FLIGHT DATA ANALYSIS PROGRAMMING REPORT

St2195 coursework

Table of Contents

For all questions, the same database connection can be created at the beginning to reduce duplicated work. Hence, I set the working directory, created a database, and read and loaded csv files needed, including flights csv, plane-data csv and airports csv, into the database as tables. In this report, I use two years of flights data for 2007 and 2008 and noticed that there are only four months of data in 2008.

Question one – When is the best time of day, day of the week, and time of year to fly to minimize delay?

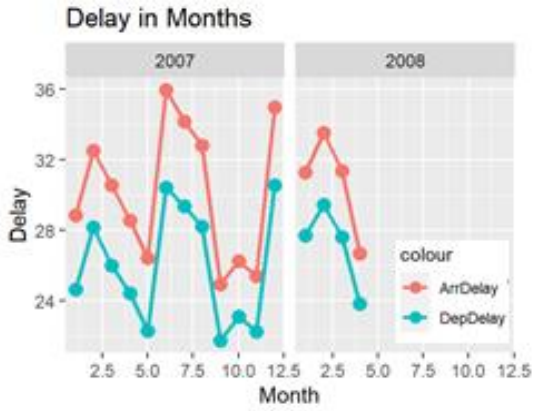
From the result graphs, R (right-hand side) has higher delays than Python. The arrival delays are higher than departure delays in R, inverse in Python. This results from ignoring early departures and arrivals in R, indicating lower delays. Usually, planes do not take off early but may land ahead of schedule. Hence, when considering early flights, arrival delays will decrease.

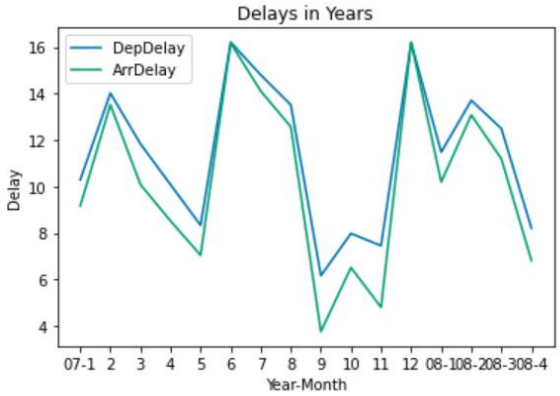
Delay in year

The best time to fly in the year is from September to November, followed by May. In contrast, public holidays such as summer vacation and Christmas had high delay.

In R, after loading the first year and second-year data frame, “NA” is assigned to the negative delay, which means earlier than the schedule. Then, I calculated the average monthly arrival and departure delays without “NA” for two years. As the “aggregate ()” function generates new tables with columns of “Group.1” and “x”, they need to be renamed for identical reason and be combined for two years. By adding the year column, it is now available to combine two years table with clearly stated years and months. Finally, use the total delay table to plot the line chart below.

In Python, the average monthly delays for two years are calculated with grouped “df1” and “df2” based on months. Only four relevant columns were selected after calculating the mean. Then, two years' data are combined into one table with a new column “period” used to identify the same months in different years. The line chart is plotted using “matplotlib”.



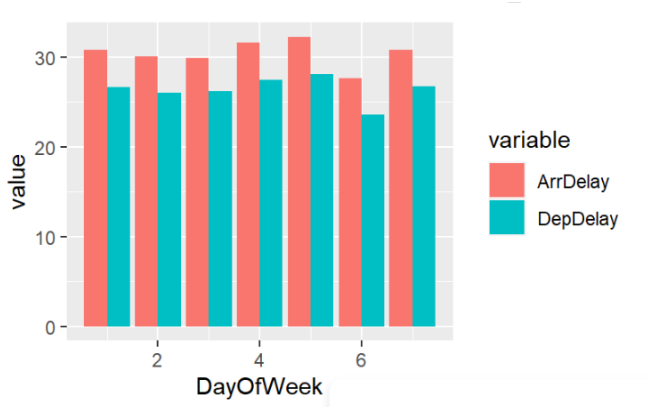


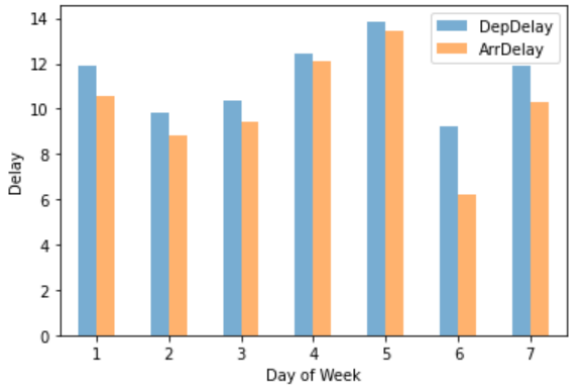
Delay in week

Saturdays have the lowest delay and Fridays have the highest.

