UNIVERSITY OF LONDON

Title: Programming for Data Science Coursework – Python and R

Candidate Number: 200584494

Course: ST2195 Programming for Data Science

Table of Contents

Q1. When is the best time of day, day of the week, and time of year to fly to minimize delays?[1]

Every question that involves working with data in Python and R starts with importing the main libraries to work with it: pandas, numpy, and matplotlib; dplyr and tidyverse. These are used to read data, use functions to manipulate data, and deploy different functions to analyze and visualize it. After datasets for flights, planes, carriers, and airports are imported, the data wrangling process starts with merging two datasets (for 1993 and 1994, since they are selected for the analysis). The count of missing values in all the fields in a merged dataset is represented in Fig.1.

```
Out[14]: Year                    0
         Month                   0
         DayofMonth              0
         DayOfWeek               0
         DepTime            126585
         CRSDepTime              0
         ArrTime            149024
         CRSArrTime              0
         UniqueCarrier           0
         FlightNum               0
         TailNum          10250549
         ActualElapsedTime  149024
         CRSElapsedTime          0
         AirTime          10250549
         ArrDelay           149024
         DepDelay           126585
         Origin                  0
         Dest                    0
         Distance            29823
         TaxiIn           10250549
         TaxiOut          10250549
         Cancelled               0
         CancellationCode 10250549
         Diverted                0
         CarrierDelay     10250549
         WeatherDelay     10250549
         NASDelay         10250549
         SecurityDelay    10250549
         LateAircraftDelay 10250549
         dtype: int64
```

Figure 1

After exploring the shape for each data frame for two years, it is seen that the number of observations differ: 5070501 and 5180048. To eliminate the concern that the outcomes will be biased towards the year with largest number of observations, I sample the largest and make them equal. Raw data contains 29 fields with 10,250,549 observations in total. There are several fields that have the maximum number of observations missing (TailNum, AirTime, TaxiIn, TaxiOut, CancellationCode, CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, and LateAircraftDelay). To cleanse the data, fields with all missing values are dropped along with the duplicated rows. Afterward, we are left with 10,250,549 observations and 19 fields – turns out there were no duplicated rows.

To find out the best time of day to fly to minimize delays CRSDepTime and ArrDelay fields are deployed (created a new data frame) to return the time of the day with a minimum value of delay time in minutes. Since the dataset is large and there are many observations for a given time of the day, the mean value of arrival delay should be taken to then capture the desired time to fly (Fig.2). From the returned table it is seen that the scheduled departure time is not in the easy human-readable format. Henceforth I manipulate that field (add zeros in from of the value to make it 4—digits and separate it by ":" delimiter by two digits) and add another

[1] All the code for this question is provided in "200584494Q1" Jupyter notebook and "200584494q1" RMarkdwon files. For every following question there is a separate Jupyter and R files. The screenshots are provided for Python and/or R code interchangeable for avoiding the page count restriction.

column, CRSDepTime_format with data in “hours:minutes:seconds” format. For doing so, the datetime library is imported additionally (Fig.3).

	CRSDepTime	ArrDelay
0	1	-1.860781
1	2	0.893491
2	3	3.000000
3	4	-1.356164
4	5	1.055058
...	...	...
1244	2356	-5.509677
1245	2357	4.595455
1246	2358	12.454545
1247	2359	2.010998
1248	2400	5.500000

Figure 2

	CRSDepTime	ArrDelay	CRSDepTime_format
0	1	-1.860781	00:01:00
1	2	0.893491	00:02:00
2	3	3.000000	00:03:00
3	4	-1.356164	00:04:00
4	5	1.055058	00:05:00
...	...	...	...
1244	2356	-5.509677	23:56:00
1245	2357	4.595455	23:57:00
1246	2358	12.454545	23:58:00
1247	2359	2.010998	23:59:00
1248	2400	5.500000	00:00:00

1249 rows × 3 columns

Figure 3

To get the result, a minimum mean value for arrival delay should be inquired and the corresponding to it time of day captured. Python yields the result, which is -28.0 minutes which is the minimum mean arrival delay. The negative value can be interpreted that at minimum the flights on average depart early by 28 minutes. This can be worse for flight attendants, but it is the minimum, as per the question requirement. The corresponding time of day for this value is 23:59 (11:54 PM) (Fig.4).

