Artificial Intelligence and Data Science

UB Number: 21043952

Name: Odiaka Keziah Yemisi

Word Count-2926

Table of Contents

PART A - DELIVERY PERFORMANCE PROGRAM (DPP)

INTRODUCTION

Delivery performance is the extent to which the goods and services that a company delivers fulfil consumer expectations. Performance is gauged from the vendor who supplies goods end to the end user (Rao et al.2011). Given the case study, the focus here is on the drivers only from the time the order is picked up till it get to the customers who made the order.

PURPOSE OF DELIVERY PERFORMANCE PROGRAM (DPP)

Thus, the purpose of the Delivery performance Program (DPP) is the evaluation of the green drivers who deliver pizza hired by Bradford SPOT scheme. That is, measuring their total performance as regards to the delivery of the pizza and this could be based on using carbon efficient means of transportation, making deliveries on or before the estimated time of arrival, showing respect to customer order during transportation of the pizza and during delivery, and leading to the customer been happy by employing Key Performance Indicators that are measurable.

MEASUREMENT AND METRICS OF DELIVERY PERFORMANCE

The task is Smiley Pizza on Time (SPOT) is asking for a design of this Deliverer (Drivers) Performance Program and its implementation based on this**,** Key Performance Indicators (KPI's) would be employed to measure the company's deliverer performance based on the measurements listed above. The KPI's includes:

1. **On time delivery:**
   is one of the key ways employed by customers to access the delivery experience, this measure could also connote respects to the customer by showing up on time with the pizza as this is one of the responsibilities of the deliverers. This is calculated from the time the order is picked up to the time it gets to the customer. This can be monitored by having in place a clock in system for the drivers once order is picked up and a real time order tracking app recording the driver's movement. It is important to factor in conditions such as the distance to the delivery location, the hours in the day the delivery is made (absolute and relative time), the delivery volume per delivery, weather condition This is to determine if the DPP is meeting its goals for timely delivery or not and is also very key for customers satisfaction as regards delivery.

2. **Respect:**
   This is to determine the deliverers performance upon arrival and can only be measured based on feedback from the customer after delivery. This could include but not limited to been courteous, polite, and considerate (when asked for help) depending on the disability or peculiarity of the customer. Cultural awareness and mode of greeting also comes under this. This can only be measured when there is a complaint via feedbacks through the use of a rating system via the app after delivering with a promised points earned system if feedback is given, as this would encourage feedbacks which otherwise would only be given when there is a problem and thus affect data gathering as regards.

3. **Happiness of the customer as regards delivery**
   This relates to the customers satisfaction based on the delivery received and its related factors. This could cover deliveries coming in timely, been considerate to help challenged customers to drop off the delivery in a particular space, The state of the pizza when it comes in; not been smeared or destroyed, the cleanliness of the driver, cultural awareness, and mannerism. This is measured by the repeat orders associated with a particular deliverer, The use of Customer Satisfaction Score (CSAT) by asking "how satisfied they were with the delivery using numbers 1-5, where 5 connotes Very satisfied and 1 not satisfied and then followed using an open-ended question to explain why they gave a particular score. To simplify the use of feedback and ratings.

4. **Green delivery:**
   This considers the use of eco-friendly modes and energy efficient mediums for transportation of pizzas during deliveries (electric bikes and cars). This can be measured by use of carbon accounting emissions tracking devices around the deliverer's vehicle or bikes per delivery; this measures, keeps tracks aids reduction in the SPOT deliverer's environmental footprint across board. The measure is both the direct (fuels and vehicle types) and indirect emissions (electricity, travel distance, heating, and cooling).

Metrics	Indicators	Measuring tools
On Time Delivery	• Pick up time • delivery distance • weather condition • routes to be followed • size of the delivery • mode of delivery • Number of complaints • Delivery time taken.	• GPS tracking app • Clock in system
Respect	• Politeness • Courteousness • consideration.	• Number of complaints • Feedback • ratings
Happiness	• Quick delivery • Order in good shape at delivery • Courtesy • Cultural awareness	• Feedback surveys • Customer Satisfaction Score (CSAT)
Green delivery	• Bike/Vehicle used for delivery	mission measurement sensors/ software e.g. CO2 AI

Table 1: Metrics table

DATA COLLECTION

Data is very vital to building and implementation of the SPOT Deliverer Performance Program. They include data from three main sources which are

1. **The SPOT Restaurant:** The data needed from this source are the drivers pick up time, number of orders per delivery, repeat orders, state of the pizza at pick up, weight of the order, name and distance per delivery, special needs per delivery, number of comments on the app, Delivery time fulfilment, complaints.

2. **The Customers:** order accuracy, address, politeness, mannerism, cleanliness, consideration, state of the pizza at delivery

3. **The Sensors and Software of the green drivers:**
   time per deliver, CO2 emissions per delivery, delivery routes, distance covered, predicted and actual delivery time, energy used up per delivery.

4. **Drivers:** Cost of delivery, expected delivery time, deliverers history.

For this model, it is important to work with large datapoints, to get the best representation of the sample size to the entire population to be able to make improved decisions, gain insights into customers behaviour, Increase customer satisfaction (Almeida 2017).

METHOD OF DATA COLLECTION AND SPECIAL DEVICE REQUIREMENT

- **From the restaurant** -use of a digital clock in system for drivers at pickup, delivery error rate. Also, a Central dashboard with information as regards repeat purchase rate per customer
- **From the customer** -we would use Customers feedback Survey from rating the deliverers with a 5 star from 1 star meaning poor delivery and 5 star denoting excellent delivery which leads to another page with pictorial badges (Pizza was not delivered, missed timeslot, Deliverer did not knock, Left pizza in unsafe place, not handled with care, was not professional options)to know what the feedback is related to and an on open ended question slot to leave feedback, customers feedback can be also be gotten through surveys .
- **Drivers-** Goggle Map application programming interface, GPS and Sensors for drivers footprint, sensors monitoring emissions, electric proof of delivery and picture of where delivery was dropped off if not handed directly to customer, routing and scheduling software (Circuits for Teams) for drivers this helps to optimize the delivery routes for various drivers to aid a smooth delivery process. This shows the fastest possible routes that covers all delivery stops for each address and keeps recipients posted with ETA notifications with real time updates that the customers can use to also to give an electric proof of delivery, possibility of arranging packages ahead of time to aid easy access to deliveries and automation of routine delivery task (Aleixo 2021).

MACHINE LEARNING ALGORITM AND JUSTIFICATION

Supervised and unsupervised learning would be employed. For, Supervised Machine Learning; classification and regression would be employed. Classification is used as labelled historical data is what would be used to evaluate the DPP, and the target variable is known and would be classified as “on- time delivery “and “late delivery”. There are well labelled data that can be employed in the prediction of outcomes. Classification can be employed in the prediction of delivery time as patterns are identified with room for accurate predictions, but we must get a large sample size to be able to train the data and sentiment analysis to identify deliverer reviews posted on the app. classification could employ the use of labels for customers if the delivery is good, bad, or terrible. Regression model to predict delivery how good the deliverer was based on time and happiness, respect, green delivery. While unsupervised learning is employed to evaluate customers reviews and data; to help analyse and form clusters from unlabelled data.

EVALUATION OF SPOT DELIVERER PERFORMANCE SYSTEM

The DPP system would be evaluated using the confusion matrix to help in providing insights into the accuracy of predictions made by a classification model. First, the target variable would be classed as “On-time Delivery” and “Late Delivery”. After which data would be gathered based on past deliveries made, deliveries outcome (Happy, respectful, on-time) and the estimated delivery times for orders raised. The Data is then trained and can predict new deliveries if they would be late, or on-time based on the data gathered and contingent upon other factors. After which the data is divided into the train (70%) to train and test (30%) to evaluate its performance and then apply the trained model to the test set to generate predictions for each delivery based on Respect+ Happiness + Green +Time+ Cleanliness. The predicted values are then compared with the actual outcome to create our confusion matrix put in four quadrants; which are True Positive which is when the model predicted the delivery would be on time and it was; True Negative is where it was predicted a delivery would be late and it was late, False Positive when we predicted a delivery would be on time but it came in late and lastly, False Negative where the model predicted a delivery would be late and it came in on time. After which model evaluation is carried out using evaluation metrics such as Accuracy, Recall, Precision, F1 score and this is interpreted, a higher score of the following indicates a system that is preforming well. The confusion matrix thus helps us to evaluate the DPP system and points out areas for improvement. After which the model is deployed.

