```
use anjamkudy;
set hive.exec.dynamic.partition =  TRUE;
set hive.exec.dynamic.partition.mode = nonstrict;
set hive.exec.max.dynamic.partitions = 1500;
```

	key	value
1	hive.exec.max.dynamic.partitions	1500

Showing all 1 rows.

```
%fs
ls /datasets/yelp
```

	path	name	size
1	dbfs:/datasets/yelp/yelp_business.csv	yelp_business.csv	42269133
2	dbfs:/datasets/yelp/yelp_review.csv	yelp_review.csv	3691614828

Showing all 2 rows.

Copying the data to the local workspace

```
%fs
cp "/datasets/yelp/yelp_review.csv" "/users/anjamkudy/yelp_review.csv"
```

```
res4: Boolean = true
```

```
%fs
 cp "/datasets/yelp/yelp_business.csv" "/users/anjamkudy/yelp_business.csv"
```

```
res5: Boolean = true
```

Creating the Review table and inserting the entires from the file

```
--drop table yelp_reviews_gp5;

CREATE TABLE IF NOT EXISTS yelp_reviews_gp5
(   funny        int
, user_id        string
, review_id    string
, text        string
, business_id    string
, stars        int
, review_date        timestamp
, useful        int
, cool        int
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
tblproperties ("skip.header.line.count"="2");
```

OK

```
Load data inpath "/users/anjamkudy/yelp_review.csv" OVERWRITE into table
yelp_reviews_gp5;
```

OK

Creating the Business table and inserting the entires from the file

```
drop table yelp_business_gp5;

CREATE TABLE IF NOT EXISTS yelp_business_gp5
( postal_code  string
, thursday_hours  string
, friday_hours  string
, latitude  double
, alcohol  string
, business_id  string
, ambience  boolean
, counterservice  boolean
, categories  string
, name  string
, bitcoin  boolean
, creditcards  boolean
, is_open  int
, neighborhood  string
, parkinglot  boolean
, review_count  int
, state  string
, address  string
, sunday_hours  string
, wednesday_hours  string
, monday_hours  string
, city  string
, tuesday_hours  string
, stars  double
, price_range  int
, longitude  double
, saturday_hours  string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
tblproperties ("skip.header.line.count"="2");
```

OK

```
Load data inpath "/users/anjamkudy/yelp_business.csv" OVERWRITE into table
yelp_business_gp5;
```

OK

Partitioning the table to optimze the frequent querries on 'USA Restaurents'

```
drop table yelp_business_gp5_country_cat;
 CREATE TABLE IF NOT EXISTS yelp_business_gp5_country_cat
( postal_code  string
, thursday_hours  string
, friday_hours  string
, latitude  double
, alcohol  string
, business_id  string
, ambience  boolean
, counterservice  boolean
, categories  string
, name  string
, bitcoin  boolean
, creditcards  boolean
, is_open  int
, neighborhood  string
, parkinglot  boolean
, review_count  int
, state  string
, address  string
, sunday_hours  string
, wednesday_hours  string
, monday_hours  string
, city  string
, tuesday_hours  string
, stars  double
, price_range  int
, longitude  double
, saturday_hours  string
)
PARTITIONED BY (country string, category string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
```

OK

Inserting Entire data from Business table into the partition table

```
Insert into table yelp_business_gp5_country_cat partition(country =
'USA',category = 'Restaurant')
select *
from yelp_business_gp5
where latitude between 19.50139 and 64.85694
and  longitude between  -172.4417 and -80.0000
and categories like '%Restaurant%'
and state not in ('ON','QC');
select count(*) from yelp_business_gp5_country_cat
```

	count(1)
1	30309

Showing all 1 rows.

```
Insert into table yelp_business_gp5_country_cat partition(country =
'USA',category = 'NonRestaurant')
select *
from yelp_business_gp5
where latitude between 19.50139 and 64.85694
and  longitude between  -172.4417 and -80.0000
and categories not like '%Restaurant%'
and state not in ('ON','QC');
select count(*) from yelp_business_gp5_country_cat
```

	count(1)
1	122399

Showing all 1 rows.

```
Insert into table yelp_business_gp5_country_cat partition(country = 
'NONUSA',category = 'Restaurant')
select *
from yelp_business_gp5
where     (latitude < 19.50139 OR latitude > 64.85694)  OR  (longitude < 
-172.4417 OR longitude > -80.0000)
and categories  like '%Restaurant%';

select count(*) from yelp_business_gp5_country_cat
```

	count(1)
1	146726

Showing all 1 rows.

```
Insert into table yelp_business_gp5_country_cat partition(country = 
'NONUSA',category = 'NonRestaurant')
select *
from yelp_business_gp5
where     (latitude < 19.50139 OR latitude > 64.85694) OR  (longitude < 
-172.4417 OR longitude > -80.0000)
and categories  not like '%Restaurant%' ;

select count(*) from yelp_business_gp5_country_cat
```

	count(1)
1	174536

Showing all 1 rows.

```
Insert into table yelp_business_gp5_country_cat partition(country =
'NONUSA',category = 'Restaurant')
select *
from yelp_business_gp5
where latitude between 19.50139 and 64.85694
and  longitude between  -172.4417 and -80.0000
and categories like '%Restaurant%'
and state  in ('ON','QC');

select count(*) from yelp_business_gp5_country_cat
```

	count(1)
1	174542

Showing all 1 rows.

```
Insert into table yelp_business_gp5_country_cat partition(country =
'NONUSA',category = 'NonRestaurant')
select *
from yelp_business_gp5
where latitude between 19.50139 and 64.85694
and  longitude between  -172.4417 and -80.0000
and categories not like '%Restaurant%'
and state  in ('ON','QC');

select count(*) from yelp_business_gp5_country_cat
```

	count(1)
1	174549

Showing all 1 rows.

```
select * from yelp_business_gp5_country_cat
where country = 'USA'
and category = 'Restaurant';
```

	postal_code	thursday_hours	friday_hours	latitude	alcohol	b

1	44221	11:00-1:00	11:00-1:00	41.1195346	full_bar	F
2	15342			40.24154801	none	X
3	28202	7:00-15:00	7:00-15:00	35.2216474		fl
4	44035	6:30-21:00	6:30-22:00	41.343078		D
5	29708	7:00-15:00	7:00-15:00	35.0472868		g
6	44107	12:00-2:00	12:00-2:00	41.4768463	full_bar	tl
7	85022			33.6070702	none	rl

