Notebook Code Data (1) Comments (62) Log Versions (62) Forks (4) Fork Notebook

kaggle Search kaggle Competitions Datasets Kernels Discussion Learn ••• Sign In

FabienDaniel

Predicting flight delays [Tutorial]

180 voters

last run 9 months ago · IPython Notebook HTML · 27,972 views
using data from 2015 Flight Delays and Cancellations · Public

Tags data visualization beginner eda tutorial regression analysis

Notebook

Predicting flight delays [*Tutorial*]

Fabien Daniel (September 2017)

In this notebook, I develop a model aimed at predicting flight delays at take-off. The purpose is not to obtain the best possible prediction but rather to emphasize on the various steps needed to build such a model. Along this path, I then put in evidence some **basic but important** concepts. Among then, I comment on the importance of the separation of the dataset during the traning stage and how **cross-validation** helps in determing accurate model parameters. I show how to build **linear** and **polynomial** models for **univariate** or **multivariate regressions** and also, I give some insight on the reason why **regularisation** helps us in developing models that generalize well.

From a ***technical point of view***, the main aspects of python covered throughout the notebook are:

- **visualization**: matplolib, seaborn, basemap
- **data manipulation**: pandas, numpy
- **modeling**: sklearn, scipy
- **class definition**: regression, figures

During the EDA, I intended to create good quality figures from which the information would be easily accessible at a first glance. An important aspect of the data scientist job consists in divulgating its findings to people who do not necessarily have knowledge in the technical aspects data scientists master. Graphics are surely the most powerful tool to achieve that goal, and mastering visualization techniques thus seems important.

Also, as soon as an action is repeated (mostly at identical) a few times, I tend to write classes or functions and eventually embed them in loops. Doing so is sometimes longer than a simple *copy-paste-edit* process but, on the one hand, this improves the readibility of the code and most importantly, this reduces the number of lines of code (and so, the number of opportunities to introduce mistakes !!). In the current notebook, I defined classes in the modeling part in order to perform regressions. I also defined a class to wrap the making of figures. This allows to create stylish figures, by tuning the matplotlib parameters, that can be subsequently re-used thanks to that template. I feel that this could be useful to create nice looking graphics and then use them extensively once you are satisfied with the tuning. Moreover, this helps to keep some homogeneity in your plots.

Acknowledgement: many thanks to J. Abécassis (https://www.kaggle.com/judithabk6) for the advices and help provided during the writing of this notebook

This notebook is composed of three parts: cleaning (section 1), exploration (section 2-5) and modeling (section 6).

Preamble**:** *overview of the dataset*

1. Cleaning

- 1.1 Dates and times
- 1.2 Filling factor

Preamble: overview of the dataset

First, I load all the packages that will be needed during this project:

Code

and then, I read the file that contains the details of all the flights that occured in 2015. I output some informations concerning the types of the variables in the dataframe and the quantity of null values for each variable:

Code

```
Dataframe dimensions: (5819079, 31)
```

Out[2]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER

column type	int64	int64	int64	int64	object	int64	object
null values (nb)	0	0	0	0	0	0	14721
null values (%)	0	0	0	0	0	0	0.252978

Each entry of the `flights.csv` file corresponds to a flight and we see that more than 5'800'000 flights have been recorded in 2015. These flights are described according to 31 variables. A description of these variables can be found here (https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time) and I briefly recall the meaning of the variables that will be used in this notebook:

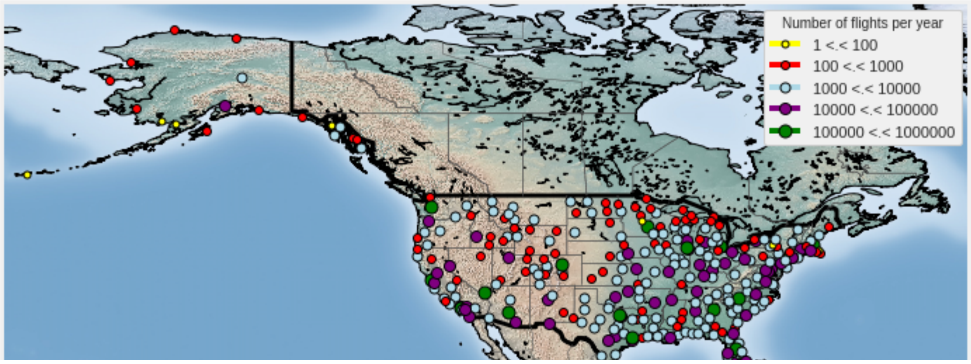
- **YEAR, MONTH, DAY, DAY_OF_WEEK**: dates of the flight
- **AIRLINE**: An identification number assigned by US DOT to identify a unique airline
- **ORIGIN_AIRPORT** and **DESTINATION_AIRPORT**: code attributed by IATA to identify the airports
- **SCHEDULED_DEPARTURE** and **SCHEDULED_ARRIVAL** : scheduled times of take-off and landing
- **DEPARTURE_TIME** and **ARRIVAL_TIME**: real times at which take-off and landing took place
- **DEPARTURE_DELAY** and **ARRIVAL_DELAY**: difference (in minutes) between planned and real times
- **DISTANCE**: distance (in miles)

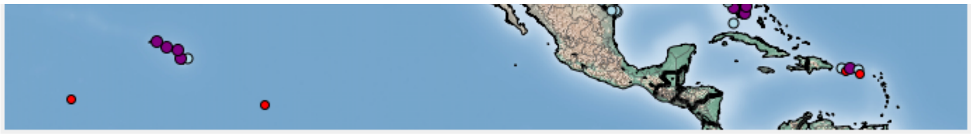
An additional file of this dataset, the `airports.csv` file, gives a more exhaustive description of the airports:

Code

To have a global overview of the geographical area covered in this dataset, we can plot the airports location and indicate the number of flights recorded during year 2015 in each of them:

Code





Given the large size of the dataset, I decide to consider only a subset of the data in order to reduce the computational time. I will just keep the flights from January 2015:

Code

1. Cleaning

1.1 Dates and times

In the initial dataframe, dates are coded according to 4 variables: **YEAR, MONTH, DAY**, and **DAY_OF_WEEK**. In fact, python offers the ***datetime*** format which is really convenient to work with dates and times and I thus convert the dates in this format:

Code

Moreover, in the **SCHEDULED_DEPARTURE** variable, the hour of the take-off is coded as a float where the two first digits indicate the hour and the two last, the minutes. This format is not convenient and I thus convert it. Finally, I merge the take-off hour with the flight date. To proceed with these transformations, I define a few functions:

Code

and I call them to modify the dataframe variables:

Code

```
Out[8]:
```

	SCHEDULED_DEPARTURE	SCHEDULED_ARRIVAL	DEPARTURE_TIME	ARRIVAL_TIME
0	2015-01-01 00:05:00	04:30:00	23:54:00	04:08:00
1	2015-01-01 00:10:00	07:50:00	00:02:00	07:41:00
2	2015-01-01 00:20:00	08:06:00	00:18:00	08:11:00
3	2015-01-01 00:20:00	08:05:00	00:15:00	07:56:00
4	2015-01-01 00:25:00	03:20:00	00:24:00	02:59:00
5	2015-01-01 00:25:00	06:02:00	00:20:00	06:10:00

Note that in practice, the content of the **DEPARTURE_TIME** and **ARRIVAL_TIME** variables can be a bit misleading since they don't contain the dates. For exemple, in the first entry of the dataframe, the scheduled departure is at 0h05 the 1st of January. The **DEPARTURE_TIME** variable indicates 23h54 and we thus don't know if the flight leaved before time or if there was a large delay. Hence, the **DEPARTURE_DELAY** and **ARRIVAL_DELAY** variables proves more useful since they directly provides the delays in minutes. Hence, in what follows, I will not use the **DEPARTURE_TIME** and **ARRIVAL_TIME** variables.

