Taxi capstone project

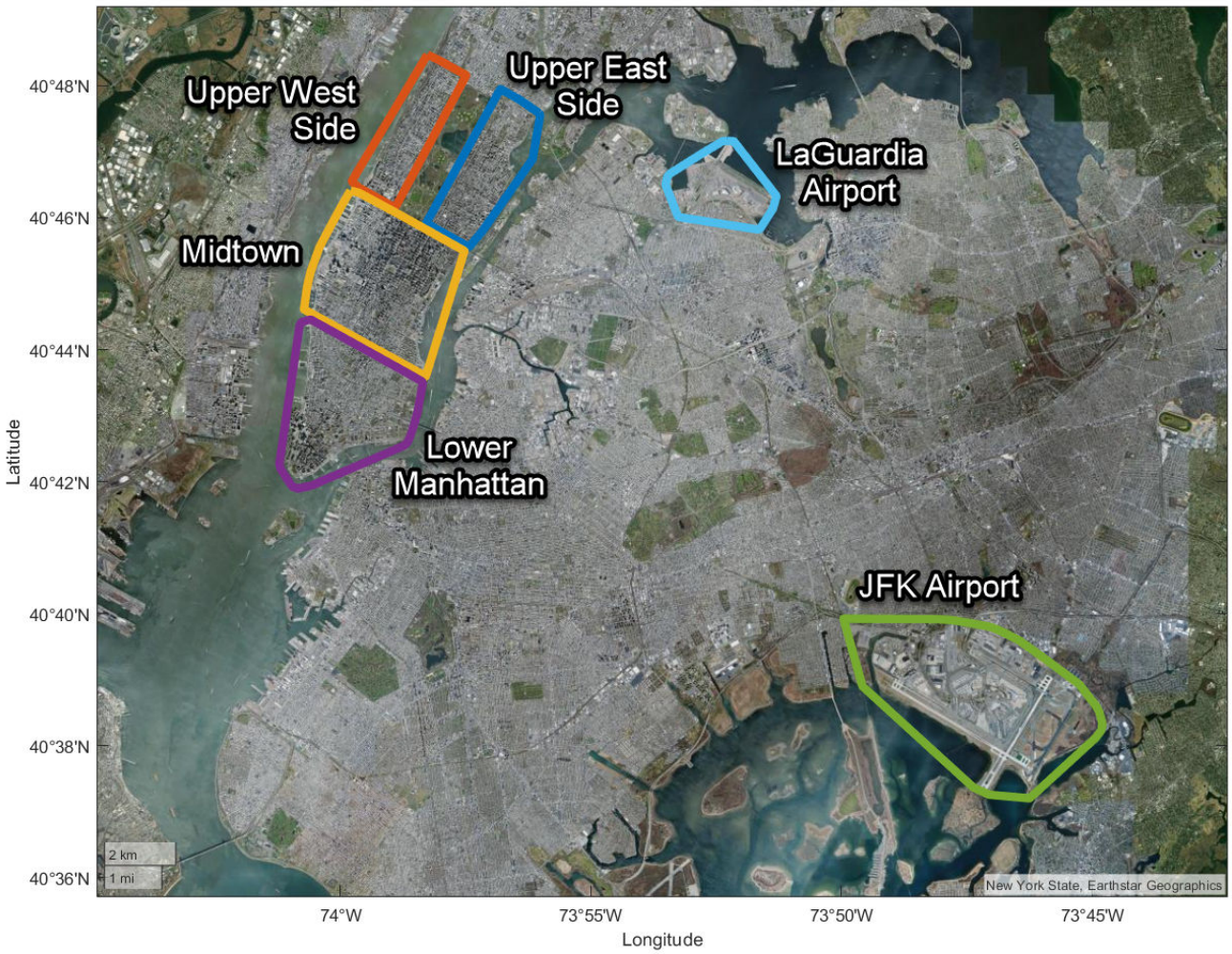
Table of Contents

Objectives

Create a new model that predicts taxi demand around Manhattan and the airports. This work will first be reviewed by other data science team members to ensure the model is accurate and likely to generalize. Present work will then present to Mr. Walker (petitioner) and to internal business team. Create a compelling final report detailing your findings to get the model implemented.

Mr. Walker has asked that your new analysis do the following:

- Focus on four custom regions in Manhattan and the airports (see map below).
- Predict if demand for taxis is “low,” “medium,” or “high.”
- Explore revenue (Fare) and cost (Duration and Distance) of each region to see if certain regions of the city are more profitable.



A model that successfully predicts demand will enable Super Taxis to focus on high demand regions and avoid low demand ones, thus improving efficiency and increasing profits.

Data

Brief description

Collected data refer to taxi recording from all 2015. A lot details, have been provided. Below is shown a list of all attributes for each record will be provided. Detailed description can be found later.

- Vendor
- PickupTime
- DropoffTime
- Passengers
- Distance
- PickupLon
- PickupLat
- RateCode
- HeldFlag
- DropoffLon

- DropoffLat
- PayType
- Fare
- ExtraCharge
- Tax
- Tip
- Tolls
- ImpSurcharge
- TotalCharge
- tzPickupBorough
- tzDropoffBorough
- PickupZone
- DropoffZone
- PickupRegion
- DropoffRegion

It is not possible provide an exaustive analysis of all attributes and possible combination in this report. Just to start to visualise some summaries and understand which kind of data we are looking for, tables with median trip distance, fare and passengers will be provided (group function has been applied on Pickup region).

	PickupRegion	GroupCount	median_Passengers
1	Lower Manhattan	580591	1
2	Midtown	1296074	1
3	Upper East Side	422235	1
4	Upper West Side	265523	1
5	JKF Airport	62178	1
6	LaGuardia Airport	70720	1
7	<undefined>	224945	1

	PickupRegion	GroupCount	median_Distance
1	Upper East Side	422235	1.5000
2	Midtown	1296074	1.5400
3	Upper West Side	265523	1.6900
4	<undefined>	224945	1.9000
5	Lower Manhattan	580591	2
6	LaGuardia Airport	70720	9.7300
7	JKF Airport	62178	17.5300

	PickupRegion	GroupCount	median_Fare
1	Upper East Side	422235	8.5000
2	Midtown	1296074	9
3	Upper West Side	265523	9
4	Lower Manhattan	580591	10
5	<undefined>	224945	10
6	LaGuardia Airport	70720	31.5000
7	JKF Airport	62178	52

Data cleaning

Not all data and features has valid and significative data, in this section whole dataset will be cleaned through following criteria:

- Pickup and dropoff region (only area of interest will be picked);
- Geoscatter plot to analyse if there are some latitude or longitude to exclude;
- Distance
- Fare
- Passengers

Pickup and dropoff region

First of all, since during this analysis, there are only some area of interest, undesired region has been removed. In particular through following code removes what has benn preiously printed as <*undefined*>.

