Taxi Deployment

Table of Contents

Data Extraction and Introduction

To get started, navigate to the Taxi Project folder, and run the script **generateTaxiPickupTable.mlx**.

Note it may take a while to finish.

This will create (and save) two tables: `pickupLocations` and `taxiPickups`. A preview and description of each table is given below.

Data Exploration and Partitioning

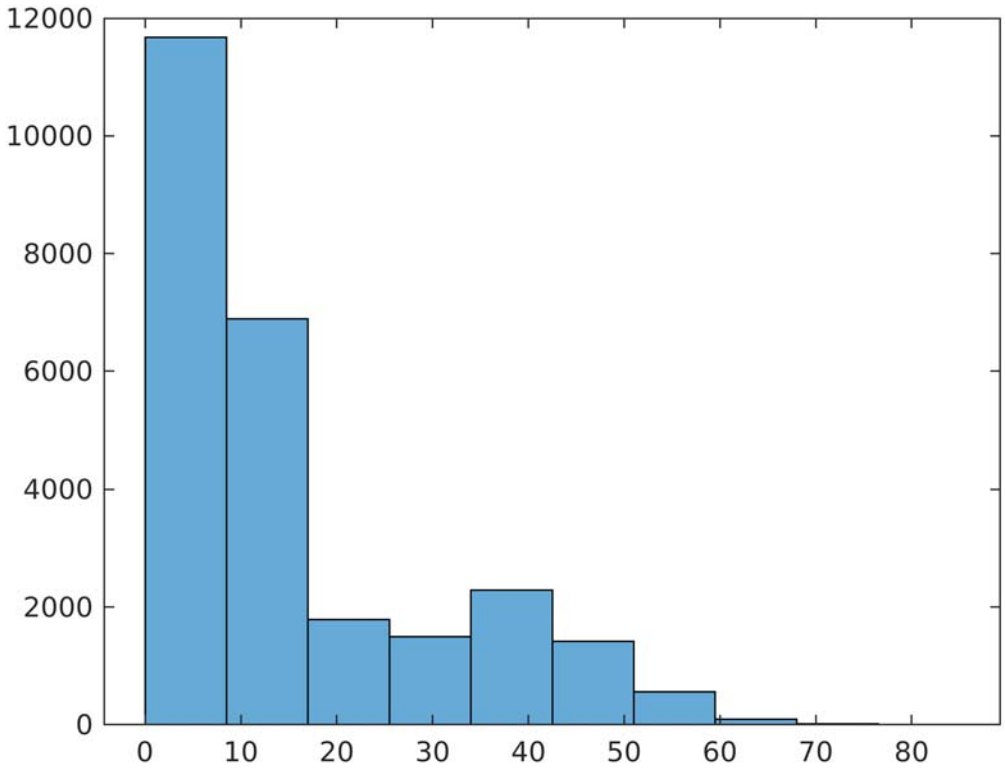
The script **generateTaxiPickupTable.mlx** in the previous section saved copies of the tables to a MAT file named **taxiPickupData.mat**. To avoid having to run the script again if you continue working after clearing your workspace, the code below loads the saved data.

```
% Make sure to follow the instructions in the previous section
if ~isempty(which('-all','taxiPickupData.mat'))
    load taxiPickupData.mat
else
    error("The file taxiPickupData.mat is not found on the MATLAB path. Add it to the p
end
```