In R, the negative delays were assigned to be “NA”. Calculate the mean of departure and arrival delays with grouped tables based on “DayOfWeek”. Same as calculating the average monthly delay, we renamed and combined average departure and arrival delay tables. The grouped bar chart can be plotted with “ggplot” by melting the table with the specified id variable-“DayOfWeek”.

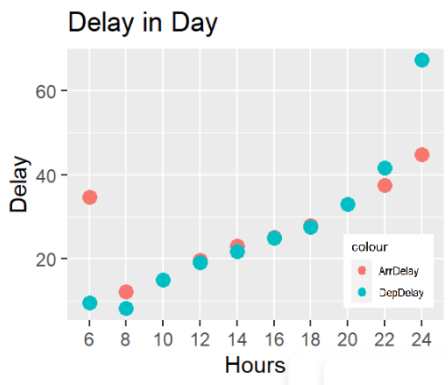
In python, the average departure and arrival delays are calculated with grouped table based on "DayOfWeek", and three relevant columns were selected after the calculation. Next, use “ax” to plot the bar chart.





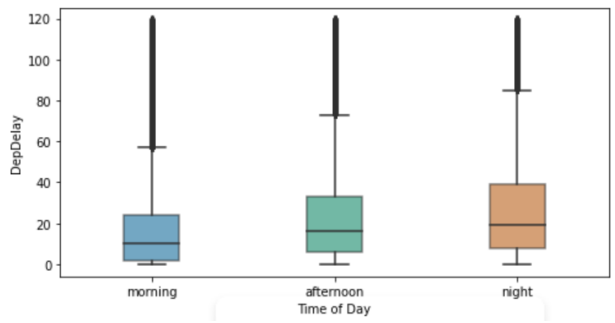
Delay in day

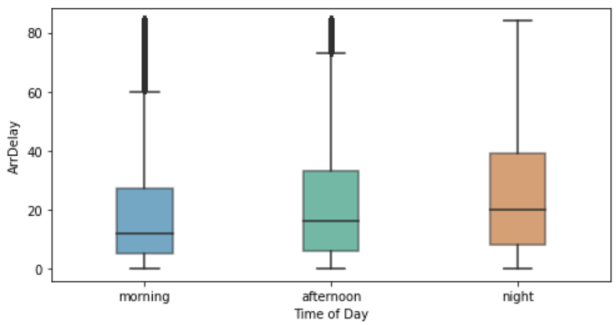
Morning has lower travel delays. Since the departure schedule is from 6 a.m. to 24 p.m., the flights that depart after midnight are viewed as departure delays and can be ignored.



In R, removing negative delay and extracting relevant columns make the table clearer. A two-hour time interval is set with 12 labels for departure and arrival times. Average arrival and departure delays are calculated by ignoring “NA” and grouped data frames based on time intervals. The chart can be plotted with “ggplot” after combining arrival and departure delays by time interval and removing data from 12 a.m. to 6 a.m. From the graph, the arrival delays are not higher than departure delays in terms of hours.

In python, the time intervals are set by assigning 6:00-12:00 to “morning”, 12:00-18:00 to “afternoon”, and 18:00-24:00 to “night”. To plot a significant boxplot using “seaborn”, the outliers less or more than the boundary values should be removed. The boxplots for departure and arrival delays are shown. As time goes by, the medians of delay rise and the whiskers become wider, implying more unstable delays. Notice that arrival delays have a more extensive interquartile range, meaning 50% of the delays spread more widely than departure delays.





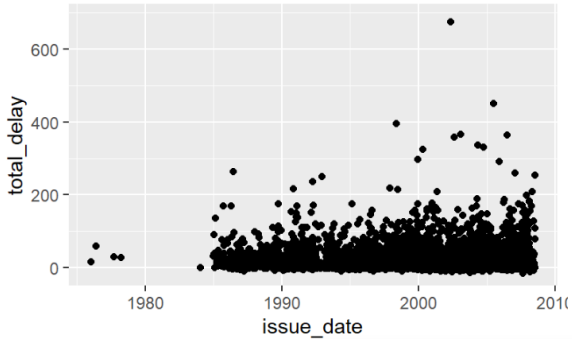
Question two – Do older planes suffer more delays?

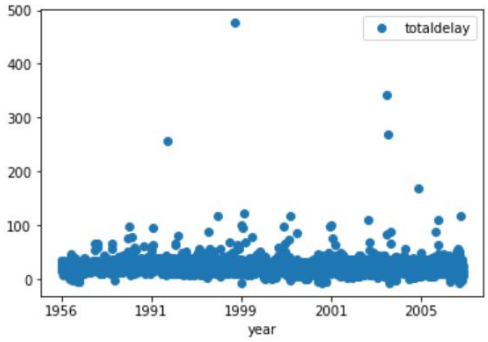
From the graphs, it is hard to state the relationship between the issue date and total delay; however, from two-sample t-tests, the results differ based on different sets of old and new planes. With the strictly defined old-plane group, data reveal more delays for old plans compared to very new planes.