1.2 Filling factor

Finally, I clean the dataframe throwing the variables I won't use and re-organize the columns to ease its reading:

Code

```
Out[9]:
```

	AIRLINE	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	DE
0	AS	ANC	SEA	2015-01-01 00:05:00	23:
1	AA	LAX	PBI	2015-01-01 00:10:00	00:
2	US	SFO	CLT	2015-01-01 00:20:00	00:
3	AA	LAX	MIA	2015-01-01 00:20:00	00:
4	AS	SEA	ANC	2015-01-01 00:25:00	00:

At this stage, I examine how complete the dataset is:

Code

```
Out[10]:
```

	variable	missing values	filling factor (%)
0	ARRIVAL_DELAY	12955	97.243429
1	ELAPSED_TIME	12955	97.243429
2	ARRIVAL_TIME	12271	97.388971
3	DEPARTURE_TIME	11657	97.519618
4	DEPARTURE_DELAY	11657	97.519618
5	AIRLINE	0	100.000000
6	ORIGIN_AIRPORT	0	100.000000
7	DESTINATION_AIRPORT	0	100.000000
8	SCHEDULED_DEPARTURE	0	100.000000
9	SCHEDULED_ARRIVAL	0	100.000000
10	SCHEDULED_TIME	0	100.000000

We see that the variables filling factor is quite good (> 97%). Since the scope of this work is not to establish the state-of-the-art in predicting flight delays, I decide to proceed without trying to impute what's missing and I simply remove the entries that contain missing values.

Code

2. Comparing airlines

As said earlier, the **AIRLINE** variable contains the airline abreviations. Their full names can be retrieved from the `airlines.csv` file.

Code

```
Out[12]:
```

	IATA_CODE	AIRLINE
0	UA	United Air Lines Inc.
1	AA	American Airlines Inc.
2	US	US Airways Inc.
3	F9	Frontier Airlines Inc.
4	B6	JetBlue Airways
5	OO	Skywest Airlines Inc.
6	AS	Alaska Airlines Inc.
7	NK	Spirit Air Lines
8	WN	Southwest Airlines Co.
9	DL	Delta Air Lines Inc.
10	EV	Atlantic Southeast Airlines
11	HA	Hawaiian Airlines Inc.
12	MQ	American Eagle Airlines Inc.
13	VX	Virgin America

For further use, I put the content of this this dataframe in a dictionary:

```
In [13]:
abbr_companies = airlines_names.set_index('IATA_CODE')['AIRLINE'].to_dict()
```

2.1 Basic statistical description of airlines

As a first step, I consider all the flights from all carriers. Here, the aim is to classify the airlines with respect to their punctuality and for that purpose, I compute a few basic statisticial parameters:

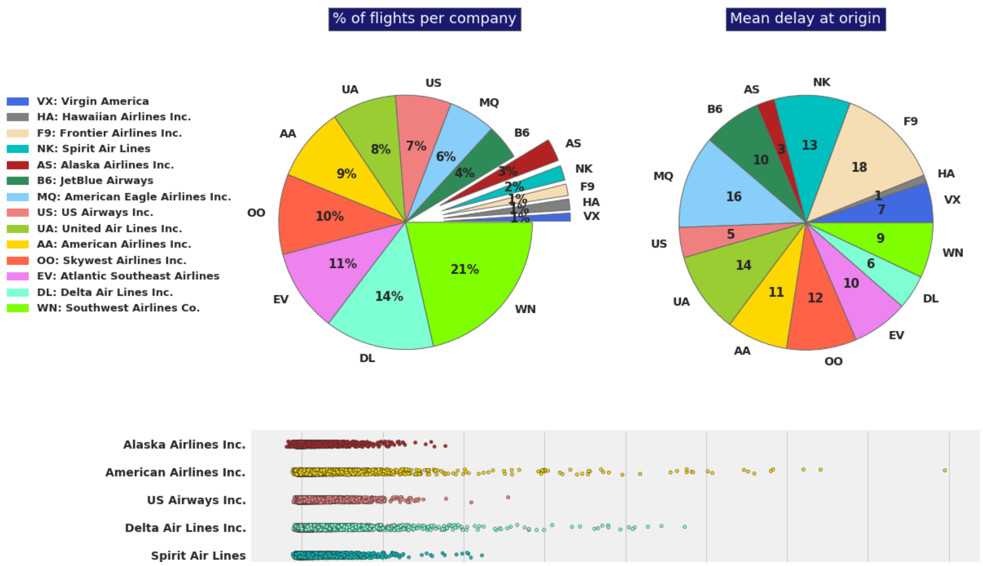
Code

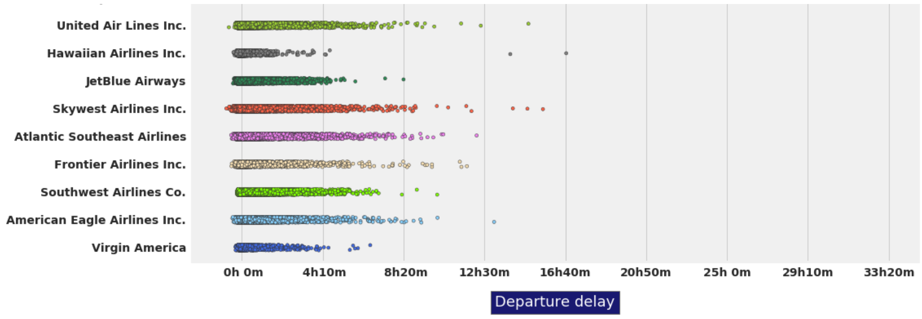
```
Out[14]:
```

	count	max	mean	min
AIRLINE				
VX	4647.0	397.0	6.896277	-20.0
HA	6408.0	1003.0	1.311954	-26.0
F9	6735.0	696.0	17.910765	-32.0
NK	8632.0	557.0	13.073100	-28.0
AS	13151.0	444.0	3.072086	-47.0
B6	20482.0	500.0	9.988331	-27.0
MQ	27568.0	780.0	15.995865	-29.0
US	32478.0	638.0	5.175011	-26.0
UA	37363.0	886.0	13.885555	-40.0
AA	43074.0	1988.0	10.548335	-29.0
OO	46655.0	931.0	11.999957	-48.0
EV	48084.0	726.0	9.678895	-33.0
DL	63676.0	1184.0	5.888215	-26.0
WN	98060.0	604.0	9.453426	-15.0

Now, in order to facilitate the lecture of that information, I construct some graphics:

Code



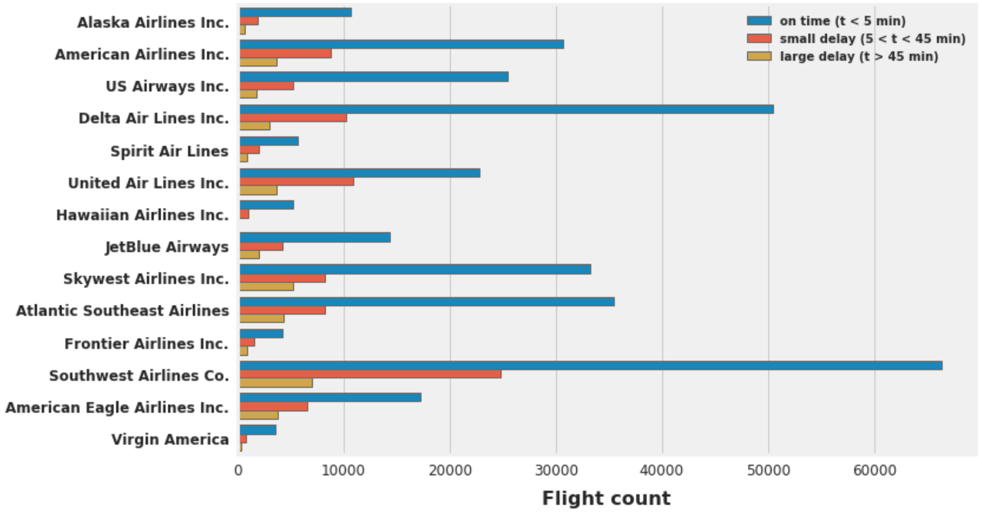


Considering the first pie chart that gives the percentage of flights per airline, we see that there is some disparity between the carriers. For exemple, *Southwest Airlines* accounts for $\sim$20% of the flights which is similar to the number of flights chartered by the 7 tiniest airlines. However, if we have a look at the second pie chart, we see that here, on the contrary, the differences among airlines are less pronounced. Excluding *Hawaiian Airlines* and *Alaska Airlines* that report extremely low mean delays, we obtain that a value of $\sim$**11$\pm$7 minutes** would correctly represent all mean delays. Note that this value is quite low which mean that the standard for every airline is to respect the schedule !

