Assignment report, group 32

Pol Mor Puigventós, Tushar Pal, Sudarshna Arya Patel

Leiden Institute of Advanced Computer Science, The Netherlands

Abstract. This document contains the analysis of a dataset for bike rentals depending on time and weather features. Then, different predictors including a linear regression model, SVM and a decision tree regressor predictor are implemented in order to predict the number of rentals based on the input features. Also, dimensionality reduction and unsupervised machine learning techniques, among other machine learning strategies, are studied. Finally, The analysis of the data, experiments and conclusions are presented.

1 Introduction

In this section we describe the tasks of the group assignment for the course Introduction to Machine Learning at Leiden University. We are reporting our analysis and findings according to the problem statement presented. All the experiments can be found in the Jupyter notebook attached with this document.

The document has the following structure. First, the dataset provided is analysed and commented. Once this is done, we formulate the problem to solve. After that, we describe all the experiments conducted and observations. Finally, we conclude the report with the major findings and possible future improvements. The figures can be found in the appendix at the end of the report.

2 Data Set

The purpose of this section is describing the data set provided, depicting the main features found and applying some pre-processing to the data in order to make it usable to solve the problem.

The data set contains data about bike rentals in a large European city. We are presented with data values for every hour of the day, every day, during two years. There are some days where there is lacking data normally between 1:00 and 5:00 in the early morning, we guess that no rentals are made in those time slots. In total, we have 17379 values. For each of those values the *date* and *time* are provided in format DD/MM/AAAA and HH respectively. Additionally, the *season*, the *year* (0 for 2011 and 1 for 2012), the name of the *month* and the *weekday*. Using these last values we will be able to exclude the *date* feature and use the others to train our model. Furthermore, we are provided with weather features for each value, including *Temperature* (we guess in degrees Celsius), *Feeling temperature*, *Humidity*, *Windspeed* and a description of the weather at that moment as *Weather*. Finally, we are told the number of subscribed, non-subscribed and total of people who used the service for each time slot. See Table 1.

Table 1: Description of the Dataset features.

Feature	**Description**	**ML analysis**	**Format**	**Distribution**
instant	Sequential value	unusable	Number	-
date	Calendar date	unusable	DD/MM/YYYY	-
season	Names of the four seasons.	Usable, via one-hot-encoding, very generalising	String	Uniform
year	Flag indicating the year.	Usable, differences observed between both years	0,1	0 for 2011, 1 for 2012
month	Names of the twelve months.	Usable, via one-hot-encoding	String(3)	Uniform
hour	Numbers from 0 to 23.	Usable, via one-hot-encoding grouping 2 hours	Number	Uniform
weekday	Names of the seven weekdays.	Usable, via a new boolean variable is_weekday	String(3)	Uniform
weather	Four possible weather situations.	Usable, via one-hot-encoding	String	-
temperature	Measured temperature, probably in degrees Celsius.	Usable, scaled and normalised	Decimal number	Gaussian
feeling_ temperature	Feeling temperature proportional to the previous value.	Usable, scaled and normalised	Decimal number	Gaussian
humidity	Measured humidity from 0 to 100.	Usable, scaled and normalised	Number	Gaussian
windspeed	Measured windspeed.	Usable, scaled and normalised	Decimal number	Gaussian
Subscribed	Number of rentals from subscribed users during the specified hour.	Usable, scaled and normalised	Number	Decreasing exponential
Non-subscribed	Number of rentals from unsubscribed users during the specified hour.	Usable, scaled and normalised	Number	Decreasing exponential (fat tail)
Total	Sum of the two previous values.	Usable, scaled and normalised	Number	Decreasing exponential (fat tail)

The exploration of the data gives us some idea of the behavior of it:

– The instant value is a sequential value, useful for a database purpose but not usable for training a model. We drop this value.

– The date is good for analysing the data but won't be usable to train a model, we are instead using some of the following features of every date. If we had a value 'day of the month' we would probably drop it too, because not relevant information can be obtained about it and we might risk over-fitting. This feature would be useful if we knew the location where the data was collected in the direction of adding the feature 'holiday' where different behavior would be expected.

– The season is somehow very correlated to the month of the year and we don't expect it to be more related than the month when predicting bike rentals behavior. Nevertheless using this feature we can generalise more than using the month value. Encoding this feature with cyclical methods can be useful, as the first and the last season have similarities.

– The year (binary) number is very relevant. We observe in average much more rentals during 2012 than in 2011. This is why when predicting rentals for the second year, the model might add a positive bias respect when predicting rentals for 2011.

- The month string is encoded with the One-Hot encoding method. Encoding this feature with cyclical methods can be useful, as the first and the last month have similarities.
- The hour number is encoded with the One-Hot encoding method. Encoding this feature with cyclical methods can be useful, as the first and the last hour have similarities.
- The weekday string is encoded with the One-Hot encoding method. **In figure 5**, we distinguish the similar behavior of days from monday to friday from the days of the weekend for non-subscribers. Monday to friday two rentals peaks are registered generally with their maximums at around 8:00 and 18:00 (with a curve rather than a peak in the second maximum, meaning a bigger timespan when renting a bike in the afternoon). On the contrary, in the weekend we observe a normal distribution of rentings along the whole day with maximum at 14:00 and minimum at 5:00.
- The weather string is encoded with the OneHot encoding method. We observe in **figure 5** that Subscribed people to the service might be less influenced by the weather, compared to the Non-Subscribed users.
- The temperature number is used scaled and normalised. We observe in **figure 6** a skewed normal distribution with a plateau between 15 and 30 degrees.
- The feeling temperature has a similar distribution as the previous feature. In addition, we observe some unusual data points in **figure 7** when the feeling temperature is around 9 and 21. Consequently, we decide to analyse and maybe verify the performance of this feature.
- The humidity is used scaled and normalised. We observe in **figure 8** a Normal distribution centered in 55% with high deviation, resulting in very similar values for humidity between 10% and 90%.
- The windspeed is used scaled and normalised. We ignore if the units are miles, kilometers or knots per hour. We detect in **figure 9** that the anemometer used only measured windspeeds above 6, so we lack values between 0 and 6. The distribution is normal, with mean around 11 and large deviation. There is also a peak in 0 assuming all windspeeds from 0 to 6 and gathering a 12,5% of the data. Also, we read that windspeed is negatively linearly correlated to the total of rentings.
- The Subscribed number is used scaled and normalised.
- The Non-subscribed number is used scaled and normalised.
- The Total number is used scaled and normalised.

Once we explored the data, we are adapting some values for the correct use of the dataset. The feature *weather* has the following possible values and occurrences in the dataset:

- *Clear or partly cloudy*: 11413
- *Mist*: 4544
- *Light rain*: 1419
- *Heavy rain*: 3

In the interest to optimally train the model, we want to avoid categorizing a feature that represents $0,01\%$ of the points in the dataset. Thus, we are considering the three time slots of *Heavy rain* as *Light rain*, resulting in a more balanced distribution of possible distinctions for this feature.

Finally, We are missing 165 values in our dataset where we suppose that no rentals were made. This is going to lead us to small deviations in our predictions, for instance never predicting zero rentals, as never happens in the train set. However, in the test set we won't find a case where no rentals were made either. So, no corrections have to be made for the purposes of this assignment.

2.1 Problem formulation

In this assignment we are asked to develop solutions to predict the number of bikes rented in an European city between 2011 and 2012. The bike rental service rents bikes to subscribed and non-subscribed customers.

3 Experiments

3.1 Linear Regressor

Feature creation

We created a LinearRegression model using sklearn inbuilt packages. The features used for creating training vectors **x** are:

1. **hour** One-Hot encoded to 24 categories.
2. **weekday** column was split into two binary categories called **is_weekday** and **is_weekend**. This was done as the behaviour during weekdays and weekends is constant, but stark when compared to each other.
3. **month** One-Hot encoded to 12 categories.
4. **season** One-Hot encoded to 4 categories.
5. **weather** One-Hot encoded to 3 categories.
6. **temperature** scaled and normalised to [0,1] interval.
7. **feeling_temperature** scaled and normalised to [0,1] interval.
8. **humidity** scaled and normalised to [0,1] interval.
9. **windspeed** scaled and normalised to [0,1] interval.
10. **year** used without alterations.

For the **y** labels, **Total**, **Subscribed**, **Non-Subscribed** values were scaled and normalised to [0, 1] interval.

Model performance

To predict the amount of bikes rented using linear regression we are using multiple linear regressions models one for **y** label. The base model with no transformations yields a train MSE of 0.010, R-Squared of 0.688 and a test MSE of 0.011, R-Squared of 0.674. We experimented with various train-test split and it is observed that optimal split lies between 70:30 and 80:20 for the given data set as it is observed in **figure 10**, This is due to the over-fitting of the model as the training split increases. However, for y-label subscribed the optimal split would lie from 60:40 to 70:30 and in case non-subscribed it is from range 70:30 to 80:20

Transformation Analysis

Additionally, we experimented with different transformations such as logarithmic and Quantile, we observed that the performance of the model improved drastically from 0.674 R-squared testing to 0.815 and 0.688 R-squared testing to 0.823 when using transformations especially logarithmic transformation as the log transformation reduces or removes the skewness of original data. Similarly, Quantile transformation helped improving the performance as well where R-Squared testing reached 0.740 from 0.674 as shown in the **figure 14**.

Encoding Analysis

Furthermore, we analysed the impact of **cyclic encoding** on feature *hour* and measured the performance. After looking upon the performance we concluded that for the this data-set, **One-Hot encoding** provides better results than cyclic encoding as the R-squared testing of the model dropped significantly from 0.674 to 0.48 which can be seen in our source code provided.

Analysis of impact(s) of various x-vectors

Furthermore, we also compared the outcomes based on particular features included in training model such as with or without humidity, temperature vs feeling_temperature, season included vs not. The impact of features on the overall model can be seen **figure 16** and **figure 18**

Reducing number of hours

In addition, we experimented with less number of features by grouping hours in 2-hour range so that we have 12 hour instead of 24. We noticed that performance dropped from 0.690 to 0.635 which is not significant, hence no major changes were notices which we can see in **figure 20**

Observations

Upon measuring the performance of above experiments our observations are as follows:

1. Log and Quantile transform on y label yields significantly better MSE and R-Squared than MinMax normalised transformation.
2. Inclusion of both the features temperature and feeling_temperature is redundant as there is no significant difference in the performance, similarly in case of inclusion of humidity and inclusion of season when using months as a feature for our model.

3.2 Decision Tree regression model

Feature creation

We created a DecisionTreeRegressor model using sklearn inbuilt packages. The features used for creating training vectors **x** are:

1. **year** with no changes.
2. **hour** One-Hot encoded to 24 categories.
3. **weekday** column was split into two binary categories called **is_weekday** and **is_weekend**. This is done as the behaviour between weekdays and weekends is significantly different.
4. **month** One-Hot encoded to 12 categories.
5. **season** One-Hot encoded to 4 categories.
6. **weather** One-Hot encoded to 3 categories.
7. **temperature** scaled and normalised to [0,1] interval.
8. **humidity** scaled and normalised to [0,1] interval.
9. **windspeed** scaled and normalised to [0,1] interval.

For the **y** labels, **Subscribed** and **Non-subscribed** values were log normalised.