Test with graph

In R, the prepared plane table with “TailNum” and “issue_date” columns is extracted from the original plane table. The diverted flights were eliminated from two-year flights data since they did not have arrival delays. A new column, "total delay", is added, which sums up the departure and arrival delays for each flight used in calculating the average total delay for each tail number. After merging the plane and flights tables by tail number and transferring the issue date to DateTime, the relationship between the issue date of planes and delay can be plotted with "ggplot".

In python, a column of total delay is added to the two-year flights table by summing the arrival and departure delays. The average total delay for each tail number is calculated with grouped table based on “TailNum”, and two relevant columns are selected to reduce complexity. Column of planes table is renamed to merge with flights data on ”TailNum”. To plot the x-axis with ascending order of years, it needs to remove “NA” from the combined table and rearrange rows depending on years. Then, the first three rows with ”0000” of the year are removed.





Two sample t-test

In R, the one-tailed two-sample t-test has a hypothesis as H_0: $\overline{X}_{old} <= \overline{X}_{new}$ and H_α: $\overline{X}_{old} > \overline{X}_{new}$. The equality of variances needs to be tested firstly with the f-test. A classification column is added, which classifies planes manufactured before 2001/10/09 (the median of issue date) as old planes and new after that. I deleted rows containing “NA”, and split data into new and old tables, which will be extracted as new and old vectors to conduct the f-test. The two-sample t-test with equal variance shows that the p-value(might state the values) is too big to reject the null hypothesis; the result concluded that there is no significant evidence to show old planes have more delays than new planes.

In python, we can analyze the distribution of the translated issue date with “describe()” function. Next, planes issued before 1998/04/02, which are the 25% oldest of all planes, are defined as old planes. On the other hand, planes issued after 2004/11/02, the 25% newest planes, are defined as new planes. Those two groups are put into two lists. The defined f-test function shows the equal variances of the two groups. The defined function "f-test()" transfers two parameters to NumPy arrays, calculates the f statistic and degree of freedom of numerator and denominator, and finally finds the p-value with cumulative distribution function. The one-tailed two-sample t-test with H_0: $\overline{X}_{old} <= \overline{X}_{new}$ and H_α: $\overline{X}_{old} >$

$\bar{X}_{new}$ is conducted. In “ttest_ind()” function, we set “equal_var = True” from the f-test result and set “alternative = ’greater’” as we want one tailed test. Since this p-value is less than the 0.05 significant level, we would reject the null hypothesis and conclude that old planes suffer more delays than new planes.

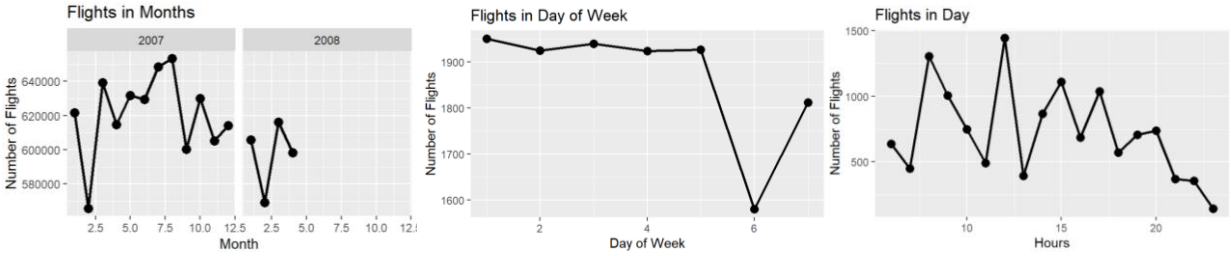
Question three – How does the number of people flying between different locations change over time?

It is hard to calculate the number of people flying, so it is assumed that each flight has the same number of passengers. By that, we can detect the number of passengers change over time and people's movement between airports.