Finally, the figure at the bottom makes a census of all the delays that were measured in January 2015. This representation gives a feeling on the dispersion of data and put in perspective the relative homogeneity that appeared in the second pie chart. Indeed, we see that while all mean delays are around 10 minutes, this low value is a consequence of the fact that a majority of flights take off on time. However, we see that occasionally, we can face really large delays that can reach a few tens of hours !

The large majority of short delays is visible in the next figure:

Code

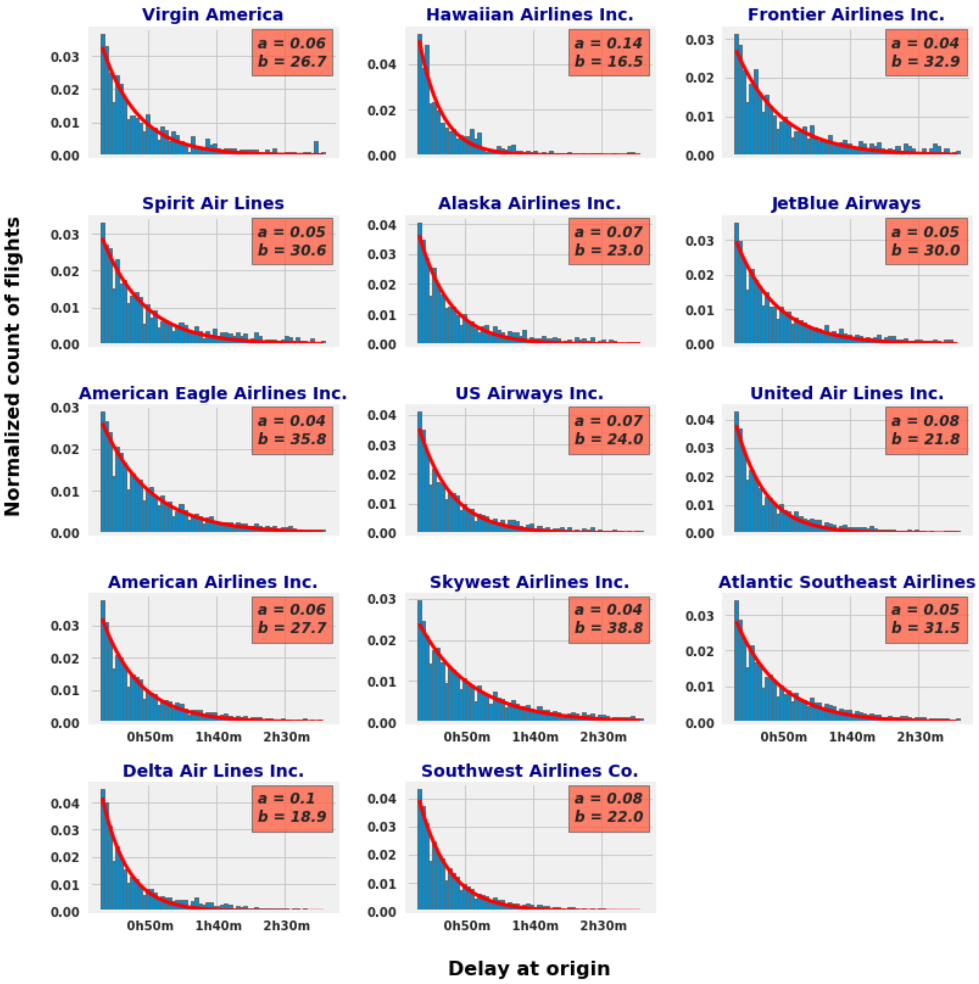


This figure gives a count of the delays of less than 5 minutes, those in the range 5 < t < 45 min and finally, the delays greater than 45 minutes. Hence, we wee that independently of the airline, delays greater than 45 minutes only account for a few percents. However, the proportion of delays in these three groups depends on the airline: as an exemple, in the case of *SkyWest Airlines*, the delays greater than 45 minutes are only lower by $\sim$30% with respect to delays in the range 5 < t < 45 min. Things are better for *SoutWest Airlines* since delays greater than 45 minutes are 4 times less frequent than delays in the range 5 < t < 45 min.

2.2 Delays distribution: establishing the ranking of airlines

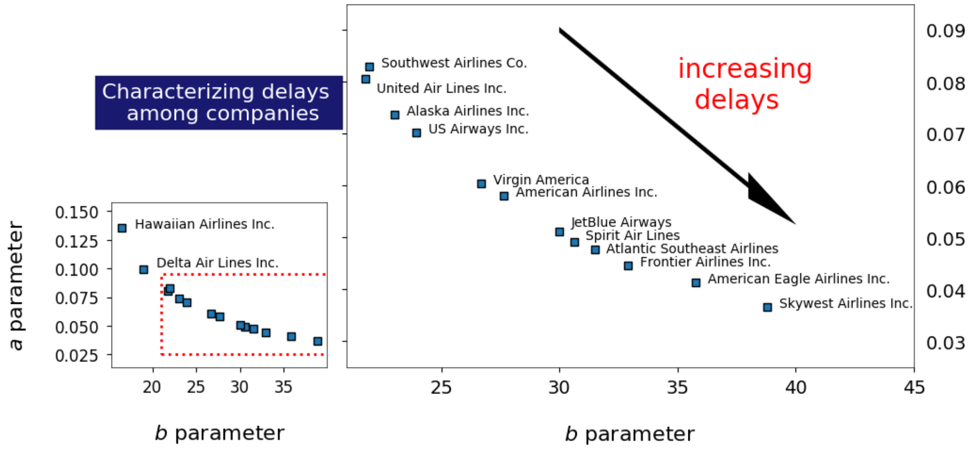
It was shown in the previous section that mean delays behave homogeneously among airlines (apart from two extrem cases) and is around 11$\pm$7 minutes. Then, we saw that this low value is a consequence of the large proportion of flights that take off on time. However, occasionally, large delays can be registred. In this section, I examine more in details the distribution of delays for every airlines:

Code



This figure shows the normalised distribution of delays that I modelised with an exponential distribution $f(x) = a\exp(-x/b)$. The a et b parameters obtained to describe each airline are given in the upper right corner of each panel. Note that the normalisation of the distribution implies that $\int f(x)\,dx \sim 1$. Here, we do not have a strict equality since the normalisation applies the histograms but not to the model function. However, this relation entails that the a et b coefficients will be correlated with $a \propto 1/b$ and hence, only one of these two values is necessary to describe the distributions. Finally, according to the value of either a or b, it is possible to establish a ranking of the companies: the low values of a will correspond to airlines with a large proportion of important delays and, on the contrary, airlines that shine from their punctuality will admit hight a values:

Code

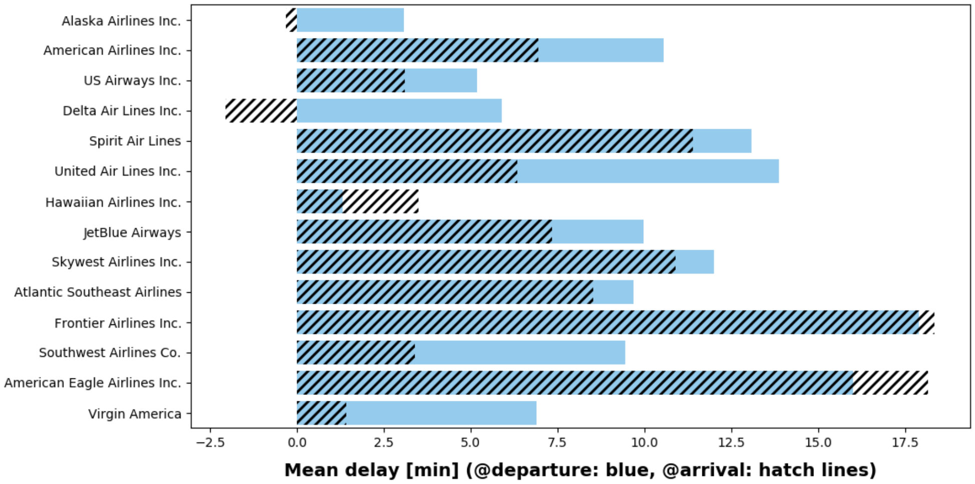


The left panel of this figure gives an overview of the a and b coefficients of the 14 airlines showing that *Hawaiian Airlines* and *Delta Airlines* occupy the first two places. The right panel zooms on 12 other airlines. We can see that *SouthWest Airlines*, which represent $\sim$20% of the total number of flights is well ranked and occupy the third position. According to this ranking, *SkyWest Airlines* is the worst carrier.