Model performance

The base model with no hyperparameter optimisation, yielded a train and test MSE of **1.145** and **1.103**, and train and test R^2 error of **0.5560**.

	column_name	importance
1	temperature	0.429331
2	humidity	0.252082
3	hour	0.013039
4	weekday	0.002537
5	month	0.000049
6	season	0.000000
7	weather	0.000000
8	year	0.000000
9	windspeed	0.000000

Table 2: Feature importances for base Decision Tree model

From the model feature coefficients above, we also observe that the **temperature, humidity, weekday** and **hour** are the most important features for the model. Bicycle riders are reluctant to ride in extreme cold or hot temperatures, as well as in highly humid (rain, hot summer day) or highly dry (winter) conditions, explaining the importance of temperature and humidity. There are also significant differences in rental numbers across hours, and between weekdays and weekends.

Hyperparameter tuning

We then select the following hyperparameters, cycling them through a range of possible values to train and evaluate model performance:

1. **test_size** : The ratio of train and test split of the data.
2. **max_depth** : The maximum number of nodes possible between root node and a leaf node.
3. **min_samples_split** : The minimum number of samples that must be present for a non leaf node to be allowed to split into leaf nodes.
4. **min_samples_leaf** : The minimum number of samples that must be present for a leaf node to be allowed to split. Note that this overrides the min_samples_split hyperparameter. If a node has samples equal to or above the min_samples_split, but upon splitting, one of the leaf has samples below the min_samples_leaf, then this split will not be allowed.

The parameters max_depth, min_samples_split and min_samples_leaf allow us to control tree complexity. A more complex model takes more time to train, and also ends up overfitting, which we want to avoid.

We observe as in **figure 21**, that the depth of the tree has the most observed effect on model performance. The optimal hyperparameter values we arrive at are:

1. **test_size** : 0.3
2. **max_depth** : 25
3. **max_samples_split** : 20
4. **max_samples_leaf** : 8

Predicting Non-subscribed user numbers

With the hyperparameters used in the base model, we get MSE and R squared as **0.8004** and **0.6260** respectively. The slightly better performance compared to Subscribed customers model, is probably due to the prominent spikes in Non-subscribed customer numbers in the morning and evening hours, compared to more muted behaviour by Subscribed customers. Non subscribed customers also have a drastic difference in weekday vs weekend rentals, while Subscribe customers do not.

Plugging in the derived optimal hyperparameter values and training again to predict Non-subscribed rentals, we observe that our test R squared improves by **61%**, and MSE drops by **79%**. The greater the depth of a tree model, the more it can learn. But after a certain depth, the model will begin to overfit.

Observations

The following observations were made:

1. The inclusion of both months and seasons is not required, since they render the other redundant. The inclusion of seasons yields better performance than months, possibly due to the reduction in dimensionality.
2. Log transformed y values yield better MSE and R squared than MinMax normalised transformation.
3. The importance of features changes when the transformation of y changes. This is most probably due to the way the model fits the samples to the transformed output.
4. On a same hyperparameters, Non-subscribed rentals is predicted better than Subscribed.
5. Optimised hyperparameters drastically increase the performance of the Non-subscribed model.

3.3 Transforming into classification problem

To transform the problem of regression into classification with five classes we decided to divide the data into bins based on target value *Total* using *pandas.cut*. Similarly we also experimented with *pandas.qcut* which divide the data into buckets based on sample quantile.

Furthermore, we turned the classification problem into multiple binary classification problems where we experimented *one-vs-all* and *one-vs-one* using *logistic regression*. The **table 3** shows the accuracy and execution time in case of both methods.

	Accuracy		Execution Time (sec)	
	Value bins	**Quantile bins**	**Value bins**	**Quantile bins**
one-vs-all	0.751	0.599	2.49	2.68
one-vs-one	0.770	0.643	1.95	2.42

Table 3: Performance table for OvO and OvR

Upon observing, we noticed that OvO provides better accuracy in lesser execution time for this dataset which is for following reasons.

For one-vs-all, number of classifiers required is n i.e. 5 as it involves one class against the rest of the dataset. However, for one-vs-one, number of classifiers would be $n(n-1)/2$ i.e. 10 as this splits the dataset into one vs one other class from the remaining classes where n is the number of original classes. As a result, one-vs-all might take less execution time as it trains the less number of classification models. However, it took longer in our scenario which is because one-vs-all compares one class to the rest of them, hence it becomes slower when we have a large dataset, it would be a slower model to train as the data is divided into two parts every-time. On the other hand, one-vs-one on splits the original data for a single binary classifier so would be faster as data as divided into batches which is 10 segments here.

Furthermore, one-vs-all contains a possibility of class dominance in original data and in that case one vs one would be a better option as it is less prone to creating an imbalance in the dataset. As a result one-vs-one provides better accuracy when compared to one-vs-all as it prevents the imbalance in output class.

3.4 Perceptron Method

The aim of this subsection is to predict the *weather* feature based on the rest of features. We apply subset selection and classify the predictions using the one-vs-one method.

The implementation of that strategy consists on creating a Multi Layer Perceptron classifier (MLPClassifier) and applying the one-vs-one method to this model. That way, we split the multi-class classification into one binary classification problem per each pair of classes. We need $k(k-1)/2$ binary classifiers, given k the number of classes. For this case, $k = 4$ and 6 classifiers are needed. For every binary classification, one class is predicted. At the end of the 6 one-vs-one classification and the one with the most predictions or votes is predicted by the one-vs-one strategy.

We are asked about the behavior of the algorithm when modifying the learning rate of the perceptron. In oder to measure different approaches, we test for the initial learning rate values 0.001 and 0.01. In addition, three learning rate strategies are expermiented. Finally, the early stopping method is used.

When it comes to the learning rate schedule for weight updates, *constant* maintains the initial learning rate value, $invscaling$ uses an inverse scaling component to gradually decrease the learning rate and $adaptive$ divides by 5 the learning rate when the training loss fails to decrease. In addition, the early stopping method is tested, creating a 10% of validation set and terminating the training when the score in this validation set stops improving.

We experiment with the three learning rate different approaches but observe the same values. See Table 4. We understand that all three strategies reach the optimal accuracy value. The striking feature observed is that using early stopping the computational time is reduced drastically and the accuracy is improved compared to the baseline model. Our intuition is that with early stopping the training stops before overfitting (the accuracy value in the training set is lower than the baseline) and generalises better in the test set.

Table 4: Accuracy obtained when testing different learning rate strategies on the one-vs-one binary perceptron model to predict the weather situation.

		Learning rate value			
		Train		*Test*	
		0.001	0.01	0.001	0.01
Learning parameters	*constant*	0.772	0.843	0.697	0.655
	invscaling	0.772	0.843	0.697	0.655
	adaptive	0.772	0.843	0.697	0.655
	Early stopping	0.709	0.735	**0.701**	0.694

When looking for hyperplanes that separate the data we observe that the classes are overlapping. So, it is not possible to linearly separate the data classes using MLP. Plus, a warning from sklearn appears while training: *ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.* We understand that this is because meteorological data has relevant entropy. Indeed, The temperature, windspeed and humidity values can't determine the weather situation accurately and the labelling of the output *situation* is also subjective. Also, the category *Heavyrain* with just 3 occurrences make more difficult this task, adding noise and complexity to the model.

3.5 Outlier detection

Sometimes the data contains the values which lie outside the range of what is expected or different from other data. These data points are called outliers. These outliers can create a deviation in the final result of machine learning models. As a result, a ML model can be improved if outliers can be detected to understand data better or replaced by median value or sometimes be removed from the data to improve the performance of overall model.

To identify outliers in our data, we used visualization techniques using *Box Plots* and *Bar Graphs*. The box plot is a standardized way of plotting the distribution of data based on the five-number summary (minimum, first quartile (Q1), median, third quartile (Q3), and maximum). It is often used to identify data distribution and detect outliers. In **figure 22** we can see that, there are a few outliers in data columns *humidity* and *windspeed*. We used box plot only on windspeed, humidity, temperature and feeling_temperature as these are only values which are numerical in nature, other features are categorical in nature and for those we use bar graphs.

In **figure 2** we see that in case of weather, feature *weather* contains *Heavy rain* which is an outlier wrt to other values present in out dataset as other values are in thousands but heavy rain is less almost zero which is negligible compared to others.

3.6 SVM Regressor

In this experiment, we trained an SVM Regressor to predict the number of bike rentals by Subscribers for a given hour in a day. We used the same features as before, with the Subscriber counts being log transformed. All hyperparameters were set to default, and KFold validation with k=5 was performed. The average train and test R^2 score across folds was 0.89 and 0.81 respectively.

We then changed only the kernel hyperparameter to **linear, poly** and **sigmoid** and performed the same KFold cross validation. The results of which are as shown in table below.

Right off the bat, we can eliminate sigmoid as a good kernel to use. Sigmoid kernel performs best when used for binary classification, not in multi class, and definitely not in regression. Linear kernel performs well on linearly separable data, which is true for our usecase as well shall see in the sections on PCA dimensionality reduction. However, it is unable to fit all datapoints on a linear curve. Polynomial kernels tend to overfit with higher degrees, hence we see a higher train R^2 score, and a significantly lesser test score. We thus conclude that for our given usecase, rbf is the best kernel to use, as it shows the highest train and test R^2scores.

kernel	train_mse	test_mse	train_R_squared	test_R_squared
linear	0.473385	0.527805	0.812078	0.76225
poly	0.271312	0.545692	0.892281	0.75262
rbf	0.275611	0.408538	0.890574	0.816634
sigmoid	2291.05	829.016	-910.337	-372.581

Table 5: Kernel performances

3.7 Dimensionality reduction with PCA

In this subsection we study the effect of applying dimensionality reduction to our dataset. We use the Principal Component Analysis (PCA) method.

In this first example, we provide all the features of the dataset to the PCA and obtain the scatter presented in Figure 23. We decide to delete the column of the dataset containing *feeling temperature* and the resulting plot of a new PCA is exactly the same. We continue deleting all the calendar features: *month, weekday, season, year* and *hour*. At this moment, we obtain the scatter presented in Figure 24. In this figure, the lighter points are distributed in a easiest way to separate them from the darker ones. We observe that the dimensionality reduction analysis does not give much importance to these classes when predicting the number of rentals, but give priority to the weather conditions.

We decide to restart the experiments and do the opposite process. We keep only the calendar features and delete all the meteorological data, obtaining the Figure 25. As all those are discrete features, we observe the data distributed in a grid format. Finally, we decide to use subset selection to get the PCA which best separates the number of bike rentals. The classes used are *year, month, hour, weekday/weekend, weather* and *feeling temperature*, obtaining the plot in Figure 26. We consider this analysis more accurate than the previous, we can distinguish clearly the location of the dark points from the lighter ones. However, we understand that this dimensionality reduction may cause underfitting in the predictor.

3.8 Hierarchical clustering and PCA

For this purpose we select the sklearn module AgglomerativeClustering. The bottom-up method is defined by measuring linkage distance recursively merge pair of clusters of sample data. We experiment with the different linkage strategies without observing major differences. We apply clustering to all the features in the dataset except the numbers of bike rentals and plot the labels in different 2D plots, modifying at each time the axis presented. For instance, when observing the labels generated in terms of months and feeling temperature, in Figure 27, we observe clusters depending on the temperatures and month distributions. Although, when analysing the scatter with axis feeling temperature and temperature, in Figure 28, we observe high correlation between both classes and is hard to distinguish any intuitive clustering.

