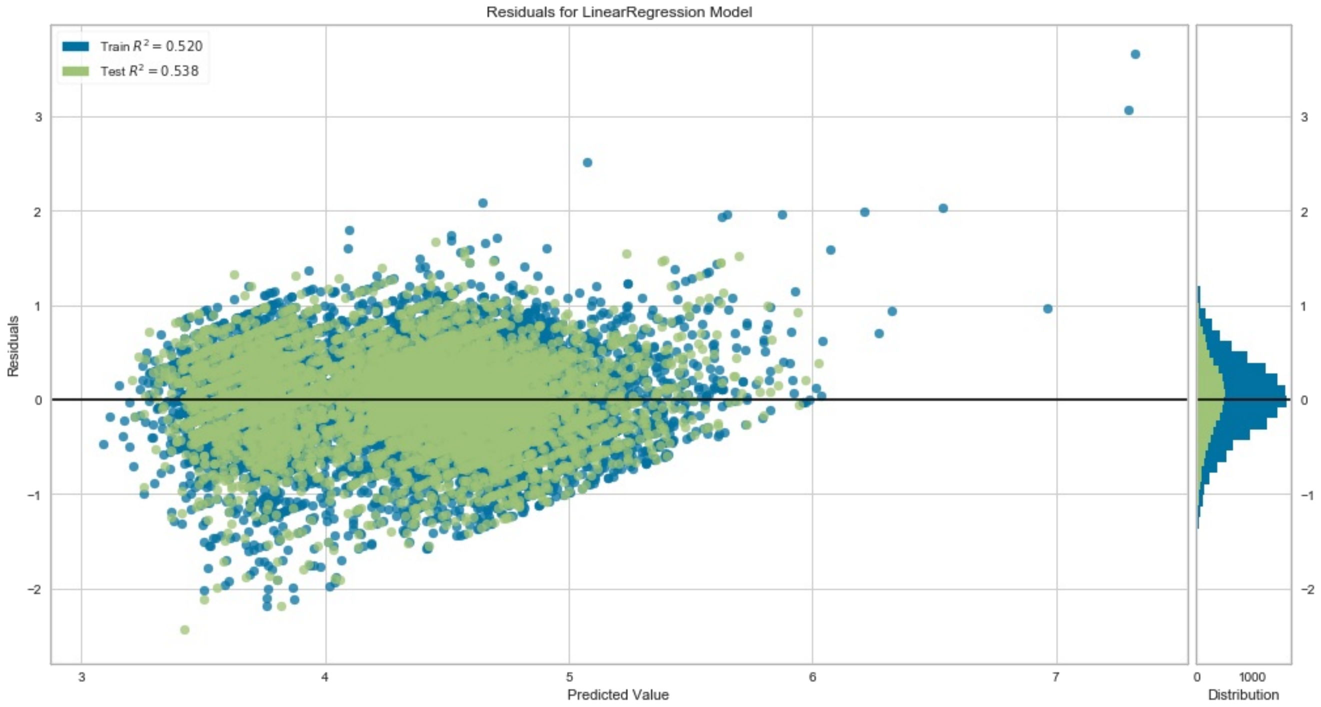
```
#6.1 Linear Regression

# Instantiate the linear model and visualizer

lm = LinearRegression()
visualizer = PredictionError(lm)

# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
visualizer.poof()
```



In [104]:

```
# Instantiate visualizer
visualizer = ResidualsPlot(lm)

# Fit the training data to the model
visualizer.fit(X_train, y_train)
```

```
# Evaluate the model on the test data
visualizer.score(X_test, y_test)
visualizer.poof()
```



In [105]:

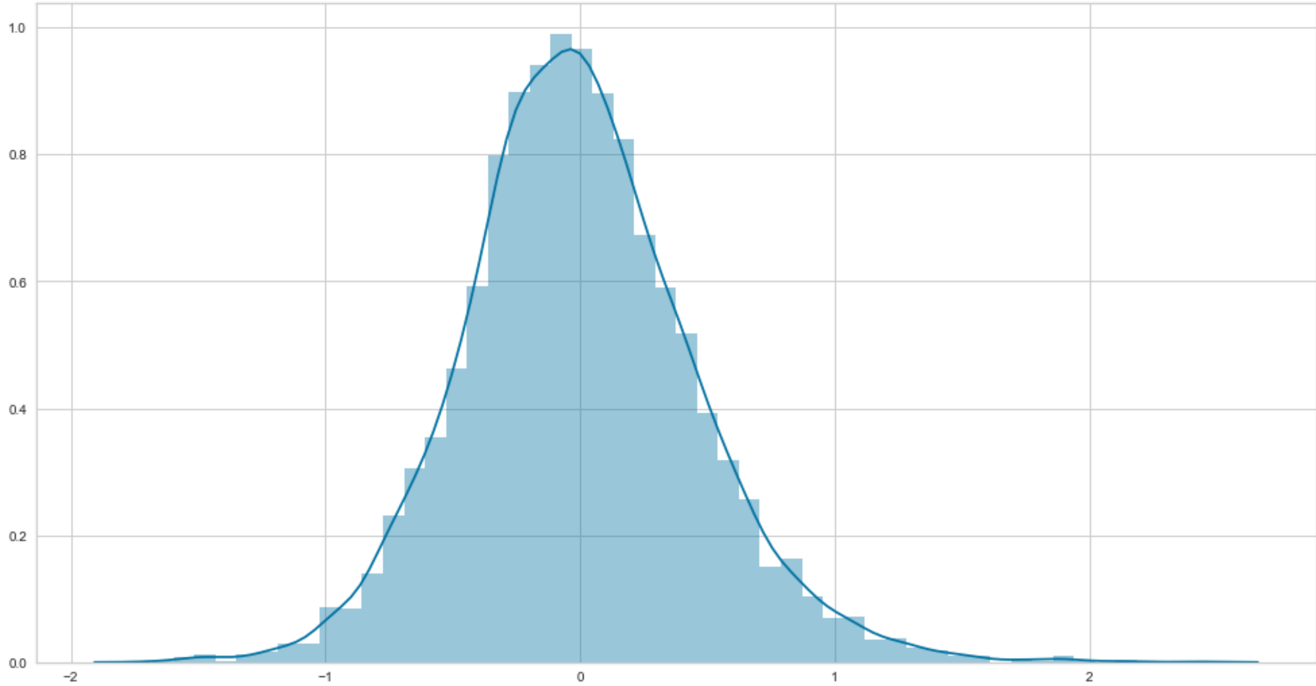
```
#Evaluating the Model
from sklearn import metrics
lm = LinearRegression()

#Train/fit lm on the training data
lm.fit(X_train,y_train)
lm.score(X_test,y_test)
predictions = lm.predict( X_test)
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
print('Coefficient of Determination:',metrics.r2_score(y_test, predictions))
print(lm.score(X_test, y_test))
a1 = y_test.ravel()
b1 = predictions.ravel()
dataf = pd.DataFrame({'Actual': a1, 'Predicted': b1})
print(dataf.head(20))
dataf = dataf.cumsum();
```

```
MAE: 0.3471407114152832
MSE: 0.20201163734592703
RMSE: 0.4494570472758515
variance: 0.5376198234970468
0.5376198234970468
      Actual  Predicted
0   5.855072   4.913738
1   4.553877   4.653816
2   4.094345   3.907751
3   3.401197   3.740622
4   4.276666   4.380818
5   4.499810   4.383582
6   3.610918   4.352306
7   4.499810   4.671262
8   5.293305   4.447974
9   5.129899   4.894703
10  3.912023   3.860715
11  4.934474   4.697241
12  4.248495   4.428812
13  3.610918   3.910122
14  4.828314   4.768890
15  5.521461   4.787944
```

```
16   4.584967   4.793336
17   4.356709   4.077844
18   4.382027   4.589578
19   3.931826   3.739603
```