Showing the first 1000 rows.

```
select *
from yelp_business_gp5
where latitude between 19.50139 and 64.85694
and  longitude between  -172.4417 and -80.0000
and categories like '%Restaurant%'
and state not in ('ON','QC');
```

	postal_code	thursday_hours	friday_hours	latitude	alcohol	b
1	44221	11:00-1:00	11:00-1:00	41.1195346	full_bar	F
2	15342			40.24154801	none	X
3	28202	7:00-15:00	7:00-15:00	35.2216474		fl
4	44035	6:30-21:00	6:30-22:00	41.343078		D
5	29708	7:00-15:00	7:00-15:00	35.0472868		g
6	44107	12:00-2:00	12:00-2:00	41.4768463	full_bar	tl
7	85022			33.6070702	none	rl

Showing the first 1000 rows.

As we can see above for getting same amount of data from table (4.57 S) vs from partition table (0.56 S), which is almost 800% times faster

	count(1)
1	30309

Showing all 1 rows.

```
%sql
use aportr;
set hive.exec.dynamic.partition =  TRUE;
set hive.exec.dynamic.partition.mode = nonstrict;
set hive.exec.max.dynamic.partitions = 1500;
```

	key	value
1	hive.exec.max.dynamic.partitions	1500

Showing all 1 rows.

```
%r
library(SparkR)
```

```
Attaching package: ‘SparkR’

The following object is masked _by_ ‘.GlobalEnv’:

    setLocalProperty

The following objects are masked from ‘package:stats’:

    cov, filter, lag, na.omit, predict, sd, var, window

The following objects are masked from ‘package:base’:

    as.data.frame, colnames, colnames<-, drop, endsWith, intersect,
    rank, rbind, sample, startsWith, subset, summary, transform, union
```

Clustering Analysis

```
%r
# initial query to run clustering
query = ("select * FROM yelp_business_gp5_country_cat")
us_rest<- sql(query)
```

```
str(us_rest)
```

```
# turning all booleans into 0 and 1
us_rest$sunday_hours_int <- ifelse(us_rest$sunday_hours == NULL, 0, 1)
us_rest$monday_hours_int <- ifelse(us_rest$monday_hours == NULL, 0, 1)
us_rest$tuesday_hours_int <- ifelse(us_rest$tuesday_hours == NULL, 0, 1)
us_rest$wednesday_hours_int <- ifelse(us_rest$wednesday_hours == NULL, 0, 1)
us_rest$thursday_hours_int <- ifelse(us_rest$thursday_hours == NULL, 0, 1)
us_rest$friday_hours_int <- ifelse(us_rest$friday_hours == NULL, 0, 1)
us_rest$saturday_hours_int <- ifelse(us_rest$saturday_hours == NULL, 0, 1)
us_rest$alcohol_int <- ifelse(us_rest$alcohol == NULL, 0, 1)
us_rest$ambience_int <- ifelse(us_rest$ambience == TRUE, 1, 0)
us_rest$creditcards_int <- ifelse(us_rest$creditcards == TRUE, 1, 0)
us_rest$bitcoin_int<- ifelse(us_rest$bitcoin == TRUE, 1, 0)
us_rest$parkinglot_int <- ifelse(us_rest$parkinglot == TRUE, 1, 0)
us_rest$counterservice_int <- ifelse(us_rest$counterservice == TRUE, 1, 0)
```

```
# turning all data types to int
us_rest$sunday_hours_int<- cast(us_rest$sunday_hours_int, "int")
us_rest$monday_hours_int<- cast(us_rest$monday_hours_int, "int")
us_rest$tuesday_hours_int<- cast(us_rest$tuesday_hours_int, "int")
us_rest$wednesday_hours_int<- cast(us_rest$wednesday_hours_int, "int")
us_rest$thursday_hours_int<- cast(us_rest$thursday_hours_int, "int")
us_rest$friday_hours_int<- cast(us_rest$friday_hours_int, "int")
us_rest$saturday_hours_int<- cast(us_rest$saturday_hours_int, "int")
us_rest$alcohol_int <- cast(us_rest$alcohol_int, "int")
us_rest$ambience_int <- cast(us_rest$ambience_int, "int")
us_rest$creditcards_int <- cast(us_rest$creditcards_int, "int")
us_rest$bitcoin_int<- cast(us_rest$bitcoin_int, "int")
us_rest$parkinglot_int <- cast(us_rest$parkinglot_int, "int")
us_rest$counterservice_int<- cast(us_rest$counterservice_int, "int")
```

```
# removing all remaining data that wont be used
us_rest$sunday_hours <- NULL
us_rest$monday_hours <- NULL
us_rest$tuesday_hours <- NULL
us_rest$wednesday_hours <- NULL
us_rest$thursday_hours <- NULL
us_rest$friday_hours <- NULL
us_rest$saturday_hours <- NULL
us_rest$alcohol <- NULL
us_rest$ambience <- NULL
us_rest$creditcards <- NULL
us_rest$bitcoin <- NULL
us_rest$parkinglot <- NULL
us_rest$counterservice <- NULL
us_rest$postal_code <- NULL
us_rest$business_id_drop <- NULL
us_rest$business_id <- NULL
us_rest$postal_code <- NULL
us_rest$latitude <- NULL
us_rest$longitude <- NULL
us_rest$name <- NULL
us_rest$neighborhood <- NULL
us_rest$review_count <- NULL
us_rest$address  <- NULL
us_rest$categories <- NULL
us_rest$postal_code_int <- NULL
us_rest$state <- NULL
us_rest$city <- NULL
us_rest$friday_hours_int <- NULL
us_rest$saturday_hours_int <- NULL
us_rest$sunday_hours_int <- NULL
us_rest$monday_hours_int <- NULL
us_rest$tuesday_hours_int <- NULL
us_rest$wednesday_hours_int <- NULL
us_rest$thursday_hours_int <- NULL
us_rest$alcohol_int <- NULL
```

```
str(us_rest)
```

```
'SparkDataFrame': 10 variables:
 $ is_open          : int 1 1 1 1 1 1
 $ stars            : num 4 2 4 2 2.5 4
 $ price_range      : int 1 NA 2 NA 3 2
 $ country          : chr "NONUSA" "NONUSA" "NONUSA" "NONUSA" "NONUSA" "NONUSA"
 $ category         : chr "NonRestaurant" "NonRestaurant" "NonRestaurant" "NonRest
aurant" "NonRestaurant" "NonRestauran
 $ ambience_int     : int 0 0 0 0 0 0
 $ creditcards_int  : int 0 1 1 0 1 1
 $ bitcoin_int      : int 0 0 0 0 0 0
 $ parkinglot_int   : int 1 0 0 0 0 0
 $ counterservice_int: int 0 0 0 0 0 0
```

```
# dropping NAs
us_rest <- dropna(us_rest)
```