In addition, we plot the hierarchical dendrogram in Figure 29, without observing relevant features.

We are also asked to project the PCA plot of the previous exercise (Figure 23) and compare it to a new PCA based on the previous clustering. We decide $n_{clusters} = 4$ and plot the result in Figure 30. Interestingly, the clustering method is able to distinguish the two left subgroups (in blue and yellow colors) from the rest of data. These areas are the ones that contain the highest values of bike rentals in the PCA plot. The other two areas of the scatter have much darker points in the PCA plot, indicating low number of rentals.

If we decide the same strategy but first applying clustering and then PCA, we observe a similar plot, in Figure 31, with different distribution of the clusters. However, the left cluster in the image is maintained, indicating the accumulation of similar points that result in high number of rentals.

We understand that the principal components and clustering extraction techniques obtain similar results, caused by a high-dimensional complex data hard to cluster or reduce its dimensionality.

3.9 Random Forest Regressor

In this experiment, we trained a random forest regressor to predict the total number of rentals for a given hour in a day. On just the default parameters, with a train test ratio of 0.2, we achieved a less than stellar OOB score of **0.3018**, as well as train and test R^2 of **.3021** and **.3266)**.

We then proceeded to perform a random search over a set of hyperparameter values. Some of the hyperparameters like **max_depth, min_samples_leaf** and **min_samples_split** have the same values as used in our DecisionTreeRegressor optimization task, as we want to compare between the two models. We used three splits of the data, over ten iterations for the random search, the final model having a max depth of 25, with around 800 estimators. Minimum samples for a leaf formation is 4, with samples required for a split being 40. OOB scoring has been set to true. The optimized model performs much better, with an OOB score of 0.92, train and test R^2 being 0.93 and 0.91 respectively.

The OOB score is calculated by taking a subset of the estimators in a random forest model, then generating the ratio of correctly predicted samples to total samples by this subset of estimators. The samples for calculating the score are those not included in the bag of samples used to train each estimator. Meanwhile, validation R^2 is calculated on the validation fold samples exclusively set aside during training. Hence, comparing them is not an apples to apples situation. OOB score gives a good idea of how well our model has generalised, while validation R^2 tells us how well our model has been trained, for a given set of hyperparameters. In this case, we see that our model does both well.

On visualizing the structure of a random estimator in the model, we see that the tree is very deep, utilizing the max_depth=25 hyperparameter completely. In contrast to this, the structure of the decision tree model from the 3rd exercise Part 1 is much shallower, even though the max depth allowed there is 25 as well. The same can be said for the width of the Random Forest estimator tree, as it is much wider with more branches.

3.10 Most Accurate Predictor(s)

Our top three contenders for solving such a regression problem were Decision Tree, SVM and Random Forest regressors. The highest R^2 scores for train and test obtained can be seen below. These were on models trained to predict total rentals.

model	train R^2	test R^2
SVM Regressor	0.89	0.81
DecisionTreeRegressor	0.93	0.90
Random Forest Regressor	0.93	0.91

Table 6: Comparison of R^2 scores across models

We see that Random Forest performed the best, narrowly edging out DecisionTreeRegressor for the top spot. SVM came in third, but the scores were quite a bit lower. Random Forest is able to avoid bias due to the fact that it uses subsets of the features, and then subsets of the samples for those features to train the various estimators. This avoids overfitting, and helps the model perform better than Decision Trees. SVM models tend to not overfit, but due to the conservative nature of the decision boundaries, perform worse than tree based models.

4 Conclusion

We agree that *Log* transformation provides the best accuracy for all the predictors. In addition, dropping similar features avoids redundancy in the train set and improves the performance and generalisation of the predictors.

We observed that optimal split (between train and test data) is different for Subscribed and Non-Subscribed datasets. Additionally, we also determined the outliers in our data to understand it better.

When it comes to the decision tree, the values of the hyperparameters that optimise the solution for Subscribers also performs accurately with Non-Subscribers. We believe this is because a wide research has been done and a selection of the hyperparameters that best fits the problem provides good accuracy for the predictor.

Transformation of this regression problem into classification using binary classifiers revealed that one-vs-one would be better predictor in comparison to one-vs-all model with an accuracy of 77%.

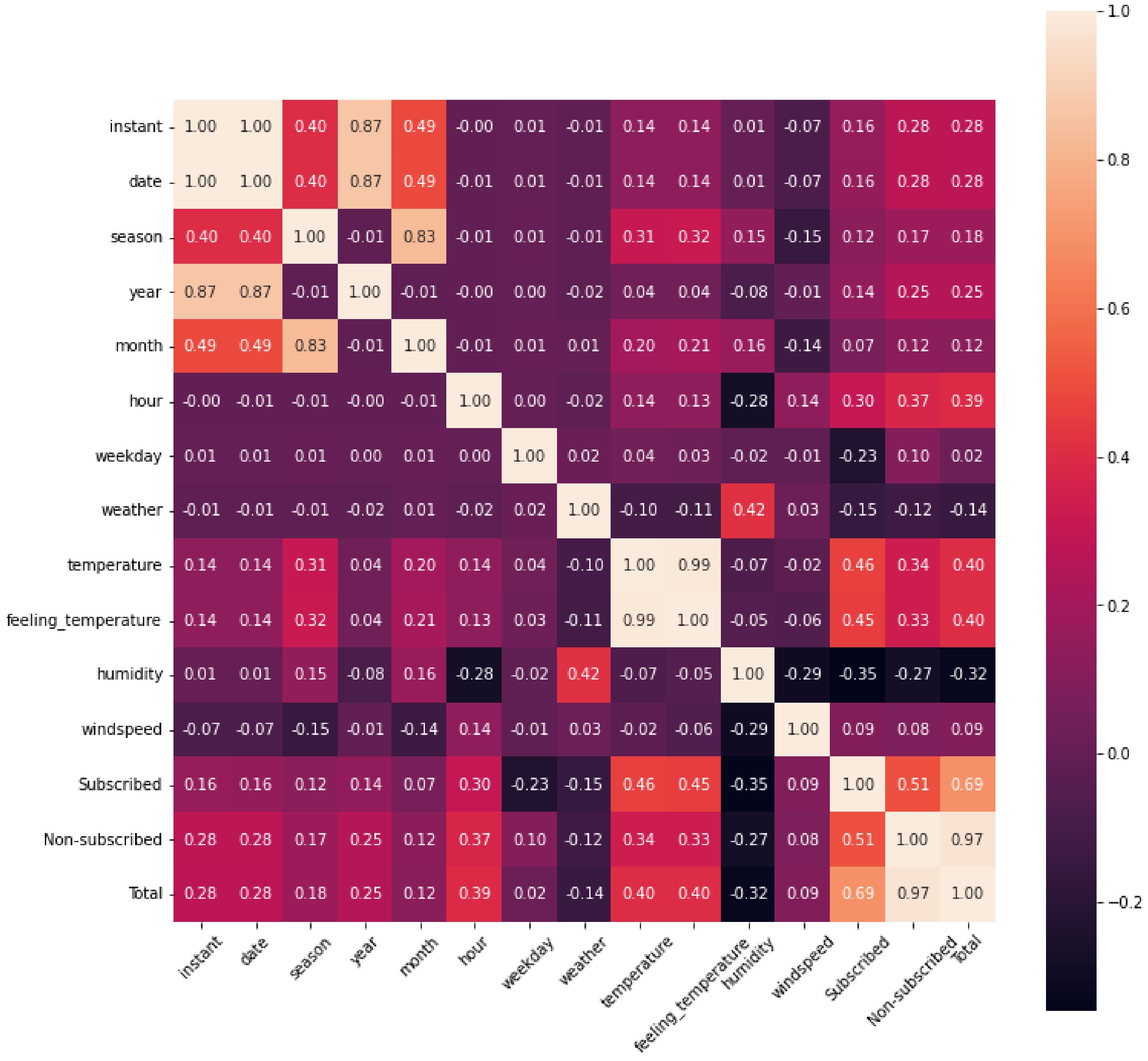
The Perceptron using the one-vs-one strategy manages to predict the weather condition with 70% of accuracy when adjusting its hyperparameters. It is crucial to use early stopping techniques to obtain this result, avoiding overfitting in the large dataset.

For different kernels in SVM Regression, different hyperparameters need to be tuned. For poly kernel, setting higher degree will cause an overfit. Sigmoid kernel does not do well for multiclass and regression. RBF usually performs best.

According to dimensionality reduction, we observe low significance of the calendar features. However, a combination of the relevant features allow the PCA to create a scatter where high number of rentals are established in the same coordinates. In addition, when using clustering techniques we observe that the algorithm is able to slightly separate the subgroup of data points generating high number of rentals from the ones that don't.

Based on our experiments, we found that Random Forest models, while more time consuming to train, perform the best. It would be best to apply Decision trees to obtain an explainable model during experimentation, while Random forest model would do well for the final model to be put into the production system.

A Figures: Data Set exploration



	instant	date	season	year	month	hour	weekday	weather	temperature	feeling_temperature	humidity	windspeed	Subscribed	Non-subscribed	Total
instant	1.00	1.00	0.40	0.87	0.49	-0.00	0.01	-0.01	0.14	0.14	0.01	-0.07	0.16	0.28	0.28
date	1.00	1.00	0.40	0.87	0.49	-0.01	0.01	-0.01	0.14	0.14	0.01	-0.07	0.16	0.28	0.28
season	0.40	0.40	1.00	-0.01	0.83	-0.01	0.01	-0.01	0.31	0.32	0.15	-0.15	0.12	0.17	0.18
year	0.87	0.87	-0.01	1.00	-0.01	-0.00	0.00	-0.02	0.04	0.04	-0.08	-0.01	0.14	0.25	0.25
month	0.49	0.49	0.83	-0.01	1.00	-0.01	0.01	0.01	0.20	0.21	0.16	-0.14	0.07	0.12	0.12
hour	-0.00	-0.01	-0.01	-0.00	-0.01	1.00	0.00	-0.02	0.14	0.13	-0.28	0.14	0.30	0.37	0.39
weekday	0.01	0.01	0.01	0.00	0.01	0.00	1.00	0.02	0.04	0.03	-0.02	-0.01	-0.23	0.10	0.02
weather	-0.01	-0.01	-0.01	-0.02	0.01	-0.02	0.02	1.00	-0.10	-0.11	0.42	0.03	-0.15	-0.12	-0.14
temperature	0.14	0.14	0.31	0.04	0.20	0.14	0.04	-0.10	1.00	0.99	-0.07	-0.02	0.46	0.34	0.40
feeling_temperature	0.14	0.14	0.32	0.04	0.21	0.13	0.03	-0.11	0.99	1.00	-0.05	-0.06	0.45	0.33	0.40
humidity	0.01	0.01	0.15	-0.08	0.16	-0.28	-0.02	0.42	-0.07	-0.05	1.00	-0.29	-0.35	-0.27	-0.32
windspeed	-0.07	-0.07	-0.15	-0.01	-0.14	0.14	-0.01	0.03	-0.02	-0.06	-0.29	1.00	0.09	0.08	0.09
Subscribed	0.16	0.16	0.12	0.14	0.07	0.30	-0.23	-0.15	0.46	0.45	-0.35	0.09	1.00	0.51	0.69
Non-subscribed	0.28	0.28	0.17	0.25	0.12	0.37	0.10	-0.12	0.34	0.33	-0.27	0.08	0.51	1.00	0.97
Total	0.28	0.28	0.18	0.25	0.12	0.39	0.02	-0.14	0.40	0.40	-0.32	0.09	0.69	0.97	1.00

Fig. 1: Feature correlations

(Return to Data Set section.)