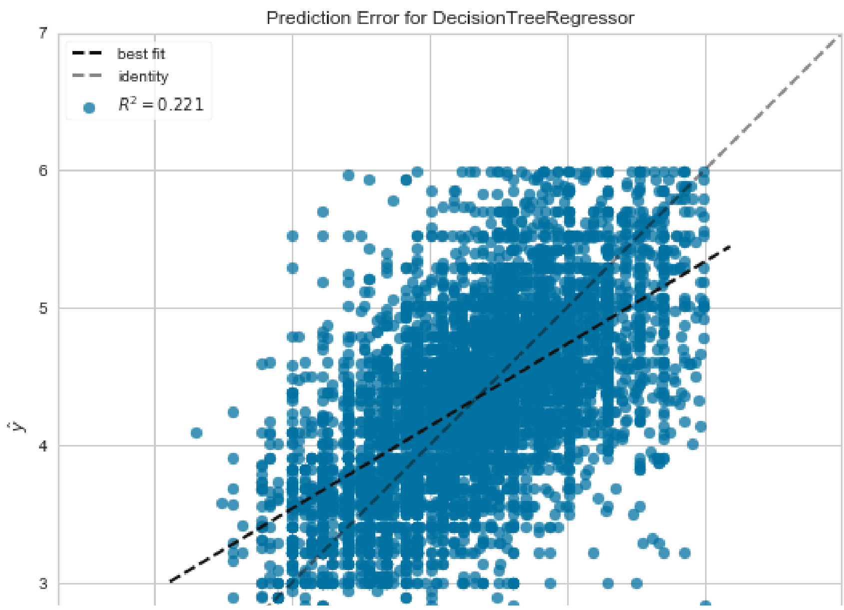
In [107]:

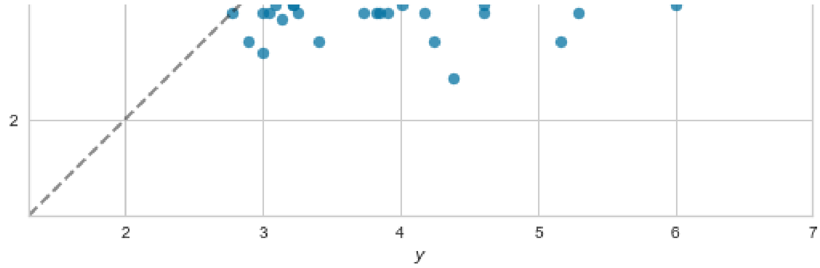
```python
#Plotting Residuals
sns.distplot((y_test-predictions),bins=50);
```



In [109]:

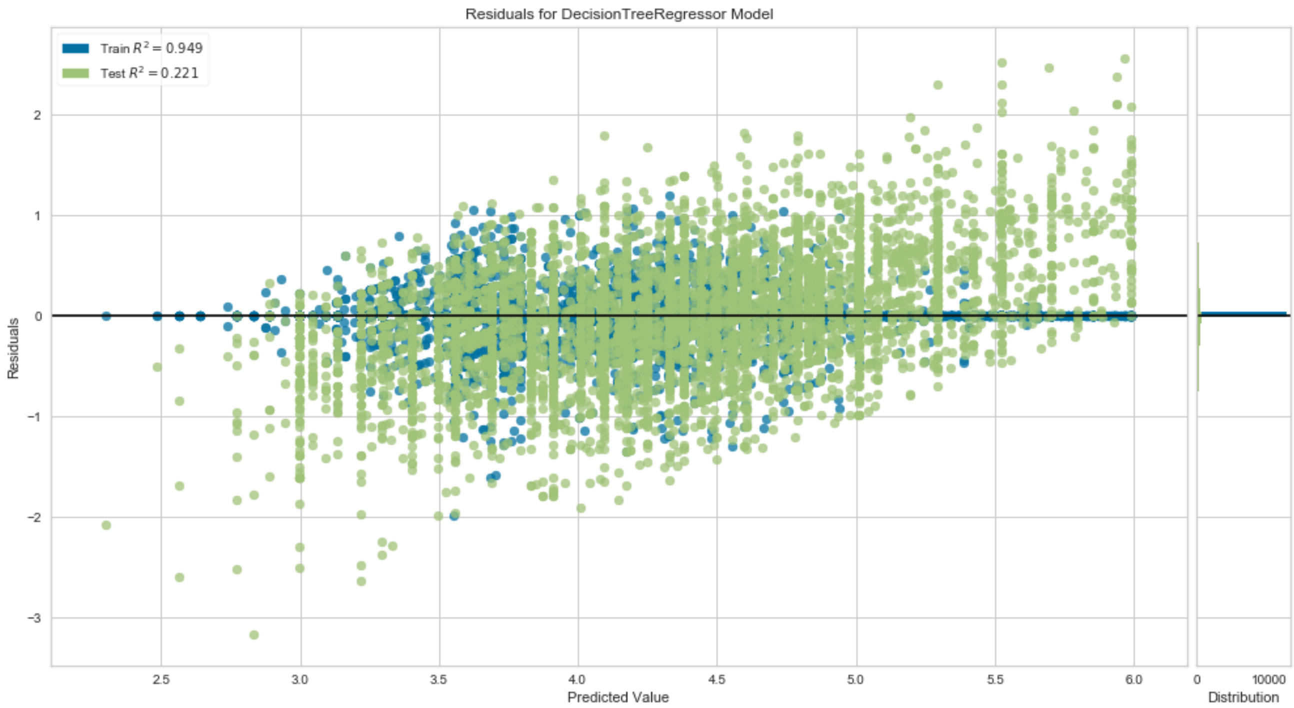
```python
#6.2 DecisionTree Regressor

# Instantiate the decision tree regressor model and visualizer
dt = DecisionTreeRegressor(random_state=0)
visualizer = PredictionError(dt)

# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
g = visualizer.poof()
```





In [110]:

```
# Instantiate the visualizer
visualizer = ResidualsPlot(dt)

# Fit the training data to the model
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
visualizer.poof()
```