POSSIBLE IMPEDIMENTS TO DPP IMPLEMENTATION

1. Cost of carrying out data analysis and installation various tools needed for the new initiative, and sensitization of deliverers.
2. There is the danger of finding drivers that are interested in green modes of transportation and getting deliverers that use energy efficient means of transport.
3. Software or app malfunctioning which can impact data gathering
4. Sufficient data gathering problem since most customers would only give feedback when there is an issue except there are incentives leading to data bias.

SOLUTIONS

1. Need to train and sensitize drivers on the new tools and advantages that could accrue from the new system implementation.
2. Having a backup software tool for each data points that are integrated and can easily be used when a software is experiencing downtime.
3. Use of incentives which could be points based and can be redeemable to get pizza at a capped number of reviews

CONCLUSION

To conclude, the above system design is necessary to improve the current performance of the Deliverer Performance Program

PART B: REGRESSION ANALYSIS OF HOUSING PRICES USING PYTHON

INTRODUCTION

This report is used to discuss housing features, its importance and pricing. This report is useful for both house buyers and sellers in helping to make informed decisions on house prices and factors that influences the price of houses.

OBJECTIVES

The objectives are:

To show the feature importance of each feature in the model
To predict house prices in the provided test set and its estimated values.

DATA DESCRIPTION

The Dataset comprises of 3000 rows for the train Data and 999 rows for the test data and 12 columns (11 predictors with 1 target variable). The following variables number of room, French doors, kitchens, and bathrooms were continuous while the other variables were employing binary features of 0 and 1, where 0 represented no and 1 represented yes. There was no null value in the dataset given.

DESCRIPTIVE ANALYSIS

Based on the given data set, the target feature shows a normal distribution with a mean of 8606 which is the average and a median of 8615 (50^{th} percentile). While the maximum value is 15035 and the minimum price is 2235 showing the presence of some outliners.

METHODOLOGY

For this study, Python was used, and three machine algorithm was employed to compare Machine Learning techniques was employed which are the Liner Regression (LR), Random Forest (RF)and K-Nearest Neighbour (KNN) to evaluate and compare the performance of different Machine Learning techniques in predicting the price of houses and to determine which technique is most effective in the predicting result. It is important to state that they all do same thing but different modelling method which is for the purpose of a comparative and robust analysis, these 3 were employed.

Linear Regression seeks to address the relationship and dependencies that exists between a target variable and the features with the aim of predicting a continuous target variable (Nasteski 2017).

Random Forest (RF) is a combination of a series of tree classifiers with each tree casting a vote. RF helps to address overfitting, noise, and outliners (Liu et al.2012).

K-Nearest Neighbour refers to an algorithm tool which is used to classify observations that are not labelled based on patterns that are nearest to the pattern of the target to deliver necessary information about the labels (Kramer 2013; Zhang 2016).

MODEL PERFORMANCE

MAE and R2 was used to test for the model performance. A higher R2 value indicates a better fit for the model as it shows the level of variance of the target variable explained by that of the predictors. By comparing the MAE and the R2. The lower the MAE, while the better the ML technique, the higher the R2 the better (closer to 1).

MAE	**LR**	**RF**	**KNN**
RESULT	13	172	471
RANK	1^{ST}	2^{ND}	3^{RD}

Table 2: Mean Absolute Error table

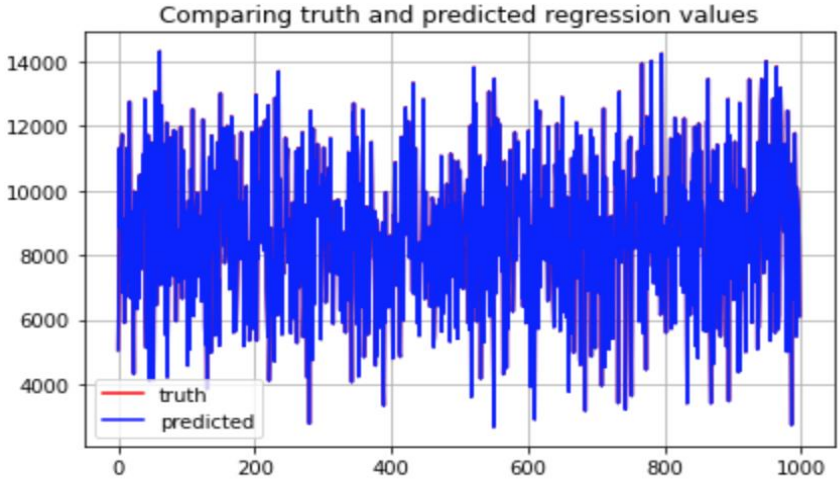
R2	LMR	RF	KNN
RESULT1	1	0.998	0.947
RANK	1ST	2ND	3RD

Table 3: R2 Score Table

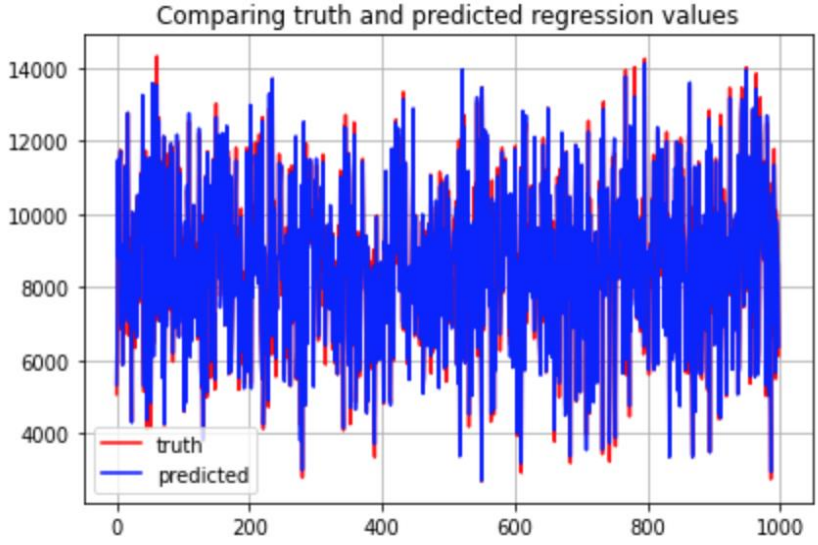
Information provided in the figures 1 and 2 reveals the values between the truth and predicted regression values of the two selected algorithms. The figures show that the values are near perfect in Linear Regression Model and closer in RF while KNN had a huge and very noticeable distance. Likewise, the Mean Absolute Error (MAE) score shows the absolute difference between the predicted and actual values been averaged across the dataset; MAE was 13 for Linear Regression while 173 for RF, and 471 for KNN indicates that the magnitude of difference between the predicted and true value of the observation; lowest in LR, followed by RF and lastly KNN. In view of this, LR is said to provide a better result for predicting house prices given the data set as the closer the value of MAE to 0, the better Furthermore, the value of the R-squared which shows the goodness of fit (relationship) between the dependent and independent variables. The R2 of 1, 0.998 and 0.947 for LR, RF and KNN respectively also further boosts the earlier standpoint. LR had a perfect fit, while RF and KNN had a high R2 both are also reliable as they are high but not perfect.

The figures below were used to show the difference in the truth and predicted regression values.

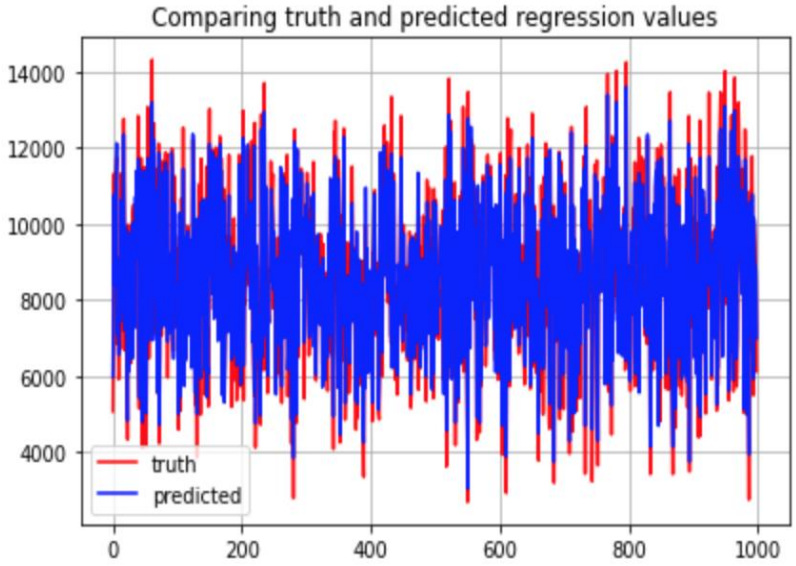
Visualisation of Truth and predicted Regression values



Linear Regression



Random Forest



K-Nearest Neighbour

The diagrams above further prove the differences that exists between the truth and predicted values by the various algorithms employed and further proves the performance of each model.