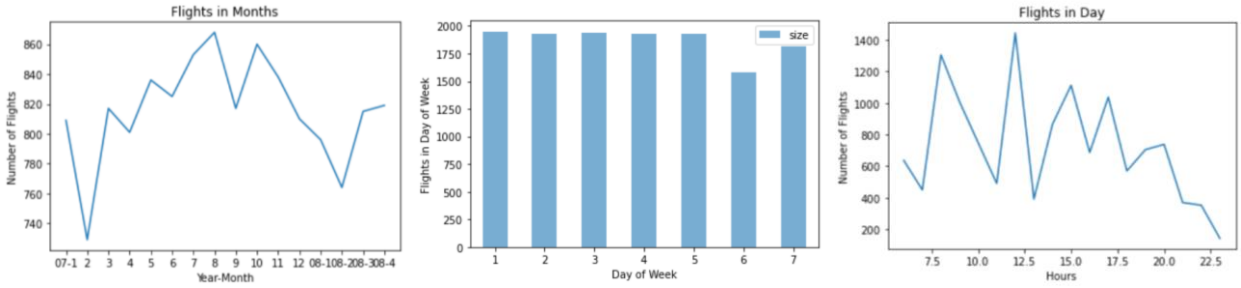
Flights change over time

The numbers of flights are graphed in different time frames, showing traffic trends. Many people travel in summer, particularly in August, while there are fewer flights in February. The stable number of flights on the weekday is much higher than flights on the weekend, especially on Saturday. The low average delay on Saturday may be due to fewer flights. Generally, flights decrease over time after lunchtime, with the highest number of flights; however, it goes up and down every two to three hours.

In R, the “A” function is defined to check whether the airport assigned is in the “Origin” column and can extract rows containing the designated airport. Function “B” works in the same way but focuses on the destinations of flights. After extracting flight data from A to B, the traffic trend in months is plotted according to the “month” table that counts the flights based on year and month. The number of flights over the week is plotted with the “week” table, which counts flights based on the “DayOfWeek” column. The time interval is decided first to plot the x-axis as hours. A new column of “time” is created by dividing “CRSDepTime” by 100 then getting the integer of the results with the”floor()” method. Finally, the number of flights in the day is plotted with the “day” table that counts flights based on hours.



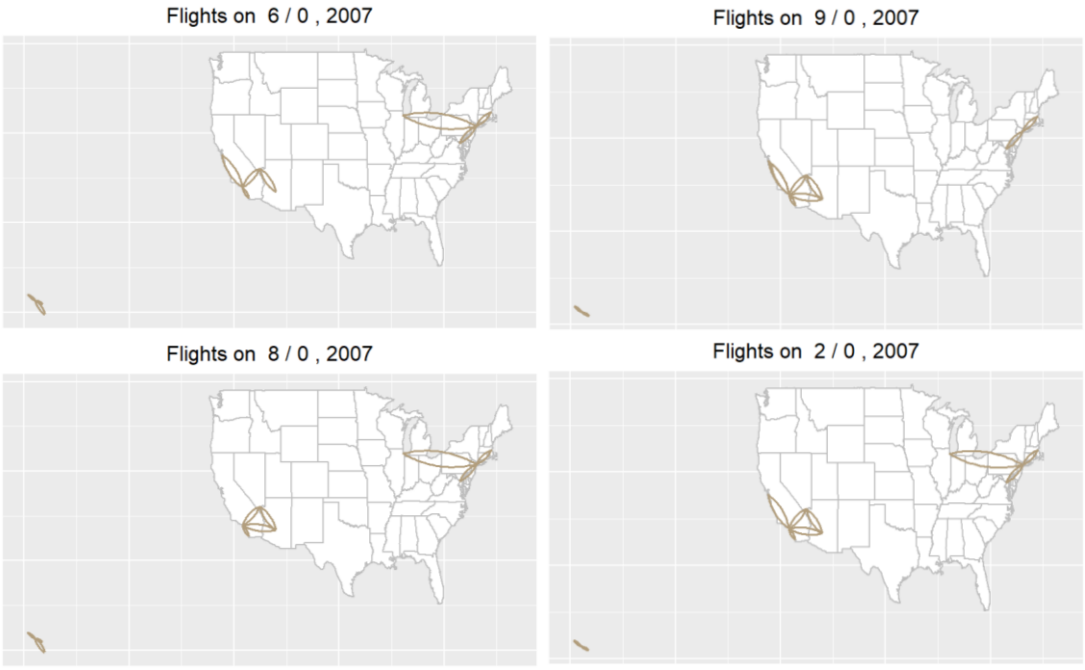
In python, two airports as origin and destination are input. The original two-year data are filtered and left only flight data from A to B. After extracting the data, use the "size()" method to count flights by month, then create a new list “period” containing the x-axis labels to plot the chart. The number of flights in the week is plotted with a “week” table, which uses the “size()” method to count flights by day of the week. The “CRSDepTime” is transferred to hours by dividing it by 100 and taking the integer with the “int()” method. The calculated results are stored in a new object, “time”, which is later placed in the data as a new column, “time”. The number of flights in the day is plotted with the “day” table that uses the “size()” method to count flights by hours.