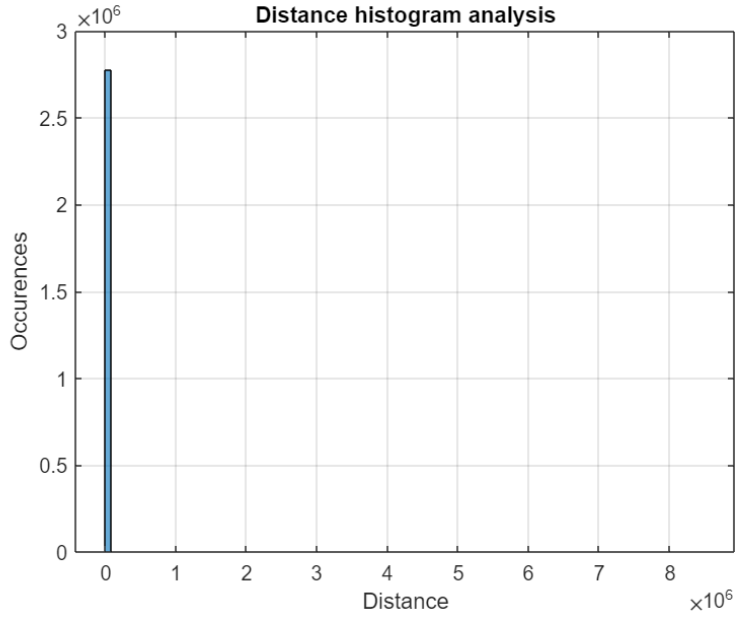
```
taxiData_min = taxiData(~(isundefined(taxiData.PickupRegion) | isundefined(taxiData.DropoffRegion)),:);
```

Distance

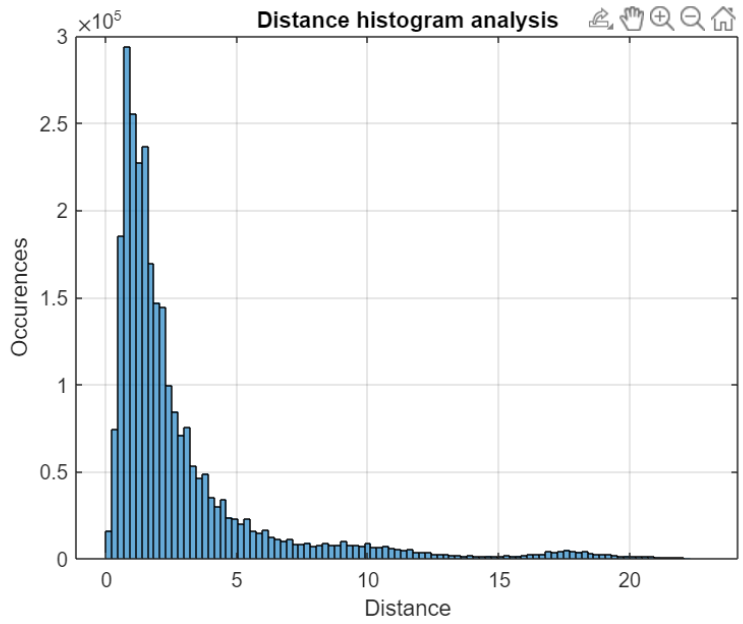
In addition to regions, information about distances could be corrupted so an in-depth analysis on this attribute has been provided.

First of all, all distances declared less than zero have been removed, then histogram analysis underline that there are records with for sure mistakes that not correspond to reality.

```
taxiData_min = taxiData_min(taxiData_min.Distance > 0,:);
```



For this reason, looking at map and calculating possible distances a range between (0, 25] miles has been taken into account. Following figure shows the result.



Passengers

Also for passengers attribute has been adopted the same idea of distance and all recors with *zero* passengers have been removed.

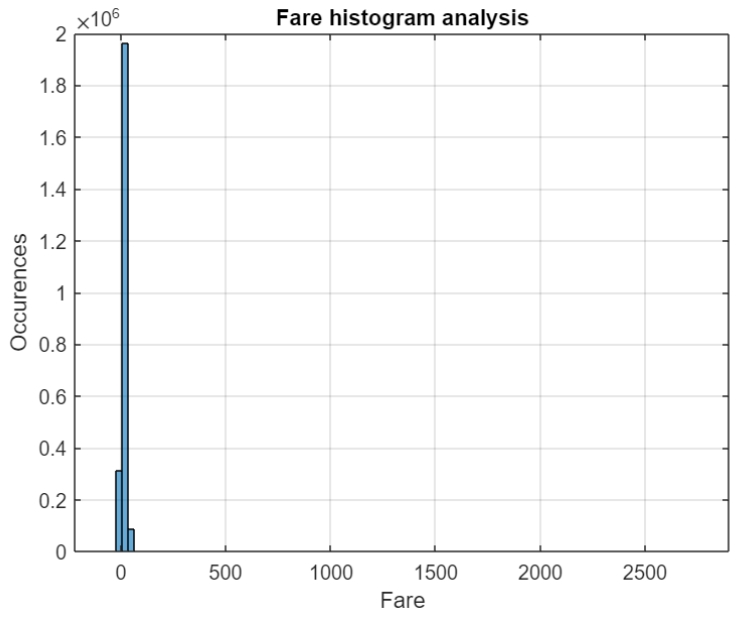
```
taxiData_min = taxiData_min(taxiData_min.Passengers > 0,:);
```