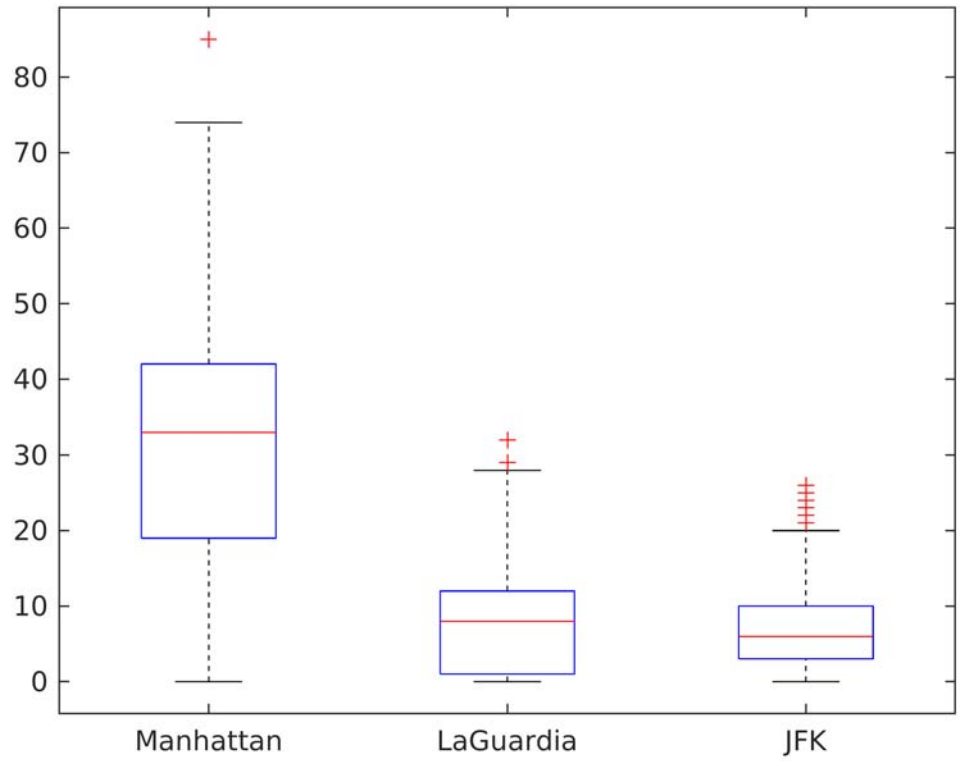
Visualization and Analysis

Analyze the data in the `taxiPickups` table. At a minimum, provide a visualization of the distribution (histogram) of the response variable `TripCount`, as well as a box plot for `TripCount` grouped by `Location`.

```
histogram(taxiPickups.TripCount,10)
```




```
boxplot(taxiPickups.TripCount, taxiPickups.Location)
```



Separate Test Data

Use `cvpartition` to separate 20% of the data set for testing later on, and create the training data. Ensure your results are repeatable by setting the random number generator seed to 10. Provide your code.

```
rng(10)
taxipartition = cvpartition(height(taxiPickups),"HoldOut", 0.2);
```

```
taxiTestIdx = test(taxipartition);

taxiTest =taxiPickups(taxiTestIdx,:);

taxitrainIdx = training(taxipartition);

taxiTrain = taxiPickups(taxitrainIdx,:);
```

Models Training and Validation

Preprocessing

As the focus of this course is machine learning, we've provided a function to do some feature engineering for you. Use **providedPreprocessing.mlx** to add the following features to your training/validation data set:

- `TimeOfDay` (numerical)

- `DayOfWeek` (categorical)
- `DayOfMonth` (numerical)
- `DayOfYear` (numerical)

For example:

```
taxiPickupsTrain = providedPreprocessing(taxiPickupsTrain)
```

```
taxiTrain = providedPreprocessing(taxiTrain);
taxiTrain = addTimeOfDay(taxiTrain);
taxiTrain = addDayOfWeek(taxiTrain);
head(taxiTrain)
```

```
ans = 8×7 table
```

	PickupTime	Location	TripCount	TimeOfDay	DayOfWeek	DayOfMonth
1	2015-01-01 ...	Manhattan	22	0	Thursday	1
2	2015-01-01 ...	LaGuardia	2	0	Thursday	1
3	2015-01-01 ...	JFK	2	0	Thursday	1
4	2015-01-01 ...	Manhattan	10	1	Thursday	1
5	2015-01-01 ...	JFK	2	1	Thursday	1
6	2015-01-01 ...	Manhattan	14	2	Thursday	1
7	2015-01-01 ...	LaGuardia	0	2	Thursday	1
8	2015-01-01 ...	JFK	0	2	Thursday	1

