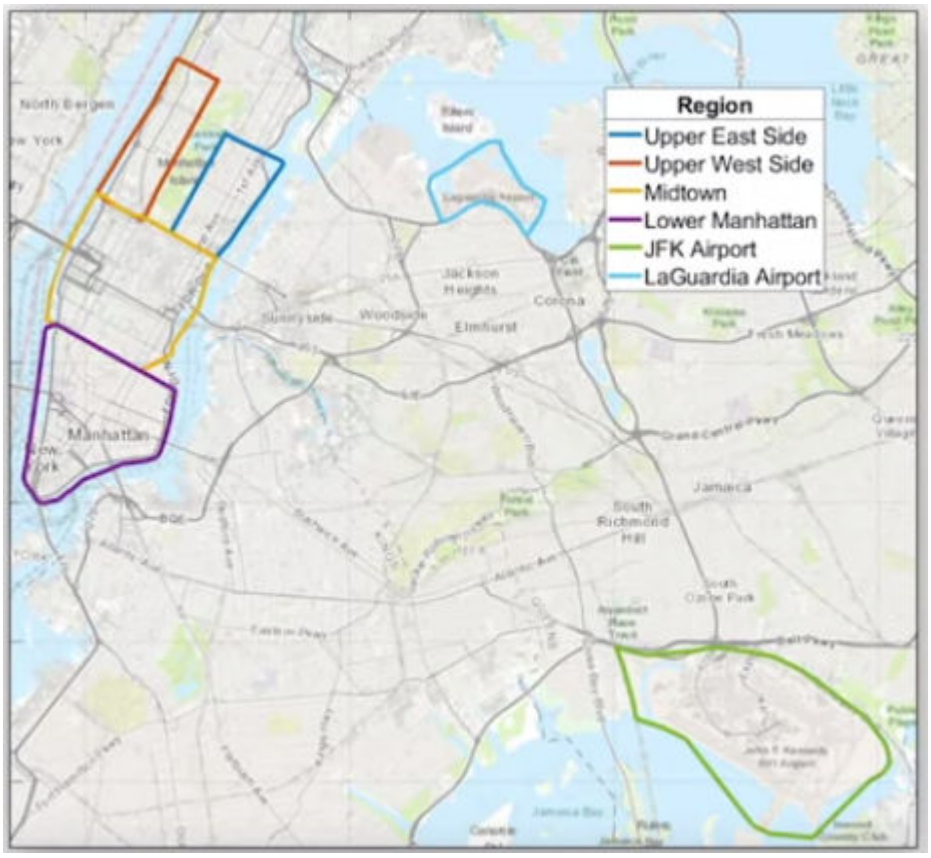
Taxi Demand in 6 Region of New York City

Table of Contents

Objectives

The objective of this project is to develop a model for forecasting of location among 6 regions of New York City and time for taxi deployment to maximize the profit along with cost and revenue for false negative and false positive prediction of high demand on test data .



Data

Brief description of the initial data.

Source of data is 12 CSV files of Yellow taxi trip data (one file for each month) for New York city during year 2015. These 12 CSV files are Imported and combined all into a single table using the import function **importTaxiDataWithoutCleaning** from Course 3.

```
taxiDatastore = fileDatastore("/MATLAB Drive/Predictive Modeling and Machine Learning/T
```

```
taxiDatastore =
  FileDatastore with properties:

                      Files: {
                             ' .../Taxi Data/yellow_tripdata_2015-01.csv';
                             ' .../Taxi Data/yellow_tripdata_2015-02.csv';
                             ' .../Taxi Data/yellow_tripdata_2015-03.csv'
                              ... and 9 more
                             }
                    Folders: {
                             '/MATLAB Drive/Predictive Modeling and Machine Learning/Taxi Data'
                             }
                UniformRead: 1
                   ReadMode: 'file'
                  BlockSize: Inf
                PreviewFcn: @importTaxiDataWithoutCleaning
      SupportedOutputFormats: ["txt"    "csv"    "xlsx"    "xls"    "parquet"    "parq"    "png"    "jpg"
                     ReadFcn: @importTaxiDataWithoutCleaning
    AlternateFileSystemRoots: {}
```

Summary of

1. data cleaning

Since the minimum **fare** allowed for a taxi trip in 2015 was **$2.50 accordingly the trip with fare less the 2.5 was removed from data.**

It was further checked that such removed data are contribution only 8% of total trip data which are not subsential data.

It was also checked average fare are in some of trips are abnormally high accordingly upper 2 percentile trips are further removed.

The code for reading and cleaning taxi data is annexed herewith.

```
figure;
[~,stdlow,stdhigh,~] = isoutlier(taxiAll.avgFare1,"mean")
```

```
stdlow = -0.5054
stdhigh = 11.7958
```

```
[~,medlow,medhigh,~] = isoutlier(taxiAll.avgFare1,"median")
```

```
medlow = -0.1937
medhigh = 10.7493
```

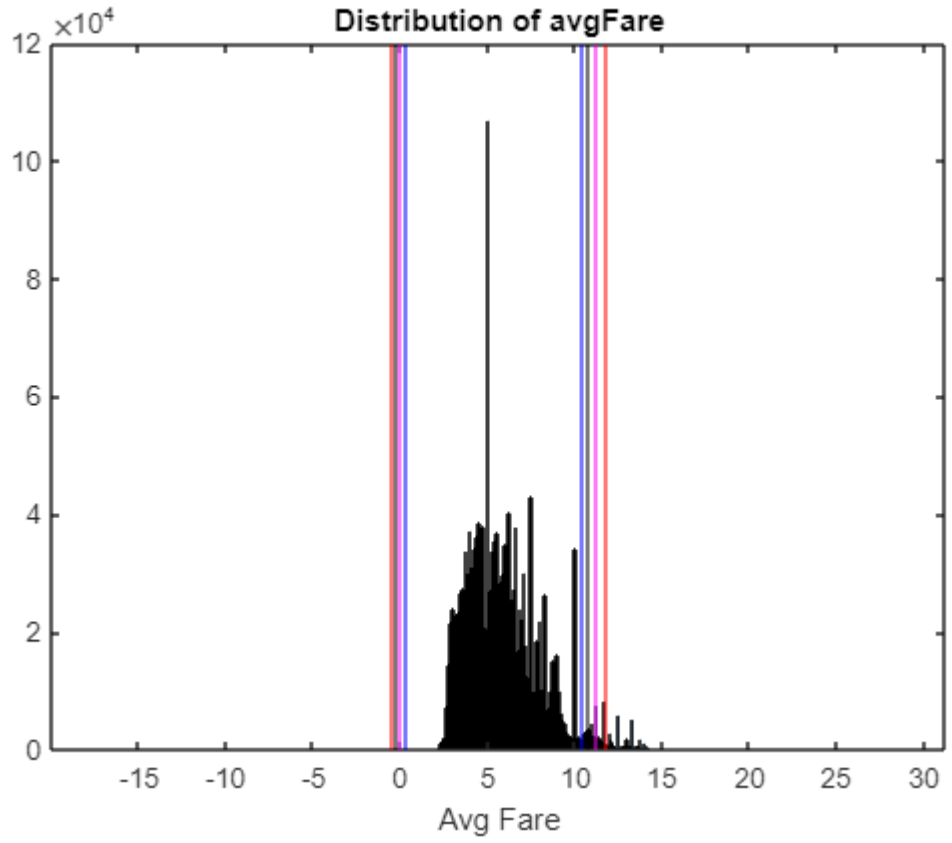
```
[~,quartlow,quarthigh,~] = isoutlier(taxiAll.avgFare1,"quartiles")
```

```
quartlow = 0.3687
quarthigh = 10.4455
```

```
[~,plow,phigh,~] = isoutlier(taxiAll.avgFare1,"percentiles", [0,98])
```

```
plow = 7.8337e-07
phigh = 11.2500
```

```
figure;
histogram(taxiAll.avgFare1)
title("Distribution of avgFare");
xlabel("Avg Fare")
xline(stdlow,"r"); xline(stdhigh,"r")
xline(quartlow,"b"); xline(quarthigh,"b")
xline(medlow,"k"); xline(medhigh,"k")
xline(plow,"m"); xline(phigh,"m")
xlim([plow-20 phigh+20])
```



2. data restructuring

Create the variables **PickupZone** and **DropoffZone**

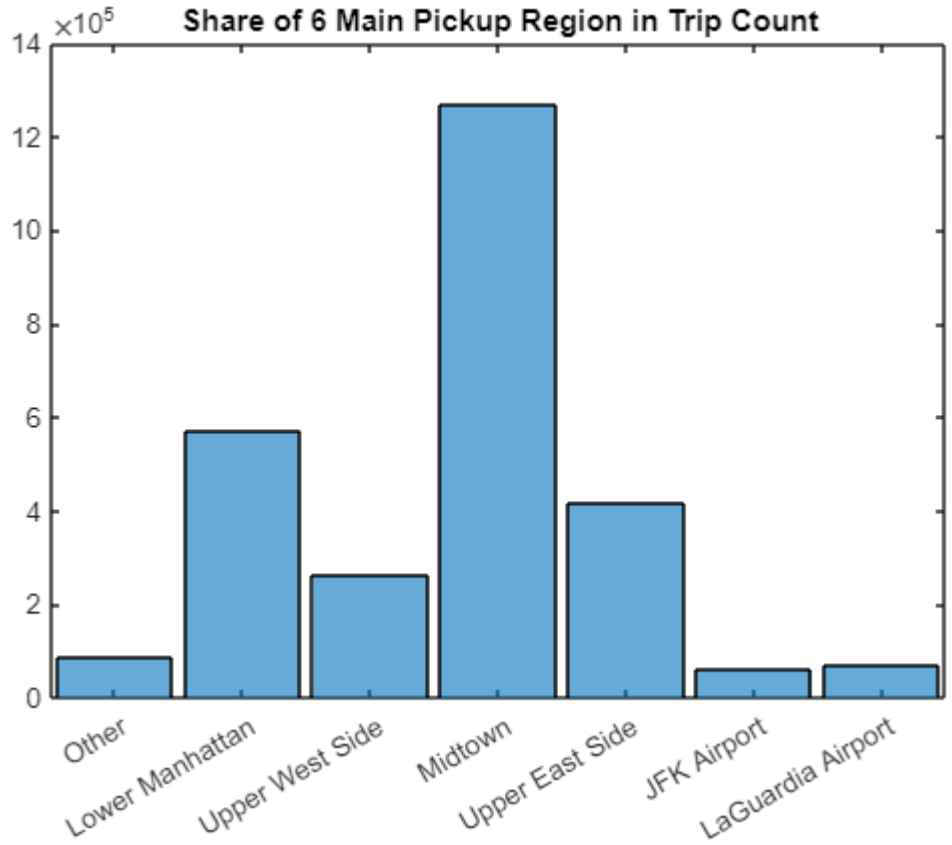
- for each taxi trip using Use the addTaxiZones function provided in Course 3.
- Assign two new variables to each taxi trip, corresponding to their **Pickup Region** and **Drop-off Region**

The code for these data restructuring is annexed herewith.