In [111]:

```
#Evaluating the Decision Tree Regressor Model
dt_reg = DecisionTreeRegressor(random_state=0)
dt_reg.fit(X_train,y_train)
dt_reg.score(X_test,y_test)
dtr_pred= dt_reg.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, dtr_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, dtr_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,dtr_pred)))
print('Coefficient of Determination:',metrics.r2_score(y_test, dtr_pred))

a1 = y_test.ravel()
b1 = dtr_pred.ravel()
dataf = pd.DataFrame({'Actual': a1, 'Predicted': b1})
print(dataf.head(20))
dataf = dataf.cumsum();
```

```
Mean Absolute Error: 0.4421603620569738
Mean Squared Error: 0.3402545361699368
Root Mean Squared Error: 0.5833134116150055
variance: 0.2211985677796503
      Actual  Predicted
```

	Actual	Predicted
0	5.855072	5.796058
1	4.553877	4.418841
2	4.094345	3.448852
3	3.401197	4.077537
4	4.276666	4.325088
5	4.499810	3.828641
6	3.610918	4.619763
7	4.499810	4.382027
8	5.293305	5.991465
9	5.129899	5.164786
10	3.912023	4.382027
11	4.934474	5.480639
12	4.248495	4.454347
13	3.610918	4.290459
14	4.828314	4.867534
15	5.521461	4.174387
16	4.584967	4.779123
17	4.356709	4.553877
18	4.382027	4.828314
19	3.931826	3.401197

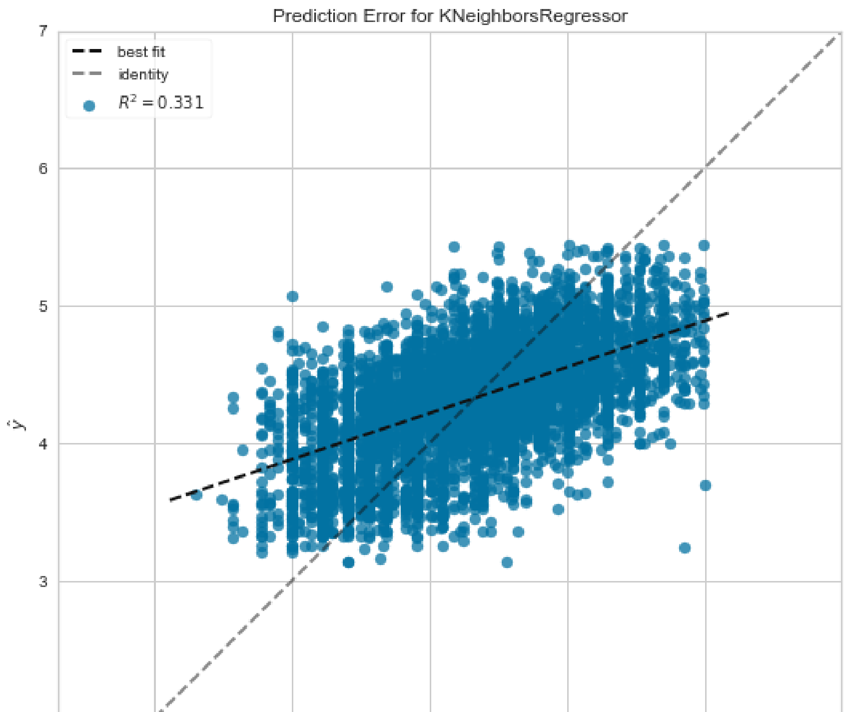
In [113]:

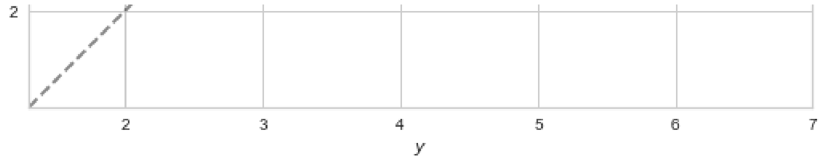
```
# Calculate mean absolute percentage error (MAPE)
errors = abs(dtr_pred - y_test)
mape = 100 * (errors / y_test)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Accuracy: 89.57 %.
```

In [115]:

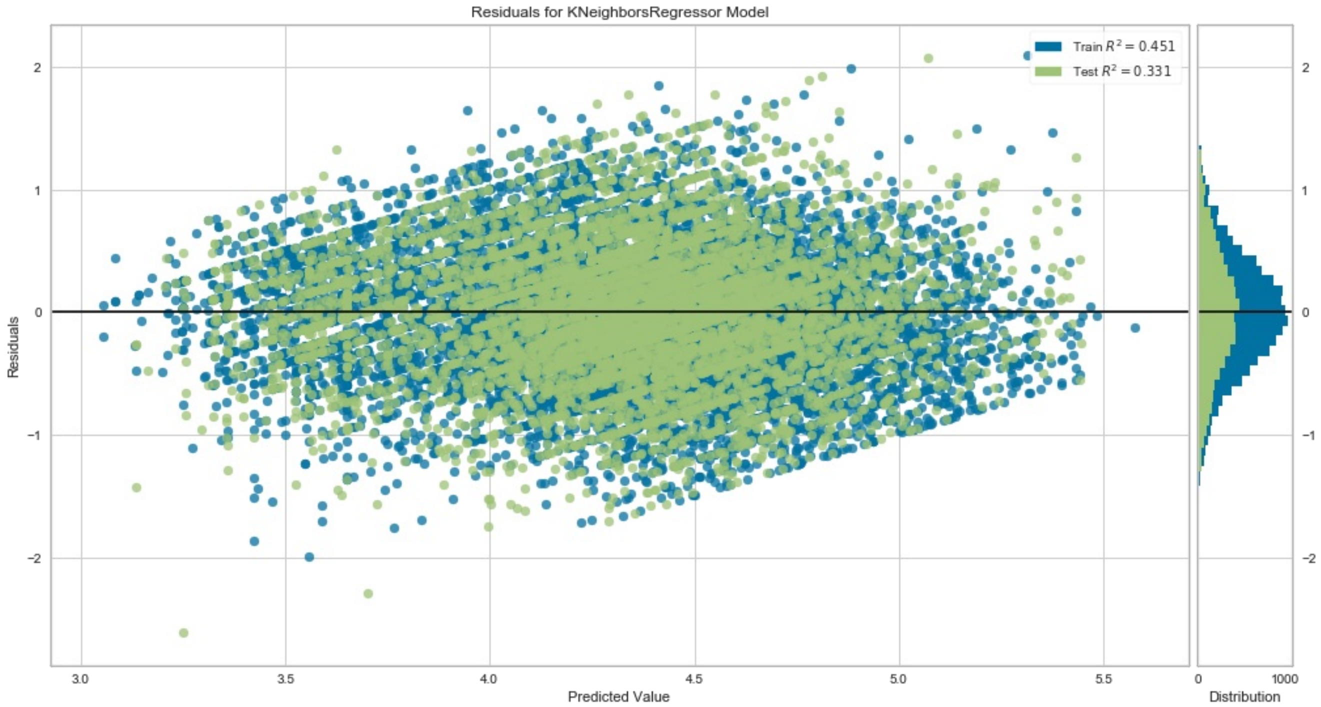
```
#6.3 KNN Regressor

# Instantiate the knn regressor model and visualizer
kn = KNeighborsRegressor(n_neighbors=10)
visualizer = PredictionError(kn)

# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
g = visualizer.poof()
```