3. Delays: take-off or landing ?

In the previous section, all the discussion was done on departure delays. However, these delays differ somewhat from the delays recorded at arrival:

Code



On this figure, we can see that delays at arrival are generally lower than at departure. This indicates that airlines adjust their flight speed in order to reduce the delays at arrival. In what follows, I will just that airlines adjust their flight speed in order to reduce the delays at arrival. In what follows, I will just consider the delays at departure but one has to keep in mind that this can differ from arrival delays.

4. Relation between the origin airport and delays

I will now try to define if there is a correlation between the delays registered and the airport of origin. I recall that in the dataset, the number of airports considered is:

Code

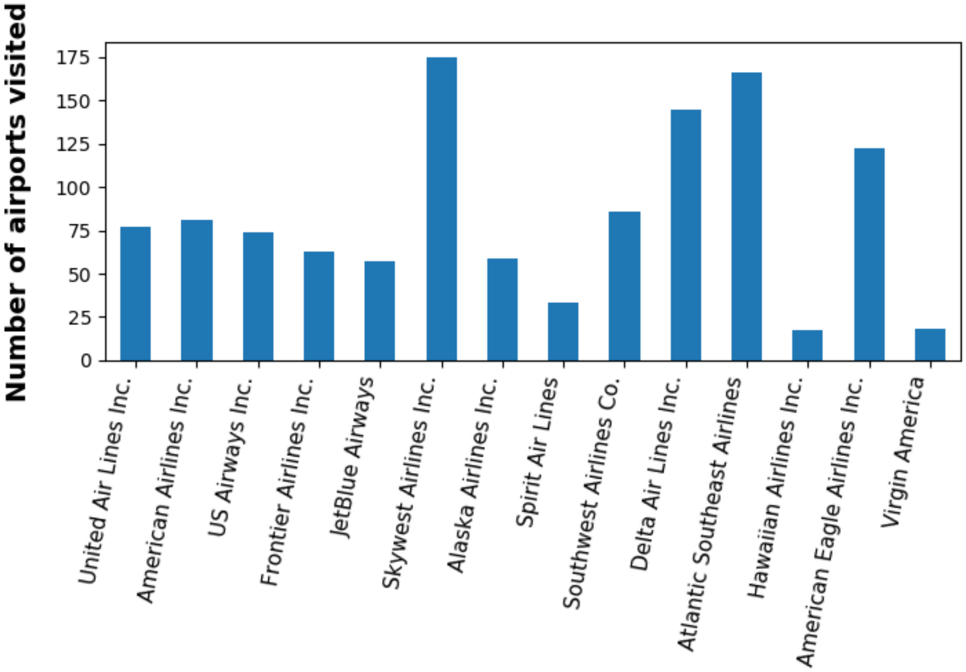
```
Nb of airports: 312
```

4.1 Geographical area covered by airlines

Here, I have a quick look at the number of destination airports for each airline:

Code

Code

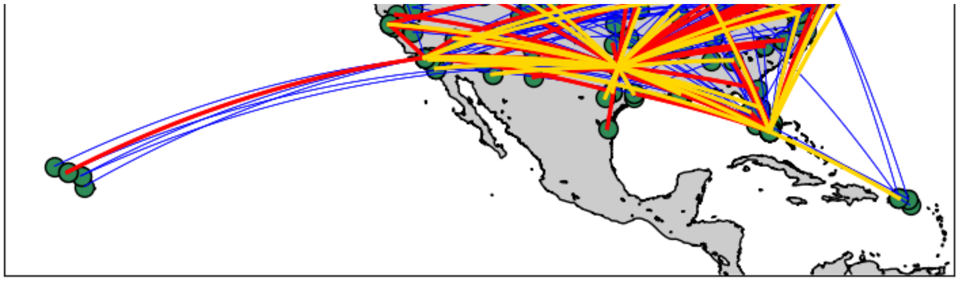


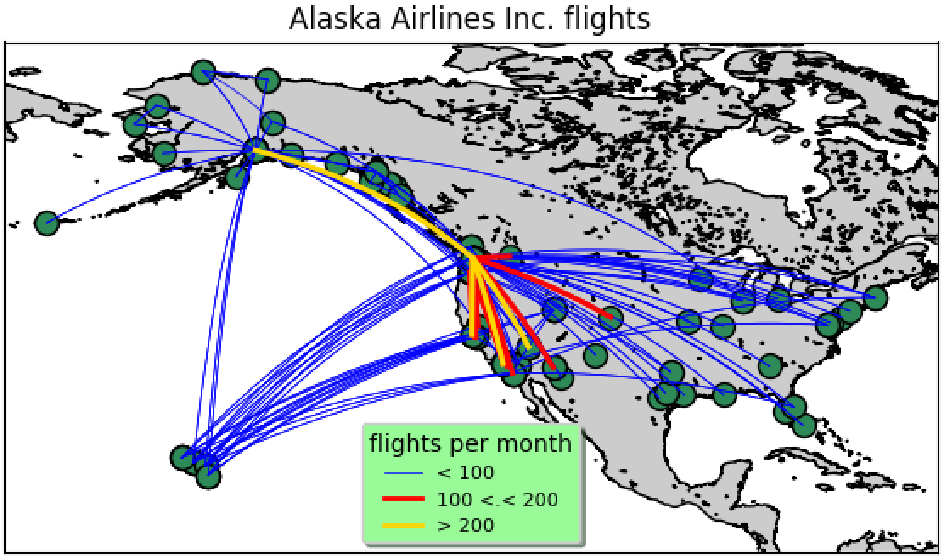
Code

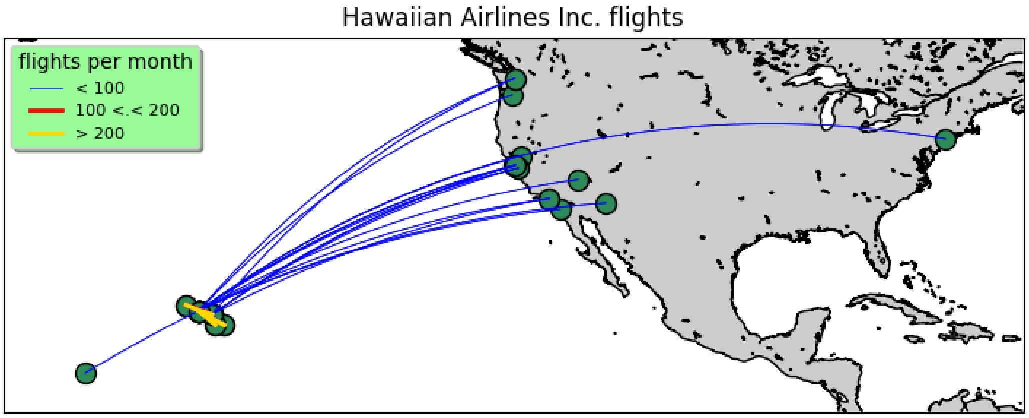
Code

Code

American Airlines Inc. flights
flights per month
< 100
100 <.< 200
> 200







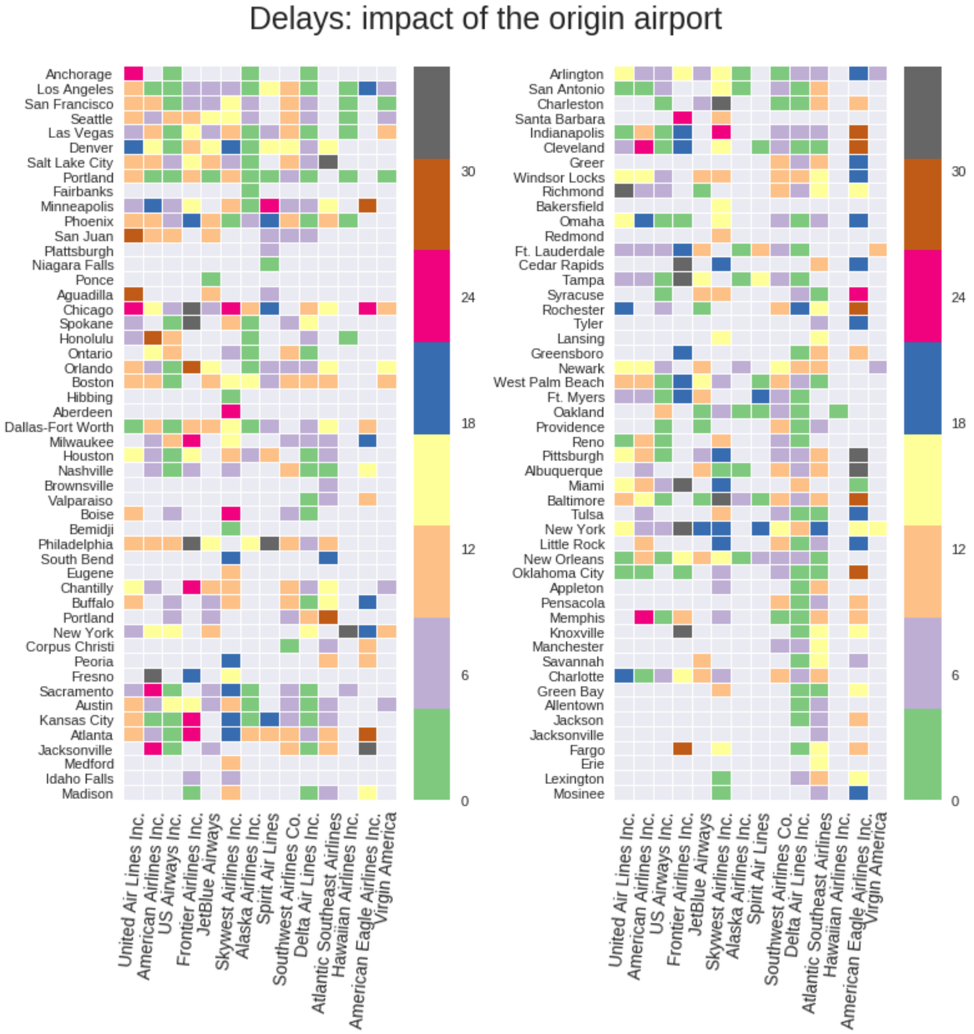
4.2 How the origin airport impact delays

In this section, I will have a look at the variations of the delays with respect to the origin airport and for every airline. The first step thus consists in determining the mean delays per airport:

Code

Since the number of airports is quite large, a graph showing all the information at once would be a bit messy, since it would represent around 4400 values (i.e. 312 airports × 14 airlines). Hence, I just represent a subset of the data:

Code



This figure allows to draw some conclusions. First, by looking at the data associated with the different airlines, we find the behavior we previously observed: for example, if we consider the right panel, it will be seen that the column associated with *American Eagle Airlines* mostly reports large delays, while the column associated with *Delta Airlines* is mainly associated with delays of less than 5 minutes. If we now look at the airports of origin, we will see that some airports favor late departures: see e.g. Denver, Chicago or New York. Conversely, other airports will mainly know on time departures such as Portland or Oakland.

Finally, we can deduce from these observations that there is a high variability in average delays, both between the different airports but also between the different airlines. This is important because it implies that in order to accurately model the delays, it will be necessary to adopt a model that is **specific to the company and the home airport** .

4.3 Flights with usual delays ?

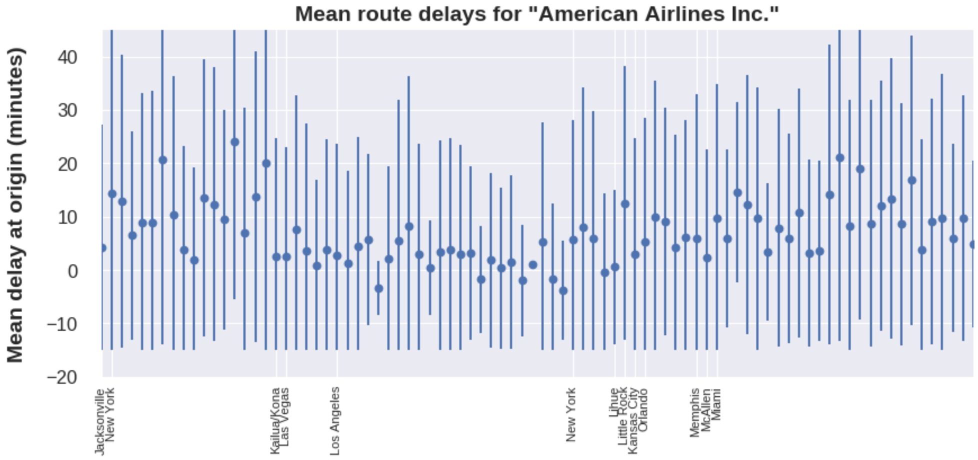
In the previous section, it has been seen that there is variability in delays when considering the different airlines and the different airports of origin. I'm now going to add a level of granularity by focusing not just on the original airports but on flights: origin $\longrightarrow$ destination. The objective here is to see if some flights are systematically delayed or if, on the contrary, there are flights that would always be on time.

In the following, I consider the case of a single airline. I list all the flights A $\longrightarrow$ B carried out by this company and for each of them, I create the list of delays that have been recorded:

Code

I then calculate the average delay on the various paths A $\longrightarrow$ B, as well as the standard deviation and once done, I create a graphical representation (for a sample of the flights):

Code



This figure gives the average delays for *American Airlines*, according to the city of origin and the destination (note that on the abscissa axis, only the origin is indicated for the sake of clarity). The error bars associated with the different paths correspond to the standard deviations. In this example, it can be seen that for a given airport of origin, delays will fluctuate depending on the destination. We see, for example, that here the greatest variations are obtained for New York or Miami where the initial average delays vary between 0 and $\sim$20 minutes.

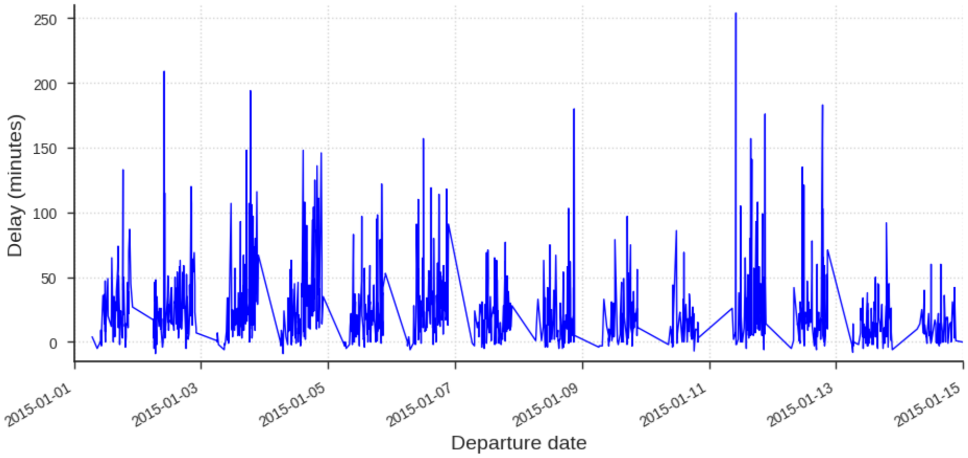
4. Temporal variability of delays

In this section, I look at the way delays vary with time. Considering the case of a specific airline and airport, delays can be easily represented by day and time (*aside*: before doing this, I define a class that I will use extensively in what follows to produce graphs):