3. Feature Engineering

Exploring the Taxi Data

```
histogram(taxiAll2.PickupRegion);
title("Share of 6 Main Pickup Region in Trip Count");
```




```
taxisummaryDistance = groupsummary(taxiAll2,"PickupRegion","median","Distance")
```

taxisummaryDistance = 7×3 table

	PickupRegion	GroupCount	median_Distance
1	Other	87652	2.6100
2	Lower Manhattan	572655	2.0200
3	Upper West Side	262459	1.7000
4	Midtown	1267304	1.6000
5	Upper East Side	415534	1.5500
6	JFK Airport	60135	17.6000
7	LaGuardia Airport	70247	9.7600

```
taxisummaryFare = groupsummary(taxiAll2,"PickupRegion","median","Fare")
```

taxisummaryFare = 7×3 table

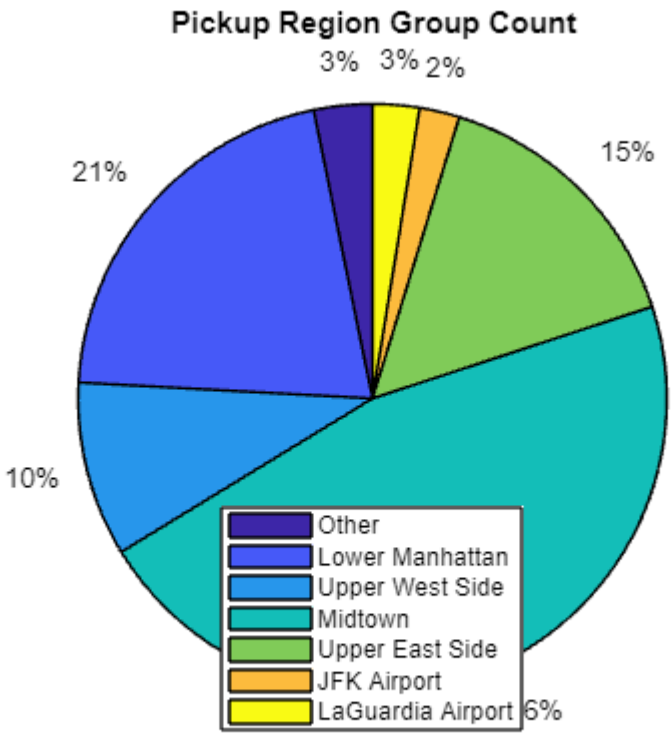
	PickupRegion	GroupCount	median_Fare
1	Other	87652	12.5000
2	Lower Manhattan	572655	10.5000
3	Upper West Side	262459	9
4	Midtown	1267304	9

	PickupRegion	GroupCount	median_Fare
5	Upper East Side	415534	8.5000
6	JFK Airport	60135	52
7	LaGuardia Airport	70247	31.5000

```
labels2 =cellstr(table2array(taxisummaryFare(:,1)))
```

```
labels2 = 7×1 cell
'Other'
'Lower Manhattan'
'Upper West Side'
'Midtown'
'Upper East Side'
'JFK Airport'
'LaGuardia Airport'
```

```
pie(taxisummaryFare.GroupCount);
title("Pickup Region Group Count");
legend(labels2,Location="south");
```



Under feature engineering following important task were performed.

- Creating DropoffTimeGroup, PickupTimeGroup and Trip Duration
- Creating the Grouped Summary Table

- Generating a categorical response feature, ‘Demand’ using the difference between pickups and drop-offs for each region and hour The categories of *Demand* are defined as below

1. Net Pickups < 0: ‘Low’
2. 0 <= Net Pickups < 15: ‘Medium’
3. Net Pickups >= 15: ‘High’

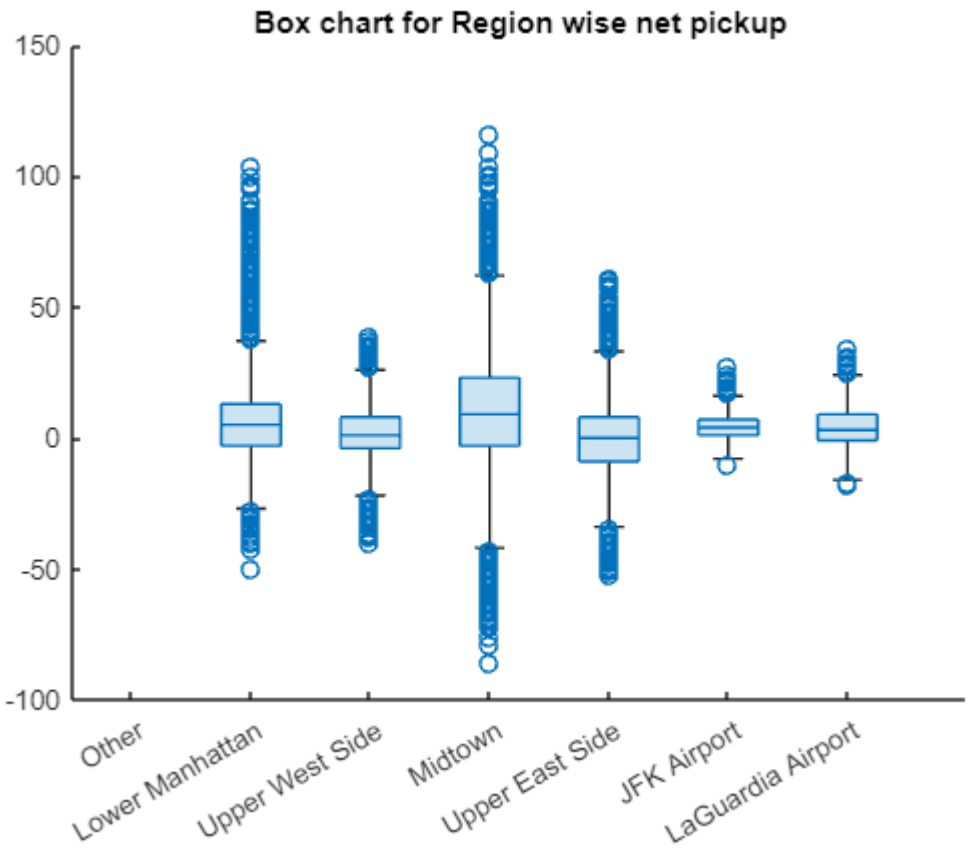
- Generating set of following predictors

1. dayOfYear
2. HoursOfDay
3. IsHoliday

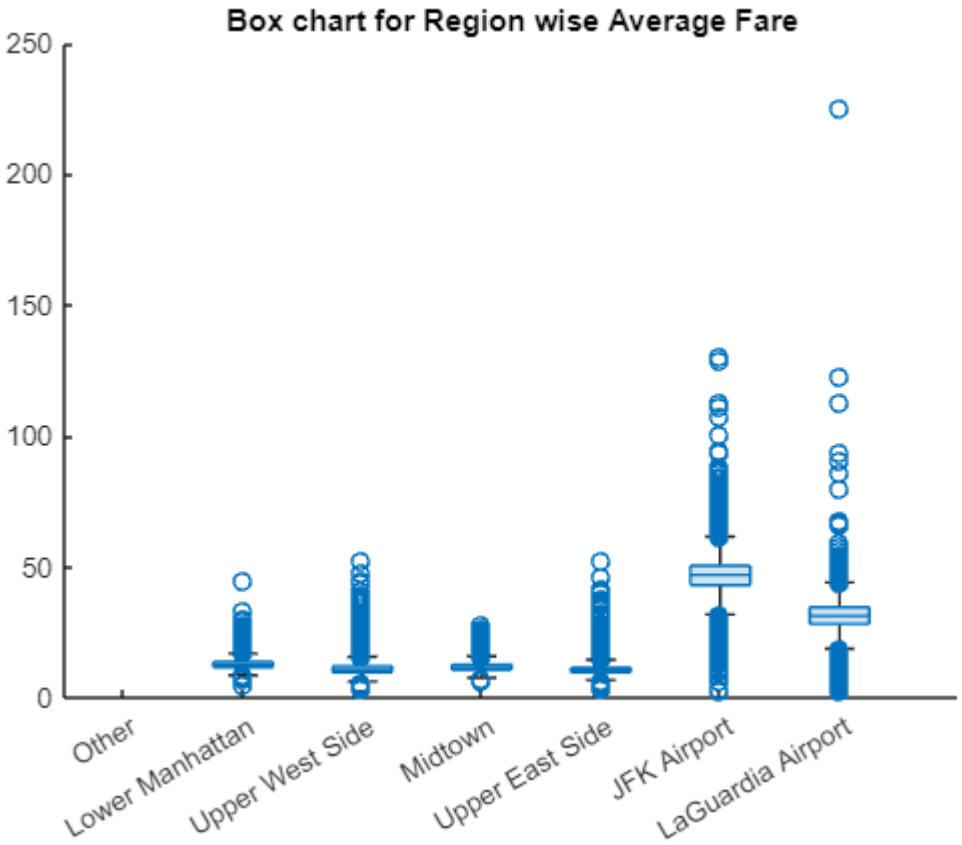
The code for feature engineering is annexed herewith.