Fare

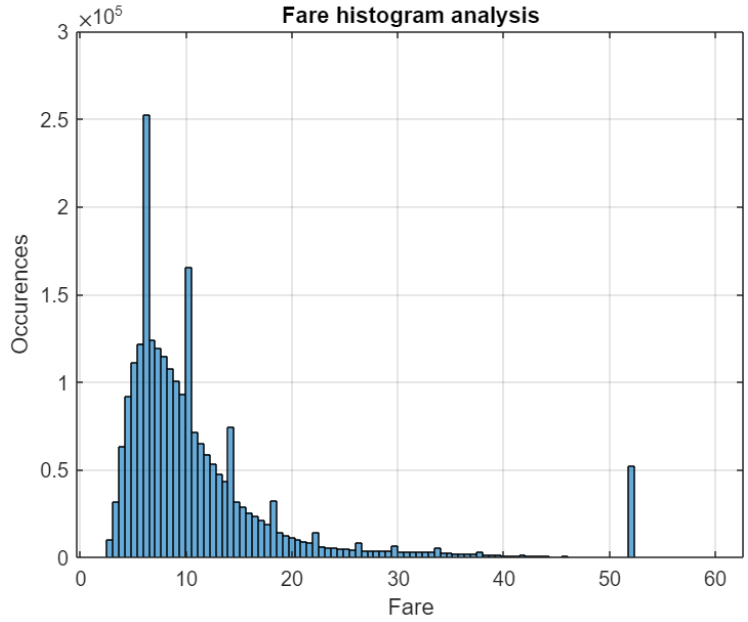
Minimum valid fare is 2.5$. This information is used to filter also fare attribute

```
taxiData_min = taxiData_min(taxiData_min.Fare > 2.5 & taxiData_min.Fare < 60, :);
```

Moreover also in this case, looking at following histogram plot, is visible that there are uncongruent values that have been removed considering the range (2.5, 60) $ that are maybe also too conservative.

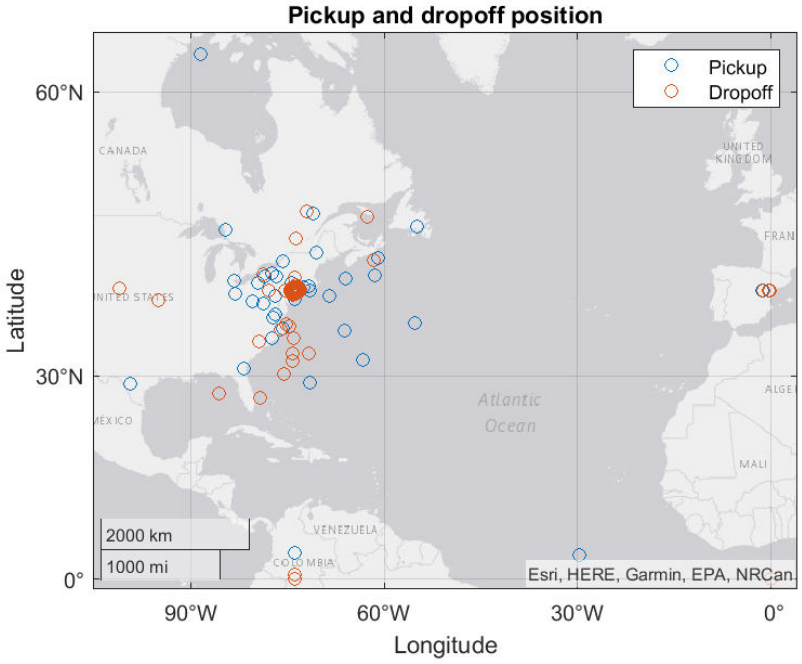


After cleaning, fare histogram has been the following.

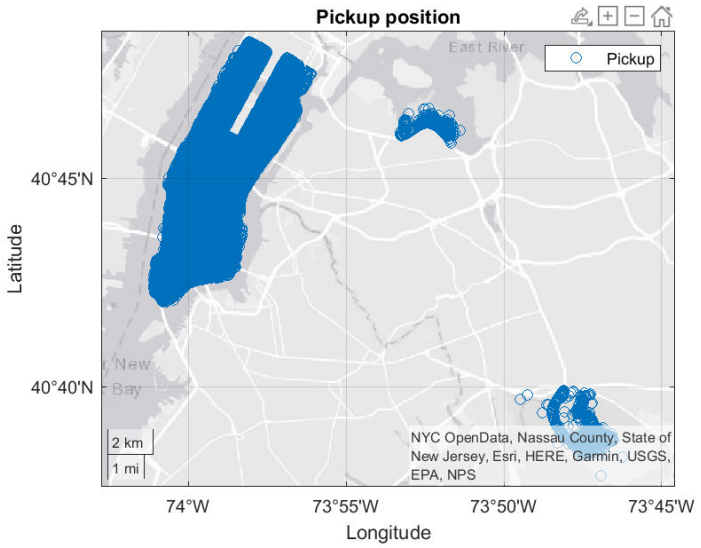


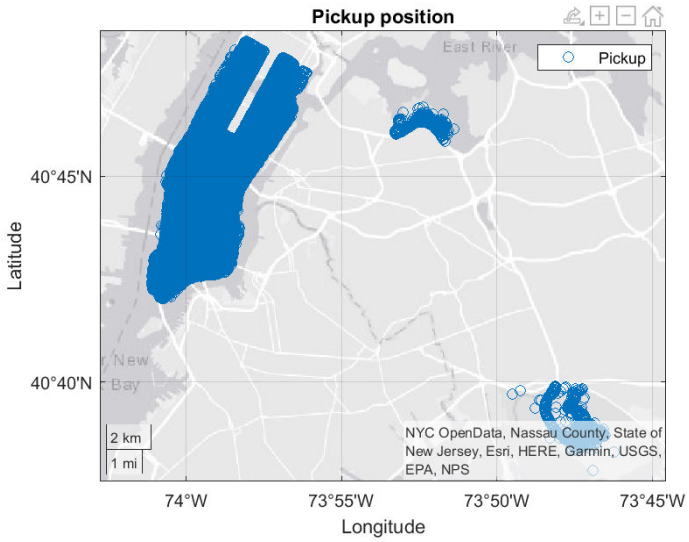
Geoscatter plot

Oother features that coud be possibly corrupted are latitude and longitude. Following figure shows a *geoscatter plot* with pickup and dropoff position.



It is clear that also these feature have data that could not be considered since there are evident undesired location. Also in this case, considering a possible range of latitude and longitude, data have been filtered, obtaining following result.





Is visible how after all cleaning procedure, records correspond only to those with desired area definition.

Data reconstructing

After data cleaning obviously a percentages od original data has been removed but from now on, looking at results, is possible to start a better analysis, looking only at the desired data.

For example, grouping taxi data respect to dropoff and pickup region (respectively), is visible that *Midtown* is the region with higher occurences.