RESULT AND INTERPRETATION

Most frequently, feature importance is used to explain classification models (Bhatt et al. 2020). Feature importance as a measure of each relevant variable unique contribution to a specific classifier, gauges how much a feature affects the outcome of a model's prediction (Casalicchio et al. 2019; Ho 2022). The knowledge of the effects of each feature in a model helps in understanding the model better (Ho 2022). The study employs the use of Linear Regression and Random Forest to show the importance of each feature
Using Linear Regression, the feature importance is based on the coefficient of each variable, the most important feature is for the house to be furnished with a coefficient of 2000, then have a woodfloor with 1890. A table is used to show the features below

	Features	Coefficients
0	french_door	240.0
1	bathroom	300.0
2	green_paint	370.0
3	qlm_security	440.0
4	kitchen	500.0
5	backyard	560.0
6	club_access	730.0
7	room	1000.0
8	solar_power	1530.0
9	woodfloor	1890.0
10	furnished	2000.0

Figure 1: Feature importance (Linear Regression)

While, based on Random Forest the feature importance are as follows: Room is the most important feature in predicting the target variable (Price) with its weighing 43 % prediction power. This is immediately followed by if the house is furnished and this accounts for 19%, wood floor runs to 17%, solar power is at 11%, club_access 2.5%, backyard kitchen1.4%, kitchen 1.1%, qlm_security 0.88%, French_door 0.87%, green paint 0.6%, and bathroom 0.4%.

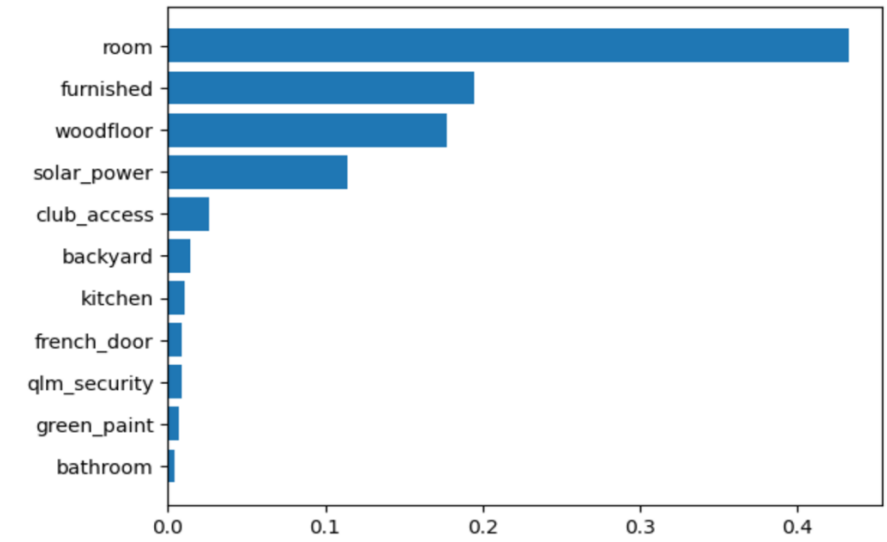


Figure 2: Feature importance (Random Forest)

The predicted house prices are provided below as against the actual prices and the difference between both values given employing the Linear Regression tool, using a snapshot of six figures at the beginning and another six figures at the end. "…" connotes all other figure in between

	A	B	C
1	**Actual price**	**Predicted Price**	**Difference**
2	5068	5055	13
3	7658	7645	13
4	11318	11305	13
5	8858	8845	13
6	11178	11165	13
7	11388	11375	13
8	…	…	…
9	5488	5475	13
10	10088	10075	13
11	9788	9775	13
12	9388	9375	13
13	8528	8515	13
14	6118	6105	13

Figure 3: Actual and Predicted price (Linear Regression)

The total given and estimated values are available in this link

CONCLUSION

The model has been able to identify the most important features that contribute to the prediction of house prices, which can be used by housing agents and home buyers to make more informed decisions.

REFERENCE

Almeida, Fernando. (2017). Benefits, Challenges and Tools of Big Data Management. Journal of Systems Integration. 8. 12-20. 10.20470/jsi.v8i4.311.

Aleixo, C. (2021) What Makes a Great Delivery Experience? Key Findings in Our Data. 12 July.

Ester, Frommelt, Kriegel and Sander (2000) Spatial Data Mining: Database Primitives, Algorithms and Efficient DBMS Support. *Data Mining and Knowledge Discovery* 4 (2), 193–216.

Kramer, O. (2013) K-Nearest Neighbors. *Dimensionality Reduction with Unsupervised Nearest Neighbors* Berlin, Heidelberg: Springer Berlin Heidelberg. 13–23.

Liu, Y., Wang, Y. and Zhang, J. (2012) New Machine Learning Algorithm: Random Forest. *Information Computing and Applications* Berlin, Heidelberg: Springer Berlin Heidelberg. 246–252.

Nasteski, V. (2017) An overview of the supervised machine learning methods. *HORIZONS.B* 4, 51–62.

Zhang, Z. (2016) Introduction to machine learning: k-nearest neighbors. *Annals of Translational Medicine* 4 (11), 218–218.

APPENDIX

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

```
import pandas as pd
from sklearn.metrics import mean_absolute_error
import seaborn as sns
```

```
def make_plot(truth, prediction):
  plt.plot(truth, color="red", label="truth")
  plt.plot(prediction, color="blue", label="predicted")
  plt.legend()
  plt.grid()
  plt.title("Comparing truth and predicted regression values")
  plt.tight_layout()
  plt.show()
```

```
train = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/msc_training_dataset.csv")
test = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/msc_testing_dataset.csv")
```

```
train.head()
```

	room	bathroom	kitchen	french_door	backyard	furnished	green_paint	\
0	3	1	2	1	1	0	1	
1	5	2	2	2	1	0	0	
2	5	2	2	2	1	0	0	
3	1	2	1	2	0	0	0	
4	2	1	2	3	1	1	0	

	solar_power	woodfloor	qlm_security	club_access	price
0	0	0	1	1	6835
1	0	0	1	1	9005
2	0	0	1	1	9005
3	0	1	1	0	5105
4	0	1	1	0	9105

```
test.head()
```

	room	bathroom	kitchen	french_door	backyard	furnished	green_paint	\
0	1	1	1	3	0	0	1	
1	5	1	1	2	0	0	0	
2	5	1	1	3	0	0	0	
3	4	2	2	1	0	1	1	
4	5	2	1	1	0	1	1	

	solar_power	woodfloor	qlm_security	club_access	price
0	1	0	1	0	5068
1	0	0	1	1	7658
2	1	1	1	1	11318
3	0	0	1	0	8858
4	1	0	0	1	11178

```
train.describe()
```

	room	bathroom	kitchen	french_door	backyard	\
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	
mean	2.990000	1.489000	1.522000	1.998333	0.490333	
std	1.424281	0.499962	0.499599	0.813153	0.499990	
min	1.000000	1.000000	1.000000	1.000000	0.000000	