K-means Clustering

```
# building the k-means model
model1 <- spark.kmeans(data = us_rest, ~ ., k=6, maxIter = 20, initMode =
"random")
summary(model1)
```

```
$k
[1] 6
```

Bisecting K-Means

```
# building the bisecting k-means
model2 <- spark.bisectingKmeans(data = us_rest, ~ ., k=6, maxIter = 10, seed=3,
minDivisibleClusterSize = 1)
summary(model2)
```

Association Rule

```
# building the query for the association rule mining
query = ("select rev.user_id, bus.name from yelp_business_gp5_country_cat bus
inner join yelp_reviews_gp5 rev
where bus.business_id=rev.business_id")
review<- sql(query)
showDF(review)
```

```
+--------------------+--------------------+
|             user_id|                name|
+--------------------+--------------------+
|xP1IYu2eGfxMWV9tj...|Delmonico Steakhouse|
|oFyOUOeGTRZhFPF9u...|Delmonico Steakhouse|
|2aeNFntqY2QDZLADN...|Delmonico Steakhouse|
|gmPP4YFrgYsYQqPYo...|Delmonico Steakhouse|
|9bxdPvAhP6cuipD5s...|Delmonico Steakhouse|
|aVOGlN9fZ-BXcbtj6...|Delmonico Steakhouse|
|KC8H7qTZVPIEnanw9...|Delmonico Steakhouse|
|3gEk6-HQ7DxjY99zy...|Delmonico Steakhouse|
|HmN7p502YMJGkBNv5...|Delmonico Steakhouse|
|3RTesI_MAwct13LWm...|Delmonico Steakhouse|
|4PIcs3X-Ro_KoczDJ...|Delmonico Steakhouse|
|EAOt1UQhJD0GG3l_j...|Delmonico Steakhouse|
|C6kw0Rny7jZAGjTj0...|Delmonico Steakhouse|
|TVMmYI09y8-zJKiDf...|Delmonico Steakhouse|
```

```
count(review)
```

```
[1] 5006919
```

```
# aggregating the user ids and the restuarants they reviewed
review_agg <- agg(groupBy(review, review$user_id), name = "collect_set")
colnames(review_agg)[2] <- "items"
showDF(review_agg,20)
```

```
+--------------------+--------------------+
|             user_id|               items|
+--------------------+--------------------+
|--CJT4d-S8UhwqHe0...|[Xtreme Cycle Rep...|
```

```
# looking at the top frequency restaurants
fpm <- spark.fpGrowth(review_agg, itemsCol="items", minSupport=0.0001,
minConfidence=0.0001)
topIS <- spark.freqItemsets(fpm)
showDF(orderBy(topIS,-topIS$freq),10)
```

```
+--------------------+-----+
|               items| freq|
+--------------------+-----+
|         [Starbucks]|14058|
|[Hash House A Go Go]| 8849|
|        [McDonald's]| 7798|
|      [Mon Ami Gabi]| 7150|
|[Chipotle Mexican...| 6745|
|  [Bacchanal Buffet]| 6647|
|      [Wicked Spoon]| 5729|
|[Buffalo Wild Wings]| 5454|
|[Gordon Ramsay Bu...| 5244|
|  [Earl of Sandwich]| 5078|
+--------------------+-----+
only showing top 10 rows
```

```
# seems like starbucks is popular regardless, makes sense because these are all
over the place
topAR<- spark.associationRules(fpm)
showDF(orderBy(topAR, -topAR$confidence), 10)
```

```
+--------------------+-----------+------------------+------------------+
|          antecedent| consequent|        confidence|              lift|
+--------------------+-----------+------------------+------------------+
|[US Post Office, ...|[Starbucks]|0.7043010752688172| 64.66328551301292|
|[Whole Foods Mark...|[Starbucks]|          0.703125| 64.55530769846351|
|[Whole Foods Mark...|[Starbucks]|0.6859903381642513|62.982140243009816|
|    [Target, Subway]|[Starbucks]|0.6821428571428572| 62.62889534174746|
|[Target, Panera B...|[Starbucks]|0.6758409785932722|62.050307311387556|
|[Trader Joe's, Wa...|[Starbucks]|0.6753246753246753|  62.0029044792169|
|[US Post Office, ...|[Starbucks]|0.6745283018867925| 61.92978784657632|
|[Target, The Home...|[Starbucks]|0.6720647773279352| 61.70360674660156|
|[US Post Office, ...|[Starbucks]| 0.670995670995671| 61.60544996332449|
|[Target, Whole Fo...|[Starbucks]|0.6641509433962264| 60.97702187970591|
+--------------------+-----------+------------------+------------------+
only showing top 10 rows
```