Relevant exploration results

```
boxchart(GroupedSummaryTable.Region,GroupedSummaryTable.NetPickups);
title("Box chart for Region wise net pickup");
```




```
boxchart(GroupedSummaryTable.Region,GroupedSummaryTable.AvgFare)
title("Box chart for Region wise Average Fare");
```




```
DemandsummaryTable = groupsummary(GroupedSummaryTable,"Demand")
```

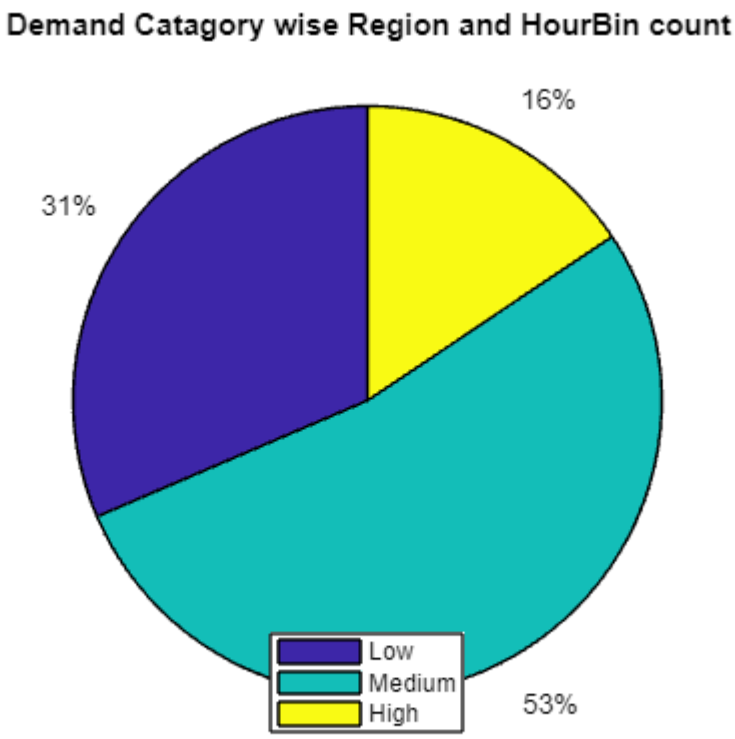
DemandsummaryTable = 3×2 table

	Demand	GroupCount
1	Low	16525
2	Medium	27816
3	High	8201

```
labels3 =cellstr(table2array(DemandsummaryTable(:,1)))
```

```
labels3 = 3×1 cell
'Low'
'Medium'
'High'
```

```
pie(DemandsummaryTable.GroupCount)
legend("Show","Location","south")
title("Demand Catagory wise Region and HourBin count");
legend(labels3,Location="south");
```




```
DemandRegiosummaryTable = groupsummary(GroupedSummaryTable,["Demand",'Region'])
```

DemandRegiosummaryTable = 18×3 table

	Demand	Region	GroupCount
1	Low	Lower Manhattan	2771
2	Low	Upper West Side	3625
3	Low	Midtown	2509
4	Low	Upper East Side	4291
5	Low	JFK Airport	1114
6	Low	LaGuardia Airport	2215
7	Medium	Lower Manhattan	4155
8	Medium	Upper West Side	4224
9	Medium	Midtown	2943
10	Medium	Upper East Side	3187
11	Medium	JFK Airport	7468
12	Medium	LaGuardia Airport	5839
13	High	Lower Manhattan	1831
14	High	Upper West Side	908

⋮

```
[p,tbl] = anova1(GroupedSummaryTable.dayOfYear,GroupedSummaryTable.Demand,'off')
```

```
p = 1.3474e-05
tbl = 4×6 cell
```

	1	2	3	4	5	6
1	'Source'	'SS'	'df'	'MS'	'F'	'Prob>F'
2	'Groups'	2.4887e+05	2	1.2444e+05	11.2171	1.3474e-05
3	'Error'	5.8283e+08	52539	1.1093e+04	[ ]	[ ]
4	'Total'	5.8308e+08	52541	[ ]	[ ]	[ ]

Null hypothesis is day of year is independant to demand The p-value of 1.347e-05 suggests rejection of/against the null hypothesis.

In other words there is sufficient evidance that day of year is dependant on demand.

```
[table,chi2,p] = crosstab(GroupedSummaryTable.Demand,GroupedSummaryTable.dayOfYear)
```

```
table = 3×365
    37    45    47    50    45    45    44    47    44    35    49    39    46 ···
    91    83    78    75    79    79    79    74    71    88    74    88    73
    16    16    19    19    20    20    21    23    29    21    21    17    25
chi2 = 635.2932
p = 0.9942
```

The returned p value of 0. 9942 which indicates that, at the 5% significance level(even at 99% significance level)

crosstab fails to reject the null hypothesis that day of year is independant to demand or day of year and demand is % independent to each other

```
[table,chi2,p] = crosstab(GroupedSummaryTable.Demand,GroupedSummaryTable.IsHoliday)
```

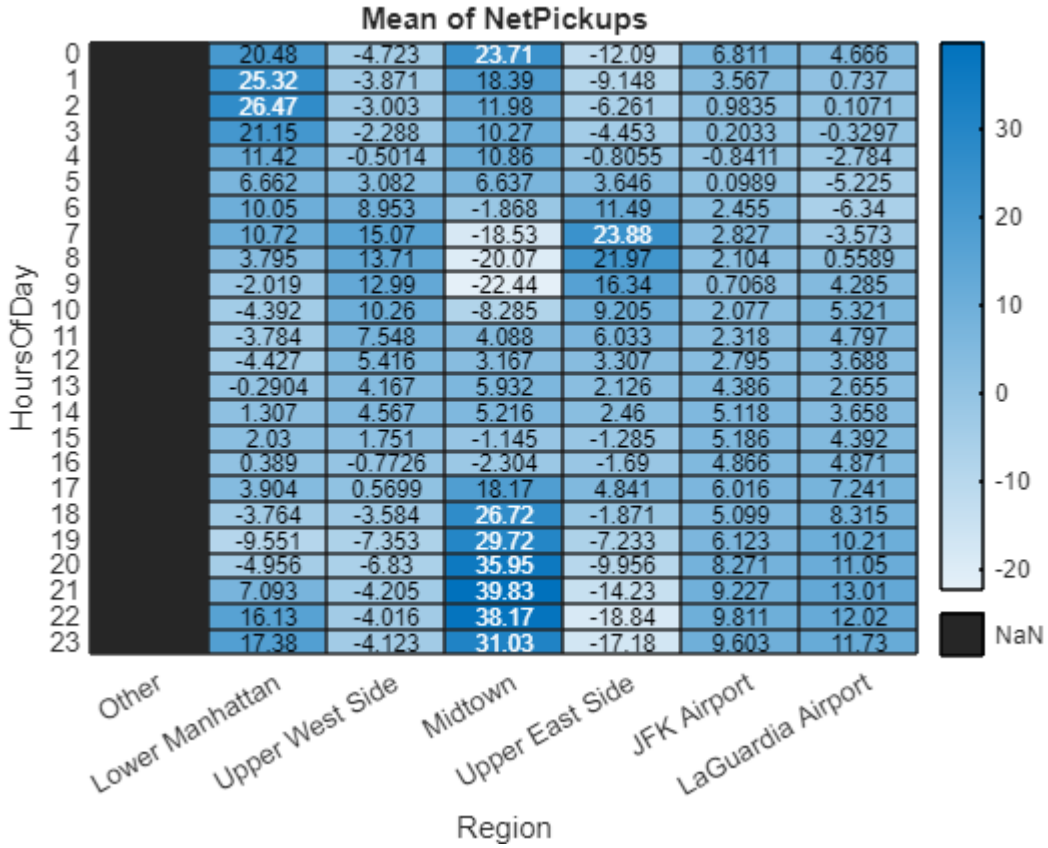
```
table = 3×2
       16508          17
       27780          36
        8182          19
chi2 = 6.8951
p = 0.0318
```

The returned p value of 0.0318 indicates that, at the 5% significance level.

Threrefore crosstab rejects the null hypothesis that IsHoliday is independant to demand.

Below heat map shows demand/net pickups depends on combination of Region and hours of days.

```
heatmap(GroupedSummaryTable, 'Region','HoursOfDay',ColorVariable="NetPickups");
```



	Other	Lower Manhattan	Upper West Side	Midtown	Upper East Side	JFK Airport	LaGuardia Airport
0		20.48	-4.723	**23.71**	-12.09	6.811	4.666
1		**25.32**	-3.871	18.39	-9.148	3.567	0.737
2		**26.47**	-3.003	11.98	-6.261	0.9835	0.1071
3		21.15	-2.288	10.27	-4.453	0.2033	-0.3297
4		11.42	-0.5014	10.86	-0.8055	-0.8411	-2.784
5		6.662	3.082	6.637	3.646	0.0989	-5.225
6		10.05	8.953	-1.868	11.49	2.455	-6.34
7		10.72	15.07	-18.53	**23.88**	2.827	-3.573
8		3.795	13.71	-20.07	21.97	2.104	0.5589
9		-2.019	12.99	-22.44	16.34	0.7068	4.285
10		-4.392	10.26	-8.285	9.205	2.077	5.321
11		-3.784	7.548	4.088	6.033	2.318	4.797
12		-4.427	5.416	3.167	3.307	2.795	3.688
13		-0.2904	4.167	5.932	2.126	4.386	2.655
14		1.307	4.567	5.216	2.46	5.118	3.658
15		2.03	1.751	-1.145	-1.285	5.186	4.392
16		0.389	-0.7726	-2.304	-1.69	4.866	4.871
17		3.904	0.5699	18.17	4.841	6.016	7.241
18		-3.764	-3.584	**26.72**	-1.871	5.099	8.315
19		-9.551	-7.353	**29.72**	-7.233	6.123	10.21
20		-4.956	-6.83	**35.95**	-9.956	8.271	11.05
21		7.093	-4.205	**39.83**	-14.23	9.227	13.01
22		16.13	-4.016	**38.17**	-18.84	9.811	12.02
23		17.38	-4.123	**31.03**	-17.18	9.603	11.73

Description of test data split

Data were split using cvpartition function with 20% data for testing and ramaining 80% data for training purpuse using code annexed herewith with.