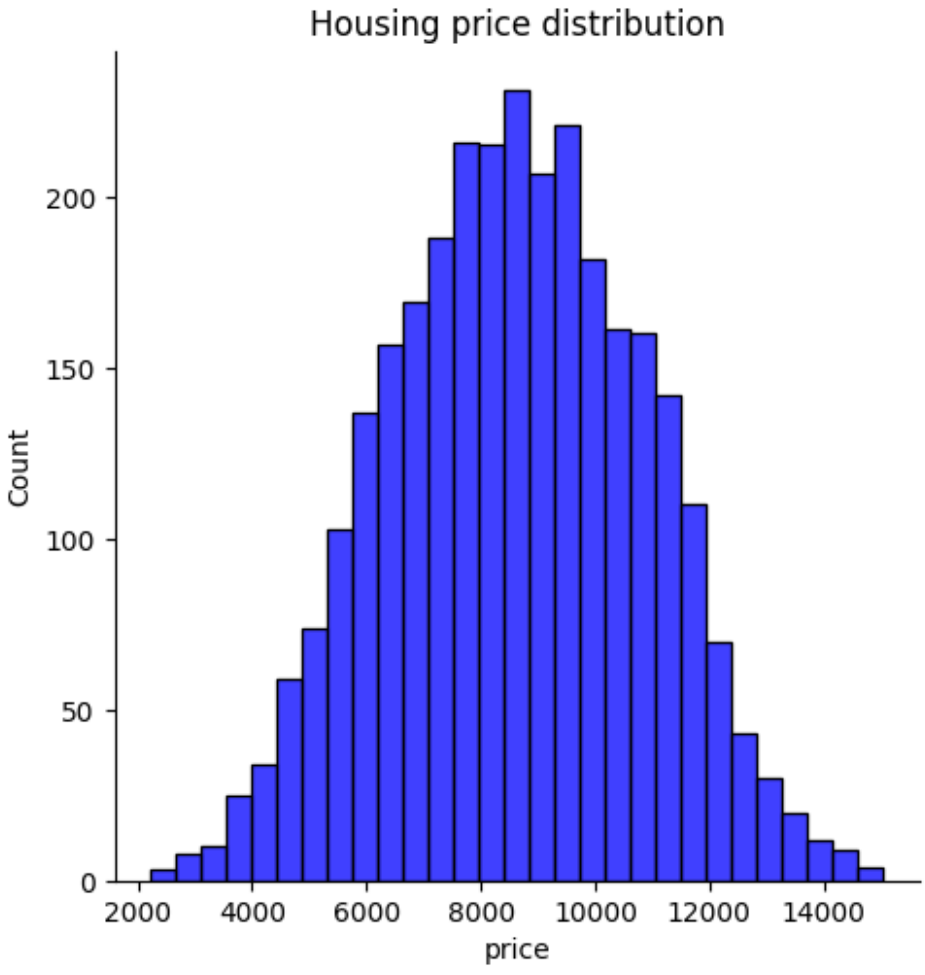
```
25%       2.000000     1.000000     1.000000     1.000000     0.000000
50%       3.000000     1.000000     2.000000     2.000000     0.000000
75%       4.000000     2.000000     2.000000     3.000000     1.000000
max       5.000000     2.000000     2.000000     3.000000     1.000000

         furnished  green_paint  solar_power    woodfloor  qlm_security  \
count  3000.000000  3000.000000  3000.000000  3000.000000   3000.000000
mean      0.488667     0.485000     0.495667     0.512333      0.480667
std       0.499955     0.499858     0.500065     0.499931      0.499709
min       0.000000     0.000000     0.000000     0.000000      0.000000
25%       0.000000     0.000000     0.000000     0.000000      0.000000
50%       0.000000     0.000000     0.000000     1.000000      0.000000
75%       1.000000     1.000000     1.000000     1.000000      1.000000
max       1.000000     1.000000     1.000000     1.000000      1.000000

       club_access         price
count  3000.000000   3000.000000
mean      0.499667   8606.600000
std       0.500083   2216.248563
min       0.000000   2235.000000
25%       0.000000   7005.000000
50%       0.000000   8615.000000
75%       1.000000  10215.000000
max       1.000000  15035.000000
```

```
sns.displot(train["price"], kind="hist", color="blue")
plt.title("Housing price distribution")
```

```
Text(0.5, 1.0, 'Housing price distribution')
```




```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   room          3000 non-null   int64
 1   bathroom      3000 non-null   int64
 2   kitchen       3000 non-null   int64
 3   french_door   3000 non-null   int64
 4   backyard      3000 non-null   int64
 5   furnished     3000 non-null   int64
 6   green_paint   3000 non-null   int64
 7   solar_power   3000 non-null   int64
 8   woodfloor     3000 non-null   int64
 9   qlm_security  3000 non-null   int64
 10  club_access   3000 non-null   int64
 11  price         3000 non-null   int64
dtypes: int64(12)
memory usage: 281.4 KB
```

```
train['price'].describe()
```

```
count     3000.000000
mean      8606.600000
std       2216.248563
min       2235.000000
25%       7005.000000
50%       8615.000000
75%      10215.000000
max      15035.000000
Name: price, dtype: float64
```

```
test.describe()
```

```
             room    bathroom     kitchen  french_door    backyard  \
count  999.000000  999.000000  999.000000   999.000000  999.000000
mean     3.019019    1.491491    1.496496     1.959960    0.510511
std      1.413731    0.500178    0.500238     0.809759    0.500140
min      1.000000    1.000000    1.000000     1.000000    0.000000
25%      2.000000    1.000000    1.000000     1.000000    0.000000
50%      3.000000    1.000000    1.000000     2.000000    1.000000
75%      4.000000    2.000000    2.000000     3.000000    1.000000
max      5.000000    2.000000    2.000000     3.000000    1.000000

        furnished  green_paint  solar_power   woodfloor  qlm_security  \
count  999.000000   999.000000    999.00000  999.000000    999.000000
mean     0.474474     0.523524      0.49049    0.501502      0.500501
std      0.499598     0.499696      0.50016    0.500248      0.500250
min      0.000000     0.000000      0.00000    0.000000      0.000000
25%      0.000000     0.000000      0.00000    0.000000      0.000000
50%      0.000000     1.000000      0.00000    1.000000      1.000000
75%      1.000000     1.000000      1.00000    1.000000      1.000000
max      1.000000     1.000000      1.00000    1.000000      1.000000

       club_access         price
count   999.000000    999.000000
mean      0.495495   8601.863864
std       0.500230   2217.898743
min       0.000000   2688.000000
25%       0.000000   7068.000000
50%       0.000000   8608.000000
```

```
75%       1.000000  10128.000000
max       1.000000  14318.000000
```

```
train_X = train.drop('price', axis=1)
test_X = test.drop('price', axis=1)
train_Y = train['price']
test_Y = test['price']
```

```
train_X.head()
```

```
   room  bathroom  kitchen  french_door  backyard  furnished  green_paint  \
0     3         1        2            1         1          0            1
1     5         2        2            2         1          0            0
2     5         2        2            2         1          0            0
3     1         2        1            2         0          0            0
4     2         1        2            3         1          1            0

   solar_power  woodfloor  qlm_security  club_access
0            0          0             1            1
1            0          0             1            1
2            0          0             1            1
3            0          1             1            0
4            0          1             1            0
```

```
test_Y
```

```
0       5068
1       7658
2      11318
3       8858
4      11178
       ...
994    10088
995     9788
996     9388
997     8528
998     6118
Name: price, Length: 999, dtype: int64
```

```
# Let's build a linear regression model
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(train_X, train_Y)
```

```
reg
```

```
LinearRegression()
```

```
# use the linear regression model to predict
predicted = reg.predict(test_X)
```

```
predicted
```

```
LR  = pd.DataFrame(predicted, columns=['LR Predicted Price'])
```

```
LR
```

```
     LR Predicted Price
0                5055.0
1                7645.0
2               11305.0
3                8845.0
4               11165.0
..                  ...
994             10075.0
```

```
995              9775.0
996              9375.0
997              8515.0
998              6105.0

[999 rows x 1 columns]
```

```
print(reg.score(train_X, train_Y))
```

```
1.0
```

```
print(mean_absolute_error(test_Y, predicted))
```