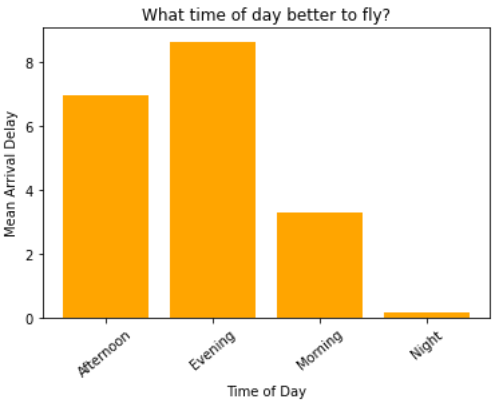
Out[32]:

	CRSDepTime	ArrDelay	CRSDepTime1
1243	2354	-28.0	23:54:00

Figure 4

Alternatively, I have assigned labels to each period of time to be: night, morning, afternoon, and evening. Other variants are assigned a label incorrectly since there would be an incorrect time format. Returning arrival delay means for these observations yield the result that it is better to fly at night, from 00:00 to 06:00 (mean arrival delay 0.13 min) (Fig.5).

	CRSDepTime_category	ArrDelay
0	Afternoon	6.956794
1	Evening	8.630958
2	Morning	3.280723
3	Night	0.134675



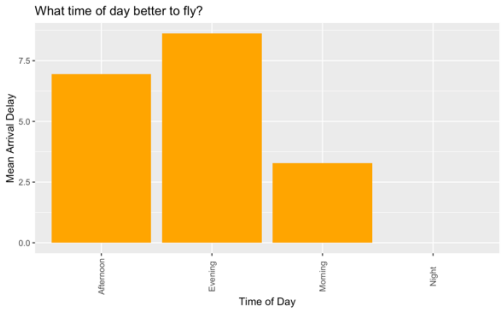
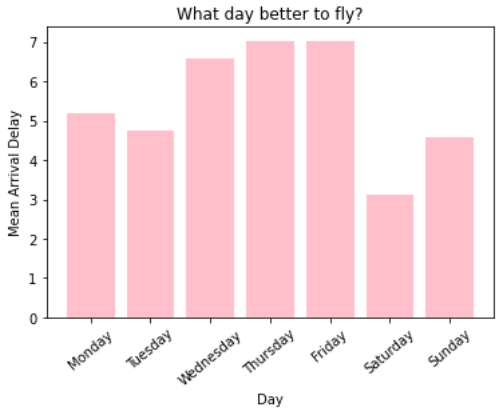


Figure 5

The next part of the question is to find the best day of the week to fly to minimize delays. By the same token, mean arrival delays are returned, but now the level of observation is based on the DayofWeek. Fig.6 provides the table with means of arrival delays for each of the seven days of the week and the returned minimum value off them, which is 3.12 minutes corresponding to the 5th day of the week – Saturday, along with visualization.

	DayOfWeek	ArrDelay
0	Monday	5.172965
1	Tuesday	4.745828
2	Wednesday	6.586357
3	Thursday	7.019461
4	Friday	7.022999
5	Saturday	3.122929
6	Sunday	4.593742