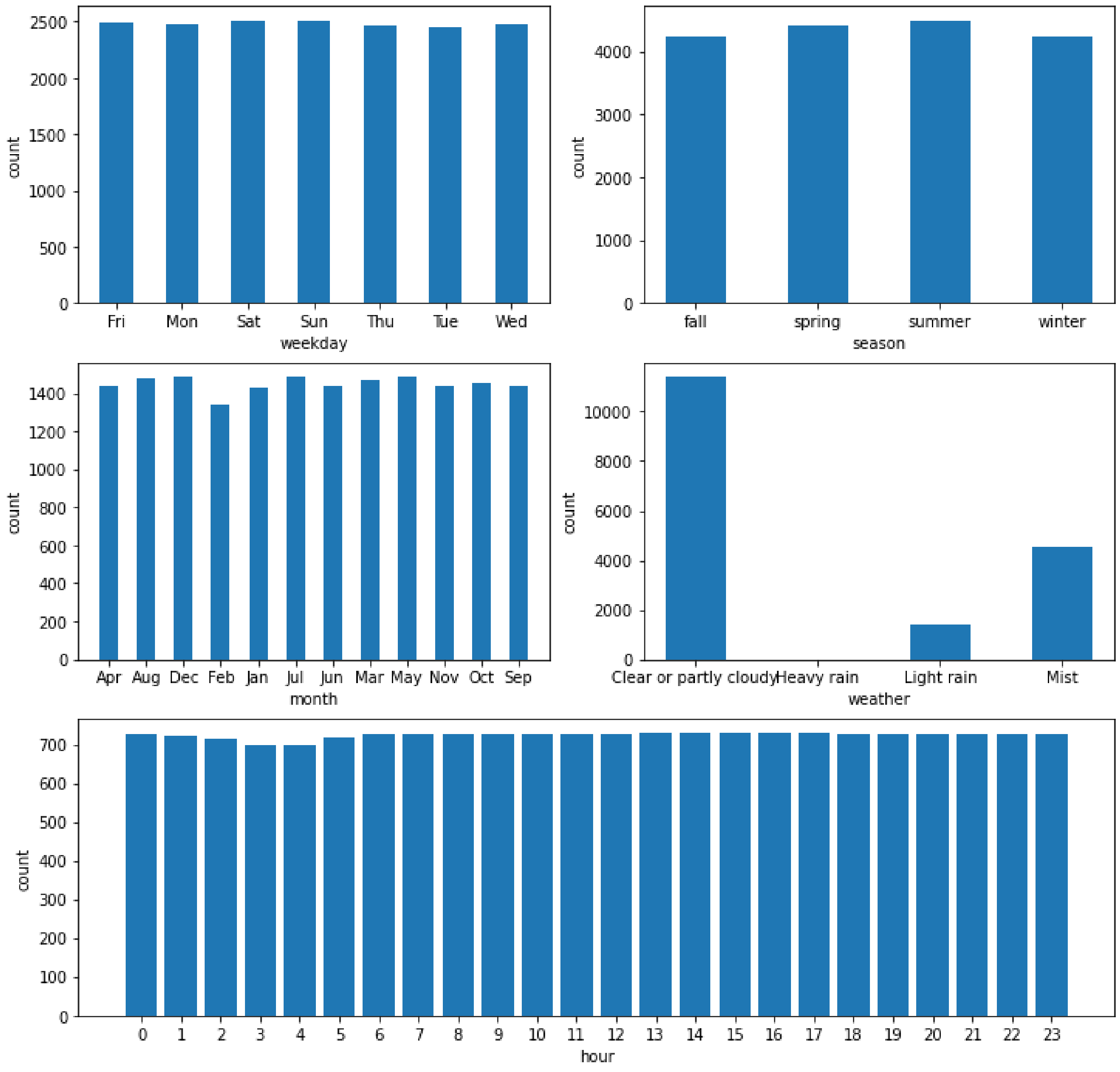


Fig. 2: Time components distribution

(**Return to Data Set section.**)

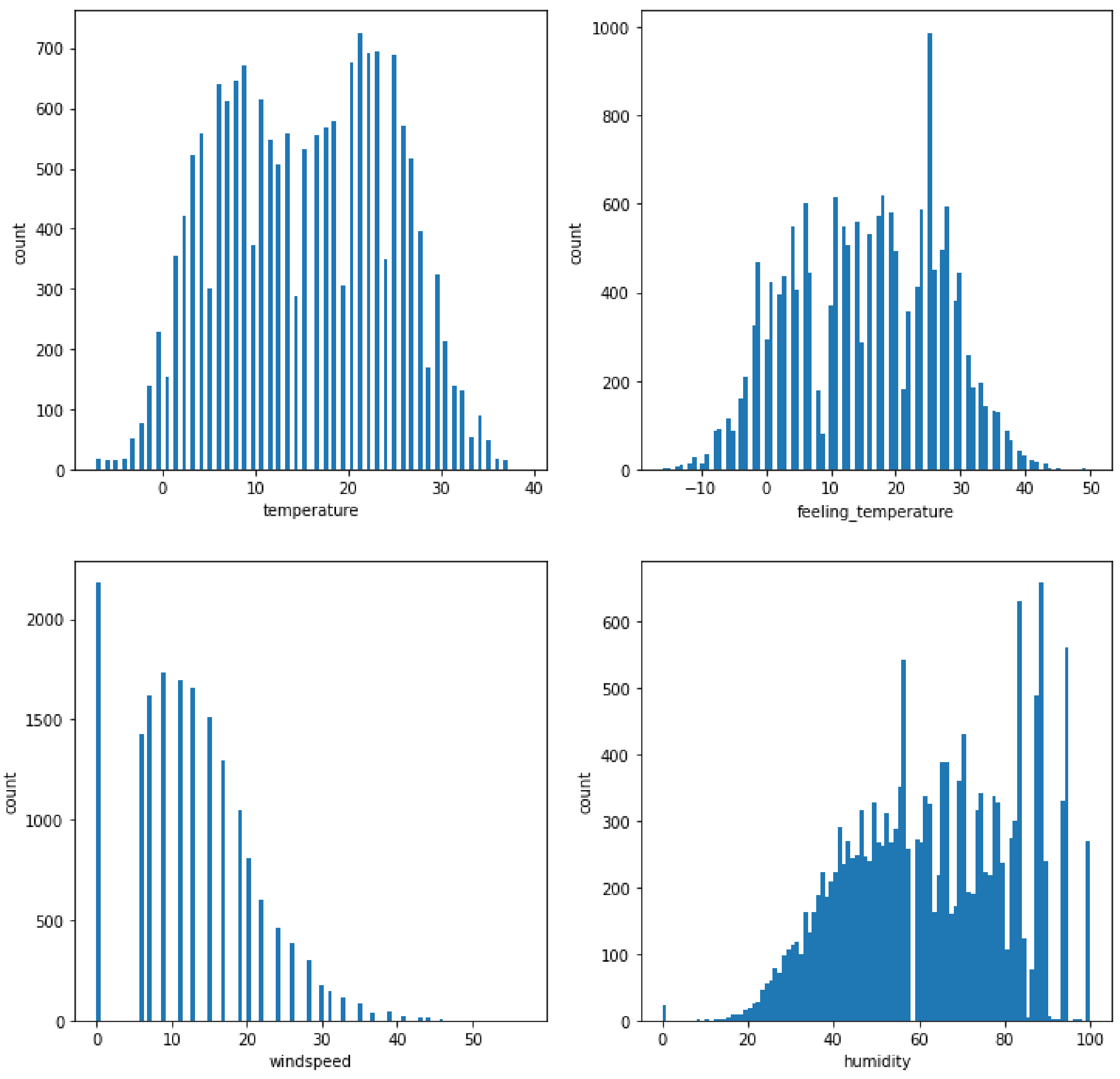


Fig. 3: Weather components distribution

(**Return to Data Set section.**)