Model Training

Train models to predict `TripCount` using the processed test/validation data. **Report your validation approach and validation $RMSE$ for your best model**. Your goal will be to get a validation $RMSE$ at or below 4.9. Include code so that your script can reproduce your final model, including the model training. **If using the app, export the training function** and include a correct call to it in your script. Note you do not need to include the generated training function code itself in your script, just a correct call to it. For example, if you'd trained a tree model in the app and exported the training function, you could include:

```
[modelStruct, validationRMSE] = trainRegressionModel(taxiPickupsTrain)
myModel = modelStruct.RegressionTree
```

```
[trainedModel, validationRMSE] = taxiRegressionModel(taxiTrain)
```

```
trainedModel = struct with fields:
            predictFcn: @(x)ensemblePredictFcn(predictorExtractionFcn(x))
     RequiredVariables: {'DayOfMonth'  'DayOfWeek'  'DayOfYear'  'Location'  'TimeOfDay'}
```

```
    RegressionEnsemble: [1×1 classreg.learning.regr.RegressionEnsemble]
                 About: 'This struct is a trained model exported from Regression Learner R2020a.'
           HowToPredict: 'To make predictions on a new table, T, use: ↲  yfit = c.predictFcn(T) ↲replacing 'c
validationRMSE = 4.7121
```

```
myModel = trainedModel.RegressionEnsemble
```

```
myModel =
  RegressionEnsemble
            PredictorNames: {'Location'  'TimeOfDay'  'DayOfWeek'  'DayOfMonth'  'DayOfYear'}
              ResponseName: 'Y'
     CategoricalPredictors: [1 3]
         ResponseTransform: 'none'
           NumObservations: 20974
                NumTrained: 454
                    Method: 'LSBoost'
              LearnerNames: {'Tree'}
      ReasonForTermination: 'Terminated normally after completing the requested number of training cycles.'
                   FitInfo: [454×1 double]
        FitInfoDescription: {2×1 cell}
            Regularization: []

  Properties, Methods
```

Model Testing and Evaluation

Testing

Preprocess the test data as needed, and use it to test your best model. Provide your code and report at least the $RMSE$ and R^2. You will need to achieve a test $RMSE$ at or below 4.9 to receive full points here.

```
taxiTest = providedPreprocessing(taxiTest);
taxiTest = addDayOfWeek(taxiTest);
taxiTest = addTimeOfDay(taxiTest);
yActual = taxiTest.TripCount;
yPredicted = trainedModel.predictFcn(taxiTest);

residuals = yActual - yPredicted;

testMetrics = rMetrics(yActual, yPredicted)
```

```
testMetrics = 1×4 table
```

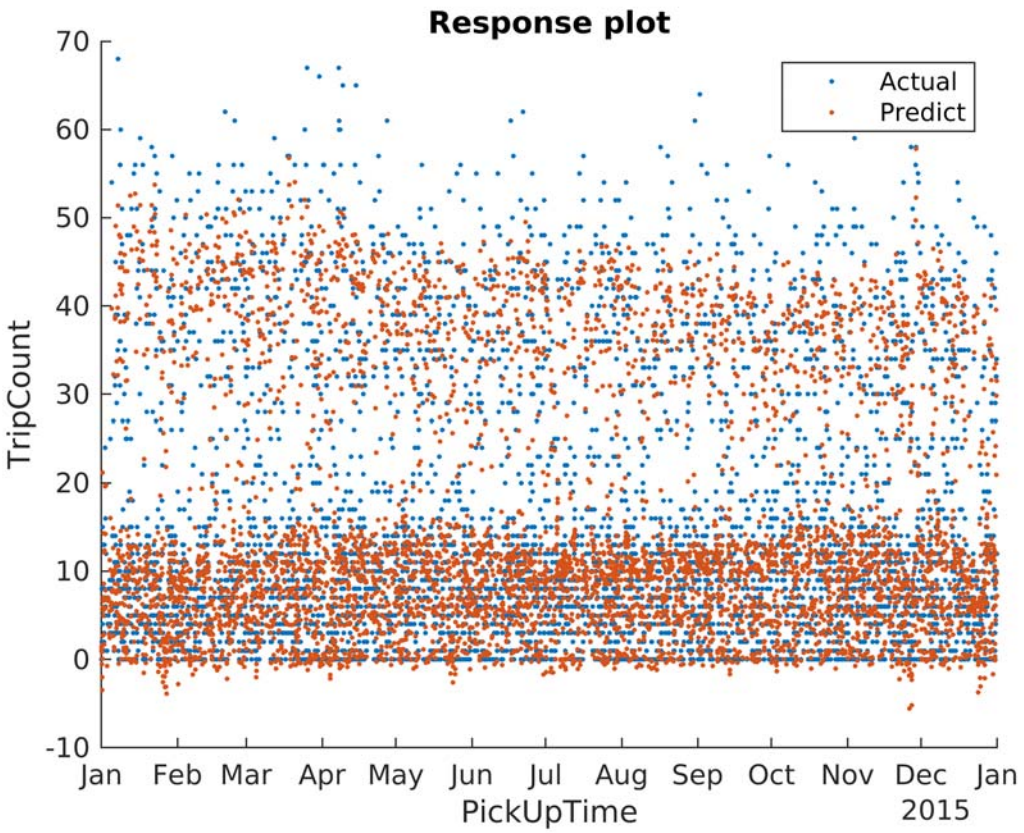
	mae	mse	rmse	Rsq
1	3.2262	20.4891	4.5265	0.9086