	DayOfWeek	ArrDelay
5	Saturday	3.122929



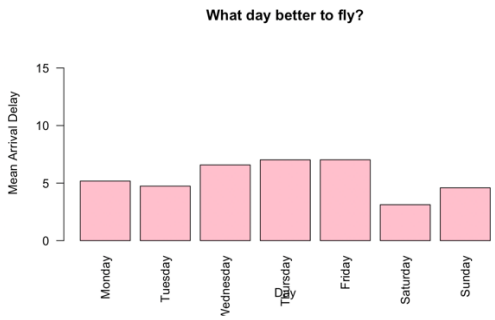
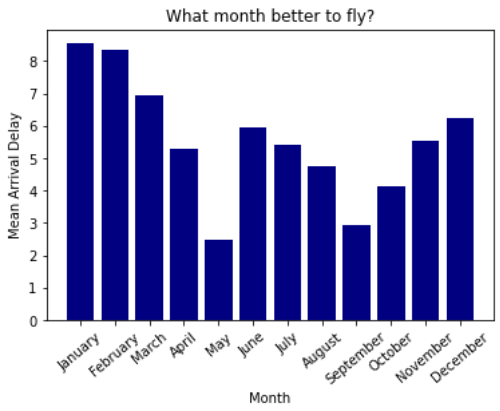


Figure 6

Lastly, to determine the optimal time of year to fly, the level of observation should be identified from the available data. Getting narrowed down from the year, there are months to look at. Therefore, fields Month and ArrDelay. The output table is provided in Fig.7, where it is seen that among 12 months through 1993 and 1994, the minimum mean arrival delay occurs on the 5th month, May, accounting for 2.48 minutes of the airplane arriving late by.

	Month	ArrDelay
0	January	8.574349
1	February	8.325955
2	March	6.957133
3	April	5.293156
4	May	2.480028
5	June	5.952352
6	July	5.396627
7	August	4.763323
8	September	2.936417
9	October	4.123382
10	November	5.556316
11	December	6.246177

	Month	ArrDelay
4	May	2.480028



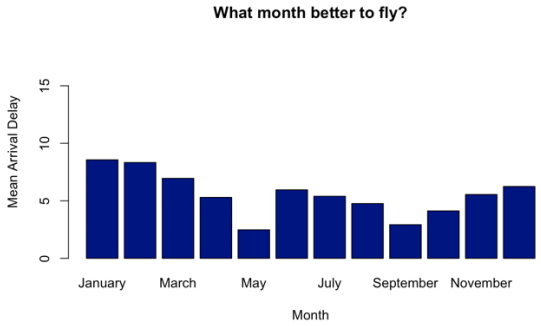


Figure 7

Q2. Do older planes suffer more delays?[2]

From the same manipulations with the data done in question 1 (reading the csv files as well as dropping null fields and finding duplicates), it can be concluded that there is limited data to straightforwardly answer this question. More specifically, not a single field in all four provided datasets contains information on whether the aircraft is old or new.

I would deploy the following approach to answer this question: first, to determine the criteria for calling the plane "old" or "new", I would create two new data frames from the flights and planes data by retrieving "TailNum" from both and "ArrDelay" from the first one and "Year" from the second to subsequently merge them based on the "TailNum". Then I would take the average of the ArrDelay for each plane (differentiated by its tail number) and add a new field "PlaneAge" which is the difference between the years studied in the current coursework (1993 and 1994) and the year in which the plane was made. Finally, I would make python return the

[2] Refer to the "200584494Q2" Jupyter and "200584494q2" RMarkdown file.

correlation plot between the PlaneAge and Arrdelay. Then, based on the outputted graph I could have made a conclusion on the question.

Unfortunately, this analysis cannot be done for 1993 and 1994 datasets, since on the very first step of cleansing null field observations and thus columns, TailNum is eliminated because all the observations were null for it for both years. This can probably be attributed to the lack of accountability inclusion in the years 1990-1995.

Q3. How does the number of people flying between different locations change over time?[3]

Since there are no fields that account for the number of people flying between different locations, I instead deploy the number of instances the flight between different locations that have happened. This can proxy for the number of people flying between different locations if there would be provided information on the plane capacity of the number of people registered at the airport for the specific flight. For answering the question using my method, after cleansing the data as in Q1 and Q2, I retrieve variables Origin, Dest, Year, and Month. I then need to analyze how the number (count distinct of occurrences between origin and destination) of flights have changed through every month for each 1993 and 1994 (Fig.8). After that, to generalize the data further I return the sum of all instances of flights between pairwise locations and make a pivot table that showcases the sum for each month of years of analysis (Fig.9).