Modeling

Statement of the objective(s)

Model has following mail objects :-

- Model which gives highest accuracy
- model whcih has good speed
- model which has lowest cost

Final model description:

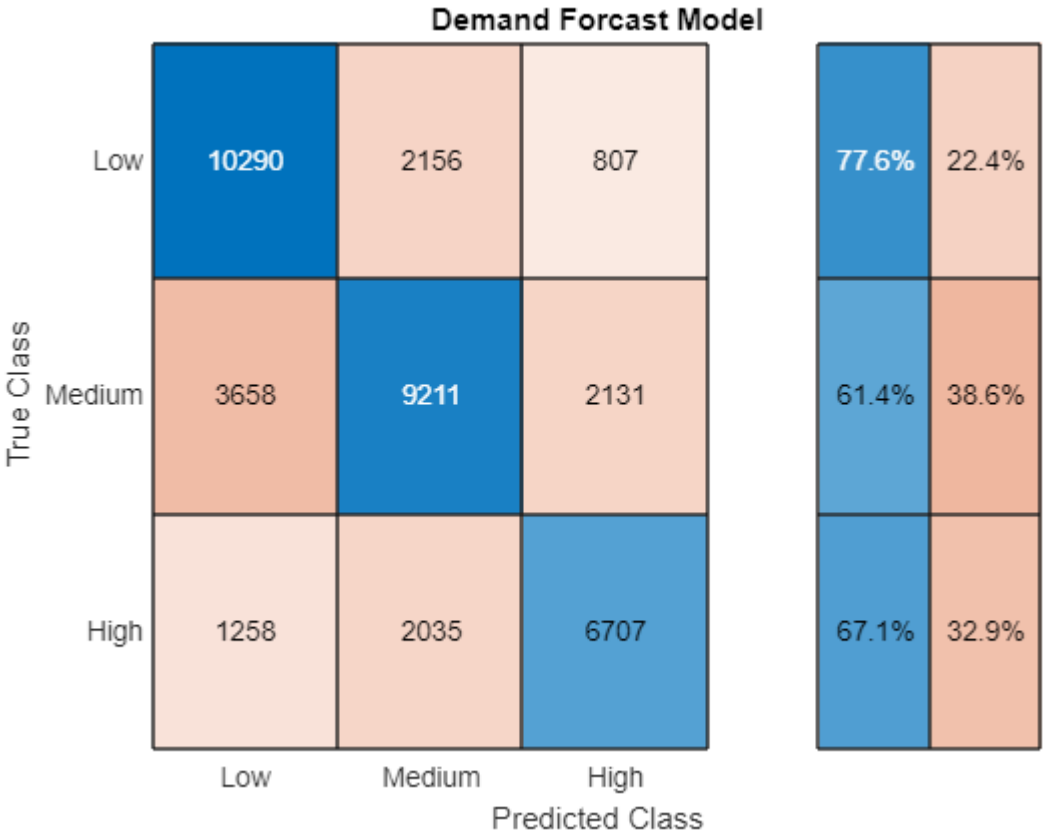
1. Model type

for getting best model we tried 3 different models as below :-

Model 1 - SVM model - This is very slow ansl gave only 40% accuracy which is very less therefore this model was not considered.

Model 2 - SVM using Gaussian Kernel - This model gave only 68.51% accuracy which is substential improvment then first model, but this model is also very slow, therefore we considered one more model to improve the spead without comprising the quality.

```
% Getting ConfusionMatrics
confusionchart(response2,validationPredictions2,"RowSummary", "row-normalized","Title",
```



Below is result of cMetrics of the model.

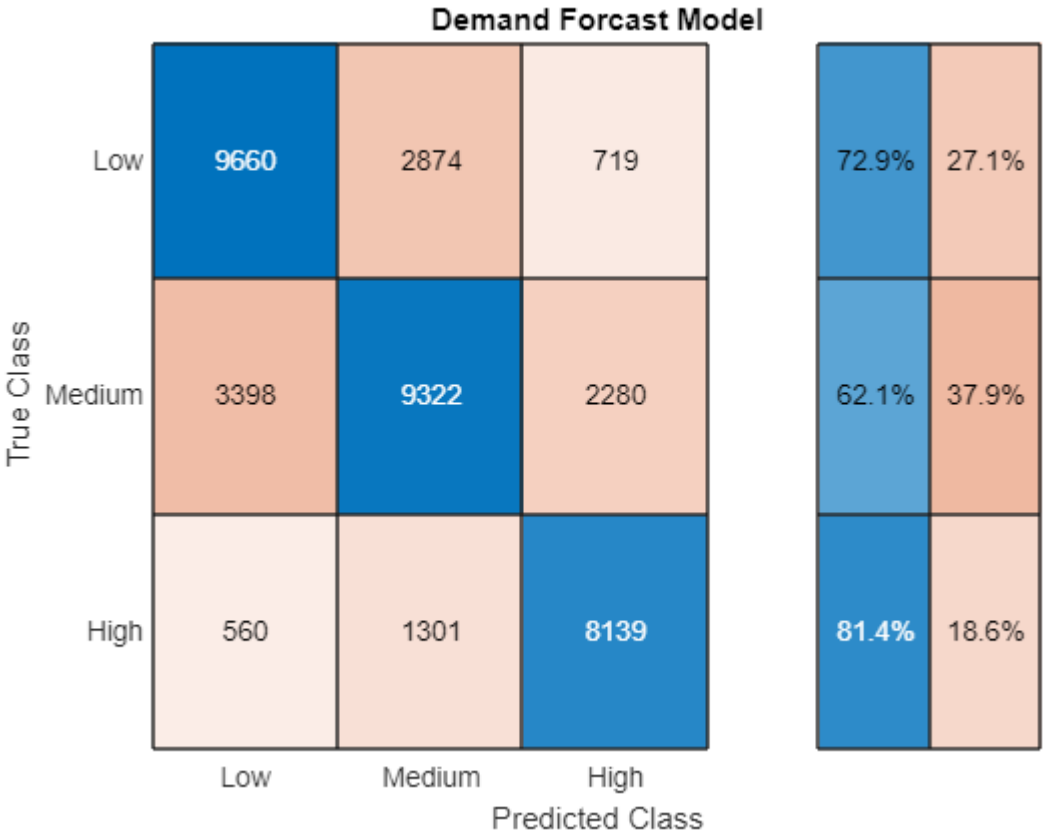
```
% Getting  class-wise metrics - CMatrics
cMetrics(response2,validationPredictions2)
```

```
Accuracy = 68.51%
ans = 5×5 table
```

	Precision	Recall	Fallout	Specificity	F1
1 Low	0.6767	0.7764	0.1966	0.8034	0.7231
2 Medium	0.6873	0.6141	0.1802	0.8198	0.6486
3 High	0.6954	0.6707	0.1040	0.8960	0.6828
4 Avg	0.6865	0.6871	0.1603	0.8397	0.6849
5 WgtAvg	0.6857	0.6851	0.1660	0.8340	0.6834

Model 3 'Bag' ensemble method - this model gave accuracy of 70.90% but there is sufficient improvemnt in speed. Below is Confusion Matrics of the model.

```
% Getting ConfusionMatrics
confusionchart(response3,validationPredictions3,"RowSummary", "row-normalized","Title",
```



Below is result of cMetrics of the model.

```
% Getting  class-wise metrics - CMatrics
cMetrics(response3,validationPredictions3)
```

```
Accuracy = 70.90%
ans = 5×5 table
```

	Precision	Recall	Fallout	Specificity	F1
1 Low	0.7094	0.7289	0.1583	0.8417	0.7190
2 Medium	0.6907	0.6215	0.1795	0.8205	0.6542
3 High	0.7307	0.8139	0.1061	0.8939	0.7701
4 Avg	0.7103	0.7214	0.1480	0.8520	0.7144
5 WgtAvg	0.7076	0.7090	0.1530	0.8470	0.7070

2. Hyperparameters

In above third model following Hyperparameters were used.

- MaxNumSplits = 27056
- NumVariablesToSample = 3

- NumLearningCycles = 11

3. Required predictor features

following features were used predictor features :-

- "Region",
- 'dayOfYear',
- 'HoursOfDay',
- 'IsHoliday'

Training description:

1. Hyperparameter optimization

In above third model using 'Bag' ensemble method through app Hyperparameter were optimized and blow are value of optimized Hyperparameters:-

MaxNumSplits = 27056

NumVariablesToSample = 3

NumLearningCycles = 11

2. Customizations to the cost

A **taxi deployment strategy** based on demand:

- 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

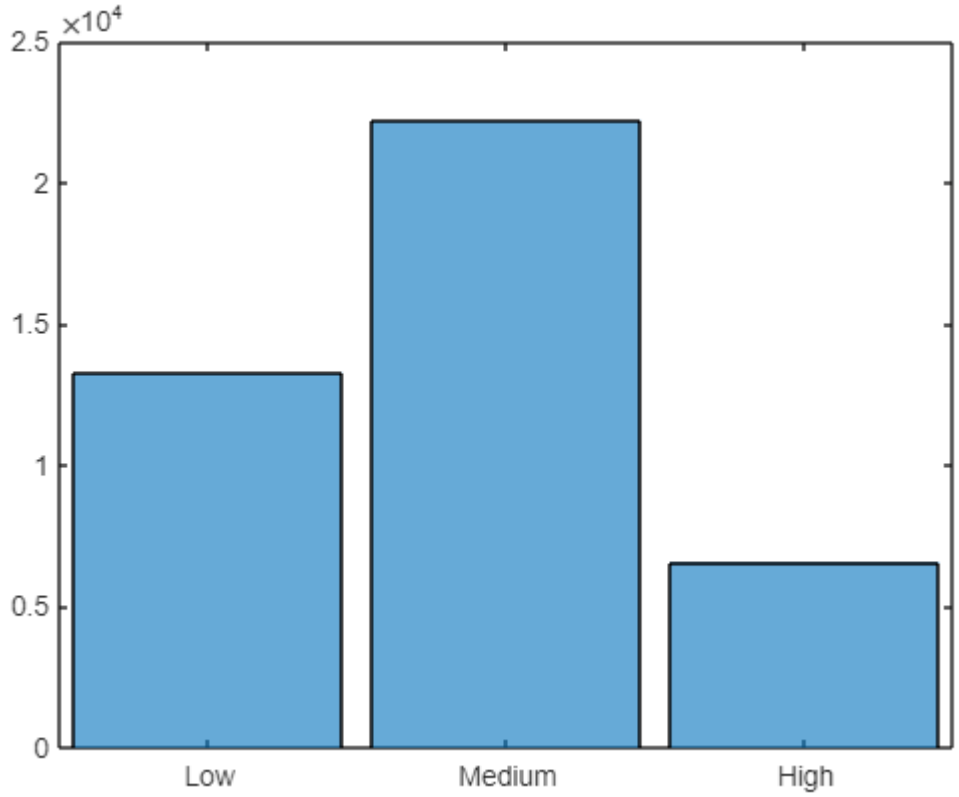
```
cost4 = [0 3 3 ; 2 0 1; 2 1 0]
```

```
cost4 = 3×3
     0     3     3
     2     0     1
     1     1     0
```