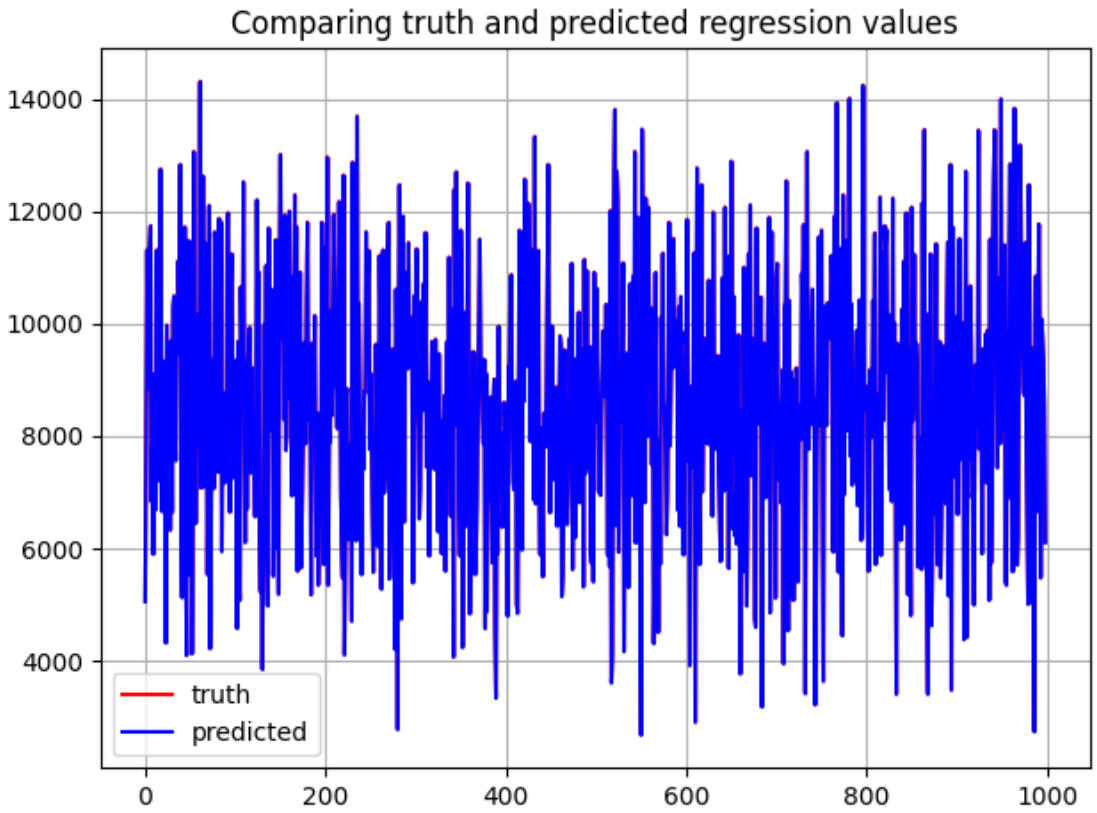
```
13.000000000000469
```

```
coefficients = reg.coef_
feature_importance = list(zip(train_X, coefficients))
feature_importance.sort(key=lambda x: abs(x[1]))
df_coef = pd.DataFrame(feature_importance, columns=['Features', 'Coefficients'])
print(df_coef)
```

```
        Features  Coefficients
0    french_door         240.0
1       bathroom         300.0
2    green_paint         370.0
3   qlm_security         440.0
4        kitchen         500.0
5       backyard         560.0
6    club_access         730.0
7           room        1000.0
8    solar_power        1530.0
9      woodfloor        1890.0
10     furnished        2000.0
```

```
import matplotlib.pyplot as plt
```

```
make_plot(test_Y, predicted)
```



```
# Now let's do random forest regression
from sklearn.ensemble import RandomForestRegressor
rf_reg = RandomForestRegressor().fit(train_X, train_Y)
rf_predicted = rf_reg.predict(test_X)
print(rf_reg.score(train_X, train_Y))
print(mean_absolute_error(test_Y, rf_predicted))
```

```
0.998469334194852
171.9256256256256
```