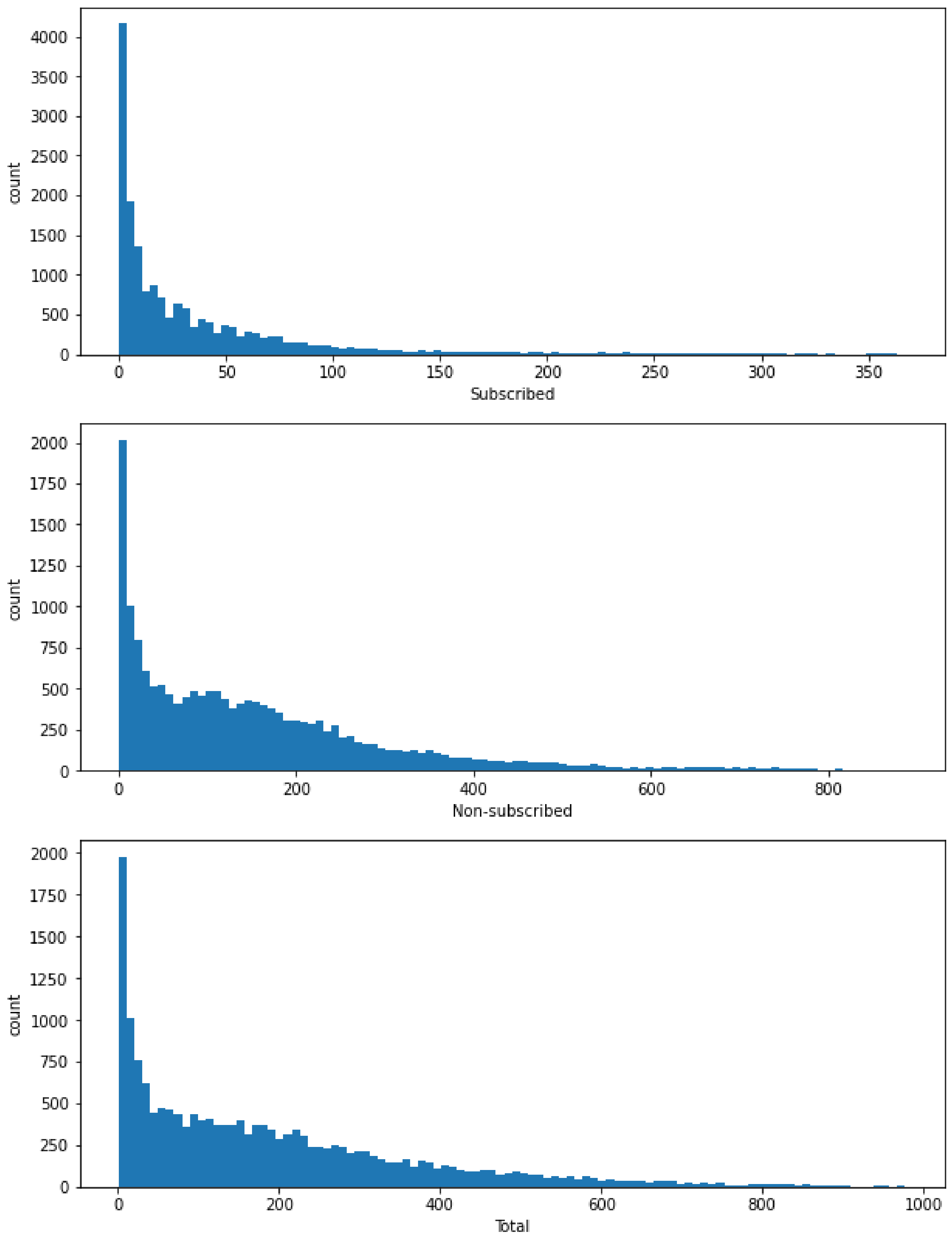


Fig. 4: Rental numbers distribution

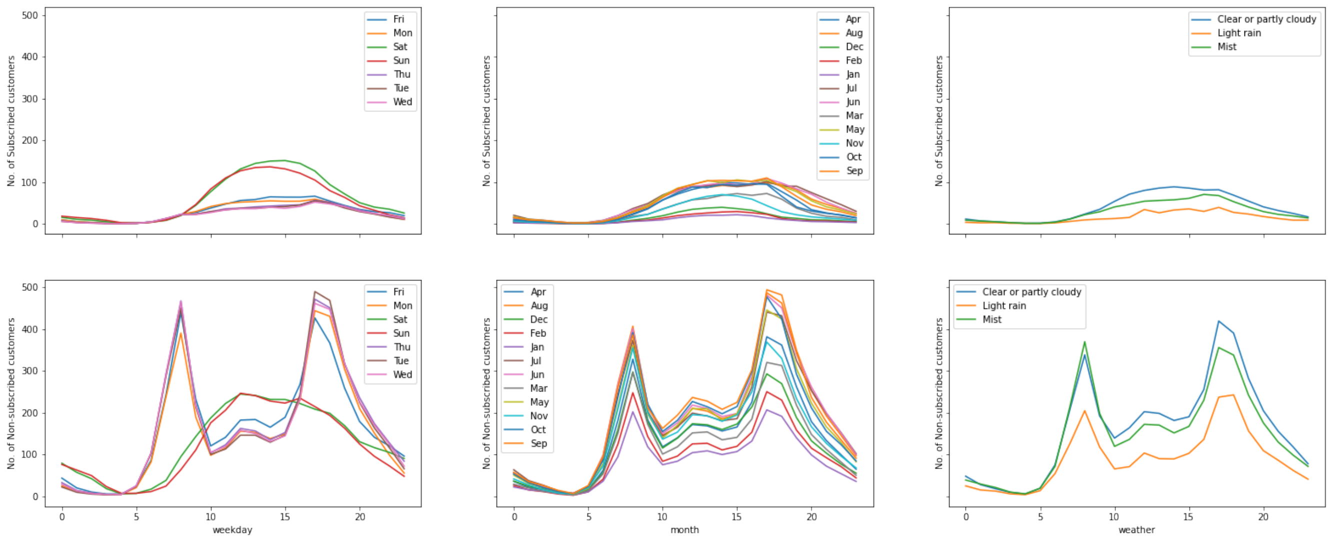


Fig. 5: Renting behaviors across time.

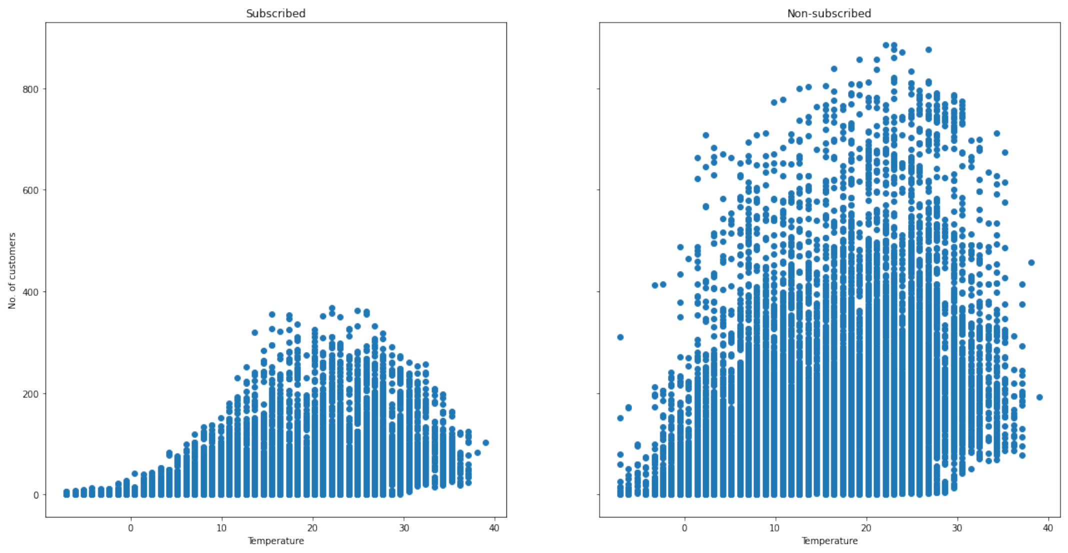


Fig. 6: Effect of temperature in bike rentals.

(**Return to Data Set section.**)

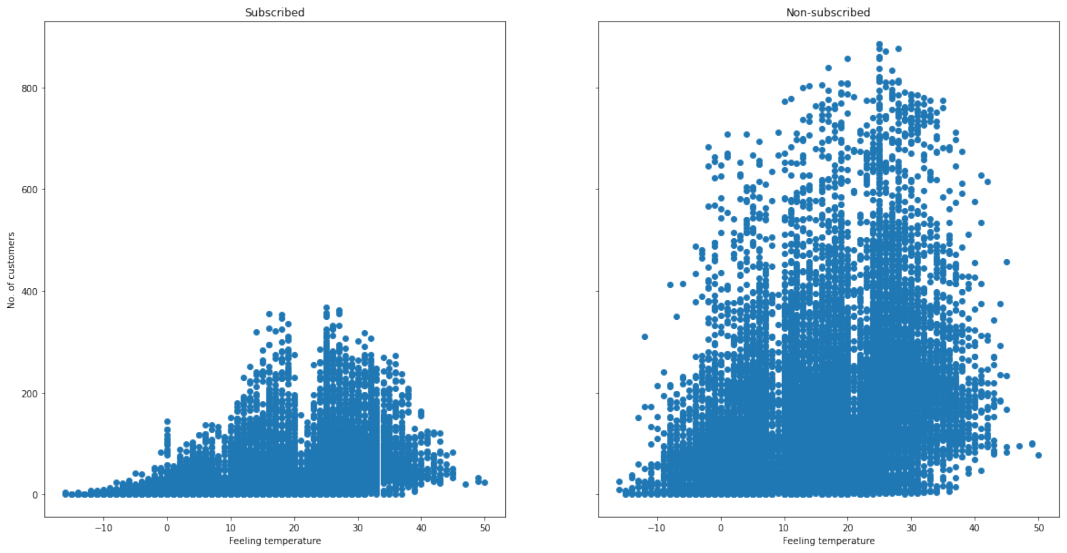


Fig. 7: Effect of feeling temperature in bike rentals.

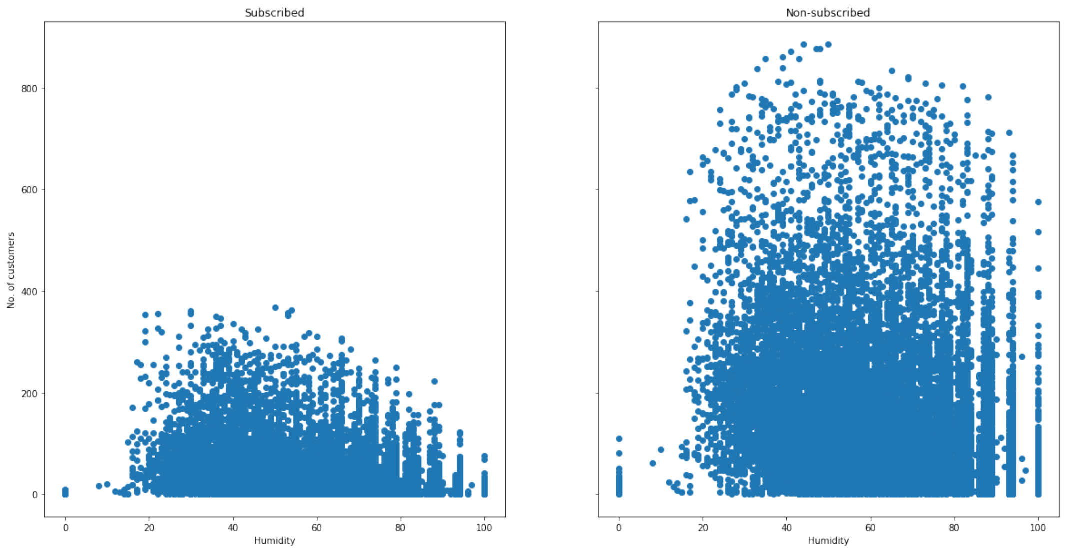


Fig. 8: Effect of humidity in bike rentals.

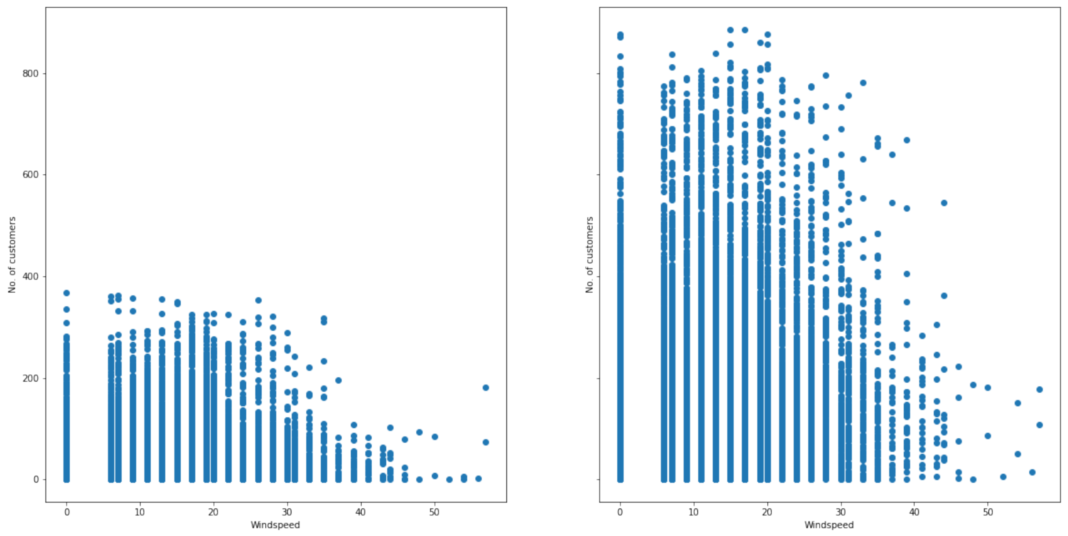


Fig. 9: Effect of windspeed in bike rentals.

