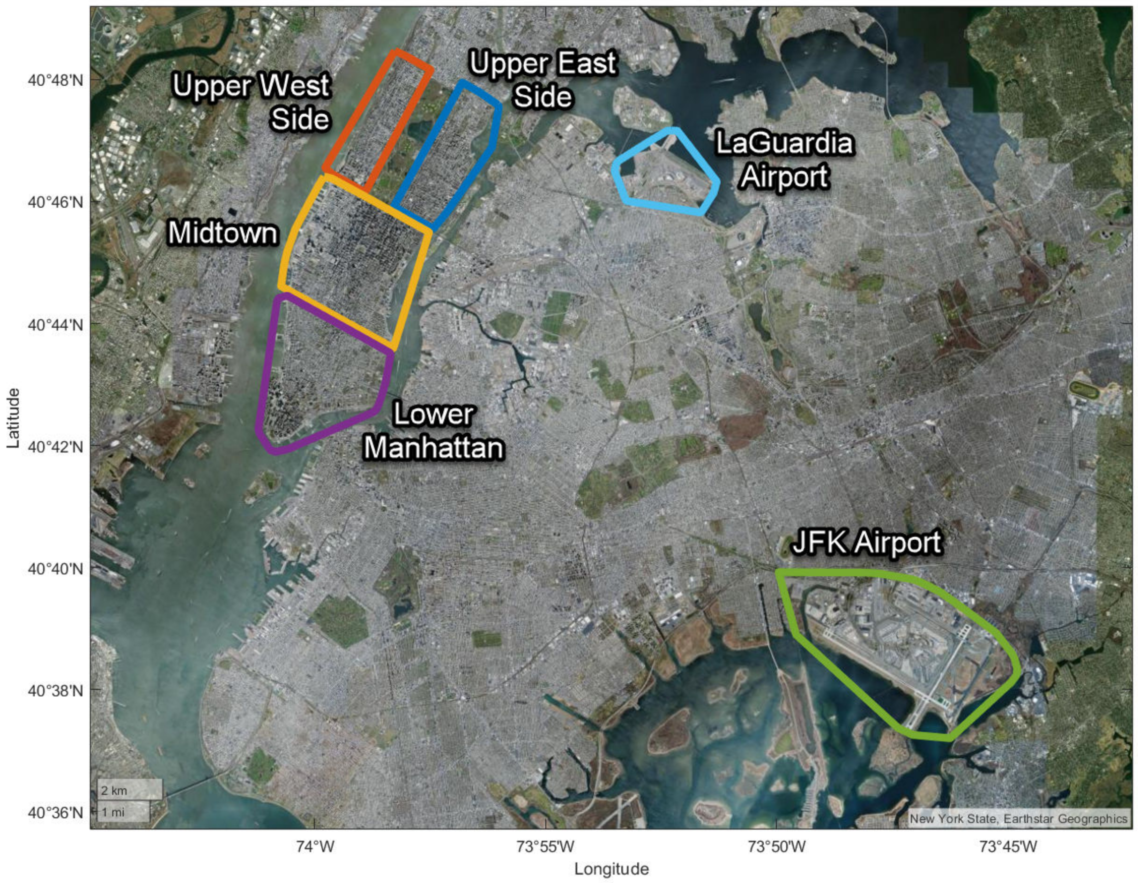
Capstone Project

Table of Contents

1. Objectives

The goal is to predict demand for taxis in six NYC regions (i.e. Manathan and airports) shown below.



The original data contains individual taxi trips that will be converted in taxi demand per region per hour. Features will have to be enginered to allow predicting the demand. The Taxi demand is labeled as "Low", "Medium" and "High. The model will have to predict demand and aim to improve efficiency and maximize profits. Two models need to be developped.

The first one is a baseline model that emphasizes on overall accuracy.

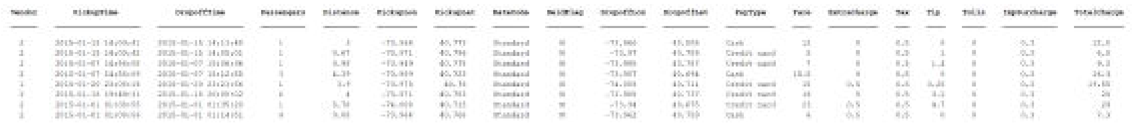
The second one is meant to evaluate a specific taxi deployment strategy that follows following rules:

- Always go to the nearest High demand region when one is available
- Go to the nearest Medium demand region if there is no High demand region available
- Never go to or stay in a Low demand region

The strategy emphasis the fact that it is critical to avoid sending taxi in "Low" demand regions. As a result, misclassification of "Low" demand is costly and should be avoided (i.e. reducing false negative for "Low"). In this strategy, mixing up High and Medium demand is not considered to be costly.

2. Data

The code used to load the initial data is shown in section 5.1. It contains the following information :



VendorID	PickupTime	DropoffTime	Passengers	Distance	PickupLon	PickupLat	RateCode	HeldFlag	DropoffLon	DropoffLat	PayType	Fare	ExtraCharge	Tax	Tip	Tolls	ImpSurcharge	TotalCharge
1	2015-01-15 14:05:41	2015-01-15 14:13:48	1	3	-73.988	40.771	Standard	N	-73.966	40.800	Cash	13	0	0.5	0	0	0.3	13.8
1	2015-01-15 14:05:42	2015-01-15 14:08:01	1	0.47	-73.971	40.796	Standard	N	-73.97	40.789	Credit card	3	0	0.5	1	0	0.3	4.8
1	2015-01-07 14:06:08	2015-01-07 15:08:08	1	0.98	-73.989	40.778	Standard	N	-73.988	40.787	Credit card	7	0	0.5	1.4	0	0.3	9.2
1	2015-01-07 14:06:09	2015-01-07 15:12:03	1	1.29	-73.989	40.723	Standard	N	-73.987	40.694	Cash	10.5	0	0.5	0	0	0.3	14.3
1	2015-01-20 23:06:16	2015-01-20 23:23:06	1	3.7	-73.976	40.74	Standard	N	-74.008	40.711	Credit card	15	0.5	0.5	3.28	0	0.3	19.88
1	2015-01-16 19:48:11	2015-01-16 20:05:02	6	4	-73.971	40.763	Standard	N	-73.989	40.737	Credit card	16	0	0.5	3.2	0	0.3	20
1	2015-01-01 01:08:55	2015-01-01 01:35:10	1	5.78	-74.009	40.713	Standard	N	-73.94	40.695	Credit card	21	0.5	0.5	4.7	0	0.3	28
1	2015-01-01 01:09:04	2015-01-01 01:14:01	4	0.80	-73.944	40.708	Standard	N	-73.942	40.709	Cash	6	0.5	0.5	0	0	0.3	7.3

The original dataset is composed of 2,922,266 taxi trip events that happened in the year 2015. This data provide the vendor, the pickup and dropoff time, the number of passengers (ranging from 0 to 9), the rate code, the held flag, dropoff and pickup longitude and latitude, the pay type. It also contains the fare, the extra charge, the tax, the tip, the tolls, surcharges and the total charge.

In section 5.1, the zones and boroughs where trips started and ended are added as well as the region. The new fields are PickupZone, tzPickupBorough, DropoffZone, tzDropoffBorough, PickupRegion and DropoffRegion.

2.1 Cleaning Data

The data cleaning process is splitted in two sections (see the code in section 5.2). The first half contains the provided data cleaning steps. These cleaning steps aim mostly to suppress badly registered information such as

1. Negative costs (i.e fare, tax, tip, toll, charge).
2. Trips with charge issues (i.e. the difference between the summation of every costs and the total charge is larger than 0.01 $)
3. Trips with 0 passengers
4. Trips with negative distance
5. Trips with missing Pickup and Dropoff longitude and latitude and with such fields that are outside the NYC area (i.e. lat limit = [40.5612 40.9637]; longitude limit = [-74.1923 -73.5982])

Finally, some simple feature were added, such as the trip duration and the average speed. These features are interesting to catch unreasonble trips.

In the custom part of section 5.2, the duration is used to filterout some trips (i.e only trips that last between 0.1 and 140 minutes are kept). However, some trips can range to many hours and many trips are around 1440 minutes (which is the full day). These trips are removed since they are most likely an error in the trip recording.

Similarly for the distance, the filtering was made to keep only trips between 0.1 to 50 mi. The upper limits is about twice the size of the NYC area. 0.6 % of the data were rejected at this step.

Fig. 1

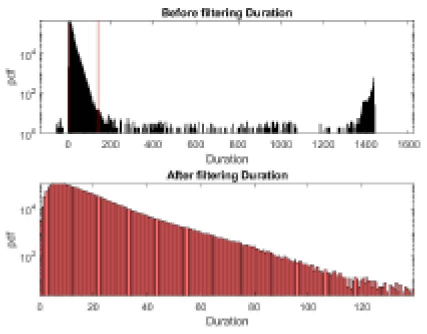
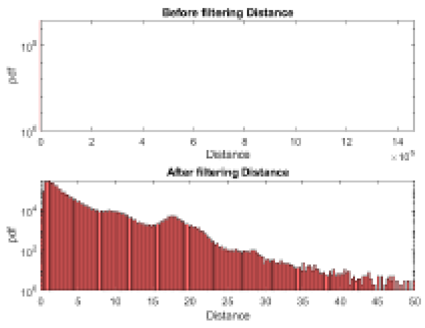
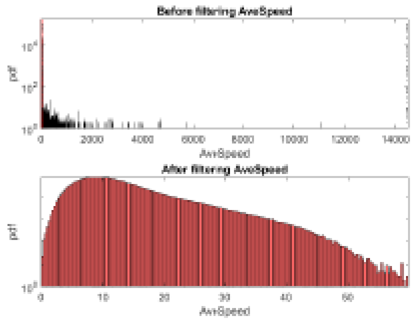


Fig. 2



If we consider a reasonable average trip speed of about 60 mph (which is farly large for a metropolitan area), another 0.06 % of the data is removed.

Fig. 3



0.07 % of data is filtered due to excessive fare (larger than 100 $). A negligible amount of outliers were found using only the total charges, which is why the rate divided by the distance was used to filter fare that seemed too expensive for very short trips (0.4 % were filtered).

Fig. 4

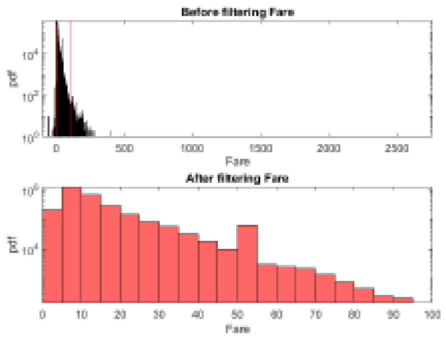


Fig. 5

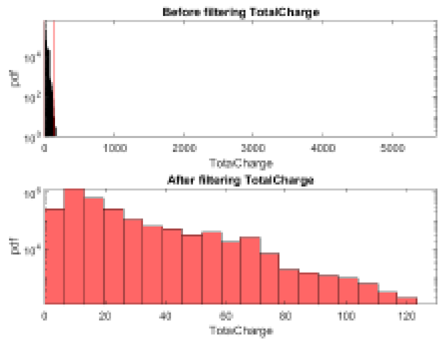
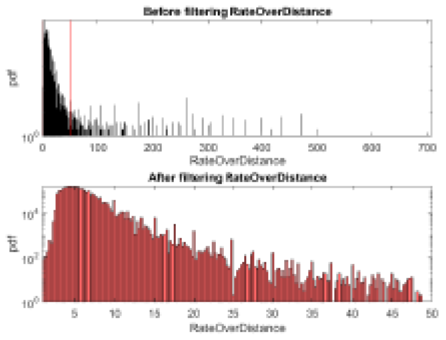


Fig. 6



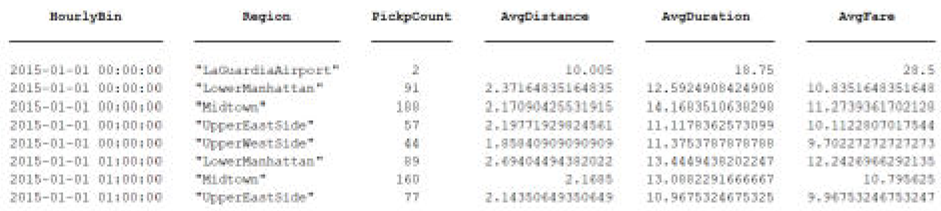
Once this filtering was accomplish, about 1.2% of the trips were removed.

Finally, a last big portion of the data was removed when rejecting pickup or drop off location outside the NYC areas. 81.6 % of the data was kept.

2.2 Restructuring Data

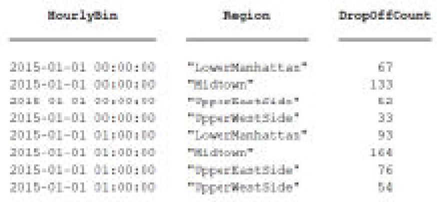
In the data restructuring section, every taxi trips are merged together per region and per hour of each days. To achieve that, the pick up time and the drop off time were rounded to the hour. For the pickup time, aside of the time and region, the pickyp count in each NYC regions were recorded, as well as the average distance, the average duration and the average fare. As for the dropoff, only the count was recorded. The first few rows of each data are shown below.

PickupGroup:



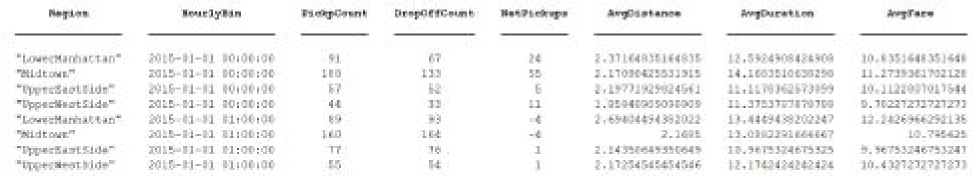
HourlyBin	Region	PickpCount	AvgDistance	AvgDuration	AvgFare
2015-01-01 00:00:00	"LaGuardiaAirport"	2	10.005	18.75	28.5
2015-01-01 00:00:00	"LowerManhattan"	91	2.37164835164835	12.5924908424908	10.8351648351648
2015-01-01 00:00:00	"Midtown"	188	2.17090425531915	14.1683510638298	11.2739361702128
2015-01-01 00:00:00	"UpperEastSide"	57	2.19771929824561	11.1178362573099	10.1122807017544
2015-01-01 00:00:00	"UpperWestSide"	44	1.85840909090909	11.3753787878788	9.70227272727273
2015-01-01 01:00:00	"LowerManhattan"	89	2.69404494382022	13.4449438202247	12.2426966292135
2015-01-01 01:00:00	"Midtown"	160	2.1695	13.0882291666667	10.795625
2015-01-01 01:00:00	"UpperEastSide"	77	2.14350649350649	10.9675324675325	9.96753246753247

DropOffGroup:



HourlyBin	Region	DropOffCount
2015-01-01 00:00:00	"LowerManhattan"	67
2015-01-01 00:00:00	"Midtown"	133
2015-01-01 00:00:00	"UpperEastSide"	52
2015-01-01 00:00:00	"UpperWestSide"	33
2015-01-01 01:00:00	"LowerManhattan"	93
2015-01-01 01:00:00	"Midtown"	164
2015-01-01 01:00:00	"UpperEastSide"	76
2015-01-01 01:00:00	"UpperWestSide"	54

Finally, a innerjoin operation is done to merge together both table with the time and region. The first few lines of the grouped data can be seen below.



Region	HourlyBin	PickpCount	DropOffCount	NetPickups	AvgDistance	AvgDuration	AvgFare
"LowerManhattan"	2015-01-01 00:00:00	91	67	24	2.37164835164835	12.5924908424908	10.8351648351648
"Midtown"	2015-01-01 00:00:00	188	133	55	2.17090425531915	14.1683510638298	11.2739361702128
"UpperEastSide"	2015-01-01 00:00:00	57	52	5	2.19771929824561	11.1178362573099	10.1122807017544
"UpperWestSide"	2015-01-01 00:00:00	44	33	11	1.85840909090909	11.3753787878788	9.70227272727273
"LowerManhattan"	2015-01-01 01:00:00	89	93	-4	2.69404494382022	13.4449438202247	12.2426966292135
"Midtown"	2015-01-01 01:00:00	160	164	-4	2.1695	13.0882291666667	10.795625
"UpperEastSide"	2015-01-01 01:00:00	77	76	1	2.14350649350649	10.9675324675325	9.96753246753247
"UpperWestSide"	2015-01-01 01:00:00	55	54	1	2.17254545454546	12.1742424242424	10.4327272727273

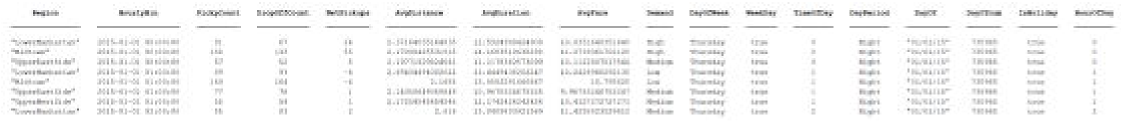
The code used to get this information is shown in section 5.3. The variable "NetPickups" which is the difference betwen the pickup and the dropoff count was added.

2.3 Feature Engineering

The data is then split into two groups, the TaxiTest and the TaxiTrain groups (section 5.4). The TaxiTrain groups contains 80 % (holdout of 0.2) and will be used to train models whereas the remaining 20 % will be stored in the TaxiTest data and will be used to validate the model.

Once the data splitting is done, some features are added. The first one is the "Demand", which is the response to predict. The demand is labeled as "high if the number of net pickup is larger than 15, "medium" if it is between 0 and 15 and negative net pick up are labeled as "low.

The other added features are related to the time of the day, the time of the year and if it is a holiday.



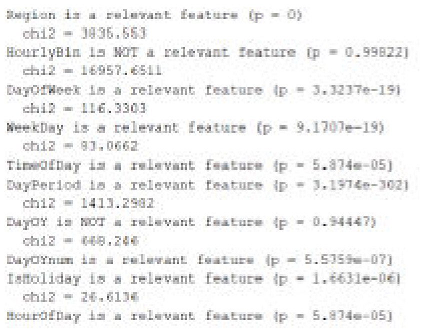
Region	HourlyBin	PickupCount	DropOffCount	NetPickups	AvgDistance	AvgDuration	AvgFare	Demand	DayOfWeek	WeekDay	TimeOfDay	DayPeriod	DayOY	DayOfYear	IsHoliday	HourOfDay
"LowerManhattan"	2015-01-01 00:00:00	91	67	24	2.37164835164835	12.5924908424908	10.8351648351648	High	Thursday	true	0	Night	"01/01/15"	[illegible]	true	0
"Midtown"	2015-01-01 00:00:00	188	133	55	2.17090425531915	14.1683510638298	11.2739361702128	High	Thursday	true	0	Night	"01/01/15"	[illegible]	true	0
"UpperEastSide"	2015-01-01 00:00:00	57	52	5	2.19771929824561	11.1178362573099	10.1122807017544	Medium	Thursday	true	0	Night	"01/01/15"	[illegible]	true	0
"LowerManhattan"	2015-01-01 01:00:00	89	93	-4	2.69404494382022	13.4449438202247	12.2426966292135	Low	Thursday	true	1	Night	"01/01/15"	[illegible]	true	1
"Midtown"	2015-01-01 01:00:00	160	164	-4	2.1695	13.0882291666667	10.795625	Low	Thursday	true	1	Night	"01/01/15"	[illegible]	true	1
"UpperEastSide"	2015-01-01 01:00:00	77	76	1	2.14350649350649	10.9675324675325	9.96753246753247	Medium	Thursday	true	1	Night	"01/01/15"	[illegible]	true	1
"UpperWestSide"	2015-01-01 01:00:00	55	54	1	2.17254545454546	12.1742424242424	10.4327272727273	Medium	Thursday	true	1	Night	"01/01/15"	[illegible]	true	1
"LowerManhattan"	2015-01-01 02:00:00	85	83	2	2.419	[illegible]	[illegible]	Medium	Thursday	true	2	Night	"01/01/15"	[illegible]	true	2

The fields "PickupCount", "DropOffCound", NetPickups are removed from the final data used to train a model because they are directly linked to the output (demand) and are not a variable known to the taxi driver at the time to make decision.The average distance, duration and fare (i.e. AvgDistance, AvgDuration and AvgFare respectively) will also be removed frome the training/test data for the same reason. The code used to remove the data are shown in section 5.5.

To evaluate if the new features are worth keeping to train the models, filter methods are used. Since the demand is a descrete predictor,

1. if the feature to analyse is discrete, the filter technique that was used was a chi squared statistic (coming from a cross-tabulation figure). The p value of the test is provided
2. if the feature to analyse is continuous, the anova test was performed. This test also return a p-value.

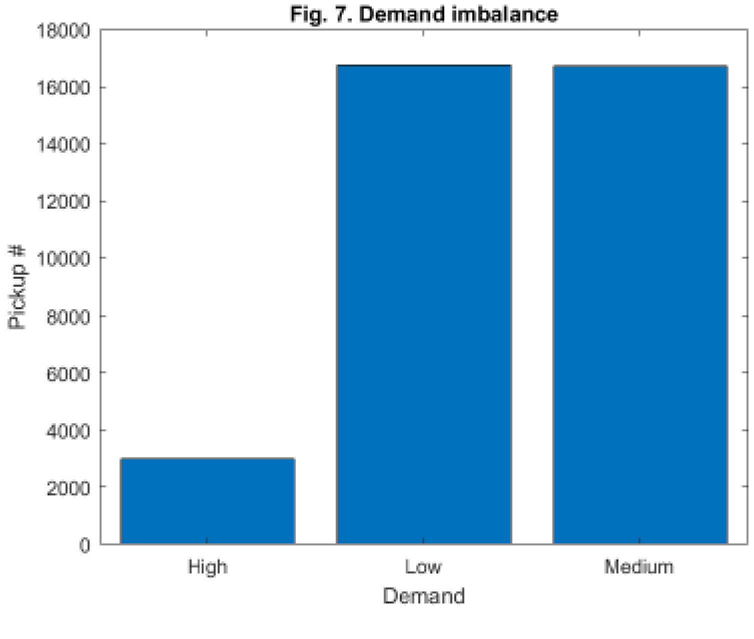
If the p value is close to 0, it strongly suggests that both variables are not independant which implies it is likely a good feature to use in the model. As a result, the hourlybin and the dayOY are removed from the Taxidata.



```
Region is a relevant feature (p = 0)
  chi2 = 3836.563
HourlyBin is NOT a relevant feature (p = 0.99822)
  chi2 = 16957.6511
DayOfWeek is a relevant feature (p = 3.3237e-19)
  chi2 = 116.3303
WeekDay is a relevant feature (p = 9.1707e-19)
  chi2 = 93.0662
TimeOfDay is a relevant feature (p = 5.874e-05)
DayPeriod is a relevant feature (p = 3.1974e-302)
  chi2 = 1413.2982
DayOY is NOT a relevant feature (p = 0.94447)
  chi2 = 668.246
DayOYnum is a relevant feature (p = 5.5759e-07)
IsHoliday is a relevant feature (p = 1.6631e-06)
  chi2 = 26.6136
HourOfDay is a relevant feature (p = 5.874e-05)
```

Finally, the last analysis made on the dataset is about the imbalance. Figure below displays the number of pick ups sorted by demand type. It can be seen that the high demand is not as well "represented" than the two other categories. This might affect the model in the next section.

Fig. 7. Demand imbalance



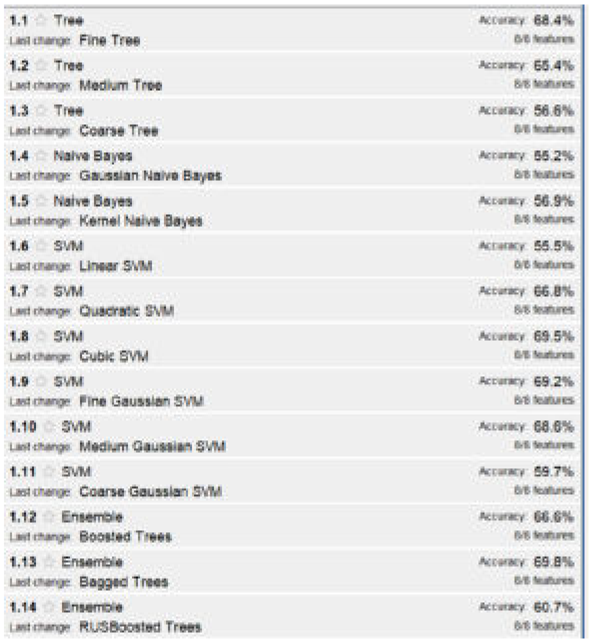
3. Modeling

3.1 Scenario 1: Maximize Accuracy

In the first scenario, the goal is simply to maximize the accuracy of the model. This figure of merit will thus be the only one used to quantify the final model although the other figure of merits such as precision, recall, fallout, specificity and F1 will be provided for future references.

3.1.1 Investigation of various models

Many models were tried as shown below.



1.1 Tree — Last change: Fine Tree — Accuracy: 68.4% — 8/8 features

1.2 Tree — Last change: Medium Tree — Accuracy: 65.4% — 8/8 features

1.3 Tree — Last change: Coarse Tree — Accuracy: 56.6% — 8/8 features

1.4 Naive Bayes — Last change: Gaussian Naive Bayes — Accuracy: 55.2% — 8/8 features

1.5 Naive Bayes — Last change: Kernel Naive Bayes — Accuracy: 56.9% — 8/8 features

1.6 SVM — Last change: Linear SVM — Accuracy: 55.5% — 8/8 features

1.7 SVM — Last change: Quadratic SVM — Accuracy: 66.8% — 8/8 features

1.8 SVM — Last change: Cubic SVM — Accuracy: 69.5% — 8/8 features

1.9 SVM — Last change: Fine Gaussian SVM — Accuracy: 69.2% — 8/8 features

1.10 SVM — Last change: Medium Gaussian SVM — Accuracy: 68.6% — 8/8 features

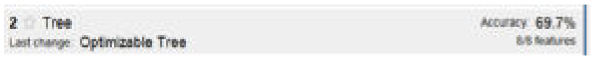
1.11 SVM — Last change: Coarse Gaussian SVM — Accuracy: 59.7% — 8/8 features

1.12 Ensemble — Last change: Boosted Trees — Accuracy: 66.6% — 8/8 features

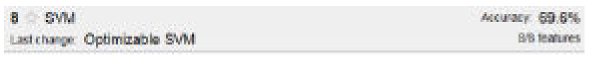
1.13 Ensemble — Last change: Bagged Trees — Accuracy: 69.8% — 8/8 features

1.14 Ensemble — Last change: RUSBoosted Trees — Accuracy: 60.7% — 8/8 features

A few model seems to be performing well such as Fine Trees, Cubic SVM, fine gaussian SVM and bagged trees. To improve the accuracy of the various models, hyperparameter optimization were done for tree and SVM.

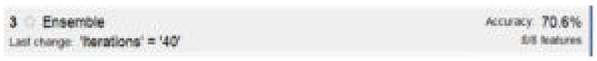


2 Tree — Last change: Optimizable Tree — Accuracy: 69.7% — 8/8 features



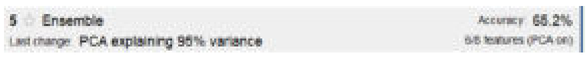
8 SVM — Last change: Optimizable SVM — Accuracy: 69.6% — 8/8 features

Ensemble optimization was tried as well. The best performing model was bagged tree.



3 Ensemble — Last change: 'Iterations' = '40' — Accuracy: 70.6% — 8/8 features

PCA was then included in the ensemble optimization but did not seemed to improve the model accuracy. The original filtering seems to have already filtered out the useless features.



5 Ensemble — Last change: PCA explaining 95% variance — Accuracy: 65.2% — 8/8 features (PCA on)

3.1.2 Final Model Description

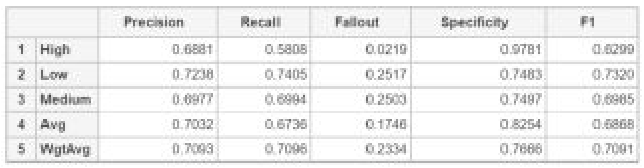
In this subsection, the final model is described. The code used to get the final model is shown in section 5.6.

The model that was selected is a bagged tree. The maximum number of decision splits is 248, the number of predictors for each split was 3 and the number of cycle was 471. The other parameters are defaults parameters. A 20-fold validation step was used to prevent overfitting. These previous number were obtained using an optimization procedure. The parameters that was optimized was the maximum number of splits and the number of learners. The predictors that were used was, Region, DayOfWeek, WeekDay, TimeOfDay, DayPeriod, DayOYnum and is Holiday.

In this training procedure, the misclassification cost was not changed.

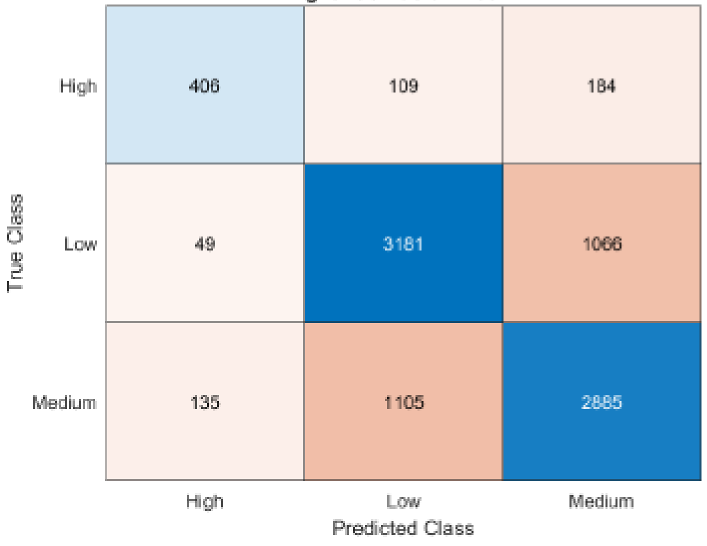
The accuracy of this model is 70.96 % when using the test data. It was 72.7 % when using the train dataset which suggest that the overfitting was very low since the accuracy reduction when using an totally independant dataset is very small.

The precision, recall, fallout, specificity and F1 are shown in the table below. The confusion matrix is also shown below.



		Precision	Recall	Fallout	Specificity	F1
1	High	0.6881	0.5808	0.0219	0.9781	0.6299
2	Low	0.7238	0.7405	0.2517	0.7483	0.7320
3	Medium	0.6977	0.6994	0.2503	0.7497	0.6985
4	Avg	0.7032	0.6736	0.1746	0.8254	0.6868
5	WgtAvg	0.7093	0.7096	0.2334	0.7666	0.7091

Fig. 8: Confusion Matrix



Finally, Fig. 9 shows the recall over fallout curve for the three value of demand (High, Low and Medium). The red dots shows the location of the actual model. The legend display the area under the curve. In this graph, we can see that the model has a fairly low false negative for "high" but the true positive rate is lower than the two other categories.

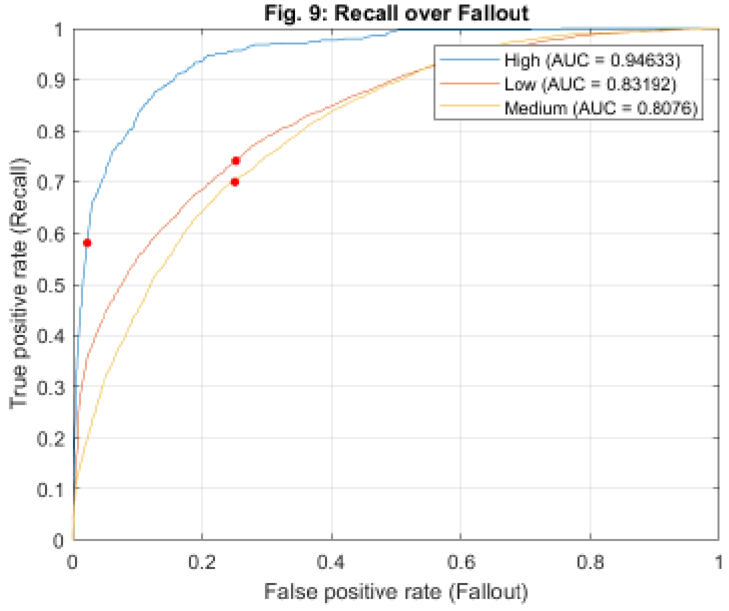


Fig. 9: Recall over Fallout

3.2 Scenario 2: Reducing False Negatives for Low demand,

In this scenario, the goal is to optimize the taxi deployement strategy. The strategy provided is:

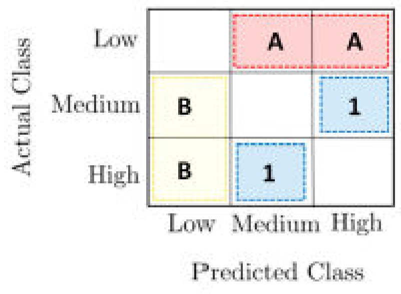
1. Always go to the nearest High demand region when one is available: The model thus need to predict well high demand
2. Go to the nearest Medium demand region if there is no High demand region available : A bad misclassification between high and medium is not so expensive
3. Never go to or stay in a Low demand region: Good classification of low demand is very important

So, to sum up, it is very important that the model has

1. low false negative for the low demand
2. low false positive for the low demand (false positive for low is less expensive than false negative)
3. Medium and high demand misclassification is not an issue

This strategy is sum up in the figure below:

Fig 10: Classification cost



3.2.1 Investigation of Different Models

For this second model, the ensemble bag tree model was still investigated but with an addiion of higher misclassification cost for the red and yellow boxes of the previous graph. Furthermore, since the "low" demand prediction is more critical in term of cost, the RUSboosted tree is also considered to take into account the class imbalance.

Another things that is considered here is the fact that misclassification between medium and high is not considered important. So, a "Demandv2" feature is added where the "Medium" and "High" categories are merged together into the new MedHigh category. Reducing the number of categories from 3 to 2 might help to improve the model prediction. This will also stongly remove the class imbalance.

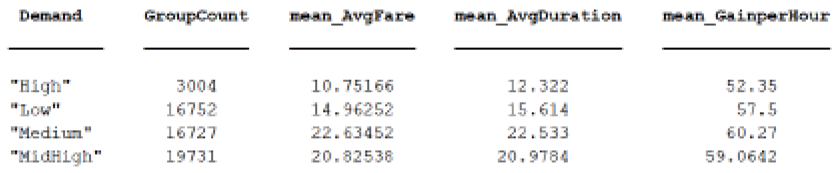
The code use to prepare these new data are shown in section 5.7

To choose the best model, some hypothesis needs to be made.

1. Every taxi that goes to a low region does not earn anything
2. Every taxi that goes to medium or high region earn the average amount of dollar of its category.
3. Two metrics can be calculated:

- The amount of loss due to model errors (red boxes from Fig. 10)
- The amount of missed opportunities due to model predicting false low demand (yellow boxes from Fig. 10)

The following chart shows the mean average fare for each categories and the mean average duration of trip. The mean gain per hour is thus calculated from the mean_AvgFare/mean_AvgDuration*60.



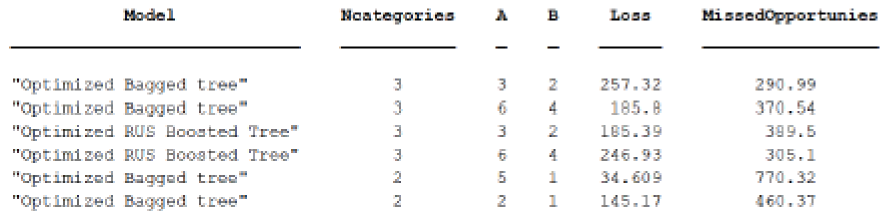
Demand	GroupCount	mean_AvgFare	mean_AvgDuration	mean_GainperHour
"High"	3004	10.75166	12.322	52.35
"Low"	16752	14.96252	15.614	57.5
"Medium"	16727	22.63452	22.533	60.27
"MidHigh"	19731	20.82538	20.9784	59.0642

Note: The MidHigh values are calculated using weighted average.

The 6 models considered are described below. Only the best models will be described in more details

1. Bagged Tree. The first model that was trained is an optimized bagged tree where A = 3 and B = 2. A second model with A = 6 and B = 2 was also trained.
2. RUSBoost. Similarly, two models were tried for RUSBoots in hope to improve the performance due to class imbalance.
3. Bagged Tree with Merged High and Medium. Two models were tried where A = 5 and 2 respectively and B was always 1.

The summary of the performances of these model are shown in the next table. Both the loss and missed opportunities are in k$.



Model	Ncategories	A	B	Loss	MissedOpportunies
"Optimized Bagged tree"	3	3	2	257.32	290.99
"Optimized Bagged tree"	3	6	4	185.8	370.54
"Optimized RUS Boosted Tree"	3	3	2	185.39	389.5
"Optimized RUS Boosted Tree"	3	6	4	246.93	305.1
"Optimized Bagged tree"	2	5	1	34.609	770.32
"Optimized Bagged tree"	2	2	1	145.17	460.37

It can be seen that reducing the number of categories is helpful to reduce the Loss. However, obviously, the missed opportunites increase (trade off usually seen on ROC curve). Since the loss is more important in this scenario, the fifth model is going to be kept here.

3.2.2 Final Model Description

The model that was selected is a bagged tree. The maximum number of decision splits is 248 and the number of cycle was 497. The other parameters are defaults parameters. A 20-fold validation step was used to prevent overfitting. These previous number were obtained using an optimization procedure. The parameters that was optimized was the maximum number of splits and the number of learners. The predictors that were used was, Region, DayOfWeek, WeekDay, TimeOfDay, DayPeriod, DayOYnum and is Holiday. Finally, in this model, the high and medium categories were merged to the medhigh category.

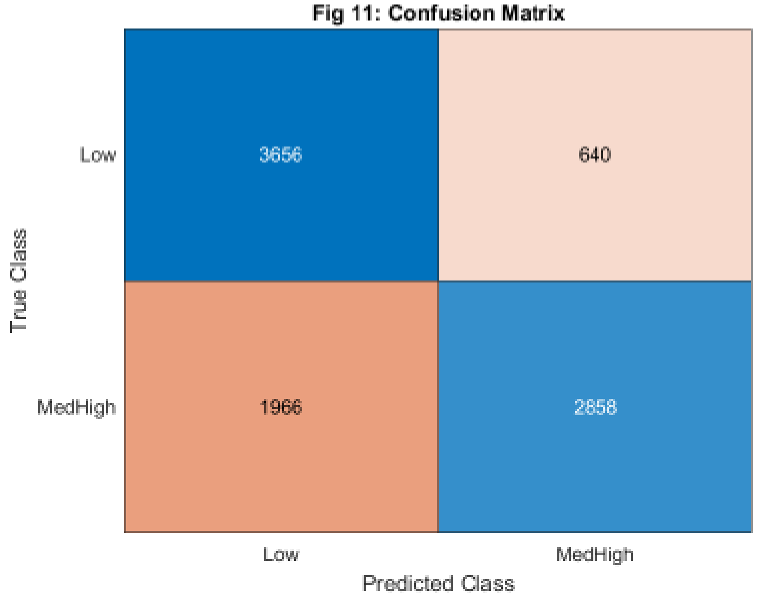
In this training procedure, the misclassification cost 5 for low classified as midhigh and 1 for midhigh classified as low.

The accuracy of this model is 71.43 % when using the test data. It was 73.66 % when using the train dataset which suggest that the overfitting was very low since the accuracy reduction when using an totally independant dataset is very small.

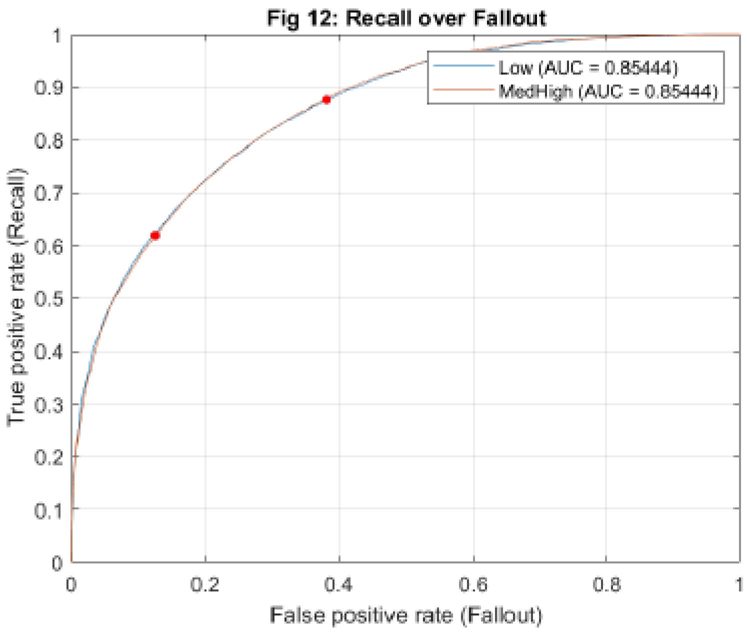
The precision, recall, fallout, specificity and F1 are shown in the table below. The confusion matrix is also shown in Fig. 11

		Precision	Recall	Fallout	Specificity	F1
1	Low	0.6503	0.8510	0.4075	0.5925	0.7372
2	MedHigh	0.8170	0.5925	0.1490	0.8510	0.6869
3	Avg	0.7337	0.7217	0.2783	0.7217	0.7120
4	WgtAvg	0.7385	0.7143	0.2708	0.7292	0.7106

Fig 11: Confusion Matrix



Finally, Fig. 12 shows the recall over fallout curve for the two value of demand (Low and MedHigh). The red dots shows the location of the actual model. The legend display the area under the curve. In this graph, we can see that both curves are overlapped, which is expected when there are only two categories. In this example, it can be seen that we favored bad prediction of Medhigh vs bad prediction of low since this approach is expected to be less costly.

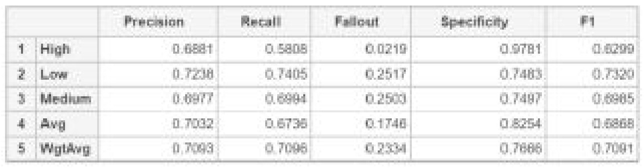


4. Conclusions

In this document, two models were presented. In the first model, the objective was to maximize the accuracy. In the second model, the goal was to minimize the loss. As a result, the model needed to predict well low demands in order to make sure that cab were not going there. Since the cost of misclassification between high and medium demand, these two categories were merged to reduce the class imbalance and minimize the loss.

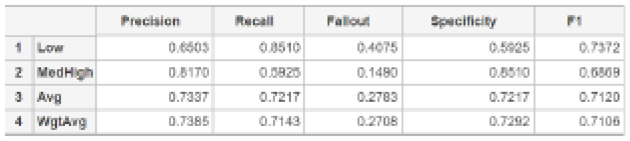
Both model performances are shown in the tables below:

Model 1



		Precision	Recall	Fallout	Specificity	F1
1	High	0.6881	0.5808	0.0219	0.9781	0.6299
2	Low	0.7238	0.7405	0.2517	0.7483	0.7320
3	Medium	0.6977	0.6994	0.2503	0.7497	0.6985
4	Avg	0.7032	0.6736	0.1746	0.8254	0.6868
5	WgtAvg	0.7093	0.7096	0.2334	0.7666	0.7091

Model 2



		Precision	Recall	Fallout	Specificity	F1
1	Low	0.6503	0.8510	0.4075	0.5925	0.7372
2	MedHigh	0.8170	0.5925	0.1490	0.8510	0.6869
3	Avg	0.7337	0.7217	0.2783	0.7217	0.7120
4	WgtAvg	0.7385	0.7143	0.2708	0.7292	0.7106

As can be seen, the goal of the second model was to improve the "low" recall value. Model two was 0.74 whereas the second model had a recall rate of 0.85. The average precision of the model is not really a good metric to compare mostly because high and medium demand were removed in the second model. Extra calculation would be required on the Model 1 data to do a faire comparison.

A big part of the trade-offs of model 2 is between Recall and Fallout (or between Loss vs Missed opportunities). In a situation where too much cabs are avaialbe in the city, "missed opportunities might be more valuable than pure "loss". To take this kind of situation into account, more information would be needed, For future improvement, the "demand" figure of merit could be replaced by a continuous feature that includes the information of pick up and drop off count, but could also include the fare per trip and the duration of each trip. If the the "demand" metric is replaced by a "$/hour" metric, a regression model could be more precise. Furthermore, an important missing data is the "amount of idle taxi per hour". Knowing the amount of idle taxi per hour per area would also allows to help weighting which metrics to favor (i.e Loss or MissedOpportunities). Finally, a better model would also include in the "demand" metric the time it takes for a cab to move from a "bad" location to a "good" one. Moving from JFK might be more expensive than moving form Midtown. Considering all these features might likely results in a "regression" problem instead of a "classification" one.

To conclude, the model #2 is better suited for a situation where every cab in low are does not get a fare and a cab in the other area gets one. Life if however more complicated than that...

5. Appendix

5.1 Importing Data

```
clear;
TaxiDataStore = fileDatastore("yellow_tripdata_2015-*.csv","ReadFcn",@importTaxiDataWithoutClea
TaxiData = readall(TaxiDataStore);

% The addTaxiZones function provided in Course 3 creates the variables PickupZone and DropoffZo
TaxiData = addTaxiZones(TaxiData);

%Assign PickupRegion DropoffRegion to each taxi trip
TaxiData.PickupRegion = TaxiData.PickupZone;
TaxiData.DropoffRegion = TaxiData.DropoffZone;

TaxiData.PickupRegion = addcats(TaxiData.PickupRegion, {'Lower Manhattan','Midtown','Upper East
TaxiData.PickupRegion = mergecats(TaxiData.PickupRegion, {'Lower Manhattan','Alphabet City','Ba
                                                          'Financial District South','Greenwich
                                                          'Lower East Side','Meatpacking/West V
                                                          'West Village','World Trade Center'})
TaxiData.PickupRegion = mergecats(TaxiData.PickupRegion, {'Midtown','Clinton East','Clinton Wes
                                                          'Penn Station/Madison Sq West','Union
                                                          'Stuy Town/Peter Cooper Village','Wes
TaxiData.PickupRegion = mergecats(TaxiData.PickupRegion, {'Upper East Side','Upper East Side No
TaxiData.PickupRegion = mergecats(TaxiData.PickupRegion, {'Upper West Side','Upper West Side No

TaxiData.DropoffRegion = addcats(TaxiData.DropoffRegion, {'Lower Manhattan','Midtown','Upper Ea
TaxiData.DropoffRegion = mergecats(TaxiData.DropoffRegion, {'Lower Manhattan','Alphabet City',
                                                          'Financial District South','Greenwich
                                                          'Lower East Side','Meatpacking/West V
                                                          'West Village','World Trade Center'})
TaxiData.DropoffRegion = mergecats(TaxiData.DropoffRegion, {'Midtown','Clinton East','Clinton W
                                                          'Penn Station/Madison Sq West','Union
                                                          'Stuy Town/Peter Cooper Village','Wes
```

```
TaxiData.DropoffRegion = mergecats(TaxiData.DropoffRegion, {'Upper East Side','Upper East Side
TaxiData.DropoffRegion = mergecats(TaxiData.DropoffRegion, {'Upper West Side','Upper West Side

TaxiData.PickupRegion = addcats(TaxiData.PickupRegion, {'Other'});
TaxiData.PickupRegion = fillmissing(TaxiData.PickupRegion, 'constant', 'Other');
TaxiData.PickupRegion = mergecats(TaxiData.PickupRegion,["Other","Allerton/Pelham Gardens","Arr
                                                         "Bay Terrace/Fort Totten","Bayside","B
                                                         "Bloomfield/Emerson Hill","Boerum Hil
                                                         "Brighton Beach","Bronx Park","Bronxda
                                                         "Bushwick South","Cambria Heights","Ca
                                                         "Charleston/Tottenville","City Island"
                                                         "Columbia Street","Coney Island","Coro
                                                         "Crown Heights South","Cypress Hills",
                                                         "East Concourse/Concourse Village","Ea
                                                         "East Harlem North","East New York","E
                                                         "Elmhurst","Elmhurst/Maspeth","Elting
                                                         "Flushing","Flushing Meadows-Corona Pa
                                                         "Glen Oaks","Glendale","Governor's Isl
                                                         "Green-Wood Cemetery","Greenpoint","Ha
                                                         "Hillcrest/Pomonok","Hollis","Homecres
                                                         "Jamaica Estates","Kensington","Kew Ga
                                                         "Long Island City/Queens Plaza","Longw
                                                         "Marine Park/Mill Basin","Mariners Har
                                                         "Mott Haven/Port Morris","Mount Hope",
                                                         "Ocean Hill","Ocean Parkway South","Ol
                                                         "Prospect Heights","Prospect Park","Pr
                                                         "Red Hook","Rego Park","Richmond Hill"
                                                         "Saint Albans","Saint George/New Brigh
                                                         "Soundview/Castle Hill","South Beach/D
                                                         "Springfield Gardens South","Spuyten D
                                                         "Sunset Park West","University Heights
                                                         "Washington Heights North","Washington
                                                         "Westchester Village/Unionport","Weste
                                                         "Williamsburg (North Side)","Williamsb
TaxiData.DropoffRegion = fillmissing(TaxiData.DropoffRegion, 'constant', 'Other');
TaxiData.DropoffRegion = addcats(TaxiData.DropoffRegion, {'Other'});
TaxiData.DropoffRegion = mergecats(TaxiData.DropoffRegion,["Other","Allerton/Pelham Gardens","A
                                                                "Bay Terrace/Fort Totten","Bays
                                                                "Bloomfield/Emerson Hill","Boer
                                                                "Brighton Beach","Bronx Park","
                                                                "Bushwick South","Cambria Heigh
                                                                "Charleston/Tottenville","City 
                                                                "Columbia Street","Coney Island
                                                                "Crown Heights South","Cypress 
                                                                "East Concourse/Concourse Villa
                                                                "East Harlem North","East New Y
                                                                "Elmhurst","Elmhurst/Maspeth","
                                                                "Flushing","Flushing Meadows-Co
                                                                "Glen Oaks","Glendale","Governo
                                                                "Green-Wood Cemetery","Greenpoi
                                                                "Hillcrest/Pomonok","Hollis","H
                                                                "Jamaica Estates","Kensington",
                                                                "Long Island City/Queens Plaza"
                                                                "Marine Park/Mill Basin","Marin
                                                                "Mott Haven/Port Morris","Mount
                                                                "Ocean Hill","Ocean Parkway Sou
                                                                "Prospect Heights","Prospect Pa
                                                                "Red Hook","Rego Park","Richmon
                                                                "Saint Albans","Saint George/Ne
                                                                "Soundview/Castle Hill","South
                                                                "Springfield Gardens South","Sp
                                                                "Sunset Park West","University
                                                                "Washington Heights North","Was
                                                                "Westchester Village/Unionport"
                                                                "Williamsburg (North Side)","Wi

TaxiData = TaxiData((~(TaxiData.PickupRegion == "Other") | ~(TaxiData.DropoffRegion == "Other")
```

5.2 Cleaning Data

```
% Data Cleaning using provided code
TaxiData = TaxiData(TaxiData.RateCode ~= "99", :);  % Valid rate code
TaxiData = TaxiData(TaxiData.Fare > 0, :);          % Valid fare
TaxiData = TaxiData(TaxiData.ExtraCharge >= 0, :);  % Positive charge
TaxiData = TaxiData(TaxiData.Tax > 0, :);           % Valid tax
TaxiData = TaxiData(TaxiData.Tip >= 0, :);          % Positive tip
TaxiData = TaxiData(TaxiData.Tolls >= 0, :);        % Positive toll
TaxiData = TaxiData(TaxiData.ImpSurcharge >= 0, :); % Positive charge
TaxiData = TaxiData(TaxiData.TotalCharge > 0, :);   % Valid total
TaxiData = TaxiData(abs(TaxiData.ImpSurcharge-0.3) < 0.01, :);
TaxiData = TaxiData(abs(TaxiData.Tax-0.5) < 0.01, :);
TaxiData = TaxiData(abs(TaxiData.Fare + TaxiData.ExtraCharge + TaxiData.Tax + TaxiData.Tip + Ta
TaxiData = TaxiData(TaxiData.Passengers > 0, :);    % At least 1 passenger
TaxiData = TaxiData(TaxiData.Distance > 0, :);      % Valid trip distance
TaxiData = standardizeMissing(TaxiData, 0, "DataVariables", ["PickupLat","PickupLon","DropoffLa
TaxiData = rmmissing(TaxiData, "DataVariables", ["PickupLat","PickupLon","DropoffLat","DropoffL
TaxiData = addDuration(TaxiData);                   % minutes
TaxiData = addAveSpeed(TaxiData);                   % mph

% Bounding latitude/longitude
lat = [40.5612 40.9637];
lon = [-74.1923 -73.5982];

% Identify pickup and drop off locations inside the region of interest
inROI = inpolygon(TaxiData.PickupLat,TaxiData.PickupLon, lat([1 2 2 1]),lon([1 1 2 2])) & ...
        inpolygon(TaxiData.DropoffLat,TaxiData.DropoffLon, lat([1 2 2 1]),lon([1 1 2 2]));

% Only keep trips that begin and end inside the region of interest.
TaxiData = TaxiData(inROI,:);
```

```
% Custom Cleaning
% Filtering Taxi duration
field = 'Duration'; Threshold = [0.1,140]; nbin = 20000;
TaxiData = FilterLoc(TaxiData, field, Threshold,nbin);

% Filtering Taxi Distance
field = 'Distance'; Threshold = [0.1,50]; nbin = 20000;
TaxiData = FilterLoc(TaxiData, field, Threshold,nbin);

% Filtering Taxi speed
field = 'AveSpeed'; Threshold = [0.1,60]; nbin = 20000;
TaxiData = FilterLoc(TaxiData, field, Threshold,nbin);

% Filtering Taxi Fare
field = 'Fare'; Threshold = [0.4,100]; nbin = 2000;
TaxiData = FilterLoc(TaxiData, field, Threshold,nbin);

% Filtering Taxi Toll
field = 'Tolls'; Threshold = [-0.0001,30]; nbin = 2000;
TaxiData = FilterLoc(TaxiData, field, Threshold,nbin);

% Filtering Taxi TotalCharge
field = 'TotalCharge'; Threshold = [0.5,130]; nbin = 2000;
TaxiData = FilterLoc(TaxiData, field, Threshold,nbin);

% Filtering Taxi RateOverDistance
TaxiData.RateOverDistance = TaxiData.Fare./TaxiData.Distance;
field = 'RateOverDistance'; Threshold = [1,50]; nbin = 20000;
TaxiData = FilterLoc(TaxiData, field, Threshold,nbin);
```

5.3 Restructuring Data

```
% Rounding PickupTime and DropoffTime
TaxiData.PickupTime = dateshift(TaxiData.PickupTime,'start','hour');
TaxiData.DropoffTime = dateshift(TaxiData.DropoffTime,'start','hour');

% Getting group summary
PickUpGroup = groupsummary(TaxiData,["PickupTime","PickupRegion"],"mean",["Distance","Duration"
PickUpGroup.Properties.VariableNames = ["HourlyBin","Region","PickpCount","AvgDistance","AvgDur
DropOffGroup = groupsummary(TaxiData,["DropoffTime","DropoffRegion"]);
DropOffGroup.Properties.VariableNames = ["HourlyBin","Region","DropOffCount"];

GroupTaxiData = innerjoin(DropOffGroup,PickUpGroup,'Keys',{'HourlyBin','Region'});
GroupTaxiData.NetPickups = GroupTaxiData.PickpCount - GroupTaxiData.DropOffCount;
GroupTaxiData = GroupTaxiData(:,[2,1,4,3,8,5,6,7]);
```

5.4 Partitioning Data

```
rng(1);
TaxiPartition = cvpartition(height(GroupTaxiData),"Holdout",0.2);
```

```
TaxiTest = GroupTaxiData(test(TaxiPartition),:);
TaxiTrain = GroupTaxiData(training(TaxiPartition),:);

TaxiTest = AddFeature(TaxiTest);
TaxiTrain = AddFeature(TaxiTrain);
```

5.5 Removing Data

```
TaxiTest = removevars(TaxiTest, {'PickpCount','DropOffCount','NetPickups','AvgDistance','AvgDur
TaxiTrain = removevars(TaxiTrain, {'PickpCount','DropOffCount','NetPickups','AvgDistance','AvgD

% Removing irrelevant fields
TaxiTest = removevars(TaxiTest, {'DayOY','HourlyBin'});
TaxiTrain = removevars(TaxiTrain, {'DayOY','HourlyBin'});

% Summary Analysis
fname = TaxiTrain.Properties.VariableNames;
figure(),
N = length(fname);
for iter = 1:length(fname)
    subplot(ceil(sqrt(N)),floor(sqrt(N)),iter)
    PickUpGroup = groupsummary(TaxiTrain,fname{iter});
    bar(categorical(PickUpGroup.(fname{iter})),PickUpGroup.GroupCount);
    xlabel(fname{iter})
    ylabel('Pickup #')
end
```

5.6 Training Scenario 1

```
[Model] = TrainBestModel(TaxiTrain)
% Model performance with Train
[t,a,C,impValues] = GetModelFOM(Model,TaxiTrain,'Demand');

% Model performance with Test
[t,a,C,impValues] = GetModelFOM(Model,TaxiTest,'Demand');
```

5.7 Training Scenario 2

```
temp = string(TaxiTest.Demand);
temp(temp == "Medium") = "MedHigh";
temp(temp == "High") = "MedHigh";
TaxiTest.Demandv2 = TaxiTest.Demand;
TaxiTest.Demandv2 = categorical(temp);
temp = string(TaxiTrain.Demand);
temp(temp == "Medium") = "MedHigh";
temp(temp == "High") = "MedHigh";
TaxiTrain.Demandv2 = TaxiTrain.Demand;
TaxiTrain.Demandv2 = categorical(temp);
Model = MinimizeLossModel_2Demands(TaxiTrain);

% Model performance with Train
```

```
[t,a,C,impValues] = GetModelFOM(Model,TaxiTrain,'Demandv2');

% Model performance with Test
[t,a,C,impValues] = GetModelFOM(Model,TaxiTest,'Demandv2');
```

5.8 Internal functions

```
function DataOut = FilterLoc(DataIn, field, Threshold,nbin)
    ind = DataIn.(field) > Threshold(1) & DataIn.(field) < Threshold(2) ;
    DataOut = DataIn;DataOut = DataOut(ind,:);
    if 0
        figure()
        subplot(2,1,1)
        histogram(DataIn.(field),nbin);
        set(gca,'Yscale','log');
        axis tight;ym = axis; ym = ym(3:4);
        hold on
        plot([Threshold(1),Threshold(1)],ym,'r')
        plot([Threshold(2),Threshold(2)],ym,'r')
        hold off
        title("Before filtering " + field)
        xlabel(field)
        ylabel('pdf')
        subplot(2,1,2)
        histogram(DataOut.(field),nbin/100,'FaceColor',[1,0,0]);
        set(gca,'Yscale','log');axis tight
        title("After filtering " + field)
        xlabel(field)
        ylabel('pdf')
        axis tight
        hold off
    end
end

function DataOut = AddFeature(DataIn)
    DataOut = AddDemand(DataIn);
    DataOut = addDayOfWeekLoc(DataOut);
    DataOut = addTimeOfDay(DataOut);
    DataOut.DayOY = string(datestr(DataOut.HourlyBin,'mm/dd/yy'));
    DataOut.DayOYnum = datenum(DataOut.DayOY);
    % Add Holiday
    holiday = readtable("2015 Bank Holidays.csv");
    holiday.Date = string(datestr(holiday.Date,'mm/dd/yy'));
    DataOut.IsHoliday = repmat(false,height(DataOut),1);
    ind = repmat(false,height(DataOut),1);
    for ii = 1:height(holiday)
        ind = ind | DataOut.DayOY == string(holiday.Date(ii));
    end
    DataOut.IsHoliday(ind) = true;
    % Add Hour of Day
    DataOut.HourOfDay = str2num(datestr(DataOut.HourlyBin,'hh'));
end

function DataOut = AddDemand(DataIn)
    DataOut = DataIn;
    DataOut.Demand = repmat("Low",height(DataOut),1);
    DataOut.Demand(DataOut.NetPickups >= 0 & DataOut.NetPickups < 15,:) = "Medium";
    DataOut.Demand(DataOut.NetPickups > 15,:) = "High";
    DataOut.Demand = categorical(DataOut.Demand);
end

function DataOut = addDayOfWeekLoc(DataIn)
    DataOut = DataIn;
    % Get weekday in string format
    [~,DataOut.DayOfWeek] = weekday(DataOut.HourlyBin,"long");
    % Convert to categorical array
    DataOut.DayOfWeek = categorical(cellstr(DataOut.DayOfWeek));
    % AddWeekDay (true = Monday to Friday, false = Satruday SUnday)
    DataOut.WeekDay = repmat(true,height(DataOut),1);
    ind = DataOut.DayOfWeek=='Saturday';
    DataOut(ind,:).WeekDay = repmat(false,sum(ind),1);
    ind = DataOut.DayOfWeek=='Sunday';
    DataOut(ind,:).WeekDay = repmat(false,sum(ind),1);
end

function DataOut = addTimeOfDay(DataIn)
    DataOut = DataIn;
    DataOut.TimeOfDay = hours(timeofday(DataOut.HourlyBin));
    DataOut.DayPeriod = repmat("Night",height(DataOut),1);
    DataOut.DayPeriod(DataOut.TimeOfDay >= 6 & DataOut.TimeOfDay < 12,:) = "am";
    DataOut.DayPeriod(DataOut.TimeOfDay >= 12 & DataOut.TimeOfDay < 18,:) = "pm";
    DataOut.DayPeriod(DataOut.TimeOfDay >= 18,:) = "Evening";
    DataOut.DayPeriod = categorical(DataOut.DayPeriod);
end

function out = FilterFeatures(Predictors,ResponseNAMEorIND,varargin)
    options = inputParser(); options.addOptional('DummyCrap',1);
    options.addOptional('corrtype','Spearman');
    options.addOptional('p_relevance',0.05);% p value threshold for relevant response
    options.addOptional('disp_fig',false);
    parse(options,varargin{:}); options = rmfield(options.Results,'DummyCrap');
    % Get field names; convert struct to table
    try % works for tables and structures
        fname = Predictors.Properties.VariableNames;
    catch
        fname = fieldnames(Predictors);
        Predictors = struct2table(Predictors);
    end
    % Find if response is continuous or descrete
    for ii = 1:length(fname)
```

```
        dataloc = [unique(Predictors.(fname{ii}))];
        cond_string(ii) = isstring(dataloc);
        cond_catogorical(ii) = iscategorical(dataloc);
        cond_time(ii) = isdatetime(dataloc);
        cond_logical(ii) = islogical(dataloc);
    end
    isdouble = ~(cond_string | cond_catogorical | cond_time | cond_logical);
    % Split Predictors from response
    if isstring(ResponseNAMEorIND) || ischar(ResponseNAMEorIND)

    else
        ResponseNAMEorIND = fname{ResponseNAMEorIND};
    end
    Response = removevars(Predictors,fname(~strcmp(fname,ResponseNAMEorIND)));
    ResponseIsDouble = isdouble(strcmp(fname,ResponseNAMEorIND));
    Predictors = removevars(Predictors,{char(ResponseNAMEorIND)});
    isdouble = isdouble(~strcmp(fname,ResponseNAMEorIND));
    fname = fname(~strcmp(fname,ResponseNAMEorIND));
    % Display Response name
    clc
    if ResponseIsDouble
        disp(ResponseNAMEorIND + " is continuous")
    else
        disp(ResponseNAMEorIND + " is discreate")
    end
    % Loop over predictors
    % Display if features are relevant (according to p metric)
    out = struct.empty;
    for ii = 1:length(fname)
        dataloc = removevars(Predictors,fname(~strcmp(fname,fname{ii})));
        if ResponseIsDouble & isdouble(ii)
            [p,corrOut,figID] = ContinuousPredictorResponsePairs(Response,dataloc,options.corrt
            chi2 = nan;
        elseif ~ResponseIsDouble & ~isdouble(ii)
        % DiscretePredictorResponsePairs
            [chi2,p,Xtab,diffXtab,figID] = DiscretePredictorResponsePairs(Response,dataloc,opti
        else

            if ResponseIsDouble
                Cont = Response;
                Discr = dataloc;
            else
                Cont = dataloc;
                Discr = Response;
            end
        [p, groupSum,anovatab,stats,figID] = ContinuousDiscretePairs(Cont,Discr,options.disp_fi
        chi2 = nan;
        end
        % Display if parameter is relevant or not
        if p <= options.p_relevance
```

```
            disp([fname{ii},' is a relevant feature (p = ',num2str(min(p)),')']);
        elseif p > options.p_relevance
            disp([fname{ii},' is NOT a relevant feature (p = ',num2str(min(p)),')']);
        else
            disp('nan')
        end
        if ~isnan(chi2)
            disp([' chi2 = ',num2str(chi2)])
        end
        out = [out,struct('Feature',fname{ii},'p',p)];
    end
end

function [p, corrOut,figID] = ContinuousPredictorResponsePairs(Response,Predictors,corrtype,dis
    % Evaluating Continuous Predictor-Response Pairs
    % corrtype = 'pearson', 'spearman' or 'kendall'
    if nargin == 2
        corrtype = 'pearson';
        disp_fig = false;
    elseif nargin == 3
        disp_fig = false;
    end
    Varloc = {Response.Properties.VariableNames{:},Predictors.Properties.VariableNames{:}};
    [corrOut,p] = corr(table2array([Response,Predictors]),'type',corrtype);
    if disp_fig
        figID{1} = figure();
        gplotmatrix(table2array([Response,Predictors]),[],[],[],[],[],[],[],Varloc);
        figID{2} = figure();
        heatmap(Varloc,Varloc,corrOut,"ColorLimits",[-1 1],"Title",corrtype + " Correlation");
    else
        figID = [];
    end
end

function [chi2,p,Xtab,diffXtab,figID] = DiscretePredictorResponsePairs(Response,Predictor,disp_
    if width(Response) ~= 1
        error("Only one Reponse is allowed")
    end
    if width(Predictor) ~= 1
        error("Only one Predictor is allowed")
    end
    if nargin == 2
        disp_fig =false;
    end
    Var1 = Response.Properties.VariableNames{1};
    Var2 = Predictor.Properties.VariableNames{1};
    [Xtab,chi2,p] = crosstab(Response.(Var1),Predictor.(Var2)); % heat map and T are identical
    if disp_fig
        figID{1} = figure();
        heatmap([Response,Predictor],Var1,Var2)
    else
        figID = [];
    end
    m = height(Response);
    expT = sum(Xtab,1).*sum(Xtab,2)/m;
    diffXtab = Xtab-expT;
    if disp_fig
        figID{2} = figure();
        heatmap(unique(Predictor.(Var2)),unique(Response.(Var1)),diffXtab,"XLabel",Var2,"YLabel
    end
end

function [p, groupSum,anovatab,stats,figID] = ContinuousDiscretePairs(Cont,Discr,disp_fig)
    if width(Cont) ~= 1
        error("Only one Cont is allowed")
    end
    if width(Discr) ~= 1
        error("Only one Discr is allowed")
    end
    Var1 = Cont.Properties.VariableNames{1};
    Var2 = Discr.Properties.VariableNames{1};
    tbl = [Cont,Discr];
    groupSum = groupsummary(tbl,Var2,"all",Var1);
    if disp_fig
        figID{1} = figure();
        boxplot(tbl.(Var1),tbl.(Var2))
        xlabel(Var2,"FontSize",16)
        ylabel(Var1,"FontSize",16)
    else
        figID = [];
    end
    [p,anovatab,stats] = anova1(tbl.(Var1),tbl.(Var2));
    if disp_fig
        ylabel(Var1)
        xlabel(Var2)
    else
        close;close;
    end
end

function [trainedClassifier, validationAccuracy] = TrainBestModel(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 in percent. In
% the app, the History list 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 02-Jan-2023 12:04:20

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'Region', 'DayOfWeek', 'WeekDay', 'TimeOfDay', 'DayPeriod', 'DayOYnum', 
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;
isCategoricalPredictor = [true, true, true, false, true, false, true, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
template = templateTree(...
            'MaxNumSplits', 427, ...
            'NumVariablesToSample', 4);
classificationEnsemble = fitcensemble(...
            predictors, ...
            response, ...
            'Method', 'Bag', ...
            'NumLearningCycles', 471, ...
```

```
                'Learners', template, ...
                'ClassNames', categorical({'High'; 'Low'; 'Medium'}));

    % Create the result struct with predict function
    predictorExtractionFcn = @(t) t(:, predictorNames);
    ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
    trainedClassifier.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));

    % Add additional fields to the result struct
    trainedClassifier.RequiredVariables = {'DayOYnum', 'DayOfWeek', 'DayPeriod', 'HourOfDay', '
    trainedClassifier.ClassificationEnsemble = classificationEnsemble;
    trainedClassifier.About = 'This struct is a trained model exported from Classification Lear
    trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n  y

    % Create the result struct with predict function
    predictorExtractionFcn = @(t) t(:, predictorNames);
    ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
    trainedClassifier.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));

    % Extract predictors and response
    % This code processes the data into the right shape for training the
    % model.
    inputTable = trainingData;
    predictorNames = {'Region', 'DayOfWeek', 'WeekDay', 'TimeOfDay', 'DayPeriod', 'DayOYnum', '
    predictors = inputTable(:, predictorNames);
    response = inputTable.Demand;
    isCategoricalPredictor = [true, true, true, false, true, false, true, false];
    % Perform cross-validation
    partitionedModel = crossval(trainedClassifier.ClassificationEnsemble, 'KFold', 20);
    % Compute validation predictions
    [validationPredictions, validationScores] = kfoldPredict(partitionedModel);
    % Compute validation accuracy
    validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
end

function [t,a,C,featureImp] = GetModelFOM(Model,Data,field)
    try
        impValues = predictorImportance(Model.ClassificationTree);
        nameloc = 'ClassificationTree';
    catch
        impValues = predictorImportance(Model.ClassificationEnsemble);
        nameloc = 'ClassificationEnsemble';
    end
    % Model performance
    [Predict,score] = Model.predictFcn(Data);
    [t,a] = cMetrics(Data.(field),Predict);
    % Create ROC
    fname = Model.(nameloc).ClassNames;
    figure(),hold all, grid on, box on
    for ii = 1:length(fname)
```

```
            [X,Y,T,AUC] = perfcurve(Data.(field),score(:,ii),fname(ii));
            ledg{ii} = [char(fname(ii)),' (AUC = ',num2str(AUC),')'];
            plot(X,Y)
            xlabel(['False positive rate (Fallout)'])
            ylabel(['True positive rate (Recall)'])
            title('ROC')
        end
        for ii = 1:length(fname)
            rowloc = find(t.Row == fname(ii));
            plot(t(rowloc,:).Fallout,t(rowloc,:).Recall,'.r','markersize',15)
        end
        legend(ledg)
        % Displau metrics
        head(t)
        order = unique(Predict);
        figure()
        C = confusionchart(confusionmat(Data.(field),Predict,'order',order),order);
        % Embedded Methods
        figure()
        pareto(impValues)
        disp(['Feature relevance in descending order'])
        [~,ind] = sort(impValues);
        featureImp = struct.empty;
        for ii = 1:length(ind)
            featureImp = [featureImp,struct('Feature',Model.(nameloc).PredictorNames{ind(ii)},...
            'Val',impValues(ind(ii)),...
            'NormVal',impValues(ind(ii))/sum(impValues))];
        end
        featureImp = struct2table(featureImp);
        head(featureImp,height(featureImp))
    end

    function [trainedClassifier, validationAccuracy] = MinimizeLossModel_3Demands(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 in percent. In
        % the app, the History list 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 07-Jan-2023 09:09:43

         Extract predictors and response
        % This code processes the data into the right shape for training the
        % model.
        inputTable = trainingData;
        predictorNames = {'Region', 'DayOfWeek', 'WeekDay', 'TimeOfDay', 'DayPeriod', 'DayOYnum',
        response = inputTable.Demand;
        isCategoricalPredictor = [true, true, true, false, true, false, true, false];
        % Train a classifier
        % This code specifies all the classifier options and trains the classifier.
        template = templateTree(...
            'MaxNumSplits', 301, ...
            'NumVariablesToSample', 'all');
        classificationEnsemble = fitcensemble(...
            predictors, ...
            response, ...
            'Method', 'Bag', ...
            'NumLearningCycles', 256, ...
            'Learners', template, ...
            'Cost', [0 4 1; 6 0 6; 1 4 0], ...
            'ClassNames', categorical({'High'; 'Low'; 'Medium'}));

        % Create the result struct with predict function
        predictorExtractionFcn = @(t) t(:, predictorNames);
        ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
        trainedClassifier.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));
        % Add additional fields to the result struct
        trainedClassifier.RequiredVariables = {'DayOYnum', 'DayOfWeek', 'DayPeriod', 'HourOfDay',
        trainedClassifier.ClassificationEnsemble = classificationEnsemble;
        trainedClassifier.About = 'This struct is a trained model exported from Classification Lea
        trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n  y
        % Extract predictors and response
        % This code processes the data into the right shape for training the
        % model.
        inputTable = trainingData;
        predictorNames = {'Region', 'DayOfWeek', 'WeekDay', 'TimeOfDay', 'DayPeriod', 'DayOYnum', 
        response = inputTable.Demand;
        isCategoricalPredictor = [true, true, true, false, true, false, true, false];
        % Perform cross-validation
        partitionedModel = crossval(trainedClassifier.ClassificationEnsemble, 'KFold', 20);
        % Compute validation predictions
        [validationPredictions, validationScores] = kfoldPredict(partitionedModel);
        % Compute validation accuracy
        validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
    end

    function [trainedClassifier, validationAccuracy] = MinimizeLossModel_2Demands(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 in percent. In
        % the app, the History list 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 07-Jan-2023 16:09:16

        % Extract predictors and response
        % This code processes the data into the right shape for training the
        % model.
        inputTable = trainingData;
        predictorNames = {'Region', 'DayOfWeek', 'WeekDay', 'TimeOfDay', 'DayPeriod', 'DayOYnum', 
        response = inputTable.Demandv2;
        isCategoricalPredictor = [true, true, true, false, true, false, true, false];
        % Train a classifier
        % This code specifies all the classifier options and trains the classifier.
        template = templateTree(...
            'MaxNumSplits', 248, ...
            'NumVariablesToSample', 'all');
        classificationEnsemble = fitcensemble(...
            predictors, ...
            response, ...
            'Method', 'Bag', ...
            'NumLearningCycles', 497, ...
            'Learners', template, ...
            'Cost', [0 2; 1 0], ...
            'ClassNames', categorical({'Low'; 'MedHigh'}));
        % Create the result struct with predict function
        predictorExtractionFcn = @(t) t(:, predictorNames);
        ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
        trainedClassifier.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));
        % Add additional fields to the result struct
        trainedClassifier.RequiredVariables = {'DayOYnum', 'DayOfWeek', 'DayPeriod', 'HourOfDay', 
        trainedClassifier.ClassificationEnsemble = classificationEnsemble;
        trainedClassifier.About = 'This struct is a trained model exported from Classification Lea
        trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n  
        % Extract predictors and response
        % This code processes the data into the right shape for training the
        % model.
        inputTable = trainingData;
        predictorNames = {'Region', 'DayOfWeek', 'WeekDay', 'TimeOfDay', 'DayPeriod', 'DayOYnum', 
        predictors = inputTable(:, predictorNames);
        response = inputTable.Demandv2;
        isCategoricalPredictor = [true, true, true, false, true, false, true, false];
        % Perform cross-validation
        partitionedModel = crossval(trainedClassifier.ClassificationEnsemble, 'KFold', 20);
        % Compute validation predictions
        [validationPredictions, validationScores] = kfoldPredict(partitionedModel);
        % Compute validation accuracy
        validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
    end
```