People movement

The maps denote the 20 most busy routes in a given time frame. There is no noticeable change in popular locations during different months. The busiest routes include flights between New York and Chicago, the District of Columbia and Boston, and flights between Las Vegas and San Francisco, Las Vegas and Phoenix. From python maps, we can see that people tend to go to resorts such as Hawaii and Miami on the weekend. On weekdays, people fly to Taxes.

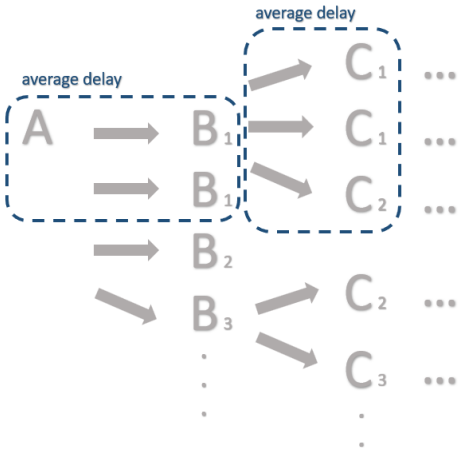
In R, values are assigned to year, month, and day variables if the assigned values are in the defined period. The original two-year data are filtered by the defined period. In the case of entering zero as the date, the data for the whole month will be filtered. Then, it is merged with the "airport_code" table to get the coordinates of the origins and destinations. The new “flight” column is created by the “mutate()” method, joining the “Origin” and “Dest” columns, and was used to calculate the “count” of each flight. The duplicated data are removed, and only the top 20 count of flights are reserved for presenting major routes. August and February are selected to showcase different moves from the busiest and most leisurely months. June and September were chosen because of the highest and lowest delays.



In python, users could enter desired year and month, while if they enter ”0” as the date, the data for the whole month will be extracted. The input number will convert from string to integer to compare with “Year”, “Month” and “DayofMonth” columns of two-year data. If it fails to extract rows with column values equal to entered values, the except statement will be printed. Merging the extracted data with the “airports” table to get the coordinates of origins and destinations. Next, creates a “flight” column that adds destination to origin representing the route and a “count” column to calculate the number of each route. Data is cleaned by removing duplicated values, and only the top 20 flights are kept to present the major routes. Two weekdays (left) and two weekends (right) are randomly selected and plotted.

Question four – Can you detect cascading failures as delays in one airport create delays in others?

There are cascading failures in ATL and its sequential airports, and the strength of causality varies in different situations. If the flight sets out behind schedule, it will undoubtedly land lately. On the other hand, the departure or arrival delay might result in departure or arrival delays in sequential airports with moderate influence.



Users can pick an airport as airport A, and the flights from A to B_1, B_2, to B_n will be found to calculate the average delay for identical flights. Then, the average delay of flights from B to C will be calculated. For example, B_1 flights to C_1 and C_2, the average delay will be calculated for all flights starting from B_1.

There are four cases where cascading delays can happen. The departure or arrival delay in one airport causes a departure or arrival delay in the destination airport, as shown in the graph below. The capital alphabets denote airports, the grey arrows represent flights from A through B, C, and D, and the coloured words label where the delay happened.

In the first situation, a flight takes off lately, resulting in a late landing at the destination. Second, the departure delay in airport A leads to departure delay in airport B flying to C. Third, the flight arrives late at B may cause late arrival in airport C flying from B. Finally, the late arrival to B creates a departure delay in C which will fly to D.

Case one

In R, an airport was assigned to A with the "trycatch()" method to pull out flights taking off from A. Bs group the original data with only flights from A. Average departure and arrival delays for each destination are calculated into a new column created with the"summaries()" method. The correlation coefficient of 0.8731652 is generated with the Pearson method by testing the correlation of "arrdelay" and "depdelay" in case one. The result implies that destination arrival delays strongly rely on departure delays in origin.

In python, only if users input the correct airport will the input transfer into the upper case. The original data are filtered where the origin equals the input; otherwise, the error statement will pop up. The average departure and arrival delays of flights from the input are calculated. The same result as in R is shown by testing the correlation of departure and arrival delay—arrival delays vary with departure delays.

Case two to four

In R, the same procedure is implemented for cases two, three, and four. Calculating delays in B(with "sammarize()" method similar to case one) and delays in C with a defined function, then testing the correlation. The two different functions of calculating departure or arrival delays are defined in cases two and three, respectively. Hence, the defined function can be used directly to calculate average departure delays in C in case four. The "get_depdelay" function is defined to keep original data, in which the origin equals the destinations of A (which are $B_{1\text{-}n}$). Then calculate the average departure delay in each B_i ; the "sapply()" method is used to add the column. Finally, testing two new columns were added to get the correlation. The correlation coefficients of cases 2-4 are 0.4443668, 0.4681582, and 0.5028986, respectively, implying moderate influence.