Prep for Supervised Learning Models

```
df_bus = sql("SELECT * FROM yelp_business_gp5_country_cat")
df_rev = sql("SELECT cool, useful, funny, business_id as business_id_drop  FROM
yelp_reviews_gp5")
```

```
# creating a data set with all features
df1 = join(df_bus, df_rev, df_bus$business_id == df_rev$business_id_drop)
```

```
str(df1)
```

```
'SparkDataFrame': 31 variables:
 $ postal_code     : chr "85033" "85033" "85033" "85033" "85033" "85033"
 $ thursday_hours  : chr "" "" "" "" "" ""
 $ friday_hours    : chr "" "" "" "" "" ""
 $ latitude        : num 33.508427 33.508427 33.508427 33.508427 33.508427 33.5084
27
 $ alcohol         : chr "" "" "" "" "" ""
 $ business_id     : chr "0BaMoKDVNv-MP84BQ9EK9A" "0BaMoKDVNv-MP84BQ9EK9A" "0BaMoK
DVNv-MP84BQ9EK9A" "0BaMoKDVNv-MP84BQ9E
```

```
# creating new variables with 0 and 1 that will replace the booleans or any
other funky variable
df1$sunday_hours_int <- ifelse(df1$sunday_hours == NULL, 0, 1)
df1$monday_hours_int <- ifelse(df1$monday_hours == NULL, 0, 1)
df1$tuesday_hours_int <- ifelse(df1$tuesday_hours == NULL, 0, 1)
df1$wednesday_hours_int <- ifelse(df1$wednesday_hours == NULL, 0, 1)
df1$thursday_hours_int <- ifelse(df1$thursday_hours == NULL, 0, 1)
df1$friday_hours_int <- ifelse(df1$friday_hours == NULL, 0, 1)
df1$saturday_hours_int <- ifelse(df1$saturday_hours == NULL, 0, 1)
df1$alcohol_int <- ifelse(df1$alcohol == NULL, 0, 1)
df1$ambience_int <- ifelse(df1$ambience == TRUE, 1, 0)
df1$creditcards_int <- ifelse(df1$creditcards == TRUE, 1, 0)
df1$bitcoin_int<- ifelse(df1$bitcoin == TRUE, 1, 0)
df1$parkinglot_int <- ifelse(df1$parkinglot == TRUE, 1, 0)
df1$counterservice_int <- ifelse(df1$counterservice == TRUE, 1, 0)
```

```
# and turning the new variables into integers
df1$sunday_hours_int<- cast(df1$sunday_hours_int, "int")
df1$monday_hours_int<- cast(df1$monday_hours_int, "int")
df1$tuesday_hours_int<- cast(df1$tuesday_hours_int, "int")
df1$wednesday_hours_int<- cast(df1$wednesday_hours_int, "int")
df1$thursday_hours_int<- cast(df1$thursday_hours_int, "int")
df1$friday_hours_int<- cast(df1$friday_hours_int, "int")
df1$saturday_hours_int<- cast(df1$saturday_hours_int, "int")
df1$alcohol_int <- cast(df1$alcohol_int, "int")
df1$ambience_int <- cast(df1$ambience_int, "int")
df1$creditcards_int <- cast(df1$creditcards_int, "int")
df1$bitcoin_int<- cast(df1$bitcoin_int, "int")
df1$parkinglot_int <- cast(df1$parkinglot_int, "int")
df1$counterservice_int<- cast(df1$counterservice_int, "int")
```

```
# removing all of the remaining variables we won't use
df1$sunday_hours <- NULL
df1$monday_hours <- NULL
df1$tuesday_hours <- NULL
df1$wednesday_hours <- NULL
df1$thursday_hours <- NULL
df1$friday_hours <- NULL
df1$saturday_hours <- NULL
df1$alcohol <- NULL
df1$ambience <- NULL
df1$creditcards <- NULL
df1$bitcoin <- NULL
df1$parkinglot <- NULL
df1$counterservice <- NULL
df1$postal_code <- NULL
df1$business_id_drop <- NULL
df1$business_id <- NULL
df1$postal_code <- NULL
df1$latitude <- NULL
df1$longitude <- NULL
df1$name <- NULL
df1$neighborhood <- NULL
df1$review_count <- NULL
df1$address  <- NULL
df1$categories <- NULL
df1$postal_code_int <- NULL
df1$state <- NULL
df1$city <- NULL
```

```
str(df1)
```

```
'SparkDataFrame': 19 variables:
 $ is_open           : int 1 1 1 1 1 1
 $ stars             : num 3.5 3.5 2.5 3 3.5 4
 $ price_range       : int 2 3 3 3 2 2
 $ cool              : int 2 1 0 0 0 0
 $ useful            : int 2 2 0 0 0 1
 $ funny             : int 2 2 0 0 0 2
 $ sunday_hours_int  : int 1 1 1 1 1 1
```

```
# removinga all NAs
df1 <- dropna(df1)
```

```
# splitting data set into 70% training and 30% test
df_list <- randomSplit(df1,c(7,3),2)
training_df <- df_list[[1]]
testing_df <- df_list[[2]]
```

```
count(testing_df)
```

```
[1] 1239020
```

```
count(training_df)
```

```
[1] 1659686
```

```
# building the linear model
model_glm <- spark.glm(training_df, stars ~ ., family = "gaussian")
```

```
# using the model to predict on our test set
Output_glm <- predict(model_glm, testing_df)
```

```
#MSE
MSE_glm = showDF(select(Output_glm, avg((Output_glm$stars-
Output_glm$prediction)^2)))
#RMSE
RMSE_glm = showDF(select(Output_glm, sqrt(avg((Output_glm$stars-
Output_glm$prediction)^2))))
#MAE
MAE_glm = showDF(select(Output_glm, avg(abs(Output_glm$stars-
Output_glm$prediction))))
#MAPE
MAPE_glm = showDF(select(Output_glm, avg(abs(Output_glm$stars-
Output_glm$prediction)/abs(Output_glm$stars))))
```

```
+------------------------------------+
|avg(POWER((stars - prediction), 2.0))|
+------------------------------------+
|                 0.41408803968052954|
+------------------------------------+
+------------------------------------------+
|SQRT(avg(POWER((stars - prediction), 2.0)))|
+------------------------------------------+
|                         0.643496728570184|
+------------------------------------------+
+------------------------------+
|avg(abs((stars - prediction)))|
+------------------------------+
|              0.50936241852966|
+------------------------------+
+--------------------------------------------+
|avg((abs((stars - prediction)) / abs(stars)))|
+--------------------------------------------+
|                          0.1561989108636673|
```

```
# building a random forest model
model_rf <- spark.randomForest(training_df, stars ~ ., "regression", numTrees =
10, maxDepth = 5)
```

```
summary(model_rf)
```

```
Formula:  stars ~ .
Number of features:  20
Features:  is_open price_range country_USA category_Restaurant cool useful funny s
unday_hours_int monday_hours_int tuesday_hours_int wednesday_hours_int thursday_ho
urs_int friday_hours_int saturday_hours_int alcohol_int ambience_int creditcards_i
nt bitcoin_int parkinglot_int counterservice_int
Feature importances:  (20,[0,1,2,3,4,5,6,15,16,17,18,19],[0.1928174153684983,0.046
44044539422076,0.09692901886779604,0.039481049824758425,0.053520613235522714,0.017
029885168803348,0.09712567703308811,0.08898783192424853,0.0218090184069029,0.00431
2212115977097,0.3392222584300986,0.0023245742300852697])
Max Depth:  5
Number of trees:  10
Tree weights:  1 1 1 1 1 1 1 1 1 1
 RandomForestRegressionModel (uid=rfr_6a490ddfd218) with 10 trees
  Tree 0 (weight 1.0):
    If (feature 0 <= 0.5)
     If (feature 18 <= 0.5)
      If (feature 1 <= 1.5)
       If (feature 6 <= 0.5)
```

```
# using the model to predict on the test set
Output_rf <- predict(model_rf, testing_df)
```

```
#MSE
MSE_rf = showDF(select(Output_rf, avg((Output_rf$stars-
Output_rf$prediction)^2)))
#RMSE
RMSE_rf = showDF(select(Output_rf, sqrt(avg((Output_rf$stars-
Output_rf$prediction)^2))))
#MAE
MAE_rf = showDF(select(Output_rf, avg(abs(Output_rf$stars-
Output_rf$prediction))))
#MAPE
MAPE_rf= showDF(select(Output_rf, avg(abs(Output_rf$stars-
Output_rf$prediction)/abs(Output_rf$stars))))
```

```
+-----------------------------------+
|avg(POWER((stars - prediction), 2.0))|
+-----------------------------------+
|                0.40908591741124595|
+-----------------------------------+
+-----------------------------------------+
|SQRT(avg(POWER((stars - prediction), 2.0)))|
+-----------------------------------------+
|                       0.6395982468794343|
+-----------------------------------------+
+------------------------------+
|avg(abs((stars - prediction)))|
+------------------------------+
```

```
# creating a decision tree model
model_dt <- spark.decisionTree(training_df, stars ~ ., "regression")
```

```
# using the model to predict on our test set
Output_dt <- predict(model_dt, testing_df)
```

```
#MSE
MSE_dt = showDF(select(Output_dt, avg((Output_dt$stars-
Output_dt$prediction)^2)))
#RMSE
RMSE_dt = showDF(select(Output_dt, sqrt(avg((Output_dt$stars-
Output_dt$prediction)^2))))
#MAE
MAE_dt = showDF(select(Output_dt, avg(abs(Output_dt$stars-
Output_dt$prediction))))
#MAPE
MAPE_dt = showDF(select(Output_dt, avg(abs(Output_dt$stars-
Output_dt$prediction)/abs(Output_dt$stars))))
```

```
+-----------------------------------+
|avg(POWER((stars - prediction), 2.0))|
+-----------------------------------+
|                0.40896232802188426|
+-----------------------------------+
+-----------------------------------------+
|SQRT(avg(POWER((stars - prediction), 2.0)))|
+-----------------------------------------+
|                       0.6395016247218488|
+-----------------------------------------+
+-----------------------------+
|avg(abs((stars - prediction)))|
+-----------------------------+
```

```
# creating the final regression model Gradient Boosted Tree
model_gbt <- spark.gbt(training_df, stars ~ ., "regression", maxIter = 10)
```

```
# applying the model to the test set
Output_gbt <- predict(model_gbt, testing_df)
```

```
#MSE
MSE_gbt = showDF(select(Output_gbt, avg((Output_gbt$stars-
Output_gbt$prediction)^2)))
#RMSE
RMSE_gbt = showDF(select(Output_gbt, sqrt(avg((Output_gbt$stars-
Output_gbt$prediction)^2))))
#MAE
MAE_gbt = showDF(select(Output_gbt, avg(abs(Output_gbt$stars-
Output_gbt$prediction))))
#MAPE
MAPE_gbt = showDF(select(Output_gbt, avg(abs(Output_gbt$stars-
Output_gbt$prediction)/abs(Output_gbt$stars))))
```

```
+-----------------------------------+
|avg(POWER((stars - prediction), 2.0))|
+-----------------------------------+
|                0.40087953070708615|
+-----------------------------------+
+-----------------------------------------+
|SQRT(avg(POWER((stars - prediction), 2.0)))|
+-----------------------------------------+
|                       0.6331504803023419|
+-----------------------------------------+
+-----------------------------+
|avg(abs((stars - prediction)))|
+-----------------------------+
```