In [116]:

```
# Instantiate the visualizer
visualizer = ResidualsPlot(kn)

# Fit the training data to the model
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
visualizer.poof()
```



In [117]:

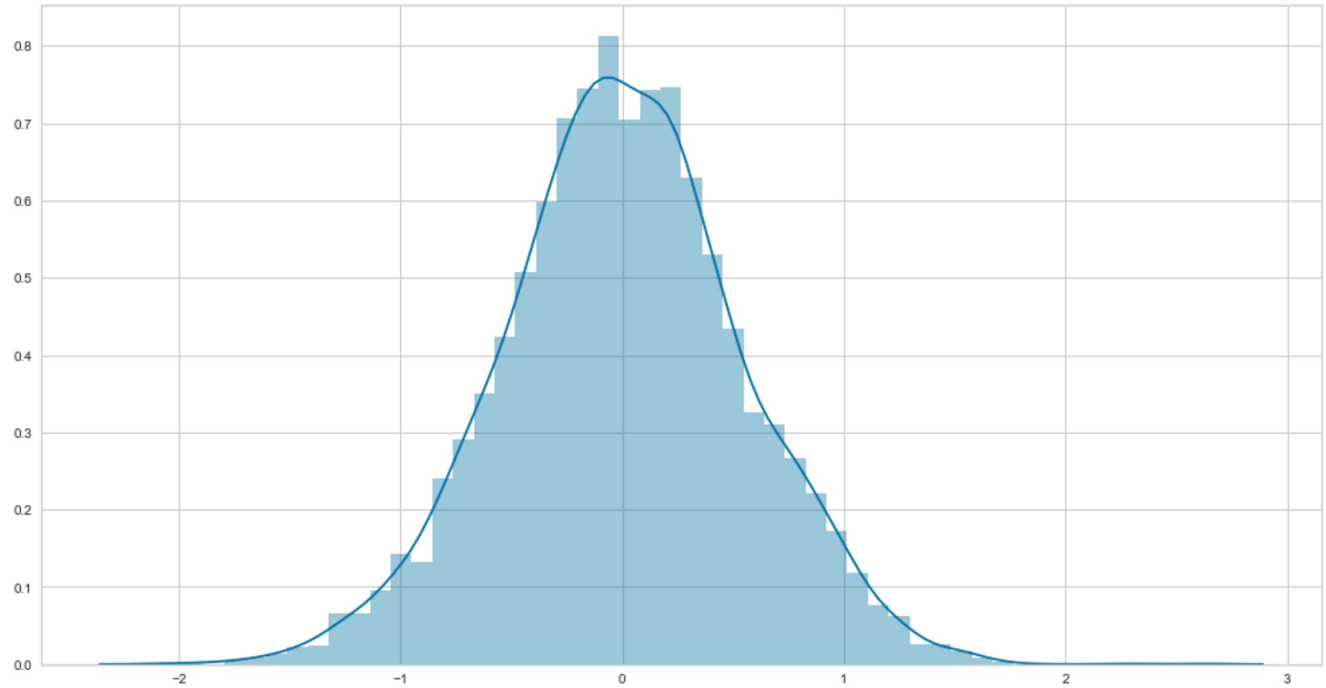
```
#Evaluating the KNN Regressor Model

knn = KNeighborsRegressor(n_neighbors=10)
knn.fit(X_train,y_train)
knn_pred= knn.predict(X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, knn_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, knn_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, knn_pred)))
print ('Coefficient of Determination:',metrics.r2_score(y_test, knn_pred))

a1 = y_test.ravel()
b1 = knn_pred.ravel()
dataf = pd.DataFrame({'Actual': a1, 'Predicted': b1})
print(dataf.head(20))
dataf = dataf.cumsum();
```

```
Mean Absolute Error: 0.42617254192126436
Mean Squared Error: 0.2923055070894922
Root Mean Squared Error: 0.5406528526600893
variance: 0.330948089246059
      Actual  Predicted
0   5.855072   4.873812
1   4.553877   4.285223
2   4.094345   3.529846
3   3.401197   3.964794
4   4.276666   4.313525
```

```
5   4.499810   4.221063
6   3.610918   4.607154
7   4.499810   4.710172
8   5.293305   4.303824
9   5.129899   4.467458
10  3.912023   4.361054
11  4.934474   4.705633
12  4.248495   4.045417
13  3.610918   4.555935
14  4.828314   4.744330
15  5.521461   4.695916
16  4.584967   4.716412
17  4.356709   4.128210
18  4.382027   4.154269
19  3.931826   4.182050
```

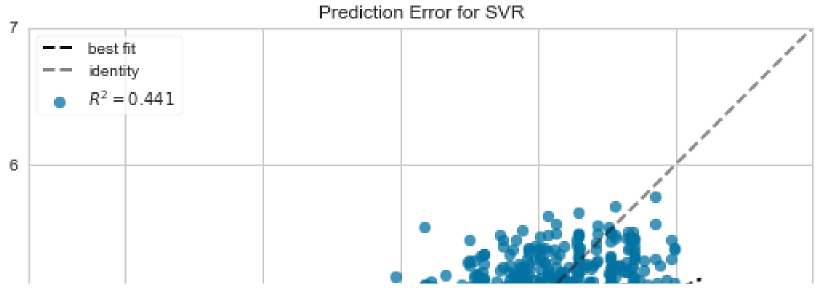
In [69]:

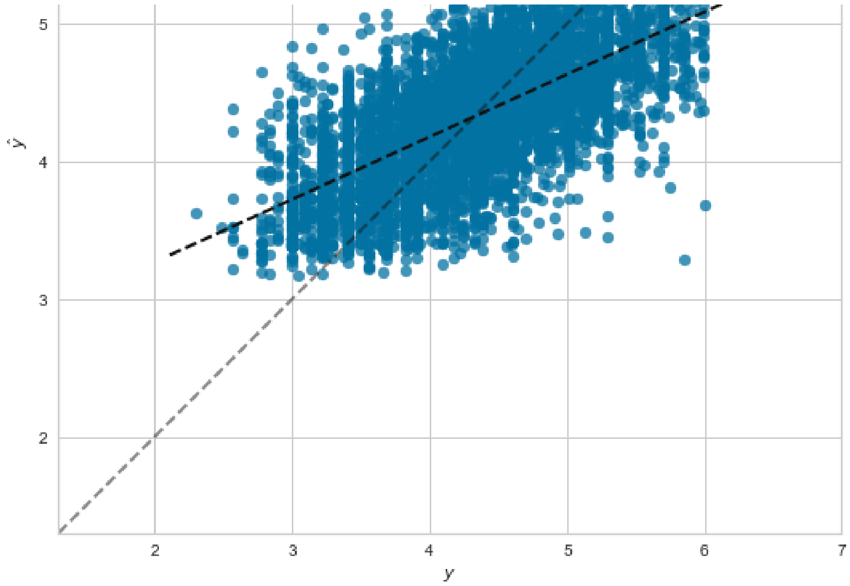
```python
#Plotting Residuals
sns.distplot((y_test-knn_pred),bins=50);
```