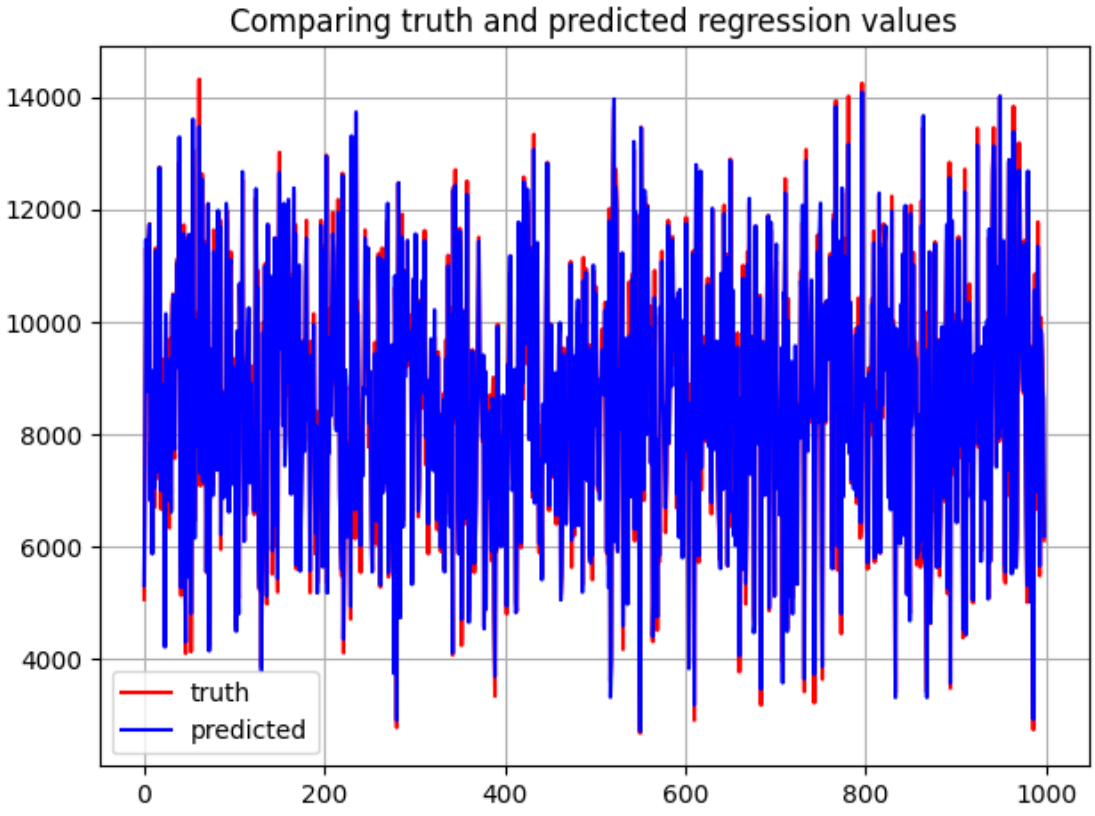
```
rf_predicted
```

```
array([ 5319.4,  7706.8, 11467.7,  8766. , 11096.2, 11467.5, 11738.3,
        6824.6,  7877.5,  5877.4,  9135.6,  8506.5,  7059.1, 11269.8,
        7340.4,  9500.9,  8941.9, 12745.4,  9804.9,  7002.5,  8887.5,
        7440.8,  7114. ,  4221.1, 10146.3,  7360.4,  7316.8,  9045.5,
        6825.9,  9344.7,  6619.5, 10041.3, 10480.1,  8266.9,  7738.6,
       10241.1,  9198.5, 11057.5,  9437.4, 13292. ,  6604.8,  5278.8,
        9954.6,  8954.9, 11569.6,  6122.8,  4309.6, 11434.8,  9613.6,
        5464.1, 11557.5, 10545.1,  4816.6,  7936.9, 13608.6,  9708. ,
        6159.6,  8404.7, 10027. ,  7785.9,  7997.5, 13474.2,  7310.9,
        8469.7, 12546.3,  7159.8, 11378. , 10566.3,  9957.5,  5562.4,
        8171.4, 12096.3,  4152.3,  6248.6,  9510.5,  8701.8,  6914.7,
       11248.6, 10873.8, 10726.5,  8836.1,  7367.7, 11985.7,  8142.6,
       11704.5,  6219.3,  8745.6,  8313.3,  6875. ,  8258.9,  7610.9,
       12109. , 11787.7,  7935.5,  6619.5, 11042.1, 11105.2,  8260.3,
        9211.9,  7369.7,  7096. ,  8666.6,  4500.7,  9840.4,  5876.8,
        4810.8, 10687. ,  9740.6,  8477.1, 12673.8,  7824.5,  6102.4,
        9183.1,  6579.7,  9195.9,  9589.9, 10253. ,  7140.8,  8919.7,
        7478.4,  7896.2,  7421.5,  6640.4,  8554.5, 12371.4, 10242.8,
       10608.1,  5751.7,  5268.1,  6203.6,  3792.4,  6214.8,  9814.5,
        6059. , 10993. ,  6660.6,  5126.9, 11743.2,  8116.8,  8813.7,
       10821.3, 10007.1,  5936.1,  7572.2,  8754.9, 11491.4,  6404.8,
        7293.4,  5440.5, 10772.4, 12659. ,  9542.6, 10421.3, 11279.8,
        8163.4, 12109. ,  7444.3,  7832.2, 10233.8, 10394.5, 12181.7,
        8916.9, 10760.7,  6940.9,  9502. ,  9356.3, 12382.5,  7428.7,
       11569.6,  5677.6,  9503.1,  8010.7, 11012.7,  5569.8,  6486.9,
        7271. ,  8340.4,  9651.1,  7702.5, 10679. , 11489.6,  8272.6,
        8963.3,  8073.9,  5578.6,  9403.4,  8465.6,  7940.3,  9950. ,
        7531.1,  6248.6,  8146.9,  5181.9,  6563.4,  7278.2,  7140.8,
       11717.5,  9572.9,  5642.2, 11277.2,  9059.1,  7960.2, 12948.1,
        5181.9,  8093.5,  7666.7,  9724.3,  8327.3, 10331.2, 11563.6,
       11209.6,  9039.6,  8902.4,  8071.9,  9202. , 11952.1,  9512.7,
        6978.6,  6080.2,  5600.3, 12605.1,  4364.9,  6292.1,  9150.9,
        8906.9,  6187.6,  7810.4,  6900.6,  7478.4,  4945.9, 13310.6,
        8003.2,  7421.1,  9087.7,  6660.1, 13735.9,  8780.3, 10145.3,
        8464.5,  6756.2,  6024.2,  7885.9,  7258.2,  8836.1,  8703.7,
       11489.6,  9219.4, 10771.3, 11320.7,  8203.9,  9125.6,  8142.6,
        6146.5,  5556.2,  7286.7,  6941.9,  9438.7,  7584.8,  6460.3,
        6978.3, 11159.6,  9848.7,  5325.2,  8895. , 11107.8,  6966. ,
        7265.2, 10343.5,  7164.2, 10961.8, 12111. ,  5543. ,  6220.3,
        9507.1,  8423.2,  9596.9,  9056.1,  3752.1, 10821.3,  8867.4,
        2920.1,  9520.1, 12475.8, 10779.3,  4742.9, 10794.8, 11509.7,
        6355.7, 10089.3,  9020.3, 10889.6,  9315.5, 11447. ,  9372.7,
        9460.3,  9822.1,  8962.5,  5332.7,  8531.7, 10770.3,  8544.8,
       11567.6,  8635. ,  8134.9,  6641.6,  7088.4,  9395.5, 10359.7,
        8744.8, 10742.4, 10047.5, 11443.4,  7955.5,  8973.7,  8302.6,
        6413.4,  8146.9,  8446. ,  9700.5,  8101.5,  7835.6,  7354.6,
       10216.9,  7579.9,  9248.1,  6480.4,  9376.7,  7300.1,  8344.5,
        5993.1,  7656.4,  8068.4,  8619. ,  5556.6,  7879.5,  9347.7,
        6995.3, 11003.7,  7971.2,  6363.9, 10119.6,  7440.8,  4115.8,
       12341.5,  8303.6, 12431.3,  9144.2,  9659.7,  6676.9,  5878.8,
```

```
11620.4, 10937.7,  4712.8,  7387.4, 10182.6,  6508.3,  7219.6,
 9416. , 12268.1,  8477.1,  4660.5,  5984.6,  6481.8,  8202.2,
 9481.6,  8748.1,  5709.2,  6099.1,  6794.2,  8398.5,  9852.4,
11431. ,  9531.7,  8888.6,  8490.3,  7937.4,  9415.3,  4547. ,
 9125.6,  4896.6,  8206.3,  7808.9,  6881.4,  8415.8,  7492.1,
 5929.4,  8646.2,  8527.8,  4943.8,  3700.3,  7485.2,  5972. ,
 9921. ,  7139.5,  7988.3,  7648.9,  6003.9,  6902.4,  8701.4,
 7826. ,  7963.4,  7704.6,  4952.3,  9227.9,  8919.7, 10359. ,
11180.9,  7786.4,  7007.7,  7210.2,  8492.4,  9161.6,  4832.2,
 4876.3,  8497.8, 11777.7,  7354.2, 10248.1,  6076.1, 11757.1,
 8920. , 12492.5, 11633.5,  9613.1, 11324.1, 12362.8, 11620.1,
 7911.1, 10194.4, 11284.4,  9169.5,  7002.1, 13069.3,  6780.5,
10258.4,  7032.8, 11413.1,  7248.8,  6057.8,  7540.2,  6776. ,
 5418.6,  8536.7,  7881.5,  7508.9,  9086.8, 10916.3, 12824.2,
 8726.7,  6733.2,  7835. ,  9951.6,  7889.2,  8329.4,  8250.2,
 7268.5,  9099.4,  6568. ,  8457.5,  8148. ,  9665.7,  9990. ,
 5057.8,  5624.1,  6837.8,  9418.2,  8369.2,  7091.2,  6403.8,
 7492.2,  8352.8,  9726.4,  8494.8, 11027.7,  6121.5,  8395.5,
 9385.3,  6370. ,  7696.2,  9377.6,  9683.2, 10391. ,  6382. ,
 7861. ,  7397. ,  9925.1,  5196.4, 10734.8,  7215.9,  8625.6,
10883.4,  8625.6,  7173.2,  9529.8,  5882.2,  5716.3,  9116.3,
 5853. , 11012.7,  8156.6,  9837.6, 10502. ,  8415.8,  7090.1,
 8313.3,  6843.6,  8986.6,  8919.4,  9500.6,  9002.8,  8703.1,
10205.4,  8646.7,  5921.8, 10360.7,  5278.7, 11796.8,  3328.4,
 3942.3,  9332.2, 12594.3, 13966.1,  6094.2, 12401.2, 11964.9,
 5916.1,  8443.1,  9145.1,  8103.1,  9411.6, 11232.5,  4598.5,
 7334.6,  9219.7,  9707.1,  7161.5,  4982.6,  9145.1,  9750.2,
 9761.2,  6893.7, 10754.3,  7379.4, 13210.8,  6315.1, 11954.4,
11651.5,  7889.2,  7421.5,  5614.2,  2723.6, 13455. , 10253.3,
 8030. ,  7024.3, 12339. ,  8773. ,  8003.2, 12042.8, 10193.9,
 9650.8,  9903. ,  7484.2,  9470. ,  4411.6,  5371.6, 10435.6,
 6009.5,  7570.2,  4759.8,  7911.1,  5845.2, 10265.3,  8471.6,
11080. ,  8562.4,  8436.2,  7610.8,  6701. ,  7322.6,  9030.5,
11717.5,  7857.2,  8847.5,  9542.2,  8859.8, 11458.6,  9688.2,
 9496.5, 10394.5,  9281.3,  6941.5, 10518.5,  6171. , 10579.5,
 7104.9,  8215.6,  5807.6,  7164.2, 10041.2,  9413.8, 11752.9,
 7713.9,  8404.7,  3837.5,  8489.8,  8558.8,  7210.7,  9580.4,
11213.5,  3193.2,  9219.7, 12799.4, 10106. ,  7059.1,  5845.2,
12199.2, 12685.3,  7405.9,  8739.1,  9574.3,  9513.5,  9556.7,
 7923.5,  9403.5, 10663.7,  7786.7,  8242.3,  8480.9,  6787.4,
12023.6,  8898.2,  8074.8,  7873.5,  7722.2,  9661.8,  7462.5,
 6609.6,  6083.9,  5616. , 10867.3,  9924.9,  7028. , 11929.2,
 8705.1,  8510.5, 11142.8,  5671.7,  9120.7,  6619.6, 12869.6,
 8264.2,  9155.1, 10564.5,  6195.8,  6965.6,  6026.5,  8034.6,
 9556.1,  7018.1,  4067.8,  8411.2,  8645.9,  5616.2, 10776.8,
 9253.5,  7432.8,  5354.2,  6461.6, 11022.4,  8506. , 12199.2,
 7294.2,  9735.9,  9574.3,  9540.8,  4476. ,  4629.6, 11711.8,
 8877.7,  7868.8, 10050.5,  7816.9, 10433.4,  3471.1,  7835. ,
 9634.2,  7469.5,  6922.1,  8571.5,  9422.7, 10993. , 11888.6,
 4917.6,  7247.9, 11774.2,  8176.8,  8785.6,  9651.6,  5126.7,
10451.8, 11384.3,  7043.5,  8340.2,  8072.3,  6571.8,  8436.2,
 9547. ,  3584.8, 10518.5, 10413.2, 12290.8,  9050.6,  4496.5,
10027.1,  5121.4,  9127.1,  6833.5,  9295.6,  4810.8,  8751.6,
 6410.9,  9578.4,  7572.2,  5336.6,  7991.8,  8486.2,  7935.1,
10731.8, 10306.6, 12077.9,  9981.8,  3653.2,  9912.9, 12869.5,
 9134.1,  7704.3,  9248.7,  8303.6,  9739.8, 10748. ,  8917.9,
 8551.9,  3735.3,  5578.6,  8014.6,  8981.5, 11248.7,  8223.2,
11548.1, 12113.9,  6851. ,  3871.4,  9688.2, 10282.9,  9470. ,
 8249.8, 10381.4,  9342.6,  9970.8, 10253.3, 11155.4, 10710.4,
 5612. ,  9762. , 11818.6,  8454. , 13834.7,  9791. ,  6012.2,
11444.5, 11027.7,  9234.3,  4814.8, 12382.5,  6892.9, 10339.3,
```

```
       11160.1, 10975.1,  9443.1,  7861. , 13160.3,  7680. , 10328. ,
        7194.8,  8151.7,  7239.7,  7029.9,  8552.8,  9773. ,  6904. ,
        7371.3,  9916.2,  9423.6,  6442. ,  6432.2, 14087.4,  8605.6,
        8726. ,  8963.3,  8986.6,  6498.2,  5721.9,  6121.5,  9089.5,
        8502.2,  8249.2, 10309.2,  9295.6, 11405.9,  5802.7,  7354.2,
        8395.3,  6997.6,  6003. , 12294.2,  5897.1, 11289.2, 10324.8,
        8532.3, 11388.3, 11419.9, 11711.6,  9161.8, 10240.8,  6886.3,
        8661. ,  7789.5,  6661. , 11966.8,  7274.4,  9934.1,  7521.4,
        3322.6,  9886.2,  7855.5,  8608. ,  9491.8,  6056. , 10310.1,
        6987.9, 11206.4, 11057.5,  6201.4, 12072.2,  9752. ,  5151.1,
        9488.3,  7263.5,  4689.6, 11929.2,  8150.7,  8755.8, 11019.5,
        8764.5,  9373.6,  9147.3,  5808.5,  7245.2,  5918.1,  8342.8,
        6121.5, 11714.7,  9095.7, 13670.8,  5802.7,  5986.8,  5399. ,
        3322.6,  9513.1,  8480.9, 11245.2,  4653.9,  6087.7,  7164.1,
        7105.6,  9603.2, 11384.6,  8358.1,  5633.9,  7515. ,  9653. ,
        5662.8,  6461.6,  7853.6,  9920.6,  8474.7,  6880. ,  6905.4,
       10053.3, 11732.9,  5265.4,  8680.4, 12564.8,  3579. , 10794.8,
       11802.3,  8753.8,  7445.3,  9505.7,  9999.5,  6433.4,  9367.4,
       11458.6,  7804.1,  8048.3,  9962.5,  8958.7,  4537. ,  8179.5,
       12315.2,  4448.7,  7577.7, 10034.9,  7539.4, 10345.6,  9926.5,
        6880.5,  7210.2,  5054. ,  8052.9,  9427.8,  8999.8,  8307.9,
       13144.6,  7837.4,  9195. ,  9315.9,  5747.4,  9564.1,  8575.5,
        8961.5,  9909.4,  6848. ,  9805.2, 10032.5,  5075.9, 11652.5,
        5741.8,  7150.7, 10827.9, 12693.4, 13132. , 12696.3,  8703.7,
        7421.1,  9669.3, 10980.9,  7945.6, 14018.6,  8152.9,  8701.4,
       10897.3, 11446. ,  5730.1,  5551.2, 10041.2,  7375.3, 11101.1,
       12887. ,  8679.3,  7268.8,  5522.1,  8215.5, 13383.9,  9067.6,
        9761.2,  5633.9,  6304.4, 10515.4, 12701.2, 11989.4,  9693.2,
        9499.7,  8968. ,  8784.1, 11378. ,  8370.5,  6539.5,  5319.8,
       12685.3, 10377.7,  8363. ,  5019.2,  9538.1,  5347.6,  2937.2,
        7433.1, 10576. ,  6988.7,  9988.1, 11344.4, 10121.1,  5662.8,
        9855.9,  9791. ,  8867.9,  8618.3,  6194.2])
```

```
make_plot(test_Y, rf_predicted)
```




```
rf_reg.feature_importances_
```

```
array([0.43260264, 0.00470294, 0.01106128, 0.00898726, 0.01438701,
       0.19506325, 0.00689138, 0.1139952 , 0.17729137, 0.0089835 ,
       0.02603417])
```

```
train.columns
```

```
Index(['room', 'bathroom', 'kitchen', 'french_door', 'backyard', 'furnished',
       'green_paint', 'solar_power', 'woodfloor', 'qlm_security',
       'club_access', 'price'],
      dtype='object')
```

```
imp_scores = zip(rf_reg.feature_importances_, train.columns)
sorted(list(imp_scores), reverse=True)
```

```
[(0.43260264196807924, 'room'),
 (0.19506324615589132, 'furnished'),
 (0.17729137404802492, 'woodfloor'),
 (0.11399519924971835, 'solar_power'),
 (0.026034168958970056, 'club_access'),
 (0.014387006640405084, 'backyard'),
 (0.011061281011654888, 'kitchen'),
 (0.008987262191067016, 'french_door'),
 (0.008983501587112376, 'qlm_security'),
 (0.006891377228734774, 'green_paint'),
 (0.004702940960342051, 'bathroom')]
```

```
rf_importances = rf_reg.feature_importances_
rf_reg.feature_names_in_
ranks_and_features = zip(rf_importances, rf_reg.feature_names_in_)
ranks_and_features = sorted(ranks_and_features,reverse=True)
for x, y in ranks_and_features:
    print(x, y)
```

```
0.43260264196807924 room
0.19506324615589132 furnished
0.17729137404802492 woodfloor
0.11399519924971835 solar_power
0.026034168958970056 club_access
0.014387006640405084 backyard
0.011061281011654888 kitchen
0.008987262191067016 french_door
0.008983501587112376 qlm_security
0.006891377228734774 green_paint
0.004702940960342051 bathroom
```

```
ranks_and_features
```

```
[(0.43260264196807924, 'room'),
 (0.19506324615589132, 'furnished'),
 (0.17729137404802492, 'woodfloor'),
 (0.11399519924971835, 'solar_power'),
 (0.026034168958970056, 'club_access'),
 (0.014387006640405084, 'backyard'),
 (0.011061281011654888, 'kitchen'),
 (0.008987262191067016, 'french_door'),
 (0.008983501587112376, 'qlm_security'),
 (0.006891377228734774, 'green_paint'),
 (0.004702940960342051, 'bathroom')]
```

```
keys = [k[1] for k in ranks_and_features ] [::-1]
keys
```

```
['bathroom',
 'green_paint',
```