B Figures: Linear regressor

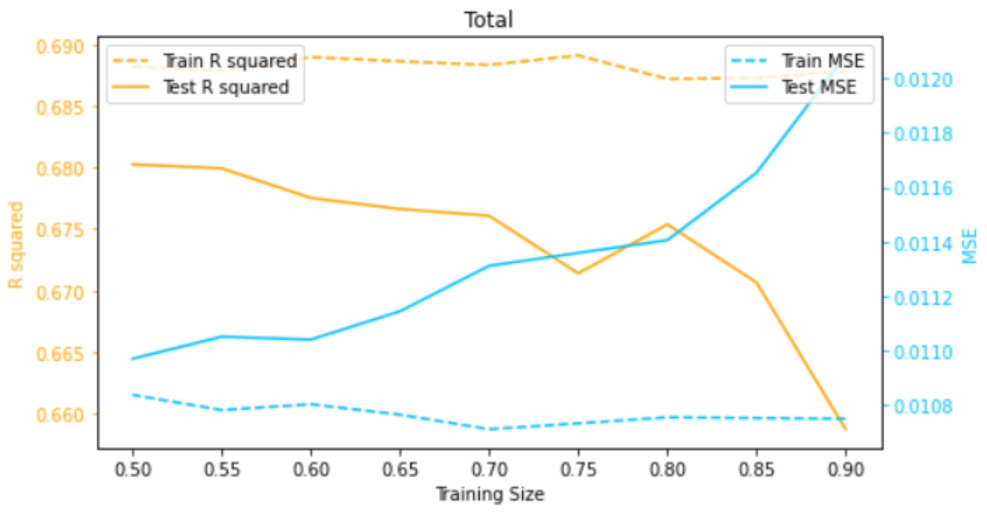


Fig. 10: R-Squared and MSE for Total

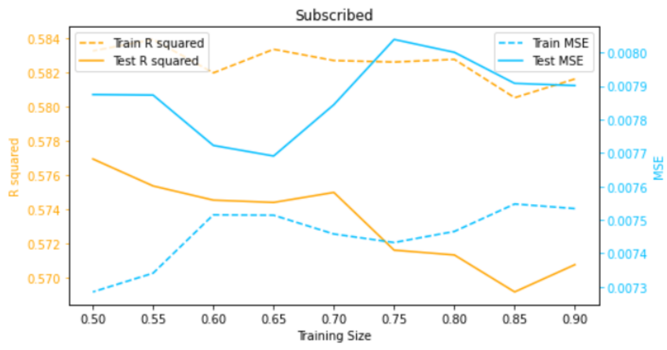


Fig. 11: R-Squared and MSE for Subscribed

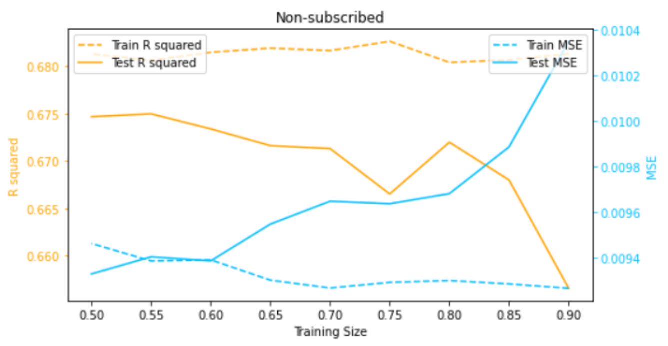


Fig. 12: R-Squared and MSE for Non-Subscribed

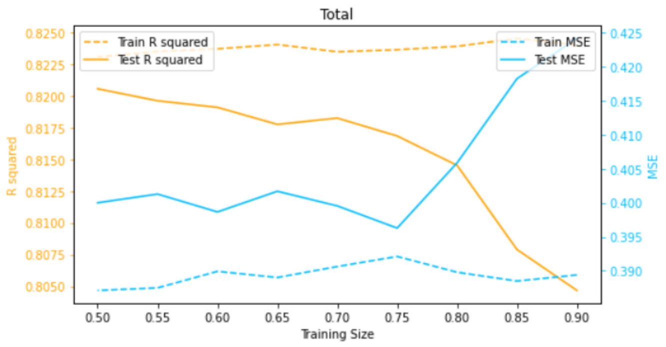


Fig. 13: Using logarithmic transformation

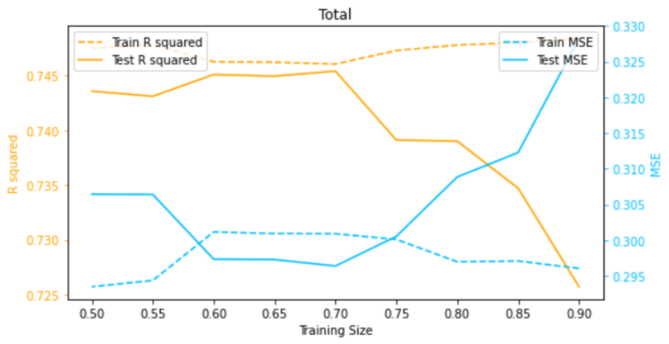


Fig. 14: Using quantile transformation

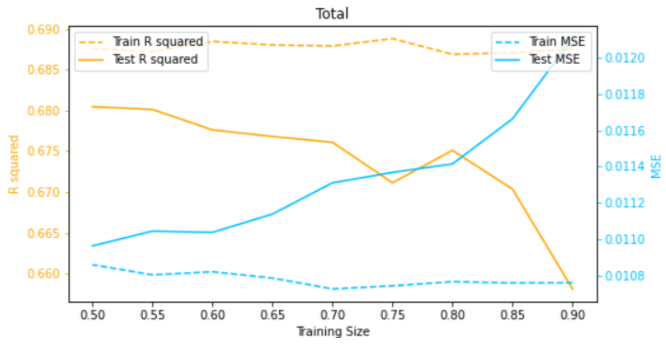


Fig. 15: temperature as a feature

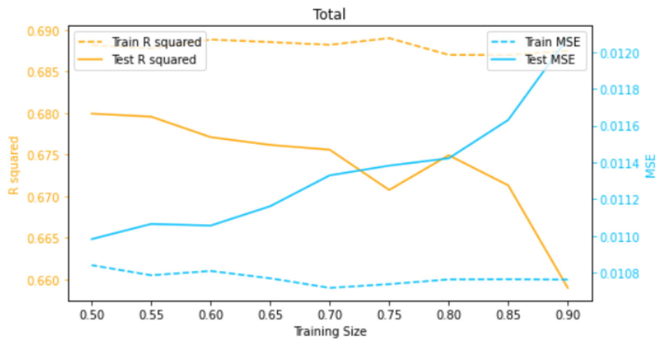


Fig. 16: feeling temperature as a feature

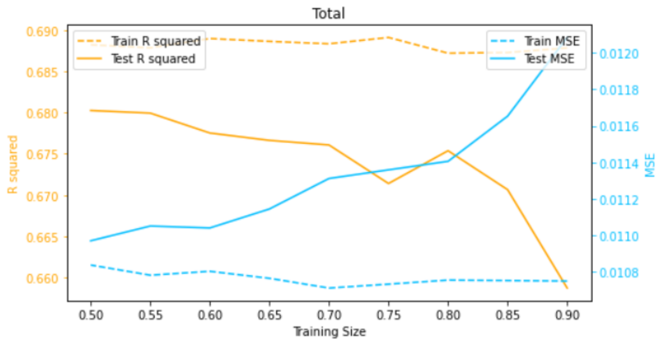


Fig. 17: humidity as a feature

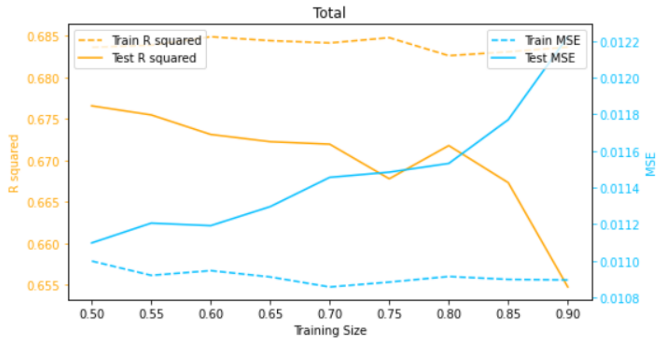


Fig. 18: humidity not as a feature

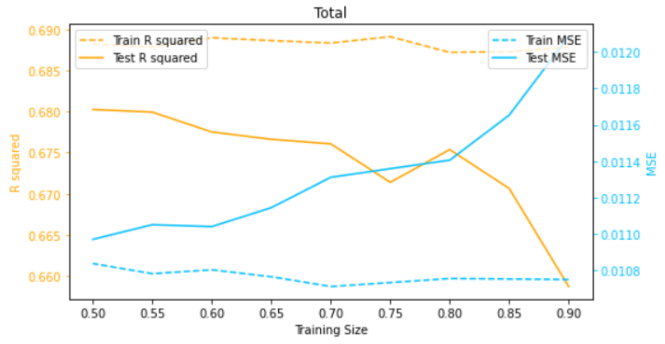


Fig. 19: 24 hour features

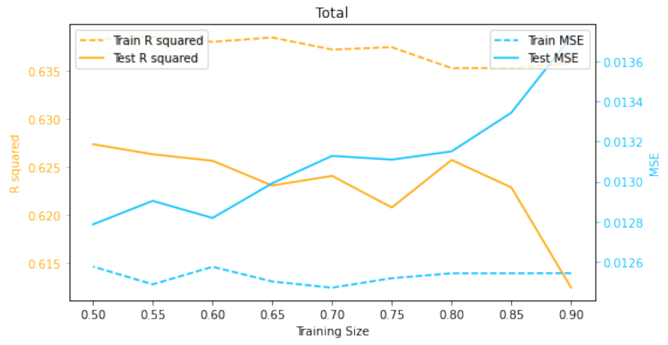


Fig. 20: Group hours by 2 hour range

(Return to Linear Regressor section.)