	DropoffRegion	GroupCount
1	Midtown	1166391
2	Lower Manhattan	507529
3	Upper East Side	399952
4	Upper West Side	230766
5	LaGuardia Airport	33128
6	JKF Airport	21675

	PickupRegion	GroupCount
1	Midtown	1171629
2	Lower Manhattan	502280
3	Upper East Side	377639
4	Upper West Side	226173
5	LaGuardia Airport	48468
6	JKF Airport	33252

To better study and analyse data, records feture has been managed. Code about these process can be found in Grouped summary table appendix section.

What emerged after this management is a summary table with eight colums that contains all previous data but grouped by:

- hour bin
- region

For each hour bin, if exists, a region summary has been calculated and following features have been extracted:

- Average distance
- Average duration
- Average fare
- Pickup count
- Dropoff count
- NetPickups

Some rows of this table are reported in following output (sorted by HourlyBin ascendent).

	1 HourlyBin	2 AvgDistance	3 AvgDuration	4 AvgFare	5 PickupCount	6 DropoffCount	7 NetPickups	8 Region
1	2015-01-01 00:00:00	2.6119	00:13:59	11.8564	91	67	24	"Lower Manhattan"
2	2015-01-01 00:00:00	2.6251	00:33:38	11.8634	45	33	12	"Upper West Side"
3	2015-01-01 00:00:00	2.6727	00:13:06	11.6585	57	52	5	"Upper East Side"
4	2015-01-01 00:00:00	2.2634	00:19:38	11.4664	190	135	55	"Midtown"
5	2015-01-01 01:00:00	2.9714	00:15:38	13.4049	89	93	-4	"Lower Manhattan"
6	2015-01-01 01:00:00	2.4310	00:12:50	11.1163	55	54	1	"Upper West Side"
7	2015-01-01 01:00:00	2.5389	00:12:11	11.0878	78	77	1	"Upper East Side"
8	2015-01-01 01:00:00	2.3539	00:13:46	11.3667	161	165	-4	"Midtown"
9	2015-01-01 02:00:00	2.7674	00:13:35	11.9887	85	83	2	"Lower Manhattan"
10	2015-01-01 02:00:00	2.4502	00:11:20	10.6265	29	26	3	"Upper West Side"

Feature engineering

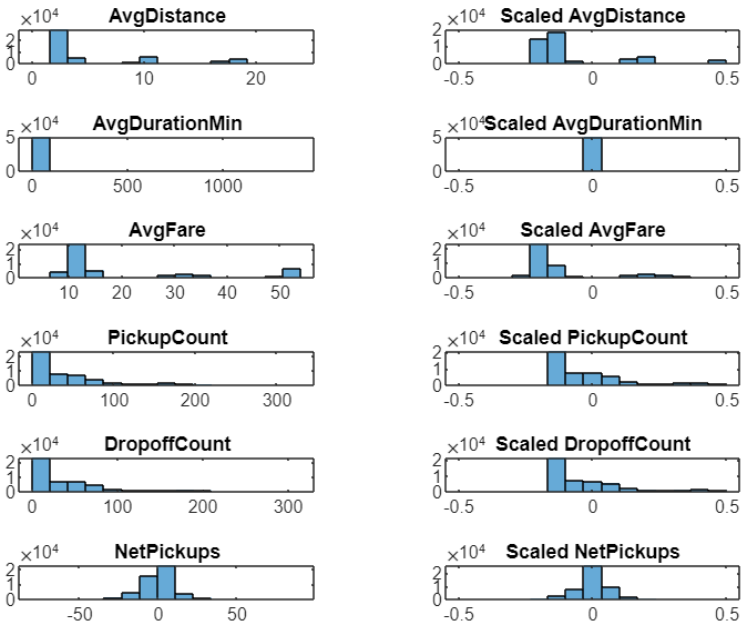
Before training the model is important to understand which features are important and which are redundant or with less information.

First of all, whole vector must be scaled in order to analyse and compare data on the same scale, otherwise conclusion could be not consistent. In order to scale data, following code will be used:

```
% Scaling the features
X = (X-mean(X))./range(X);
```

By this, through mean and range, the vector will be standardised (like min/max function in Python).

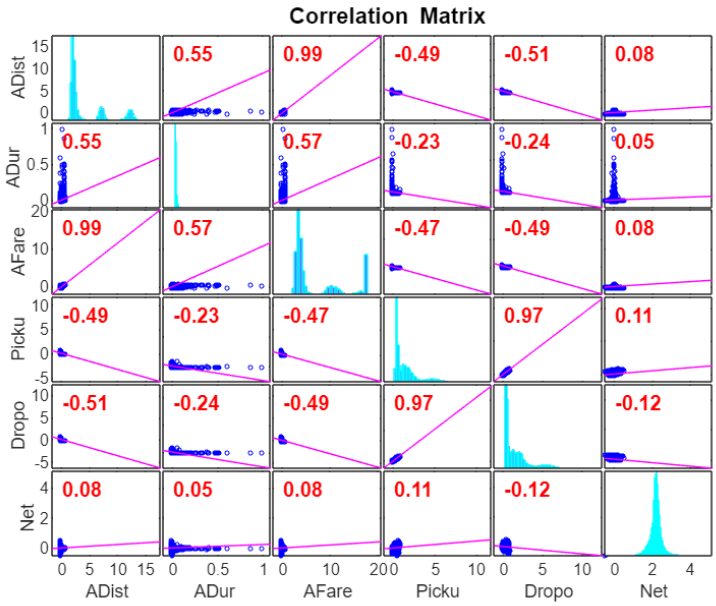
In this case the whole dataset has been scaled just for evaluation, in following section (*Data split*) **before** scaling, dataset has been splitted because otherwise train subset will be "corrupted" by the test set.



Previous figure shows a comparison between original and scaled feature. The transformation used above also centers the features so that they all have a mean value of 0. While mean-centering is not necessary for the purposes of variance thresholding, it does make the distributions easier to compare visually.

Correlation and p-value analysis

One possible analysis that can be computed during feature engineerig process is the correlation analysis. There is no only one type of correlation, in this case, both Pearson and Spearman correlation (non linear correspondence) has been used in addition with *Flag* parameter for testing whether correlations are significant. Results for linear method (Pearson) will be shown in following figure.



What emerge from this analysis is that there is an evident high correlation between distance and fare features and also between pickups and dropoffs, and this was to be expected, while there is no significant information about other feature correalation, also p-value table is not so easy to use in order to extrapolate significant information.

		ADist	ADur	AFare	Picku	Dropo	Net
1	ADist	1.0000e+00	0	0	0	0	4.1707e-71
2	ADur	0	1.0000e+00	0	0	0	5.0619e-28
3	AFare	0	0	1.0000e+00	0	0	3.2056e-74
4	Picku	0	0	0	1.0000e+00	0	8.0238e-127
5	Dropo	0	0	0	0	1.0000e+00	1.9369e-163
6	Net	4.1707e-71	5.0619e-28	3.2056e-74	8.0238e-127	1.9369e-163	1.0000e+00

Variance analysis

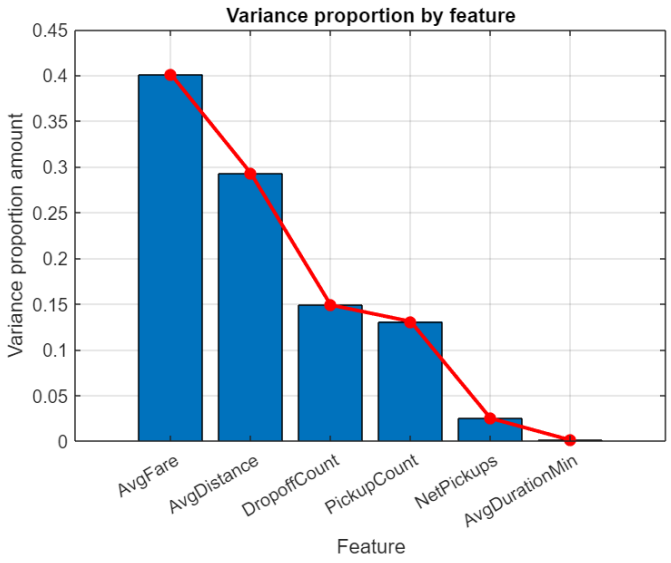
Another of the possible analysis is variace study. In this case, to compute variance has been used *var* function.

Sorting the features by variance makes comparison easier, as does displaying them in a `table`.

		AvgFare	AvgDistance	DropoffCount	PickupCount	NetPickups	AvgDurationMin
1	Variance:	0.0867	0.0634	0.0322	0.0317	0.0054	2.2401e-04

At fist impact is visible how the average distance is the feature with higher variance while average duration is the one with lower variance. This could imply that duration could not have not significative information during prediction but is only a partial conclusion.

While scaled predictor variances can be compared directly, it's also common to convert them into proportions so it's easier to see what proportion of variance you lose by discarding a given feature, or alternatively, how much of the total variance is preserved by the features you retain. Additionally, since proportions are consistent across different datasets, this can help you make consistent choices for your thresholds. For example, you might decide for a particular model to keep all features that account for at least 10% of the variance, or to discard as many low-variance features as possible while making sure that at least 90% of the total variance is retained.



Also with proportion variance comparison, what emerged is that the average duration is the feature that contains less information with a percentages lower that 0.01%.

Relevant exploration result

After previous considerations, what has been decided is to remove average duration from main features that will be used to train the model. This because the information that this feature has is lower respect to the other and moreover could be influenced by external factor as traffic, road work, ecc ecc so it could "not correspond to the reality of taxi trip itself".

Moreover, since distance and fare feature have very high correlation, it means that the amount of information during the training is quite the so it will be considered only one of them.

From variance analysis also net pickups doesn't cointain a high quantity of information like other feature so it will be removed at first.

Feature engineering procedure has thus permitted to obtain the minimum state vector of attributes that will be considered during model train. In particular:

- Region
- Dropoff count
- Pickup count
- Average fare
- Date & time

Additional data that has been provided is the *holiday* table in which are specified all days that are considered as "day off". Using function *ismember,* additional column has been inserted. This value is a boolean parameter and correspond to 1 if the realted day is holiday and 0 otherwise. Also this column will be loaded during model training. Consideration about its importance will be provided later.

Data split

Split whole datset is foundamental in order to create a train and a test set that permits to generalised the model and its behaviour. To do this procedure *cvpartition* function has been used and to avoid mistakes and permits reproducibility the control random number generator parameter (*rng*) has been se tot *default.* By this the starting point will be the same each time.

cvpartition function will be used defining an 80/20 partition: the 80% of the data will be set as training subset while the ramaining 20% corresponds to test subset.

Code about split procedure will be find in Data split appendix section.

```
Dataset dimension: 50843

Train set dimension: 40675

Test set dimension: 10168
```

Modeling

Objectives

While it is possible to use a regression model to predict net pickups, this approach would require model users to interpret predicted values before deciding where to dispatch the taxi fleet. This requires a deeper understanding of the 'typical' response values to determine taxi demand. In order to have more quickly and clearly inform drivers and dispatchers, has been decided to use a classification model (easier to understand respect to regression one).

Response variable creation

Since the scope of this analysis is to obtain the region in which at each time, the company must allocate resources, the idea is to convert net pickups to a categorical feature, 'Demand', and use it as the response variable for the model. To create *Demand*, net pickup values will be change to categorical values according to the following definitions:

- Net Pickups < 0: 'Low'
- 0 <= Net Pickups < 15: 'Medium'
- Net Pickups >= 15: 'High'

Final model description

Model type

Since has been declared that classification model, for this pourpose, is better respect to regression one, Matlab classification learner app has been used in order to perform training task.

Feature imported during new session are those listed before in section *Relevant exploration result* section.

Multiple models have been tested before arriving to the better one that has been obtained as a *Fine tree* classification model with a 5 fold cross hyperparameter validation and a custom misclassification cost matrix. Last model version exported as function can be found in Classification model appendix section.

Hyperparameters

Fine tree automatically hyperparameter is defined as the number of split and the realtive split criterion. In this case following definitions have been set.

Model Hyperparameters

Preset: Fine Tree
Maximum number of splits: 100
Split criterion: Gini's diversity index
Surrogate decision splits: Off

Required prediction features

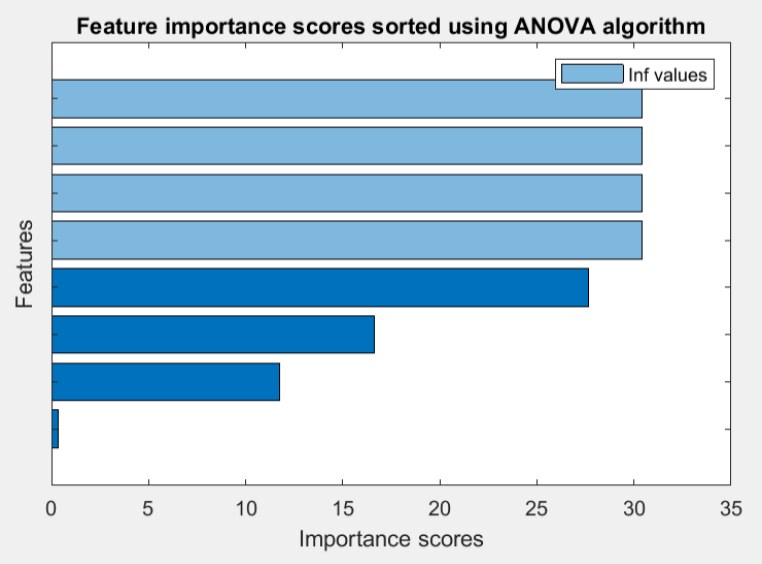
Through the *feature importance selection* box, provided directly inside the learner app, a feature ranking analysis could be provided in order to understand and validate if the choice of selected feature has correct or not.

To underline the usage of the *HourlyBin* column through separate columns about hour, day and month.

Following feature represent the ANOVA feature ranking agorithm application on following features:

- Region
- Dropoff count

- Pickup count
- Average fare
- Hour
- Month
- Is holiday
- Day



Selecting a feature raking boundaries equal to 15, *is holiday* and *day* feature have been removed at first to maintain the minimum number of feature that can identify the highest importance information.

Training description

After features selection directly through classification learner ANOVA ranking, multiple models have been tested, in particular:

- Fine, Medium, Coarse and optimizable tree
- Linear and quadratic discriminant models and
- Gaussian naive bayes models

All these model are at first trained using a **default** misclassification cost matrix, in order to have a sort of baseline for futher analysis.

Some of them (for example the optimizable tree), returned a very high accuracy score on training data (up to 95%) that indicates that probably we are encountering an overfitting problem. This can cause an undesiderable lack of generalization when test data will be used. These models are thus deleted from the possible models to be used.

Considering only the remaining models after this first process, the fine tree resulted as the best model with the higher trade off between training and test accuracy.

Results are reported below.

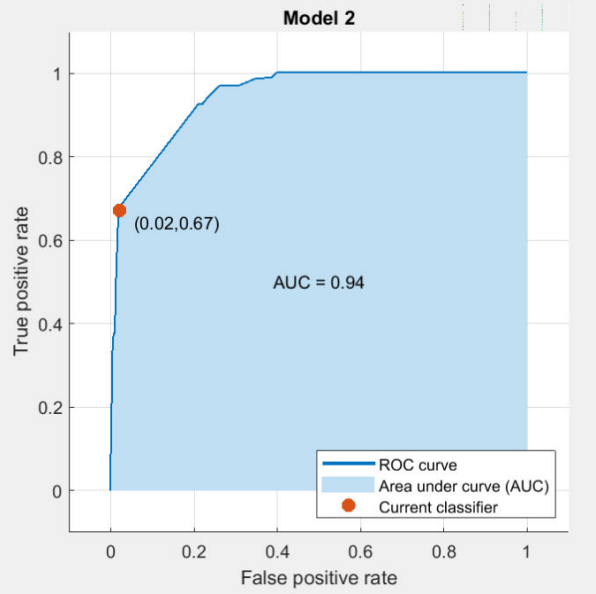
Training Results

Accuracy (Validation)	88.3%
Total cost (Validation)	4757
Prediction speed	~560000 obs/sec
Training time	1.8816 sec

Test Results

Accuracy (Test)	75.2%
Total cost (Test)	2519

Is visible that has been obtained a good accuracy index both for train and test and it seems there is no overfitting. Following figure will show also confusion matrix and ROC curve for test subset.

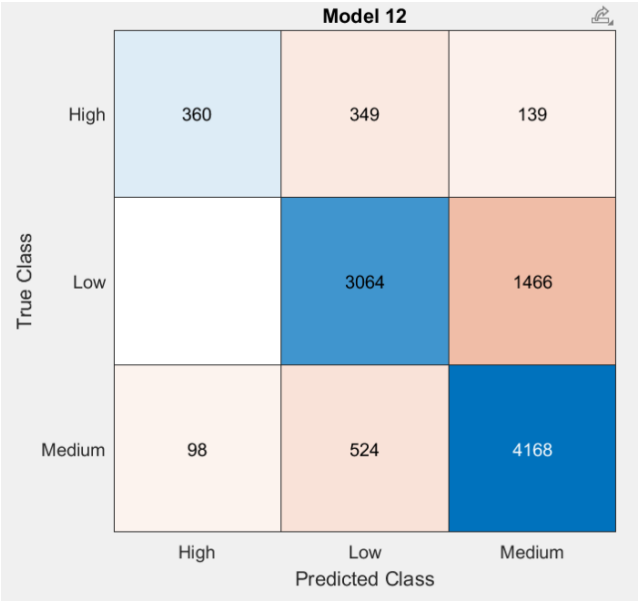


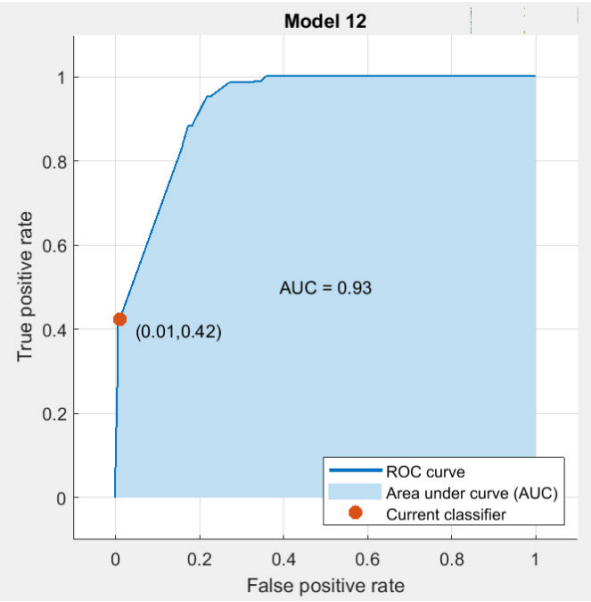
Cost matrix optimization

Looking at the previous confusion matrix and considering the pourpose of this work, a possible idea could be to reduce the number of false positive for *high* demand class in order to minimise the case in which the predicted class is high even if the true class is different. To approach this problem, the costs matrix has been modified and in particular has been set different weight for each couple of true and predicted class as defined in following table.

		Predicted Class		
		High	**Low**	**Medium**
True Class	**High**	0	2	2
	Low	4	0	3
	Medium	4	3	0

Results validate what expected with lower values on false positive responses. In this case false negarive are a little bit worse respect to the previous case with default cost matrix but the whole accuracy on test is higher and is desirable to underline that is is only **one of the possible solution** depending on what the client expects and requeires as final results. The is no only one solution to that problem.





Test metrics

Final resulted model has been used to predict demand feature on test datset and results are the following

Training Results

Accuracy (Validation)	86.6%
Total cost (Validation)	14402
Prediction speed	~590000 obs/sec
Training time	1.9559 sec

Test Results

Accuracy (Test)	74.7%
Total cost (Test)	7338

As metrics to evaluate the goodness of this model, *cmetrics* function has been used. In paticular, functions inputs `ytrue` and `ypred` must be column vectors of equal length while output `t` is a table containing the following metrics for each class:

- Precision
- Recall
- Fallout
- Specificity
- F_1

```
Accuracy = 74.67%
```

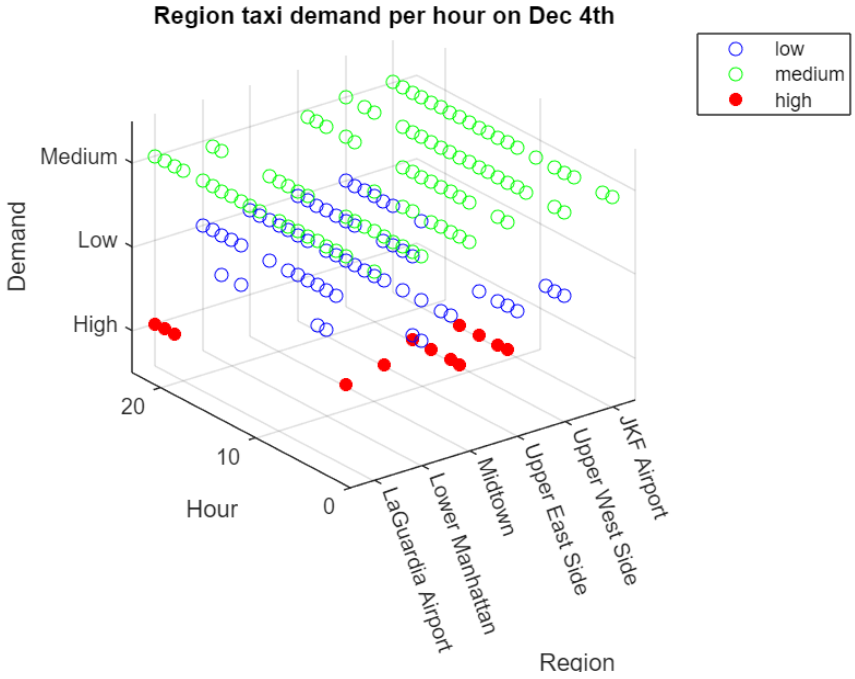
`t = 5×5 table`

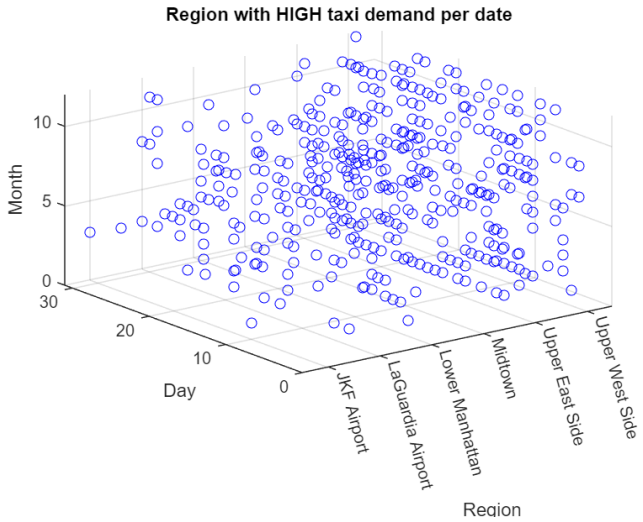
		Precision	Recall	Fallout	Specificity	F1
1	High	7.8603e-01	4.2453e-01	1.0515e-02	9.8948e-01	5.5130e-01
2	Low	7.7826e-01	6.7638e-01	1.5484e-01	8.4516e-01	7.2375e-01
3	Medium	7.2198e-01	8.7015e-01	2.9844e-01	7.0156e-01	7.8917e-01
4	Avg	7.6209e-01	6.5702e-01	1.5460e-01	8.4540e-01	6.8807e-01
5	WgtAvg	7.5239e-01	7.4666e-01	2.1045e-01	7.8955e-01	7.4019e-01

Conclusion

This analysis and this model training permits to obtain a classifier through which it is possible to predict the demand of a specific region in a predefined date and hour, based on taxi data recorded on whole 2015.

Following figures show an example of the model prediction on test dataset.





Is visible how, in this case machine learning could be a valid solution in order to allocate resources in a specif region to maximise profits.

Overall analysis

After a fist cleaning process, data filtered corresponds to the records about the desired areas without corrupted outliers. Moreover features selection and management permits to isolate only mean/average attributes that facilitate the analysis and prediction comprehension.

Model chosen correspond to a classification one because has been considered a better approach respect to a regression model due to an easier result understanding.

Accuracy obtained on test subdatset is 75%. This permits to look at the result with a quite large confidence about the correctness of the prediction. It is however to underline that the solution find is not the unique and could not be the best even if can be considered satisfying.

Overall recommendations

With this material the commiter can predict region with higher demand at a specific date and hour. For example, considering April 10th at 6am:

```
taxiTest_prediction_high_SDate = taxiTest_prediction_high(taxiTest_prediction_high.Day == 10 & ...
    taxiTest_prediction_high.Month == 4 & ...
    taxiTest_prediction_high.Hour == 6,:);
```

Displaying the predicted region, is immediatly visible which reagion has the higher demand for taxi, in this case *Lower Manhattan*.

Final raccomendation is thus to analyse the results and allocate resources in regions in which (with a 75% of probability) the deman will be higher than other zones.

Appendix

Grouped summary table

```
taxiData_min_summary = summarizedTable(taxiData_min);
taxiData_min_summary = taxiData_min_summary(~isnan(taxiData_min_summary.AvgDistance),:);
taxiData_min_summary = taxiData_min_summary(~isnan(taxiData_min_summary.NetPickups),:);
taxiData_min_summary = sortrows(taxiData_min_summary,'Time','ascend');

taxiData_min_summary_table = timetable2table(taxiData_min_summary,'ConvertRowTimes',true);
taxiData_min_summary_table.Properties.VariableNames = ["HourlyBin", ...
        "AvgDistance", ...
        "AvgDuration", ...
        "AvgFare", ...
        "PickupCount", ...
        "DropoffCount", ...
        "NetPickups", ...
        "Region"];

function taxiData_min_summary = summarizedTable(taxiData_min)
    % "Lower Manhattan","Midtown","Upper East Side","Upper West Side","JKF Airport","LaGuardia Airport"
    taxiData_min_LM = taxiData_min(taxiData_min.PickupRegion == "Lower Manhattan" | ...
           taxiData_min.DropoffRegion ==  "Lower Manhattan",:);
    taxiData_min_M = taxiData_min(taxiData_min.PickupRegion == "Midtown" | ...
           taxiData_min.DropoffRegion ==  "Midtown",:);
    taxiData_min_UES = taxiData_min(taxiData_min.PickupRegion == "Upper East Side" | ...
           taxiData_min.DropoffRegion ==  "Upper East Side",:);
    taxiData_min_UWS = taxiData_min(taxiData_min.PickupRegion == "Upper West Side" | ...
           taxiData_min.DropoffRegion ==  "Upper West Side",:);
    taxiData_min_JFK = taxiData_min(taxiData_min.PickupRegion == "JKF Airport" | ...
           taxiData_min.DropoffRegion ==  "JKF Airport",:);
    taxiData_min_LG = taxiData_min(taxiData_min.PickupRegion == "LaGuardia Airport" | ...
           taxiData_min.DropoffRegion ==  "LaGuardia Airport",:);
```

```
    taxiData_min_LM = tableRetime(taxiData_min_LM,"Lower Manhattan");
    taxiData_min_M = tableRetime(taxiData_min_M,"Midtown");
    taxiData_min_UES = tableRetime(taxiData_min_UES,"Upper East Side");
    taxiData_min_UWS = tableRetime(taxiData_min_UWS,"Upper West Side");
    taxiData_min_JFK = tableRetime(taxiData_min_JFK,"JKF Airport");
    taxiData_min_LG = tableRetime(taxiData_min_LG,"LaGuardia Airport");

    taxiData_min_summary = vertcat(taxiData_min_LM, ...
            taxiData_min_LG, ...
            taxiData_min_JFK, ...
            taxiData_min_UWS, ...
            taxiData_min_UES, ...
            taxiData_min_M);
end

function [partial_table] = tableRetime(table, name)
    taxiData = timetable(table.PickupTime,table.Distance,(table.DropoffTime-table.PickupTime),table.Fare);
    taxiData = retime(taxiData,'hourly','mean');
    taxiData.Properties.VariableNames = ["AvgDistance","AvgDuration","AvgFare"];

    taxiData = taxiData(~isnan(taxiData.AvgDistance),:);

    taxiData_pickup = timetable(table(table.PickupRegion == name,:).PickupTime, ...
           table(table.PickupRegion == name,:).PickupTime);
    taxiData_pickup = retime(taxiData_pickup,'hourly','count');
    taxiData_pickup.Properties.VariableNames = "PickupCount";

    taxiData_dropoff = timetable(table(table.DropoffRegion == name,:).DropoffTime, ...
           table(table.DropoffRegion == name,:).DropoffTime);
    taxiData_dropoff = retime(taxiData_dropoff,'hourly','count');
    taxiData_dropoff.Properties.VariableNames = "DropoffCount";

    taxiData_pickdrop = outerjoin(taxiData_pickup,taxiData_dropoff);

    taxiData_pickdrop.NetPickups = taxiData_pickdrop.PickupCount - taxiData_pickdrop.DropoffCount;

    taxiData_pickdrop = taxiData_pickdrop(~isnan(taxiData_pickdrop.PickupCount),:);

    partial_table = outerjoin(taxiData,taxiData_pickdrop);
    partial_table.Region = strings(height(partial_table),1);
    partial_table.Region(partial_table.Region == "") = name;

    clear taxiData taxiData_pickup taxiData_dropoff taxiData_pickdrop
end
```

Data split

```
taxiPartition = cvpartition(height(taxiData_min_summary_table),"HoldOut",0.2);

taxiTestIdx = test(taxiPartition);
taxiTest = taxiData_min_summary_table(taxiTestIdx,:);

taxiTrainIdx = training(taxiPartition);
taxiTrain = taxiData_min_summary_table(taxiTrainIdx,:);
```

Classification model

```
function [trainedClassifier, validationAccuracy] = trainClassifierFinetree(trainingData)
```

```
% [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 24-Aug-2022 14:15:55


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'AvgDistance', ...
    'AvgFare',  ...
    'PickupCount',  ...
    'DropoffCount',  ...
    'NetPickups',  ...
    'Day',  ...
    'Month',  ...
    'Hour',  ...
    'Region',  ...
    'isHoliday'};
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;
```

```
isCategoricalPredictor = [false, false, false, false, false, false, false, false, true, true];

% Data transformation: Select subset of the features
% This code selects the same subset of features as were used in the app.
includedPredictorNames = predictors.Properties.VariableNames([false true true true false false true true true
predictors = predictors(:,includedPredictorNames);
isCategoricalPredictor = isCategoricalPredictor([false true true true false false true true true false]);

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'Cost', [0 2 2; 4 0 3; 4 3 0], ...
    'ClassNames', categorical({'High'; 'Low'; 'Medium'}));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
featureSelectionFcn = @(x) x(:,includedPredictorNames);
treePredictFcn = @(x) predict(classificationTree, x);
trainedClassifier.predictFcn = @(x) treePredictFcn(featureSelectionFcn(predictorExtractionFcn(x)));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'AvgDistance',  ...
    'AvgFare',  ...
    'Day',  ...
    'DropoffCount',  ...
    'Hour',  ...
    'Month',  ...
    'NetPickups',  ...
    'PickupCount',  ...
    'Region',  ...
    'isHoliday'};
trainedClassifier.ClassificationTree = classificationTree;
trainedClassifier.About = 'This struct is a trained model exported from Classification Learner R2022a.';

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'AvgDistance',  ...
    'AvgFare',  ...
    'PickupCount',  ...
    'DropoffCount',  ...
    'NetPickups',  ...
    'Day',  ...
    'Month',  ...
    'Hour',  ...
    'Region',  ...
    'isHoliday'};
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, true, true];
```

```
% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationTree, 'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```