Moving to Classification Models

```
str(df1)
```

```
'SparkDataFrame': 19 variables:
 $ is_open           : int 1 1 0 1 1 0
 $ stars             : num 3.5 3 4 2.5 3.5 4
 $ price_range       : int 2 3 2 1 2 1
 $ cool              : int 0 0 0 0 0 0
 $ useful            : int 5 0 2 0 0 1
 $ funny             : int 1 0 0 0 0 0
 $ sunday_hours_int  : int 1 1 1 1 1 1
 $ monday_hours_int  : int 1 1 1 1 1 1
 $ tuesday_hours_int : int 1 1 1 1 1 1
 $ wednesday_hours_int: int 1 1 1 1 1 1
 $ thursday_hours_int : int 1 1 1 1 1 1
 $ friday_hours_int  : int 1 1 1 1 1 1
 $ saturday_hours_int : int 1 1 1 1 1 1
 $ alcohol_int       : int 1 1 1 1 1 1
 $ ambience_int      : int 0 0 0 0 0 0
 $ creditcards_int   : int 1 1 1 1 1 1
 $ bitcoin_int       : int 0 0 0 0 0 0
 $ parkinglot_int    : int 0 0 1 1 1 1
```

```
# new spark DF for classification
df1_class <- df1
```

```
# trying to figure out what the average rating is so we can draw a line between relatively good and relatively bad
%sql
select avg(stars) from yelp_business_gp5
```

	avg(stars)
1	3.6322108219649203

Showing all 1 rows.

```
# landed on something just under the average
df1_class$good_bad_stars <- ifelse(df1_class$stars>3.5, 1, 0)
```

```
# removing the old stars variable, we will be predicting good and bad now
df1_class$stars <- NULL
```

```
# making sure there are no NAs
df1_class <- dropna(df1_class)
```

```
str(df1_class)
```

```
'SparkDataFrame': 19 variables:
 $ is_open           : int 1 0 0 0 0 1
 $ price_range       : int 2 1 2 2 1 1
 $ cool              : int 0 1 0 1 0 0
 $ useful            : int 2 2 0 1 1 1
 $ funny             : int 1 0 0 0 0 0
 $ sunday_hours_int  : int 1 1 1 1 1 1
 $ monday_hours_int  : int 1 1 1 1 1 1
 $ tuesday_hours_int : int 1 1 1 1 1 1
```

```
# randomly splitting the training and test set
df_list <- randomSplit(df1_class,c(7,3),2)
training_df <- df_list[[1]]
testing_df <- df_list[[2]]
```

```
# Logistic classification model
model_log <- spark.logit(training_df, good_bad_stars  ~ ., maxIter = 10,
regParam = 0.3, elasticNetParam = 0.8)
# applying the model to the test set
Output_log <- predict(model_log, testing_df)
```

```
# figuring our the number of correct, total and accuracy
correct_log <- nrow(where(Output_log, Output_log$good_bad_stars ==
Output_log$prediction))
total_log <- nrow(Output_log)
accuracy_log = correct_log/total_log
```

```
accuracy_log
```

```
[1] 0.5477474
```

```
# calculating the precision and recall
TP_log <- nrow(where(Output_log, Output_log$good_bad_stars == 1 &
Output_log$prediction == 1))
FP_log <- nrow(where(Output_log, Output_log$good_bad_stars == 0 &
Output_log$prediction == 1))
FN_log <- nrow(where(Output_log, Output_log$good_bad_stars == 1 &
Output_log$prediction == 0))
Precision_log = TP_log/(TP_log+FP_log)
Recall_log = TP_log/(TP_log+FN_log)
```

```
Precision_log
```

```
[1] 0.5477474
```

```
Recall_log
```

```
[1] 1
```

Now taking a look at how well the model works on the training data-indicating overfitting

```
Output_log_train <- predict(model_log, training_df)
correct_log_train <- nrow(where(Output_log_train,
Output_log_train$good_bad_stars == Output_log_train$prediction))
total_log_train <- nrow(Output_log_train)
accuracy_log_train = correct_log_train/total_log_train
```

```
# accuracy against the training set isn't great, which is an indicator that the
model is not over fit
accuracy_log_train
```

```
[1] 0.5470344
```

```
# decision tree classifier
model_dt <- spark.decisionTree(training_df, good_bad_stars  ~ .,
"classification")
# applying the model to the testing set
Output_dt <- predict(model_dt, testing_df)
```

```
# calculating the correct, total, and accuracy
correct_dt <- nrow(where(Output_dt, Output_dt$good_bad_stars ==
Output_dt$prediction))
total_dt <- nrow(Output_dt)
accuracy_dt = correct_dt/total_dt
```

```
accuracy_dt
```

```
[1] 0.5878113
```

```
# calculating the precision and recall
TP_dt <- nrow(where(Output_dt, Output_dt$good_bad_stars == 1 &
Output_dt$prediction == 1))
FP_dt <- nrow(where(Output_dt, Output_dt$good_bad_stars == 0 &
Output_dt$prediction == 1))
FN_dt <- nrow(where(Output_dt, Output_dt$good_bad_stars == 1 &
Output_dt$prediction == 0))
Precision_dt = TP_dt/(TP_log+FP_dt)
Recall_dt = TP_dt/(TP_dt+FN_dt)
```

```
Precision_dt
```

```
[1] 0.5494938
```

```
Recall_dt
```

```
[1] 0.9178629
```

Checking for overfitting

```
Output_dt_train <- predict(model_log, training_df)
correct_dt_train <- nrow(where(Output_dt_train, Output_dt_train$good_bad_stars
== Output_dt_train$prediction))
total_dt_train <- nrow(Output_dt_train)
accuracy_dt_train = correct_dt_train/total_dt_train
```

```
# accuracy against the training set isn't great, which is an indicator that the
model is not over fit
accuracy_dt_train
```

```
[1] 0.5470344
```

```
# building the random forest model
model_rf <- spark.randomForest(training_df, good_bad_stars ~ . ,
"classification", numTrees = 20, maxDepth = 5)
Output_rf <- predict(model_rf, testing_df)
```

```
# calculating the accuracy
correct_rf <- nrow(where(Output_rf, Output_rf$good_bad_stars ==
Output_rf$prediction))
total_rf <- nrow(Output_rf)
accuracy_rf = correct_rf/total_rf
```

```
accuracy_rf
```

```
[1] 0.5914723
```

```
# calculating the precision and recall
TP_rf <- nrow(where(Output_rf, Output_rf$good_bad_stars == 1 &
Output_rf$prediction == 1))
FP_rf <- nrow(where(Output_rf, Output_rf$good_bad_stars == 0 &
Output_rf$prediction == 1))
FN_rf <- nrow(where(Output_rf, Output_rf$good_bad_stars == 1 &
Output_rf$prediction == 0))
Precision_rf = TP_rf/(TP_rf+FP_rf)
Recall_rf = TP_rf/(TP_rf+FN_rf)
```

```
Precision_rf
```

```
[1] 0.5834422
```

```
Recall_rf
```

```
[1] 0.8885924
```

Checking for overfitting

```
Output_rf_train <- predict(model_rf, training_df)
correct_rf_train <- nrow(where(Output_rf_train, Output_rf_train$good_bad_stars
== Output_rf_train$prediction))
total_rf_train <- nrow(Output_rf_train)
accuracy_rf_train = correct_rf_train/total_rf_train
```

```
# accuracy against the training set isn't great, which is an indicator that the
model is not over fit
accuracy_rf_train
```

```
[1] 0.5909957
```

```
# final classification model gradient boosted classifier
model_gbt <- spark.gbt(training_df, good_bad_stars ~ ., "classification",
maxIter = 10)
```

```
Output_gbt <- predict(model_gbt, testing_df)
```

```
# GBT accuracy
correct_gbt <- nrow(where(Output_gbt, Output_gbt$good_bad_stars ==
Output_gbt$prediction))
total_gbt <- nrow(Output_gbt)
accuracy_gbt = correct_gbt/total_gbt
```

```
accuracy_gbt
```

```
[1] 0.6020678
```

```
# GBT precision and recall
TP_gbt <- nrow(where(Output_gbt, Output_gbt$good_bad_stars == 1 &
Output_gbt$prediction == 1))
FP_gbt <- nrow(where(Output_gbt, Output_gbt$good_bad_stars == 0 &
Output_gbt$prediction == 1))
FN_gbt <- nrow(where(Output_gbt, Output_gbt$good_bad_stars == 1 &
Output_gbt$prediction == 0))
Precision_gbt = TP_gbt/(TP_gbt+FP_gbt)
Recall_gbt = TP_gbt/(TP_gbt+FN_gbt)
```

```
Precision_gbt
```

```
[1] 0.5994152
```

```
Recall_gbt
```

```
[1] 0.8245569
```

Checking for overfitting

```
Output_gbt_train <- predict(model_gbt, training_df)
correct_gbt_train <- nrow(where(Output_gbt_train,
Output_gbt_train$good_bad_stars == Output_gbt_train$prediction))
total_gbt_train <- nrow(Output_gbt_train)
accuracy_gbt_train = correct_gbt_train/total_gbt_train
```

```
# accuracy against the training set isn't great, which is an indicator that the
model is not over fit
accuracy_gbt_train
```

```
[1] 0.6021634
```

Switching to run a model to predict if restuarant is open or closed

```
# creating a new dataset used to predict of a restaurant is open or closed
df1_isopen <- df1
```

```
df1_isopen$country <- NULL
df1_isopen$category <- NULL
```

```
str(df1_isopen)
```

```
'SparkDataFrame': 19 variables:
 $ is_open            : int 1 1 1 1 1 1
 $ stars              : num 4 4 4 4 4 4
 $ price_range        : int 4 4 4 4 4 4
 $ cool               : int 0 0 0 0 8 0
 $ useful             : int 1 0 0 1 9 0
 $ funny              : int 0 0 0 0 7 0
 $ sunday_hours_int   : int 1 1 1 1 1 1
 $ monday_hours_int   : int 1 1 1 1 1 1
 $ tuesday_hours_int  : int 1 1 1 1 1 1
 $ wednesday_hours_int: int 1 1 1 1 1 1
 $ thursday_hours_int : int 1 1 1 1 1 1
 $ friday_hours_int   : int 1 1 1 1 1 1
 $ saturday_hours_int : int 1 1 1 1 1 1
 $ alcohol_int        : int 1 1 1 1 1 1
 $ ambience_int       : int 0 0 0 0 0 0
 $ creditcards_int    : int 1 1 1 1 1 1
 $ bitcoin_int        : int 0 0 0 0 0 0
```

```
# randomly splitting the data set into a training and test set
df_list <- randomSplit(df1_isopen,c(7,3),2)
training_df <- df_list[[1]]
testing_df <- df_list[[2]]
```

```
# first model logistic
model_log <- spark.logit(training_df, is_open  ~ ., maxIter = 10, regParam =
0.3, elasticNetParam = 0.8)
# applying the model to the test set
Output_log <- predict(model_log, testing_df)
```

```
# calculating the accuracy
correct_log <- nrow(where(Output_log, Output_log$is_open ==
Output_log$prediction))
total_log <- nrow(Output_log)
accuracy_log = correct_log/total_log
```

```
accuracy_log
```

```
[1] 0.8712181
```

```
# precision and recall
TP_log <- nrow(where(Output_log, Output_log$is_open == 1 &
Output_log$prediction == 1))
FP_log <- nrow(where(Output_log, Output_log$is_open == 0 &
Output_log$prediction == 1))
FN_log <- nrow(where(Output_log, Output_log$is_open == 1 &
Output_log$prediction == 0))
Precision_log = TP_log/(TP_log+FP_log)
Recall_log = TP_log/(TP_log+FN_log)
```

```
Precision_log
```

```
[1] 0.8712181
```

```
Recall_log
```

```
[1] 1
```

```
# decision tree classifier
model_dt <- spark.decisionTree(training_df, is_open  ~ ., "classification")
# applying the model to the test set
Output_dt <- predict(model_dt, testing_df)
```

```
# calculating the accuracy
correct_dt <- nrow(where(Output_dt, Output_dt$is_open == Output_dt$prediction))
total_dt <- nrow(Output_dt)
accuracy_dt = correct_dt/total_dt
```

```
accuracy_dt
```

```
[1] 0.8712348
```

```
# precision and recall
TP_dt <- nrow(where(Output_dt, Output_dt$is_open == 1 & Output_dt$prediction ==
1))
FP_dt <- nrow(where(Output_dt, Output_dt$is_open == 0 & Output_dt$prediction ==
1))
FN_dt <- nrow(where(Output_dt, Output_dt$is_open == 1 & Output_dt$prediction ==
0))
Precision_dt = TP_dt/(TP_log+FP_dt)
Recall_dt = TP_dt/(TP_dt+FN_dt)
```

```
Precision_dt
```

```
[1] 0.8712327
```

```
Recall_dt
```

```
[1] 1
```

```
# building the random forest model
model_rf <- spark.randomForest(training_df, is_open ~ ., "classification",
numTrees = 20, maxDepth = 5)
# applying the model to the testing set
Output_rf <- predict(model_rf, testing_df)
```

```
# accuracy of the RF model
correct_rf <- nrow(where(Output_rf, Output_rf$is_open == Output_rf$prediction))
total_rf <- nrow(Output_rf)
accuracy_rf = correct_rf/total_rf
```

```
accuracy_rf
```

```
[1] 0.8712181
```

```
# precision and recall
TP_rf <- nrow(where(Output_rf, Output_rf$is_open == 1 & Output_rf$prediction ==
1))
FP_rf <- nrow(where(Output_rf, Output_rf$is_open == 0 & Output_rf$prediction ==
1))
FN_rf <- nrow(where(Output_rf, Output_rf$is_open == 1 & Output_rf$prediction ==
0))
Precision_rf = TP_rf/(TP_rf+FP_rf)
Recall_rf = TP_rf/(TP_rf+FN_rf)
```

```
Precision_rf
```

```
[1] 0.8712181
```

```
Recall_rf
```

```
[1] 1
```

```
# final model for is open GBT
model_gbt <- spark.gbt(training_df, is_open ~ ., "classification", maxIter =
10)
# applying the model to the testing set
Output_gbt <- predict(model_gbt, testing_df)
```

```
# calculating the accuracy of the model
correct_gbt <- nrow(where(Output_gbt, Output_gbt$is_open ==
Output_gbt$prediction))
total_gbt <- nrow(Output_gbt)
accuracy_gbt = correct_gbt/total_gbt
```

```
accuracy_gbt
```

```
[1] 0.8713175
```

```
# calculating the precision and recall
TP_gbt <- nrow(where(Output_gbt, Output_gbt$is_open == 1 &
Output_gbt$prediction == 1))
FP_gbt <- nrow(where(Output_gbt, Output_gbt$is_open == 0 &
Output_gbt$prediction == 1))
FN_gbt <- nrow(where(Output_gbt, Output_gbt$is_open == 1 &
Output_gbt$prediction == 0))
Precision_gbt = TP_gbt/(TP_gbt+FP_gbt)
Recall_gbt = TP_gbt/(TP_gbt+FN_gbt)
```

```
Precision_gbt
```

```
[1] 0.8713072
```

```
Recall_gbt
```

```
[1] 0.9999962
```

Collaborative Filtering

```
# building the sparkDFs used for the collaborative filtering
df_rev = sql("SELECT user_id, business_id, stars FROM yelp_reviews_gp5")
df_us = sql("SELECT distinct(user_id) FROM yelp_reviews_gp5")
```

```
# will need a list of distinct business id for the data set
df_bus_unique = sql("SELECT distinct(business_id) as business_id FROM
yelp_business_gp5_country_cat")
```

```
# dropping all NAs
df_bus_unique <- dropna(df_bus_unique)
df_rev <- dropna(df_rev)
df_us <- dropna(df_us)
```

```
# converting to R dataframes
df_us <- as.data.frame(df_us)
df_rev <- as.data.frame(df_rev)
df_bus <- as.data.frame(df_bus_unique)
```

```
# creating new columns for the unique items, essentially turning the unique
string into a number
df_bus$b_id <- 1:nrow(df_bus)
df_us$id <- 1:nrow(df_us)
```

```
# joing the two datasets
df = merge(df_rev, df_us, by = "user_id")
```

```
# joing the two datasets
df_coll = merge(df, df_bus, by = "business_id")
```

```
# droping the columns that are strings and keeps the numbers/integers
df_coll$business_id <- NULL
df_coll$user_id <- NULL
```

```
# back to spark data frame and getting it into an appropiate order
df_coll <- as.DataFrame(df_coll)
df_coll <- df_coll[,c(2,3,1)]
```

```
# splitting into test and training
df_list <- randomSplit(df_coll, c(7,3), 2)
training <- df_list[[1]]
testing <- df_list[[2]]
```

```
# building the ALS model
model <- spark.als(data = training, userCol = "id", itemCol = "b_id", rank =
10, regParam = 0.01, maxIter = 5, ratingCol = "stars")
stats <- summary(model)
```

```
+-------+----+-----+------------+
|     id|b_id|stars|  prediction|
+-------+----+-----+------------+
|1186198| 148|    4|   1.4599621|
|1196731| 148|    5|         NaN|
| 707390| 148|    1|  0.04368615|
| 137790| 463|    5| -0.13948959|
| 112922| 496|    1|  -0.9333067|
|1312076| 833|    4|   0.6817585|
| 681925| 833|    4|   0.7641014|
|1132418|1088|    5|   2.4684827|
```