Evaluation

Discuss the results from training and testing. How well did your model generalize to new data? Include at least a plot of the residuals, and discuss your observations from the plot(s).

```
scatter(taxiTest.PickupTime, yActual, ".")
hold on
scatter(taxiTest.PickupTime, yPredicted, ".")
hold off
legend("Actual", "Predict")
xlabel("PickUpTime")
```

```
ylabel("TripCount")
title("Response plot")
```



From the testMetrics above, it is clear that the model generalises well without overfitting.

Also from the scatter plot, there is high correlation between the actual and the predicted variables.

We can conclude that the model serves well.

Model Application, Results, and Analysis

Apply Model

As the focus of this course is machine learning, we've provided some skeleton code in this section. Below, we create a new table of staring features for 2016.

Note that you will need to use the variable names provided in comments or make your own additional edits to the code.

```
taxiPickups2020 = table;
taxiPickups2020.PickupTime = taxiPickups.PickupTime + years(5);
taxiPickups2020.Location = taxiPickups.Location;
taxiPickups2020 = providedPreprocessing(taxiPickups2020);
% Display only the first 8 rows of the table
head(taxiPickups2020)
```

```
ans = 8×6 table
```

	PickupTime	Location	TimeOfDay	DayOfWeek	DayOfMonth	DayOfYear
1	2020-01-01 ...	Manhattan	5.1000	Wednesday	1	1
2	2020-01-01 ...	LaGuardia	5.1000	Wednesday	1	1
3	2020-01-01 ...	JFK	5.1000	Wednesday	1	1
4	2020-01-01 ...	Manhattan	6.1000	Wednesday	1	1
5	2020-01-01 ...	LaGuardia	6.1000	Wednesday	1	1
6	2020-01-01 ...	JFK	6.1000	Wednesday	1	1
7	2020-01-01 ...	Manhattan	7.1000	Wednesday	1	1
8	2020-01-01 ...	LaGuardia	7.1000	Wednesday	1	1

Choose your favorite day in 2016 and edit the variable `myDay` below which will be used to extract that day from the table.

```
myDay = datetime("2020-12-6")
```

```
myDay = datetime

   06-Dec-2020
```

```
taxiPickupsMyDay = taxiPickups2020(day(taxiPickups2020.PickupTime,"dayofyear") == day(m
```

Now, use your best model to predict `TripCount` on the day you've chosen and add it to the table.

```
TripCountPred = trainedModel.predictFcn(taxiPickupsMyDay);
TripCountPred = table(TripCountPred);
% Join tables
taxiPickupsMyDay1 = [taxiPickupsMyDay, TripCountPred];
```

Again, to focus on machine learning, we have provided the necessary table manipulations and calculations below to use your model predictions to give the predicted fraction of trips happening in each hour on your selected day.