In python, adopting the same procedure for cases two, three, and four, calculating delays in B (with the same "if" method in case one) and delays in C with empty lists and "for loops" then testing the correlation. For example, an empty list is created to calculate departure delays from B_1 to $C_{1\text{-}n}$. Then original data with the origin equal to B_1 are kept to calculate the average departure delay of B_1. The calculated value is put into the list afterward. It can calculate every average departure delay of $B_{1\text{-}n}$ with the "for loop". After having two columns of delays, test the correlation of those two to find out whether one column will follow against the other columns. Notice that in python, we abide by the definition of cascading failure, which is one delay affects others. Thus, the data are ignored when testing the

correlation if there is no delay at the first airport. As a result, the slightly small correlations of 0.44436684, 0.4523051761, and 0.479277388 in cases two to four are due to some eliminated data while still implying moderate influence.

Question five – Use the available variables to construct a model that predicts delay.

$$Y = \beta 0(constant) + \beta 1X1 + \beta 2X2 + \beta 3X3 + \cdots\cdots\cdots\cdots + \beta nXn$$

It uses machine learning to build multiple regression models for predicting departure and arrival delays. The four steps of constructing a model involve deciding which machine learning algorithm should use, preparing data, training and testing the model, and measuring the model's accuracy.

Decide machine learning algorithm

Predicting a delay in minutes needs an algorithm with a continuous dependent variable. Meanwhile, since the data have many columns regarding the delay and I want to use as many predictors as possible to increase precision, the algorithms should have many independent variables. By constraints, multiple regression is the most suitable model.

Prepare data

Since the original two-year data have categorical predictor variables that do not fit the model and some irrelevant columns, data preparation is needed first. There are different ways of transferring categorical variables to numeric. Numeric level variables are created for “Origin”, “Dest”, and “UniqueCarrier”. In R, the numeric columns are prepared by establishing an object containing every unique element in the column and transferring the categorical column using the “factor ()” method with the established object as the level. In python, the string of categorical columns is changed to categories, arranged in order using the “cat.reorder_categories()” method with the first argument of a list containing every unique element in the column. Last, the numeric columns are done by transferring the categorical string to ranking numbers.

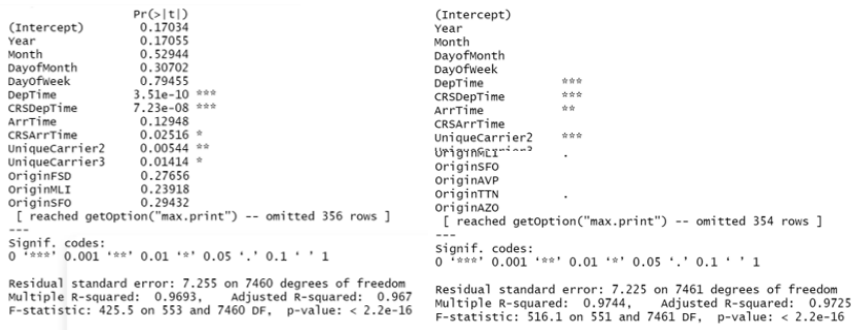
Some less critical columns such as “TailNum”, “Diverted”, “Cancelled”, ”CancellationCode”, “TaxiIn”, “TaxiOutand” and all rows containing “NA” are removed to prevent them from influencing the predictions. Finally, data are divided into two sets of “df_D” and “df_A” by dropping “ArrDelay” or “DepDelay” respectively, for predicting departure and arrival delays.

Train and test the model

The same strategy is implemented to train and test departure and arrival models. Hence, here will only explain the procedure of training and testing the departure model. In the beginning, 10000 data are randomly sampled because of the constraint of computer capacity.

In R, firstly, make the model run 123 times with “set.seed()” method and create “split” object which splits the dependent variable by preserving relative ratios of different labels and sets the splitting ratio equals to 0.8. Second, frame training set at 80% of data using the “subset()” method, which takes all rows presenting “TRUE” in “split” logical vector, and frame testing set with the residual rows. Third, fit the model with the training set and summary the model. The summaries show how the models perform, with departure summary on the left and arrival summary on the right. For example, the summary of the departure model shows departure time dominates departure delays (with three stars), carrier numbers

2 and 11 accounts for quite a proportion of delays (with two stars), etc. The r squared represents how much the model accounts for the total variance of the data. The adjusted r-squared eliminates the effects of incremental predictors since the more predictors we have, the higher the r squared will be. It denotes high reliability with the 0.967 and 0.9725 adjusted r-squared of the departure and arrival model. Before predicting the testing set, it is necessary to remove some data with levels not existing in the training set since the sampled data is not big enough to let each airport present in the training and testing set.