C Figures: Decision Tree Regressor

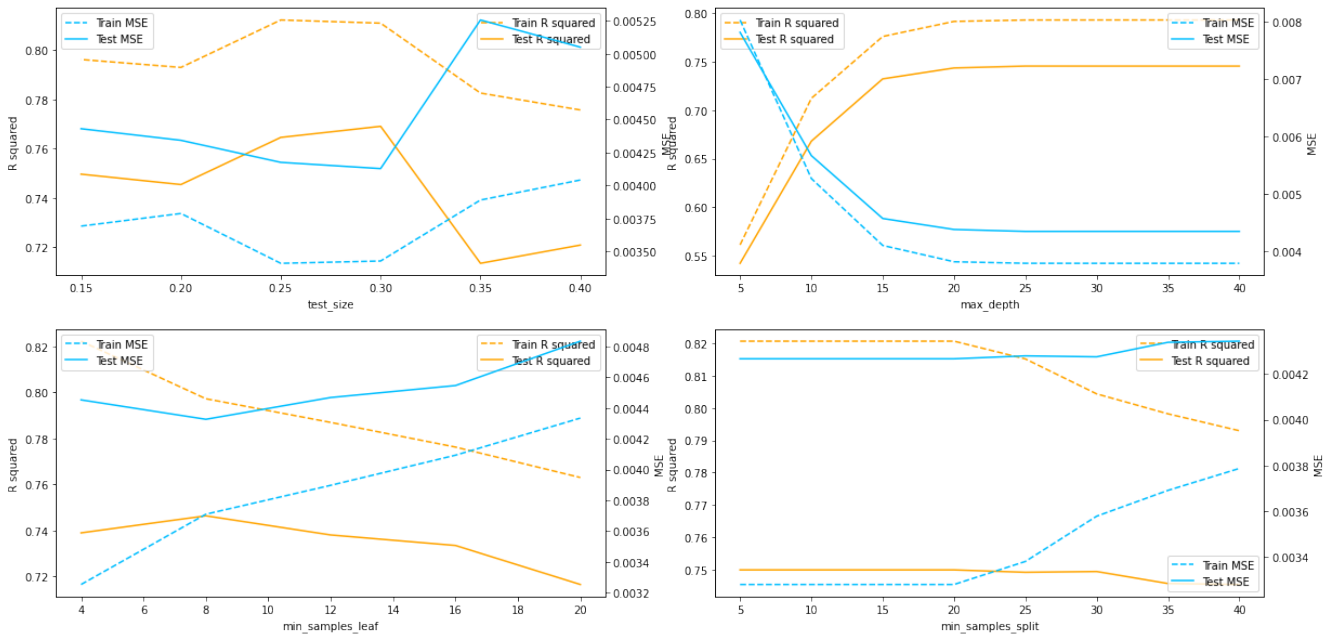


Fig. 21: Decision Tree hyperparameter tuning

(Return to Decision Tree Regression Model section.)

D Figures: Outliers Detection

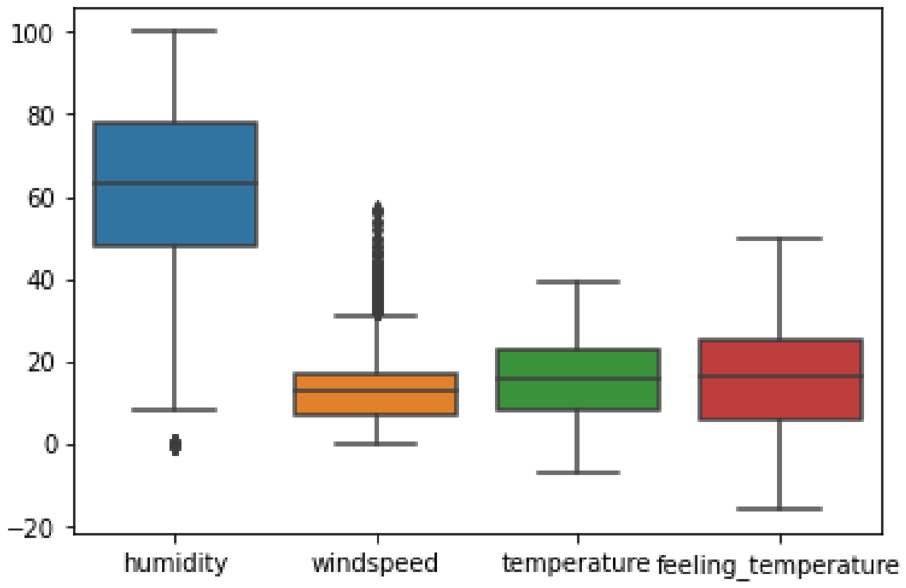


Fig. 22: Outlier Detection using box plot

(Return to Outlier Detection section.)

E Figures: Dimensionality reduction with PCA

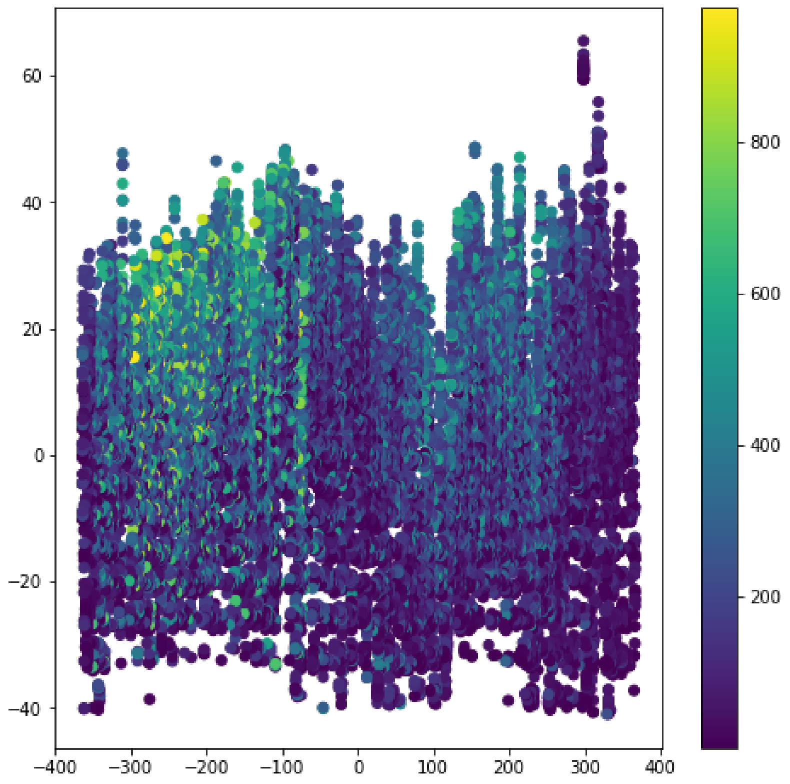


Fig. 23: Plot of PCA with all features of the dataset. The axis x and y present the new dimensions created by the algorithm and the color of the data points is the total number of rentals.

(**Return to Dimensionality reduction with PCA section.**)

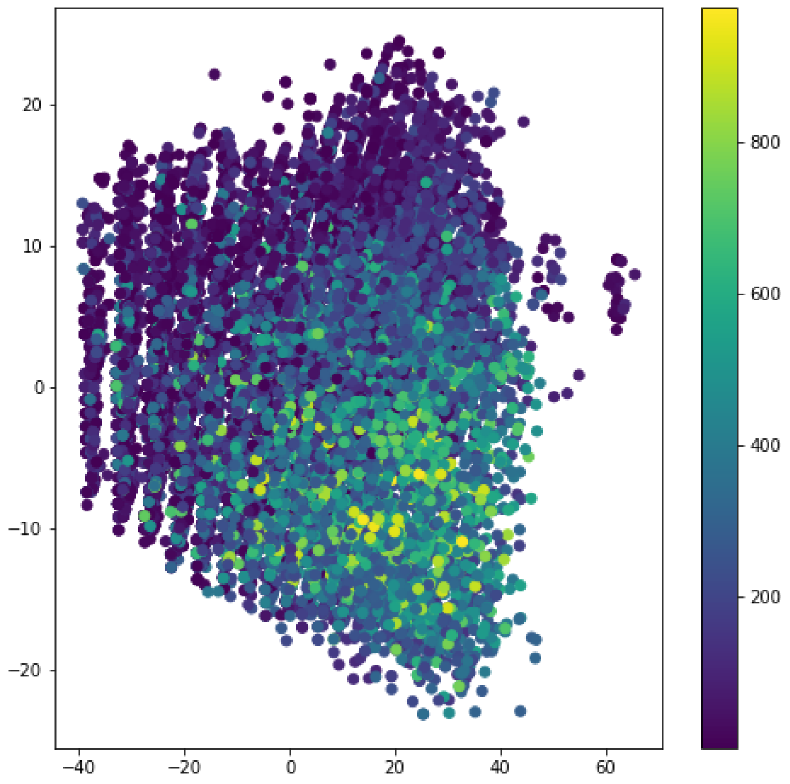


Fig. 24: Plot of PCA on weather features of the dataset. The axis x and y present the new dimensions created by the algorithm and the color of the data points is the total number of rentals.

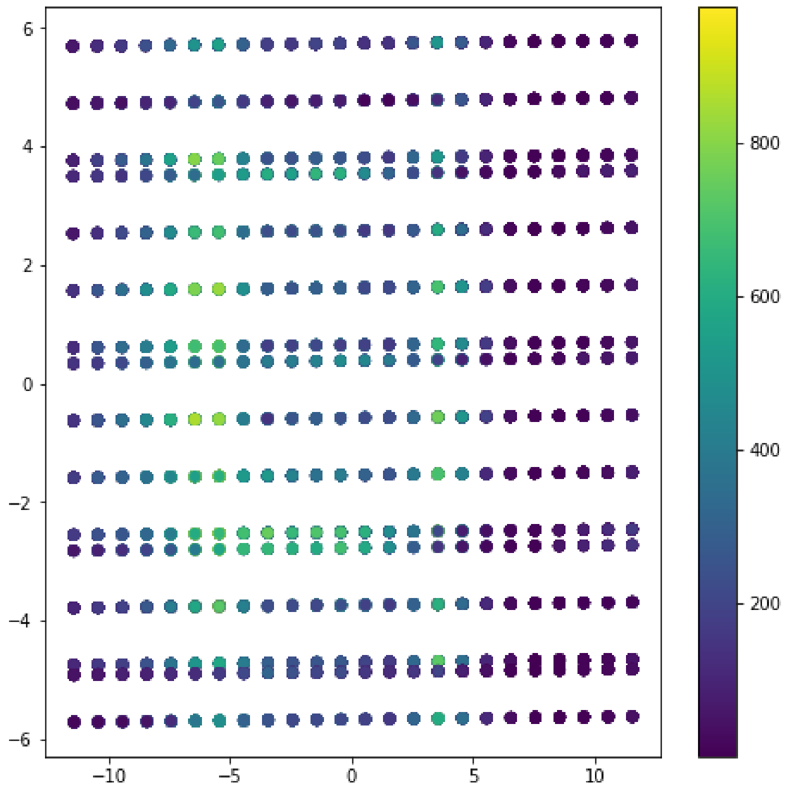


Fig. 25: Plot of PCA on calendar features of the dataset. The axis x and y present the new dimensions created by the algorithm and the color of the data points is the total number of rentals.

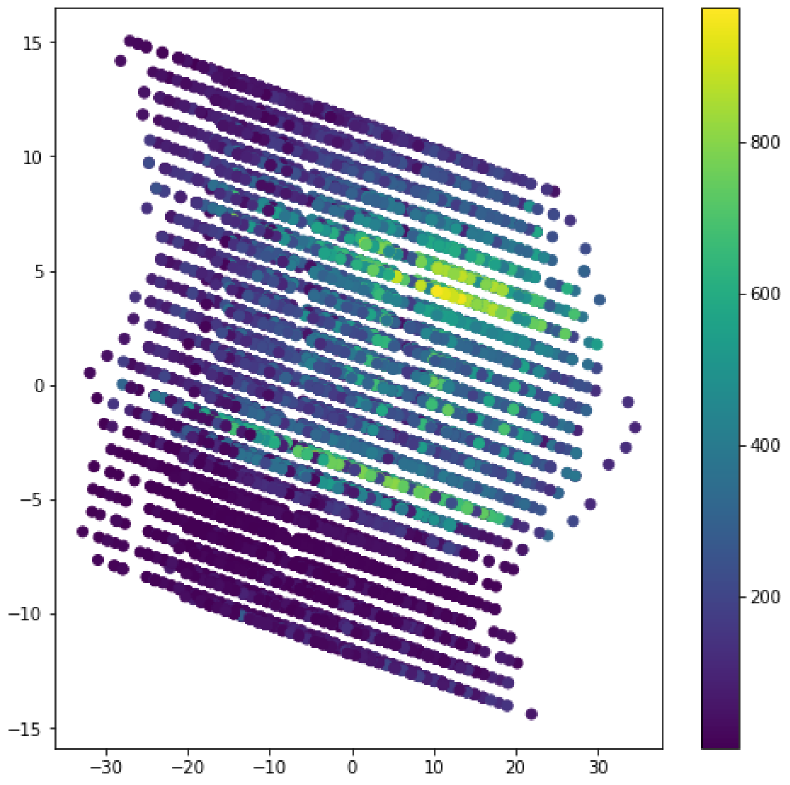


Fig. 26: Plot of PCA on some features of the dataset, achieving the best configuration. The axis x and y present the new dimensions created by the algorithm and the color of the data points is the total number of rentals.