Uncomment below once you have defined the table `taxiPickupsMyDay` above.

```
taxiPickupsMyDayTotals = groupsummary(taxiPickupsMyDay1,"PickupTime","sum","TripCountPr
```

```
taxiPickupsMyDayTotals = 24×3 table
```

	PickupTime	GroupCount	sum_TripCountPred
1	2020-12-06 ...	3	32.9978
2	2020-12-06 ...	3	30.1490
3	2020-12-06 ...	3	20.2459
4	2020-12-06 ...	3	15.4552
5	2020-12-06 ...	3	12.6414
6	2020-12-06 ...	3	11.3161

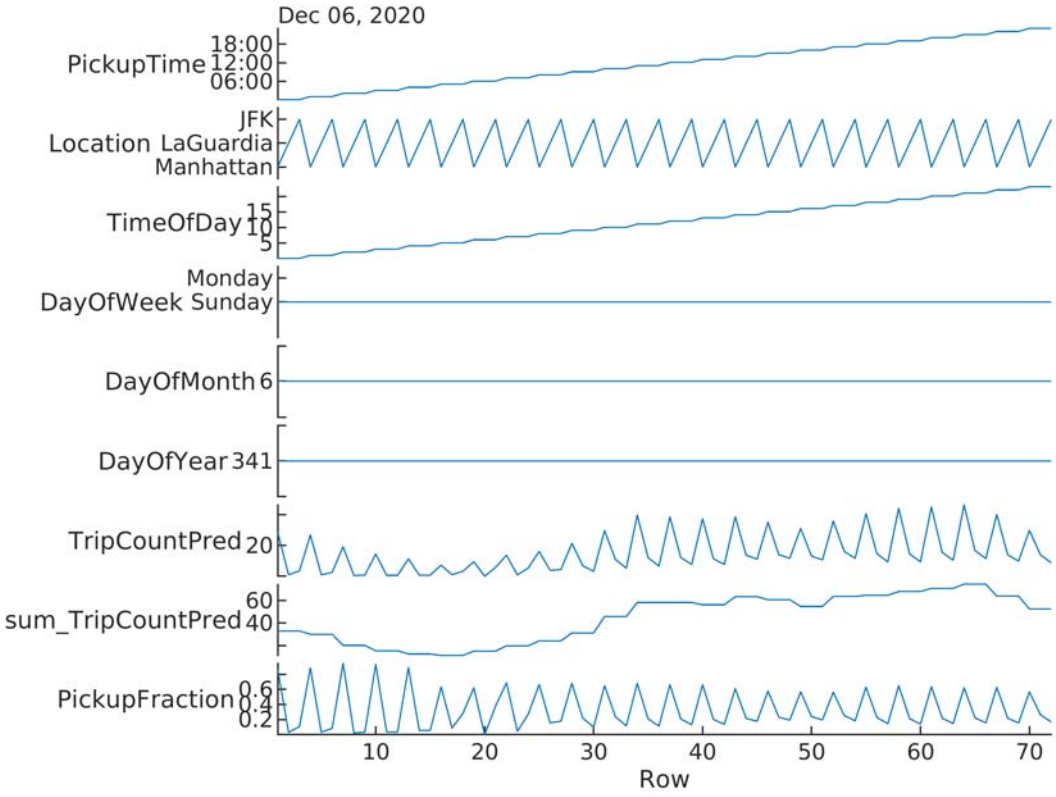
	PickupTime	GroupCount	sum_TripCountPred
7	2020-12-06 ...	3	15.1798
8	2020-12-06 ...	3	19.8532
9	2020-12-06 ...	3	24.2006
10	2020-12-06 ...	3	31.3430
11	2020-12-06 ...	3	45.8025
12	2020-12-06 ...	3	58.2787
13	2020-12-06 ...	3	58.2868
14	2020-12-06 ...	3	56.2156
15	2020-12-06 ...	3	63.4010
16	2020-12-06 ...	3	60.7305
17	2020-12-06 ...	3	54.7478
18	2020-12-06 ...	3	63.5941
19	2020-12-06 ...	3	64.6038
20	2020-12-06 ...	3	67.9770
21	2020-12-06 ...	3	70.7938
22	2020-12-06 ...	3	74.4874
23	2020-12-06 ...	3	63.8718
24	2020-12-06 ...	3	52.5175

```
taxiPickupsMyDay1 = join(taxiPickupsMyDay1,taxiPickupsMyDayTotals,"RightVariables","sum
taxiPickupsMyDay1.PickupFraction = taxiPickupsMyDay1.TripCountPred./taxiPickupsMyDay1.s
taxiPickupsMyDayFractions = unstack(taxiPickupsMyDay1,"PickupFraction","Location","Grou
```

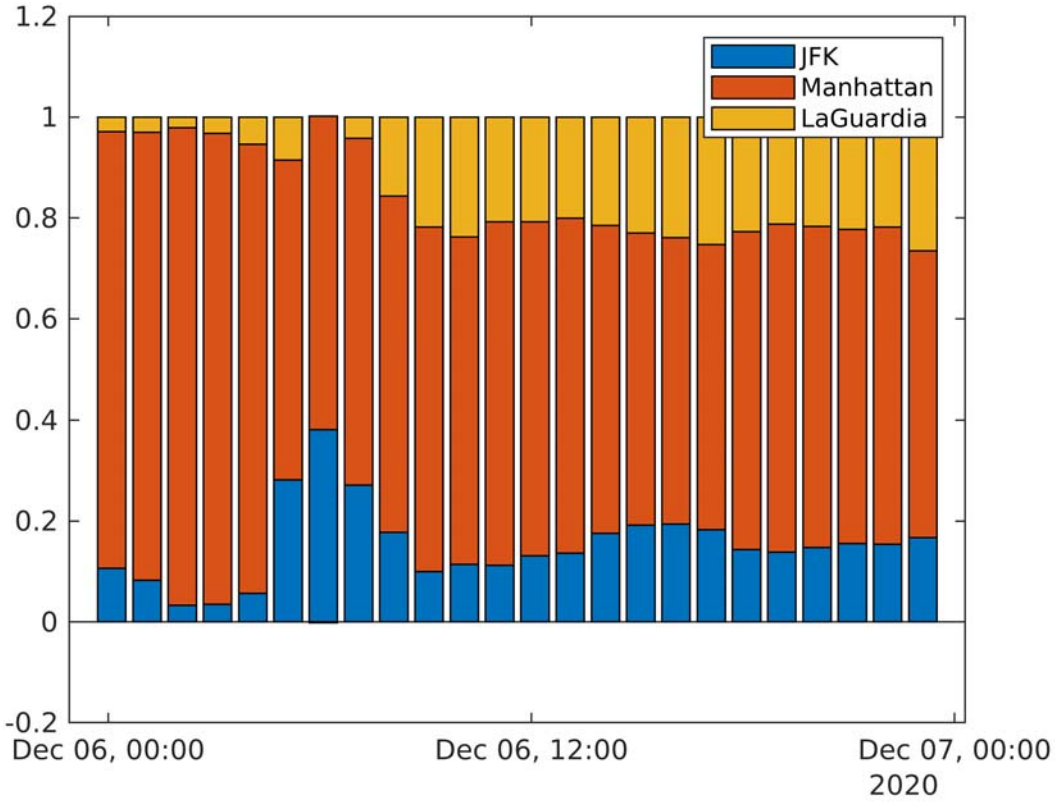
Use the Results to Allocate Fleet

Now it is time to present to Mr. Walker. Discuss the results you were able to obtain, and provide recommendations how you would allocate the fleet of taxis on the chosen day. Provide your reasoning, and also present your case using at least one visualization, e.g. a stacked bar plot.

```
stackedplot(taxiPickupsMyDay1)
```




```
bar(taxiPickupsMyDayFractions.PickupTime, [taxiPickupsMyDayFractions.JFK, taxiPickupsMy
legend('JFK','Manhattan','LaGuardia')
```



From the result obtained from the predictions and the stacked box plots, observations show that Manhattan has highest probability of getting much trip on the chosen day.

I will advice Mr Walker to allocate more Taxis to Manhattan.

Sending vehicles to JFK may bring have some trips. But since profit maximising is the aim, all vehicles should be allocated to Manhattan only.