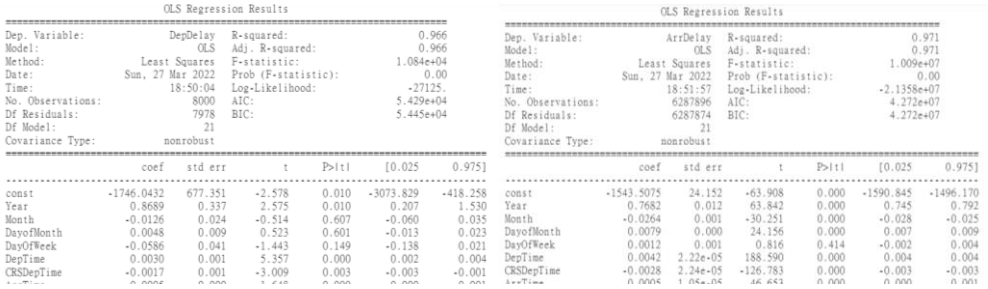
```
                Pr(>|t|)
(Intercept)      0.17034
Year             0.17055
Month            0.52944
DayofMonth       0.30702
DayOfWeek        0.79455
DepTime       3.51e-10 ***
CRSDepTime    7.23e-08 ***
ArrTime          0.12948
CRSArrTime       0.02516 *
UniqueCarrier2   0.00544 **
UniqueCarrier3   0.01414 *
OriginFSD        0.27656
OriginMLI        0.23918
OriginSFO        0.29432
 [ reached getOption("max.print") -- omitted 356 rows ]
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.255 on 7460 degrees of freedom
Multiple R-squared:  0.9693,    Adjusted R-squared:  0.967
F-statistic: 425.5 on 553 and 7460 DF,  p-value: < 2.2e-16
```

```
(Intercept)
Year
Month
DayofMonth
DayOfWeek
DepTime          ***
CRSDepTime       ***
ArrTime          **
CRSArrTime
UniqueCarrier2   ***
UniqueCarrier3   .
OriginSFO
OriginAVP
OriginTTN        .
OriginAZO
 [ reached getOption("max.print") -- omitted 354 rows ]
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.225 on 7461 degrees of freedom
Multiple R-squared:  0.9744,    Adjusted R-squared:  0.9725
F-statistic: 516.1 on 551 and 7461 DF,  p-value: < 2.2e-16
```

In python, the target variable is classified and put into “y” object. Four sets are split with “train_test_split()” method importing from the “sklearn.model_selection”. Two sets that start with "y" only have the target variable. Other “x_***” sets include predictor variables but exclude the target variable. The training sets are named as “*_train”, on the other hand, testing sets are named “*_test”. Next, an object of LinearRegression class is created to train data with the “LR.fit()” method. Testing set without target variable is fitted with the model to make predictions. The constant array of the training set, which presents the constant in multiple regression, is created with the “add_constant()” method. Then “fit()” method is called on the OLS object for fitting the regression model to the data. An extensive description of the regression performance is obtained with the “summary()” method. The high adjusted r-squared of 0.966 for the departure model and 0.971 for the arrival model means the two models can explain the data variances quite well. The smaller the p-value, the more significant and powerful the predictor is. For example, departure time has a p-value close to zero, indicating a highly significant predictor. As we can see, most predictors in the arrival model have a strong impact on the target variable.



OLS Regression Results

Dep. Variable:	DepDelay	R-squared:	0.966
Model:	OLS	Adj. R-squared:	0.966
Method:	Least Squares	F-statistic:	1.084e+04
Date:	Sun, 27 Mar 2022	Prob (F-statistic):	0.00
Time:	18:50:04	Log-Likelihood:	-27125.
No. Observations:	8000	AIC:	5.429e+04
Df Residuals:	7978	BIC:	5.445e+04
Df Model:	21		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1746.0432	677.351	-2.578	0.010	-3073.829	-418.258
Year	0.8689	0.337	2.575	0.010	0.207	1.530
Month	-0.0126	0.024	-0.514	0.607	-0.060	0.035
DayofMonth	0.0048	0.009	0.523	0.601	-0.013	0.023
DayOfWeek	-0.0586	0.041	-1.443	0.149	-0.138	0.021
DepTime	0.0030	0.001	5.357	0.000	0.002	0.004
CRSDepTime	-0.0017	0.001	-3.009	0.003	-0.003	-0.001

OLS Regression Results

Dep. Variable:	ArrDelay	R-squared:	0.971
Model:	OLS	Adj. R-squared:	0.971
Method:	Least Squares	F-statistic:	1.009e+07
Date:	Sun, 27 Mar 2022	Prob (F-statistic):	0.00
Time:	18:51:57	Log-Likelihood:	-2.1358e+07
No. Observations:	6287896	AIC:	4.272e+07
Df Residuals:	6287874	BIC:	4.272e+07
Df Model:	21		
Covariance Type:	nonrobust		