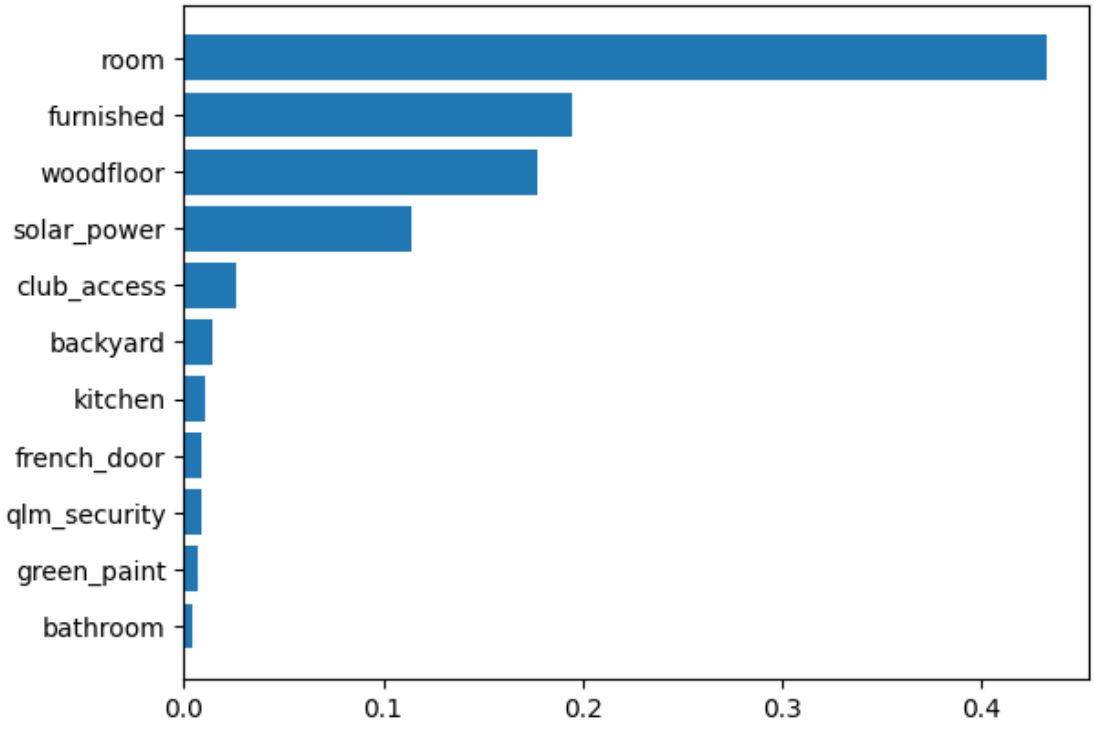
```
 'qlm_security',
 'french_door',
 'kitchen',
 'backyard',
 'club_access',
 'solar_power',
 'woodfloor',
 'furnished',
 'room']

values = [k[0] for k in ranks_and_features ][::-1]
values
```

```
[0.004702940960342051,
 0.006891377228734774,
 0.008983501587112376,
 0.008987262191067016,
 0.011061281011654888,
 0.014387006640405084,
 0.026034168958970056,
 0.11399519924971835,
 0.17729137404802492,
 0.19506324615589132,
 0.43260264196807924]
```