In [119]:

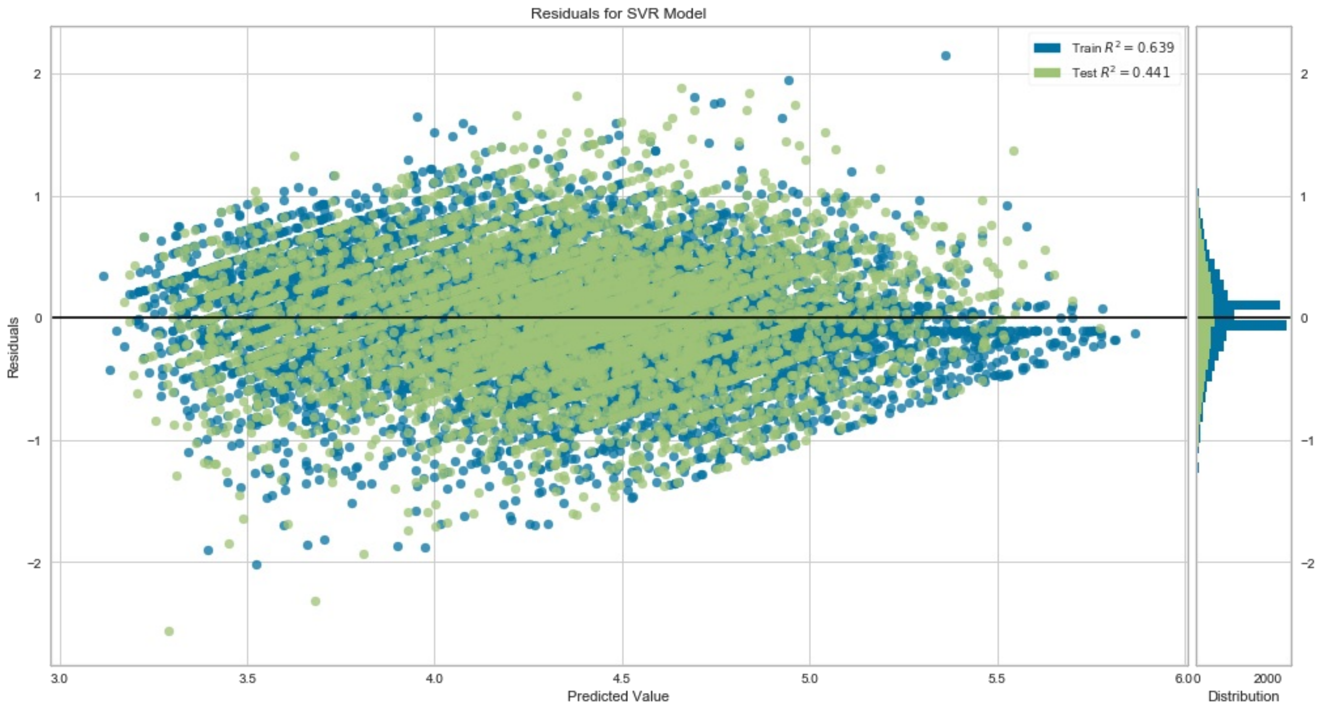
```python
#6.4 SVM Regression

# Instantiate the svr model and visualizer
svr = SVR(gamma='auto')
visualizer = PredictionError(svr)

# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
g = visualizer.poof()
```





In [120]:

```
# Instantiate the visualizer
visualizer = ResidualsPlot(svr)

# Fit the training data to the model
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
visualizer.poof()
```



In [121]:

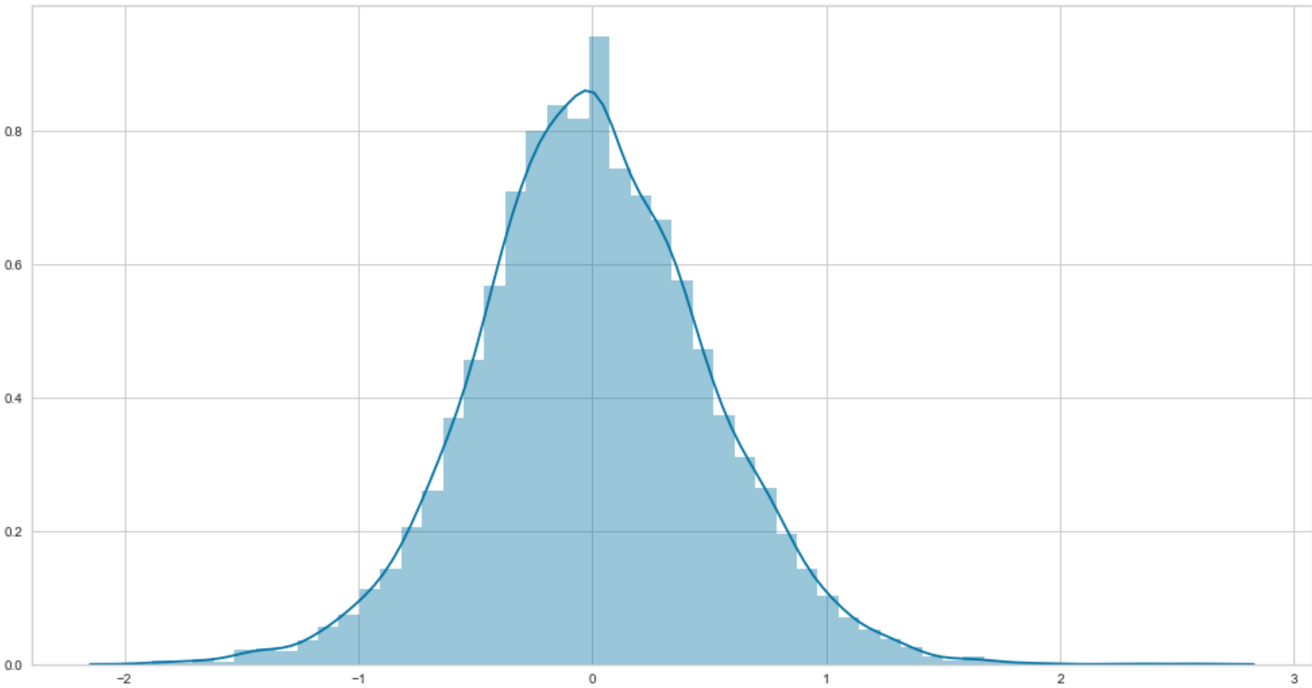
```
#Evaluate the SVM Regressor
svr_reg = SVR(gamma='auto')
svr_reg.fit(X_train, y_train)
svr_pred= svr_reg.predict(X_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, svr_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, svr_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, svr_pred)))
print ('Coefficient of Determination:',metrics.r2_score(y_test, svr_pred))

a1 = y_test.ravel()
```

```
a1 = y_test.ravel()
b1 = svr_pred.ravel()
dataf = pd.DataFrame({'Actual': a1, 'Predicted': b1})
print(dataf.head(20))
dataf = dataf.cumsum();
```

```
Mean Absolute Error: 0.38604173169912465
Mean Squared Error: 0.2443872659829381
Root Mean Squared Error: 0.49435540452486015
variance: 0.44062714076831666
      Actual  Predicted
0   5.855072   4.870401
1   4.553877   4.173498
2   4.094345   3.747557
3   3.401197   3.755990
4   4.276666   4.282114
5   4.499810   4.347855
6   3.610918   4.213845
7   4.499810   4.506866
8   5.293305   4.485104
9   5.129899   4.373920
10  3.912023   3.915586
11  4.934474   4.708553
12  4.248495   3.985521
13  3.610918   4.295458
14  4.828314   4.599557
15  5.521461   4.665991
16  4.584967   4.845876
17  4.356709   4.034329
18  4.382027   4.529775
19  3.931826   3.905168
```