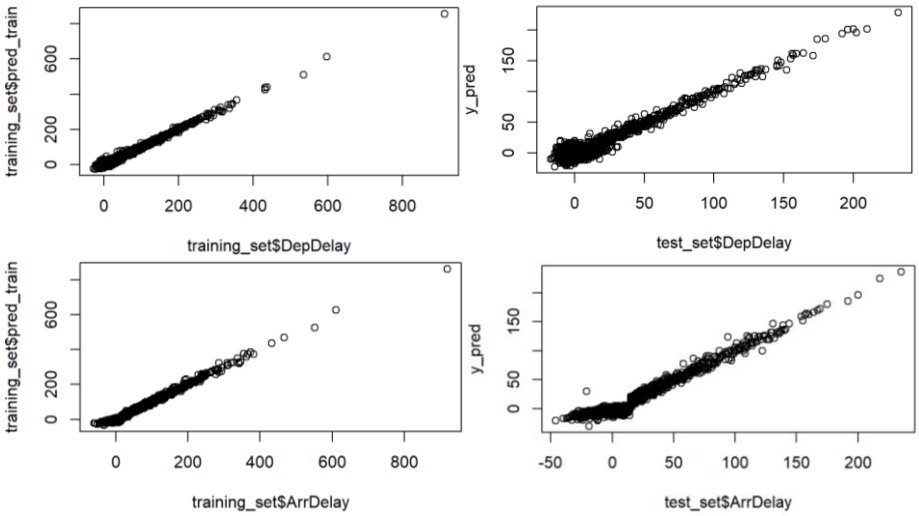
	coef	std err	t	P>|t|	[0.025	0.975]
const	-1543.5075	24.152	-63.908	0.000	-1590.845	-1496.170
Year	0.7682	0.012	63.842	0.000	0.745	0.792
Month	-0.0264	0.001	-30.251	0.000	-0.028	-0.025
DayofMonth	0.0079	0.000	24.156	0.000	0.007	0.009
DayOfWeek	0.0012	0.001	0.816	0.414	-0.002	0.004
DepTime	0.0042	2.22e-05	188.590	0.000	0.004	0.004
CRSDepTime	-0.0028	2.24e-05	-126.783	0.000	-0.003	-0.003

Measure accuracy

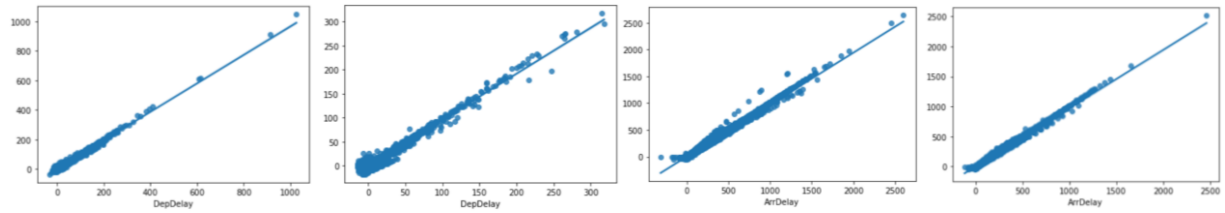
By graphs

There are two ways to test the model's accuracy: plot the relationship between predictions and actual values and calculate the root mean square error. A reliable and valid model will have a straight line graph and low RMSE.

In R, the predicted and actual departure delays of the training set are plotted on the left with a new column, “pred_train” which is generated by fitting training data with the model with “predict()” method. The testing set's predicted, and actual departure delays are plotted on the right. The obvious straight line of the relationship proves that the models predict well.



In python, the predictions of training data are generated with the “LR.predict()” method and are plotted with the actual dependent values in the training set. The predicted and actual delays of the testing set are plotted in the first and third order. Similar to R, the apparent straight line of the relationship shows the models predict well.



By RMSE

In R, the accuracy scores are generated for each model's training and testing set. A new column is added to the training set with the “mutate()” method to calculate the distances of predictions and actual values. RMSE is calculated by calculating the mean of squared error and taking the square root. The RMSEs are 6.9996, 7.2952, 6.9719, and 7.4659 for training and testing sets of departure delays and arrival delays, respectively. Compared to the testing set, the RMSEs of the training set are slightly smaller, which is expected as the model is trained according to the training set.

In python, the RMSEs of training and testing sets of departure and arrival models are calculated by taking the square root of mean square error, which is generated with the “metrics.mean_squared_error()” method. The RMSEs are 7.1832, 7.0744, 7.2259, and 7.2162 for training and testing set of departure delays and training and testing set of arrival delays, respectively. In this case, RMSEs of the testing set are more miniature than the training set, which might result from a smaller testing set (impossible since we have the same test size as in R) or an excellent fitting testing set.