Code

Code

Code

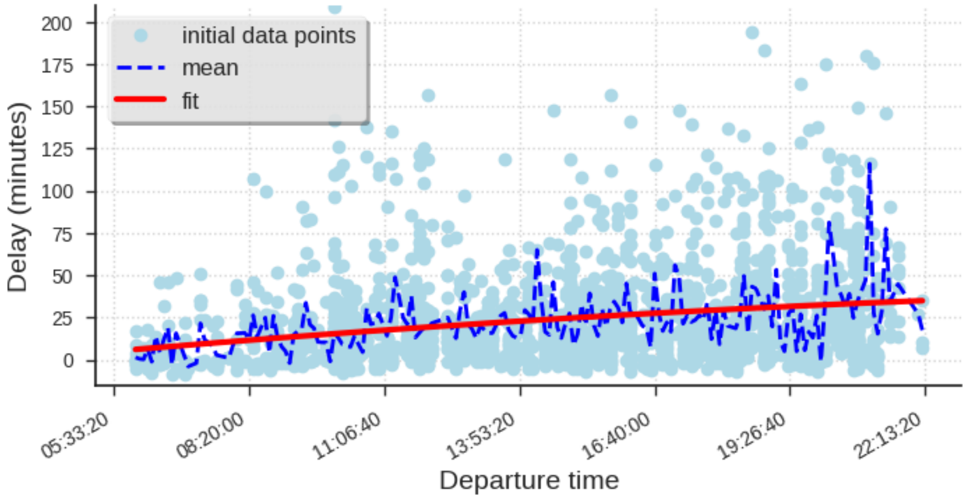


This figure shows the existence of cycles, both in the frequency of the delays but also in their magnitude. In fact, intuitively, it seems quite logical to observe such cycles since they will be a consequence of the day-night alternation and the fact that the airport activity will be greatly reduced (if not inexistent) during the night. This suggests that a **important variable** in the modeling of delays will be **take-off time**. To check this hypothesis, I look at the behavior of the mean delay as a function of departure time, aggregating the data of the current month:

Code

which visually gives:

Code



Here, we can see that the average delay tends to increase with the departure time of day: flights leave on time in the morning and the delay grows almost monotonously up to 30 minutes at the end of the day. In fact, this behavior is quite general and looking at other aiports or companies, we would find similar trends.

6. Predicting flight delays

The previsous sections dealt with an exploration of the dataset. Here, I start with the modeling of flight delays. In this section, my goal is to create a model that uses a window of 3 weeks to predict the delays of the following week. Hence, I decide to work on the data of January with the aim of predicting the delays of the epoch $23^{th} - 31^{th}$ of Januaray

Code

5.1 Model nº1: one airline, one airport

I first decide to model the delays by considering separately the different airlines and by splitting the data according to the different home airports. This first model can be seen as a *"toy-model"* that enables to identify problems that may arise at the production stage. When treating the whole dataset, the number of fits will be large. Hence we have to be sure that the automation of the whole process is robust enough to insure the quality of the fits.

5.1.1 Pitfalls

a) Unsufficient statistics

First of all, I consider the *American Airlines* flights and make a census of the number of flights that left each airport:

Code

```
Out[36]:
```

	count	max	mean	min
ORIGIN_AIRPORT				
JAC	25.0	47.0	-3.640000	-19.0
GUC	22.0	199.0	13.227273	-24.0
SDF	19.0	55.0	8.421053	-8.0
LIT	9.0	74.0	12.555556	-5.0
MTJ	3.0	51.0	26.000000	-2.0

Looking at this list, we can see that the less visited aiports only have a few flights in a month. Thus, in the least favorable case, it is impossible to perform a regression.

b) Extreme delays

Another pitfall to avoid is that of "accidental" delays: a particular attention should be paid to extreme delays. Indeed, during the exploration, it was seen that occasionally, delays of several hours (even tens of hours) could be recorded. This type of delay is however marginal (a few %) and the cause of these delays is probably linked to unpredictable events (weather, breakdown, accident, ...). On the other hand, taking into account delays of this type will likely introduce a bias in the analysis. Moreover, the weight taking into account a delay of this type will likely introduce a bias in the analysis. Moreover, the weight taken by large values will be significant if we have a small statistics.

In order to illustrate this, I first define a function that calculates the mean flights delay per airline and per airport:

Code

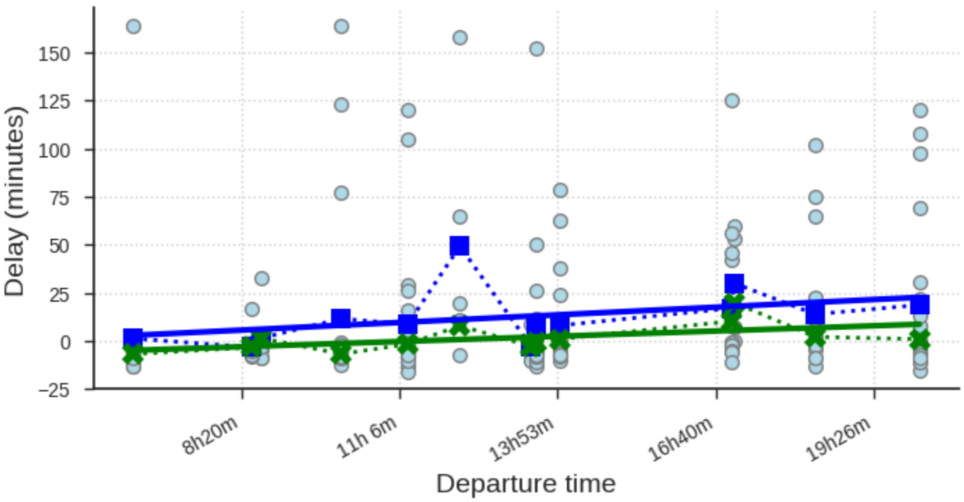
and then a function that performs a linear regression on these values:

Code

I then consider two scenarios. In the first case, I take all the initial values and in the second case, I eliminate all delays greater than 1h before calculating the average delay. The comparison of the two cases is quite explicit:

Code

Code



First of all, in this figure, the points corresponding to the individual flights are represented by the points in gray. The mean of these points gives the mean delays and the mean of the set of initial points corresponds to the blue squares. By removing extreme delays (> 1h), one obtains the average delays represented by the green crosses. Thus, in the first case, the fit (solid blue curve) leads to a prediction which corresponds to an average delay of $\sim$ 10 minutes larger than the predicton obtained in the second case (green curve), and this, at any hour of the day.

In conclusion, we see in this example that the way in which we manage the extreme delays will have an important impact on the modeling. Note, however, that the current example corresponds to a *chosen case* where the impact of extreme delays is magnified by the limited number of flights. Presumably, the impact of such delays will be less pronounced in the majority of cases.

5.1.2 Polynomial degree: splitting the dataset

In practice, rather than performing a simple linear regression, we can improve the model doing a fit with a polynomial of order N. Doing so, it is necessary to define the degree N which is optimal to represent the data. When increasing the polynomial order, it is important **to prevent over-fitting** and we do this by splitting the dataset in **test and training sets**. A problem that may arise with this procedure is that the model ends by *indirectly* learning the contents of the test set and is thus biased. To avoid this, the data can be re-separated into 3 sets: *train*, *test* and *validation*. An alternative to this technique, which is often more robust, is the so-called cross-validation method. This method consists of performing a first separation of the data in *training* and *test* sets. As always, learning is done on the training set, but to avoid over-learning, it is split into several pieces that are used alternately for training and testing.