In [76]:

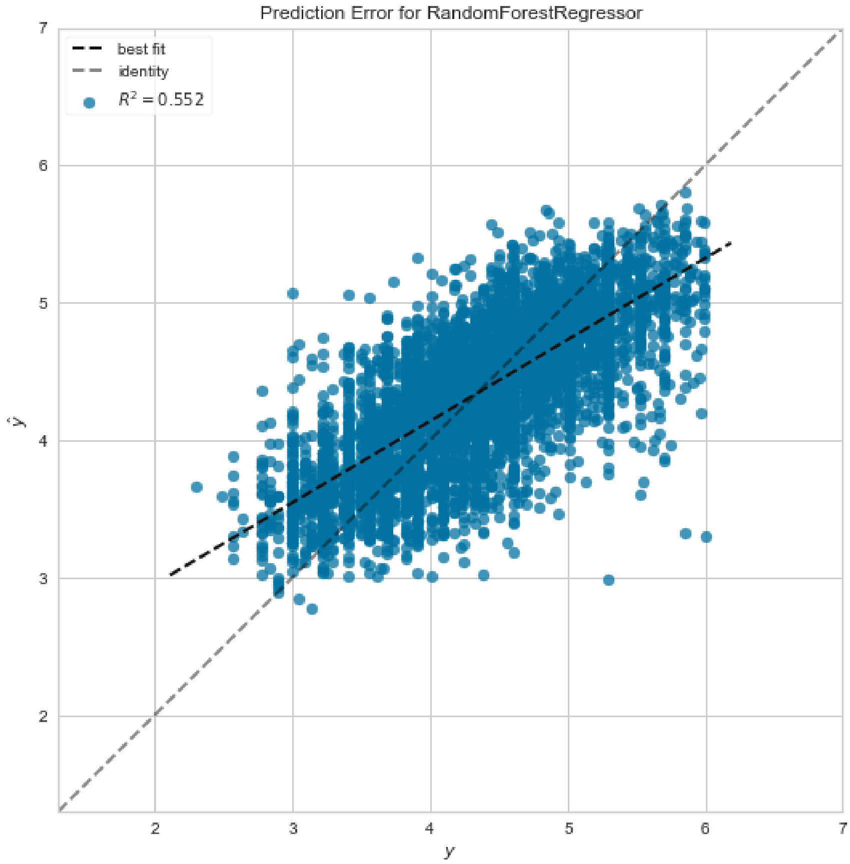
```
#Plotting Residuals
plt.sca
sns.distplot((y_test-svr_pred),bins=50);
```



In [78]:

```
#6.5 Random Forest Regression

# Instantiate the Random Forest Regression model and visualizer
rn = RandomForestRegressor(random_state=0,n_estimators=100)

visualizer = PredictionError(rn)

# Fit the training data to the visualizer
```

```
# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
g = visualizer.poof()
```



In [79]:

```
#Instantiate the visualizer
visualizer = ResidualsPlot(rn)

# Fit the training data to the model
visualizer.fit(X_train, y_train)

# Evaluate the model on the test data
visualizer.score(X_test, y_test)
visualizer.poof()
```

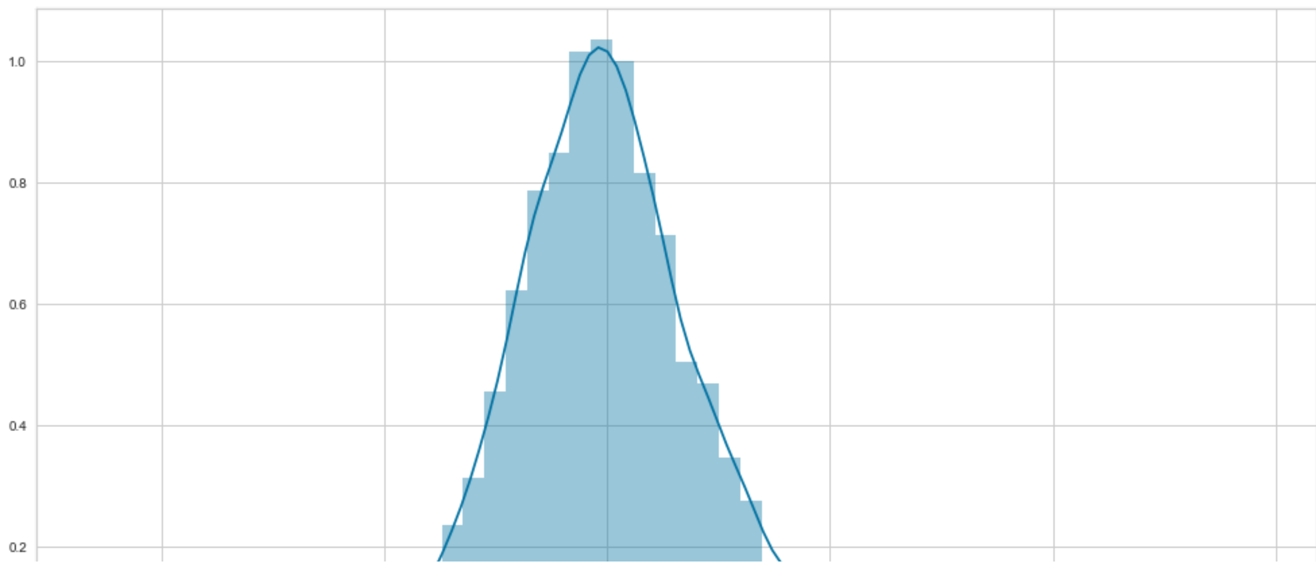
In [80]:

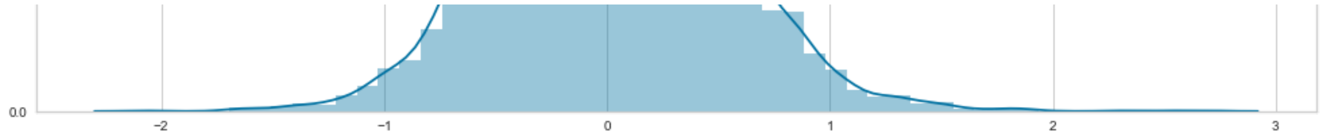
```
#Evaluating the Model
regr = RandomForestRegressor(random_state=0,n_estimators=100)
regr.fit(X_train,y_train)
rfr_pred= regr.predict(X_test)
print('MAE is ' , metrics.mean_absolute_error(rfr_pred,y_test))
print ('Coefficient of Determination:',metrics.r2_score(y_test, rfr_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, rfr_pred)))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, rfr_pred))

a1 = y_test.ravel()
b1 = rfr_pred.ravel()
dataf = pd.DataFrame({'Actual': a1, 'Predicted': b1})
print(dataf.head(20))
dataf = dataf.cumsum();
```

```
MAE is  0.33864798192778517
variance: 0.5523990507952058
Root Mean Squared Error: 0.44221563205419245
Mean Squared Error: 0.19555466523308893
      Actual  Predicted
0   5.855072   5.500489
1   4.553877   4.538150
2   4.094345   3.765531
3   3.401197   3.749079
4   4.276666   4.325088
5   4.499810   4.258385
6   3.610918   4.582330
7   4.499810   4.439576
8   5.293305   4.877691
9   5.129899   4.590483
10  3.912023   3.807678
11  4.934474   5.273979
12  4.248495   4.695987
13  3.610918   3.988805
14  4.828314   4.776520
15  5.521461   4.870984
16  4.584967   4.886310
17  4.356709   4.226966
18  4.382027   4.623901
19  3.931826   3.669725
```

In [82]:

```
#Plotting Residuals
plt.sca
sns.distplot((y_test-rfr_pred),bins=50);
```





In [122]:

```
#6.6 To Summarize Mean Squared Error For All the Models
mse=[]
Regressors=['Linear Regression','Decision Tree','KNN','SVM','Random Forest Regressor']
models=[LinearRegression(),DecisionTreeRegressor(),KNeighborsRegressor(n_neighbors=10),SVR(gamma='a
uto'),RandomForestRegressor(random_state=0,n_estimators=100)]
for i in models:
    model = i
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    mse.append(metrics.mean_squared_error(y_test,y_pred))
models_dataframe=pd.DataFrame(mse,index=Regressors)
models_dataframe.columns=['MSE']
models_dataframe
```

Out[122]:

	MSE
Linear Regression	0.202012
Decision Tree	0.343124
KNN	0.292306
SVM	0.244387
Random Forest Regressor	0.195555

In [180]:

```
#6.7 To Summarize variance For All the Models
var=[]
Regressors=['Linear Regression','Decision Tree','KNN','SVM','Random Forest Regressor']
models=[LinearRegression(),DecisionTreeRegressor(),KNeighborsRegressor(n_neighbors=10),SVR(gamma='a
uto'),RandomForestRegressor(random_state=0,n_estimators=100)]
for i in models:
    model = i
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    var.append(metrics.r2_score(y_test,y_pred))
models_dataframe=pd.DataFrame(var,index=Regressors)
models_dataframe.columns=['R2 score']
models_dataframe
```

Out[180]:

	R2 score
Linear Regression	0.537620
Decision Tree	0.223208
KNN	0.330948
SVM	0.440627
Random Forest Regressor	0.552399

In [88]:

```
#7 Cross Validation

#7.1 Cross Validation for linear Regression Model
X = newdataset.iloc[: , : -1].values
y = newdataset.iloc[: , -1].values
```

In [137]:

```
#7.1 Cross validation for Linear regression
from sklearn.model_selection import cross_val_score
lm = LinearRegression()

scores = cross_val_score(lm, X, y, cv= 8)
print('Accuracy of every fold in cross validation:', abs(scores))
print('Mean of the validation score:', abs(scores.mean()))
Mscores = cross_val_score(lm, X, y, cv=8, scoring='neg_mean_squared_error')
print('MSE of every fold in cross validation:', -Mscores)
print('MEan of MSE:', -Mscores.mean())
```

```
Accuracy of every fold in cross validation: [0.541273   0.5552823  0.49207933 0.49876206 0.54336625
0.44165461
 0.54723699 0.45754093]
Mean of the validation score: 0.509649433214036
MSE of every fold in cross validation: [0.1656275  0.19582438 0.21754662 0.21734937 0.22528553 0.22
479446
 0.19408048 0.21141776]
MEan of MSE: 0.20649076333079674
```

In [139]:

```
#7.2 Cross validation for Random Forest

rf = RandomForestRegressor(random_state=0,n_estimators=100)

scores = cross_val_score(rf, X, y, cv= 8)
print('Accuracy of every fold in cross validation:', abs(scores))
print('Mean of the validation score:', abs(scores.mean()))
Mscores = cross_val_score(rf, X, y, cv=8, scoring='neg_mean_squared_error')
print('MSE of every fold in cross validation:', -Mscores)
print('MEan of MSE:', -Mscores.mean())
```

```
Accuracy of every fold in cross validation: [0.49622187 0.54451644 0.49740895 0.50407984
0.57116561 0.44638317
 0.52863207 0.4512711 ]
Mean of the validation score: 0.5049598821910983
MSE of every fold in cross validation: [0.18189362 0.20056495 0.2152639  0.21504345 0.21157039
0.2228907
 0.20205563 0.21386136]
MEan of MSE: 0.20789300058643437
```

In [140]:

```
#7.3 Cross validation for KNN

kn = KNeighborsRegressor(n_neighbors=10)

scores = cross_val_score(kn, X, y, cv= 8)
print('Accuracy of every fold in cross validation:', abs(scores))
print('Mean of the validation score:', abs(scores.mean()))
Mscores = cross_val_score(kn, X, y, cv=8, scoring='neg_mean_squared_error')
print('MSE of every fold in cross validation:', -Mscores)
print('MEan of MSE:', -Mscores.mean())
```

```
Accuracy of every fold in cross validation: [0.23424189 0.41427133 0.32652453 0.32164137
0.34742112 0.25896711
 0.32653728 0.23924856]
Mean of the validation score: 0.3086066497661693
MSE of every fold in cross validation: [0.27648385 0.25791632 0.2884551  0.29415335 0.32195732
0.29834596
 0.28868518 0.29649494]
MEan of MSE: 0.29031150090585645
```