F Figures: Hierarchical clustering and PCA

(Return to Hierarchical clustering and PCA.)

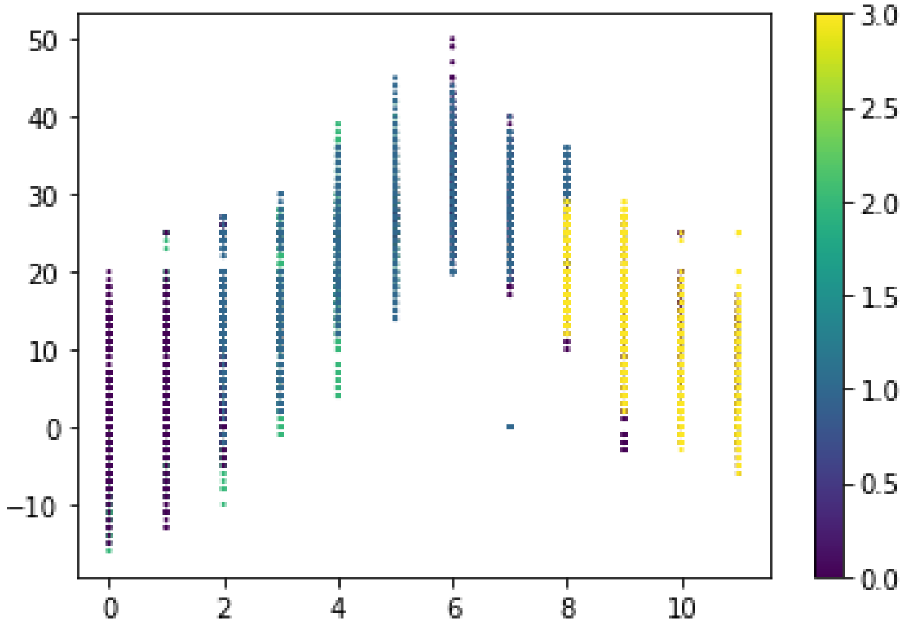


Fig. 27: Plot of the labels generated by the clustering algorithm respecting month and feeling temperature axis.

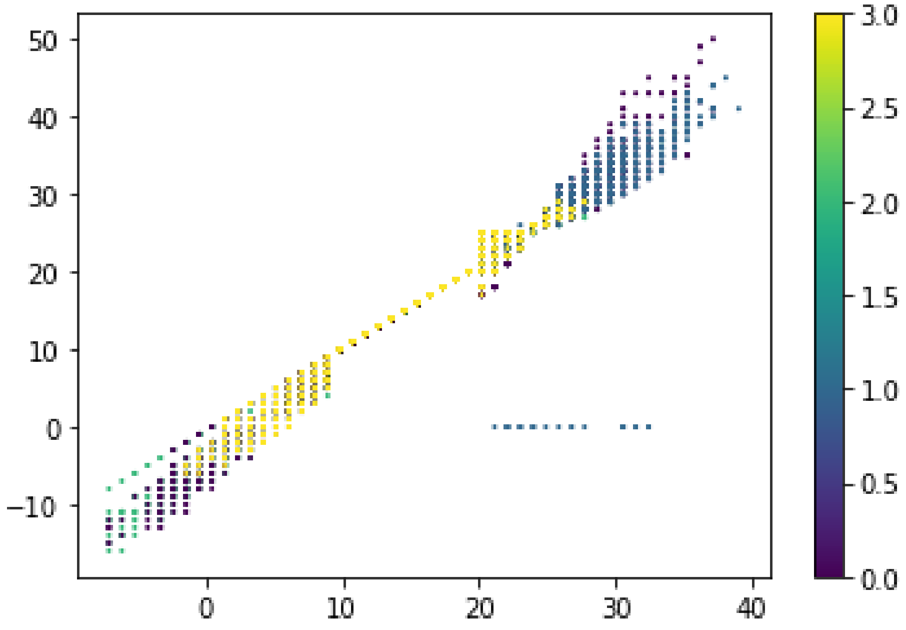


Fig. 28: Plot of the labels generated by the clustering algorithm respecting temperature and feeling temperature axis.

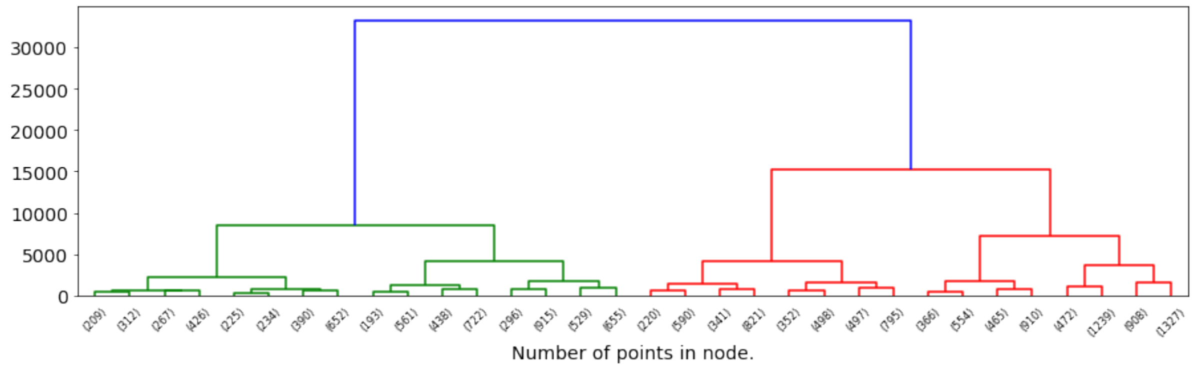


Fig. 29: Hierarchical Clustering Dendrogram on the data

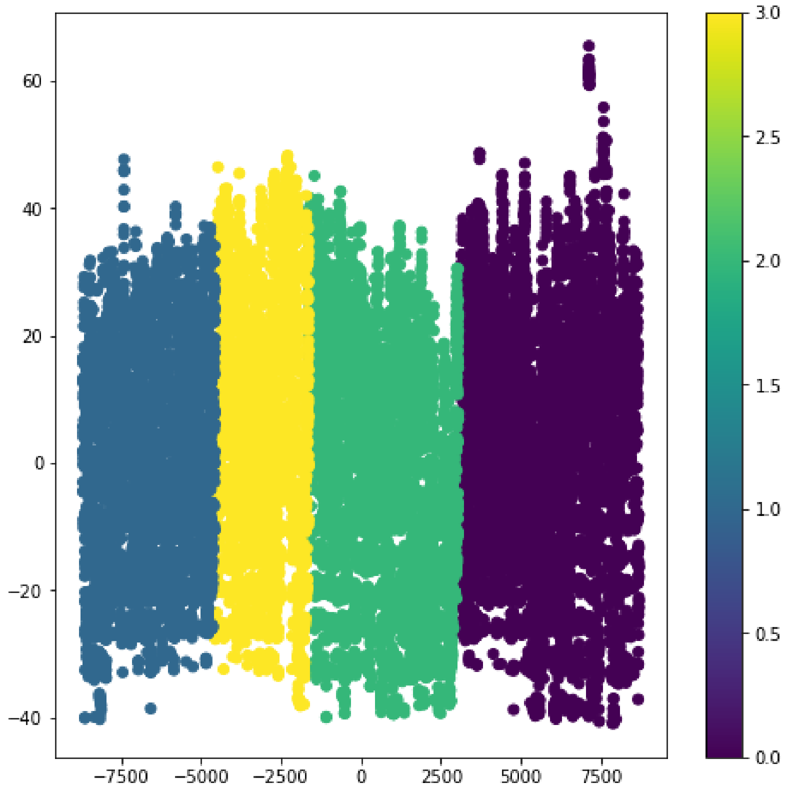


Fig. 30: Plot of applying PCA to the data and then clustering with n=4.

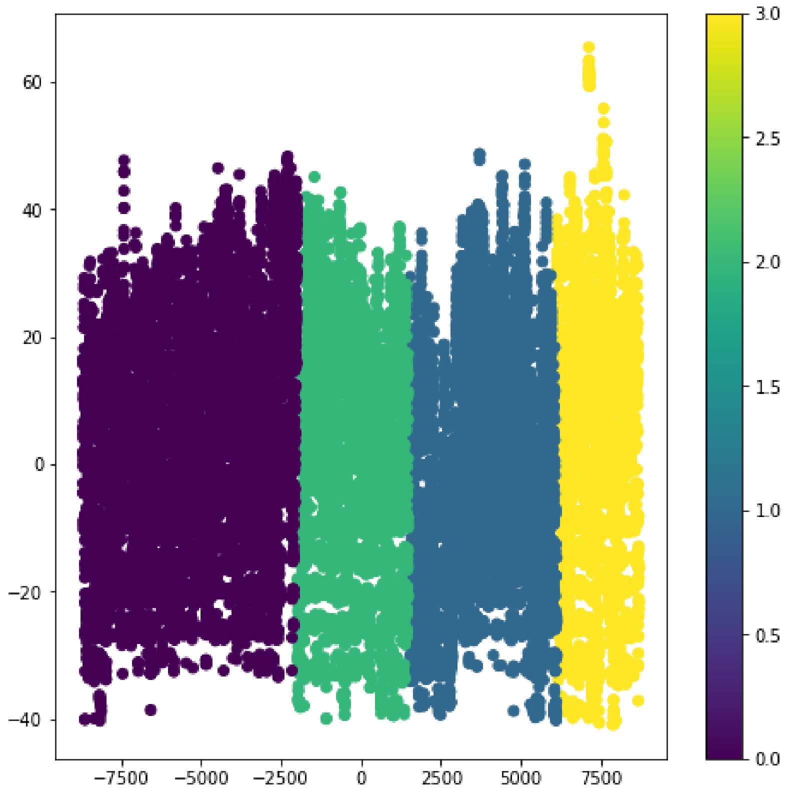


Fig. 31: Plot of clustering with n=4 and then applying PCA to the data.

References

1. *https://www.pluralsight.com/guides/cleaning-up-data-from-outliers*
2. *https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/*
3. *https://analyticsindiamag.com/primer-ensemble-learning-bagging-boosting/*
4. *https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74*
5. *https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/*
6. *https://medium.com/@ravitejareddy05yeruva/support-vector-machine-comparison-of-simple-and-kernel-for-two-data-sets-5bd4c6132b7*
7. *https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710*
8. *https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html*