3. Other actions (if any, e.g. data over/under sampling)

Below is status of class imbalance,

```
histogram(GroupedSummaryTrain.Demand)
```



To remove class imbalnce we need to split the training data by Response Class

```
GroupedSummaryTrainLow = GroupedSummaryTrain(GroupedSummaryTrain.Demand == "Low",:);
GroupedSummaryTrainMedium = GroupedSummaryTrain(GroupedSummaryTrain.Demand == "Medium",
GroupedSummaryTrainHigh = GroupedSummaryTrain(GroupedSummaryTrain.Demand == "High",:);
```

then we will do Oversample the Minority Class Data

```
GroupedSummaryTrainUnderSampled =  datasample(GroupedSummaryTrainMedium, 15000, "Replac
GroupedSummaryTrainOverSampled =  datasample(GroupedSummaryTrainHigh, 10000, "Replace",
```

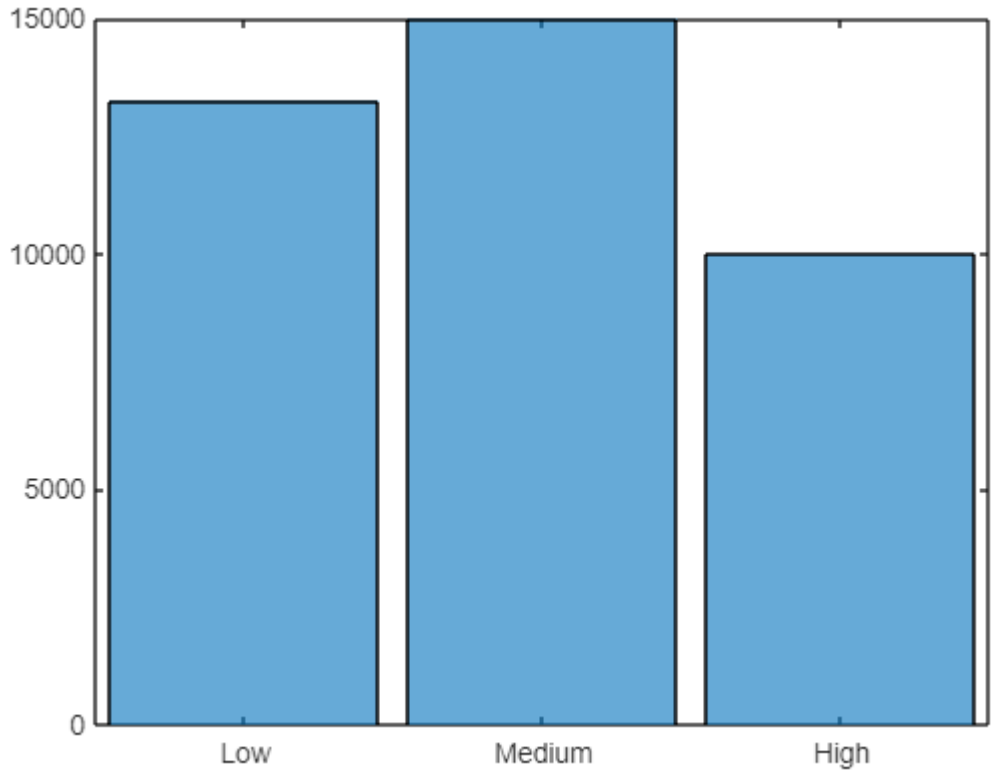
and in last Combine Majority, Minority and normal Class Data

```
GroupedSummaryTrainSampled = [GroupedSummaryTrainLow; GroupedSummaryTrainUnderSampled;
```

Visualizations after removing class imbalance

New data set's distribution with a histogram.

```
histogram(GroupedSummaryTrainSampled.Demand)
```



The training data set now has roughly the same number of observations in each class.

4. Validation method and metrics

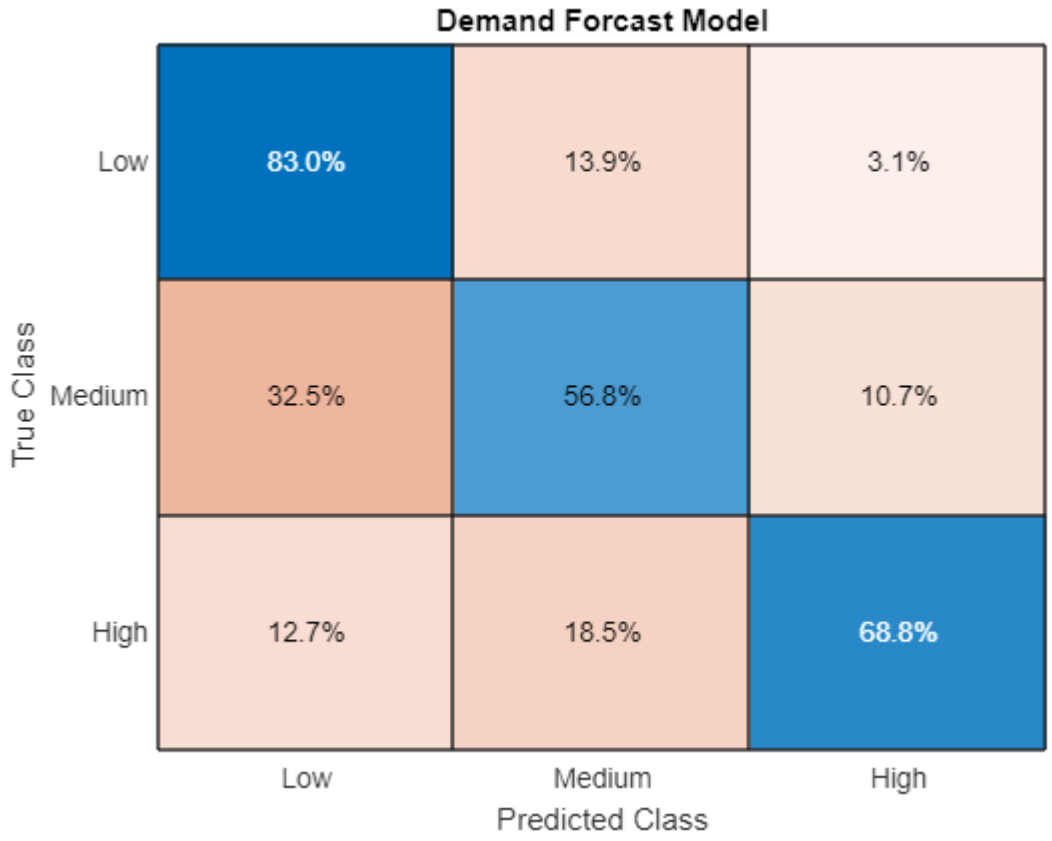
```
inputTable4 = GroupedSummaryTrainSampled;
predictorNames4 = {'Region', 'dayOfYear', 'HoursOfDay', 'IsHoliday'};
predictors4 = inputTable4(:, predictorNames4);
response4 = inputTable4.Demand;

 template4 = templateTree(...
    'MaxNumSplits', 27056, ...
    'NumVariablesToSample', 3);

classificationEnsemble4 = fitcensemble(...
    predictors4, ...
    response4, ...
    'Method', 'Bag', ...
    'NumLearningCycles', 11, ...
    'Learners', template4, ...
    "Cost", cost4,...
    'ClassNames', categorical({'Low'; 'Medium'; 'High'}, {'Low' 'Medium' 'High'}));

% Perform cross-validation
partitionedMode4 = crossval(classificationEnsemble4, 'KFold', 5);

% Compute validation predictions
```

```
[validationPredictions4, validationScores4] = kfoldPredict(partitionedMode4);

% Compute validation accuracy
validationAccuracy4 = 1 - kfoldLoss(partitionedMode4, 'LossFun', 'ClassifError');

validationAccuracy4
```

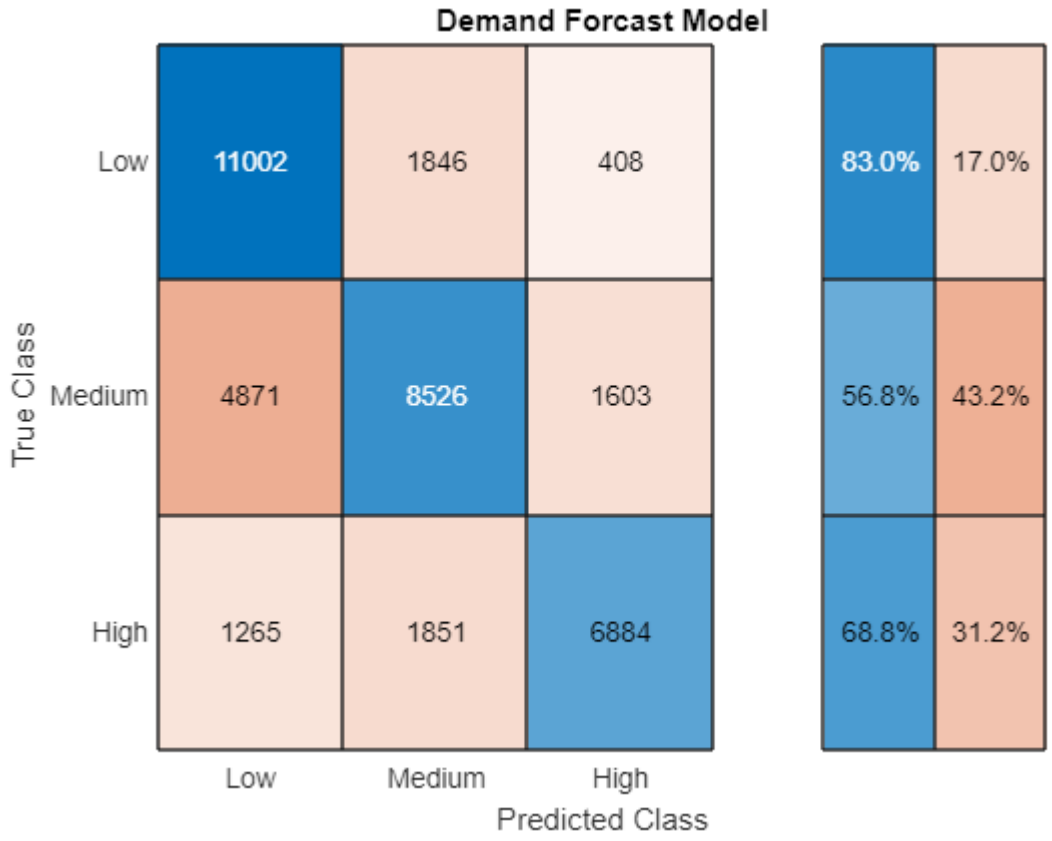
```
validationAccuracy4 = 0.6904
```

Getting ConfusionMatrics

```
confusionchart(response4,validationPredictions4,"Normalization","row-normalized","Title
```




```
confusionchart(response4,validationPredictions4,"RowSummary", "row-normalized","Title",
```



Getting class-wise metrics - cMetrics

```
cMetrics(response4,validationPredictions4)
```

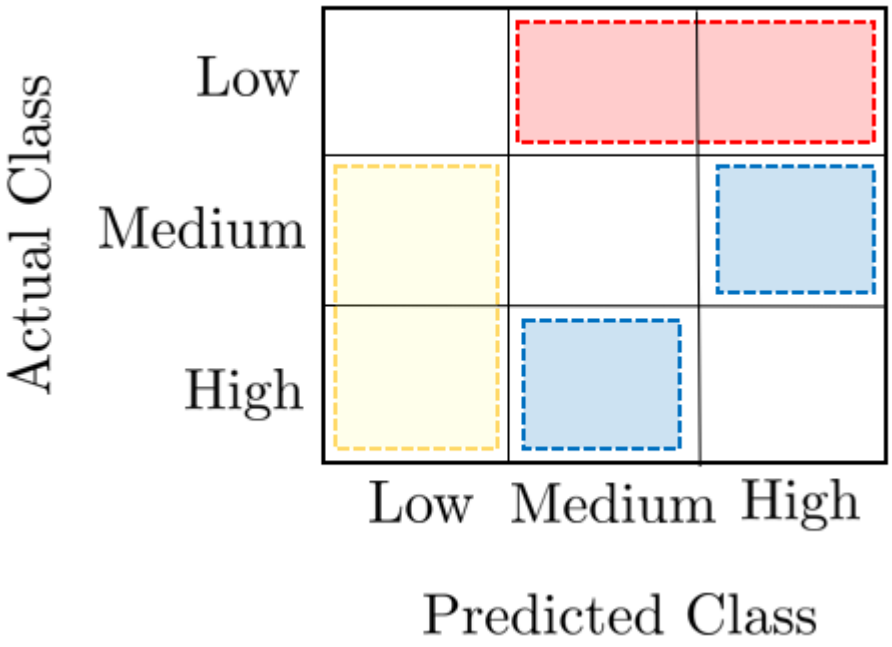
```
Accuracy = 69.04%
ans = 5×5 table
```

	Precision	Recall	Fallout	Specificity	F1
1 Low	0.6420	0.8300	0.2454	0.7546	0.7240
2 Medium	0.6975	0.5684	0.1590	0.8410	0.6264
3 High	0.7739	0.6884	0.0712	0.9288	0.7287
4 Avg	0.7045	0.6956	0.1585	0.8415	0.6930
5 WgtAvg	0.6982	0.6904	0.1660	0.8340	0.6869

Test metrics

As we can see cost of error under model 3 is 6690 whereas cost of error under model 4 is 6740.

This is because cost of error is calculated using below matrix whereas red cell are given highest cost of 3 and blue cell are given lowest cost of 1 and yellow cells were given medium cost of 2. Model 4 is based on this given matrix therefore its accuracy was little comprosed but cost of error was reduced.




```
% Get the true and predicted classes
GroupedSummaryTest.predictUnderModel3 = predict(classificationEnsemble3,GroupedSummaryT
GroupedSummaryTest.predictUnderModel4 = predict(classificationEnsemble4,GroupedSummaryT
Model3ErrorCost = (height(GroupedSummaryTest(GroupedSummaryTest.predictUnderModel3 == '
    (height(GroupedSummaryTest(not(GroupedSummaryTest.predictUnderModel3 == "Low") & Gr
    ((height(GroupedSummaryTest(GroupedSummaryTest.predictUnderModel3 == "Medium" & Gro
    height(GroupedSummaryTest(GroupedSummaryTest.predictUnderModel3 == "High" & Grouped
```

```
Model3ErrorCost = 6690
```

```
Model4ErrorCost = (height(GroupedSummaryTest(GroupedSummaryTest.predictUnderModel4 == '
    (height(GroupedSummaryTest(not(GroupedSummaryTest.predictUnderModel4 == "Low") & Gr
    ((height(GroupedSummaryTest(GroupedSummaryTest.predictUnderModel4 == "Medium" & Gro
    height(GroupedSummaryTest(GroupedSummaryTest.predictUnderModel4 == "High" & Grouped
```

```
Model2ErrorCost = 6744
```

Summary of models and/or features determined to be suboptimal

In given situation model 4 is best fit and practicle model.

Appendix

Import a single file of taxi data

```
% taxiTable = importTaxiDataWithoutCleaning(filename) creates a table by reading taxi data from the f


function taxiTable = importTaxiDataWithoutCleaning(filename)
% Setup the import options
% This code has been generated automatically using the Import Tool
opts = delimitedTextImportOptions("NumVariables", 19);
```

```
% Specify range and delimiter
opts.DataLines = [2 Inf];
opts.Delimiter = ",";

% Specify column names and types
opts.VariableNames = ["Vendor", "PickupTime", "DropoffTime", "Passengers", "Distance", "PickupLon", "
opts.VariableTypes = ["categorical", "datetime", "datetime", "double", "double", "double", "double",

% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Specify variable properties
opts = setvaropts(opts, ["Vendor", "RateCode", "HeldFlag", "PayType"], "EmptyFieldRule", "auto");
opts = setvaropts(opts, "PickupTime", "InputFormat", "yyyy-MM-dd HH:mm:ss");
opts = setvaropts(opts, "DropoffTime", "InputFormat", "yyyy-MM-dd HH:mm:ss");
% Import the data
taxiTable = readtable(filename, opts);
% Add descriptive category names
% Give descriptive names to the categories used in variables PayType and RateCode....
% Since all codes are not present in % every file, first identify which codes are used,...
% and then replace them with their corresponding descriptive names.
codes = {'1','2','3','4','5','6'};

% Descriptive names corresponding to PayType codes 1-6.
payNames = {'Credit card','Cash','No charge','Dispute','Unknown','Voided'};

% Identify and rename the PayType codes that appear in taxiTable
payCodes = ismember(codes, categories(taxiTable.PayType));
taxiTable.PayType = renamecats(taxiTable.PayType, payNames(payCodes));

% Descriptive names corresponding to RateCats codes 1-6.
% RateCodes not included in the "codes" list are not changed.
rateNames = {'Standard','JFK','Newark','Nassau','Negotiated','Group'};

% Identify and rename the RateCode codes that appear in taxiTable
rateCodes = ismember(codes, categories(taxiTable.RateCode));
taxiTable.RateCode = renamecats(taxiTable.RateCode, codes(rateCodes), rateNames(rateCodes));

end
```

Code for reading and cleaning taxi data

```
% import combined Taxi Data
taxiAll = readall(taxiDatastore )

%trip with fare less the 2.5 was removed from data
lessthenMinFare = taxiAll((taxiAll.Fare<2.50),:)

%trip with fare less the 2.5 are contribution only 8% of total trip data
100*height(lessthenMinFare)/height(taxiAll)
taxiAll = taxiAll((taxiAll.Fare>=2.50),:)

%To Remove average abnormally hig fare -
%upper 2 percentile trips are further removed
```

```
taxiAll.avgFare1 = taxiAll.Fare./taxiAll.Distance
figure;
plot(taxiAll.avgFare1)
title("Average Fare");
xlabel("Average Fare")
ylabel("Frequency")
[~,stdlow,stdhigh,~] = isoutlier(taxiAll.avgFare1,"mean")
[~,medlow,medhigh,~] = isoutlier(taxiAll.avgFare1,"median")
[~,quartlow,quarthigh,~] = isoutlier(taxiAll.avgFare1,"quartiles")
[~,plow,phigh,~] = isoutlier(taxiAll.avgFare1,"percentiles", [0,99])
figure;
histogram(taxiAll.avgFare1)
xline(stdlow,"r"); xline(stdhigh,"r")
xline(quartlow,"b"); xline(quarthigh,"b")
xline(medlow,"k"); xline(medhigh,"k")
xline(plow,"m"); xline(phigh,"m")
xlim([plow-20 phigh+20])
[taxiAll2AvgFare1Outliers,~] = isoutlier(taxiAll.avgFare1,"percentiles", [0,98])
taxiAll = taxiAll(taxiAll2AvgFare1Outliers == 0,:)
```

Add taxi zone

```
% taxiTable = addTaxiZones(taxiTable) adds four features to taxiTable: PickupZone, tzPickupBorough, D
% taxiTable = addTaxiZones(taxiTable, shpfile) uses the boundaries defined in the specified shapefile
% Zones are smaller city regions (e.g. Midtown Center, SoHo). NYC recognizes 260 zones defined as "NY
% Groups of zones form boroughs. NYC Taxi Zones contain six boroughs: the Bronx, Brooklyn, EWR, Manha

% See Also
% inpolygon | addNeighborhood
function taxiTable = addTaxiZones(taxiTable, shpfile)
% If no shape file is specified, use the file provided in taxi data
if nargin < 2
    shpfile = "TaxiZones.shp";
end

% Load the shape file data and convert to a table
shapes = shaperead(shpfile, "UseGeoCoords",true);
shapes = struct2table(shapes);

% Convert variables to categoricals
shapes.zone = categorical(shapes.zone);
shapes.borough = categorical(shapes.borough);
boroNames = categories(shapes.borough);
zones = categories(shapes.zone);

% Create feature variables as empty categorical arrays.
% Define the possible categories using names obtained from the shapefile.
taxiTable.tzPickupBorough = categorical(NaN(height(taxiTable),1),1:length(boroNames),boroNames);
taxiTable.tzDropoffBorough = categorical(NaN(height(taxiTable),1),1:length(boroNames),boroNames);
taxiTable.PickupZone = categorical(NaN(height(taxiTable),1),1:length(zones),zones);
taxiTable.DropoffZone = categorical(NaN(height(taxiTable),1),1:length(zones),zones);
% Add zone information
% For each shape (zone), use pick up/drop off Lat/Lon values, shape Lat/Lon values, and the inpolygo
for i = 1:height(shapes)

    % Identify trips with pick up locations in this zone
    pickupRows = inpolygon(taxiTable.PickupLon, taxiTable.PickupLat, shapes.Lon{i}, shapes.Lat{i});
```

```
    % Identify trips with drop off locations in this zone
    dropoffRows = inpolygon(taxiTable.DropoffLon, taxiTable.DropoffLat, shapes.Lon{i}, shapes.Lat{i})

    % Update taxi zone features with shape's zone name
    taxiTable.PickupZone(pickupRows) = shapes.zone(i);
    taxiTable.DropoffZone(dropoffRows) = shapes.zone(i);

    % Update borough features with shape's borough name
    taxiTable.tzPickupBorough(pickupRows) = shapes.borough(i);
    taxiTable.tzDropoffBorough(dropoffRows) = shapes.borough(i);
end

end
```

Data restructuring

```
% Create the variables PickupZone and DropoffZone using  addTaxiZones function

taxiAll2 = addTaxiZones(taxiAll)

%Assign PickupRegion DropoffRegion to each taxi trip
taxiAll2.PickupRegion = taxiAll2.PickupZone
taxiAll2.DropoffRegion = taxiAll2.DropoffZone
taxiAll2.PickupRegion = addcats(taxiAll2.PickupRegion, {'Lower Manhattan','Midtown','Upper East Side'
taxiAll2.PickupRegion = mergecats(taxiAll2.PickupRegion, {'Lower Manhattan','Alphabet City','Battery
taxiAll2.PickupRegion = mergecats(taxiAll2.PickupRegion, {'Midtown','Clinton East','Clinton West','Mi
taxiAll2.PickupRegion = mergecats(taxiAll2.PickupRegion, {'Upper East Side','Upper East Side North','
taxiAll2.PickupRegion = mergecats(taxiAll2.PickupRegion, {'Upper West Side','Upper West Side North','
taxiAll2.DropoffRegion = addcats(taxiAll2.DropoffRegion, {'Lower Manhattan','Midtown','Upper East Sid
taxiAll2.DropoffRegion = mergecats(taxiAll2.DropoffRegion, {'Lower Manhattan','Alphabet City','Batter
taxiAll2.DropoffRegion = mergecats(taxiAll2.DropoffRegion, {'Midtown','Clinton East','Clinton West','
taxiAll2.DropoffRegion = mergecats(taxiAll2.DropoffRegion, {'Upper East Side','Upper East Side North'
taxiAll2.DropoffRegion = mergecats(taxiAll2.DropoffRegion, {'Upper West Side','Upper West Side North'
taxiAll2.PickupRegion = addcats(taxiAll2.PickupRegion, {'Other'})
taxiAll2.DropoffRegion = addcats(taxiAll2.DropoffRegion, {'Other'})
taxiAll2.PickupRegion = fillmissing(taxiAll2.PickupRegion, 'constant', 'Other');
taxiAll2.PickupRegion = mergecats(taxiAll2.PickupRegion,["Other","Allerton/Pelham Gardens","Arden Hei
taxiAll2.DropoffRegion = fillmissing(taxiAll2.DropoffRegion, 'constant', 'Other');
taxiAll2.DropoffRegion = mergecats(taxiAll2.DropoffRegion,["Other", "Allerton/Pelham Gardens","Arden
taxiAll2 = taxiAll2((not(taxiAll2.PickupRegion == "Other") | not(taxiAll2.DropoffRegion == "Other")),
```

Feature Engineering

```
% Creating DropoffTimeGroup, PickupTimeGroup and Trip Duration
taxiAll2.DropoffTimeGroup = dateshift(taxiAll2.DropoffTime,'start','hour')
taxiAll2.PickupTimeGroup = dateshift(taxiAll2.PickupTime,'start','hour')
taxiAll2.Duration = taxiAll2.DropoffTime-taxiAll2.PickupTime

% Creating the Grouped Summary Table
AvgDistance = groupsummary(taxiAll2,["PickupRegion","PickupTimeGroup"],"mean","Distance","IncludeEmpt
AvgFare = groupsummary(taxiAll2,["PickupRegion","PickupTimeGroup"],"mean","Fare","IncludeEmptyGroups"
AvgDuration = groupsummary(taxiAll2,["PickupRegion","PickupTimeGroup"],"mean","Duration","IncludeEmpt
DropCount = groupsummary(taxiAll2,["DropoffRegion","DropoffTimeGroup"],"IncludeEmptyGroups",true)
GroupedSummaryTable = join(AvgDistance,AvgDuration)
GroupedSummaryTable = join(GroupedSummaryTable,AvgFare)
GroupedSummaryTable = join(GroupedSummaryTable,DropCount,...
    "LeftKeys", [1,2],...
    "RightKeys",[1,2])
```

```
GroupedSummaryTable = renamevars(GroupedSummaryTable,["PickupRegion","PickupTimeGroup","GroupCount_Gr
                ["Region","HourlyBin","PickupCount", "DropoffCount", "AvgDistance", "AvgDuration", '
GroupedSummaryTable  = movevars(GroupedSummaryTable , "DropoffCount", "before", "AvgDistance")
GroupedSummaryTable.NetPickups = GroupedSummaryTable.PickupCount - GroupedSummaryTable.DropoffCount

%Generating a categorical response feature, 'Demand'
GroupedSummaryTable = GroupedSummaryTable(not(GroupedSummaryTable.Region == "Other"),:)
m2 = max(GroupedSummaryTable.NetPickups)
m1 = min(GroupedSummaryTable.NetPickups)
edges = [m1,0, 15, m2]
GroupedSummaryTable.Demand = discretize(GroupedSummaryTable.NetPickups ,edges)
GroupedSummaryTable.Demand = categorical(GroupedSummaryTable.Demand)
GroupedSummaryTable.Demand = renamecats(GroupedSummaryTable.Demand, "1", "Low")
GroupedSummaryTable.Demand = renamecats(GroupedSummaryTable.Demand, "2", "Medium")
GroupedSummaryTable.Demand = renamecats(GroupedSummaryTable.Demand, "3", "High")

% Generating set of predictors

GroupedSummaryTable.dayOfYear = day(GroupedSummaryTable.HourlyBin,"dayofyear")
GroupedSummaryTable.HoursOfDay = hour(GroupedSummaryTable.HourlyBin)
DateStrings = {'01-01-2015';    '19-01-2015';    '16-02-2015';    '03-04-2015';    '25-05-2015';    '
t = datetime(DateStrings,'InputFormat','dd-MM-yyyy')
GroupedSummaryTable.IsHoliday = ismember(GroupedSummaryTable.HourlyBin ,t)
```

Test Data Split

```
rng(10);

GroupSumaryPartitions = cvpartition(height(GroupedSummaryTable),"HoldOut",0.2);

GroupedSummaryTestIdx = test(GroupSumaryPartitions);
GroupedSummaryTest = GroupedSummaryTable(GroupedSummaryTestIdx,:);

GroupedSummaryTrainIdx = training(GroupSumaryPartitions);
GroupedSummaryTrain = GroupedSummaryTable(GroupedSummaryTrainIdx,:);
```

Model 1 - SVM model

```
inputTable = GroupedSummaryTrain;
predictorNames = {'Region', 'dayOfYear', 'HoursOfDay', 'IsHoliday'};
predictors = inputTable(:, predictorNames);
response = inputTable.Demand;

cost = [0 1 1 ; 1 0 1; 1 1 0]

md1 = fitcecoc(GroupedSummaryTrain,"Demand",...
    "PredictorNames",["Region",'dayOfYear','HoursOfDay','IsHoliday'],...
    "Learners","svm",...
    "Cost",cost,...
    "Coding","onevsone")

% Perform cross-validation
partitionedModel = crossval(md1, 'KFold', 5);

% Compute validation predictions
[validationPredictions1, validationScores1] = kfoldPredict(partitionedModel);
```

```
% Compute validation accuracy
validationAccuracy1 = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');

%Getting ConfusionMatrics
confusionchart(response,validationPredictions1,"Normalization","row-normalized","Title","Demand Forca

% Getting  class-wise metrics - CMatrics
cMetrics(response,validationPredictions1)
```

SVM using Gaussian Kernel

```
inputTable2 = GroupedSummaryTrainSampled;
response2 = inputTable2.Demand;
cost = [0 1 1 ; 1 0 1; 1 1 0]

template2 = templateSVM(...
    'KernelFunction', "gaussian", ...
    'Standardize', true);
cost = [0 1 1 ; 1 0 1; 1 1 0]

 md2 = fitcecoc(GroupedSummaryTrainSampled,"Demand",...
    "PredictorNames",["Region",'dayOfYear','HoursOfDay','IsHoliday'],...
    "Learners", template2,...
    "Cost",cost,...
    "Coding","onevsone")

 % Perform cross-validation
partitionedMode2 = crossval(md2, 'KFold', 5);

% Compute validation predictions
[validationPredictions2, validationScores2] = kfoldPredict(partitionedMode2);

% Compute validation accuracy
validationAccuracy2 = 1 - kfoldLoss(partitionedMode2, 'LossFun', 'ClassifError');

validationAccuracy2

% Getting ConfusionMatrics
confusionchart(response,validationPredictions2,"Normalization","row-normalized","Title","Demand Forca
confusionchart(response,validationPredictions2,"RowSummary", "row-normalized","Title","Demand Forcast

% Getting  class-wise metrics - CMatrics
cMetrics(response2,validationPredictions2)
```

'Bag' ensemble method

```
inputTable3 = GroupedSummaryTrainSampled;
predictorNames3 = {'Region', 'dayOfYear', 'HoursOfDay', 'IsHoliday'};
predictors3 = inputTable3(:, predictorNames3);
response3 = inputTable3.Demand;

cost = [0 1 1 ; 1 0 1; 1 1 0]

 template3 = templateTree(...
    'MaxNumSplits', 27056, ...
    'NumVariablesToSample', 3);
```

```
classificationEnsemble3 = fitcensemble(...
    predictors3, ...
    response3, ...
    'Method', 'Bag', ...
    'NumLearningCycles', 11, ...
    'Learners', template3, ...
    "Cost",cost,...
    'ClassNames', categorical({'Low'; 'Medium'; 'High'}, {'Low' 'Medium' 'High'}));


% Perform cross-validation
partitionedMode3 = crossval(classificationEnsemble3, 'KFold', 5);

% Compute validation predictions
[validationPredictions3, validationScores3] = kfoldPredict(partitionedMode3);

% Compute validation accuracy
validationAccuracy3 = 1 - kfoldLoss(partitionedMode3, 'LossFun', 'ClassifError');

validationAccuracy3

% Getting ConfusionMatrics
confusionchart(response3,validationPredictions3,"Normalization","row-normalized","Title","Demand Forc
confusionchart(response3,validationPredictions3,"RowSummary", "row-normalized","Title","Demand Forcas

% Getting  class-wise metrics - CMatrics
cMetrics(response3,validationPredictions3)
```

cMetrics

```
% Compute class-wise metrics
%t = cMetrics(ytrue,ypred) computes class-wise metrics from the true class labels, ytrue, and the cor
%[t,a] = cMetrics(ytrue,ypred) also returns the decimal value for model accuracy in variable a.
%Inputs ytrue and ypred must be column vectors of equal length.
%Output t is a table containing the following metrics for each class:
%Precision
%Recall
%Fallout
%Specificity

%The last two rows of table t contain the average metric values across all classes (Avg), and the ave

%See Also
%accuracy | precision | recall | fallout | specificity | f1
function [t,a] = cMetrics(ytrue,ypred)
% create a table where each variable is the output of a metric function
t = [precision(ytrue,ypred), ...
     recall(ytrue,ypred), ...
     fallout(ytrue,ypred), ...
     specificity(ytrue,ypred), ...
     f1(ytrue,ypred)];

% Compute accuracy and write the percentage to the screen
a = accuracy(ytrue,ypred);
fprintf('Accuracy = %0.2f%%',a*100);

% Compute average of each metric across all classes
```

```
avg = mean(t{:,:},1);

% Categories are in alphabetical order by default
% Place ytrue classes in the same order as rows in t
ytrue = reordercats(categorical(ytrue),t.Properties.RowNames);

% Count the number of occurrences of each class
wt = countcats(ytrue);

% Compute weighted average of each metric across all classes
% eqn: wtavg = sum(metric(i) * wt(i) / total number of labels)
wtavg = sum(t{:,:}.*wt/sum(wt),1);
tavg = array2table([avg;wtavg], ...
    "VariableNames",["Precision","Recall","Fallout","Specificity","F1"],...
    "RowNames",["Avg","WgtAvg"]);

% Add table tavg to the bottom of table t
t = [t;tavg];
end

%Compute model accuracy
%a = accuracy(ytrue,ypred) computes the accuracy of a model from ytrue, which contains the correct la
%Inputs ytrue and ypred must be column vectors of equal length.
%Output a is a value between 0 and 1.

%Algorithm
%Accuracy is the number of true positives and true negatives divided by the total number of observati


%See Also
%confusionmat
function a = accuracy(ytrue,ypred)
% Create the confusion matrix
CM = confusionmat(ytrue,ypred);
The confusion matrix compares true and predicted labels. The number of predictions that are equal to
% Correctly identified
sumCorrect = sum(diag(CM));
% Total number of observations
sumAll = sum(CM,'all');

% Compute accuracy
a = sumCorrect/sumAll;
end

%Calculate class-wise precision
%t = precision(ytrue,ypred) computes the class-wise precision from the true class labels, ytrue, and
%Inputs ytrue and ypred must be column vectors of equal length.

%Algorithm
%The precision for class  is defined as the number correctly labeled as class k divided by the total


%See Also
%confusionmat | predictiveTotals
function t = precision(ytrue,ypred)
% Create the confusion matrix
[CM,classnames] = confusionmat(ytrue,ypred);
```

```
The confusion matrix compares true and predicted labels. The number of predictions that are equal to
n = length(CM);
Precision = zeros(n,1);
% Calculate the precision for each class
for k = 1:n
    % obtain by class counts of true positive and false positive classifications
    [TP,~,FP,~] = predictiveTotals(CM,k);
    Precision(k) = TP/(TP+FP);
end

% Create output variable table t
t = table(Precision,'RowNames',string(classnames));
end

%Calculate class-wise recall
%t = recall(ytrue,ypred) computes the class-wise recall from the true class labels, ytrue, and the co
%Inputs ytrue and ypred must be column vectors of equal length.

%Algorithm
%The recall for class  is defined as the number correctly labeled as class k divided by number labele


%See Also
%confusionmat | predictiveTotals
function t = recall(ytrue,ypred)
% Create the confusion matrix
[CM,classnames] = confusionmat(ytrue,ypred);
The confusion matrix compares true and predicted labels. The number of predictions that are equal to
n = length(CM);
Recall = zeros(n,1);
% Calculate the precision for each class
for k = 1:n
    % obtain by class counts of true positive and false negative classifications
    [TP,FN,~,~] = predictiveTotals(CM,k);
    Recall(k) = TP/(TP+FN);
end

% Create output variable table t
t = table(Recall,'RowNames',string(classnames));
end

%Calculate class-wise fallout
%t = fallout(ytrue,ypred) computes the class-wise fallout from the true class labels, ytrue, and the
%Inputs ytrue and ypred must be column vectors of equal length.

%Algorithm
%The fallout for class  is defined as:


%See Also
%confusionmat | predictiveTotals
function t = fallout(ytrue,ypred)
% Create the confusion matrix
[CM,classnames] = confusionmat(ytrue,ypred);
The confusion matrix compares true and predicted labels. The number of predictions that are equal to
n = length(CM);
Fallout = zeros(n,1);
```

```
% Calculate the precision for each class
for k = 1:n
    % obtain by class counts of false positive and true negative classifications
    [~,~,FP,TN] = predictiveTotals(CM,k);
    Fallout(k) = FP/(FP+TN);
end

% Create output variable table t
t = table(Fallout,'RowNames',string(classnames));
end

%Calculate class-wise  scores
%t = f1(ytrue,ypred) computes class-wise  scores from the true class labels, ytrue, and the correspon
%Inputs ytrue and ypred must be column vectors of equal length.

%Algorithm
%The  score for class  is defined as the harmonic mean of the precision and recall of class :


%See Also
%precision | recall
function t = f1(ytrue,ypred)
% Obtain class-wise precision and recall values
p = precision(ytrue,ypred);
r = recall(ytrue,ypred);

% Compute F1 score for each class using elementwise operators
F1 = 2*(p.Precision.*r.Recall)./(p.Precision+r.Recall);

% Create output variable table t
t = table(F1,'RowNames',p.Properties.RowNames);
end
%Calculate class-wise specificity
%t = specificity(ytrue,ypred) computes class-wise specificity for the true class labels, ytrue, and t
%Inputs ytrue and ypred must be column vectors of equal length.

%Algorithm
%The specificity for class  is defined as:


%See Also
%confusionmat | predictiveTotals
function t = specificity(ytrue,ypred)
% Create the confusion matrix
[CM,classnames] = confusionmat(ytrue,ypred);
The confusion matrix compares true and predicted labels. The number of predictions that are equal to
n = length(CM);
Specificity = zeros(n,1);
% Calculate the precision for each class
for k = 1:n
    % obtain by class counts of false positive and True negative classifications
    [~,~,FP,TN] = predictiveTotals(CM,k);
    Specificity(k) = TN/(TN+FP);
end

% Create output variable table t
t = table(Specificity,'RowNames',string(classnames));
```

```
end

%Count class predictive totals by type
%[TP,FN,FP,TN] = predictiveTotals(CM,k) computes the True Positive, False Negative, False Positive, a

%Algorithm
%These predictive totals can be calculated from the confusion matrix using the following equations.




%where:
% is the entry in row  and column  of the confusion matrix CM
% is the number of true positives for class
% is the number of false negatives for class
% is the number of false positives for class
% is the number of true negatives for class


function [TP,FN,FP,TN] = predictiveTotals(CM,k)
% in a confusion matrix, rows represent the true class, and columns represent the predicted class
% Entries along the diagonal (row==column) indicate the number of observations
% whose label was correctly predicted.
TP = CM(k,k);                   % True positive

% Row k means observations that are actually class k
FN = sum(CM(k,:))-TP;           % False negative

% Column k means observations that were predicted to be class k
FP = sum(CM(:,k))-TP;           % False positive

TN = sum(CM,'all')-TP-FN-FP;    % True negative

end
```