In [141]:

```
#7.4 Cross validation for Decision tree

dt = DecisionTreeRegressor(random_state=10)
```

```
scores = cross_val_score(dt, X, y, cv= 8)
print('Accuracy of every fold in cross validation:', abs(scores))
print('Mean of the validation score:', abs(scores.mean()))
Mscores = cross_val_score(dt, X, y, cv=8, scoring='neg_mean_squared_error')
print('MSE of every fold in cross validation:', -Mscores)
print('MEan of MSE:', -Mscores.mean())
```

```
Accuracy of every fold in cross validation: [0.11881243 0.26581176 0.20538147 0.15972736
0.23813867 0.05930223
 0.11035235 0.07741884]
Mean of the validation score: 0.13501342888564677
MSE of every fold in cross validation: [0.31816069 0.32328813 0.34034167 0.36436334 0.37587308
0.37873269
 0.38135457 0.41991275]
MEan of MSE: 0.36275336558771537
```

In [142]:

```
#7.5 Cross validation for SVR

svr = SVR(gamma='auto')

scores = cross_val_score(svr, X, y, cv= 8)
print('Accuracy of every fold in cross validation:', abs(scores))
print('Mean of the validation score:', abs(scores.mean()))
Mscores = cross_val_score(svr, X, y, cv=8, scoring='neg_mean_squared_error')
print('MSE of every fold in cross validation:', -Mscores)
print('MEan of MSE:', -Mscores.mean())
```

```
Accuracy of every fold in cross validation: [0.37642864 0.50832369 0.42242273 0.41391611
0.45595343 0.3736427
 0.46118163 0.38469612]
Mean of the validation score: 0.42457063014450824
MSE of every fold in cross validation: [0.22514604 0.21650186 0.24738111 0.25414071 0.26841165
0.25217662
 0.2309688  0.23980827]
MEan of MSE: 0.24181688230041232
```

In [173]:

```
# 8.1  Analysis for checking hypothesis
#taking only 4 variables

X = newdataset[[
    'room_type_Entire home/apt',
    'accommodates',
    'beds', 'bedrooms']]
y = newdataset['price_log']
```

In [154]:

```
#8.2 Training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

In [175]:

```
print(X_train.shape)
```

```
(14350, 4)
```

In [150]:

```
#8.3 Linear Regression
#Evaluating the Model
from sklearn import metrics
lm = LinearRegression()

#Train/fit lm on the training data
lm.fit(X_train,y_train)
```

```
lm.score(X_test,y_test)
predictions = lm.predict( X_test)
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
print('Coefficient of Determination:',metrics.r2_score(y_test, predictions))
print(lm.score(X_test, y_test))
a1 = y_test.ravel()
b1 = predictions.ravel()
dataf = pd.DataFrame({'Actual': a1, 'Predicted': b1})
print(dataf.head(20))
dataf = dataf.cumsum();
```

```
MAE: 0.37925473947695626
MSE: 0.23711599410554182
RMSE: 0.48694557612277556
variance: 0.45727020162482657
0.4572702016248266
      Actual  Predicted
0   5.855072   4.389727
1   4.553877   4.380922
2   4.094345   3.748783
3   3.401197   3.748783
4   4.276666   4.389727
5   4.499810   4.389727
6   3.610918   4.389727
7   4.499810   4.291067
8   5.293305   4.366234
9   5.129899   5.002831
10  3.912023   3.748783
11  4.934474   4.621111
12  4.248495   4.464894
13  3.610918   3.748783
14  4.828314   4.389727
15  5.521461   4.777329
16  4.584967   4.771446
17  4.356709   4.282262
18  4.382027   4.389727
19  3.931826   3.748783
```

In [155]:

```
#8.4 random forest
#Evaluating the Model
regr = RandomForestRegressor(random_state=0,n_estimators=100)
regr.fit(X_train,y_train)
rfr_pred= regr.predict(X_test)
print('MAE is ' , metrics.mean_absolute_error(rfr_pred,y_test))
print ('Coefficient of Determination:',metrics.r2_score(y_test, rfr_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, rfr_pred)))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, rfr_pred))

a1 = y_test.ravel()
b1 = rfr_pred.ravel()
dataf = pd.DataFrame({'Actual': a1, 'Predicted': b1})
print(dataf.head(20))
dataf = dataf.cumsum();
```

```
MAE is  0.3759661171116998
variance: 0.4622530578709334
Root Mean Squared Error: 0.48470507381875266
Mean Squared Error: 0.23493900858564248
      Actual  Predicted
0   5.855072   4.396290
1   4.553877   4.303880
2   4.094345   3.748059
3   3.401197   3.748059
4   4.276666   4.396290
5   4.499810   4.396290
6   3.610918   4.396290
7   4.499810   4.233323
8   5.293305   4.370981
9   5.129899   4.951033
10  3.912023   3.748059
11  4.934474   4.661425
```

```
12  4.248495   4.480894
13  3.610918   3.748059
14  4.828314   4.396290
15  5.521461   4.865823
16  4.584967   4.849761
17  4.356709   4.102841
18  4.382027   4.396290
19  3.931826   3.748059
```

In [160]:

```
#8.4 Cross validation for Random Forest
rf = RandomForestRegressor(random_state=0,n_estimators=100)

scores = cross_val_score(rf, X, y, cv= 15)
print('Accuracy of every fold in cross validation:', abs(scores))
print('Mean of the validation score:', abs(scores.mean()))
Mscores = cross_val_score(rf, X, y, cv=15, scoring='neg_mean_squared_error')
print('MSE of every fold in cross validation:', -Mscores)
print('MEan of MSE:', -Mscores.mean())
```

```
Accuracy of every fold in cross validation: [0.3772647  0.50825255 0.46290356 0.52691106 0.4396414€
0.46216401
 0.44624371 0.40154996 0.5091735  0.45856015 0.30534585 0.42943197
 0.53280541 0.42354479 0.39510341]
Mean of the validation score: 0.4452597393303206
MSE of every fold in cross validation: [0.21245004 0.18656929 0.20607863 0.24069602 0.23305446
0.2365518
 0.22599977 0.27702896 0.25028158 0.22459061 0.28312606 0.22846859
 0.2063492  0.23000578 0.22755406]
MEan of MSE: 0.23125365639291853
```

In [161]:

```
#8.5 Cross validation for Linear Regression

lm = LinearRegression()

scores = cross_val_score(lm, X, y, cv= 15)
print('Accuracy of every fold in cross validation:', abs(scores))
print('Mean of the validation score:', abs(scores.mean()))
Mscores = cross_val_score(lm, X, y, cv=15, scoring='neg_mean_squared_error')
print('MSE of every fold in cross validation:', -Mscores)
print('MEan of MSE:', -Mscores.mean())
```

```
Accuracy of every fold in cross validation: [0.37918829 0.50532231 0.45802735 0.51200261
0.42054653 0.44833803
 0.43437954 0.37143517 0.49497771 0.44590961 0.31317018 0.41823587
 0.51037608 0.38472555 0.40128497]
Mean of the validation score: 0.433194654350315
MSE of every fold in cross validation: [0.2117938  0.18768102 0.20794958 0.24828108 0.24099608 0.24
263276
 0.23084179 0.29096942 0.25752028 0.22983808 0.27993703 0.23295176
 0.21625573 0.24549467 0.22522864]
MEan of MSE: 0.23655811469875593
```

In [ ]: