CS5812- Predictive Data Analysis Assessment/Coursework

Department of Computer Science, Brunel University

Student ID:

2022/2023

1. Data Description and Research Question

This dataset describes how online reservation channels have greatly impacted how hotels are booked and led to changes in customer behaviour. Cancelled reservations and no-shows are increasingly common, with reasons such as changes in plans or scheduling conflicts. While guests benefit from the option to cancel for free or at a low cost, this can be a challenge for hotels as it may result in a decrease in revenue[1]. Our target variable here is the average room price.

The hotel reservation dataset of 19 columns and 36,276 rows was retrieved from the Kaggle website in 2022. The original dataset we retrieved was randomized by my team to enable us to demonstrate some data cleaning knowledge learnt in this course. We added random noise by manually deleting some instances and changing some spelling in the data.

The data dictionary below describes the variables of the dataset.

Name	Data Type	Note
Booking_ID	Categorical	Unique identifier of each booking
no_of_adults	Numerical	Number of adults
no_of_children	Numerical	Number of children
no_of_weekend_nights	Numerical	Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay in the hotel
no_of_week_nights	Numerical	Number of weeknights (Monday to Friday) the guest stayed or booked to stay in the hotel
type_of_meal_plan	Categorical	Type of meal plan booked by the customer
required_car_parking_space	Numerical	Does the customer require a car parking space? (0 - No, 1- Yes)
room_type_reserved	Categorical	Type of room reserved by the customer. The values are ciphered (encoded) by INN Hotels
lead_time	Numerical	Number of days between the date of booking and the arrival date
arrival_year	Numerical	Year of arrival date
arrival_month	Numerical	Month of arrival date
arrival_date	Numerical	Date of the month
market_segment_type	Categorical	Market segment designation.
repeated_guest	Numerical	Is the customer a repeated guest? (0 - No, 1- Yes)
no_of_previous_cancellations	Numerical	Number of previous bookings that were canceled by the customer prior to the current booking
no_of_previous_bookings_not_canceled	Numerical	Number of previous bookings not canceled by the customer prior to the current booking
no_of_special_requests	Numerical	Total number of special requests made by the customer (e.g. high floor, view from the room, etc)
booking_status	Categorical	Flag indicating if the booking was canceled or not.
avg_price_per_room	Numerical	Average price per day of the reservation; prices of the rooms are dynamic. (in euros)

My research question is to predict the revenue per room (average price per room) made or forfeited by the hotel given other customer reservation details including if there was a cancellation or not.

2. Data Preparation and Cleaning

The data preparation, cleaning and Exploratory Data Analysis stages were performed using R-Studio. We commenced by loading the required packages (ggplot2 and validate), after which the dataset was loaded and viewed.

[1] https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset?select=Hotel+Reservations.csv

The variables of our data set were read correctly except for type_of_meal_plan, room_type_reserved, market_segment_type, and booking_status which were read in as characters instead of factors. We decided to leave them as factors because we intend to recode these variables during data transformation.

The following steps were taken for **Quality Check** and **Cleaning**:

a. Detecting missing values: by summing columns with NA in the dataset, we recorded one missing value in avg_price_per_room.
b. Detecting Duplicates: We recorded no duplicates by checking the dimension of unique instances and summing duplicated instances. However, for further checks, we removed the Booking ID and over 10,000 rows appeared to be duplicated. Knowing that the dataset is about hotel bookings, these instances do not seem implausible, and we decided the booking ID distinguished the instances and they are a part of our data, so we did not take them out.
c. Data Validation: This was done using the "Validator" function. We recorded 2 failures (in room_type_reserved and meal_plan_type variables) and one missing value (as detected in "a" above). The two fails were wrong spellings.
d. Data Cleaning
    i. Fixing the wrong spelling: "RoomType 1" was corrected to "Room Type 1" and "MealPlan 1" was corrected to "Meal Plan 1".
    ii. Dealing with missing values: The missing value (" "), was first re-coded to NA, and then the median was imputed. We created a new variable name for our dataset to avoid overwriting the original dataset.
    iii. Dealing with duplicates: Since we recorded no duplicates, nothing was done here.
    iv. (Simple) outlier detection: The boxplot of our target variable (avg_price_per_room) alone showed no outliers however when we compared against another variable, we had outliers and one price above 500 was significantly different from the rest of the data. We decided to explore this further during EDA before deciding if it should be taken out.

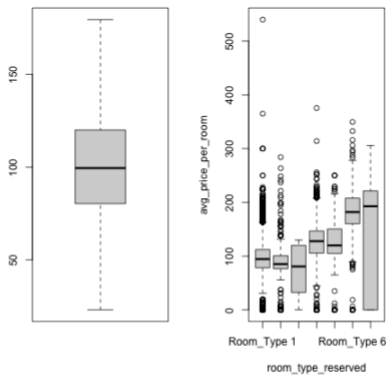


Figure 1: Outlier Detection

Feature Selection/Extraction: At this point, we dropped the booking ID to proceed with EDA. The Booking ID would not add significant information to the dataset.

Data Transformation: We re-coded all the categorical variables to numbers for ease of use in PCA and modelling, but we decided to perform this task just before Principal Component Analysis in EDA. It was more convenient to perform graphical visualization on subsets of categorical and numerical data separately rather than on a large dataset of 18 numerical variables. This especially affected the "pairs" plot (Scatter plot matrix).

3. Exploratory Data Analysis (EDA)

The EDA followed the following steps:

a. Statistical Exploration: Using str, Summary and head we ensured the data types were okay and there was no further cleaning required. Our dataset was clean.

Then we created label vectors for numerical and categorical variables.

b. Graphical Analysis for Numerical data:

i. Individual visualization of numerical variables: From the plots, none of the variables looks implausible. However, we noted that the average price is skewed to the right, we can investigate further if there are outliers responsible for this skew in data.

ii. **Checking multicollinearity:** From the correlation matrix and the pairs (scatter plot matrix) we observed no significant correlation between the numerical variables, but we noted 0.47 for the correlation between no_of_previous_bookings_not_canceled and no_of_previous_cancelations.

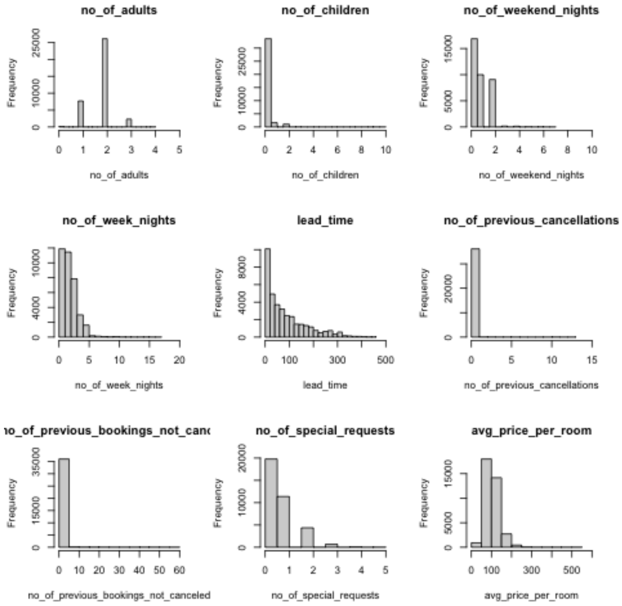


Figure 2: Independent visualization of numerical

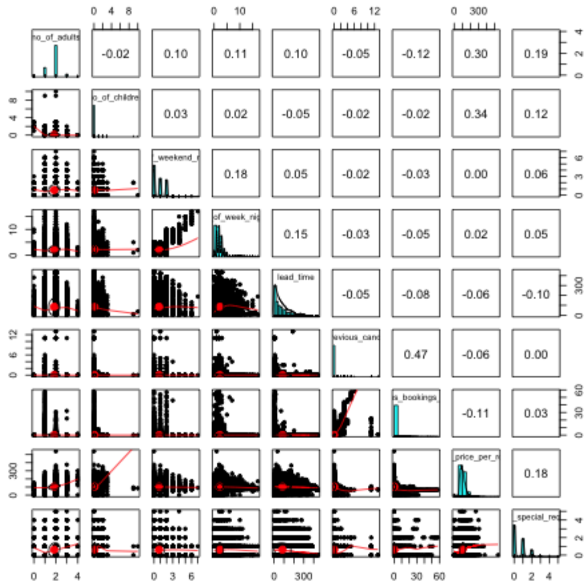


Figure 3: Correlation Pairs plot

c. Graphical analysis for categorical data:

i. Individual visualization of the categorical variables: There was no implausible detail about the categorical variables (See Figure A1 in the appendix).

ii. Visualizing the relationship of the booking status against other categorical data: There was no implausible detail about the categorical variables (See Figure A2 in the appendix).

d. Comparing average price per room and other variables: We could observe outliers in most of the instances here. The outlier for the price above 500 is likely to be skewing our data so we will remove it at this point. We don't have sufficient reason to remove the other outliers as they are most likely part of our data. After removing the price greater than 500 we obtained the plot in Figure 5.

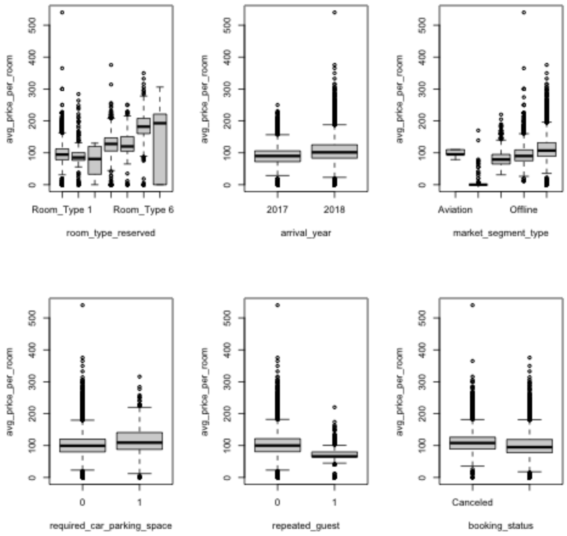


Figure 4: Box plots with Outliers

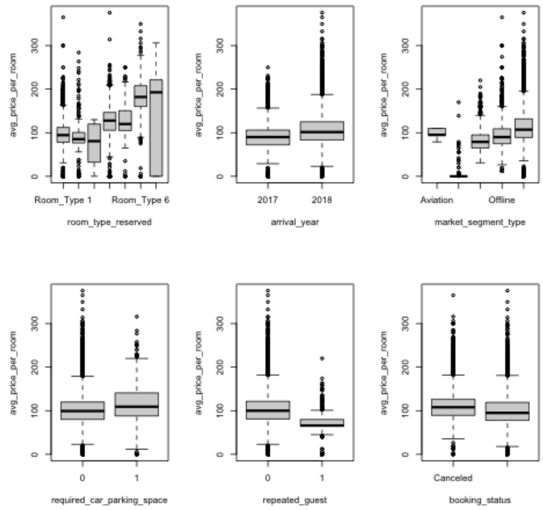


Figure 5: Box plots after the outlier is removed.

e. **Principal Component Analysis (PCA):** Before this stage, we re-coded all the categorical variables (type_of_meal_plan, room_type_reserved, market_segment_type, booking_status) to numbers (1,2,3, etc) so the entire data could be used for PCA and modelling. After encoding their variable type was changed from character to numerical.
We then performed PCA on the entire data excluding the target variable avg_price_per_room. From the **Scree plot** of the cumulative value of PEV for the increasing number of additional PCs (Figure 7), we added an 80% threshold line to inform the feature extraction. According to the plot, 10 PCs contributed 80% of the information in the data and were therefore selected.

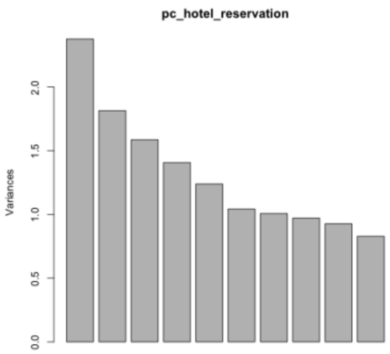


Figure 6: Visualizing the PC results as a the proportion of explained variance (PEV)

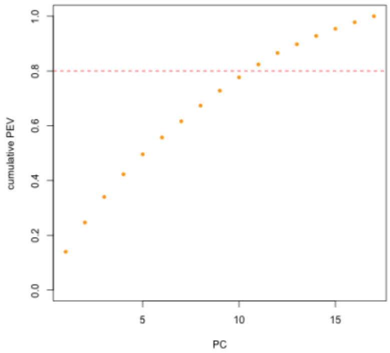


Figure 7: Scree Plot showing PC contributions

Inspecting the PC loadings: We can see that 40% of PC 1 contained attributes of market_segment_type, repeated_guest and no_of_previous_bookings_not_cancelled. Similarly, 40% of PC 2, contained, no_of_children and 80% of PC7 contains the arrival date.

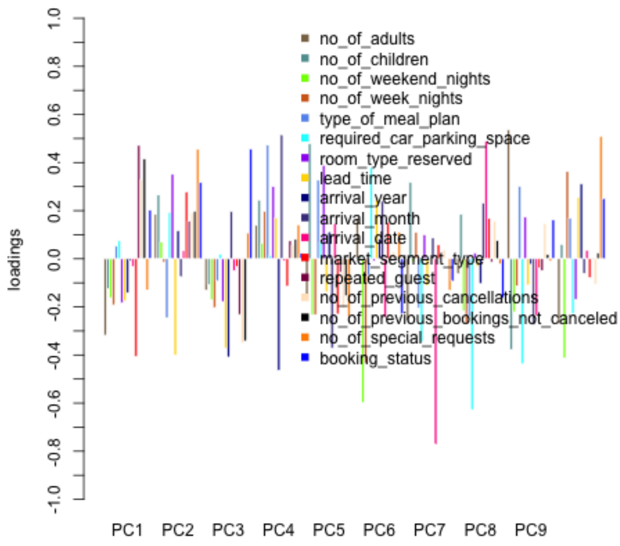


Figure 8: Inspecting PCA Loadings

Creating a new data frame using the PC's: After the principal component analysis was completed, we created a new dataset containing the PC's and re-joined our target variable, avg_price_per_room to continue our modelling. This data frame was written to our working directory as a .csv file.

4. Machine Learning Prediction - Random Forest Regressor

The modelling section of the coursework was completed using Python. I started by importing numpy, pandas, sklearn, random forest regressor, k nearest neighbour and other sklearn packages (Appendix - Figure A4).

Then, I loaded the CSV file containing the PCs and target variable into Google Colab and viewed the data.

Reading the input and label: The input data and label were represented as 2 arrays.

Split dataset: The data was split in a ratio of 70% for the training set and 30% for the testing set at a random state of 42.

RF model Training: From sklearn.ensemble, the RandomForestRegressor algorithm was used to train the model on the training set. The estimated number of trees was 100 trees (estimators). The code run time was 35 secs.

RF Hyperparameters Tuning: By creating an elbow plot (Figure 10) of the different number of trees (n_estimators) against the RMSE value, I arrived at an optimal number of estimators at N=50 after which the RMSE value does not reduce significantly anymore.

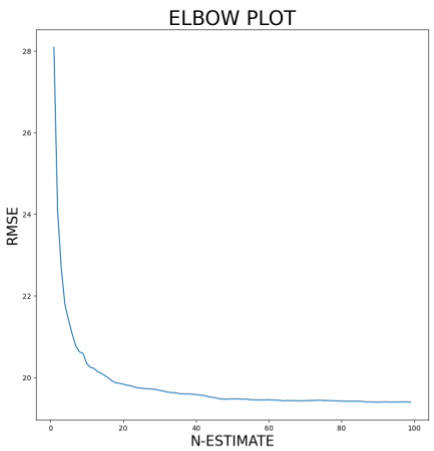


Figure 9: Elbow plot showing Tuning RF Tuning

RF Model Testing: The trained model (using the optimal parameters) was tested on the test set, and I created a scatterplot of the actual price against the predicted price (Figure A7 in the appendix), to visualize the model prediction.

5. Deep learning prediction – Convolution Neural Network (CNN)

The packages for the Deep learning model were loaded which are mainly Keras and Tensor flow (Figure A4). Using the same Python Colab notebook as for the machine learning model, the data set was already loaded.

Reading the input and label: The input data and label were represented as 2 arrays.

Split dataset: The data was split in a ratio of 70% for the training set and 30% for the testing set at a random state of 42.

Converting data to image: As a CNN requirement, I added an additional dimension to both training and testing samples to form an image.

CNN Model Training: From keras, the CNN algorithm keras.Sequential was used to train a LeNet-5 CNN model. The structure of the model comprised, 2 convolution layers, 2 max-pooling layers, and 2 fully connected (FC) layers with a "Relu" activation function in each of the FC layers. The Model is constructed using an SGD optimizer (back propagation algorithm) and a mean squared error loss function. Model parameters are initially set to: Learning rate is 1e-11, epochs is 100, and batch_size is 64. The code run time was 4:09s and a loss of 3903.3164 on the training set.

Hyperparameters Tuning: By changing the learning rate manually, the model only provided results with a very small learning rate of 1e-11. The number of epochs and batch size were changed in iterations of batch_size_list = [5, 10, 15, 20] and epoch_list = [5, 10, 50, 100]. In each case, the model was trained and RMSE was calculated. The optimal parameters obtained were batch size 15 and epoch 10 with a loss of 3897.3206 on the training set.

CNN Model Testing: The model was tested against the test sample and the result was plotted (Figure A4 in the appendix) showing the actual price versus the predicted price. From the plot, we can see the random scatter of the points implying that our model is performing poorly as it fails to make predictions that correlate with the actual prices.

Data Normalization

Considering the small learning rate and poor performance, to improve our model, I decided to normalize my data and repeat the process.

CNN Model Training: The LeNet model was the same as before, but the parameters were adjusted. Model parameters are set to: Learning rate was not at 3e-2, epochs a 10, and batch_size at 15. The code run time was 2m and a training loss of 0.0046.

CNN Model Testing: After the model was tested, the predicted price was plotted against the actual price as shown below. The plot shows that the model significantly improved by normalization.

6. Performance Evaluation and Comparison of Methods

a. **Random Forest Model Evaluation**: Calculating Root Mean Squared Error (RMSE) for our model, the error was 19.476 with 50 trees at a computational speed of 16s.
b. **CNN Model Evaluation**: By evaluating our model prior to and after normalization, we could see that the normalization significantly improved our model with a much larger learning rate. See the plot in the appendix (figure A4 and A5).

Stage	Learning rate	Run time	Loss	RMSE
Before normalization	1e-11	4.09s	3903.3164	62.937
After normalization	1e-2	2m	0.0046	25.44

c. Comparison of Machine Learning Methods

Some additional machine learning methods were implemented using the same dataset and their performance is recorded in the table below. The data was split at 70:30 for train and test data at a random state of 42. SVM had smallest RMSE and kNN and Decision tree were fastest.

Method	**Parameters**	**Runtime**	**RMSE**
Random Forest	50 estimators (trees)	16s	19.476
kNN	k at 70	0s	35.567
Decision Tree	max_depth=10, max_features=None, min_samples_leaf=5, min_samples_split=4	0s	24.756
SVM	Kernel : RBF	3m	0.639
NN	Optimiser=rmsprop, epochs=100, batch_size=50, activation=relu(input) & linear(output) , loss(training)=17.373, loss(testing)=17.700	10m 30s	24.469

d. Comparison of Deep Learning Methods

Some additional deep learning methods were implemented using the same dataset and their performance is recorded in the table below. The data was split at 70:30 for train and test data at a random state of 42. ANN had lowest RMSE while CNN was fastest.

Method	**Learning rate**	**Epochs**	**Batch size**	**Activation function in output layer**	**Run time**	**Loss**	**RMSE**
CNN	1e-2	10	15	Relu	33s	0.0046	25.344
ANN	1e-8	10	20	Relu	17m	2.4746	1.573
DANN	6e-2	200	16	tanh	7m 12s	0.0038	23.141

e. Comparing by methods by RMSE and Runtime: The plots below (Figure 10 & 11) compare the models by their RMSE and runtime. The SVM model was observed to have the lowest RMSE value while Decision Tree and kNN had the shortest runtime.

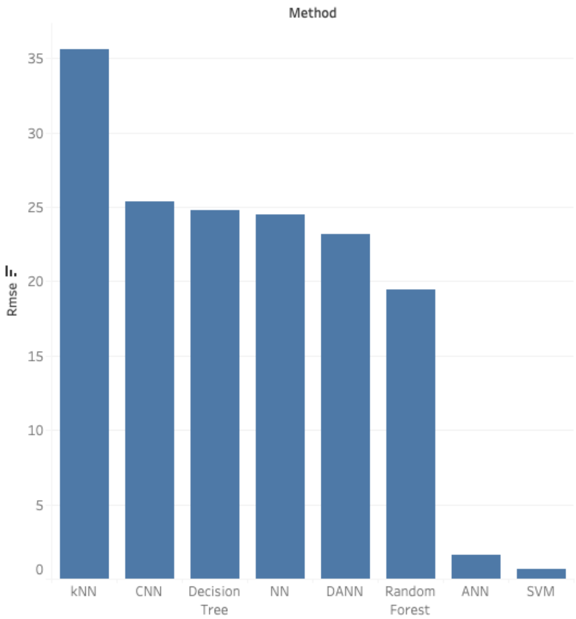


Figure 10: Comparison of methods by RMSE

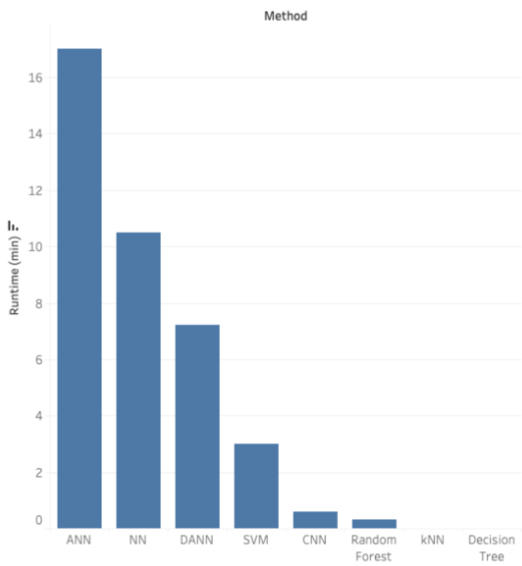


Figure 11: Comparison of methods by the runtime

Comparison of Results: Comparing both machine learning and deep learning methods, the machine learning methods seem to be more accurate for regression predictions. Machine learning methods were most accurate (SVM- lowest RMSE) and fast (Decision tree and kNN). The prediction methods were also compared by plots in the Appendix - Figures A7 to A14. The scatter plots of actual price against prediction show SVM to give the best correlation and kNN shows the least correlation just like the RMSE values signify.

7. Discussion of the findings

I discovered that the quality of our data decides how well our model performs. Comparing our model before and after one significant outlier was taken out showed some good improvement.
I had some challenges implementing Random Forest (RF) regressor. One was the long computational time using R. R could not compute this quickly with the data size, but it did faster with a subset of the data. R in the terminal worked better and computed the model faster for the entire dataset. And Python worked best with a much faster computational time.
Python also seemed easy to use for computing Random Forest and Decision trees because I did not have to specify a "method" which I encountered errors working with R.
For the deep learning prediction, the CNN model did not seem to improve significantly through pruning, but it was much better after normalization.
Some methods were more suited for classification problems (eg. Decision tree and kNN) although we were able to apply them to our regression research question.

Summary of Findings

S/N	Method	Values	Limitations
1	Random Forest	Efficient for high dimensional data	Complex, not easily explained and has high computational speed.
2	k-NN	Fast in training and is simple (trained in 0s)	Low accuracy (produced largest RMSE)
3	Decision Tree	Transparent model, fast computation	Sensitive to small perturbations, tendency for overfitting
4	Neural Network (NN)	Efficient for complex data	Not easily explained
5	Support Vector Machine	Very high accuracy (was the best model with lowest RMSE)	Slow to train for large datasets, not easily explained
6	Convolution Neural Network	Efficient for complex data	Computationally intense and difficult to explain. Required many steps for training. More suitable for images.
7	Deep Artificial NN	Efficient for complex data	Complex and difficult to explain.

8. Author Contribution statement

All the steps for data cleaning and EDA was implemented together as a group of three (3) and although the prediction methods were shared, we held regular meetings to discuss challenges faced, resolve coding issues and discussed results.

S/N	Contribution	Author(s)
1	Conception and design	Ukamaka.O., Sybil.M. and Ceren. H.
2	Acquisition of data	Ukamaka.O., Sybil.M. and Ceren. H.
3	Analysis and interpretation of data	Ukamaka.O., Sybil.M. and Ceren. H.
4	Evaluation of data	Ukamaka.O., Sybil.M. and Ceren. H.
5	Visualisation of data	Ukamaka.O., Sybil.M. and Ceren. H.
6	Random Forest Prediction	Ukamaka.O.
7	K Nearest Neighbour Prediction	Ukamaka.O.
8	Decision Tree Prediction	Sybil.M.
9	Neural Network	Sybil.M.
10	Support Vector Machine (SVM)	Ceren. H.
11	Convolution Neural Network Prediction	Ukamaka.O.
12	Artificial Neural Network (ANN)	Ceren. H.
13	Deep Artificial Neural Network (DANN)	Sybil.M.
14	Discussion of prediction results	Ukamaka.O., Sybil.M. and Ceren. H.
15	Discussion of DMP	Ukamaka.O., Sybil.M. and Ceren. H.

Data Management Plan for Research Students

1. Overview

Researcher: Ukamaka Nkechi Oragwu, Sybil Abrema Meselebe, Ceren Haydaroglu
Project title: CS5812 - Predictive Data Analysis Coursework (Group 4)
Project duration: 4 months (January to April 2023)
Project context: Predictive Data Analysis combines the assessment for Deep Learning and Machine Learning modules for Artificial Intelligence Master's Program. The aim of this assignment is to generate value and insight from the processing of heterogenous data. This will be done by implementing several methods/techniques/algorithms, evaluating them and comparing the effectiveness of the adopted models.

2. Defining your data/research sources

2.1 Where will your data/research sources come from? Our data is publicly available online on the Kaggle website at https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset
2.2 How often will you get new data? We only need to download the data once.

2.3 How much data/information will you generate?
The data is 3.2 MB. The randomized data is still 3.2MB. No more changes will be done on the data.
2.4 What file formats will you use?
The data will be used as a csv file.

3. Organising your data

3.1How will you structure and name your folders and files?
My folders are named by "Course_code Title Date". Updated files bear new date. Data files have 'hotel reservation' prefixing them.
3.2 What additional information is required to understand each data file?
The final data being used has been processed and cleaned in RStudio and is a data dictionary included in this report.
3.3 What different versions of each data file or source will you create?
The original data set has 19 columns (same as retrieved version), this is then worked on several times and randomized, outliers removed and transformed using principal components.

4. Looking after your data

4.1 Where will you store your data?
The data will be stored primarily on a local drive and backed up to the Cloud.

4.2 How will your data be backed up? The data will be backed up to the cloud and new updates saved automatically.
4.3 How will you test whether you can restore from your backups? Updates will be backed to cloud in real time. An alternative method would be to my iCloud from a mobile phone to check if the update is present.

5. Sharing your data

5.1 Who owns the data you generate? This is publicly available dataset and as such the updated version will be available to all members of the group and teaching team for CS5812.
5.2 Who else has a right to see or use this data? Module coordinators and group members.
5.3 Who else should reasonably have access to this data when you share it? There is no intention of sharing this data.
5.4 What should/shouldn't be shared and why? This coursework is only for assessment purposes and should not be shared.

6. Archiving your data

6.1 What should be archived beyond the end of your project? Everything should be archived.

6.2 For how long should it be stored? For as long as needed.
6.3 When will files be moved into the data archive/repository? Upon completion of the analysis of each file it is automatically archived. The marker archives after marking is also completed.
6.4 Where will the data be stored? Cloud storage and Brunel University data repository.
6.5 Who is responsible for moving data to the data archive and maintaining it? Group members will move data to cloud while module coordinator moves to Brunel University data repository.
6.6 Who should have access and under what conditions? As permitted by the school authorities.

7. Executing your plan

7.1 Who is responsible for making sure this plan is followed? The module coordinator is responsible, and group 4 members will follow through as well.
7.2 How often will this plan be reviewed and updated? There will be no updates once submitted.
7.3 What actions have you identified from the rest of this plan? *N/A*
7.4 What further information do you need to carry out these actions? *N/A*

Appendix 1- Figures

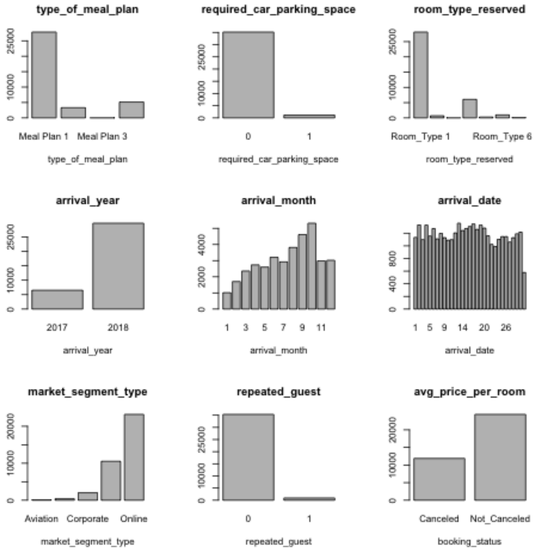


Figure A1: Individual visualization of categorical

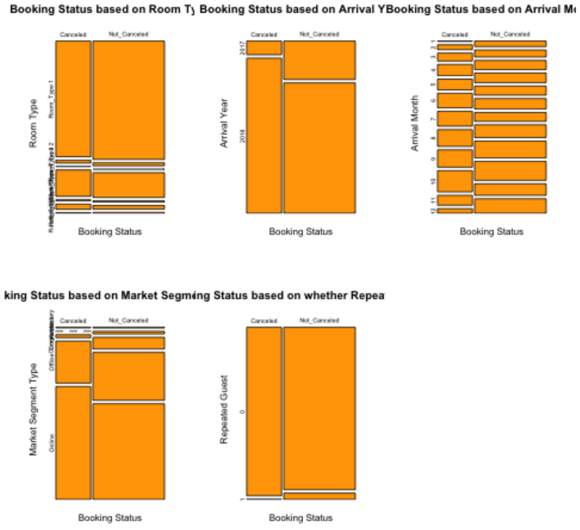


Figure A2: Booking status with other variables.

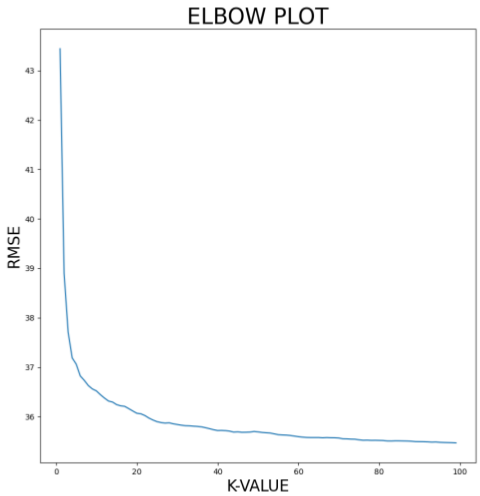


Figure A3: Elbow plot for kNN hyperparameter Tuning

```
[ ] import numpy as np #helps for array operation
    import pandas as pd #helps to read the data
    import matplotlib.pyplot as plt #helps with graphical plots
    from sklearn.model_selection import train_test_split #helps to split training data and testing data
    from sklearn.preprocessing import StandardScaler #helps for standardation of input data
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.metrics import mean_squared_error # for calculating the cost function
    import keras            #helps for CNN model construction
    import tensorflow as tf #helps for CNN model construction
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.metrics import r2_score
    %matplotlib inline
```

Figure A4: Libraries Loaded for Modelling

CNN MODEL PREDICTION BEFORE AND AFTER NORMALIZATION

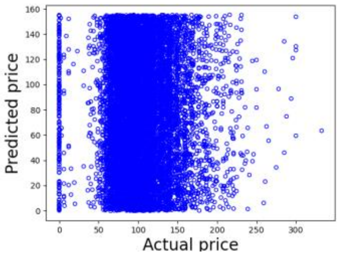


Figure A5: CNN Performance Before Normalization

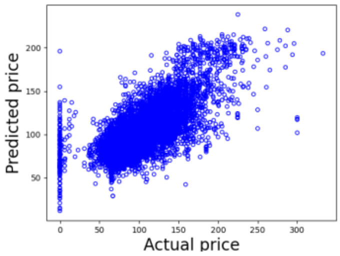


Figure A6: CNN Performance After Normalization

COMPARISON OF MODELS BY PREDICTION PLOTS

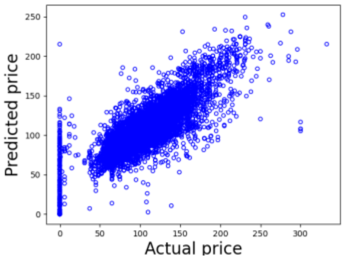


Figure A7: RF Prediction

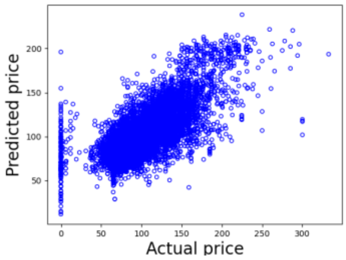


Figure A8: CNN Prediction

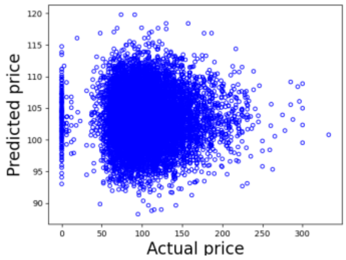


Figure A9: kNN Prediction

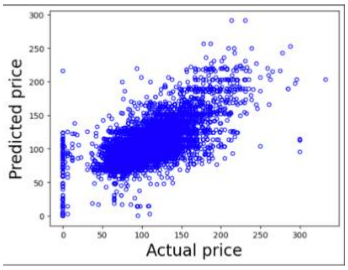


Figure A10: Decision Tree Prediction

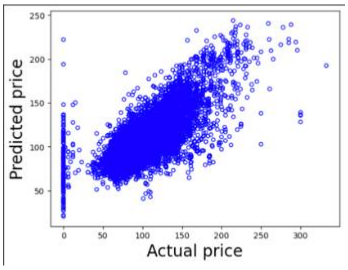


Figure A11: Neural Network Prediction

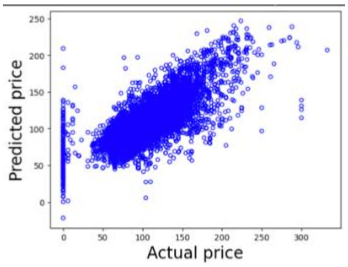


Figure A12: DANN Prediction

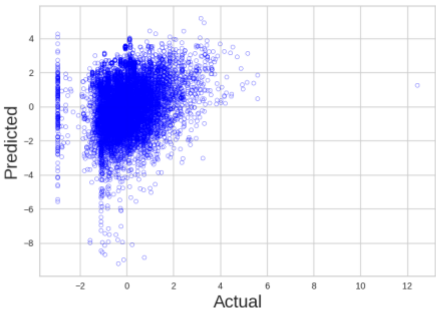


Figure A13: ANN Prediction

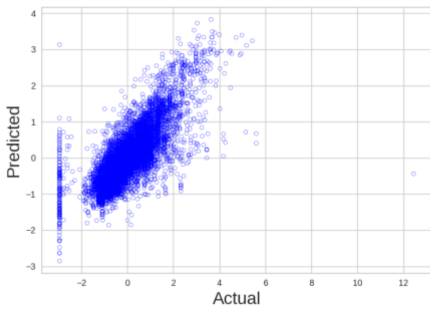


Figure A14: SVM Prediciton

Appendix 2- Software - CS5812 Predictive Data Analysis Group work

1. Load and use Packages

2. Load Data and View Data

3. Data Preparation- Quality check and Cleaning

4. Data Transoformation, Feature extraction/selection

5. Data Exploration- Statistical, Graphical, Principal Component Analysis

#1. Load and use Packages Loading necessary libraries

```
library(ggplot2)
library(validate)

##
## Attaching package: 'validate'

## The following object is masked from 'package:ggplot2':
##
##     expr
```

2. Load and View Data

```
hotel_reservation <- read.csv("hotel reservation randomised.csv")

#making hotel_reservation data a data frame
hotel_reservation <- data.frame(hotel_reservation)

#Viewing the data
head(hotel_reservation)

##   Booking_ID no_of_adults no_of_children no_of_weekend_nights no_of_wee
k_nights
## 1   INN00001            2              0                    1
2
## 2   INN00002            2              0                    2
3
## 3   INN00003            1              0                    2
1
## 4   INN00004            2              0                    0
2
## 5   INN00005            2              0                    1
1
## 6   INN00006            2              0                    0
2
##   type_of_meal_plan required_car_parking_space room_type_reserved lead_
time
## 1       Meal Plan 1                          0        Room_Type 1
```

```
224
## 2      Not Selected                          0        Room_Type 1
5
## 3       Meal Plan 1                          0        Room_Type 1
1
## 4       Meal Plan 1                          0        Room_Type 1
211
## 5      Not Selected                          0        Room_Type 1
48
## 6       Meal Plan 2                          0        Room_Type 1
346
##   arrival_year arrival_month arrival_date market_segment_type repeated_
guest
## 1         2017            10            2             Offline
0
## 2         2018            11            6              Online
0
## 3         2018             2           28              Online
0
## 4         2018             5           20              Online
0
## 5         2018             4           11              Online
0
## 6         2018             9           13              Online
0
##   no_of_previous_cancellations no_of_previous_bookings_not_canceled
## 1                            0                                    0
## 2                            0                                    0
## 3                            0                                    0
## 4                            0                                    0
## 5                            0                                    0
## 6                            0                                    0
##   no_of_special_requests booking_status avg_price_per_room
## 1                      0   Not_Canceled              65.00
## 2                      1   Not_Canceled             106.68
## 3                      0       Canceled              60.00
## 4                      0       Canceled             100.00
## 5                      0       Canceled              94.50
## 6                      1       Canceled             115.00
```

```
summary(hotel_reservation)
```

```
##   Booking_ID         no_of_adults   no_of_children    no_of_weekend_nig
hts
##  Length:36275       Min.   :0.000   Min.   : 0.0000   Min.   :0.0000
##  Class :character   1st Qu.:2.000   1st Qu.: 0.0000   1st Qu.:0.0000
##  Mode  :character   Median :2.000   Median : 0.0000   Median :1.0000
##                     Mean   :1.845   Mean   : 0.1053   Mean   :0.8107
##                     3rd Qu.:2.000   3rd Qu.: 0.0000   3rd Qu.:2.0000
##                     Max.   :4.000   Max.   :10.0000   Max.   :7.0000
##
##  no_of_week_nights type_of_meal_plan  required_car_parking_space
##  Min.   : 0.000    Length:36275       Min.   :0.00000
##  1st Qu.: 1.000    Class :character   1st Qu.:0.00000
##  Median : 2.000    Mode  :character   Median :0.00000
##  Mean   : 2.204                       Mean   :0.03099
```

```
##  3rd Qu.: 3.000                       3rd Qu.:0.00000
##  Max.   :17.000                       Max.   :1.00000
##
##  room_type_reserved   lead_time       arrival_year  arrival_month
##  Length:36275       Min.   :  0.00   Min.   :2017   Min.   : 1.000
##  Class :character   1st Qu.: 17.00   1st Qu.:2018   1st Qu.: 5.000
##  Mode  :character   Median : 57.00   Median :2018   Median : 8.000
##                     Mean   : 85.23   Mean   :2018   Mean   : 7.424
##                     3rd Qu.:126.00   3rd Qu.:2018   3rd Qu.:10.000
##                     Max.   :443.00   Max.   :2018   Max.   :12.000
##
##   arrival_date  market_segment_type repeated_guest
##  Min.   : 1.0   Length:36275        Min.   :0.00000
##  1st Qu.: 8.0   Class :character    1st Qu.:0.00000
##  Median :16.0   Mode  :character    Median :0.00000
##  Mean   :15.6                       Mean   :0.02564
##  3rd Qu.:23.0                       3rd Qu.:0.00000
##  Max.   :31.0                       Max.   :1.00000
##
##  no_of_previous_cancellations no_of_previous_bookings_not_canceled
##  Min.   : 0.00000             Min.   : 0.0000
##  1st Qu.: 0.00000             1st Qu.: 0.0000
##  Median : 0.00000             Median : 0.0000
##  Mean   : 0.02335             Mean   : 0.1534
##  3rd Qu.: 0.00000             3rd Qu.: 0.0000
##  Max.   :13.00000             Max.   :58.0000
##
##  no_of_special_requests booking_status     avg_price_per_room
##  Min.   :0.0000         Length:36275       Min.   :  0.00
##  1st Qu.:0.0000         Class :character   1st Qu.: 80.30
##  Median :0.0000         Mode  :character   Median : 99.45
##  Mean   :0.6197                            Mean   :103.42
##  3rd Qu.:1.0000                            3rd Qu.:120.00
##  Max.   :5.0000                            Max.   :540.00
##                                            NA's   :1
```

```
str(hotel_reservation)
```

```
## 'data.frame':    36275 obs. of  19 variables:
##  $ Booking_ID                          : chr  "INN00001" "INN00002" "IN
N00003" "INN00004" ...
##  $ no_of_adults                        : int  2 2 1 2 2 2 2 2 3 2 ...
##  $ no_of_children                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_weekend_nights                : int  1 2 2 0 1 0 1 1 0 0 ...
##  $ no_of_week_nights                   : int  2 3 1 2 1 2 3 3 4 5 ...
##  $ type_of_meal_plan                   : chr  "Meal Plan 1" "Not Select
ed" "Meal Plan 1" "Meal Plan 1" ...
##  $ required_car_parking_space          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ room_type_reserved                  : chr  "Room_Type 1" "Room_Type
1" "Room_Type 1" "Room_Type 1" ...
##  $ lead_time                           : int  224 5 1 211 48 346 34 83
121 44 ...
##  $ arrival_year                        : int  2017 2018 2018 2018 2018
2018 2017 2018 2018 2018 ...
##  $ arrival_month                       : int  10 11 2 5 4 9 10 12 7 10
...
```

```
##  $ arrival_date                        : int  2 6 28 20 11 13 15 26 6 1
8 ...
##  $ market_segment_type                 : chr  "Offline" "Online" "Onlin
e" "Online" ...
##  $ repeated_guest                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_cancellations        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_bookings_not_canceled: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_special_requests              : int  0 1 0 0 0 1 1 1 1 3 ...
##  $ booking_status                      : chr  "Not_Canceled" "Not_Cance
led" "Canceled" "Canceled" ...
##  $ avg_price_per_room                  : num  65 106.7 60 100 94.5 ...
```

The variables of our data set were read in correctly except for the following: - type_of_meal_plan - room_type_reserved - market_segment_type - booking_status They were read in as characters instead of factors, however, these variables will be recoded to integers during data transformation so we will leave them as characters for now.

3. Quality Check and Cleaning

a. Detecting missing values
b. Detecting Duplicates
c. Data Validation
d. Data Cleaning

- Dealing with missing values
- Dealing with duplicates
- (Simple) outlier detection

```
#a. Detecting missing values
colSums(is.na(hotel_reservation))

##                           Booking_ID                                 no_of_adul
ts
##                                    0                                           
0
##                       no_of_children                         no_of_weekend_nigh
ts
##                                    0                                           
0
##                     no_of_week_nights                            type_of_meal_pl
an
##                                    0                                           
0
##            required_car_parking_space                          room_type_reserv
ed
##                                    0                                           
0
##                            lead_time                                 arrival_ye
ar
##                                    0                                           
0
##                        arrival_month                                 arrival_da
te
##                                    0                                           
0
##                  market_segment_type                             repeated_gue
```

```
st
##                                    0
0
##           no_of_previous_cancellations  no_of_previous_bookings_not_cancel
ed
##                                    0
0
##                 no_of_special_requests                        booking_stat
us
##                                    0
0
##                     avg_price_per_room
##                                    1
```

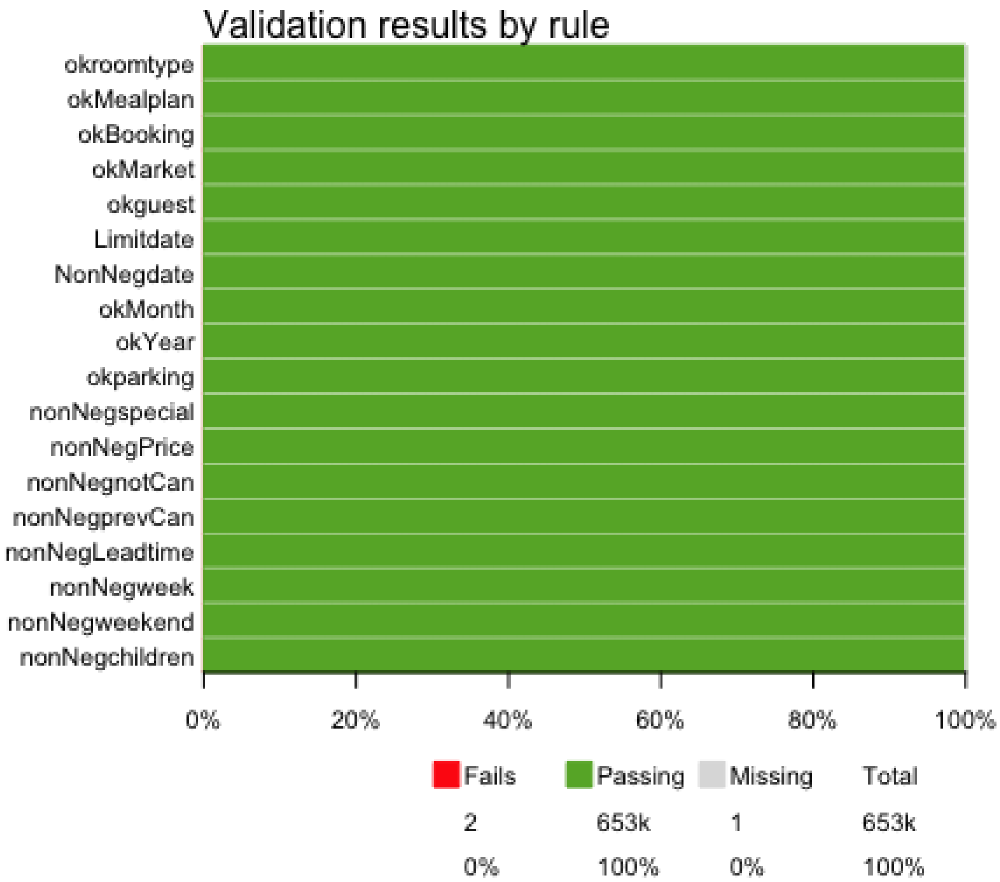
```
#b. Detecting duplicate values
dim(hotel_reservation)

## [1] 36275    19

dim(unique(hotel_reservation))

## [1] 36275    19

sum(duplicated(hotel_reservation))

## [1] 0
```

Missing Values: We recorded 1 missing value in avg_price_per_room variable. Duplicate Instances: There are no duplicates in our dataset.

```
#c. Validation Process

Mydf.Rules <- validator(
  nonNegchildren = no_of_children >=0,
  nonNegweekend = no_of_weekend_nights>=0,
  nonNegweek = no_of_week_nights>=0,
  nonNegLeadtime = lead_time>=0,
  nonNegprevCan = no_of_previous_cancellations>=0,
  nonNegnotCan = no_of_previous_bookings_not_canceled>=0,
  nonNegPrice = avg_price_per_room>=0,
  nonNegspecial = no_of_special_requests>=0,
  okMealplan= is.element(type_of_meal_plan,c("Not Selected","Meal Plan 1",
"Meal Plan 2", "Meal Plan 3")),
  okparking= is.element(required_car_parking_space,c("0","1")),
  okroomtype= is.element(room_type_reserved,c("Room_Type 1","Room_Type 2",
"Room_Type 3", "Room_Type 4", "Room_Type 5","Room_Type 6", "Room_Type 7"))
,
  okYear = arrival_year >= 2017 & arrival_year <=2018,
  okMonth = arrival_month> 0 & arrival_month <=12,
  NonNegdate = arrival_date>0,
  Limitdate = arrival_date<=31,
  okguest= is.element(repeated_guest,c("0","1")),
  okMarket = is.element(market_segment_type, c("Aviation", "Complementary"
, "Corporate", "Offline", "Online")),
  okBooking = is.element(booking_status, c("Canceled","Not_Canceled")))
```

```
qual.check <- confront(hotel_reservation,Mydf.Rules)
summary(qual.check)

##                name items passes fails nNA error warning
## 1  nonNegchildren 36275  36275     0   0 FALSE   FALSE
## 2   nonNegweekend 36275  36275     0   0 FALSE   FALSE
## 3      nonNegweek 36275  36275     0   0 FALSE   FALSE
## 4  nonNegLeadtime 36275  36275     0   0 FALSE   FALSE
## 5   nonNegprevCan 36275  36275     0   0 FALSE   FALSE
## 6    nonNegnotCan 36275  36275     0   0 FALSE   FALSE
## 7     nonNegPrice 36275  36274     0   1 FALSE   FALSE
## 8   nonNegspecial 36275  36275     0   0 FALSE   FALSE
## 9      okMealplan 36275  36274     1   0 FALSE   FALSE
## 10      okparking 36275  36275     0   0 FALSE   FALSE
## 11     okroomtype 36275  36274     1   0 FALSE   FALSE
## 12         okYear 36275  36275     0   0 FALSE   FALSE
## 13        okMonth 36275  36275     0   0 FALSE   FALSE
## 14     NonNegdate 36275  36275     0   0 FALSE   FALSE
## 15      Limitdate 36275  36275     0   0 FALSE   FALSE
## 16        okguest 36275  36275     0   0 FALSE   FALSE
## 17       okMarket 36275  36275     0   0 FALSE   FALSE
## 18      okBooking 36275  36275     0   0 FALSE   FALSE
##
expression
## 1
no_of_children - 0 >= -1e-08
## 2
no_of_weekend_nights - 0 >= -1e-08
## 3
no_of_week_nights - 0 >= -1e-08
## 4
lead_time - 0 >= -1e-08
## 5
no_of_previous_cancellations - 0 >= -1e-08
## 6
no_of_previous_bookings_not_canceled - 0 >= -1e-08
## 7
avg_price_per_room - 0 >= -1e-08
## 8
no_of_special_requests - 0 >= -1e-08
## 9                                               is.element(type_of_meal
_plan, c("Not Selected", "Meal Plan 1", "Meal Plan 2", "Meal Plan 3"))
## 10
is.element(required_car_parking_space, c("0", "1"))
## 11 is.element(room_type_reserved, c("Room_Type 1", "Room_Type 2", "Room
_Type 3", "Room_Type 4", "Room_Type 5", "Room_Type 6", "Room_Type 7"))
## 12
arrival_year - 2017 >= -1e-08 & arrival_year - 2018 <= 1e-08
## 13
arrival_month > 0 & arrival_month - 12 <= 1e-08
## 14
arrival_date > 0
## 15
arrival_date - 31 <= 1e-08
## 16
is.element(repeated_guest, c("0", "1"))
```

```
## 17                                            is.element(market_segment_t
ype, c("Aviation", "Complementary", "Corporate", "Offline", "Online"))
## 18
is.element(booking_status, c("Canceled", "Not_Canceled"))

plot(qual.check, xlab="")
```



Here we see that there are 2 rules that failed our validation test and these are in the room_type_reserved and meal_plan_type variables. And 1 missing value in the avg_room_price variable.

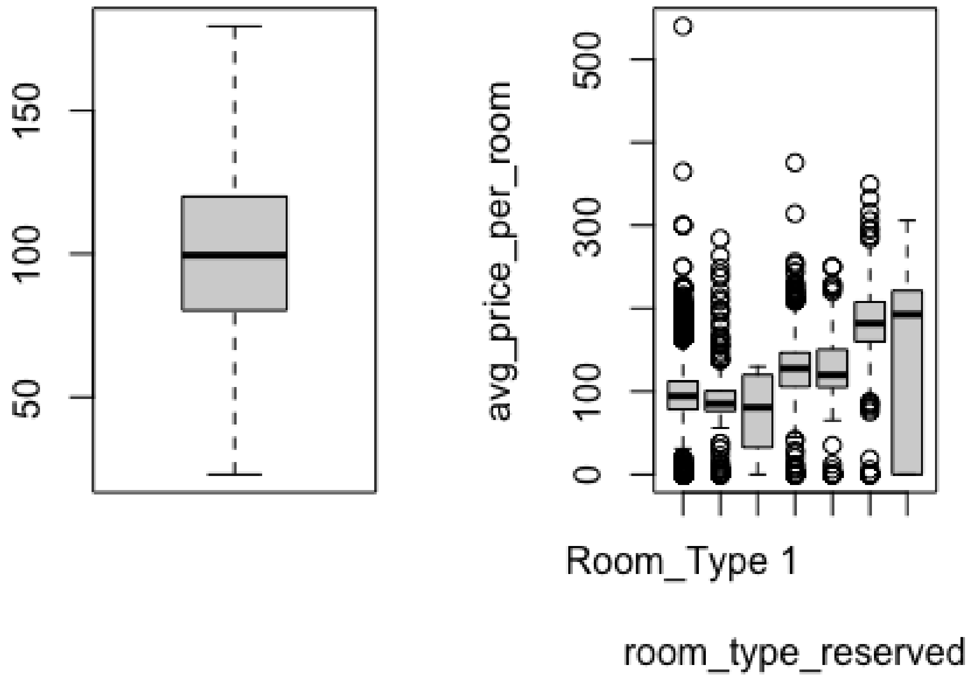
```
#investigating the failure in room_type_reserved and meal_plan_type
table(hotel_reservation$type_of_meal_plan)

##
##   Meal Plan 1  Meal Plan 2  Meal Plan 3   MealPlan 1 Not Selected
##         27834         3305            5            1         5130

table(hotel_reservation$room_type_reserved)

##
## Room_Type 1 Room_Type 2 Room_Type 3 Room_Type 4 Room_Type 5 Room_Type 6
##       28129         692           7        6057         265         966
## Room_Type 7  RoomType 1
##         158           1
```

We can see that there was a wrong spelling for Meal Plan 1 as MealPlan 1, and Room_Type 1 was misspelt as RoomType1.

```r
# d. Data Cleaning

# Fixing the wrong spelling
hotel_reservation$room_type_reserved[hotel_reservation$room_type_reserved
== "RoomType 1"] <- "Room_Type 1"
table(hotel_reservation$room_type_reserved)

##
## Room_Type 1 Room_Type 2 Room_Type 3 Room_Type 4 Room_Type 5 Room_Type 6
##       28130         692           7        6057         265         966
## Room_Type 7
##         158

hotel_reservation$type_of_meal_plan[hotel_reservation$type_of_meal_plan ==
"MealPlan 1"] <- "Meal Plan 1"
table(hotel_reservation$type_of_meal_plan)

##
##  Meal Plan 1  Meal Plan 2  Meal Plan 3 Not Selected
##        27835         3305            5         5130

# Fixing the missing value
hotel_reservation$avg_price_per_room <- as.numeric(hotel_reservation$avg_p
rice_per_room)
hotel_reservation$avg_price_per_room[hotel_reservation$avg_price_per_room
== " "] <- NA #Recoding missing value

hotel_reserve_noNA <- hotel_reservation #Creating new data frame before im
puting
hotel_reserve_noNA$avg_price_per_room[is.na(hotel_reserve_noNA$avg_price_p
er_room)] <- median(hotel_reserve_noNA$avg_price_per_room, na.rm = T)
summary(hotel_reserve_noNA$avg_price_per_room)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   80.30   99.45  103.42  120.00  540.00
```

Our variables are correctly represented now

```r
# (Simple) Outlier Detection for the target variable

# inspect the Fare distribution using summary statistics
summary(hotel_reserve_noNA$avg_price_per_room)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   80.30   99.45  103.42  120.00  540.00

# generate a boxplot of the avg_price_per_room variable
#png(file = "hotelreserve boxplot_price.png")
opar <- par(no.readonly = TRUE)
par(mfrow = c(1,2))
boxplot(hotel_reserve_noNA$avg_price_per_room, outline=FALSE)
boxplot(avg_price_per_room ~ room_type_reserved, data = hotel_reserve_noNA
)
```




```
par(opar)
#dev.off()
```

We do not see any outliers in the average room price alone but when compared with other variables we see that one price instance is significantly different from the rest and may be skewing the data. We will take a closer look at this outlier in the EDA before deciding if to take it out.

#3.Feature Selection/extraction The booking_ID column does not add significant information to our dataset so we will be dropping this column

```
#dropping booking_ID column
hotel_reserve_noNA <- hotel_reserve_noNA[,-1]
head(hotel_reserve_noNA)

##   no_of_adults no_of_children no_of_weekend_nights no_of_week_nights
## 1            2              0                    1                 2
## 2            2              0                    2                 3
## 3            1              0                    2                 1
## 4            2              0                    0                 2
## 5            2              0                    1                 1
## 6            2              0                    0                 2
##   type_of_meal_plan required_car_parking_space room_type_reserved lead_
time
## 1       Meal Plan 1                          0        Room_Type 1
224
## 2      Not Selected                          0        Room_Type 1
5
## 3       Meal Plan 1                          0        Room_Type 1
```

```
1
## 4       Meal Plan 1                          0         Room_Type 1
211
## 5      Not Selected                          0         Room_Type 1
48
## 6       Meal Plan 2                          0         Room_Type 1
346
##   arrival_year arrival_month arrival_date market_segment_type repeated_
guest
## 1         2017            10            2             Offline
0
## 2         2018            11            6              Online
0
## 3         2018             2           28              Online
0
## 4         2018             5           20              Online
0
## 5         2018             4           11              Online
0
## 6         2018             9           13              Online
0
##   no_of_previous_cancellations no_of_previous_bookings_not_canceled
## 1                            0                                    0
## 2                            0                                    0
## 3                            0                                    0
## 4                            0                                    0
## 5                            0                                    0
## 6                            0                                    0
##   no_of_special_requests booking_status avg_price_per_room
## 1                      0   Not_Canceled              65.00
## 2                      1   Not_Canceled             106.68
## 3                      0       Canceled              60.00
## 4                      0       Canceled             100.00
## 5                      0       Canceled              94.50
## 6                      1       Canceled             115.00
```

#4. Exploratory Data Analysis a. Statistical Exploration

```
summary(hotel_reserve_noNA) #summary of our cleaned data

##   no_of_adults   no_of_children    no_of_weekend_nights no_of_week_nigh
ts
##  Min.   :0.000   Min.   : 0.0000   Min.   :0.0000       Min.   : 0.000
##  1st Qu.:2.000   1st Qu.: 0.0000   1st Qu.:0.0000       1st Qu.: 1.000
##  Median :2.000   Median : 0.0000   Median :1.0000       Median : 2.000
##  Mean   :1.845   Mean   : 0.1053   Mean   :0.8107       Mean   : 2.204
##  3rd Qu.:2.000   3rd Qu.: 0.0000   3rd Qu.:2.0000       3rd Qu.: 3.000
##  Max.   :4.000   Max.   :10.0000   Max.   :7.0000       Max.   :17.000
##  type_of_meal_plan  required_car_parking_space room_type_reserved
##  Length:36275       Min.   :0.00000           Length:36275
##  Class :character   1st Qu.:0.00000           Class :character
##  Mode  :character   Median :0.00000           Mode  :character
##                     Mean   :0.03099
##                     3rd Qu.:0.00000
##                     Max.   :1.00000
##    lead_time       arrival_year  arrival_month     arrival_date
```

```
##  Min.   :  0.00   Min.   :2017   Min.   : 1.000   Min.   : 1.0
##  1st Qu.: 17.00   1st Qu.:2018   1st Qu.: 5.000   1st Qu.: 8.0
##  Median : 57.00   Median :2018   Median : 8.000   Median :16.0
##  Mean   : 85.23   Mean   :2018   Mean   : 7.424   Mean   :15.6
##  3rd Qu.:126.00   3rd Qu.:2018   3rd Qu.:10.000   3rd Qu.:23.0
##  Max.   :443.00   Max.   :2018   Max.   :12.000   Max.   :31.0
##  market_segment_type repeated_guest    no_of_previous_cancellations
##  Length:36275        Min.   :0.00000   Min.   : 0.00000
##  Class :character    1st Qu.:0.00000   1st Qu.: 0.00000
##  Mode  :character    Median :0.00000   Median : 0.00000
##                      Mean   :0.02564   Mean   : 0.02335
##                      3rd Qu.:0.00000   3rd Qu.: 0.00000
##                      Max.   :1.00000   Max.   :13.00000
##  no_of_previous_bookings_not_canceled no_of_special_requests booking_st
atus
##  Min.   : 0.0000                      Min.   :0.0000         Length:362
75
##  1st Qu.: 0.0000                      1st Qu.:0.0000         Class :cha
racter
##  Median : 0.0000                      Median :0.0000         Mode  :cha
racter
##  Mean   : 0.1534                      Mean   :0.6197
##  3rd Qu.: 0.0000                      3rd Qu.:1.0000
##  Max.   :58.0000                      Max.   :5.0000
##  avg_price_per_room
##  Min.   :  0.00
##  1st Qu.: 80.30
##  Median : 99.45
##  Mean   :103.42
##  3rd Qu.:120.00
##  Max.   :540.00
```

```
str(hotel_reserve_noNA)
```

```
## 'data.frame':    36275 obs. of  18 variables:
##  $ no_of_adults                        : int  2 2 1 2 2 2 2 2 3 2 ...
##  $ no_of_children                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_weekend_nights                : int  1 2 2 0 1 0 1 1 0 0 ...
##  $ no_of_week_nights                   : int  2 3 1 2 1 2 3 3 4 5 ...
##  $ type_of_meal_plan                   : chr  "Meal Plan 1" "Not Select
ed" "Meal Plan 1" "Meal Plan 1" ...
##  $ required_car_parking_space          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ room_type_reserved                  : chr  "Room_Type 1" "Room_Type
1" "Room_Type 1" "Room_Type 1" ...
##  $ lead_time                           : int  224 5 1 211 48 346 34 83
121 44 ...
##  $ arrival_year                        : int  2017 2018 2018 2018 2018
2018 2017 2018 2018 2018 ...
##  $ arrival_month                       : int  10 11 2 5 4 9 10 12 7 10
...
##  $ arrival_date                        : int  2 6 28 20 11 13 15 26 6 1
8 ...
##  $ market_segment_type                 : chr  "Offline" "Online" "Onlin
e" "Online" ...
##  $ repeated_guest                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_cancellations        : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ no_of_previous_bookings_not_canceled: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_special_requests              : int  0 1 0 0 0 1 1 1 1 3 ...
##  $ booking_status                      : chr  "Not_Canceled" "Not_Cance
led" "Canceled" "Canceled" ...
##  $ avg_price_per_room                  : num  65 106.7 60 100 94.5 ...
```

```
head(hotel_reserve_noNA)
```

```
##   no_of_adults no_of_children no_of_weekend_nights no_of_week_nights
## 1            2              0                    1                 2
## 2            2              0                    2                 3
## 3            1              0                    2                 1
## 4            2              0                    0                 2
## 5            2              0                    1                 1
## 6            2              0                    0                 2
##   type_of_meal_plan required_car_parking_space room_type_reserved lead_
time
## 1       Meal Plan 1                          0        Room_Type 1
224
## 2      Not Selected                          0        Room_Type 1
5
## 3       Meal Plan 1                          0        Room_Type 1
1
## 4       Meal Plan 1                          0        Room_Type 1
211
## 5      Not Selected                          0        Room_Type 1
48
## 6       Meal Plan 2                          0        Room_Type 1
346
##   arrival_year arrival_month arrival_date market_segment_type repeated_
guest
## 1         2017            10            2             Offline
0
## 2         2018            11            6              Online
0
## 3         2018             2           28              Online
0
## 4         2018             5           20              Online
0
## 5         2018             4           11              Online
0
## 6         2018             9           13              Online
0
##   no_of_previous_cancellations no_of_previous_bookings_not_canceled
## 1                            0                                    0
## 2                            0                                    0
## 3                            0                                    0
## 4                            0                                    0
## 5                            0                                    0
## 6                            0                                    0
##   no_of_special_requests booking_status avg_price_per_room
## 1                      0   Not_Canceled              65.00
## 2                      1   Not_Canceled             106.68
## 3                      0       Canceled              60.00
## 4                      0       Canceled             100.00
```

```
## 5                         0       Canceled              94.50
## 6                         1       Canceled             115.00
```

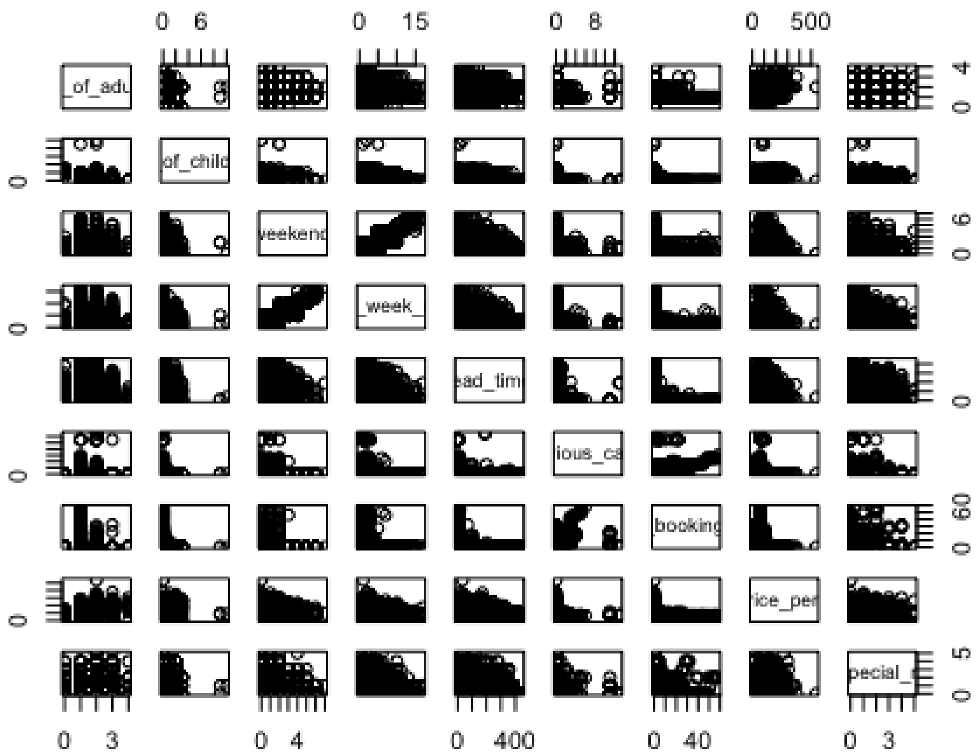
```
# creating label vectors for numerical and categorical variables
hotel_reservation_num <- c("no_of_adults", "no_of_children", "no_of_weeken
d_nights", "no_of_week_nights","lead_time", "no_of_previous_cancellations"
, "no_of_previous_bookings_not_canceled", "avg_price_per_room", "no_of_spe
cial_requests")

hotel_reservation_cat <- c("type_of_meal_plan", "required_car_parking_spac
e", "room_type_reserved",  "market_segment_type","repeated_guest", "bookin
g_status","arrival_date","arrival_year","arrival_month")
```

Visualizing Numerical Data

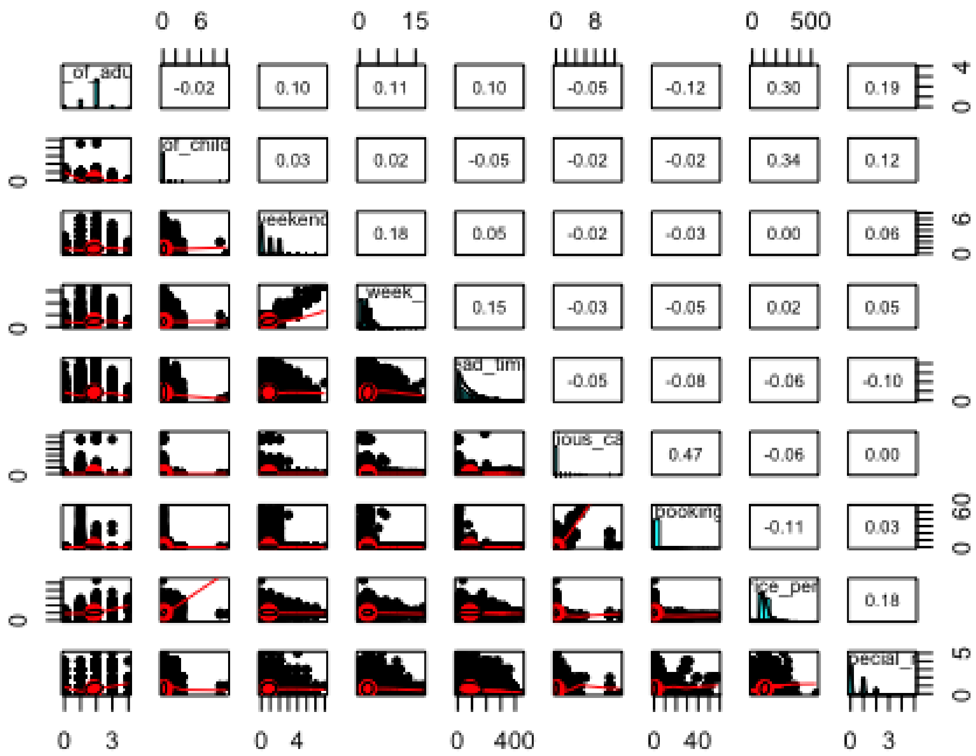
```
# exploring relationships among features: correlation matrix
hotel_reservation_num_cor <- cor(hotel_reserve_noNA[hotel_reservation_num]
)

# visualize the correlation matrix
hotel_reservation_num_cor

##                                      no_of_adults no_of_children
## no_of_adults                           1.00000000    -0.01978707
## no_of_children                        -0.01978707     1.00000000
## no_of_weekend_nights                   0.10331578     0.02947758
## no_of_week_nights                      0.10562190     0.02439811
## lead_time                              0.09728651    -0.04709128
## no_of_previous_cancellations          -0.04742575    -0.01638958
## no_of_previous_bookings_not_canceled  -0.11916579    -0.02118896
## avg_price_per_room                     0.29688259     0.33773135
## no_of_special_requests                 0.18940095     0.12448619
##                                      no_of_weekend_nights no_of_week_ni
ghts
## no_of_adults                                  0.103315775        0.1056
2190
## no_of_children                                0.029477584        0.0243
9811
## no_of_weekend_nights                          1.000000000        0.1795
7676
## no_of_week_nights                             0.179576764        1.0000
0000
## lead_time                                     0.046595440        0.1496
5016
## no_of_previous_cancellations                 -0.020690482       -0.0300
8040
## no_of_previous_bookings_not_canceled         -0.026311984       -0.0493
4374
## avg_price_per_room                           -0.004513731        0.0227
6267
## no_of_special_requests                        0.060592526        0.0459
9365
##                                        lead_time no_of_previous_cancell
ations
## no_of_adults                          0.09728651                 -0.047
425747
```

```
## no_of_children                       -0.04709128                   -0.016
389584
## no_of_weekend_nights                  0.04659544                   -0.020
690482
## no_of_week_nights                     0.14965016                   -0.030
080402
## lead_time                             1.00000000                   -0.045
722982
## no_of_previous_cancellations         -0.04572298                    1.000
000000
## no_of_previous_bookings_not_canceled -0.07813666                    0.468
146833
## avg_price_per_room                   -0.06260275                   -0.063
339719
## no_of_special_requests               -0.10164497                   -0.003
317358
##                                      no_of_previous_bookings_not_cancel
ed
## no_of_adults                                                   -0.119165
79
## no_of_children                                                 -0.021188
96
## no_of_weekend_nights                                           -0.026311
98
## no_of_week_nights                                              -0.049343
74
## lead_time                                                      -0.078136
66
## no_of_previous_cancellations                                    0.468146
83
## no_of_previous_bookings_not_canceled                            1.000000
00
## avg_price_per_room                                             -0.113682
97
## no_of_special_requests                                          0.027376
58
##                                      avg_price_per_room no_of_special_r
equests
## no_of_adults                                0.296882590              0.18
9400951
## no_of_children                              0.337731352              0.12
4486186
## no_of_weekend_nights                       -0.004513731              0.06
0592526
## no_of_week_nights                           0.022762671              0.04
5993653
## lead_time                                  -0.062602751             -0.10
1644974
## no_of_previous_cancellations               -0.063339719             -0.00
3317358
## no_of_previous_bookings_not_canceled       -0.113682967              0.02
7376578
## avg_price_per_room                          1.000000000              0.18
4375523
```

```
## no_of_special_requests                          0.184375523             1.00
0000000

# plot the relationships among features - scatterplot matrix
pairs(hotel_reserve_noNA[hotel_reservation_num])
```




```
# plot a more informative scatterplot matrix
#png(file = "hotelreserve pairs plot.png")
psych::pairs.panels(hotel_reserve_noNA[hotel_reservation_num])
```




```
#dev.off()
```

There are no significant correlations between the numerical variables

```
head(hotel_reserve_noNA)

##   no_of_adults no_of_children no_of_weekend_nights no_of_week_nights
## 1            2              0                    1                 2
## 2            2              0                    2                 3
## 3            1              0                    2                 1
## 4            2              0                    0                 2
## 5            2              0                    1                 1
## 6            2              0                    0                 2
##   type_of_meal_plan required_car_parking_space room_type_reserved lead_
time
## 1       Meal Plan 1                          0        Room_Type 1
224
## 2      Not Selected                          0        Room_Type 1
5
## 3       Meal Plan 1                          0        Room_Type 1
1
## 4       Meal Plan 1                          0        Room_Type 1
211
## 5      Not Selected                          0        Room_Type 1
48
## 6       Meal Plan 2                          0        Room_Type 1
346
##   arrival_year arrival_month arrival_date market_segment_type repeated_
guest
```

```
## 1         2017            10           2            Offline
0
## 2         2018            11           6             Online
0
## 3         2018             2          28             Online
0
## 4         2018             5          20             Online
0
## 5         2018             4          11             Online
0
## 6         2018             9          13             Online
0
##   no_of_previous_cancellations no_of_previous_bookings_not_canceled
## 1                            0                                    0
## 2                            0                                    0
## 3                            0                                    0
## 4                            0                                    0
## 5                            0                                    0
## 6                            0                                    0
##   no_of_special_requests booking_status avg_price_per_room
## 1                      0   Not_Canceled              65.00
## 2                      1   Not_Canceled             106.68
## 3                      0       Canceled              60.00
## 4                      0       Canceled             100.00
## 5                      0       Canceled              94.50
## 6                      1       Canceled             115.00
```

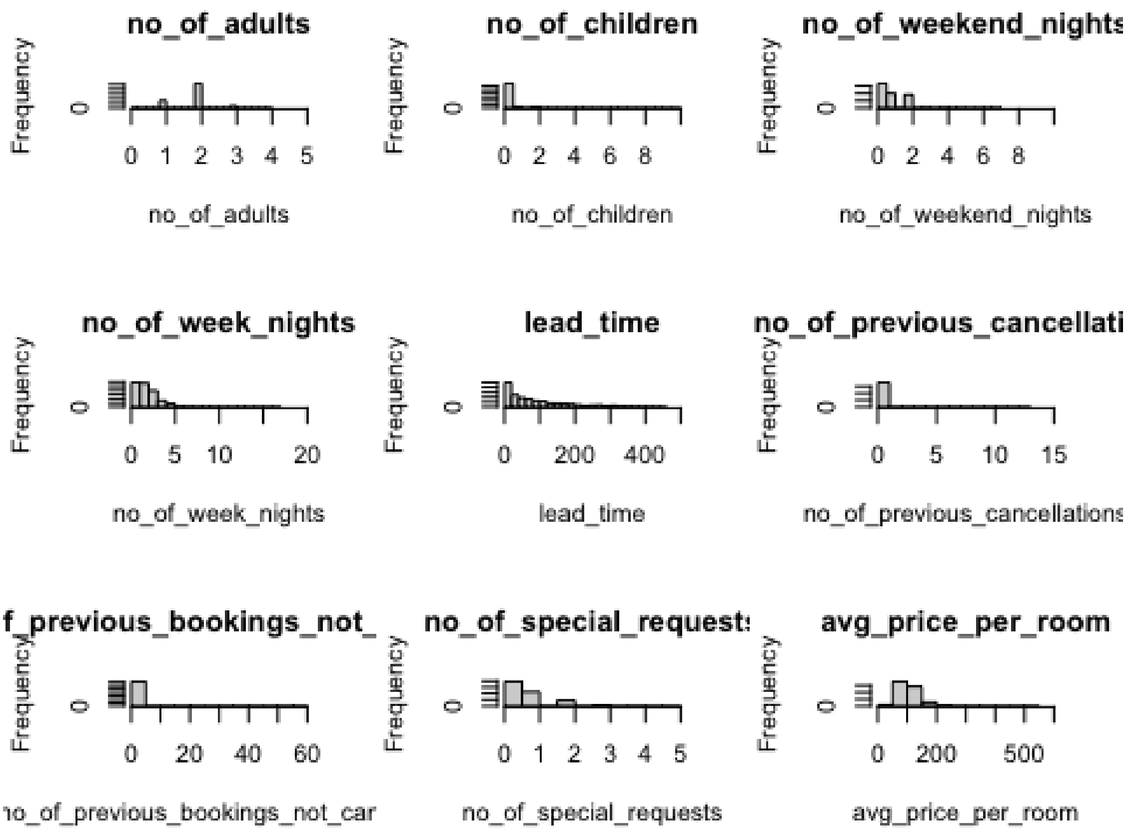
```
summary(hotel_reserve_noNA)
```

```
##   no_of_adults   no_of_children    no_of_weekend_nights no_of_week_nigh
ts
##  Min.   :0.000   Min.   : 0.0000   Min.   :0.0000       Min.   : 0.000
##  1st Qu.:2.000   1st Qu.: 0.0000   1st Qu.:0.0000       1st Qu.: 1.000
##  Median :2.000   Median : 0.0000   Median :1.0000       Median : 2.000
##  Mean   :1.845   Mean   : 0.1053   Mean   :0.8107       Mean   : 2.204
##  3rd Qu.:2.000   3rd Qu.: 0.0000   3rd Qu.:2.0000       3rd Qu.: 3.000
##  Max.   :4.000   Max.   :10.0000   Max.   :7.0000       Max.   :17.000
##  type_of_meal_plan  required_car_parking_space room_type_reserved
##  Length:36275       Min.   :0.00000            Length:36275
##  Class :character   1st Qu.:0.00000            Class :character
##  Mode  :character   Median :0.00000            Mode  :character
##                     Mean   :0.03099
##                     3rd Qu.:0.00000
##                     Max.   :1.00000
##    lead_time       arrival_year  arrival_month     arrival_date
##  Min.   :  0.00   Min.   :2017   Min.   : 1.000   Min.   : 1.0
##  1st Qu.: 17.00   1st Qu.:2018   1st Qu.: 5.000   1st Qu.: 8.0
##  Median : 57.00   Median :2018   Median : 8.000   Median :16.0
##  Mean   : 85.23   Mean   :2018   Mean   : 7.424   Mean   :15.6
##  3rd Qu.:126.00   3rd Qu.:2018   3rd Qu.:10.000   3rd Qu.:23.0
##  Max.   :443.00   Max.   :2018   Max.   :12.000   Max.   :31.0
##  market_segment_type repeated_guest    no_of_previous_cancellations
##  Length:36275        Min.   :0.00000   Min.   : 0.00000
##  Class :character    1st Qu.:0.00000   1st Qu.: 0.00000
##  Mode  :character    Median :0.00000   Median : 0.00000
##                      Mean   :0.02564   Mean   : 0.02335
```

```
##                        3rd Qu.:0.00000   3rd Qu.: 0.00000
##                        Max.   :1.00000   Max.   :13.00000
##  no_of_previous_bookings_not_canceled no_of_special_requests booking_st
atus
##  Min.   : 0.0000                      Min.   :0.0000         Length:362
75
##  1st Qu.: 0.0000                      1st Qu.:0.0000         Class :cha
racter
##  Median : 0.0000                      Median :0.0000         Mode  :cha
racter
##  Mean   : 0.1534                      Mean   :0.6197
##  3rd Qu.: 0.0000                      3rd Qu.:1.0000
##  Max.   :58.0000                      Max.   :5.0000
##  avg_price_per_room
##  Min.   :  0.00
##  1st Qu.: 80.30
##  Median : 99.45
##  Mean   :103.42
##  3rd Qu.:120.00
##  Max.   :540.00
```

Independent graphical views of the numeric variables:

```
#png(file = "hotelreserve histogram plots.png")
opar <- par(no.readonly = TRUE)
par(mfrow = c(3,3)) #since we have 9 plots to show we use a 3x3 matrix
hist(hotel_reserve_noNA[, 1], main = names(hotel_reserve_noNA)[1], xlab =
names(hotel_reserve_noNA)[1], xlim = c(0,5))
hist(hotel_reserve_noNA[, 2], main = names(hotel_reserve_noNA)[2], xlab =
names(hotel_reserve_noNA)[2], xlim = c(0,10))
hist(hotel_reserve_noNA[, 3], main = names(hotel_reserve_noNA)[3], xlab =
names(hotel_reserve_noNA)[3], xlim = c(0,10))
hist(hotel_reserve_noNA[, 4], main = names(hotel_reserve_noNA)[4], xlab =
names(hotel_reserve_noNA)[4], xlim = c(0,20))
hist(hotel_reserve_noNA[, 8], main = names(hotel_reserve_noNA)[8], xlab =
names(hotel_reserve_noNA)[8], xlim = c(0,500))
hist(hotel_reserve_noNA[, 14], main = names(hotel_reserve_noNA)[14], xlab
= names(hotel_reserve_noNA)[14], xlim = c(0,15))
hist(hotel_reserve_noNA[, 15], main = names(hotel_reserve_noNA)[15], xlab
= names(hotel_reserve_noNA)[15], xlim = c(0,60))
hist(hotel_reserve_noNA[, 16], main = names(hotel_reserve_noNA)[16], xlab
= names(hotel_reserve_noNA)[16], xlim = c(0,5))
hist(hotel_reserve_noNA[, 18], main = names(hotel_reserve_noNA)[18], xlab
= names(hotel_reserve_noNA)[18], xlim = c(0,600))
```




```
par(opar)
#dev.off()
```

For the average price per room, our histogram looks skewed to the right. This will be investigated further when the price is compared to other variables.

Categorical Data

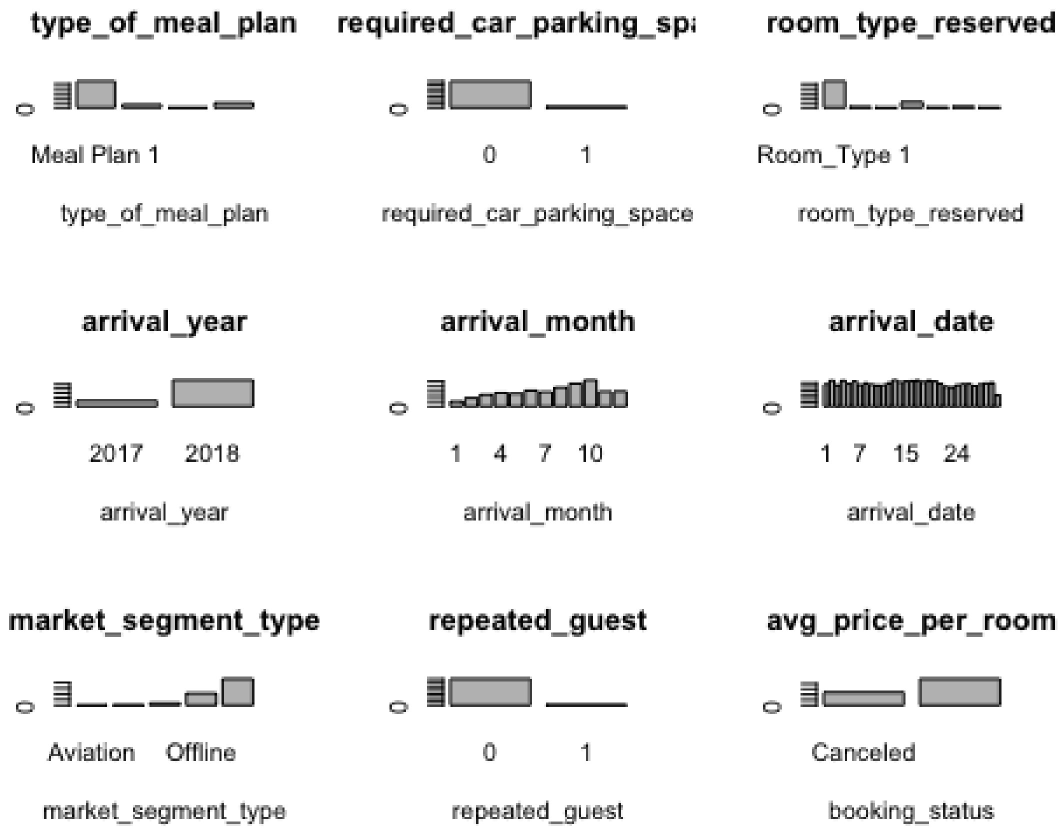
```
#  frequency tables for each categorical variable
hotel_reservation_cat_table <- apply(hotel_reserve_noNA[,c("type_of_meal_p
lan", "required_car_parking_space", "room_type_reserved", "arrival_year","
arrival_date","arrival_month", "market_segment_type", "repeated_guest", "b
ooking_status")], 2, table)

# visualize the table
hotel_reservation_cat_table

## $type_of_meal_plan
##
##  Meal Plan 1  Meal Plan 2  Meal Plan 3 Not Selected
##        27835         3305            5         5130
##
## $required_car_parking_space
##
##     0     1
## 35151  1124
##
## $room_type_reserved
##
```

```
## Room_Type 1 Room_Type 2 Room_Type 3 Room_Type 4 Room_Type 5 Room_Type 6
##       28130         692           7        6057         265         966
## Room_Type 7
##         158
##
## $arrival_year
##
##  2017  2018
##  6514 29761
##
## $arrival_date
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14
15   16
## 1133 1331 1098 1327 1154 1273 1110 1198 1130 1089 1098 1204 1358 1242 1
273 1306
##   17   18   19   20   21   22   23   24   25   26   27   28   29   30
31
## 1345 1260 1327 1281 1158 1023  990 1103 1146 1146 1059 1129 1190 1216
578
##
## $arrival_month
##
##    1    2    3    4    5    6    7    8    9   10   11   12
## 1014 1704 2358 2736 2598 3203 2920 3813 4611 5317 2980 3021
##
## $market_segment_type
##
##      Aviation Complementary     Corporate       Offline        Online
##           125           391          2017         10528         23214
##
## $repeated_guest
##
##     0     1
## 35345   930
##
## $booking_status
##
##     Canceled Not_Canceled
##        11885        24390
```

Bar plots for to analyze categorical variables individually

```
#png(file = "hotelreserve bar plots.png")
opar <- par(no.readonly = TRUE)
par(mfrow = c(3,3)) #since we have 7 plots to show we use a 3x3 matrix
barplot(table(hotel_reserve_noNA[, 5]), main = names(hotel_reserve_noNA)[5
], xlab = names(hotel_reserve_noNA)[5])
barplot(table(hotel_reserve_noNA[, 6]), main = names(hotel_reserve_noNA)[6
], xlab = names(hotel_reserve_noNA)[6])
barplot(table(hotel_reserve_noNA[, 7]), main = names(hotel_reserve_noNA)[7
], xlab = names(hotel_reserve_noNA)[7])
barplot(table(hotel_reserve_noNA[, 9]), main = names(hotel_reserve_noNA)[9
], xlab = names(hotel_reserve_noNA)[9])
barplot(table(hotel_reserve_noNA[, 10]), main = names(hotel_reserve_noNA)[
10], xlab = names(hotel_reserve_noNA)[10])
```

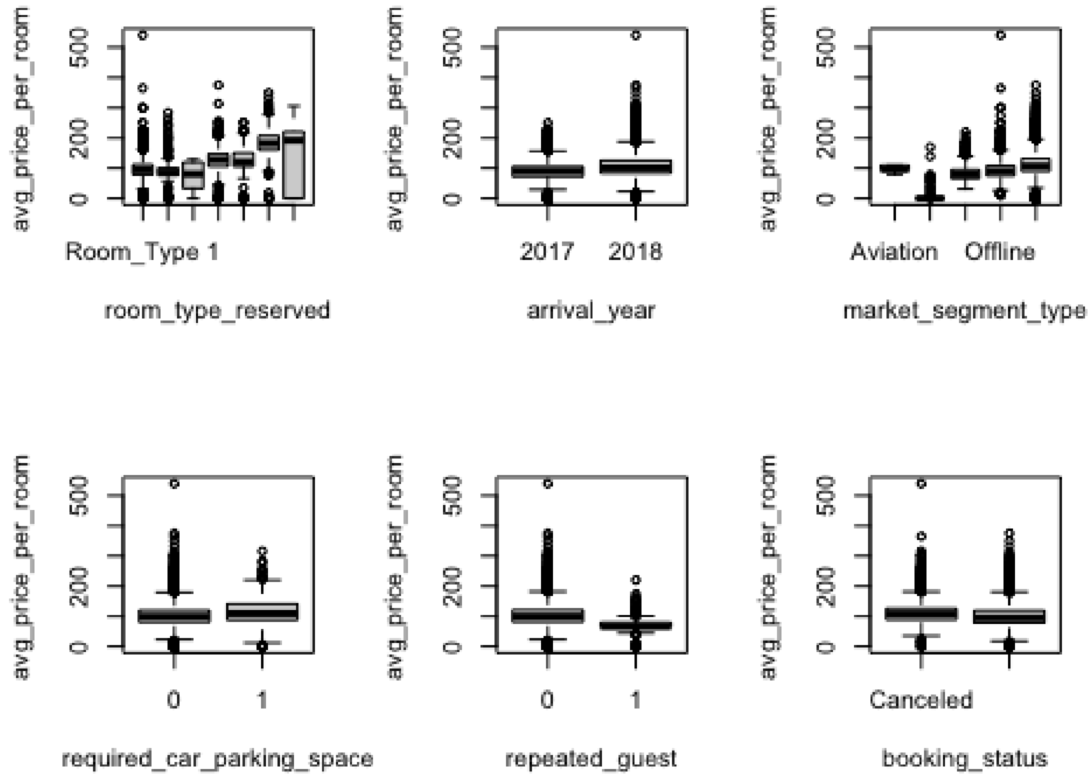
```
barplot(table(hotel_reserve_noNA[, 11]), main = names(hotel_reserve_noNA)[
11], xlab = names(hotel_reserve_noNA)[11])
barplot(table(hotel_reserve_noNA[, 12]), main = names(hotel_reserve_noNA)[
12], xlab = names(hotel_reserve_noNA)[12])
barplot(table(hotel_reserve_noNA[, 13]), main = names(hotel_reserve_noNA)[
13], xlab = names(hotel_reserve_noNA)[13])
barplot(table(hotel_reserve_noNA[, 17]), main = names(hotel_reserve_noNA)[
18], xlab = names(hotel_reserve_noNA)[17])
```




```
par(opar)
#dev.off()
```

Comparing relationships between the average room price and other variables

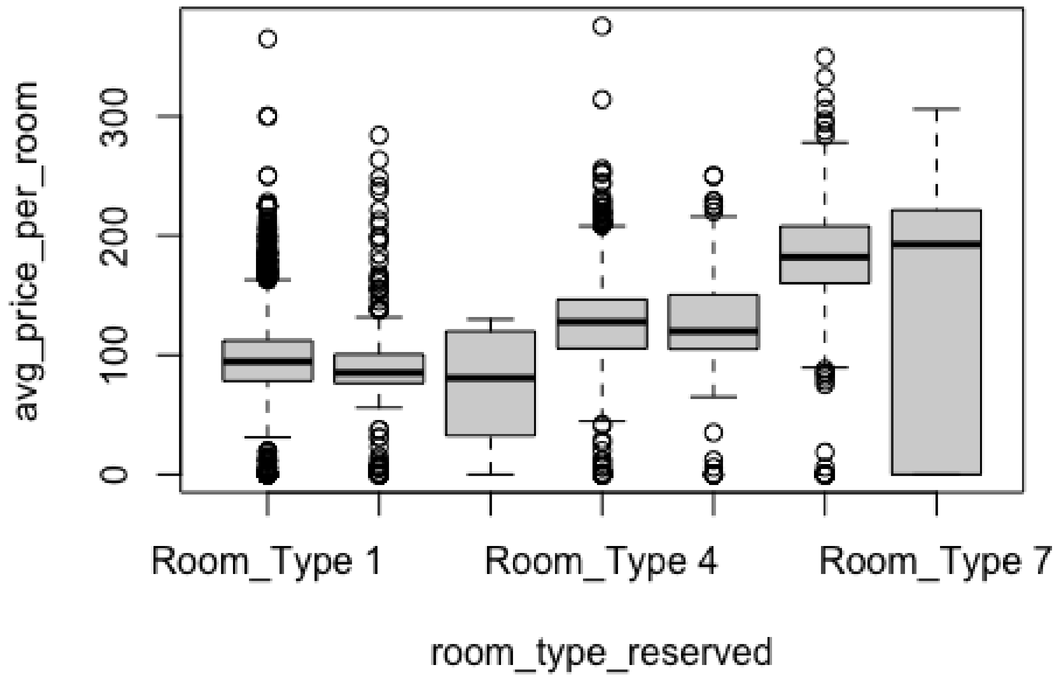
```
# plot avg_price_per_room distribution by group of categorical variables -
boxplot
#png(file = "hotelreserve box plots price.png")
opar <- par(no.readonly = TRUE)
par(mfrow = c(2,3))
boxplot(avg_price_per_room ~ room_type_reserved, data = hotel_reserve_noNA
)
boxplot(avg_price_per_room ~ arrival_year, data = hotel_reserve_noNA)
boxplot(avg_price_per_room ~ market_segment_type, data = hotel_reserve_noN
A)
boxplot(avg_price_per_room ~ required_car_parking_space, data = hotel_rese
rve_noNA)
boxplot(avg_price_per_room ~ repeated_guest, data = hotel_reserve_noNA)
boxplot(avg_price_per_room ~ booking_status, data = hotel_reserve_noNA)
```



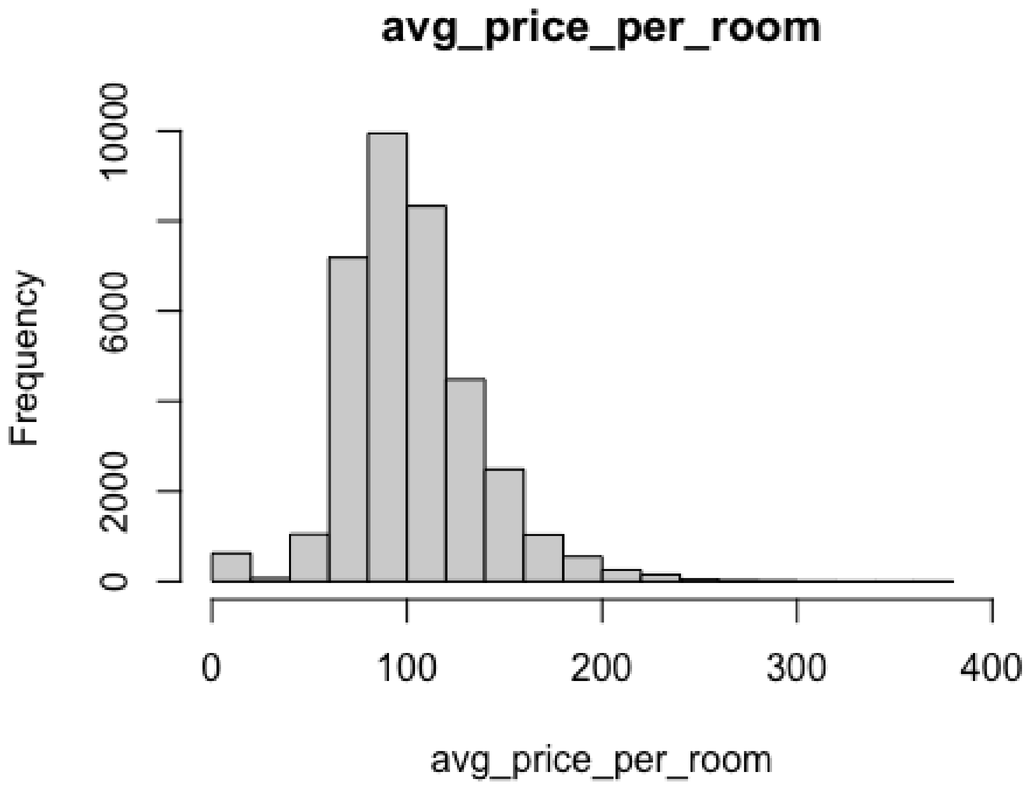

```
par(opar)
#dev.off()
```

From here we can see consistently that there are outliers. However the outlier that looks most plausible is the price above 500 which is significantly distant from the rest of the points. We will take this point out but we do not have sufficient reason to remove the other outliers as they are most likely part of our data.

```
#Removing the outlier (instance in average price greater than 500)
# outliers rows can be extracted by conditional selection
hotel_reserve_noOut <- hotel_reserve_noNA[hotel_reserve_noNA$avg_price_per
_room <= 500, ]
boxplot(avg_price_per_room ~ room_type_reserved, data = hotel_reserve_noOu
t)
```

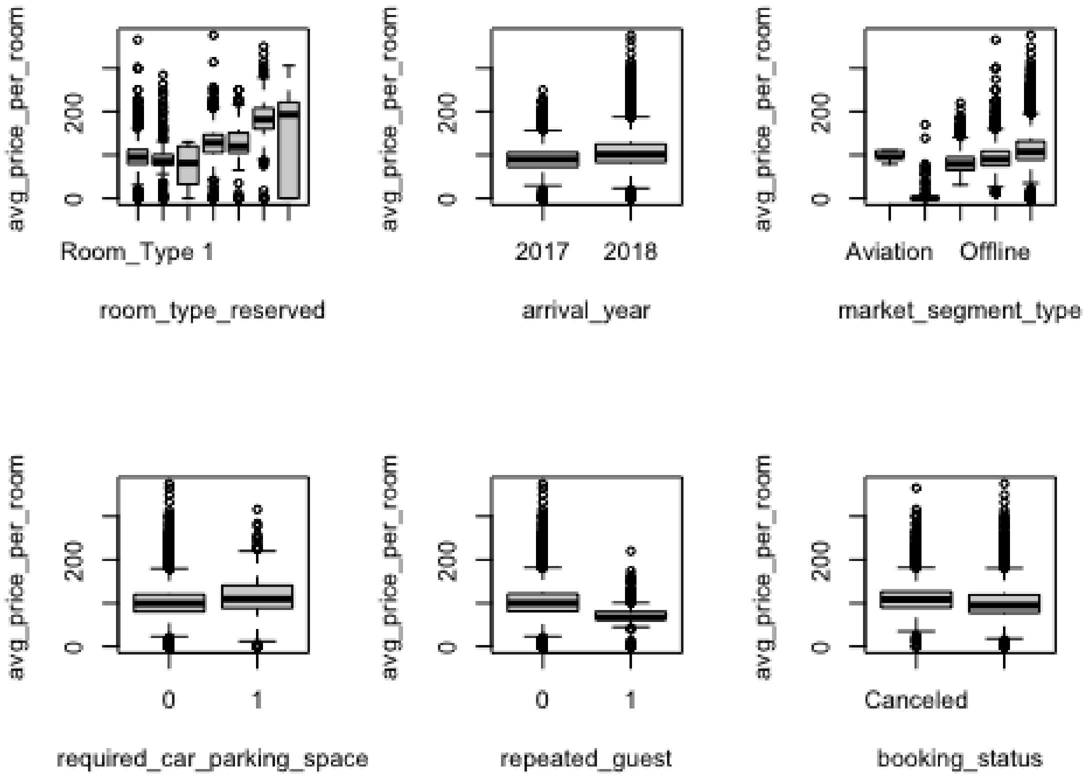



```
#visualizing the average price per room
summary(hotel_reserve_noOut$avg_price_per_room)

##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.00   80.30   99.45  103.41  120.00  375.50

hist(hotel_reserve_noOut[, 18], main = names(hotel_reserve_noOut)[18], xla
b = names(hotel_reserve_noOut)[18], xlim = c(0,400))
```



The average price per room is still a little skewed to the right but atleast better than before the outlier was removed.

```
#comparing the relationship of price with other variables without the outl
ier
#png(file = "hotelreserve no Out box plots price.png")
opar <- par(no.readonly = TRUE)
par(mfrow = c(2,3))
boxplot(avg_price_per_room ~ room_type_reserved, data = hotel_reserve_noOu
t)
boxplot(avg_price_per_room ~ arrival_year, data = hotel_reserve_noOut)
boxplot(avg_price_per_room ~ market_segment_type, data = hotel_reserve_noO
ut)
boxplot(avg_price_per_room ~ required_car_parking_space, data = hotel_rese
rve_noOut)
boxplot(avg_price_per_room ~ repeated_guest, data = hotel_reserve_noOut)
boxplot(avg_price_per_room ~ booking_status, data = hotel_reserve_noOut)
```



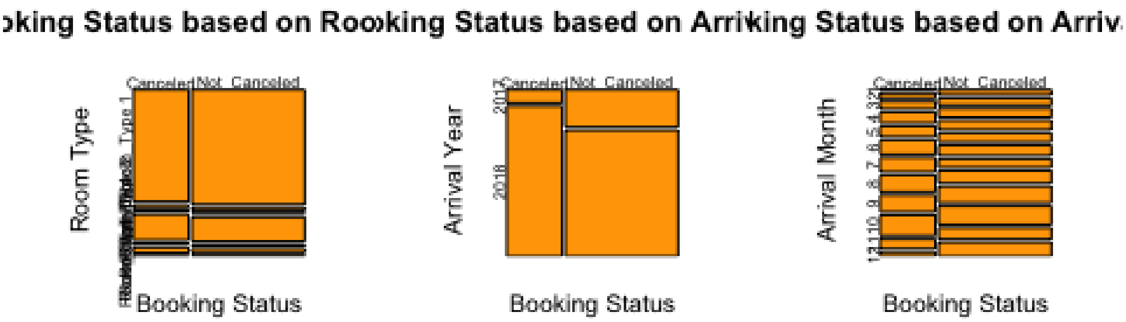

```
par(opar)
#dev.off()
```

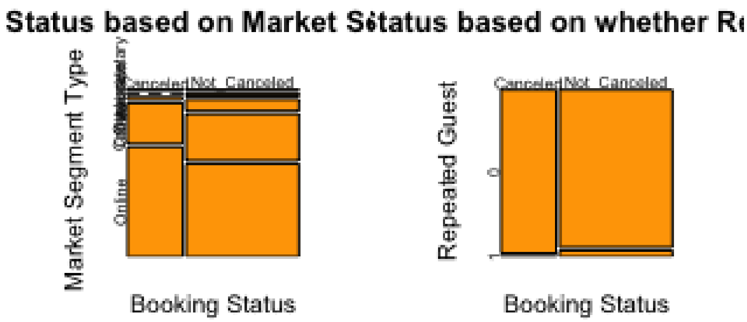
Our data looks good to proceed with.

Mosaic Plots - Categorical Variables against each other

```
#png(file = "hotelreserve_noOut mosaic plots .png")
opar <- par(no.readonly = TRUE)
par(mfrow = c(2,3))
counts <- table(hotel_reserve_noOut$booking_status, hotel_reserve_noOut$ro
om_type_reserved)
mosaicplot(counts, xlab='Booking Status', ylab='Room Type',main='Booking S
tatus based on Room Type', col='orange')

counts <- table(hotel_reserve_noOut$booking_status, hotel_reserve_noOut$ar
rival_year)
mosaicplot(counts, xlab='Booking Status', ylab='Arrival Year',main='Bookin
g Status based on Arrival Year', col='orange')

counts <- table(hotel_reserve_noOut$booking_status, hotel_reserve_noOut$ar
rival_month)
mosaicplot(counts, xlab='Booking Status', ylab='Arrival Month',main='Booki
ng Status based on Arrival Month', col='orange')

counts <- table(hotel_reserve_noOut$booking_status, hotel_reserve_noOut$ma
rket_segment_type)
mosaicplot(counts, xlab='Booking Status', ylab='Market Segment Type',main=
'Booking Status based on Market Segment Type', col='orange')
```

```
counts <- table(hotel_reserve_noOut$booking_status, hotel_reserve_noOut$re
peated_guest)
mosaicplot(counts, xlab='Booking Status', ylab='Repeated Guest',main='Book
ing Status based on whether Repeated Guest', col='orange')
par(opar)
```






```
#dev.off()
```

Data Transformation

We will now be re-coding the variables below to enable easy manipulation of our data in the following sections (PCA and Modelling) - type_of_meal_plan - room_type_reserved - market_segment_type - booking_status

```
hotel_reserve_noOut$type_of_meal_plan[hotel_reserve_noOut$type_of_meal_pla
n == "Not Selected"] <- 0
hotel_reserve_noOut$type_of_meal_plan[hotel_reserve_noOut$type_of_meal_pla
n == "Meal Plan 1"] <- 1
hotel_reserve_noOut$type_of_meal_plan[hotel_reserve_noOut$type_of_meal_pla
n == "Meal Plan 2"] <- 2
hotel_reserve_noOut$type_of_meal_plan[hotel_reserve_noOut$type_of_meal_pla
n == "Meal Plan 3"] <- 3

hotel_reserve_noOut$room_type_reserved[hotel_reserve_noOut$room_type_reser
ved == "Room_Type 1"] <- 1
hotel_reserve_noOut$room_type_reserved[hotel_reserve_noOut$room_type_reser
ved == "Room_Type 2"] <- 2
```

```
hotel_reserve_noOut$room_type_reserved[hotel_reserve_noOut$room_type_reser
ved == "Room_Type 3"] <- 3
hotel_reserve_noOut$room_type_reserved[hotel_reserve_noOut$room_type_reser
ved == "Room_Type 4"] <- 4
hotel_reserve_noOut$room_type_reserved[hotel_reserve_noOut$room_type_reser
ved == "Room_Type 5"] <- 5
hotel_reserve_noOut$room_type_reserved[hotel_reserve_noOut$room_type_reser
ved == "Room_Type 6"] <- 6
hotel_reserve_noOut$room_type_reserved[hotel_reserve_noOut$room_type_reser
ved == "Room_Type 7"] <- 7

hotel_reserve_noOut$market_segment_type[hotel_reserve_noOut$market_segment
_type == "Aviation"] <- 1
hotel_reserve_noOut$market_segment_type[hotel_reserve_noOut$market_segment
_type == "Complementary"] <- 2
hotel_reserve_noOut$market_segment_type[hotel_reserve_noOut$market_segment
_type == "Corporate"] <- 3
hotel_reserve_noOut$market_segment_type[hotel_reserve_noOut$market_segment
_type == "Offline"] <- 4
hotel_reserve_noOut$market_segment_type[hotel_reserve_noOut$market_segment
_type == "Online"] <- 5

hotel_reserve_noOut$booking_status[hotel_reserve_noOut$booking_status == "
Canceled"] <- 1
hotel_reserve_noOut$booking_status[hotel_reserve_noOut$booking_status == "
Not_Canceled"] <- 2
```

Visualize the encoded data

```
summary(hotel_reserve_noOut) #summary of our cleaned data
```

```
##   no_of_adults    no_of_children    no_of_weekend_nights no_of_week_nigh
ts
##  Min.   :0.000   Min.   : 0.0000   Min.   :0.0000       Min.   : 0.000
##  1st Qu.:2.000   1st Qu.: 0.0000   1st Qu.:0.0000       1st Qu.: 1.000
##  Median :2.000   Median : 0.0000   Median :1.0000       Median : 2.000
##  Mean   :1.845   Mean   : 0.1053   Mean   :0.8107       Mean   : 2.204
##  3rd Qu.:2.000   3rd Qu.: 0.0000   3rd Qu.:2.0000       3rd Qu.: 3.000
##  Max.   :4.000   Max.   :10.0000   Max.   :7.0000       Max.   :17.000
##  type_of_meal_plan  required_car_parking_space room_type_reserved
##  Length:36274       Min.   :0.00000            Length:36274
##  Class :character   1st Qu.:0.00000            Class :character
##  Mode  :character   Median :0.00000            Mode  :character
##                     Mean   :0.03099
##                     3rd Qu.:0.00000
##                     Max.   :1.00000
##    lead_time       arrival_year  arrival_month     arrival_date
##  Min.   :  0.00   Min.   :2017   Min.   : 1.000   Min.   : 1.0
##  1st Qu.: 17.00   1st Qu.:2018   1st Qu.: 5.000   1st Qu.: 8.0
##  Median : 57.00   Median :2018   Median : 8.000   Median :16.0
##  Mean   : 85.23   Mean   :2018   Mean   : 7.424   Mean   :15.6
##  3rd Qu.:126.00   3rd Qu.:2018   3rd Qu.:10.000   3rd Qu.:23.0
##  Max.   :443.00   Max.   :2018   Max.   :12.000   Max.   :31.0
##  market_segment_type repeated_guest    no_of_previous_cancellations
##  Length:36274        Min.   :0.00000   Min.   : 0.00000
##  Class :character    1st Qu.:0.00000   1st Qu.: 0.00000
```

```
##  Mode  :character    Median :0.00000   Median : 0.00000
##                      Mean   :0.02564   Mean   : 0.02335
##                      3rd Qu.:0.00000   3rd Qu.: 0.00000
##                      Max.   :1.00000   Max.   :13.00000
##  no_of_previous_bookings_not_canceled no_of_special_requests booking_st
atus
##  Min.   : 0.0000                      Min.   :0.0000         Length:362
74
##  1st Qu.: 0.0000                      1st Qu.:0.0000         Class :cha
racter
##  Median : 0.0000                      Median :0.0000         Mode  :cha
racter
##  Mean   : 0.1534                      Mean   :0.6197
##  3rd Qu.: 0.0000                      3rd Qu.:1.0000
##  Max.   :58.0000                      Max.   :5.0000
##  avg_price_per_room
##  Min.   :  0.00
##  1st Qu.: 80.30
##  Median : 99.45
##  Mean   :103.41
##  3rd Qu.:120.00
##  Max.   :375.50
```

```
str(hotel_reserve_noOut)
```

```
## 'data.frame':    36274 obs. of  18 variables:
##  $ no_of_adults                        : int  2 2 1 2 2 2 2 2 3 2 ...
##  $ no_of_children                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_weekend_nights                : int  1 2 2 0 1 0 1 1 0 0 ...
##  $ no_of_week_nights                   : int  2 3 1 2 1 2 3 3 4 5 ...
##  $ type_of_meal_plan                   : chr  "1" "0" "1" "1" ...
##  $ required_car_parking_space          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ room_type_reserved                  : chr  "1" "1" "1" "1" ...
##  $ lead_time                           : int  224 5 1 211 48 346 34 83
121 44 ...
##  $ arrival_year                        : int  2017 2018 2018 2018 2018
2018 2017 2018 2018 2018 ...
##  $ arrival_month                       : int  10 11 2 5 4 9 10 12 7 10
...
##  $ arrival_date                        : int  2 6 28 20 11 13 15 26 6 1
8 ...
##  $ market_segment_type                 : chr  "4" "5" "5" "5" ...
##  $ repeated_guest                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_cancellations        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_bookings_not_canceled: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_special_requests              : int  0 1 0 0 0 1 1 1 1 3 ...
##  $ booking_status                      : chr  "2" "2" "1" "1" ...
##  $ avg_price_per_room                  : num  65 106.7 60 100 94.5 ...
```

```
head(hotel_reserve_noOut)
```

```
##   no_of_adults no_of_children no_of_weekend_nights no_of_week_nights
## 1            2              0                    1                 2
## 2            2              0                    2                 3
## 3            1              0                    2                 1
## 4            2              0                    0                 2
```

```
## 5          2              0                    1                 1
## 6          2              0                    0                 2
##   type_of_meal_plan required_car_parking_space room_type_reserved lead_
time
## 1                 1                          0                  1
224
## 2                 0                          0                  1
5
## 3                 1                          0                  1
1
## 4                 1                          0                  1
211
## 5                 0                          0                  1
48
## 6                 2                          0                  1
346
##   arrival_year arrival_month arrival_date market_segment_type repeated_
guest
## 1         2017            10            2                   4
0
## 2         2018            11            6                   5
0
## 3         2018             2           28                   5
0
## 4         2018             5           20                   5
0
## 5         2018             4           11                   5
0
## 6         2018             9           13                   5
0
##   no_of_previous_cancellations no_of_previous_bookings_not_canceled
## 1                            0                                    0
## 2                            0                                    0
## 3                            0                                    0
## 4                            0                                    0
## 5                            0                                    0
## 6                            0                                    0
##   no_of_special_requests booking_status avg_price_per_room
## 1                      0              2              65.00
## 2                      1              2             106.68
## 3                      0              1              60.00
## 4                      0              1             100.00
## 5                      0              1              94.50
## 6                      1              1             115.00
```

The factors below have been successfully encoded but they are still being read as character so we will convert them to numerical: - type_of_meal_plan - room_type_reserved - market_segment_type - booking_status

```
hotel_reserve_noOut$type_of_meal_plan <- as.numeric(hotel_reserve_noOut$ty
pe_of_meal_plan)
hotel_reserve_noOut$room_type_reserved <- as.numeric(hotel_reserve_noOut$r
oom_type_reserved)
hotel_reserve_noOut$market_segment_type <- as.numeric(hotel_reserve_noOut$
market_segment_type)
```

```
hotel_reserve_noOut$booking_status <- as.numeric(hotel_reserve_noOut$booki
ng_status)
str(hotel_reserve_noOut)

## 'data.frame':    36274 obs. of  18 variables:
##  $ no_of_adults                        : int  2 2 1 2 2 2 2 2 3 2 ...
##  $ no_of_children                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_weekend_nights                : int  1 2 2 0 1 0 1 1 0 0 ...
##  $ no_of_week_nights                   : int  2 3 1 2 1 2 3 3 4 5 ...
##  $ type_of_meal_plan                   : num  1 0 1 1 0 2 1 1 1 1 ...
##  $ required_car_parking_space          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ room_type_reserved                  : num  1 1 1 1 1 1 1 4 1 4 ...
##  $ lead_time                           : int  224 5 1 211 48 346 34 83
121 44 ...
##  $ arrival_year                        : int  2017 2018 2018 2018 2018
2018 2017 2018 2018 2018 ...
##  $ arrival_month                       : int  10 11 2 5 4 9 10 12 7 10
...
##  $ arrival_date                        : int  2 6 28 20 11 13 15 26 6 1
8 ...
##  $ market_segment_type                 : num  4 5 5 5 5 5 5 5 4 5 ...
##  $ repeated_guest                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_cancellations        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_bookings_not_canceled: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_special_requests              : int  0 1 0 0 0 1 1 1 1 3 ...
##  $ booking_status                      : num  2 2 1 1 1 1 2 2 2 2 ...
##  $ avg_price_per_room                  : num  65 106.7 60 100 94.5 ...
```

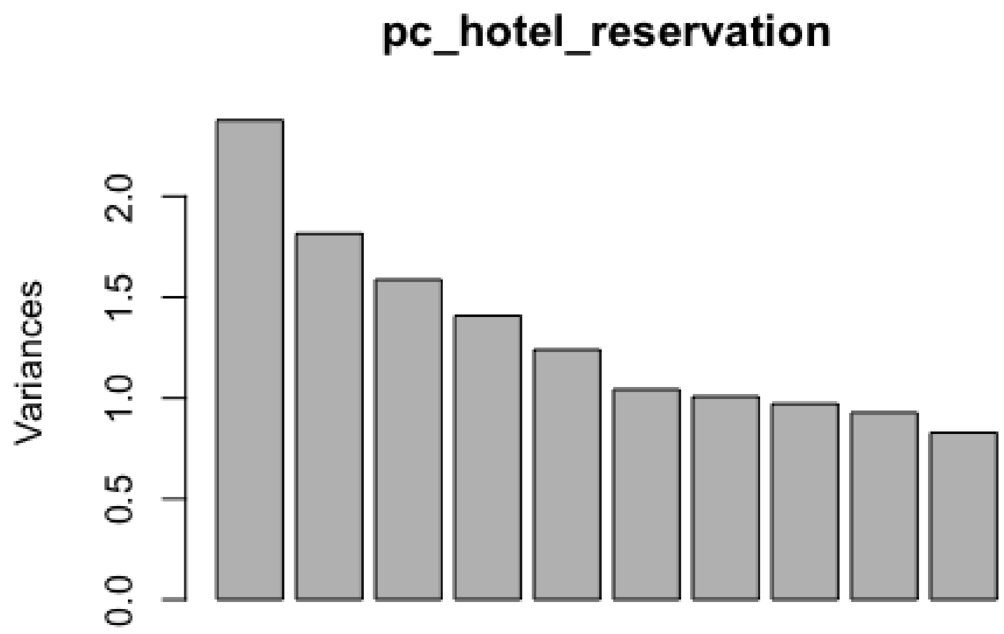
Principal Component Analysis

```
# Performing PCA on all the variables except our target variable avg_price
_per_room
pc_hotel_reservation <- prcomp(hotel_reserve_noOut[,c(1,2,3,4,5,6,7,8,9,10
,11,12,13,14,15,16,17)], center = T, scale. = T)
attributes(pc_hotel_reservation)

## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"

summary(pc_hotel_reservation)

## Importance of components:
##                           PC1    PC2     PC3     PC4    PC5     PC6
PC7
## Standard deviation     1.5419 1.3473 1.25973 1.18635 1.1132 1.02109 1.0
0371
## Proportion of Variance 0.1399 0.1068 0.09335 0.08279 0.0729 0.06133 0.0
5926
## Cumulative Proportion  0.1399 0.2466 0.33997 0.42276 0.4957 0.55699 0.6
1625
##                            PC8     PC9    PC10    PC11   PC12    PC13
PC14
## Standard deviation     0.98560 0.96307 0.91029 0.89449 0.8429 0.73831 0
.71683
```

```
## Proportion of Variance 0.05714 0.05456 0.04874 0.04706 0.0418 0.03207 0
.03023
## Cumulative Proportion  0.67340 0.72795 0.77670 0.82376 0.8656 0.89762 0
.92785
##                           PC15    PC16    PC17
## Standard deviation     0.66718 0.63490 0.61507
## Proportion of Variance 0.02618 0.02371 0.02225
## Cumulative Proportion  0.95403 0.97775 1.00000
```

Visual Analysis of PCA results

```
# calculate the proportion of explained variance (PEV) from the std values
pc_hotel_reservation_var <- pc_hotel_reservation$sdev^2
pc_hotel_reservation_var

##  [1] 2.3774818 1.8151355 1.5869288 1.4074283 1.2392624 1.0426320 1.0074
375
##  [8] 0.9714132 0.9275011 0.8286243 0.8001041 0.7105618 0.5451076 0.5138
486
## [15] 0.4451277 0.4030969 0.3783084

pc_hotel_reservation_PEV <- pc_hotel_reservation_var / sum(pc_hotel_reserv
ation_var)
pc_hotel_reservation_PEV

##  [1] 0.13985187 0.10677268 0.09334875 0.08278990 0.07289779 0.06133129
##  [7] 0.05926103 0.05714195 0.05455889 0.04874261 0.04706495 0.04179776
## [13] 0.03206516 0.03022639 0.02618398 0.02371158 0.02225344

# plot of the variance per PC
#png(file = "hotelreserve_noOut PC PEV .png")
plot(pc_hotel_reservation)
```



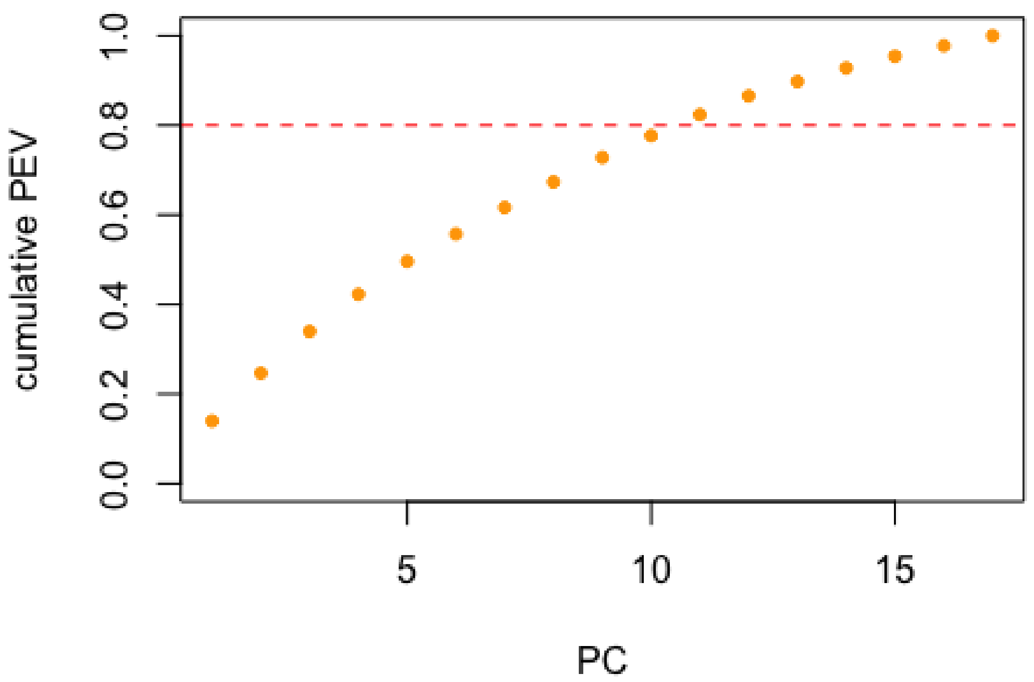

```
#dev.off()
```

Plot of the cumulative value of PEV for increasing number of additional PCs

We added an 80% threshold line to inform the feature extraction

according to the plot the first 10 PCs should be selected

```
#Scree Plot
#png(file = "hotelreserve PC Scree Plot.png")
opar <- par(no.readonly = TRUE)
plot(
  cumsum(pc_hotel_reservation_PEV),
  ylim = c(0,1),
  xlab = 'PC',
  ylab = 'cumulative PEV',
  pch = 20,
  col = 'orange'
)
abline(h = 0.8, col = 'red', lty = 'dashed')
```




```
par(opar)
#dev.off()
```

From here we can see that 10 PC's contribute to 80% of the information in the dataset.

Getting and inspecting the loadings for each PC

```
pc_hotel_reservation_loadings <- pc_hotel_reservation$rotation
pc_hotel_reservation_loadings
```

```
##                                           PC1         PC2           P
C3
## no_of_adults                     -0.318138429  0.18543746 -0.130782
03
## no_of_children                   -0.124688084  0.26627099 -0.108120
53
## no_of_weekend_nights             -0.162511112  0.06984672 -0.169696
32
## no_of_week_nights                -0.191893804 -0.01470706 -0.202883
98
## type_of_meal_plan                 0.052635707 -0.24721632 -0.092355
29
## required_car_parking_space        0.074971865  0.19439823  0.019847
71
## room_type_reserved               -0.183831167  0.35123646 -0.178802
44
## lead_time                        -0.174796728 -0.40044758 -0.371414
87
## arrival_year                     -0.142571687  0.11845760 -0.409333
```

```
80
## arrival_month                          -0.008699844 -0.07505234  0.196640
85
## arrival_date                           -0.032255212  0.03341048 -0.050258
17
## market_segment_type                    -0.406045402  0.27919308 -0.032516
31
## repeated_guest                          0.471261525  0.15816283 -0.232152
53
## no_of_previous_cancellations            0.329235034  0.18969988 -0.348584
79
## no_of_previous_bookings_not_canceled   0.415169135  0.19886326 -0.341613
67
## no_of_special_requests                 -0.130620240  0.45498221  0.107793
74
## booking_status                          0.202590240  0.31733093  0.457027
34
##                                                  PC4          PC5
PC6
## no_of_adults                            0.139062372 -0.170244369  0.16763
7461
## no_of_children                          0.244433135  0.477617571  0.01262
7459
## no_of_weekend_nights                    0.063865881 -0.232273431 -0.59843
6905
## no_of_week_nights                       0.197951229 -0.231956691 -0.43390
9296
## type_of_meal_plan                       0.473615023  0.326948622 -0.06884
5718
## required_car_parking_space              0.014775416  0.072199595  0.38030
6736
## room_type_reserved                      0.301579553  0.389344816 -0.00853
7873
## lead_time                               0.171411125 -0.138446609  0.24997
1609
## arrival_year                           -0.464851942  0.113042989  0.08350
7592
## arrival_month                           0.516795299 -0.370775684  0.24037
2304
## arrival_date                           -0.008422843  0.147330830 -0.24432
2807
## market_segment_type                    -0.115239470 -0.230370803  0.15001
7759
## repeated_guest                          0.074343277 -0.056981065  0.01372
7524
## no_of_previous_cancellations            0.057358836 -0.204740635  0.02894
7832
## no_of_previous_bookings_not_canceled   0.081344366 -0.154559747  0.00923
7412
## no_of_special_requests                  0.142607203 -0.241174280  0.11200
1904
## booking_status                          0.022418476 -0.005357345 -0.22977
1784
##                                                   PC7          PC8
PC9
```

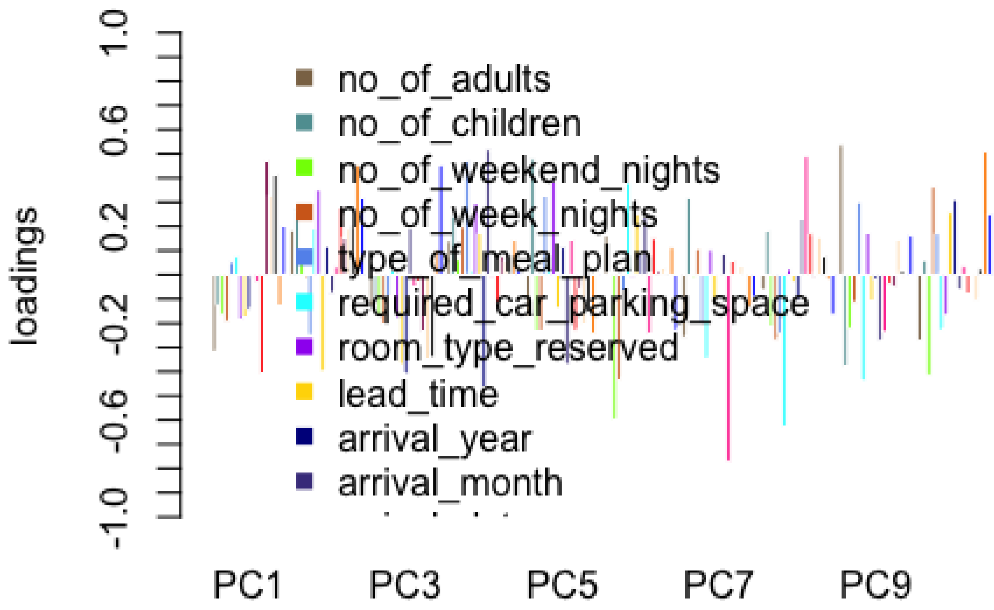
```
## no_of_adults                              -2.576871e-01 -0.06253300  0.53623
875
## no_of_children                             3.188647e-01  0.18603162 -0.37744
305
## no_of_weekend_nights                      -1.234539e-02 -0.21545706 -0.22071
742
## no_of_week_nights                          1.091667e-01 -0.26872327 -0.11319
443
## type_of_meal_plan                         -2.062806e-01 -0.24024691  0.30158
056
## required_car_parking_space                -3.471048e-01 -0.62814343 -0.43751
839
## room_type_reserved                         1.010936e-01  0.02329001  0.17375
986
## lead_time                                 -1.114517e-01 -0.03180831 -0.10952
407
## arrival_year                              -4.660257e-03 -0.10342381 -0.02344
023
## arrival_month                              8.737237e-02  0.23166391 -0.27359
370
## arrival_date                              -7.717505e-01  0.48852713 -0.23673
740
## market_segment_type                        5.873144e-02  0.16884536 -0.03790
930
## repeated_guest                            -3.083058e-05 -0.01321782 -0.04999
741
## no_of_previous_cancellations               3.840226e-02  0.15622512  0.14739
955
## no_of_previous_bookings_not_canceled  1.735078e-03  0.07647624  0.01845
996
## no_of_special_requests                    -1.318109e-01 -0.02173276 -0.01086
007
## booking_status                            -9.285647e-02 -0.16302419  0.16224
873
##                                                   PC10          PC11
PC12
## no_of_adults                              -0.271553866  0.0992698287  0.1630
0781
## no_of_children                             0.060810205 -0.0976260468 -0.1776
3783
## no_of_weekend_nights                      -0.414246539 -0.4983077744  0.0689
3340
## no_of_week_nights                          0.364129199  0.6308920564 -0.0444
1337
## type_of_meal_plan                          0.170728050 -0.2601480726 -0.2783
8159
## required_car_parking_space                -0.227776323  0.1605667740 -0.1222
7252
## room_type_reserved                        -0.168413483  0.1677102296  0.3232
1547
## lead_time                                  0.256917868 -0.1978472300  0.0189
6859
## arrival_year                               0.311269301 -0.1475909995  0.3423
0628
## arrival_month                             -0.061314986  0.0085797573  0.3395
```

```
2865
## arrival_date                          0.034225918  0.1443026269  0.0133
3168
## market_segment_type                  -0.078672594  0.0093559845 -0.4169
3118
## repeated_guest                       -0.001532874  0.0001567824  0.2865
1859
## no_of_previous_cancellations         -0.105324700  0.0612670416 -0.4950
6978
## no_of_previous_bookings_not_canceled  0.023725267 -0.0215423919  0.0440
4206
## no_of_special_requests                0.509959304 -0.3506110229 -0.0253
0209
## booking_status                        0.251484429 -0.0603070324  0.0405
5167
##                                            PC13         PC14          PC
15
## no_of_adults                          0.10874509 -0.51014627 -0.0181816
72
## no_of_children                        0.15768988 -0.49451759 -0.0436032
85
## no_of_weekend_nights                  0.05280095  0.02048826  0.0166678
24
## no_of_week_nights                    -0.03818085 -0.05197669  0.0128755
64
## type_of_meal_plan                    -0.16015907  0.20296160 -0.1006983
60
## required_car_parking_space            0.04854765  0.04258691 -0.0326535
27
## room_type_reserved                    0.01520774  0.46383276  0.0614989
32
## lead_time                             0.24845721 -0.13301102  0.0168832
95
## arrival_year                          0.27022556  0.20764406 -0.1198946
68
## arrival_month                         0.21535256  0.22227342 -0.1193270
25
## arrival_date                          0.02005782  0.01878586  0.0045498
66
## market_segment_type                  -0.26779222  0.21040643 -0.1631638
29
## repeated_guest                       -0.31273228 -0.17042197  0.4696802
08
## no_of_previous_cancellations          0.49549654  0.19735791  0.2659546
38
## no_of_previous_bookings_not_canceled -0.24634787 -0.08373985 -0.7234582
87
## no_of_special_requests               -0.23489974  0.01793716  0.2588450
41
## booking_status                        0.46732814 -0.06139036 -0.2122973
95
##                                            PC16          PC17
## no_of_adults                         -0.10430110 -1.398448e-01
## no_of_children                       -0.10899316 -8.433810e-02
## no_of_weekend_nights                  0.01715400  5.044112e-03
```

```
## no_of_week_nights                       -0.04840367 -6.583829e-02
## type_of_meal_plan                       -0.35330841 -1.558329e-01
## required_car_parking_space               0.03131787 -2.881246e-02
## room_type_reserved                       0.28016064  2.621052e-01
## lead_time                                0.21568156  5.503984e-01
## arrival_year                            -0.34647040 -2.541261e-01
## arrival_month                           -0.28136573 -2.209904e-01
## arrival_date                            -0.02405811 -4.145549e-03
## market_segment_type                     -0.41683112  3.706844e-01
## repeated_guest                          -0.41490973  2.927040e-01
## no_of_previous_cancellations             0.05864546 -1.478964e-01
## no_of_previous_bookings_not_canceled     0.20136174 -3.423441e-05
## no_of_special_requests                   0.32334020 -2.112424e-01
## booking_status                          -0.17195415  4.113460e-01
```

Plotting first 10/17 PCs as barplots

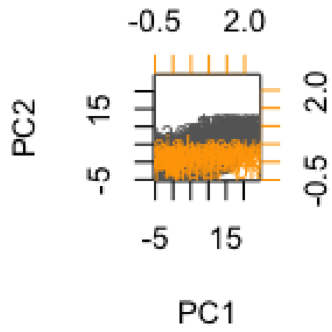
```
#png(file = "hotelreserve PC loadings.png")
opar <- par(no.readonly = TRUE)
colvector = c('burlywood4', 'cadetblue', 'chartreuse', 'chocolate', 'cornf
lowerblue', 'cyan','purple','gold','darkblue','darkslateblue', 'deeppink',
'red', 'deeppink4', 'bisque','black','darkorange','blue')

labvector = c('PC1', 'PC2', 'PC3', 'PC4', 'PC5','PC6',"PC7","PC8","PC9","P
C10")
barplot(
  pc_hotel_reservation_loadings[,c(1:10)],
  beside = T,
  yaxt = 'n',
  names.arg = labvector,
  col = colvector,
  ylim = c(-1,1),
  border = 'white',
  ylab = 'loadings'
)
axis(2, seq(-1,1,0.1))
legend(
  'topright',
  bty = 'n',
  col = colvector,
  pch = 15,
  row.names(pc_hotel_reservation_loadings)
)
```

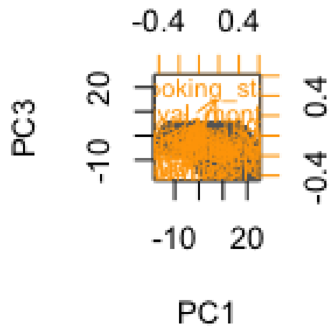


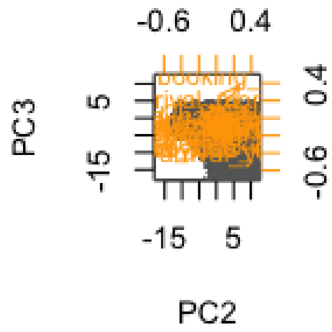

```
par(opar)
#dev.off()
```

Generating a biplot for each pair of important PCs (and show them on the same page)

```
# generate a biplot for each pair of important PCs (and show them on the s
ame page)
#   note: the option choices is used to select the PCs - default is 1:2
#png(file = "hotelreserve PC biplot.png")
opar <- par(no.readonly = TRUE)
par(mfrow = c(2,2))
biplot(
  pc_hotel_reservation,
  scale = 0,
  col = c('grey40','orange')
)
biplot(
  pc_hotel_reservation,
  choices = c(1,3),
  scale = 0,
  col = c('grey40','orange')
)
biplot(
  pc_hotel_reservation,
  choices = c(2,3),
  scale = 0,
  col = c('grey40','orange')
)
par(opar)
```








```
#dev.off()

#Hotel_reservation_cleaned <- write.csv(hotel_reserve_noOut, "HotelReserva
tionClean2.csv")
```

Creating a new data frame for the significant PC's and the average price per room

```
df2<- pc_hotel_reservation$x[,c(1,2,3,4,5,6,7,8,9,10)]
head(df2)

##             PC1         PC2        PC3        PC4        PC5        PC6
## 1  0.52037127 -1.66856375  0.9274377  1.5478862 -0.9653901  0.3405382
## 2 -0.61736771  1.22439969  0.7916546 -0.6958705 -2.1282103 -0.6014270
## 3  0.06452015 -0.55908129 -0.6575124 -2.0316992  0.9921149 -1.4371093
## 4 -0.71763592 -1.45500405 -1.3331134 -0.8383802  0.1975531  1.0226342
## 5 -0.51028576 -0.09989665 -0.4996816 -2.3764327 -0.3540033  0.4853107
## 6 -1.03426819 -2.14480542 -1.6753671  1.2787078 -0.2469431  1.9233033
##            PC7        PC8         PC9       PC10
## 1  0.9032258 -0.6435880  0.41065273 -0.6089206
## 2  1.2616942 -0.2504536 -0.59996076 -0.4971390
## 3 -0.6538572  0.6776372 -0.96578912 -1.0647277
## 4 -0.5255071  0.5630776  0.17643233  0.1580429
## 5  0.7901308  0.4882149 -0.08424167 -1.4339735
## 6 -0.5658988 -0.1042361  0.45169554  1.4585034

df3 <- cbind(df2,hotel_reserve_noOut$avg_price_per_room)
head(df3)

##             PC1         PC2        PC3        PC4        PC5        PC6
## 1  0.52037127 -1.66856375  0.9274377  1.5478862 -0.9653901  0.3405382
```

```
## 2 -0.61736771  1.22439969  0.7916546 -0.6958705 -2.1282103 -0.6014270
## 3  0.06452015 -0.55908129 -0.6575124 -2.0316992  0.9921149 -1.4371093
## 4 -0.71763592 -1.45500405 -1.3331134 -0.8383802  0.1975531  1.0226342
## 5 -0.51028576 -0.09989665 -0.4996816 -2.3764327 -0.3540033  0.4853107
## 6 -1.03426819 -2.14480542 -1.6753671  1.2787078 -0.2469431  1.9233033
##          PC7        PC8         PC9       PC10
## 1  0.9032258 -0.6435880  0.41065273 -0.6089206  65.00
## 2  1.2616942 -0.2504536 -0.59996076 -0.4971390 106.68
## 3 -0.6538572  0.6776372 -0.96578912 -1.0647277  60.00
## 4 -0.5255071  0.5630776  0.17643233  0.1580429 100.00
## 5  0.7901308  0.4882149 -0.08424167 -1.4339735  94.50
## 6 -0.5658988 -0.1042361  0.45169554  1.4585034 115.00
```

```
colnames(df3)
```

```
##  [1] "PC1"  "PC2"  "PC3"  "PC4"  "PC5"  "PC6"  "PC7"  "PC8"  "PC9"  "PC
10"
## [11] ""
```

```
colnames(df3)[colnames(df3) == ""] <- "avg_price_per_room"
colnames(df3)
```

```
##  [1] "PC1"                "PC2"                "PC3"
##  [4] "PC4"                "PC5"                "PC6"
##  [7] "PC7"                "PC8"                "PC9"
## [10] "PC10"               "avg_price_per_room"
```

```
head(df3)
```

```
##           PC1         PC2        PC3        PC4        PC5        PC6
## 1  0.52037127 -1.66856375  0.9274377  1.5478862 -0.9653901  0.3405382
## 2 -0.61736771  1.22439969  0.7916546 -0.6958705 -2.1282103 -0.6014270
## 3  0.06452015 -0.55908129 -0.6575124 -2.0316992  0.9921149 -1.4371093
## 4 -0.71763592 -1.45500405 -1.3331134 -0.8383802  0.1975531  1.0226342
## 5 -0.51028576 -0.09989665 -0.4996816 -2.3764327 -0.3540033  0.4853107
## 6 -1.03426819 -2.14480542 -1.6753671  1.2787078 -0.2469431  1.9233033
##          PC7        PC8         PC9       PC10 avg_price_per_room
## 1  0.9032258 -0.6435880  0.41065273 -0.6089206              65.00
## 2  1.2616942 -0.2504536 -0.59996076 -0.4971390             106.68
## 3 -0.6538572  0.6776372 -0.96578912 -1.0647277              60.00
## 4 -0.5255071  0.5630776  0.17643233  0.1580429             100.00
## 5  0.7901308  0.4882149 -0.08424167 -1.4339735              94.50
## 6 -0.5658988 -0.1042361  0.45169554  1.4585034             115.00
```

```
#This data set will be used for both machine learning and deep learning me
thods in python
Hotel_reservation_PC <- write.csv(df3, "HotelReservationPC2.csv")
```

Appendix 3- Software – Machine Learning and Deep Learning Predictions using Python.

```
from google.colab import drive # Mount the google drive for data loading
drive.mount('/content/drive')

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

Import some related dependencies

1. Numpy: a package for array transformation
2. Pandas: a package for loading data with .csv/.xlsx formats
3. Matplotlib: a package for data visualization
4. Skearn: a package including many machine learning approaches
5. Tensorflow: a package for neural networks modeling
6. Keras: a package for neural networks modeling which is established on Tensorflow

```
import numpy as np #helps for array operation
import pandas as pd #helps to read the data
import matplotlib.pyplot as plt #helps with graphical plots
from sklearn.model_selection import train_test_split #helps to split train
ing data and testing data
from sklearn.preprocessing import StandardScaler #helps for standardation
of input data
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error # for calculating the cost
function
import keras            #helps for CNN model construction
import tensorflow as tf #helps for CNN model construction
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
%matplotlib inline
```

Load data using Pandas package

7. Data visualization
8. Data segmentation
9. Training data and testing data split

```
data = pd.read_csv('/content/drive/MyDrive/ColabNotebooks/CS5812/HotelRese
rvationPC2.csv')  #Load data with the corresponding path in google drive

data.head()

   Unnamed: 0       PC1       PC2       PC3       PC4       PC5       PC6
\
0           1  0.520371 -1.668564  0.927438  1.547886 -0.965390  0.340538
1           2 -0.617368  1.224400  0.791655 -0.695871 -2.128210 -0.601427
2           3  0.064520 -0.559081 -0.657512 -2.031699  0.992115 -1.437109
3           4 -0.717636 -1.455004 -1.333113 -0.838380  0.197553  1.022634
4           5 -0.510286 -0.099897 -0.499682 -2.376433 -0.354003  0.485311
```

```
        PC7       PC8       PC9      PC10  avg_price_per_room
0  0.903226 -0.643588  0.410653 -0.608921               65.00
1  1.261694 -0.250454 -0.599961 -0.497139              106.68
2 -0.653857  0.677637 -0.965789 -1.064728               60.00
3 -0.525507  0.563078  0.176432  0.158043              100.00
4  0.790131  0.488215 -0.084242 -1.433973               94.50
```

```
print(data.shape)
```

```
(36274, 12)
```

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36274 entries, 0 to 36273
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          36274 non-null  int64
 1   PC1                 36274 non-null  float64
 2   PC2                 36274 non-null  float64
 3   PC3                 36274 non-null  float64
 4   PC4                 36274 non-null  float64
 5   PC5                 36274 non-null  float64
 6   PC6                 36274 non-null  float64
 7   PC7                 36274 non-null  float64
 8   PC8                 36274 non-null  float64
 9   PC9                 36274 non-null  float64
 10  PC10                36274 non-null  float64
 11  avg_price_per_room  36274 non-null  float64
dtypes: float64(11), int64(1)
memory usage: 3.3 MB
None
```

Machine Learning Mehtod 1 - Random Forest Regressor

```
X = data.iloc[:,:-1]    # convert the input data to be an array
y = data.iloc[:,-1:]
print ('Shape of input:', X.shape)
print ('Shape of labels:', y.shape)
```

```
Shape of input: (36274, 11)
Shape of labels: (36274, 1)
```

```
 # Randomly split training data and test data with a ratio of 7:3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)
```

RF Model Training

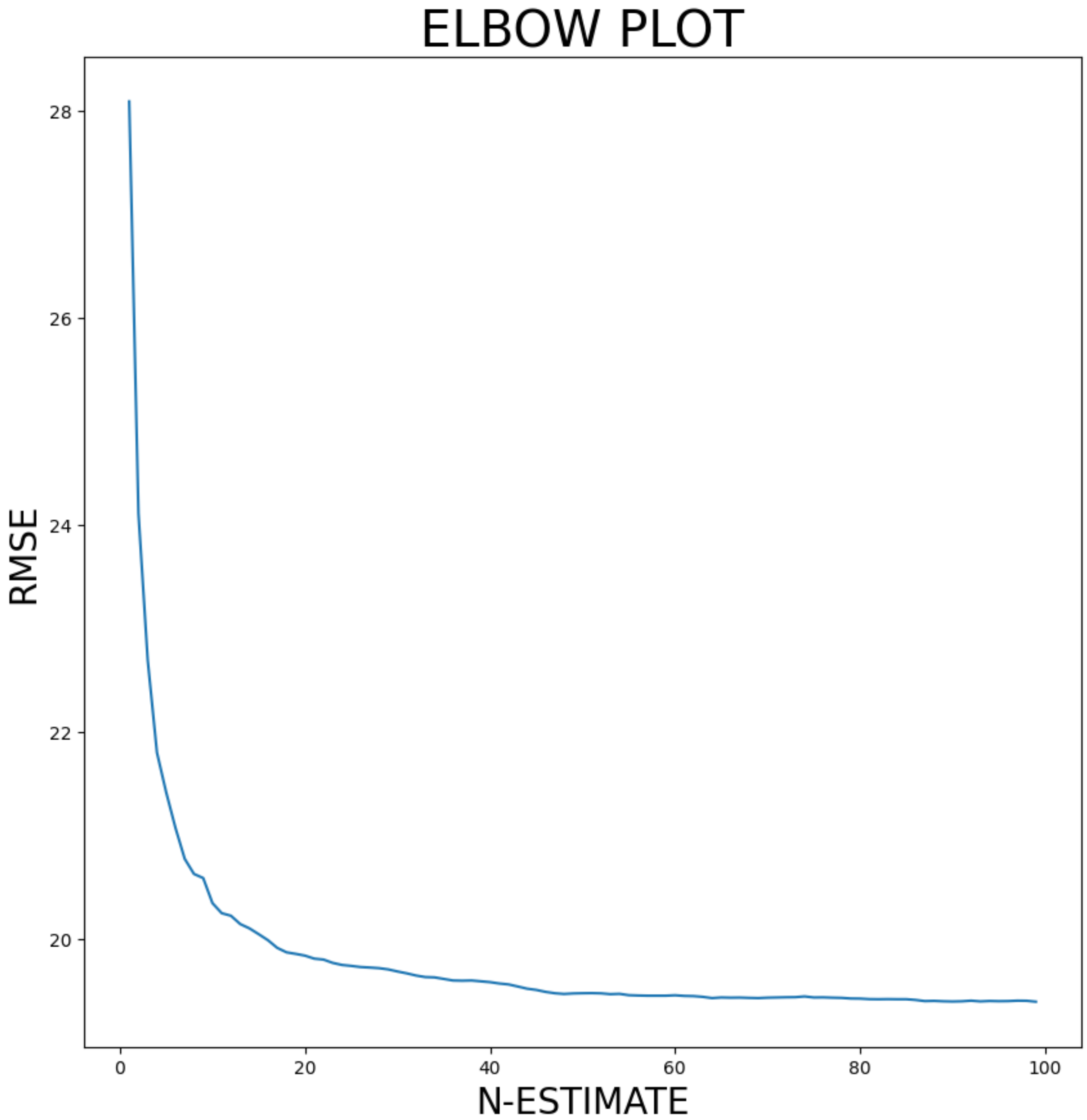
```
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
regressor.fit(X_train, y_train.values.ravel())
```

```
RandomForestRegressor(random_state=0)
```

RF Model Hyperparameters Tuning

```
# Plotting the Elbow plot
fig,ax=plt.subplots(figsize=(10,10))
```

```
n_list=np.arange(1,100,1)
R_dict={} # To store n and rmse pairs
for i in n_list:
#Random Forest Model Creation
    regressor = RandomForestRegressor(n_estimators = int(i), random_state
= 0)
    regressor.fit(X_train, y_train.values.ravel())
    y_pred = regressor.predict(X_test)

#Storing RMSE
    rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'
))
    R_dict[i]=rmse
#Plotting the results
ax.plot(R_dict.keys(),R_dict.values())
ax.set_xlabel('N-ESTIMATE', fontsize=20)
ax.set_ylabel('RMSE' ,fontsize=20)
ax.set_title('ELBOW PLOT' ,fontsize=28)

#fig.savefig('/content/drive/MyDrive/ColabNotebooks/CS5812/RF ELBOW PLOT.p
ng')   # save the figure to file
#plt.close(fig)    # close the figure window

Text(0.5, 1.0, 'ELBOW PLOT')
```



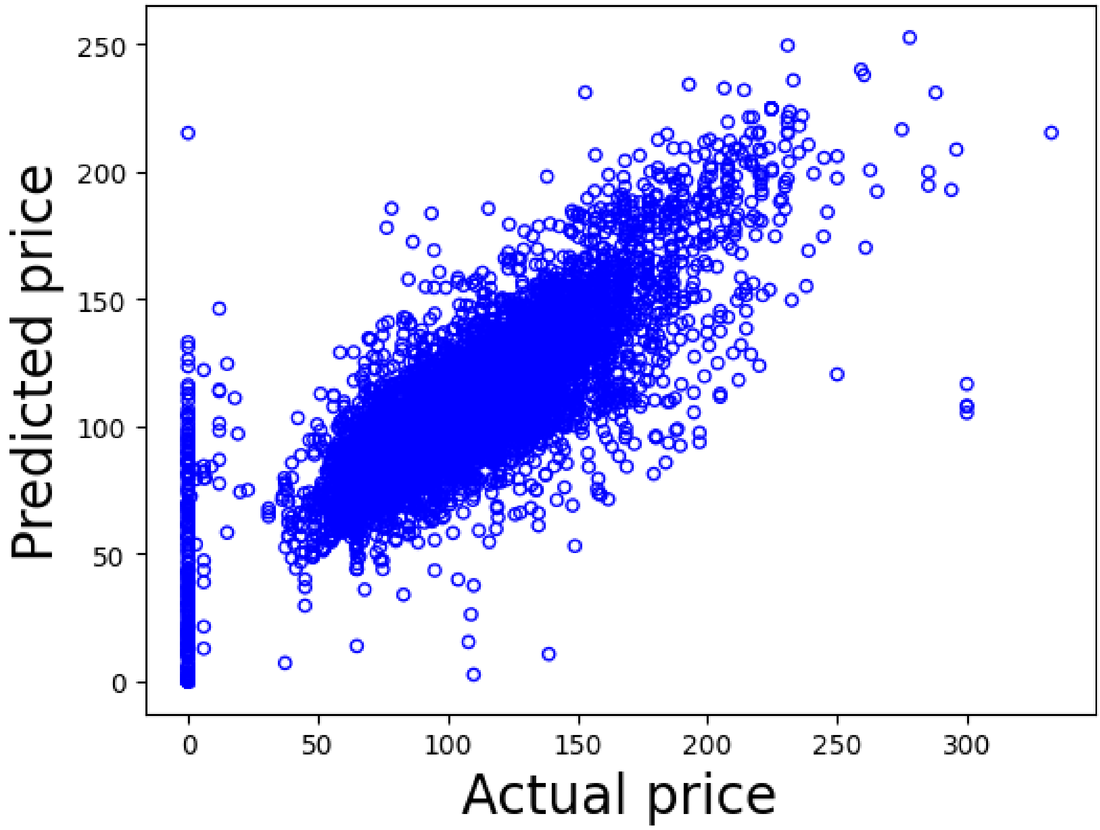
The optiman number of estimators is 50 after which the RMSE value does not reduce any longer.

```
# Model training with optimal parameters
regressor = RandomForestRegressor(n_estimators = 50, random_state = 0)
regressor.fit(X_train, y_train.values.ravel())

RandomForestRegressor(n_estimators=50, random_state=0)
```

RF Model Testing

```
# Predicting the target values of the test set
y_pred = regressor.predict(X_test)
# Visualization
#fig, ax = plt.subplots( nrows=1, ncols=1 )  # create figure & 1 axis

plt.scatter(y_test, y_pred, s=20, marker='o', edgecolor=['blue'], c='none'
)
plt.xlabel('Actual price', fontsize=20)
plt.ylabel('Predicted price', fontsize=20)
plt.show()
```

```
#fig.savefig('/content/drive/MyDrive/ColabNotebooks/CS5812/RandomForestMod
el.png')   # save the figure to file
#plt.close(fig)    # close the figure window
```



RF Model Evaluation

```
# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)


RMSE:  19.476
```

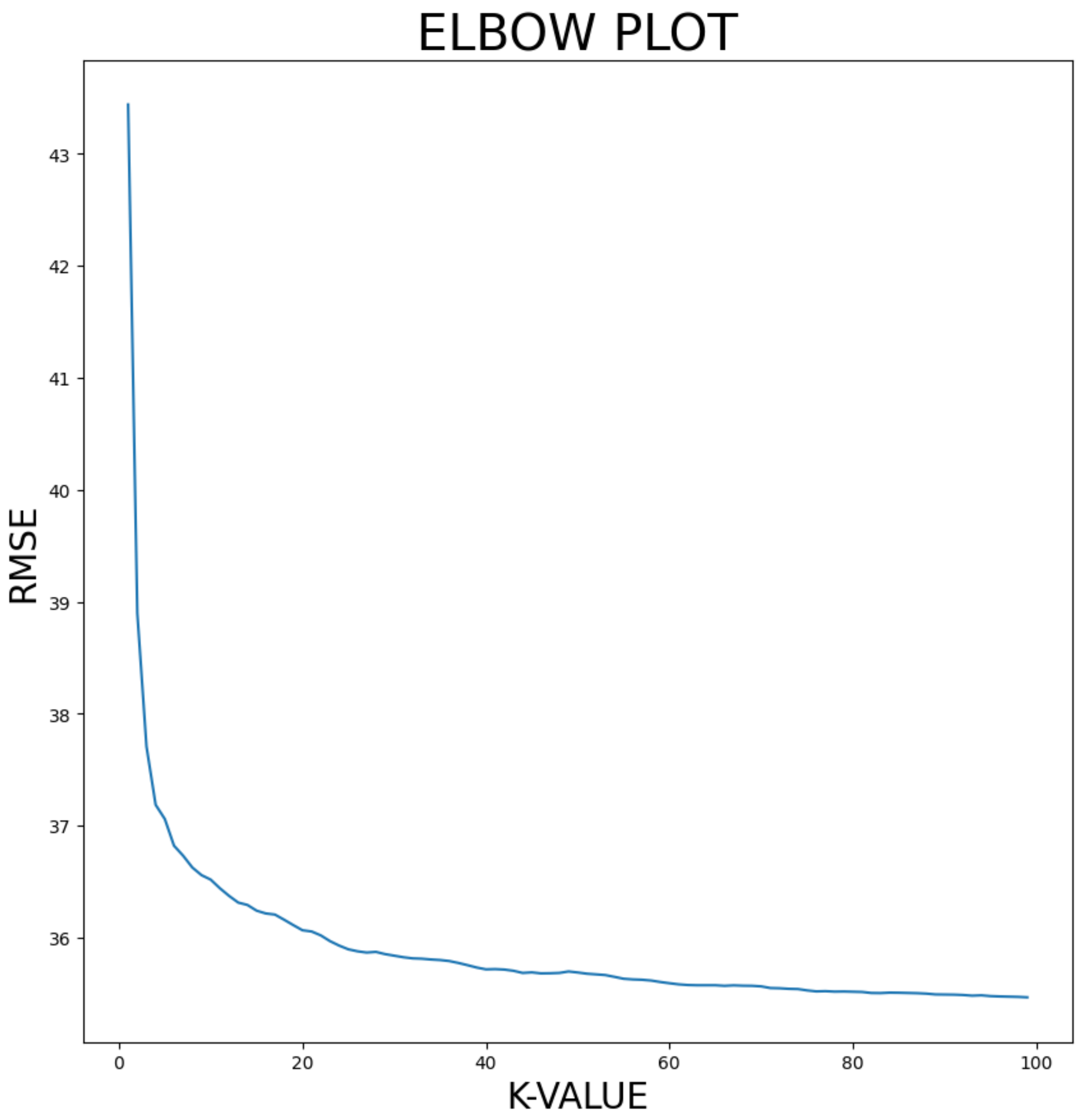
Machine Learning Method 2 - k-Nearest Neighbours

```
X = data.iloc[:,:-1]    # convert the input data to be an array
y = data.iloc[:,-1:]
print ('Shape of input:', X.shape)
print ('Shape of labels:', y.shape)

Shape of input: (36274, 11)
Shape of labels: (36274, 1)

# Randomly split training data and test data with a ratio of 7:3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)
```

KNN Hyperparameters Tuning

```
fig,ax=plt.subplots(figsize=(10,10))
k_list=np.arange(1,100,1)
knn_dict={} # To store k and rmse pairs
for i in k_list:
```

```
#Knn Model Creation
    knn=KNeighborsRegressor(n_neighbors=int(i))
    model_knn=knn.fit(X_train,y_train)
    y_knn_pred=model_knn.predict(X_test)
#Storing MSE
    #mse=mean_squared_error(y_test,y_knn_pred)
    rmse = float(format(np.sqrt(mean_squared_error(y_test, y_knn_pred)), '
.3f'))
    knn_dict[i]=rmse
#Plotting the results
ax.plot(knn_dict.keys(),knn_dict.values())
ax.set_xlabel('K-VALUE', fontsize=20)
ax.set_ylabel('RMSE' ,fontsize=20)
ax.set_title('ELBOW PLOT' ,fontsize=28)

#fig.savefig('/content/drive/MyDrive/ColabNotebooks/CS5812/kNNModel.png')
# save the figure to file
#plt.close(fig)    # close the figure window

Text(0.5, 1.0, 'ELBOW PLOT')
```



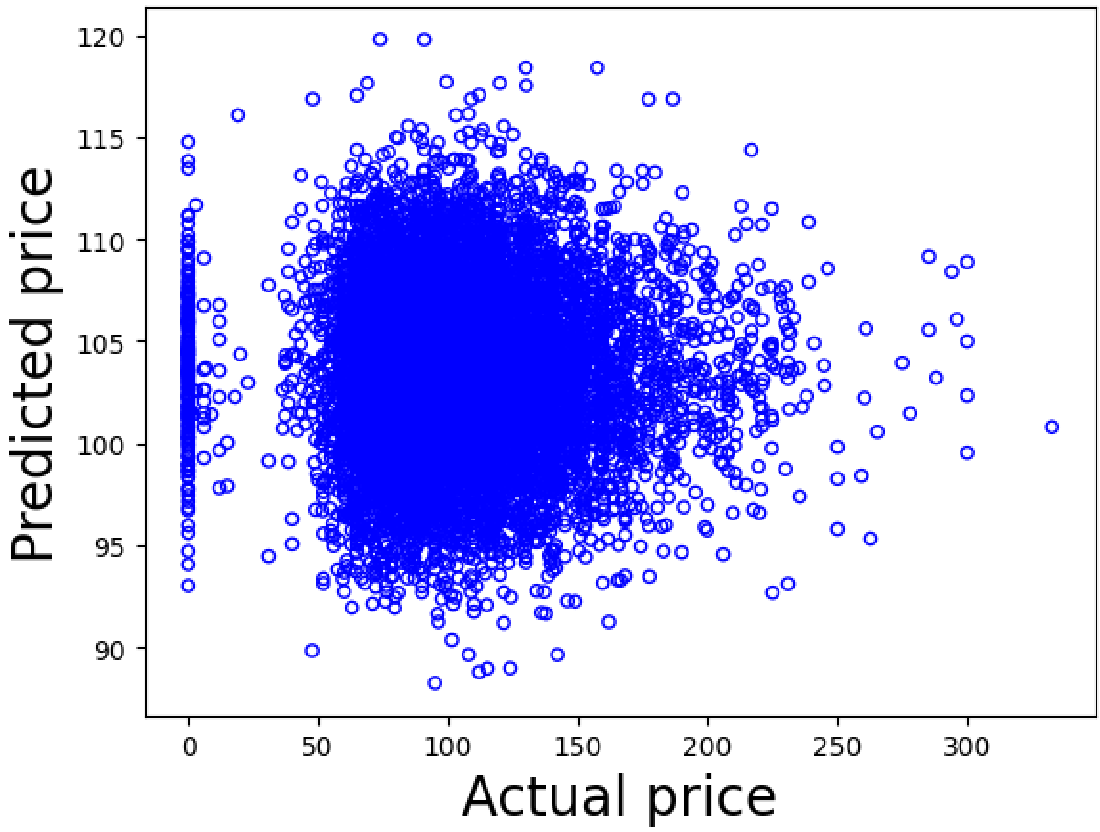
We have the best k-value when RMSE is lowest at k = 70. After this, the RMSE value no longer falls. So we can recreate our model with the optimal k-value.

KNN Model Training

```
#Knn Model Training
knn=KNeighborsRegressor(n_neighbors=int(70))
model_knn=knn.fit(X_train,y_train)
```

KNN Model Testing

```
# Predicting the target values of the test set
y_knn_pred = model_knn.predict(X_test)
# Visualization
#fig, ax = plt.subplots( nrows=1, ncols=1 )  # create figure & 1 axis

plt.scatter(y_test, y_knn_pred, s=20, marker='o', edgecolor=['blue'], c='n
one')
plt.xlabel('Actual price', fontsize=20)
plt.ylabel('Predicted price', fontsize=20)
plt.show()

#fig.savefig('/content/drive/MyDrive/ColabNotebooks/CS5812/kNNModel.png')
# save the figure to file
#plt.close(fig)    # close the figure window
```



KNN Model Evaluation

```
# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_knn_pred)), '.3f'
))
print("\nRMSE: ", rmse)
```

```
RMSE:  35.567
```

Deep Learning Regression Model (Convolution Neural Network-CNN)

```
X = data.iloc[:,:-1]
y = data.iloc[:,-1:].to_numpy().reshape(-1, 1)
print ('Shape of input:', X.shape)
print ('Shape of labels:', y.shape)

Shape of input: (36274, 11)
Shape of labels: (36274, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42) # Randomly split training data and test data with a ratio
of 7:3

X_train = np.expand_dims(X_train, axis=2) # Add a dimension for each train
ing sample - 1 x 11 x 1
X_train = np.expand_dims(X_train, axis=2) # Add a dimension for each train
ing sample to form an image - 1 x 11 x 1 x 1

X_test = np.expand_dims(X_test, axis=2)   # Add a dimension for each testi
ng sample - 1 x 11 x 1
X_test = np.expand_dims(X_test, axis=2)   # Add a dimension for each testi
ng sample to form an image - 1 x 11 x 1 x 1

print (X_train.shape)
print (X_test.shape)

(25391, 11, 1, 1)
(10883, 11, 1, 1)
```

CNN Model Training

```
# LeNet-5 structure for model training
model = keras.Sequential()
model.add(keras.layers.Conv2D(filters=6, kernel_size=(5,1), strides=(1,1),
padding='same', activation='relu', input_shape=(11, 1, 1)))  # The first c
onvolutional layer. Input shape is 11x1x1 and activation function is Recti
fied Linear Unit (ReLU).\
                                                            # Kernel size a
nd stride are set as (5,1) and (1,1), respectively. The number of convolut
ional kernels is 6. Besides, the zero padding is 'same', thus the output s
hape is (11/1) x 1 x 6 = 11x1x6, which is the same as the input shape.
model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1)))  # The f
irst maxpooling layer. The pool size and stride are both (2,1), thus the o
utput shape is 5x1x6, where 5 = (11-2)/2 + 1.
model.add(keras.layers.Conv2D(filters=16, kernel_size=(5,1), strides=(1,1)
, padding='same', activation='relu')) # The second convolutional layer. Th
e output shape is 5x1x16, where 5 = 5/1.
model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1)))  # The s
econd maxpooling layer. The pool size and stride are both (2,1). The outpu
t shape is 2x1x16, where 2 = (5-2)/2 +1.
model.add(keras.layers.Flatten())                           # The previous
output is flattened to be a vector.
model.add(keras.layers.Dense(120, activation='relu'))       # The first ful
ly connected layer.
```

```
model.add(keras.layers.Dense(84, activation='relu'))       # The second fu
lly connected layer.
model.add(keras.layers.Dense(1))                           # we have 1 neu
ron for output (SalePrice).
model.summary()                                            # Summary the c
onstructed model.
model.compile(tf.keras.optimizers.SGD(learning_rate = 1e-11), 'mean_square
d_error') # Model construction with a SGD optimizer and a mean squared err
or loss function.
model.fit(X_train, y_train, epochs = 100, batch_size = 64, verbose = 2)
# Model training with some hyperparameters.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 11, 1, 6)          36

 max_pooling2d (MaxPooling2D  (None, 5, 1, 6)          0
 )

 conv2d_1 (Conv2D)           (None, 5, 1, 16)          496

 max_pooling2d_1 (MaxPooling  (None, 2, 1, 16)         0
 2D)

 flatten (Flatten)           (None, 32)                0

 dense (Dense)               (None, 120)               3960

 dense_1 (Dense)             (None, 84)                10164

 dense_2 (Dense)             (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
397/397 - 6s - loss: 662342.8750 - 6s/epoch - 16ms/step
Epoch 2/100
397/397 - 3s - loss: 238289.7969 - 3s/epoch - 6ms/step
Epoch 3/100
397/397 - 4s - loss: 94334.7578 - 4s/epoch - 10ms/step
Epoch 4/100
397/397 - 2s - loss: 41422.5703 - 2s/epoch - 6ms/step
Epoch 5/100
397/397 - 2s - loss: 20903.1738 - 2s/epoch - 6ms/step
Epoch 6/100
397/397 - 2s - loss: 11729.2764 - 2s/epoch - 6ms/step
Epoch 7/100
397/397 - 3s - loss: 7701.0835 - 3s/epoch - 7ms/step
Epoch 8/100
397/397 - 3s - loss: 5824.9155 - 3s/epoch - 9ms/step
Epoch 9/100
```

```
397/397 - 3s - loss: 4872.9097 - 3s/epoch - 7ms/step
Epoch 10/100
397/397 - 2s - loss: 4388.8340 - 2s/epoch - 6ms/step
Epoch 11/100
397/397 - 2s - loss: 4142.4434 - 2s/epoch - 4ms/step
Epoch 12/100
397/397 - 2s - loss: 4016.8147 - 2s/epoch - 5ms/step
Epoch 13/100
397/397 - 1s - loss: 3952.6965 - 1s/epoch - 3ms/step
Epoch 14/100
397/397 - 1s - loss: 3919.8892 - 948ms/epoch - 2ms/step
Epoch 15/100
397/397 - 1s - loss: 3903.1907 - 1s/epoch - 3ms/step
Epoch 16/100
397/397 - 1s - loss: 3894.6360 - 1s/epoch - 4ms/step
Epoch 17/100
397/397 - 1s - loss: 3890.2695 - 1s/epoch - 3ms/step
Epoch 18/100
397/397 - 1s - loss: 3888.0615 - 1s/epoch - 3ms/step
Epoch 19/100
397/397 - 1s - loss: 3886.9314 - 932ms/epoch - 2ms/step
Epoch 20/100
397/397 - 1s - loss: 3886.3333 - 935ms/epoch - 2ms/step
Epoch 21/100
397/397 - 1s - loss: 3886.0164 - 988ms/epoch - 2ms/step
Epoch 22/100
397/397 - 1s - loss: 3885.8616 - 920ms/epoch - 2ms/step
Epoch 23/100
397/397 - 1s - loss: 3885.7886 - 934ms/epoch - 2ms/step
Epoch 24/100
397/397 - 1s - loss: 3885.7483 - 921ms/epoch - 2ms/step
Epoch 25/100
397/397 - 1s - loss: 3885.7268 - 932ms/epoch - 2ms/step
Epoch 26/100
397/397 - 1s - loss: 3885.7136 - 908ms/epoch - 2ms/step
Epoch 27/100
397/397 - 1s - loss: 3885.7024 - 1s/epoch - 3ms/step
Epoch 28/100
397/397 - 1s - loss: 3885.7046 - 929ms/epoch - 2ms/step
Epoch 29/100
397/397 - 1s - loss: 3885.7061 - 1s/epoch - 3ms/step
Epoch 30/100
397/397 - 1s - loss: 3885.7019 - 1s/epoch - 4ms/step
Epoch 31/100
397/397 - 2s - loss: 3885.7068 - 2s/epoch - 4ms/step
Epoch 32/100
397/397 - 1s - loss: 3885.7034 - 1s/epoch - 3ms/step
Epoch 33/100
397/397 - 1s - loss: 3885.7039 - 975ms/epoch - 2ms/step
Epoch 34/100
397/397 - 1s - loss: 3885.7009 - 891ms/epoch - 2ms/step
Epoch 35/100
397/397 - 1s - loss: 3885.7041 - 931ms/epoch - 2ms/step
Epoch 36/100
397/397 - 1s - loss: 3885.6995 - 918ms/epoch - 2ms/step
```

```
Epoch 37/100
397/397 - 1s - loss: 3885.7021 - 893ms/epoch - 2ms/step
Epoch 38/100
397/397 - 1s - loss: 3885.7063 - 972ms/epoch - 2ms/step
Epoch 39/100
397/397 - 1s - loss: 3885.7043 - 930ms/epoch - 2ms/step
Epoch 40/100
397/397 - 1s - loss: 3885.7046 - 958ms/epoch - 2ms/step
Epoch 41/100
397/397 - 1s - loss: 3885.7039 - 983ms/epoch - 2ms/step
Epoch 42/100
397/397 - 1s - loss: 3885.6990 - 1s/epoch - 3ms/step
Epoch 43/100
397/397 - 2s - loss: 3885.6980 - 2s/epoch - 4ms/step
Epoch 44/100
397/397 - 2s - loss: 3885.7068 - 2s/epoch - 4ms/step
Epoch 45/100
397/397 - 1s - loss: 3885.7039 - 1s/epoch - 4ms/step
Epoch 46/100
397/397 - 1s - loss: 3885.7000 - 995ms/epoch - 3ms/step
Epoch 47/100
397/397 - 1s - loss: 3885.7034 - 1s/epoch - 3ms/step
Epoch 48/100
397/397 - 1s - loss: 3885.7021 - 960ms/epoch - 2ms/step
Epoch 49/100
397/397 - 1s - loss: 3885.6980 - 959ms/epoch - 2ms/step
Epoch 50/100
397/397 - 1s - loss: 3885.7019 - 926ms/epoch - 2ms/step
Epoch 51/100
397/397 - 1s - loss: 3885.6990 - 997ms/epoch - 3ms/step
Epoch 52/100
397/397 - 1s - loss: 3885.7024 - 901ms/epoch - 2ms/step
Epoch 53/100
397/397 - 1s - loss: 3885.7041 - 919ms/epoch - 2ms/step
Epoch 54/100
397/397 - 1s - loss: 3885.7021 - 877ms/epoch - 2ms/step
Epoch 55/100
397/397 - 1s - loss: 3885.7017 - 947ms/epoch - 2ms/step
Epoch 56/100
397/397 - 1s - loss: 3885.6997 - 1s/epoch - 3ms/step
Epoch 57/100
397/397 - 1s - loss: 3885.7009 - 1s/epoch - 4ms/step
Epoch 58/100
397/397 - 2s - loss: 3885.7009 - 2s/epoch - 4ms/step
Epoch 59/100
397/397 - 1s - loss: 3885.7024 - 1s/epoch - 3ms/step
Epoch 60/100
397/397 - 1s - loss: 3885.6909 - 1s/epoch - 3ms/step
Epoch 61/100
397/397 - 1s - loss: 3885.6978 - 1s/epoch - 3ms/step
Epoch 62/100
397/397 - 1s - loss: 3885.7043 - 961ms/epoch - 2ms/step
Epoch 63/100
397/397 - 1s - loss: 3885.7007 - 941ms/epoch - 2ms/step
Epoch 64/100
```

```
397/397 - 1s - loss: 3885.7012 - 935ms/epoch - 2ms/step
Epoch 65/100
397/397 - 1s - loss: 3885.7034 - 937ms/epoch - 2ms/step
Epoch 66/100
397/397 - 1s - loss: 3885.6997 - 1s/epoch - 3ms/step
Epoch 67/100
397/397 - 1s - loss: 3885.7039 - 958ms/epoch - 2ms/step
Epoch 68/100
397/397 - 1s - loss: 3885.6975 - 979ms/epoch - 2ms/step
Epoch 69/100
397/397 - 1s - loss: 3885.7002 - 1s/epoch - 4ms/step
Epoch 70/100
397/397 - 1s - loss: 3885.7056 - 1s/epoch - 4ms/step
Epoch 71/100
397/397 - 1s - loss: 3885.6985 - 1s/epoch - 4ms/step
Epoch 72/100
397/397 - 1s - loss: 3885.6990 - 1s/epoch - 3ms/step
Epoch 73/100
397/397 - 1s - loss: 3885.6963 - 941ms/epoch - 2ms/step
Epoch 74/100
397/397 - 1s - loss: 3885.7024 - 1s/epoch - 3ms/step
Epoch 75/100
397/397 - 1s - loss: 3885.6948 - 966ms/epoch - 2ms/step
Epoch 76/100
397/397 - 1s - loss: 3885.7029 - 972ms/epoch - 2ms/step
Epoch 77/100
397/397 - 1s - loss: 3885.7000 - 941ms/epoch - 2ms/step
Epoch 78/100
397/397 - 1s - loss: 3885.6997 - 949ms/epoch - 2ms/step
Epoch 79/100
397/397 - 1s - loss: 3885.7039 - 970ms/epoch - 2ms/step
Epoch 80/100
397/397 - 1s - loss: 3885.7019 - 969ms/epoch - 2ms/step
Epoch 81/100
397/397 - 1s - loss: 3885.7046 - 920ms/epoch - 2ms/step
Epoch 82/100
397/397 - 1s - loss: 3885.7029 - 1s/epoch - 3ms/step
Epoch 83/100
397/397 - 1s - loss: 3885.6995 - 1s/epoch - 4ms/step
Epoch 84/100
397/397 - 1s - loss: 3885.6990 - 1s/epoch - 4ms/step
Epoch 85/100
397/397 - 1s - loss: 3885.7061 - 1s/epoch - 3ms/step
Epoch 86/100
397/397 - 1s - loss: 3885.7007 - 931ms/epoch - 2ms/step
Epoch 87/100
397/397 - 1s - loss: 3885.6968 - 880ms/epoch - 2ms/step
Epoch 88/100
397/397 - 1s - loss: 3885.7019 - 972ms/epoch - 2ms/step
Epoch 89/100
397/397 - 1s - loss: 3885.7029 - 987ms/epoch - 2ms/step
Epoch 90/100
397/397 - 1s - loss: 3885.7029 - 1s/epoch - 3ms/step
Epoch 91/100
397/397 - 1s - loss: 3885.7012 - 946ms/epoch - 2ms/step
```

```
Epoch 92/100
397/397 - 1s - loss: 3885.7041 - 909ms/epoch - 2ms/step
Epoch 93/100
397/397 - 1s - loss: 3885.6997 - 1s/epoch - 3ms/step
Epoch 94/100
397/397 - 1s - loss: 3885.7012 - 1s/epoch - 3ms/step
Epoch 95/100
397/397 - 1s - loss: 3885.7017 - 1s/epoch - 3ms/step
Epoch 96/100
397/397 - 1s - loss: 3885.6978 - 1s/epoch - 4ms/step
Epoch 97/100
397/397 - 2s - loss: 3885.7056 - 2s/epoch - 4ms/step
Epoch 98/100
397/397 - 2s - loss: 3885.7021 - 2s/epoch - 5ms/step
Epoch 99/100
397/397 - 1s - loss: 3885.7041 - 936ms/epoch - 2ms/step
Epoch 100/100
397/397 - 1s - loss: 3885.7085 - 950ms/epoch - 2ms/step

<keras.callbacks.History at 0x7f58281a6eb0>
```

##CNN Hyperparameter Tuning

Finding the best parameters using manual grid search

```
# Defining a function to find the best parameters for CNN
def FunctionFindBestParams(X_train, y_train, X_test, y_test):
    # Defining the list of hyper parameters to try
    batch_size_list=[5, 10, 15, 20]
    epoch_list  =   [5, 10, 50, 100]
    SearchResultsData=pd.DataFrame(columns=['TrialNumber', 'Parameters', '
\nRMSE'])
    # initializing the trials
    TrialNumber=0
    for batch_size_trial in batch_size_list:
        for epochs_trial in epoch_list:
            TrialNumber+=1
            # create CNN model
            model = keras.Sequential()
            # Defining the first layer of the model
            model.add(keras.layers.Conv2D(filters=6, kernel_size=(5,1), st
rides=(1,1), padding='same', activation='relu', input_shape=(11, 1, 1)))#
Kernel size and stride are set as (5,1) and (1,1), respectively. The numbe
r of convolutional kernels is 6. Besides, the zero padding is 'same', thus
the output shape is (11/1) x 1 x 6 = 11x1x6, which is the same as the inpu
t shape.
            model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1
)))  # The first maxpooling layer. The pool size and stride are both (2,1)
, thus the output shape is 5x1x6, where 5 = (11-2)/2 + 1.
            model.add(keras.layers.Conv2D(filters=16, kernel_size=(5,1), s
trides=(1,1), padding='same', activation='relu')) # The second convolution
al layer. The output shape is 5x1x16, where 5 = 5/1.
            model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1
)))  # The second maxpooling layer. The pool size and stride are both (2,1
). The output shape is 2x1x16, where 2 = (5-2)/2 +1.
            model.add(keras.layers.Flatten())                            # T
he previous output is flattened to be a vector.
```

```
            model.add(keras.layers.Dense(120, activation='relu'))      # T
he first fully connected layer.
            model.add(keras.layers.Dense(84, activation='relu'))       # T
he second fully connected layer.
            model.add(keras.layers.Dense(1))                           # w
e have 1 neuron for output (SalePrice).
            model.summary()                                            # S
ummary the constructed model.
            model.compile(tf.keras.optimizers.SGD(learning_rate = 1e-11),
'mean_squared_error') # Model construction with a SGD optimizer and a mean
squared error loss function.
            model.fit(X_train, y_train, epochs = epochs_trial, batch_size
= batch_size_trial, verbose = 2)            # Model training with some hy
perparameters.
            rmse = float(format(np.sqrt(mean_squared_error(y_test, model.p
redict(X_test))), '.3f'))
            # printing the results of the current iteration
            print(TrialNumber, 'Parameters:','batch_size:', batch_size_tri
al,'-', 'epochs:',epochs_trial, '\nRMSE: ', rmse)
            SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[
[TrialNumber, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],
                                                                    column
s=['TrialNumber', 'Parameters', '\nRMSE'] ))
    return(SearchResultsData)
######################################################
# Calling the function
ResultsData=FunctionFindBestParams(X_train, y_train, X_test, y_test)

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 11, 1, 6)          36

 max_pooling2d_2 (MaxPooling  (None, 5, 1, 6)          0
 2D)

 conv2d_3 (Conv2D)           (None, 5, 1, 16)          496

 max_pooling2d_3 (MaxPooling  (None, 2, 1, 16)         0
 2D)

 flatten_1 (Flatten)         (None, 32)                0

 dense_3 (Dense)             (None, 120)               3960

 dense_4 (Dense)             (None, 84)                10164

 dense_5 (Dense)             (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
```

```
5079/5079 - 10s - loss: 9476.1729 - 10s/epoch - 2ms/step
Epoch 2/5
5079/5079 - 10s - loss: 3891.9038 - 10s/epoch - 2ms/step
Epoch 3/5
5079/5079 - 9s - loss: 3891.9075 - 9s/epoch - 2ms/step
Epoch 4/5
5079/5079 - 9s - loss: 3891.4094 - 9s/epoch - 2ms/step
Epoch 5/5
5079/5079 - 10s - loss: 3891.9082 - 10s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
1 Parameters: batch_size: 5 - epochs: 5
RMSE:  62.811
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_4 (Conv2D)           (None, 11, 1, 6)          36        
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 5, 1, 6)          0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 5, 1, 16)          496       
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 2, 1, 16)         0         
 2D)                                                             
                                                                 
 flatten_2 (Flatten)         (None, 32)                0         
                                                                 
 dense_6 (Dense)             (None, 120)               3960      
                                                                 
 dense_7 (Dense)             (None, 84)                10164     
                                                                 
 dense_8 (Dense)             (None, 1)                 85        
                                                                 
=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pan das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/10
5079/5079 - 9s - loss: 38369.3945 - 9s/epoch - 2ms/step
Epoch 2/10
5079/5079 - 10s - loss: 3899.1685 - 10s/epoch - 2ms/step
Epoch 3/10
5079/5079 - 10s - loss: 3899.5176 - 10s/epoch - 2ms/step
Epoch 4/10
5079/5079 - 9s - loss: 3898.6858 - 9s/epoch - 2ms/step
Epoch 5/10
5079/5079 - 10s - loss: 3899.2976 - 10s/epoch - 2ms/step
```

```
Epoch 6/10
5079/5079 - 10s - loss: 3899.3735 - 10s/epoch - 2ms/step
Epoch 7/10
5079/5079 - 11s - loss: 3899.3987 - 11s/epoch - 2ms/step
Epoch 8/10
5079/5079 - 8s - loss: 3899.5146 - 8s/epoch - 2ms/step
Epoch 9/10
5079/5079 - 10s - loss: 3898.9185 - 10s/epoch - 2ms/step
Epoch 10/10
5079/5079 - 9s - loss: 3899.1699 - 9s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
2 Parameters: batch_size: 5 - epochs: 10
RMSE:  62.864
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 11, 1, 6)          36

 max_pooling2d_6 (MaxPooling  (None, 5, 1, 6)          0
 2D)

 conv2d_7 (Conv2D)           (None, 5, 1, 16)          496

 max_pooling2d_7 (MaxPooling  (None, 2, 1, 16)         0
 2D)

 flatten_3 (Flatten)         (None, 32)                0

 dense_9 (Dense)             (None, 120)               3960

 dense_10 (Dense)            (None, 84)                10164

 dense_11 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/50
5079/5079 - 11s - loss: 53307.6055 - 11s/epoch - 2ms/step
Epoch 2/50
5079/5079 - 10s - loss: 3894.4004 - 10s/epoch - 2ms/step
Epoch 3/50
5079/5079 - 9s - loss: 3894.4109 - 9s/epoch - 2ms/step
Epoch 4/50
5079/5079 - 10s - loss: 3894.4316 - 10s/epoch - 2ms/step
Epoch 5/50
```

```
5079/5079 - 10s - loss: 3894.3506 - 10s/epoch - 2ms/step
Epoch 6/50
5079/5079 - 10s - loss: 3894.3950 - 10s/epoch - 2ms/step
Epoch 7/50
5079/5079 - 9s - loss: 3894.2769 - 9s/epoch - 2ms/step
Epoch 8/50
5079/5079 - 10s - loss: 3894.1870 - 10s/epoch - 2ms/step
Epoch 9/50
5079/5079 - 9s - loss: 3894.4534 - 9s/epoch - 2ms/step
Epoch 10/50
5079/5079 - 10s - loss: 3894.3259 - 10s/epoch - 2ms/step
Epoch 11/50
5079/5079 - 10s - loss: 3894.5090 - 10s/epoch - 2ms/step
Epoch 12/50
5079/5079 - 10s - loss: 3894.1877 - 10s/epoch - 2ms/step
Epoch 13/50
5079/5079 - 9s - loss: 3894.1575 - 9s/epoch - 2ms/step
Epoch 14/50
5079/5079 - 10s - loss: 3894.3525 - 10s/epoch - 2ms/step
Epoch 15/50
5079/5079 - 10s - loss: 3894.4688 - 10s/epoch - 2ms/step
Epoch 16/50
5079/5079 - 8s - loss: 3894.1843 - 8s/epoch - 2ms/step
Epoch 17/50
5079/5079 - 10s - loss: 3894.5024 - 10s/epoch - 2ms/step
Epoch 18/50
5079/5079 - 10s - loss: 3894.0830 - 10s/epoch - 2ms/step
Epoch 19/50
5079/5079 - 9s - loss: 3894.5139 - 9s/epoch - 2ms/step
Epoch 20/50
5079/5079 - 10s - loss: 3894.3447 - 10s/epoch - 2ms/step
Epoch 21/50
5079/5079 - 10s - loss: 3894.4014 - 10s/epoch - 2ms/step
Epoch 22/50
5079/5079 - 8s - loss: 3894.1956 - 8s/epoch - 2ms/step
Epoch 23/50
5079/5079 - 10s - loss: 3894.2854 - 10s/epoch - 2ms/step
Epoch 24/50
5079/5079 - 9s - loss: 3894.2998 - 9s/epoch - 2ms/step
Epoch 25/50
5079/5079 - 9s - loss: 3894.5315 - 9s/epoch - 2ms/step
Epoch 26/50
5079/5079 - 10s - loss: 3894.4917 - 10s/epoch - 2ms/step
Epoch 27/50
5079/5079 - 10s - loss: 3894.3513 - 10s/epoch - 2ms/step
Epoch 28/50
5079/5079 - 9s - loss: 3894.3079 - 9s/epoch - 2ms/step
Epoch 29/50
5079/5079 - 10s - loss: 3894.2625 - 10s/epoch - 2ms/step
Epoch 30/50
5079/5079 - 11s - loss: 3894.0491 - 11s/epoch - 2ms/step
Epoch 31/50
5079/5079 - 9s - loss: 3894.4983 - 9s/epoch - 2ms/step
Epoch 32/50
5079/5079 - 10s - loss: 3894.4480 - 10s/epoch - 2ms/step
```

```
Epoch 33/50
5079/5079 - 10s - loss: 3894.2620 - 10s/epoch - 2ms/step
Epoch 34/50
5079/5079 - 8s - loss: 3894.4883 - 8s/epoch - 2ms/step
Epoch 35/50
5079/5079 - 10s - loss: 3894.3711 - 10s/epoch - 2ms/step
Epoch 36/50
5079/5079 - 10s - loss: 3894.3054 - 10s/epoch - 2ms/step
Epoch 37/50
5079/5079 - 9s - loss: 3894.3486 - 9s/epoch - 2ms/step
Epoch 38/50
5079/5079 - 10s - loss: 3894.2212 - 10s/epoch - 2ms/step
Epoch 39/50
5079/5079 - 10s - loss: 3894.5547 - 10s/epoch - 2ms/step
Epoch 40/50
5079/5079 - 8s - loss: 3894.1965 - 8s/epoch - 2ms/step
Epoch 41/50
5079/5079 - 10s - loss: 3894.4001 - 10s/epoch - 2ms/step
Epoch 42/50
5079/5079 - 10s - loss: 3894.4658 - 10s/epoch - 2ms/step
Epoch 43/50
5079/5079 - 8s - loss: 3894.3130 - 8s/epoch - 2ms/step
Epoch 44/50
5079/5079 - 10s - loss: 3894.4351 - 10s/epoch - 2ms/step
Epoch 45/50
5079/5079 - 10s - loss: 3894.1965 - 10s/epoch - 2ms/step
Epoch 46/50
5079/5079 - 8s - loss: 3894.5359 - 8s/epoch - 2ms/step
Epoch 47/50
5079/5079 - 9s - loss: 3894.3860 - 9s/epoch - 2ms/step
Epoch 48/50
5079/5079 - 9s - loss: 3894.3203 - 9s/epoch - 2ms/step
Epoch 49/50
5079/5079 - 8s - loss: 3894.3691 - 8s/epoch - 2ms/step
Epoch 50/50
5079/5079 - 9s - loss: 3894.4102 - 9s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
3 Parameters: batch_size: 5 - epochs: 50
RMSE:  62.834
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 11, 1, 6)          36

 max_pooling2d_8 (MaxPooling  (None, 5, 1, 6)          0
 2D)

 conv2d_9 (Conv2D)           (None, 5, 1, 16)          496

 max_pooling2d_9 (MaxPooling  (None, 2, 1, 16)         0
 2D)

 flatten_4 (Flatten)         (None, 32)                0
```

```
 dense_12 (Dense)           (None, 120)              3960

 dense_13 (Dense)           (None, 84)               10164

 dense_14 (Dense)           (None, 1)                85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/100
5079/5079 - 10s - loss: 20800.9336 - 10s/epoch - 2ms/step
Epoch 2/100
5079/5079 - 10s - loss: 3899.6895 - 10s/epoch - 2ms/step
Epoch 3/100
5079/5079 - 8s - loss: 3893.8340 - 8s/epoch - 2ms/step
Epoch 4/100
5079/5079 - 9s - loss: 3893.7705 - 9s/epoch - 2ms/step
Epoch 5/100
5079/5079 - 10s - loss: 3893.7673 - 10s/epoch - 2ms/step
Epoch 6/100
5079/5079 - 8s - loss: 3893.8184 - 8s/epoch - 2ms/step
Epoch 7/100
5079/5079 - 9s - loss: 3893.8462 - 9s/epoch - 2ms/step
Epoch 8/100
5079/5079 - 9s - loss: 3893.8179 - 9s/epoch - 2ms/step
Epoch 9/100
5079/5079 - 8s - loss: 3893.8074 - 8s/epoch - 2ms/step
Epoch 10/100
5079/5079 - 10s - loss: 3893.7542 - 10s/epoch - 2ms/step
Epoch 11/100
5079/5079 - 10s - loss: 3893.8948 - 10s/epoch - 2ms/step
Epoch 12/100
5079/5079 - 9s - loss: 3893.8250 - 9s/epoch - 2ms/step
Epoch 13/100
5079/5079 - 9s - loss: 3893.8264 - 9s/epoch - 2ms/step
Epoch 14/100
5079/5079 - 9s - loss: 3893.7122 - 9s/epoch - 2ms/step
Epoch 15/100
5079/5079 - 8s - loss: 3893.9316 - 8s/epoch - 2ms/step
Epoch 16/100
5079/5079 - 9s - loss: 3893.8562 - 9s/epoch - 2ms/step
Epoch 17/100
5079/5079 - 10s - loss: 3893.7993 - 10s/epoch - 2ms/step
Epoch 18/100
5079/5079 - 9s - loss: 3893.7346 - 9s/epoch - 2ms/step
Epoch 19/100
5079/5079 - 9s - loss: 3893.7693 - 9s/epoch - 2ms/step
```

```
Epoch 20/100
5079/5079 - 9s - loss: 3893.8499 - 9s/epoch - 2ms/step
Epoch 21/100
5079/5079 - 9s - loss: 3893.8564 - 9s/epoch - 2ms/step
Epoch 22/100
5079/5079 - 8s - loss: 3893.8242 - 8s/epoch - 2ms/step
Epoch 23/100
5079/5079 - 9s - loss: 3893.8560 - 9s/epoch - 2ms/step
Epoch 24/100
5079/5079 - 9s - loss: 3893.8074 - 9s/epoch - 2ms/step
Epoch 25/100
5079/5079 - 8s - loss: 3893.8149 - 8s/epoch - 2ms/step
Epoch 26/100
5079/5079 - 10s - loss: 3893.6978 - 10s/epoch - 2ms/step
Epoch 27/100
5079/5079 - 9s - loss: 3893.8286 - 9s/epoch - 2ms/step
Epoch 28/100
5079/5079 - 8s - loss: 3893.8450 - 8s/epoch - 2ms/step
Epoch 29/100
5079/5079 - 9s - loss: 3893.7617 - 9s/epoch - 2ms/step
Epoch 30/100
5079/5079 - 9s - loss: 3893.8953 - 9s/epoch - 2ms/step
Epoch 31/100
5079/5079 - 8s - loss: 3893.7695 - 8s/epoch - 2ms/step
Epoch 32/100
5079/5079 - 9s - loss: 3893.8093 - 9s/epoch - 2ms/step
Epoch 33/100
5079/5079 - 10s - loss: 3893.7910 - 10s/epoch - 2ms/step
Epoch 34/100
5079/5079 - 8s - loss: 3893.8601 - 8s/epoch - 2ms/step
Epoch 35/100
5079/5079 - 10s - loss: 3893.8298 - 10s/epoch - 2ms/step
Epoch 36/100
5079/5079 - 10s - loss: 3893.6262 - 10s/epoch - 2ms/step
Epoch 37/100
5079/5079 - 9s - loss: 3893.8604 - 9s/epoch - 2ms/step
Epoch 38/100
5079/5079 - 10s - loss: 3893.8418 - 10s/epoch - 2ms/step
Epoch 39/100
5079/5079 - 10s - loss: 3893.7932 - 10s/epoch - 2ms/step
Epoch 40/100
5079/5079 - 8s - loss: 3893.8311 - 8s/epoch - 2ms/step
Epoch 41/100
5079/5079 - 9s - loss: 3893.7935 - 9s/epoch - 2ms/step
Epoch 42/100
5079/5079 - 10s - loss: 3893.8633 - 10s/epoch - 2ms/step
Epoch 43/100
5079/5079 - 9s - loss: 3893.3948 - 9s/epoch - 2ms/step
Epoch 44/100
5079/5079 - 9s - loss: 3893.6843 - 9s/epoch - 2ms/step
Epoch 45/100
5079/5079 - 9s - loss: 3893.7913 - 9s/epoch - 2ms/step
Epoch 46/100
5079/5079 - 9s - loss: 3893.6184 - 9s/epoch - 2ms/step
Epoch 47/100
```

```
5079/5079 - 9s - loss: 3893.8799 - 9s/epoch - 2ms/step
Epoch 48/100
5079/5079 - 9s - loss: 3893.8345 - 9s/epoch - 2ms/step
Epoch 49/100
5079/5079 - 8s - loss: 3893.7734 - 8s/epoch - 2ms/step
Epoch 50/100
5079/5079 - 9s - loss: 3893.8076 - 9s/epoch - 2ms/step
Epoch 51/100
5079/5079 - 9s - loss: 3893.7158 - 9s/epoch - 2ms/step
Epoch 52/100
5079/5079 - 9s - loss: 3893.7627 - 9s/epoch - 2ms/step
Epoch 53/100
5079/5079 - 9s - loss: 3893.8564 - 9s/epoch - 2ms/step
Epoch 54/100
5079/5079 - 9s - loss: 3893.8491 - 9s/epoch - 2ms/step
Epoch 55/100
5079/5079 - 8s - loss: 3893.7793 - 8s/epoch - 2ms/step
Epoch 56/100
5079/5079 - 9s - loss: 3893.8530 - 9s/epoch - 2ms/step
Epoch 57/100
5079/5079 - 9s - loss: 3893.7581 - 9s/epoch - 2ms/step
Epoch 58/100
5079/5079 - 8s - loss: 3893.8318 - 8s/epoch - 2ms/step
Epoch 59/100
5079/5079 - 9s - loss: 3893.7292 - 9s/epoch - 2ms/step
Epoch 60/100
5079/5079 - 9s - loss: 3893.8521 - 9s/epoch - 2ms/step
Epoch 61/100
5079/5079 - 8s - loss: 3893.8477 - 8s/epoch - 2ms/step
Epoch 62/100
5079/5079 - 9s - loss: 3893.8113 - 9s/epoch - 2ms/step
Epoch 63/100
5079/5079 - 9s - loss: 3893.7344 - 9s/epoch - 2ms/step
Epoch 64/100
5079/5079 - 8s - loss: 3893.7856 - 8s/epoch - 2ms/step
Epoch 65/100
5079/5079 - 9s - loss: 3893.6890 - 9s/epoch - 2ms/step
Epoch 66/100
5079/5079 - 9s - loss: 3893.6545 - 9s/epoch - 2ms/step
Epoch 67/100
5079/5079 - 8s - loss: 3893.4949 - 8s/epoch - 2ms/step
Epoch 68/100
5079/5079 - 9s - loss: 3893.7837 - 9s/epoch - 2ms/step
Epoch 69/100
5079/5079 - 9s - loss: 3893.8276 - 9s/epoch - 2ms/step
Epoch 70/100
5079/5079 - 8s - loss: 3893.8552 - 8s/epoch - 2ms/step
Epoch 71/100
5079/5079 - 9s - loss: 3893.8462 - 9s/epoch - 2ms/step
Epoch 72/100
5079/5079 - 8s - loss: 3893.7822 - 8s/epoch - 2ms/step
Epoch 73/100
5079/5079 - 9s - loss: 3893.6941 - 9s/epoch - 2ms/step
Epoch 74/100
5079/5079 - 9s - loss: 3893.8604 - 9s/epoch - 2ms/step
```

```
Epoch 75/100
5079/5079 - 8s - loss: 3893.7632 - 8s/epoch - 2ms/step
Epoch 76/100
5079/5079 - 9s - loss: 3893.7976 - 9s/epoch - 2ms/step
Epoch 77/100
5079/5079 - 9s - loss: 3893.5503 - 9s/epoch - 2ms/step
Epoch 78/100
5079/5079 - 8s - loss: 3893.8755 - 8s/epoch - 2ms/step
Epoch 79/100
5079/5079 - 9s - loss: 3893.8623 - 9s/epoch - 2ms/step
Epoch 80/100
5079/5079 - 9s - loss: 3893.6331 - 9s/epoch - 2ms/step
Epoch 81/100
5079/5079 - 8s - loss: 3893.8530 - 8s/epoch - 2ms/step
Epoch 82/100
5079/5079 - 9s - loss: 3893.6792 - 9s/epoch - 2ms/step
Epoch 83/100
5079/5079 - 9s - loss: 3893.7375 - 9s/epoch - 2ms/step
Epoch 84/100
5079/5079 - 8s - loss: 3893.7310 - 8s/epoch - 2ms/step
Epoch 85/100
5079/5079 - 9s - loss: 3893.7466 - 9s/epoch - 2ms/step
Epoch 86/100
5079/5079 - 9s - loss: 3893.8357 - 9s/epoch - 2ms/step
Epoch 87/100
5079/5079 - 8s - loss: 3893.8286 - 8s/epoch - 2ms/step
Epoch 88/100
5079/5079 - 9s - loss: 3893.7844 - 9s/epoch - 2ms/step
Epoch 89/100
5079/5079 - 8s - loss: 3893.7114 - 8s/epoch - 2ms/step
Epoch 90/100
5079/5079 - 9s - loss: 3893.7607 - 9s/epoch - 2ms/step
Epoch 91/100
5079/5079 - 9s - loss: 3893.5520 - 9s/epoch - 2ms/step
Epoch 92/100
5079/5079 - 8s - loss: 3893.8340 - 8s/epoch - 2ms/step
Epoch 93/100
5079/5079 - 9s - loss: 3893.6699 - 9s/epoch - 2ms/step
Epoch 94/100
5079/5079 - 9s - loss: 3893.8311 - 9s/epoch - 2ms/step
Epoch 95/100
5079/5079 - 8s - loss: 3893.7083 - 8s/epoch - 2ms/step
Epoch 96/100
5079/5079 - 9s - loss: 3893.7781 - 9s/epoch - 2ms/step
Epoch 97/100
5079/5079 - 9s - loss: 3893.8083 - 9s/epoch - 2ms/step
Epoch 98/100
5079/5079 - 8s - loss: 3893.6870 - 8s/epoch - 2ms/step
Epoch 99/100
5079/5079 - 9s - loss: 3893.8433 - 9s/epoch - 2ms/step
Epoch 100/100
5079/5079 - 9s - loss: 3893.8289 - 9s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
4 Parameters: batch_size: 5 - epochs: 100
RMSE:  62.829
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_10 (Conv2D)          (None, 11, 1, 6)          36        
                                                                 
 max_pooling2d_10 (MaxPoolin  (None, 5, 1, 6)          0         
 g2D)                                                            
                                                                 
 conv2d_11 (Conv2D)          (None, 5, 1, 16)          496       
                                                                 
 max_pooling2d_11 (MaxPoolin  (None, 2, 1, 16)         0         
 g2D)                                                            
                                                                 
 flatten_5 (Flatten)         (None, 32)                0         
                                                                 
 dense_15 (Dense)            (None, 120)               3960      
                                                                 
 dense_16 (Dense)            (None, 84)                10164     
                                                                 
 dense_17 (Dense)            (None, 1)                 85        
                                                                 
=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/5
2540/2540 - 6s - loss: 6277.2617 - 6s/epoch - 2ms/step
Epoch 2/5
2540/2540 - 6s - loss: 3896.8516 - 6s/epoch - 2ms/step
Epoch 3/5
2540/2540 - 4s - loss: 3896.9272 - 4s/epoch - 2ms/step
Epoch 4/5
2540/2540 - 4s - loss: 3896.8955 - 4s/epoch - 2ms/step
Epoch 5/5
2540/2540 - 6s - loss: 3896.9019 - 6s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
5 Parameters: batch_size: 10 - epochs: 5
RMSE:  62.854
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_12 (Conv2D)          (None, 11, 1, 6)          36        
                                                                 
 max_pooling2d_12 (MaxPoolin  (None, 5, 1, 6)          0         
 g2D)                                                            
```

```
 conv2d_13 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_13 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_6 (Flatten)         (None, 32)                0

 dense_18 (Dense)            (None, 120)               3960

 dense_19 (Dense)            (None, 84)                10164

 dense_20 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/10
2540/2540 - 5s - loss: 5325.3970 - 5s/epoch - 2ms/step
Epoch 2/10
2540/2540 - 5s - loss: 3896.2422 - 5s/epoch - 2ms/step
Epoch 3/10
2540/2540 - 4s - loss: 3896.4749 - 4s/epoch - 2ms/step
Epoch 4/10
2540/2540 - 5s - loss: 3896.0933 - 5s/epoch - 2ms/step
Epoch 5/10
2540/2540 - 5s - loss: 3896.4641 - 5s/epoch - 2ms/step
Epoch 6/10
2540/2540 - 5s - loss: 3896.3926 - 5s/epoch - 2ms/step
Epoch 7/10
2540/2540 - 6s - loss: 3896.2993 - 6s/epoch - 2ms/step
Epoch 8/10
2540/2540 - 5s - loss: 3896.4221 - 5s/epoch - 2ms/step
Epoch 9/10
2540/2540 - 4s - loss: 3896.4817 - 4s/epoch - 2ms/step
Epoch 10/10
2540/2540 - 6s - loss: 3896.4258 - 6s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
6 Parameters: batch_size: 10 - epochs: 10
RMSE:  62.849
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_14 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)
```

```
 conv2d_15 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_15 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_7 (Flatten)         (None, 32)                0

 dense_21 (Dense)            (None, 120)               3960

 dense_22 (Dense)            (None, 84)                10164

 dense_23 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/50
2540/2540 - 5s - loss: 46617.5273 - 5s/epoch - 2ms/step
Epoch 2/50
2540/2540 - 4s - loss: 3889.6274 - 4s/epoch - 2ms/step
Epoch 3/50
2540/2540 - 5s - loss: 3889.7026 - 5s/epoch - 2ms/step
Epoch 4/50
2540/2540 - 4s - loss: 3889.5332 - 4s/epoch - 2ms/step
Epoch 5/50
2540/2540 - 4s - loss: 3889.3567 - 4s/epoch - 2ms/step
Epoch 6/50
2540/2540 - 5s - loss: 3889.5508 - 5s/epoch - 2ms/step
Epoch 7/50
2540/2540 - 4s - loss: 3889.6741 - 4s/epoch - 2ms/step
Epoch 8/50
2540/2540 - 4s - loss: 3889.4524 - 4s/epoch - 2ms/step
Epoch 9/50
2540/2540 - 5s - loss: 3889.2664 - 5s/epoch - 2ms/step
Epoch 10/50
2540/2540 - 4s - loss: 3889.5884 - 4s/epoch - 2ms/step
Epoch 11/50
2540/2540 - 4s - loss: 3888.6638 - 4s/epoch - 2ms/step
Epoch 12/50
2540/2540 - 5s - loss: 3889.6084 - 5s/epoch - 2ms/step
Epoch 13/50
2540/2540 - 4s - loss: 3889.2502 - 4s/epoch - 2ms/step
Epoch 14/50
2540/2540 - 4s - loss: 3889.6235 - 4s/epoch - 2ms/step
Epoch 15/50
2540/2540 - 5s - loss: 3889.0908 - 5s/epoch - 2ms/step
```

```
Epoch 16/50
2540/2540 - 4s - loss: 3889.5984 - 4s/epoch - 2ms/step
Epoch 17/50
2540/2540 - 4s - loss: 3889.6809 - 4s/epoch - 2ms/step
Epoch 18/50
2540/2540 - 6s - loss: 3889.3501 - 6s/epoch - 2ms/step
Epoch 19/50
2540/2540 - 4s - loss: 3889.4001 - 4s/epoch - 2ms/step
Epoch 20/50
2540/2540 - 4s - loss: 3889.5498 - 4s/epoch - 2ms/step
Epoch 21/50
2540/2540 - 6s - loss: 3889.6091 - 6s/epoch - 2ms/step
Epoch 22/50
2540/2540 - 4s - loss: 3889.3965 - 4s/epoch - 2ms/step
Epoch 23/50
2540/2540 - 4s - loss: 3889.5149 - 4s/epoch - 2ms/step
Epoch 24/50
2540/2540 - 5s - loss: 3889.3438 - 5s/epoch - 2ms/step
Epoch 25/50
2540/2540 - 4s - loss: 3889.7036 - 4s/epoch - 2ms/step
Epoch 26/50
2540/2540 - 4s - loss: 3889.5251 - 4s/epoch - 2ms/step
Epoch 27/50
2540/2540 - 5s - loss: 3889.6028 - 5s/epoch - 2ms/step
Epoch 28/50
2540/2540 - 4s - loss: 3889.6289 - 4s/epoch - 2ms/step
Epoch 29/50
2540/2540 - 4s - loss: 3889.1677 - 4s/epoch - 2ms/step
Epoch 30/50
2540/2540 - 5s - loss: 3889.3010 - 5s/epoch - 2ms/step
Epoch 31/50
2540/2540 - 4s - loss: 3889.5869 - 4s/epoch - 2ms/step
Epoch 32/50
2540/2540 - 4s - loss: 3889.5537 - 4s/epoch - 2ms/step
Epoch 33/50
2540/2540 - 5s - loss: 3889.5725 - 5s/epoch - 2ms/step
Epoch 34/50
2540/2540 - 4s - loss: 3889.5105 - 4s/epoch - 2ms/step
Epoch 35/50
2540/2540 - 5s - loss: 3889.6265 - 5s/epoch - 2ms/step
Epoch 36/50
2540/2540 - 5s - loss: 3889.4924 - 5s/epoch - 2ms/step
Epoch 37/50
2540/2540 - 5s - loss: 3889.6421 - 5s/epoch - 2ms/step
Epoch 38/50
2540/2540 - 5s - loss: 3889.2637 - 5s/epoch - 2ms/step
Epoch 39/50
2540/2540 - 4s - loss: 3889.3718 - 4s/epoch - 2ms/step
Epoch 40/50
2540/2540 - 4s - loss: 3889.2822 - 4s/epoch - 2ms/step
Epoch 41/50
2540/2540 - 5s - loss: 3889.1475 - 5s/epoch - 2ms/step
Epoch 42/50
2540/2540 - 4s - loss: 3889.4792 - 4s/epoch - 2ms/step
Epoch 43/50
```

```
2540/2540 - 4s - loss: 3889.5671 - 4s/epoch - 2ms/step
Epoch 44/50
2540/2540 - 5s - loss: 3889.5269 - 5s/epoch - 2ms/step
Epoch 45/50
2540/2540 - 4s - loss: 3889.4399 - 4s/epoch - 2ms/step
Epoch 46/50
2540/2540 - 4s - loss: 3889.3640 - 4s/epoch - 2ms/step
Epoch 47/50
2540/2540 - 5s - loss: 3889.4207 - 5s/epoch - 2ms/step
Epoch 48/50
2540/2540 - 4s - loss: 3889.5742 - 4s/epoch - 2ms/step
Epoch 49/50
2540/2540 - 4s - loss: 3889.5906 - 4s/epoch - 2ms/step
Epoch 50/50
2540/2540 - 5s - loss: 3889.5591 - 5s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
7 Parameters: batch_size: 10 - epochs: 50
RMSE:  62.794
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_16 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_17 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_17 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_8 (Flatten)         (None, 32)                0

 dense_24 (Dense)            (None, 120)               3960

 dense_25 (Dense)            (None, 84)                10164

 dense_26 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/100
2540/2540 - 5s - loss: 145643.6719 - 5s/epoch - 2ms/step
Epoch 2/100
2540/2540 - 4s - loss: 3896.0261 - 4s/epoch - 2ms/step
```

```
Epoch 3/100
2540/2540 - 5s - loss: 3896.0564 - 5s/epoch - 2ms/step
Epoch 4/100
2540/2540 - 4s - loss: 3896.0510 - 4s/epoch - 2ms/step
Epoch 5/100
2540/2540 - 4s - loss: 3896.1060 - 4s/epoch - 2ms/step
Epoch 6/100
2540/2540 - 6s - loss: 3896.0002 - 6s/epoch - 2ms/step
Epoch 7/100
2540/2540 - 4s - loss: 3896.0383 - 4s/epoch - 2ms/step
Epoch 8/100
2540/2540 - 5s - loss: 3896.0437 - 5s/epoch - 2ms/step
Epoch 9/100
2540/2540 - 6s - loss: 3895.9407 - 6s/epoch - 2ms/step
Epoch 10/100
2540/2540 - 4s - loss: 3896.1533 - 4s/epoch - 2ms/step
Epoch 11/100
2540/2540 - 5s - loss: 3896.1008 - 5s/epoch - 2ms/step
Epoch 12/100
2540/2540 - 4s - loss: 3895.5239 - 4s/epoch - 2ms/step
Epoch 13/100
2540/2540 - 4s - loss: 3896.0012 - 4s/epoch - 2ms/step
Epoch 14/100
2540/2540 - 5s - loss: 3895.9316 - 5s/epoch - 2ms/step
Epoch 15/100
2540/2540 - 4s - loss: 3896.0876 - 4s/epoch - 2ms/step
Epoch 16/100
2540/2540 - 4s - loss: 3896.0061 - 4s/epoch - 2ms/step
Epoch 17/100
2540/2540 - 5s - loss: 3895.9729 - 5s/epoch - 2ms/step
Epoch 18/100
2540/2540 - 4s - loss: 3896.0930 - 4s/epoch - 2ms/step
Epoch 19/100
2540/2540 - 4s - loss: 3895.7864 - 4s/epoch - 2ms/step
Epoch 20/100
2540/2540 - 5s - loss: 3896.0002 - 5s/epoch - 2ms/step
Epoch 21/100
2540/2540 - 4s - loss: 3895.9856 - 4s/epoch - 2ms/step
Epoch 22/100
2540/2540 - 4s - loss: 3896.0964 - 4s/epoch - 2ms/step
Epoch 23/100
2540/2540 - 5s - loss: 3895.7974 - 5s/epoch - 2ms/step
Epoch 24/100
2540/2540 - 4s - loss: 3895.9016 - 4s/epoch - 2ms/step
Epoch 25/100
2540/2540 - 4s - loss: 3895.8789 - 4s/epoch - 2ms/step
Epoch 26/100
2540/2540 - 5s - loss: 3895.8540 - 5s/epoch - 2ms/step
Epoch 27/100
2540/2540 - 4s - loss: 3896.0789 - 4s/epoch - 2ms/step
Epoch 28/100
2540/2540 - 4s - loss: 3895.7156 - 4s/epoch - 2ms/step
Epoch 29/100
2540/2540 - 5s - loss: 3895.9609 - 5s/epoch - 2ms/step
Epoch 30/100
```

```
2540/2540 - 4s - loss: 3895.8442 - 4s/epoch - 2ms/step
Epoch 31/100
2540/2540 - 4s - loss: 3896.1099 - 4s/epoch - 2ms/step
Epoch 32/100
2540/2540 - 5s - loss: 3895.8713 - 5s/epoch - 2ms/step
Epoch 33/100
2540/2540 - 4s - loss: 3895.9031 - 4s/epoch - 2ms/step
Epoch 34/100
2540/2540 - 4s - loss: 3896.0381 - 4s/epoch - 2ms/step
Epoch 35/100
2540/2540 - 6s - loss: 3895.8787 - 6s/epoch - 2ms/step
Epoch 36/100
2540/2540 - 4s - loss: 3895.8589 - 4s/epoch - 2ms/step
Epoch 37/100
2540/2540 - 4s - loss: 3895.9438 - 4s/epoch - 2ms/step
Epoch 38/100
2540/2540 - 5s - loss: 3895.9219 - 5s/epoch - 2ms/step
Epoch 39/100
2540/2540 - 4s - loss: 3896.0303 - 4s/epoch - 2ms/step
Epoch 40/100
2540/2540 - 4s - loss: 3896.0405 - 4s/epoch - 2ms/step
Epoch 41/100
2540/2540 - 6s - loss: 3896.0415 - 6s/epoch - 2ms/step
Epoch 42/100
2540/2540 - 4s - loss: 3896.0381 - 4s/epoch - 2ms/step
Epoch 43/100
2540/2540 - 4s - loss: 3895.9988 - 4s/epoch - 2ms/step
Epoch 44/100
2540/2540 - 5s - loss: 3895.9294 - 5s/epoch - 2ms/step
Epoch 45/100
2540/2540 - 4s - loss: 3895.9392 - 4s/epoch - 2ms/step
Epoch 46/100
2540/2540 - 4s - loss: 3895.9575 - 4s/epoch - 2ms/step
Epoch 47/100
2540/2540 - 5s - loss: 3896.0684 - 5s/epoch - 2ms/step
Epoch 48/100
2540/2540 - 4s - loss: 3895.9526 - 4s/epoch - 2ms/step
Epoch 49/100
2540/2540 - 4s - loss: 3896.0198 - 4s/epoch - 2ms/step
Epoch 50/100
2540/2540 - 5s - loss: 3895.9402 - 5s/epoch - 2ms/step
Epoch 51/100
2540/2540 - 4s - loss: 3896.0681 - 4s/epoch - 2ms/step
Epoch 52/100
2540/2540 - 4s - loss: 3895.9568 - 4s/epoch - 2ms/step
Epoch 53/100
2540/2540 - 5s - loss: 3895.9492 - 5s/epoch - 2ms/step
Epoch 54/100
2540/2540 - 4s - loss: 3895.8132 - 4s/epoch - 2ms/step
Epoch 55/100
2540/2540 - 4s - loss: 3895.9641 - 4s/epoch - 2ms/step
Epoch 56/100
2540/2540 - 5s - loss: 3896.0913 - 5s/epoch - 2ms/step
Epoch 57/100
2540/2540 - 4s - loss: 3896.0391 - 4s/epoch - 2ms/step
```

```
Epoch 58/100
2540/2540 - 4s - loss: 3896.0354 - 4s/epoch - 2ms/step
Epoch 59/100
2540/2540 - 5s - loss: 3895.7073 - 5s/epoch - 2ms/step
Epoch 60/100
2540/2540 - 4s - loss: 3895.8289 - 4s/epoch - 2ms/step
Epoch 61/100
2540/2540 - 4s - loss: 3896.0452 - 4s/epoch - 2ms/step
Epoch 62/100
2540/2540 - 5s - loss: 3895.8857 - 5s/epoch - 2ms/step
Epoch 63/100
2540/2540 - 4s - loss: 3895.8711 - 4s/epoch - 2ms/step
Epoch 64/100
2540/2540 - 4s - loss: 3895.8589 - 4s/epoch - 2ms/step
Epoch 65/100
2540/2540 - 5s - loss: 3895.9199 - 5s/epoch - 2ms/step
Epoch 66/100
2540/2540 - 4s - loss: 3896.0845 - 4s/epoch - 2ms/step
Epoch 67/100
2540/2540 - 4s - loss: 3895.8936 - 4s/epoch - 2ms/step
Epoch 68/100
2540/2540 - 5s - loss: 3896.0576 - 5s/epoch - 2ms/step
Epoch 69/100
2540/2540 - 4s - loss: 3896.0293 - 4s/epoch - 2ms/step
Epoch 70/100
2540/2540 - 4s - loss: 3895.9758 - 4s/epoch - 2ms/step
Epoch 71/100
2540/2540 - 5s - loss: 3895.9700 - 5s/epoch - 2ms/step
Epoch 72/100
2540/2540 - 4s - loss: 3896.0054 - 4s/epoch - 2ms/step
Epoch 73/100
2540/2540 - 5s - loss: 3895.8936 - 5s/epoch - 2ms/step
Epoch 74/100
2540/2540 - 6s - loss: 3895.9417 - 6s/epoch - 2ms/step
Epoch 75/100
2540/2540 - 5s - loss: 3895.2583 - 5s/epoch - 2ms/step
Epoch 76/100
2540/2540 - 5s - loss: 3896.0813 - 5s/epoch - 2ms/step
Epoch 77/100
2540/2540 - 4s - loss: 3895.8809 - 4s/epoch - 2ms/step
Epoch 78/100
2540/2540 - 4s - loss: 3895.9468 - 4s/epoch - 2ms/step
Epoch 79/100
2540/2540 - 5s - loss: 3895.9722 - 5s/epoch - 2ms/step
Epoch 80/100
2540/2540 - 4s - loss: 3895.8577 - 4s/epoch - 2ms/step
Epoch 81/100
2540/2540 - 4s - loss: 3896.0400 - 4s/epoch - 2ms/step
Epoch 82/100
2540/2540 - 5s - loss: 3895.9829 - 5s/epoch - 2ms/step
Epoch 83/100
2540/2540 - 4s - loss: 3896.0649 - 4s/epoch - 2ms/step
Epoch 84/100
2540/2540 - 4s - loss: 3896.0500 - 4s/epoch - 2ms/step
Epoch 85/100
```

```
2540/2540 - 5s - loss: 3895.7786 - 5s/epoch - 2ms/step
Epoch 86/100
2540/2540 - 4s - loss: 3895.8132 - 4s/epoch - 2ms/step
Epoch 87/100
2540/2540 - 4s - loss: 3896.0713 - 4s/epoch - 2ms/step
Epoch 88/100
2540/2540 - 5s - loss: 3896.0750 - 5s/epoch - 2ms/step
Epoch 89/100
2540/2540 - 4s - loss: 3895.9448 - 4s/epoch - 2ms/step
Epoch 90/100
2540/2540 - 4s - loss: 3895.9077 - 4s/epoch - 2ms/step
Epoch 91/100
2540/2540 - 5s - loss: 3896.0327 - 5s/epoch - 2ms/step
Epoch 92/100
2540/2540 - 4s - loss: 3895.9526 - 4s/epoch - 2ms/step
Epoch 93/100
2540/2540 - 4s - loss: 3895.8296 - 4s/epoch - 2ms/step
Epoch 94/100
2540/2540 - 5s - loss: 3895.9141 - 5s/epoch - 2ms/step
Epoch 95/100
2540/2540 - 4s - loss: 3895.9985 - 4s/epoch - 2ms/step
Epoch 96/100
2540/2540 - 4s - loss: 3895.4148 - 4s/epoch - 2ms/step
Epoch 97/100
2540/2540 - 5s - loss: 3896.1628 - 5s/epoch - 2ms/step
Epoch 98/100
2540/2540 - 4s - loss: 3895.8257 - 4s/epoch - 2ms/step
Epoch 99/100
2540/2540 - 4s - loss: 3895.7849 - 4s/epoch - 2ms/step
Epoch 100/100
2540/2540 - 5s - loss: 3896.0344 - 5s/epoch - 2ms/step
341/341 [==============================] - 1s 3ms/step
8 Parameters: batch_size: 10 - epochs: 100
RMSE:  62.845
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_18 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_19 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_19 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_9 (Flatten)         (None, 32)                0

 dense_27 (Dense)            (None, 120)               3960

 dense_28 (Dense)            (None, 84)                10164

 dense_29 (Dense)            (None, 1)                 85
```

```
=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/5
1693/1693 - 4s - loss: 5345.5879 - 4s/epoch - 2ms/step
Epoch 2/5
1693/1693 - 3s - loss: 3899.6829 - 3s/epoch - 2ms/step
Epoch 3/5
1693/1693 - 3s - loss: 3899.6001 - 3s/epoch - 2ms/step
Epoch 4/5
1693/1693 - 4s - loss: 3899.0537 - 4s/epoch - 2ms/step
Epoch 5/5
1693/1693 - 3s - loss: 3899.7139 - 3s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
9 Parameters: batch_size: 15 - epochs: 5
RMSE:  62.875
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_20 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_21 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_21 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_10 (Flatten)        (None, 32)                0

 dense_30 (Dense)            (None, 120)               3960

 dense_31 (Dense)            (None, 84)                10164

 dense_32 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
```

```
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/10
1693/1693 - 5s - loss: 14662.0264 - 5s/epoch - 3ms/step
Epoch 2/10
1693/1693 - 3s - loss: 3992.1309 - 3s/epoch - 2ms/step
Epoch 3/10
1693/1693 - 3s - loss: 3889.6096 - 3s/epoch - 2ms/step
Epoch 4/10
1693/1693 - 3s - loss: 3888.2231 - 3s/epoch - 2ms/step
Epoch 5/10
1693/1693 - 4s - loss: 3888.1038 - 4s/epoch - 2ms/step
Epoch 6/10
1693/1693 - 3s - loss: 3888.0391 - 3s/epoch - 2ms/step
Epoch 7/10
1693/1693 - 3s - loss: 3888.0015 - 3s/epoch - 2ms/step
Epoch 8/10
1693/1693 - 3s - loss: 3887.9753 - 3s/epoch - 2ms/step
Epoch 9/10
1693/1693 - 4s - loss: 3887.9575 - 4s/epoch - 2ms/step
Epoch 10/10
1693/1693 - 3s - loss: 3887.9272 - 3s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
10 Parameters: batch_size: 15 - epochs: 10
RMSE:  62.783
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_22 (Conv2D)          (None, 11, 1, 6)          36        
                                                                 
 max_pooling2d_22 (MaxPoolin  (None, 5, 1, 6)          0         
 g2D)                                                            
                                                                 
 conv2d_23 (Conv2D)          (None, 5, 1, 16)          496       
                                                                 
 max_pooling2d_23 (MaxPoolin  (None, 2, 1, 16)         0         
 g2D)                                                            
                                                                 
 flatten_11 (Flatten)        (None, 32)                0         
                                                                 
 dense_33 (Dense)            (None, 120)               3960      
                                                                 
 dense_34 (Dense)            (None, 84)                10164     
                                                                 
 dense_35 (Dense)            (None, 1)                 85        
                                                                 
=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
```

```
<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/50
1693/1693 - 4s - loss: 4643.9380 - 4s/epoch - 2ms/step
Epoch 2/50
1693/1693 - 3s - loss: 3889.8330 - 3s/epoch - 2ms/step
Epoch 3/50
1693/1693 - 4s - loss: 3889.5493 - 4s/epoch - 2ms/step
Epoch 4/50
1693/1693 - 3s - loss: 3889.5886 - 3s/epoch - 2ms/step
Epoch 5/50
1693/1693 - 3s - loss: 3889.6299 - 3s/epoch - 2ms/step
Epoch 6/50
1693/1693 - 3s - loss: 3889.4011 - 3s/epoch - 2ms/step
Epoch 7/50
1693/1693 - 4s - loss: 3889.5596 - 4s/epoch - 2ms/step
Epoch 8/50
1693/1693 - 3s - loss: 3889.0974 - 3s/epoch - 2ms/step
Epoch 9/50
1693/1693 - 3s - loss: 3889.7083 - 3s/epoch - 2ms/step
Epoch 10/50
1693/1693 - 3s - loss: 3889.8933 - 3s/epoch - 2ms/step
Epoch 11/50
1693/1693 - 4s - loss: 3889.7485 - 4s/epoch - 2ms/step
Epoch 12/50
1693/1693 - 3s - loss: 3889.0505 - 3s/epoch - 2ms/step
Epoch 13/50
1693/1693 - 3s - loss: 3889.5159 - 3s/epoch - 2ms/step
Epoch 14/50
1693/1693 - 3s - loss: 3889.5618 - 3s/epoch - 2ms/step
Epoch 15/50
1693/1693 - 3s - loss: 3889.7893 - 3s/epoch - 2ms/step
Epoch 16/50
1693/1693 - 4s - loss: 3889.6589 - 4s/epoch - 2ms/step
Epoch 17/50
1693/1693 - 3s - loss: 3889.5186 - 3s/epoch - 2ms/step
Epoch 18/50
1693/1693 - 3s - loss: 3889.8633 - 3s/epoch - 2ms/step
Epoch 19/50
1693/1693 - 3s - loss: 3889.5723 - 3s/epoch - 2ms/step
Epoch 20/50
1693/1693 - 4s - loss: 3889.7993 - 4s/epoch - 2ms/step
Epoch 21/50
1693/1693 - 3s - loss: 3889.6406 - 3s/epoch - 2ms/step
Epoch 22/50
1693/1693 - 3s - loss: 3889.6682 - 3s/epoch - 2ms/step
Epoch 23/50
1693/1693 - 3s - loss: 3889.8965 - 3s/epoch - 2ms/step
Epoch 24/50
1693/1693 - 4s - loss: 3889.2080 - 4s/epoch - 3ms/step
Epoch 25/50
1693/1693 - 3s - loss: 3889.6519 - 3s/epoch - 2ms/step
```

```
Epoch 26/50
1693/1693 - 3s - loss: 3889.7908 - 3s/epoch - 2ms/step
Epoch 27/50
1693/1693 - 3s - loss: 3889.8606 - 3s/epoch - 2ms/step
Epoch 28/50
1693/1693 - 4s - loss: 3889.8638 - 4s/epoch - 3ms/step
Epoch 29/50
1693/1693 - 3s - loss: 3889.8230 - 3s/epoch - 2ms/step
Epoch 30/50
1693/1693 - 3s - loss: 3889.7893 - 3s/epoch - 2ms/step
Epoch 31/50
1693/1693 - 4s - loss: 3889.6519 - 4s/epoch - 2ms/step
Epoch 32/50
1693/1693 - 4s - loss: 3889.7202 - 4s/epoch - 2ms/step
Epoch 33/50
1693/1693 - 3s - loss: 3889.8613 - 3s/epoch - 2ms/step
Epoch 34/50
1693/1693 - 3s - loss: 3889.5647 - 3s/epoch - 2ms/step
Epoch 35/50
1693/1693 - 3s - loss: 3889.1475 - 3s/epoch - 2ms/step
Epoch 36/50
1693/1693 - 4s - loss: 3889.4636 - 4s/epoch - 2ms/step
Epoch 37/50
1693/1693 - 3s - loss: 3889.5757 - 3s/epoch - 2ms/step
Epoch 38/50
1693/1693 - 3s - loss: 3889.5798 - 3s/epoch - 2ms/step
Epoch 39/50
1693/1693 - 3s - loss: 3889.6226 - 3s/epoch - 2ms/step
Epoch 40/50
1693/1693 - 4s - loss: 3889.7827 - 4s/epoch - 2ms/step
Epoch 41/50
1693/1693 - 3s - loss: 3889.8357 - 3s/epoch - 2ms/step
Epoch 42/50
1693/1693 - 3s - loss: 3889.5793 - 3s/epoch - 2ms/step
Epoch 43/50
1693/1693 - 3s - loss: 3889.2717 - 3s/epoch - 2ms/step
Epoch 44/50
1693/1693 - 4s - loss: 3889.5066 - 4s/epoch - 2ms/step
Epoch 45/50
1693/1693 - 3s - loss: 3889.3035 - 3s/epoch - 2ms/step
Epoch 46/50
1693/1693 - 3s - loss: 3889.7073 - 3s/epoch - 2ms/step
Epoch 47/50
1693/1693 - 3s - loss: 3889.7307 - 3s/epoch - 2ms/step
Epoch 48/50
1693/1693 - 4s - loss: 3889.5840 - 4s/epoch - 2ms/step
Epoch 49/50
1693/1693 - 3s - loss: 3889.5422 - 3s/epoch - 2ms/step
Epoch 50/50
1693/1693 - 3s - loss: 3889.5596 - 3s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
11 Parameters: batch_size: 15 - epochs: 50
RMSE:  62.793
Model: "sequential_12"
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_24 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_25 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_25 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_12 (Flatten)        (None, 32)                0

 dense_36 (Dense)            (None, 120)               3960

 dense_37 (Dense)            (None, 84)                10164

 dense_38 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/100
1693/1693 - 5s - loss: 12343.5762 - 5s/epoch - 3ms/step
Epoch 2/100
1693/1693 - 3s - loss: 3888.4468 - 3s/epoch - 2ms/step
Epoch 3/100
1693/1693 - 3s - loss: 3888.3840 - 3s/epoch - 2ms/step
Epoch 4/100
1693/1693 - 3s - loss: 3888.0554 - 3s/epoch - 2ms/step
Epoch 5/100
1693/1693 - 4s - loss: 3888.6223 - 4s/epoch - 3ms/step
Epoch 6/100
1693/1693 - 3s - loss: 3888.2961 - 3s/epoch - 2ms/step
Epoch 7/100
1693/1693 - 3s - loss: 3888.2375 - 3s/epoch - 2ms/step
Epoch 8/100
1693/1693 - 3s - loss: 3888.3533 - 3s/epoch - 2ms/step
Epoch 9/100
1693/1693 - 4s - loss: 3888.3093 - 4s/epoch - 3ms/step
Epoch 10/100
1693/1693 - 3s - loss: 3888.4817 - 3s/epoch - 2ms/step
Epoch 11/100
1693/1693 - 3s - loss: 3888.4575 - 3s/epoch - 2ms/step
Epoch 12/100
1693/1693 - 3s - loss: 3888.4304 - 3s/epoch - 2ms/step
```

```
Epoch 13/100
1693/1693 - 4s - loss: 3888.3711 - 4s/epoch - 2ms/step
Epoch 14/100
1693/1693 - 3s - loss: 3888.3091 - 3s/epoch - 2ms/step
Epoch 15/100
1693/1693 - 3s - loss: 3888.3696 - 3s/epoch - 2ms/step
Epoch 16/100
1693/1693 - 3s - loss: 3888.2617 - 3s/epoch - 2ms/step
Epoch 17/100
1693/1693 - 3s - loss: 3888.2971 - 3s/epoch - 2ms/step
Epoch 18/100
1693/1693 - 4s - loss: 3888.4231 - 4s/epoch - 2ms/step
Epoch 19/100
1693/1693 - 3s - loss: 3888.2241 - 3s/epoch - 2ms/step
Epoch 20/100
1693/1693 - 3s - loss: 3888.4304 - 3s/epoch - 2ms/step
Epoch 21/100
1693/1693 - 3s - loss: 3888.3589 - 3s/epoch - 2ms/step
Epoch 22/100
1693/1693 - 4s - loss: 3888.2983 - 4s/epoch - 2ms/step
Epoch 23/100
1693/1693 - 3s - loss: 3888.3596 - 3s/epoch - 2ms/step
Epoch 24/100
1693/1693 - 3s - loss: 3888.4780 - 3s/epoch - 2ms/step
Epoch 25/100
1693/1693 - 3s - loss: 3888.2041 - 3s/epoch - 2ms/step
Epoch 26/100
1693/1693 - 4s - loss: 3888.2715 - 4s/epoch - 2ms/step
Epoch 27/100
1693/1693 - 3s - loss: 3888.3306 - 3s/epoch - 2ms/step
Epoch 28/100
1693/1693 - 3s - loss: 3888.3999 - 3s/epoch - 2ms/step
Epoch 29/100
1693/1693 - 3s - loss: 3888.1624 - 3s/epoch - 2ms/step
Epoch 30/100
1693/1693 - 4s - loss: 3888.3662 - 4s/epoch - 2ms/step
Epoch 31/100
1693/1693 - 3s - loss: 3888.1499 - 3s/epoch - 2ms/step
Epoch 32/100
1693/1693 - 3s - loss: 3888.4258 - 3s/epoch - 2ms/step
Epoch 33/100
1693/1693 - 3s - loss: 3888.5137 - 3s/epoch - 2ms/step
Epoch 34/100
1693/1693 - 3s - loss: 3888.4675 - 3s/epoch - 2ms/step
Epoch 35/100
1693/1693 - 4s - loss: 3888.2444 - 4s/epoch - 2ms/step
Epoch 36/100
1693/1693 - 3s - loss: 3887.9231 - 3s/epoch - 2ms/step
Epoch 37/100
1693/1693 - 3s - loss: 3888.2170 - 3s/epoch - 2ms/step
Epoch 38/100
1693/1693 - 3s - loss: 3888.3501 - 3s/epoch - 2ms/step
Epoch 39/100
1693/1693 - 4s - loss: 3888.4673 - 4s/epoch - 2ms/step
Epoch 40/100
```

```
1693/1693 - 3s - loss: 3888.4221 - 3s/epoch - 2ms/step
Epoch 41/100
1693/1693 - 3s - loss: 3888.4084 - 3s/epoch - 2ms/step
Epoch 42/100
1693/1693 - 3s - loss: 3888.4661 - 3s/epoch - 2ms/step
Epoch 43/100
1693/1693 - 4s - loss: 3888.3564 - 4s/epoch - 2ms/step
Epoch 44/100
1693/1693 - 3s - loss: 3888.1409 - 3s/epoch - 2ms/step
Epoch 45/100
1693/1693 - 3s - loss: 3888.4160 - 3s/epoch - 2ms/step
Epoch 46/100
1693/1693 - 3s - loss: 3888.4163 - 3s/epoch - 2ms/step
Epoch 47/100
1693/1693 - 3s - loss: 3888.1660 - 3s/epoch - 2ms/step
Epoch 48/100
1693/1693 - 3s - loss: 3888.4253 - 3s/epoch - 2ms/step
Epoch 49/100
1693/1693 - 3s - loss: 3888.1604 - 3s/epoch - 2ms/step
Epoch 50/100
1693/1693 - 3s - loss: 3888.3687 - 3s/epoch - 2ms/step
Epoch 51/100
1693/1693 - 3s - loss: 3888.1135 - 3s/epoch - 2ms/step
Epoch 52/100
1693/1693 - 4s - loss: 3888.2649 - 4s/epoch - 2ms/step
Epoch 53/100
1693/1693 - 3s - loss: 3888.0767 - 3s/epoch - 2ms/step
Epoch 54/100
1693/1693 - 3s - loss: 3888.4751 - 3s/epoch - 2ms/step
Epoch 55/100
1693/1693 - 3s - loss: 3888.3188 - 3s/epoch - 2ms/step
Epoch 56/100
1693/1693 - 4s - loss: 3888.1240 - 4s/epoch - 3ms/step
Epoch 57/100
1693/1693 - 3s - loss: 3888.5391 - 3s/epoch - 2ms/step
Epoch 58/100
1693/1693 - 3s - loss: 3888.2795 - 3s/epoch - 2ms/step
Epoch 59/100
1693/1693 - 3s - loss: 3888.3621 - 3s/epoch - 2ms/step
Epoch 60/100
1693/1693 - 4s - loss: 3888.3450 - 4s/epoch - 3ms/step
Epoch 61/100
1693/1693 - 3s - loss: 3888.0293 - 3s/epoch - 2ms/step
Epoch 62/100
1693/1693 - 3s - loss: 3888.0652 - 3s/epoch - 2ms/step
Epoch 63/100
1693/1693 - 3s - loss: 3888.4700 - 3s/epoch - 2ms/step
Epoch 64/100
1693/1693 - 4s - loss: 3888.4382 - 4s/epoch - 2ms/step
Epoch 65/100
1693/1693 - 3s - loss: 3888.4219 - 3s/epoch - 2ms/step
Epoch 66/100
1693/1693 - 3s - loss: 3888.3782 - 3s/epoch - 2ms/step
Epoch 67/100
1693/1693 - 3s - loss: 3888.4287 - 3s/epoch - 2ms/step
```

```
Epoch 68/100
1693/1693 - 4s - loss: 3888.1560 - 4s/epoch - 2ms/step
Epoch 69/100
1693/1693 - 3s - loss: 3888.3362 - 3s/epoch - 2ms/step
Epoch 70/100
1693/1693 - 3s - loss: 3888.1650 - 3s/epoch - 2ms/step
Epoch 71/100
1693/1693 - 3s - loss: 3888.4731 - 3s/epoch - 2ms/step
Epoch 72/100
1693/1693 - 4s - loss: 3888.1194 - 4s/epoch - 2ms/step
Epoch 73/100
1693/1693 - 4s - loss: 3888.5264 - 4s/epoch - 2ms/step
Epoch 74/100
1693/1693 - 3s - loss: 3888.3271 - 3s/epoch - 2ms/step
Epoch 75/100
1693/1693 - 3s - loss: 3888.3228 - 3s/epoch - 2ms/step
Epoch 76/100
1693/1693 - 3s - loss: 3888.4709 - 3s/epoch - 2ms/step
Epoch 77/100
1693/1693 - 4s - loss: 3888.3972 - 4s/epoch - 2ms/step
Epoch 78/100
1693/1693 - 3s - loss: 3888.2778 - 3s/epoch - 2ms/step
Epoch 79/100
1693/1693 - 3s - loss: 3888.3755 - 3s/epoch - 2ms/step
Epoch 80/100
1693/1693 - 3s - loss: 3888.2324 - 3s/epoch - 2ms/step
Epoch 81/100
1693/1693 - 4s - loss: 3888.2622 - 4s/epoch - 2ms/step
Epoch 82/100
1693/1693 - 3s - loss: 3888.1414 - 3s/epoch - 2ms/step
Epoch 83/100
1693/1693 - 3s - loss: 3888.4731 - 3s/epoch - 2ms/step
Epoch 84/100
1693/1693 - 3s - loss: 3888.4043 - 3s/epoch - 2ms/step
Epoch 85/100
1693/1693 - 4s - loss: 3888.3315 - 4s/epoch - 3ms/step
Epoch 86/100
1693/1693 - 3s - loss: 3888.3774 - 3s/epoch - 2ms/step
Epoch 87/100
1693/1693 - 3s - loss: 3888.1343 - 3s/epoch - 2ms/step
Epoch 88/100
1693/1693 - 3s - loss: 3888.3884 - 3s/epoch - 2ms/step
Epoch 89/100
1693/1693 - 4s - loss: 3888.4685 - 4s/epoch - 2ms/step
Epoch 90/100
1693/1693 - 3s - loss: 3888.3457 - 3s/epoch - 2ms/step
Epoch 91/100
1693/1693 - 3s - loss: 3888.4524 - 3s/epoch - 2ms/step
Epoch 92/100
1693/1693 - 3s - loss: 3888.2920 - 3s/epoch - 2ms/step
Epoch 93/100
1693/1693 - 4s - loss: 3888.3916 - 4s/epoch - 2ms/step
Epoch 94/100
1693/1693 - 3s - loss: 3888.2632 - 3s/epoch - 2ms/step
Epoch 95/100
```

```
1693/1693 - 3s - loss: 3888.2788 - 3s/epoch - 2ms/step
Epoch 96/100
1693/1693 - 3s - loss: 3888.4751 - 3s/epoch - 2ms/step
Epoch 97/100
1693/1693 - 3s - loss: 3888.2939 - 3s/epoch - 2ms/step
Epoch 98/100
1693/1693 - 4s - loss: 3888.3176 - 4s/epoch - 2ms/step
Epoch 99/100
1693/1693 - 3s - loss: 3888.1582 - 3s/epoch - 2ms/step
Epoch 100/100
1693/1693 - 3s - loss: 3888.3274 - 3s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
12 Parameters: batch_size: 15 - epochs: 100
RMSE:  62.784
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_26 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_27 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_27 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_13 (Flatten)        (None, 32)                0

 dense_39 (Dense)            (None, 120)               3960

 dense_40 (Dense)            (None, 84)                10164

 dense_41 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/5
1270/1270 - 4s - loss: 13113.1260 - 4s/epoch - 3ms/step
Epoch 2/5
1270/1270 - 2s - loss: 3923.1646 - 2s/epoch - 2ms/step
Epoch 3/5
1270/1270 - 2s - loss: 3889.4280 - 2s/epoch - 2ms/step
Epoch 4/5
1270/1270 - 2s - loss: 3889.0659 - 2s/epoch - 2ms/step
```

```
Epoch 5/5
1270/1270 - 2s - loss: 3889.0522 - 2s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
13 Parameters: batch_size: 20 - epochs: 5
RMSE:  62.792
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_28 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_29 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_29 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_14 (Flatten)        (None, 32)                0

 dense_42 (Dense)            (None, 120)               3960

 dense_43 (Dense)            (None, 84)                10164

 dense_44 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/10
1270/1270 - 4s - loss: 554592.9375 - 4s/epoch - 3ms/step
Epoch 2/10
1270/1270 - 2s - loss: 3911.9187 - 2s/epoch - 2ms/step
Epoch 3/10
1270/1270 - 2s - loss: 3890.4695 - 2s/epoch - 2ms/step
Epoch 4/10
1270/1270 - 2s - loss: 3890.4436 - 2s/epoch - 2ms/step
Epoch 5/10
1270/1270 - 2s - loss: 3890.4468 - 2s/epoch - 2ms/step
Epoch 6/10
1270/1270 - 3s - loss: 3890.4517 - 3s/epoch - 3ms/step
Epoch 7/10
1270/1270 - 3s - loss: 3890.3696 - 3s/epoch - 2ms/step
Epoch 8/10
1270/1270 - 2s - loss: 3890.4260 - 2s/epoch - 2ms/step
Epoch 9/10
```

```
1270/1270 - 2s - loss: 3890.2534 - 2s/epoch - 2ms/step
Epoch 10/10
1270/1270 - 2s - loss: 3890.5159 - 2s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
14 Parameters: batch_size: 20 - epochs: 10
RMSE:  62.803
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_30 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_30 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_31 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_31 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_15 (Flatten)        (None, 32)                0

 dense_45 (Dense)            (None, 120)               3960

 dense_46 (Dense)            (None, 84)                10164

 dense_47 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/50
1270/1270 - 4s - loss: 76028.3281 - 4s/epoch - 4ms/step
Epoch 2/50
1270/1270 - 2s - loss: 3895.7700 - 2s/epoch - 2ms/step
Epoch 3/50
1270/1270 - 2s - loss: 3895.4705 - 2s/epoch - 2ms/step
Epoch 4/50
1270/1270 - 2s - loss: 3895.5986 - 2s/epoch - 2ms/step
Epoch 5/50
1270/1270 - 2s - loss: 3895.5969 - 2s/epoch - 2ms/step
Epoch 6/50
1270/1270 - 3s - loss: 3895.4451 - 3s/epoch - 3ms/step
Epoch 7/50
1270/1270 - 2s - loss: 3895.5857 - 2s/epoch - 2ms/step
Epoch 8/50
1270/1270 - 2s - loss: 3895.6128 - 2s/epoch - 2ms/step
```

```
Epoch 9/50
1270/1270 - 2s - loss: 3895.3806 - 2s/epoch - 2ms/step
Epoch 10/50
1270/1270 - 2s - loss: 3895.4915 - 2s/epoch - 2ms/step
Epoch 11/50
1270/1270 - 3s - loss: 3895.5713 - 3s/epoch - 2ms/step
Epoch 12/50
1270/1270 - 3s - loss: 3895.4851 - 3s/epoch - 2ms/step
Epoch 13/50
1270/1270 - 2s - loss: 3895.4004 - 2s/epoch - 2ms/step
Epoch 14/50
1270/1270 - 2s - loss: 3895.5225 - 2s/epoch - 2ms/step
Epoch 15/50
1270/1270 - 2s - loss: 3895.6570 - 2s/epoch - 2ms/step
Epoch 16/50
1270/1270 - 2s - loss: 3895.6899 - 2s/epoch - 2ms/step
Epoch 17/50
1270/1270 - 4s - loss: 3895.4175 - 4s/epoch - 3ms/step
Epoch 18/50
1270/1270 - 2s - loss: 3895.6348 - 2s/epoch - 2ms/step
Epoch 19/50
1270/1270 - 2s - loss: 3895.5947 - 2s/epoch - 2ms/step
Epoch 20/50
1270/1270 - 2s - loss: 3895.6431 - 2s/epoch - 2ms/step
Epoch 21/50
1270/1270 - 2s - loss: 3895.4546 - 2s/epoch - 2ms/step
Epoch 22/50
1270/1270 - 3s - loss: 3895.1055 - 3s/epoch - 2ms/step
Epoch 23/50
1270/1270 - 3s - loss: 3895.5144 - 3s/epoch - 2ms/step
Epoch 24/50
1270/1270 - 2s - loss: 3895.3906 - 2s/epoch - 2ms/step
Epoch 25/50
1270/1270 - 2s - loss: 3895.5081 - 2s/epoch - 2ms/step
Epoch 26/50
1270/1270 - 2s - loss: 3895.6777 - 2s/epoch - 2ms/step
Epoch 27/50
1270/1270 - 2s - loss: 3895.5986 - 2s/epoch - 2ms/step
Epoch 28/50
1270/1270 - 4s - loss: 3895.6104 - 4s/epoch - 3ms/step
Epoch 29/50
1270/1270 - 2s - loss: 3895.5725 - 2s/epoch - 2ms/step
Epoch 30/50
1270/1270 - 2s - loss: 3895.5293 - 2s/epoch - 2ms/step
Epoch 31/50
1270/1270 - 2s - loss: 3895.4656 - 2s/epoch - 2ms/step
Epoch 32/50
1270/1270 - 2s - loss: 3895.6343 - 2s/epoch - 2ms/step
Epoch 33/50
1270/1270 - 3s - loss: 3895.5898 - 3s/epoch - 2ms/step
Epoch 34/50
1270/1270 - 3s - loss: 3895.6045 - 3s/epoch - 2ms/step
Epoch 35/50
1270/1270 - 2s - loss: 3895.4219 - 2s/epoch - 2ms/step
Epoch 36/50
```

```
1270/1270 - 2s - loss: 3895.4785 - 2s/epoch - 2ms/step
Epoch 37/50
1270/1270 - 2s - loss: 3895.5642 - 2s/epoch - 2ms/step
Epoch 38/50
1270/1270 - 2s - loss: 3895.5525 - 2s/epoch - 2ms/step
Epoch 39/50
1270/1270 - 4s - loss: 3895.2529 - 4s/epoch - 3ms/step
Epoch 40/50
1270/1270 - 2s - loss: 3895.1641 - 2s/epoch - 2ms/step
Epoch 41/50
1270/1270 - 2s - loss: 3895.7715 - 2s/epoch - 2ms/step
Epoch 42/50
1270/1270 - 2s - loss: 3895.6023 - 2s/epoch - 2ms/step
Epoch 43/50
1270/1270 - 2s - loss: 3895.5435 - 2s/epoch - 2ms/step
Epoch 44/50
1270/1270 - 4s - loss: 3895.6289 - 4s/epoch - 3ms/step
Epoch 45/50
1270/1270 - 3s - loss: 3895.5059 - 3s/epoch - 2ms/step
Epoch 46/50
1270/1270 - 2s - loss: 3895.5229 - 2s/epoch - 2ms/step
Epoch 47/50
1270/1270 - 2s - loss: 3895.5945 - 2s/epoch - 2ms/step
Epoch 48/50
1270/1270 - 2s - loss: 3895.5676 - 2s/epoch - 2ms/step
Epoch 49/50
1270/1270 - 3s - loss: 3895.1787 - 3s/epoch - 2ms/step
Epoch 50/50
1270/1270 - 3s - loss: 3895.8359 - 3s/epoch - 2ms/step
341/341 [==============================] - 1s 2ms/step
15 Parameters: batch_size: 20 - epochs: 50
RMSE:  62.841
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_32 (Conv2D)          (None, 11, 1, 6)          36

 max_pooling2d_32 (MaxPoolin  (None, 5, 1, 6)          0
 g2D)

 conv2d_33 (Conv2D)          (None, 5, 1, 16)          496

 max_pooling2d_33 (MaxPoolin  (None, 2, 1, 16)         0
 g2D)

 flatten_16 (Flatten)        (None, 32)                0

 dense_48 (Dense)            (None, 120)               3960

 dense_49 (Dense)            (None, 84)                10164

 dense_50 (Dense)            (None, 1)                 85

=================================================================
```

```
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________

<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumber, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],

Epoch 1/100
1270/1270 - 3s - loss: 40807.1133 - 3s/epoch - 2ms/step
Epoch 2/100
1270/1270 - 2s - loss: 3909.4539 - 2s/epoch - 2ms/step
Epoch 3/100
1270/1270 - 2s - loss: 3899.3032 - 2s/epoch - 2ms/step
Epoch 4/100
1270/1270 - 4s - loss: 3899.0256 - 4s/epoch - 3ms/step
Epoch 5/100
1270/1270 - 2s - loss: 3899.0276 - 2s/epoch - 2ms/step
Epoch 6/100
1270/1270 - 2s - loss: 3899.0061 - 2s/epoch - 2ms/step
Epoch 7/100
1270/1270 - 2s - loss: 3898.9802 - 2s/epoch - 2ms/step
Epoch 8/100
1270/1270 - 2s - loss: 3898.9858 - 2s/epoch - 2ms/step
Epoch 9/100
1270/1270 - 3s - loss: 3898.9021 - 3s/epoch - 2ms/step
Epoch 10/100
1270/1270 - 3s - loss: 3899.0210 - 3s/epoch - 2ms/step
Epoch 11/100
1270/1270 - 2s - loss: 3898.9617 - 2s/epoch - 2ms/step
Epoch 12/100
1270/1270 - 2s - loss: 3898.9971 - 2s/epoch - 2ms/step
Epoch 13/100
1270/1270 - 2s - loss: 3898.9507 - 2s/epoch - 2ms/step
Epoch 14/100
1270/1270 - 2s - loss: 3899.0244 - 2s/epoch - 2ms/step
Epoch 15/100
1270/1270 - 4s - loss: 3898.9907 - 4s/epoch - 3ms/step
Epoch 16/100
1270/1270 - 2s - loss: 3898.9897 - 2s/epoch - 2ms/step
Epoch 17/100
1270/1270 - 2s - loss: 3898.9712 - 2s/epoch - 2ms/step
Epoch 18/100
1270/1270 - 2s - loss: 3898.7976 - 2s/epoch - 2ms/step
Epoch 19/100
1270/1270 - 2s - loss: 3898.9749 - 2s/epoch - 2ms/step
Epoch 20/100
1270/1270 - 3s - loss: 3898.9087 - 3s/epoch - 2ms/step
Epoch 21/100
1270/1270 - 3s - loss: 3898.9250 - 3s/epoch - 2ms/step
Epoch 22/100
1270/1270 - 2s - loss: 3898.9575 - 2s/epoch - 2ms/step
Epoch 23/100
```

```
1270/1270 - 2s - loss: 3898.9153 - 2s/epoch - 2ms/step
Epoch 24/100
1270/1270 - 2s - loss: 3898.9272 - 2s/epoch - 2ms/step
Epoch 25/100
1270/1270 - 2s - loss: 3898.8779 - 2s/epoch - 2ms/step
Epoch 26/100
1270/1270 - 3s - loss: 3898.9246 - 3s/epoch - 3ms/step
Epoch 27/100
1270/1270 - 2s - loss: 3898.9512 - 2s/epoch - 2ms/step
Epoch 28/100
1270/1270 - 2s - loss: 3898.9702 - 2s/epoch - 2ms/step
Epoch 29/100
1270/1270 - 2s - loss: 3898.9497 - 2s/epoch - 2ms/step
Epoch 30/100
1270/1270 - 2s - loss: 3898.9680 - 2s/epoch - 2ms/step
Epoch 31/100
1270/1270 - 3s - loss: 3898.8735 - 3s/epoch - 2ms/step
Epoch 32/100
1270/1270 - 3s - loss: 3898.9368 - 3s/epoch - 2ms/step
Epoch 33/100
1270/1270 - 2s - loss: 3898.9636 - 2s/epoch - 2ms/step
Epoch 34/100
1270/1270 - 2s - loss: 3898.9182 - 2s/epoch - 2ms/step
Epoch 35/100
1270/1270 - 2s - loss: 3898.9609 - 2s/epoch - 2ms/step
Epoch 36/100
1270/1270 - 2s - loss: 3898.8838 - 2s/epoch - 2ms/step
Epoch 37/100
1270/1270 - 4s - loss: 3898.7612 - 4s/epoch - 3ms/step
Epoch 38/100
1270/1270 - 2s - loss: 3898.9131 - 2s/epoch - 2ms/step
Epoch 39/100
1270/1270 - 2s - loss: 3898.7922 - 2s/epoch - 2ms/step
Epoch 40/100
1270/1270 - 2s - loss: 3899.0288 - 2s/epoch - 2ms/step
Epoch 41/100
1270/1270 - 2s - loss: 3898.9771 - 2s/epoch - 2ms/step
Epoch 42/100
1270/1270 - 3s - loss: 3898.7610 - 3s/epoch - 3ms/step
Epoch 43/100
1270/1270 - 3s - loss: 3898.8904 - 3s/epoch - 2ms/step
Epoch 44/100
1270/1270 - 2s - loss: 3898.8699 - 2s/epoch - 2ms/step
Epoch 45/100
1270/1270 - 2s - loss: 3899.0017 - 2s/epoch - 2ms/step
Epoch 46/100
1270/1270 - 2s - loss: 3898.7717 - 2s/epoch - 2ms/step
Epoch 47/100
1270/1270 - 3s - loss: 3898.8345 - 3s/epoch - 2ms/step
Epoch 48/100
1270/1270 - 3s - loss: 3899.0144 - 3s/epoch - 2ms/step
Epoch 49/100
1270/1270 - 2s - loss: 3898.9067 - 2s/epoch - 2ms/step
Epoch 50/100
1270/1270 - 2s - loss: 3898.8940 - 2s/epoch - 2ms/step
```

```
Epoch 51/100
1270/1270 - 2s - loss: 3898.9834 - 2s/epoch - 2ms/step
Epoch 52/100
1270/1270 - 2s - loss: 3898.9429 - 2s/epoch - 2ms/step
Epoch 53/100
1270/1270 - 4s - loss: 3898.8806 - 4s/epoch - 3ms/step
Epoch 54/100
1270/1270 - 2s - loss: 3898.9060 - 2s/epoch - 2ms/step
Epoch 55/100
1270/1270 - 2s - loss: 3898.9348 - 2s/epoch - 2ms/step
Epoch 56/100
1270/1270 - 2s - loss: 3898.9077 - 2s/epoch - 2ms/step
Epoch 57/100
1270/1270 - 2s - loss: 3898.8762 - 2s/epoch - 2ms/step
Epoch 58/100
1270/1270 - 3s - loss: 3898.7686 - 3s/epoch - 2ms/step
Epoch 59/100
1270/1270 - 3s - loss: 3898.9441 - 3s/epoch - 2ms/step
Epoch 60/100
1270/1270 - 2s - loss: 3898.9270 - 2s/epoch - 2ms/step
Epoch 61/100
1270/1270 - 2s - loss: 3898.8828 - 2s/epoch - 2ms/step
Epoch 62/100
1270/1270 - 2s - loss: 3898.9355 - 2s/epoch - 2ms/step
Epoch 63/100
1270/1270 - 2s - loss: 3898.9312 - 2s/epoch - 2ms/step
Epoch 64/100
1270/1270 - 3s - loss: 3898.8542 - 3s/epoch - 3ms/step
Epoch 65/100
1270/1270 - 2s - loss: 3898.9866 - 2s/epoch - 2ms/step
Epoch 66/100
1270/1270 - 2s - loss: 3898.9512 - 2s/epoch - 2ms/step
Epoch 67/100
1270/1270 - 2s - loss: 3898.9224 - 2s/epoch - 2ms/step
Epoch 68/100
1270/1270 - 2s - loss: 3898.9497 - 2s/epoch - 2ms/step
Epoch 69/100
1270/1270 - 4s - loss: 3898.9092 - 4s/epoch - 3ms/step
Epoch 70/100
1270/1270 - 2s - loss: 3898.9573 - 2s/epoch - 2ms/step
Epoch 71/100
1270/1270 - 2s - loss: 3898.8445 - 2s/epoch - 2ms/step
Epoch 72/100
1270/1270 - 2s - loss: 3898.8872 - 2s/epoch - 2ms/step
Epoch 73/100
1270/1270 - 2s - loss: 3898.8286 - 2s/epoch - 2ms/step
Epoch 74/100
1270/1270 - 4s - loss: 3898.9358 - 4s/epoch - 3ms/step
Epoch 75/100
1270/1270 - 2s - loss: 3898.9446 - 2s/epoch - 2ms/step
Epoch 76/100
1270/1270 - 2s - loss: 3898.9099 - 2s/epoch - 2ms/step
Epoch 77/100
1270/1270 - 2s - loss: 3898.9104 - 2s/epoch - 2ms/step
Epoch 78/100
```

```
1270/1270 - 2s - loss: 3898.6936 - 2s/epoch - 2ms/step
Epoch 79/100
1270/1270 - 3s - loss: 3898.8159 - 3s/epoch - 3ms/step
Epoch 80/100
1270/1270 - 3s - loss: 3898.9016 - 3s/epoch - 2ms/step
Epoch 81/100
1270/1270 - 2s - loss: 3898.6243 - 2s/epoch - 2ms/step
Epoch 82/100
1270/1270 - 2s - loss: 3899.0032 - 2s/epoch - 2ms/step
Epoch 83/100
1270/1270 - 2s - loss: 3898.9712 - 2s/epoch - 2ms/step
Epoch 84/100
1270/1270 - 3s - loss: 3898.7434 - 3s/epoch - 2ms/step
Epoch 85/100
1270/1270 - 3s - loss: 3898.8860 - 3s/epoch - 2ms/step
Epoch 86/100
1270/1270 - 2s - loss: 3898.8503 - 2s/epoch - 2ms/step
Epoch 87/100
1270/1270 - 2s - loss: 3898.9758 - 2s/epoch - 2ms/step
Epoch 88/100
1270/1270 - 2s - loss: 3898.8123 - 2s/epoch - 2ms/step
Epoch 89/100
1270/1270 - 3s - loss: 3898.9211 - 3s/epoch - 2ms/step
Epoch 90/100
1270/1270 - 3s - loss: 3898.8220 - 3s/epoch - 2ms/step
Epoch 91/100
1270/1270 - 2s - loss: 3898.9460 - 2s/epoch - 2ms/step
Epoch 92/100
1270/1270 - 2s - loss: 3898.8064 - 2s/epoch - 2ms/step
Epoch 93/100
1270/1270 - 2s - loss: 3898.8455 - 2s/epoch - 2ms/step
Epoch 94/100
1270/1270 - 3s - loss: 3898.9128 - 3s/epoch - 2ms/step
Epoch 95/100
1270/1270 - 3s - loss: 3898.9402 - 3s/epoch - 3ms/step
Epoch 96/100
1270/1270 - 2s - loss: 3898.9270 - 2s/epoch - 2ms/step
Epoch 97/100
1270/1270 - 2s - loss: 3898.9290 - 2s/epoch - 2ms/step
Epoch 98/100
1270/1270 - 2s - loss: 3898.7324 - 2s/epoch - 2ms/step
Epoch 99/100
1270/1270 - 2s - loss: 3898.8066 - 2s/epoch - 2ms/step
Epoch 100/100
1270/1270 - 3s - loss: 3898.8987 - 3s/epoch - 3ms/step
341/341 [==============================] - 1s 2ms/step
16 Parameters: batch_size: 20 - epochs: 100
RMSE:  62.871
```

```
<ipython-input-52-cab26893af0b>:37: FutureWarning: The frame.append method
is deprecated and will be removed from pandas in a future version. Use pan
das.concat instead.
  SearchResultsData=SearchResultsData.append(pd.DataFrame(data=[[TrialNumb
er, str(batch_size_trial)+'-'+str(epochs_trial), rmse]],
```

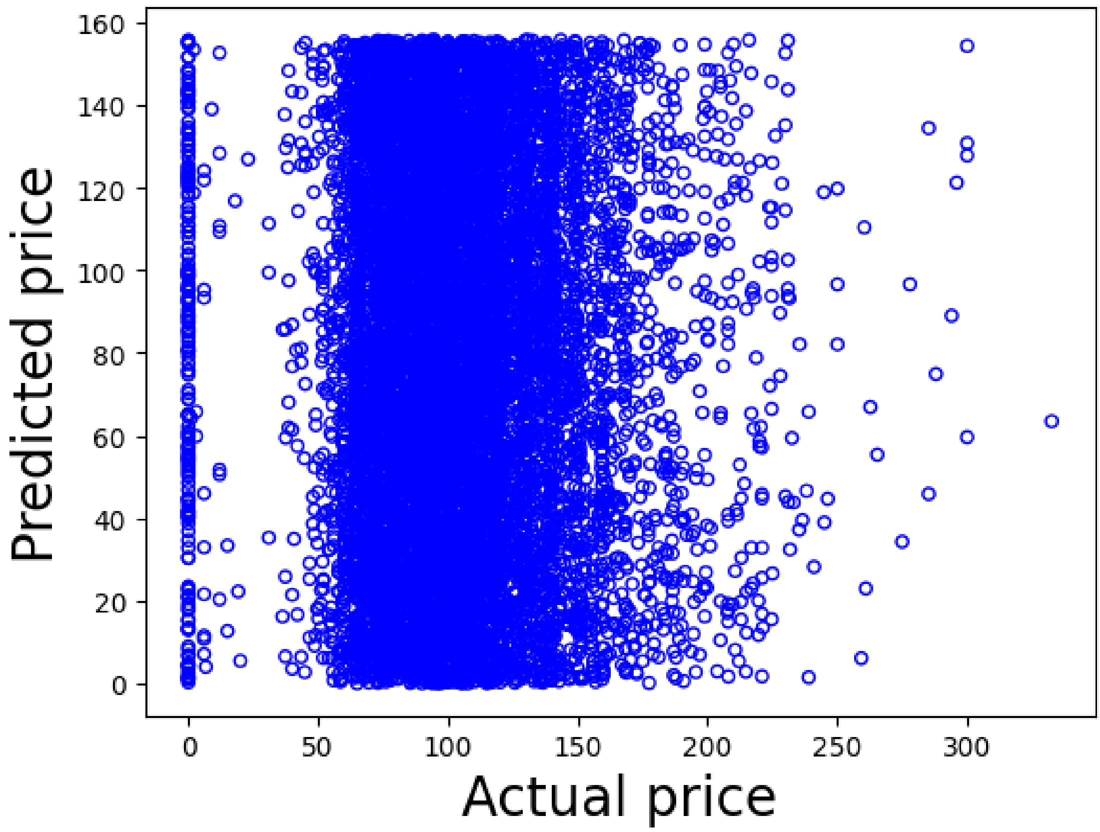
CNN Model Training with optimal parameters

```
# LeNet-5 structure for model training
model = keras.Sequential()
model.add(keras.layers.Conv2D(filters=6, kernel_size=(5,1), strides=(1,1),
padding='same', activation='relu', input_shape=(11, 1, 1)))  # The first c
onvolutional layer. Input shape is 11x1x1 and activation function is Recti
fied Linear Unit (ReLU).\
                                                           # Kernel size a
nd stride are set as (5,1) and (1,1), respectively. The number of convolut
ional kernels is 6. Besides, the zero padding is 'same', thus the output s
hape is (11/1) x 1 x 6 = 11x1x6, which is the same as the input shape.
model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1)))  # The f
irst maxpooling layer. The pool size and stride are both (2,1), thus the o
utput shape is 5x1x6, where 5 = (11-2)/2 + 1.
model.add(keras.layers.Conv2D(filters=16, kernel_size=(5,1), strides=(1,1)
, padding='same', activation='relu')) # The second convolutional layer. Th
e output shape is 5x1x16, where 5 = 5/1.
model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1)))  # The s
econd maxpooling layer. The pool size and stride are both (2,1). The outpu
t shape is 2x1x16, where 2 = (5-2)/2 +1.
model.add(keras.layers.Flatten())                          # The previous
output is flattened to be a vector.
model.add(keras.layers.Dense(120, activation='relu'))      # The first ful
ly connected layer.
model.add(keras.layers.Dense(84, activation='relu'))       # The second fu
lly connected layer.
model.add(keras.layers.Dense(1))                           # we have 1 neu
ron for output (SalePrice).
model.summary()                                            # Summary the c
onstructed model.
model.compile(tf.keras.optimizers.SGD(learning_rate = 1e-11), 'mean_square
d_error') # Model construction with a SGD optimizer and a mean squared err
or loss function.
model.fit(X_train, y_train, epochs = 10, batch_size = 15, verbose = 2)
# Model training with some hyperparameters.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 11, 1, 6)          36

 max_pooling2d (MaxPooling2D  (None, 5, 1, 6)          0
 )

 conv2d_1 (Conv2D)           (None, 5, 1, 16)          496

 max_pooling2d_1 (MaxPooling  (None, 2, 1, 16)         0
 2D)

 flatten (Flatten)           (None, 32)                0

 dense (Dense)               (None, 120)               3960

 dense_1 (Dense)             (None, 84)                10164
```

```
 dense_2 (Dense)              (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1693/1693 - 9s - loss: 5844.7217 - 9s/epoch - 5ms/step
Epoch 2/10
1693/1693 - 5s - loss: 3917.5786 - 5s/epoch - 3ms/step
Epoch 3/10
1693/1693 - 6s - loss: 3895.3079 - 6s/epoch - 4ms/step
Epoch 4/10
1693/1693 - 3s - loss: 3895.1597 - 3s/epoch - 2ms/step
Epoch 5/10
1693/1693 - 3s - loss: 3895.0632 - 3s/epoch - 2ms/step
Epoch 6/10
1693/1693 - 3s - loss: 3895.1160 - 3s/epoch - 2ms/step
Epoch 7/10
1693/1693 - 4s - loss: 3895.1067 - 4s/epoch - 2ms/step
Epoch 8/10
1693/1693 - 3s - loss: 3894.8655 - 3s/epoch - 2ms/step
Epoch 9/10
1693/1693 - 3s - loss: 3895.2314 - 3s/epoch - 2ms/step
Epoch 10/10
1693/1693 - 3s - loss: 3895.0596 - 3s/epoch - 2ms/step

<keras.callbacks.History at 0x7fd290287f70>
```

CNN Model testing

```
y_pred = model.predict(X_test)     # Prediction of the testing set

# Visualization
#fig, ax = plt.subplots( nrows=1, ncols=1 )  # create figure & 1 axis

plt.scatter(y_test, y_pred, s=20, marker='o', edgecolor=['blue'], c='none'
)
plt.xlabel('Actual price', fontsize=20)
plt.ylabel('Predicted price', fontsize=20)
plt.show()

#fig.savefig('/content/drive/MyDrive/ColabNotebooks/CS5812/CNN model befor
e Normalization.png')    # save the figure to file
#plt.close(fig)     # close the figure window

341/341 [==============================] - 1s 2ms/step
```



CNN Model Evaluation

```
model.evaluate(X_test, y_test)  #performance evaluation

341/341 [==============================] - 1s 2ms/step - loss: 3949.1582

3949.158203125

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)


RMSE:  62.842
```

Data normalization

We find the learning rate is very small, which is confusing. To solve this issue, a data normalization is performed.

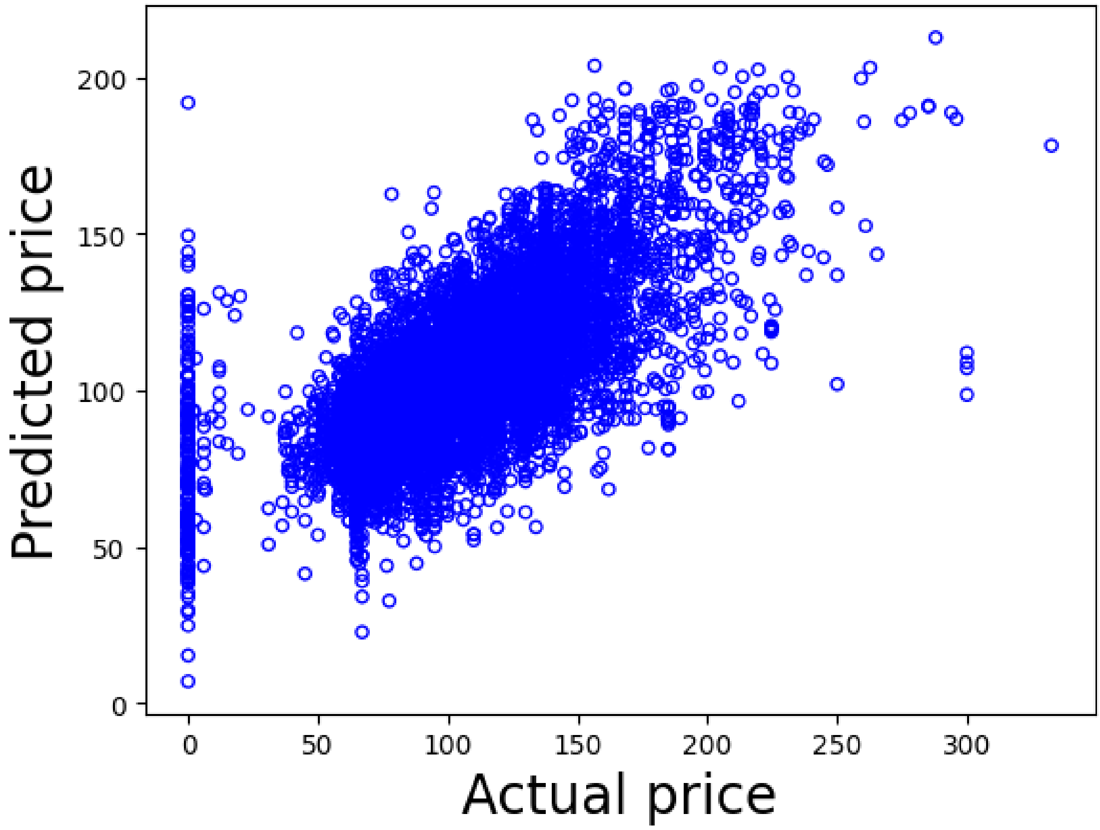
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42) # Randomly split training data and testing data with a rat
io of 7:3

scaler = StandardScaler()    # Standardization function
scaler.fit(X_train)          # Standardization fitting using training data
X_train = scaler.transform(X_train) # Standardization transformation of tr
aining data
X_test = scaler.transform(X_test)    # Standardization transformation of te
sting data
```

```
X_train = np.expand_dims(X_train, axis=2) # Add a dimension for each train
ing sample - 1 x 11 x 1
X_train = np.expand_dims(X_train, axis=2) # Add a dimension for each train
ing sample to form an image - 1 x 11 x 1 x 1
X_test = np.expand_dims(X_test, axis=2)   # Add a dimension for each testi
ng sample - 1 x 11 x 1
X_test = np.expand_dims(X_test, axis=2)   # Add a dimension for each testi
ng sample to form an image - 1 x 11 x 1 x 1

y_train_max = max(y_train)       # The maximum value of training labels
y_train = y_train/y_train_max    # Normalization of training labels
y_test = y_test/y_train_max      # Normalization of testing labels

# LeNet-5 structure for model training
model = keras.Sequential()
model.add(keras.layers.Conv2D(filters=6, kernel_size=(5,1), strides=(1,1),
padding='same', activation='relu', input_shape=(11, 1, 1)))  # The first c
onvolutional layer. Input shape is 8x1x1 and activation function is Rectif
ied Linear Unit (ReLU).\
                                                           # Kernel size a
nd stride are set as (5,1) and (1,1), respectively. The number of convolut
ional kernels is 6. Besides, the zero padding is 'same', thus the output s
hape is (8/1) x 1 x 6 = 8x1x6, which is the same as the input shape.
model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1)))  # The f
irst maxpooling layer. The pool size and stride are both (2,1), thus the o
utput shape is 4x1x6, where 4 = (8-2)/2 + 1.
model.add(keras.layers.Conv2D(filters=16, kernel_size=(5,1), strides=(1,1)
, padding='same', activation='relu')) # The second convolutional layer. Th
e output shape is 4x1x16, where 4 = 4/1.
model.add(keras.layers.MaxPool2D(pool_size=(2,1), strides=(2,1)))  # The s
econd maxpooling layer. The pool size and stride are both (2,1). The outpu
t shape is 2x1x16, where 2 = (4-2)/2+1.
model.add(keras.layers.Flatten())                          # Flatten the p
revious output to be a vector.
model.add(keras.layers.Dense(120, activation='relu'))      # The first ful
ly connected layer.
model.add(keras.layers.Dense(84, activation='relu'))       # The second fu
lly connected layer.
model.add(keras.layers.Dense(1))                           # we have 1 neu
ron for output (SalePrice).
model.summary()                                            # Summary the c
onstructed model.
model.compile(tf.keras.optimizers.SGD(learning_rate = 3e-2), 'mean_squared
_error') # Model construction with a SGD optimizer and a mean squared erro
r loss function.
model.fit(X_train, y_train, epochs = 10, batch_size = 15, verbose = 2)
# Model training with some hyperparameters.
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 11, 1, 6)          36

 max_pooling2d_6 (MaxPooling  (None, 5, 1, 6)          0
 2D)
```

```
 conv2d_7 (Conv2D)           (None, 5, 1, 16)          496

 max_pooling2d_7 (MaxPooling  (None, 2, 1, 16)         0
 2D)

 flatten_3 (Flatten)         (None, 32)                0

 dense_9 (Dense)             (None, 120)               3960

 dense_10 (Dense)            (None, 84)                10164

 dense_11 (Dense)            (None, 1)                 85

=================================================================
Total params: 14,741
Trainable params: 14,741
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1693/1693 - 3s - loss: 0.0068 - 3s/epoch - 2ms/step
Epoch 2/10
1693/1693 - 4s - loss: 0.0056 - 4s/epoch - 2ms/step
Epoch 3/10
1693/1693 - 3s - loss: 0.0052 - 3s/epoch - 2ms/step
Epoch 4/10
1693/1693 - 3s - loss: 0.0049 - 3s/epoch - 2ms/step
Epoch 5/10
1693/1693 - 3s - loss: 0.0048 - 3s/epoch - 2ms/step
Epoch 6/10
1693/1693 - 3s - loss: 0.0047 - 3s/epoch - 2ms/step
Epoch 7/10
1693/1693 - 4s - loss: 0.0046 - 4s/epoch - 2ms/step
Epoch 8/10
1693/1693 - 3s - loss: 0.0045 - 3s/epoch - 2ms/step
Epoch 9/10
1693/1693 - 3s - loss: 0.0045 - 3s/epoch - 2ms/step
Epoch 10/10
1693/1693 - 3s - loss: 0.0044 - 3s/epoch - 2ms/step

<keras.callbacks.History at 0x7fd2805e7670>

prediction = model.predict(X_test)    # Prediction of the testing set

# Visualization
fig, ax = plt.subplots( nrows=1, ncols=1 )  # create figure & 1 axis
plt.scatter(y_test*y_train_max, prediction*y_train_max, s=20, marker='o',
edgecolor=['blue'], c='none')
plt.xlabel('Actual price', fontsize=20)
plt.ylabel('Predicted price', fontsize=20)
plt.show()

fig.savefig('/content/drive/MyDrive/ColabNotebooks/CS5812/CNN model after
Normalization.png')   # save the figure to file
plt.close(fig)    # close the figure window
```

```
341/341 [==============================] - 1s 2ms/step
```




```
y_train_max**2 * model.evaluate(X_test, y_test)

341/341 [==============================] - 1s 2ms/step - loss: 0.0046

array([642.34260971])

# RMSE (Root Mean Square Error)
rmse = float(format(np.sqrt(mean_squared_error(y_test*y_train_max, predict
ion*y_train_max)), '.3f'))
print("\nRMSE: ", rmse)


RMSE:  25.344
```