	Origin	Dest	Year	Month	count
0	ABE	ATL	1993	1	61
1	ABE	ATL	1993	2	53
2	ABE	ATL	1993	3	60
3	ABE	ATL	1993	4	57
4	ABE	ATL	1993	5	60
...	...	...	...	...	...
73086	YAK	JNU	1994	8	29
73087	YAK	JNU	1994	9	30
73088	YAK	JNU	1994	10	29
73089	YAK	JNU	1994	11	29
73090	YAK	JNU	1994	12	28

73091 rows × 5 columns

Figure 8

		count
Month	**Year**	
1	**1993**	422755
	1994	414953
2	**1993**	383950
	1994	378781
3	**1993**	432980
	1994	426263
4	**1993**	420051
	1994	412127
5	**1993**	430185
	1994	427209
6	**1993**	424599
	1994	417417
7	**1993**	436269
	1994	434331
8	**1993**	441745
	1994	441481
9	**1993**	416357
	1994	421718
10	**1993**	428476
	1994	439916
11	**1993**	407900
	1994	419251
12	**1993**	425234
	1994	437054

Month <int>	Year <int>	sum_count <int>
1	1993	422755
1	1994	414942
2	1993	383950
2	1994	378663
3	1993	432980
3	1994	426431
4	1993	420051
4	1994	412173
5	1993	430185
5	1994	427191

Figure 9

Alternatively, I can see, side by side, how the distinct instances of flights have changed between pairs of locations between 1993 and 1994 years. For that I make python again read both data frames, group observations by year, make count, and merge (Fig. 10).

[3] Refer to the "200584494Q3" Jupyter and "200584494q3" RMarkdown file.

Origin	Dest	1993	1994
LAX	SFO	17735	17428.0
SFO	LAX	17605	17526.0
LAX	LAS	13701	13056.0
	PHX	13410	13303.0
LAS	LAX	13286	12644.0
...	...	...	...
HSV	MCI	1	NaN
SYR	BUF	1	NaN
HSV	BTR	1	NaN
PHL	BWI	1	NaN
CMH	ALB	1	NaN

3425 rows × 2 columns

Origin <chr>	Dest <chr>	1993 <int>	1994 <int>
ABE	ATL	706	709
ABE	BWI	113	484
ABE	CLT	336	468
ABE	DCA	*NA*	224
ABE	DTW	941	948
ABE	LGA	144	58
ABE	MDT	1626	1494
ABE	ORD	1180	1278
ABE	PIT	1466	1433
ABE	RDU	3	*NA*

Figure 10

The number of flight instances has generally decreased from 1993 to 1994 looking at Figures 9 and 10, where both on the level of comparing Months and distinct flight routes the values fall from 1993 to 1994.

Q4. Can you detect cascading failures as delays in one airport create delays in other?[4]

To detect cascading failures, I will need to differentiate the plane and thus deploy the "TailNum" variable. Since for the chosen analysis years – 1993 and 1994 both this field is dropped by the virtue of not having observations for it, I will choose another two years of observation – 2003 and 2004. After the similar manipulations with data are performed as in the above three parts, 'TailNum', ' Origin', 'Dest', 'DepDelay', 'ArrDelay', 'DepTime' fields are selected for the analysis. DepDelay and ArrDelay are subsequently reformatted in to dummy variables, where 0 indicates that there was no delay (plane arrived and departed on time or early), and 1 indicates delay (otherwise). Exploring the number of flight instances for each plane and focusing on the specific plane I selected "N226SW" plane tail number (Fig.11).