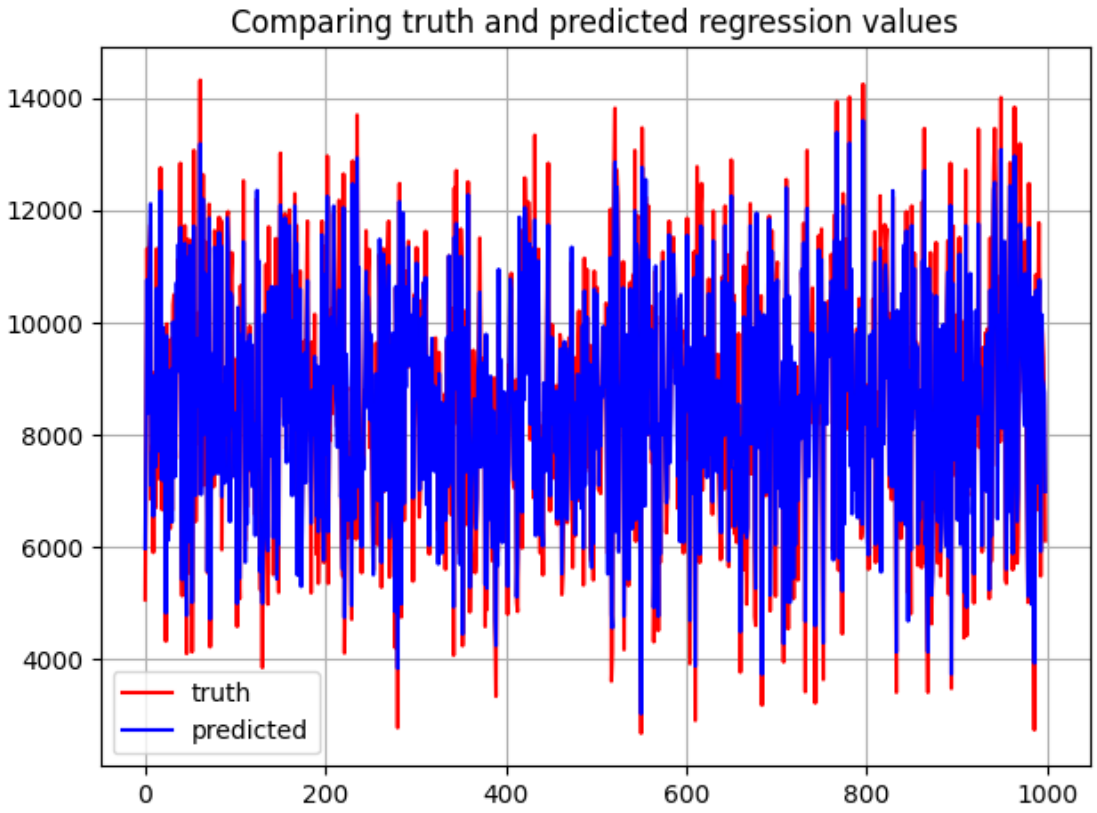
```
plt.barh(keys, values)
```

```
<BarContainer object of 11 artists>
```



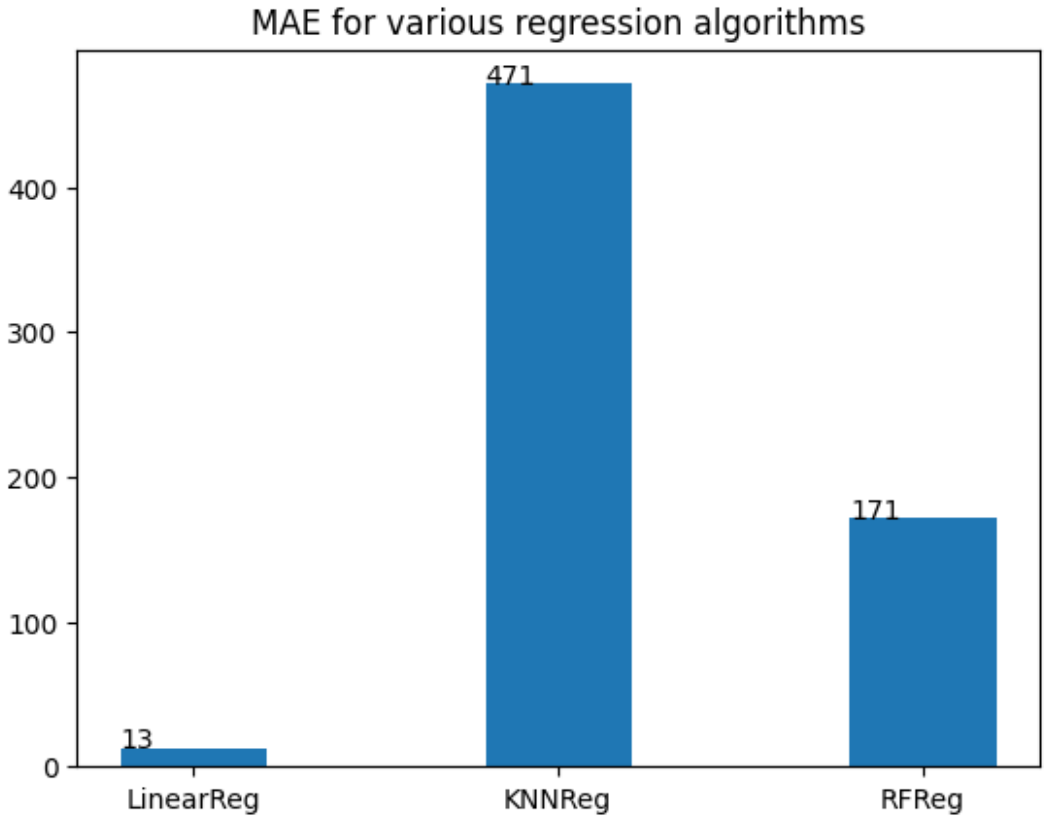

```
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors=7)
neigh.fit(train_X, train_Y)
knn_predicted = neigh.predict(test_X)
print(neigh.score(train_X, train_Y))
print(mean_absolute_error(test_Y, knn_predicted))
make_plot(test_Y, knn_predicted)
```

```
0.9471383275821861
471.3446303446304
```




```
# Let's plot the mean absolute error of all the algorithms we have tried
lin_mae = mean_absolute_error(test_Y, predicted)
knn_mae = mean_absolute_error(test_Y, knn_predicted)
rf_mae = mean_absolute_error(test_Y, rf_predicted)
errors = [lin_mae, knn_mae, rf_mae]
labels = ["LinearReg", "KNNReg", "RFReg"]
bars = plt.bar(labels, errors, width=0.4)
for bar in bars:
    yval = int(bar.get_height())
    plt.text(bar.get_x(), yval + .005, yval)
#plt.grid()
plt.title("MAE for various regression algorithms")
```

```
Text(0.5, 1.0, 'MAE for various regression algorithms')
```




```
import numpy as np
New_Table = test.join(LR)

New_Table
```

	room	bathroom	kitchen	french_door	backyard	furnished	green_paint	\
0	1	1	1	3	0	0	1	
1	5	1	1	2	0	0	0	
2	5	1	1	3	0	0	0	
3	4	2	2	1	0	1	1	
4	5	2	1	1	0	1	1	
..	...	...	...	...	...	...	...	
994	5	2	2	3	1	1	0	
995	5	1	2	3	1	1	0	
996	3	2	2	1	0	1	1	
997	3	2	1	1	1	0	0	
998	2	1	2	1	0	1	1	

	solar_power	woodfloor	qlm_security	club_access	price	\
0	1	0	1	0	5068	
1	0	0	1	1	7658	
2	1	1	1	1	11318	
3	0	0	1	0	8858	
4	1	0	0	1	11178	
..	...	...	...	...	...	
994	0	0	0	0	10088	
995	0	0	0	0	9788	
996	1	0	1	0	9388	
997	1	1	0	0	8528	
998	0	0	0	0	6118	

	LR Predicted Price
0	5055.0
1	7645.0
2	11305.0
3	8845.0

```
4               11165.0
..                  ...
994             10075.0
995              9775.0
996              9375.0
997              8515.0
998              6105.0

[999 rows x 13 columns]
```

```
New_Table.to_csv("/content/drive/MyDrive/Colab Notebooks/New_Table.csv")
```