Note that if the data set is small, the separation in test & training sets can introduce a bias in the estimation of the parameters. In practice, the *cross-validation* method avoids such bias. In fact, in the current model, we will encounter this type of problem and in what follows, I will highlight this. For example, we can consider an extreme case where, after separation, the training set would contain only hours $<$20h and the test set would have hours$>$ 20h. The model would then be unable to reproduce precisely this data, of which it would not have seen equivalent during the training. The cross-validation method avoids this bias because all the data are used successively to drive the model.

a) Bias introduced by the separation of the data set

In order to test the impact of data separation on model determination, I first define the class *fit_polynome* :

Code

The *fit_polynome* class allows you to perform all operations related to a fit and to save the results. When calling the **split()** method, the variable '*method*' defines how the initial data is separated:

- *method = 'all'* : all input data is used to train and then test the model
- *method = 'split'* : we use the *train_test_split()* method of sklearn to define test & training sets

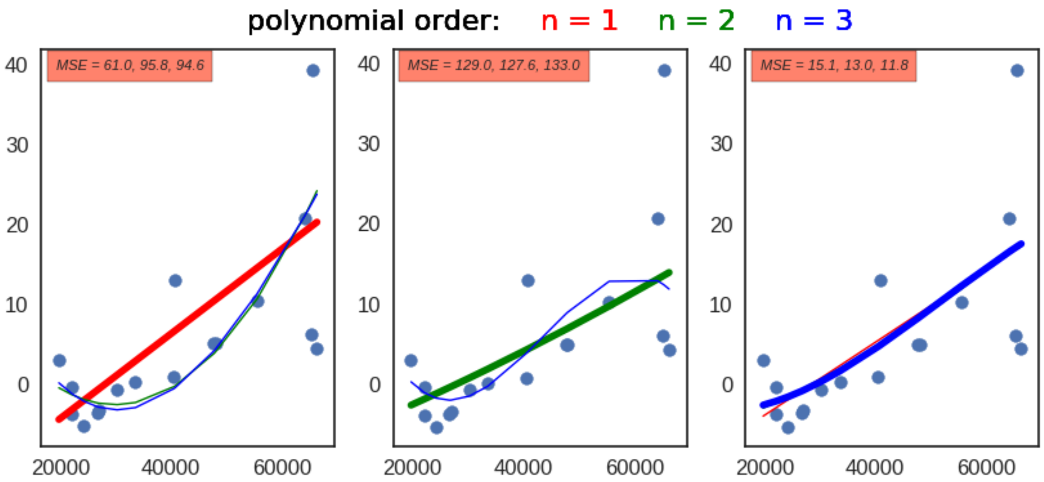
Then, the other methods of the class have the following functions:

- **train (n)** : drives the data on the training set and makes a polynomial of order n
- **predict (X)** : calculates the Y points associated with the X input and for the previously driven model
- **calc_score ()** : calculates the model score in relation to the test set data

In order to illustrate the bias introduced by the selection of the test set, I proceed in the following way: I carry out several "train / test" separation of a data set and for each case, I fit polynomials of orders **n = 1, 2 and 3** , by calculating their respective scores. Then, I show that according to the choice of separation, the best score can be obtained with any of the values of **n** . In practice, it is enough to carry out a dozen models to obtain this result. Moreover, this bias is introduced by the choice of the separation "train / test" and results from the small size of the dataset to be modeled. In fact, in the following, I take as an example the case of the airline *American Airlines* (the second biggest airline) and the airport of id 1129804, which is the airport with the most registered flights for that company. This is one of the least favorable scenarios for the emergence of this kind of bias, which, nevertheless, is present:

Code

```
modèle nº1 , min. pour n = 1, score = 61.0
modèle nº2 , min. pour n = 1, score = 16.1
modèle nº3 , min. pour n = 1, score = 174.1
modèle nº4 , min. pour n = 2, score = 127.6
modèle nº5 , min. pour n = 3, score = 11.8
```



In this figure, the panels from left to right correspond to 3 separations of the data in train and test sets, for which the best models are obtained respectively with polynomials of order 1, 2 and 3. On each of these panels the 3 fits polynomials have been represented and the best model corresponds to the thick curve.

b) Selection by cross-validation

One of the advantages of the cross-validation method is that it avoids the bias that has just been put forward when choosing the polynomial degree. In order to use this method, I define a new class that I will use later to perform the fits:

Code

This class has two methods:

- **train (n, nb_folds)** : defined 'nb_folds' training sets from the initial dataset and drives a 'n' order polynomial on each of these sets. This method returns as a result the Y predictions obtained for the different test sets.
- **calc_score (n, nb_folds)** : performs the same procedure as a **train** method except that this method calculates the fit score and not the predicted values on the different test data.

By default, the *'K-fold'* method is used by sklearn *cross_val_predict ()* and *cross_val_score ()* methods. These methods are deterministic in the choice of the K folds, which implies that for a fixed K value, the results obtained using these methods will always be identical. As seen in the previous example, this was not the case when using the *train_test_split()* method. Thus, if we take the same dataset as in the previous example, the method of cross validation makes it possible to choose the best polynomial degree:

Code

```
Max possible number of folds: 16

n=1 -> MSE = 130.629
n=2 -> MSE = 151.79
n=3 -> MSE = 159.455
n=4 -> MSE = 162.631
n=5 -> MSE = 166.966
n=6 -> MSE = 173.08
n=7 -> MSE = 181.361
```

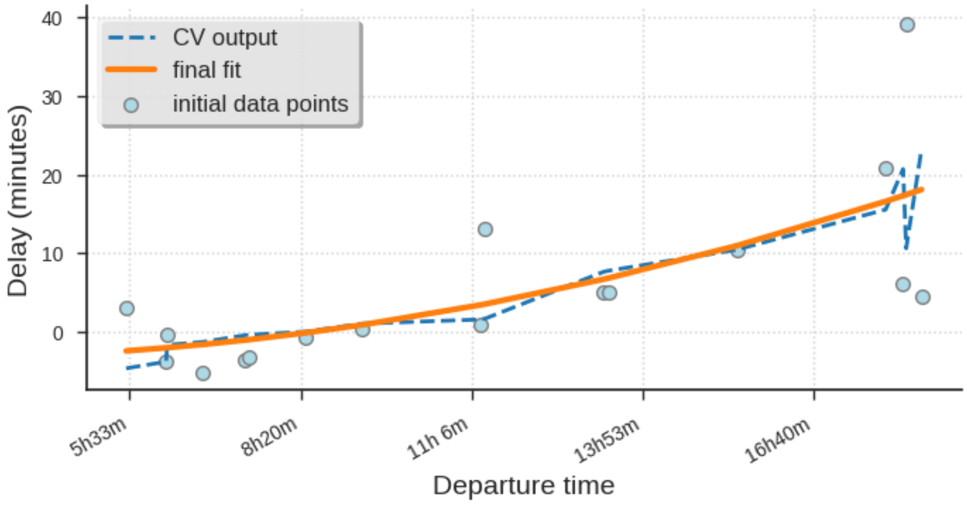
We can see that using this method gives us that the best model (ie the best generalized model) is obtained with a polynomial of order 2. At this stage of the procedure, the choice of the polynomial order a has been validated and we can now use all the data in order to perform the fit:

Code

Thus, in the following figure, the juxtaposition of the K = 50 polynomial fits corresponding to the cross validation calculation leads to the red curve. The polynomial fit corresponding to the final model corresponds to the blue curve.

Code

Code



Code

```
Out[48]:
         56.862847718920953
```

5.1.3 Model test: prediction of end-January delays

At this stage, the model was driven is tested on the training set which include the data of the first 3 weeks of January. We now look at the comparison of predictions and observations for the fourth week of January:

Code

and the MSE score of the model is:

Code

```
Out[50]:
        108.67130851577079
```

To get an idea of the meaning of such a value for the MSE, we can assume a constant error on each point of the dataset. In which case, at each point i, we have:

$$y_i - f(x_i) = cste = \sqrt{MSE}$$

thus giving the difference in minutes between the predicted delay and the actual delay. In this case, the difference between the model and the observations is thus typically:

Code

```
Out[51]:
        'Ecart = 10.42 min'
```

5.2 Model nº2: One airline, all airports

In the previous section, the model only considered one airport. This procedure is potentially inefficient because it is likely that some of the observations can be extrapolated from an ariport to another. Thus, it may be advantageous to make a single fit, which would take all the airports into account. In particular, this will allow to predict delays on airports for which the number of data is low with a better accuracy.