	TailNum	Origin	Dest	DepDelay	ArrDelay	DepTime
2871418	N226SW	SFO	CIC	1.0	1.0	3.0
1791771	N226SW	IYK	LAX	0.0	0.0	449.0
1256807	N226SW	IYK	LAX	0.0	0.0	452.0
5549768	N226SW	PSC	SEA	0.0	0.0	454.0
1249870	N226SW	PSP	LAX	0.0	1.0	458.0
...	...	...	...	...	...	...
5064348	N226SW	SFO	ACV	1.0	1.0	2306.0
4959471	N226SW	LAX	MRY	1.0	1.0	2319.0
6736788	N226SW	SFO	RDD	1.0	1.0	2320.0
5612405	N226SW	SFO	ACV	1.0	1.0	2342.0
798350	N226SW	ACV	SMF	1.0	0.0	2400.0

5860 rows × 6 columns

Figure 11

[4] Refer to the "200584494Q4" Jupyter and "200584494q4" RMarkdown file.

The occurrences of departure and arrival delays are then calculated based on the dummy variables formatted before. The following rule is applied: the cascading delay count to have happened if the number of departure and arrival delays is equal to three. After such instances are summed, Cascading Failures field is added accounting for 499 such instances for N226SW plane (Fig.12).

	TailNum	Cascading Failures
0	N226SW	474

Figure 12

Since there are 6126 flight instances for the chosen plane, to detect a rate of cascading delay as the percentage of the total flights, the simple formula was applied: (474/5860)*100 (Fig.13).

```
Plane with TailNum N226SW has a cascading delay rate of: 8.088737201365188 % of total flights
```

Figure 13

	TailNum <chr>	Origin <chr>	Dest <chr>	DepDelay <dbl>	ArrDelay <dbl>	DepTime <int>
2871404	N226SW	SFO	CIC	1	1	3
1791757	N226SW	IYK	LAX	0	0	449
1256794	N226SW	IYK	LAX	0	0	452
9681288	N226SW	PSC	SEA	0	0	454
1249857	N226SW	PSP	LAX	0	1	458
6967441	N226SW	PSC	SEA	0	1	500
9618209	N226SW	IYK	LAX	0	0	500
193982	N226SW	IPL	LAX	0	1	502
2317628	N226SW	PSC	SEA	0	0	502
9538054	N226SW	PSC	SEA	1	1	503

1–10 of 5,837 rows Previous 1 2 3 4 5 6 … 100 Next

TailNum <chr>	Cascading Failures <int>
N226SW	459

Figure 14

It is important to note that R, on the other hand, outputs the total rows for “N226SW” plane to be of total 5,837 and the count of cascading failures for it to be 459 so that the output in terms of the rate of cascading delays is 7.86% since it skips some data (Fig.14).

Q5. Use the available variables to construct the model that predicts delays.[5]

The prediction model that I am going to use in this part for the python code is Gradient Boosting Classifier and Decision Tree Classifier. For the subsequent analysis, I only want to include the flights that are not canceled and diverted, with further criteria towards the data – to only include numerical data with fields that don’t contain any null values. As above, the sample of 50000323 from the 1994 data file is sampled to eliminate the skewness of data and thus bias. For simplicity further, I manipulate the fields to: CRSDepTime and CRSArrTime to be divided by 100 and insert a new field TotDleay to account for the overall delay. Besides, I apply a threshold of 30 minutes to account for the additional Delay column to represent 1 if the total delay happened to be less than 30 minutes, 2 if more than 30 minutes and 0 otherwise. When the Gradient Boosting Classifier models’ results are obtained after the test and train split to be respectively 80 and 20 percent of the data, results are undesirable (Fig. 15) – the prediction, along with others is low.

[5] Refer to the “200584494Q5” Jupyter and “200584494q5” RMarkdown file.

```
Prediction: 0.00452
Accuracy: 0.00452
AUC score: 0.50000
Precision: 0.00436
Recall: 0.00452
```

Figure 15

This is attributed to the eliminated columns of the data cleansing process, where several columns have accounted for null values. Therefore, I select other years of observation to approach the current question – 2006 and 2007, the most recent ones available on the website (2008 has missing values too). For these years, the cleansing process is similar to the above, while for the elimination of skewness, 7003802 observations are sampled from the 2007 data file. Figure 15 shows the data for the chosen years is consistent with the requirement to be full.

```
Month               0
DayOfWeek           0
DayofMonth          0
Origin              0
Dest                0
CRSDepTime          0
DepDelay            0
CRSArrTime          0
ArrDelay            0
CarrierDelay        0
WeatherDelay        0
NASDelay            0
SecurityDelay       0
LateAircraftDelay   0
Delay               0
dtype: int64
```

Figure 16

The merged data is sampled to contain 250000 observations due to the computational cost. After the cleansing and data manipulation steps performed in 2006 and 2007 merged data frame as above, the Gradient Boosting Classifier yields the following result: (Fig. 17).

```
Prediction: 0.92464
Accuracy: 0.92464
AUC score: 0.75306
Precision: 0.92570
Recall: 0.92464
```

Figure 17

The results are not perfect but are still good. The other model of the current question to be tried is the Decision Tree Classifier. With the same merged data file, I assign a model, which yields the following results: (Fig. 18), which yields much better results.

```
Prediction: 0.99914
Accuracy: 0.99914
AUC score: 0.98164
Precision: 1.00000
Recall: 0.99914
```

Figure 18

Thus, as an outcome of the current analysis, I would choose the Decision Tree Classifier model that predicts delays with a prediction and accuracy score of 0.99914 and an AUC score of 0.98164.

As for the R code, the Decision Tree Classifier model deploys the same partitioning for data (80%/20%). For computational reasons, the data is sampled to contain 25000 observations

as in python, the splits are validated using the dimension function. After the “rpart.plot” library is imported, the decision tree looks the following way: Figure 19.

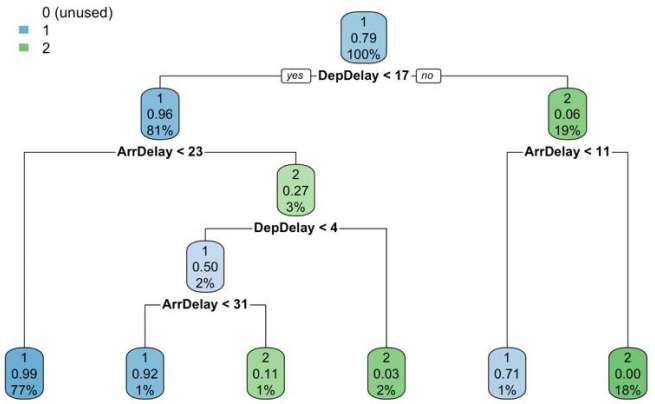


Figure 19

The current visualization for the decision tree displays the probability of the total Delay (again, 1 if the total delay happened to be less than 30 minutes, 2 if more). At the very top node, it is evident that the total number of flights in cleaned and partitioned data that have had a < 30-minute delay is 79%. Passing further, it is seen that there are 81% of flights where the departure delay happened to be less than 17 minutes, with a probability of Delay to have a total delay to be more than 30 minutes 96%.

Finally, the decision tree classifier prediction model is constructed from the test set to estimate predictions in terms of classes – 1 and 2. Figure 20 shows that 39100 flights are identified correctly to have a total delay of fewer than 30 minutes, with 113 incorrectly. The numbers are 10076 and 504 for the flights with a total delay of more than 30 minutes respectively.

```
   predict_unseen
        0     1     2
0       0   141    66
1       0 39100   113
2       0   504 10076
```

Figure 20

The evaluation of the model performance reveals a score of 98 percent for the test set, and I am satisfied with the result.

```
[1] "Accuracy for test 0.98352"
```

Figure 21