Code

Code

```
Out[53]:
        (1831, 3)
```

In the *merged_df* dataframe, airports are referenced by an identifier given in the **ORIGIN_AIRPORT** variable. The corresponding labels can't be used directly in a fit and I thus use the *one-hot-encoding* method:

Code

```
Out[54]:
          [(0, 'ABQ'), (1, 'ATL'), (2, 'AUS'), (3, 'BDL'), (4, 'BHM')]
```

Above, I have assigned a label to each airport. The correspondence between the label and the original identifier has been saved in the *label_airport* list. Now I proceed with the "One Hot Encoding" by creating a matrix where instead of the **ORIGIN_AIRPORT** variable that contained M labels, we build a matrix with M columns, filled of 0 and 1 depending on the correspondance with particular airports:

Code

```
(1831, 82) (1831, 1)
```

5.2.1 Linear regression

The matrices X and Y thus created can be used to perform a linear regression:

Code

```
MSE = 53.7430736542
```

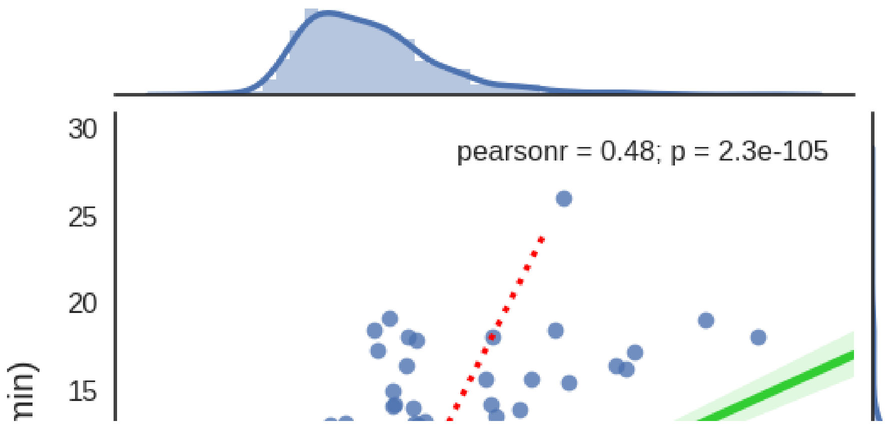
Here, I calculated the MSE score of the fit. In practice, we can have a feeling of the quality of the fit by considering the number of predictions where the differences with real values is greater than 15 minutes:

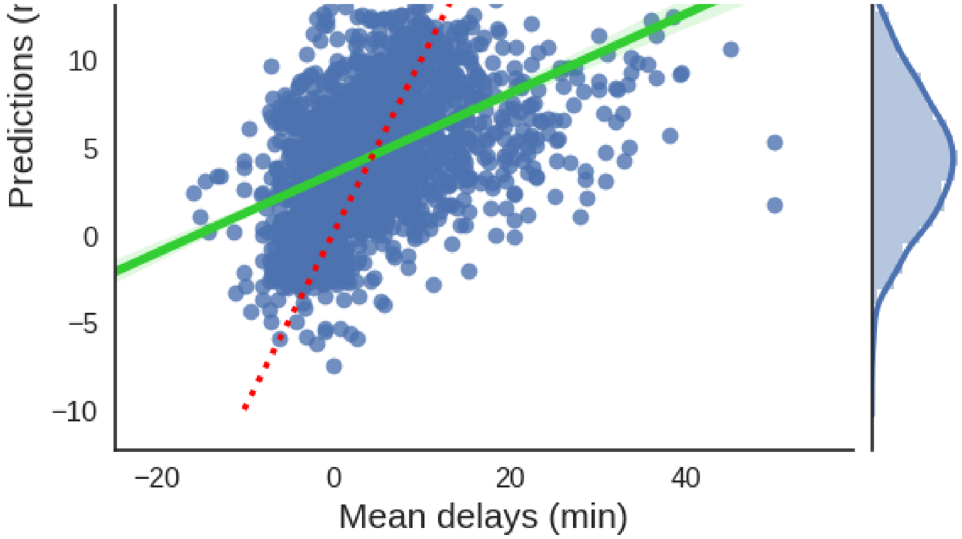
Code

```
Out[57]:
          '5.30%'
```

In practice, this model tends to underestimate the large delays, which can be seen in the following figure:

Code





5.2.2 Polynomial regression

I will now extend the previous fit by using a polynomial rather than a linear function:

Code

```
Out[59]:
          LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=F
          alse)
```

Code

```
          MSE = 49.5025438214
```

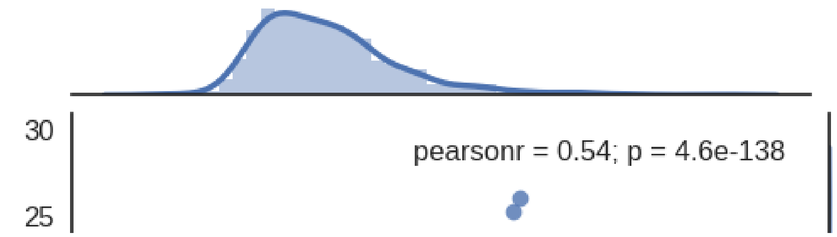
We can see that a polynomial fit improves slightly the MSE score. In practice, the percentage of values where the difference between predictions and real delays is greater than 15 minutes is:

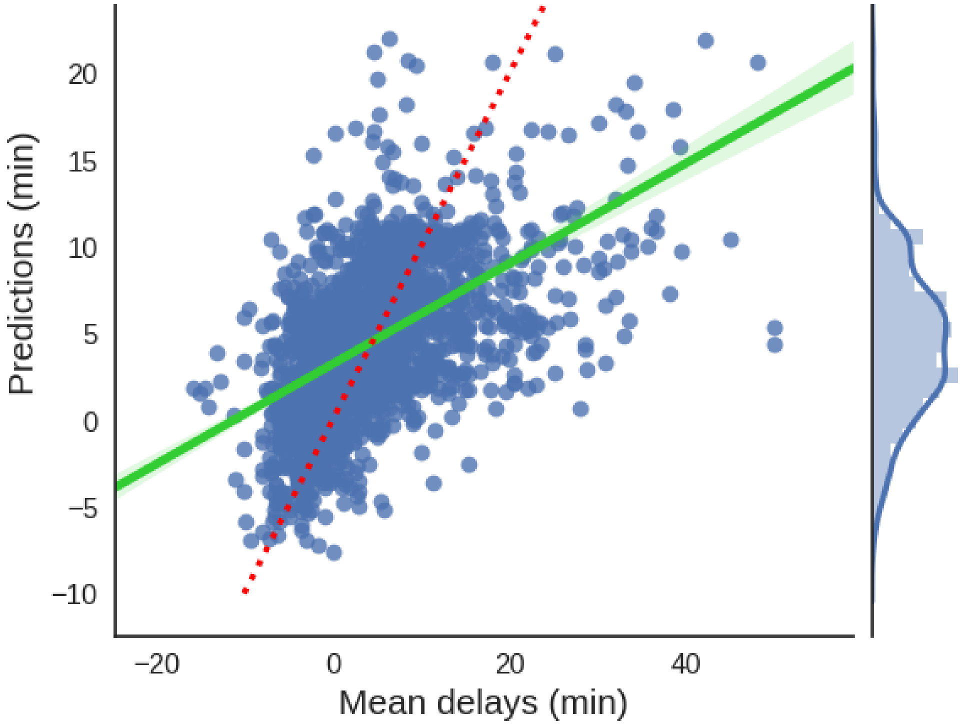
Code

```
Out[61]:
          '4.81%'
```

And as before, it can be seen that model tends to be worse in the case of large delays:

Code





5.2.3 Setting the free parameters

Above, the two models were fit and tested on the training set. In practice, as mentioned above, there is a risk of overfitting by proceeding that way and the free parameters of the model will be biased. Hence, the model will not allow a good generalization. In what follows, I will therefore split the datas in order to train and then test the model. The purpose will be to determine the polynomial degree which allows the best generalization of the predictions:

Code

As before, I fit the model on the training set:

Code

```
Out[64]:
(1281, 82)
```

Code

```
Mean squared error =  46.3925583884
```

Now, by testing on the test set we get:

Code

```
Mean squared error =  208151.428081
```

Here, we see that the **fit is particularly bad with a MSE > 500** (the exact value depends on the run and on the splitting of the dataset), which means that the fit performs poorly when generalyzing to other data. Now let's examine in detail the reasons why we have such a bad score. Below, I examing all the terms of the MSE calculation and identify the largest terms:

Code

```
253024.0 -503.0 0.0
55989950.2 7508.6 26.0
57930603.6 7615.6 4.4
262655.8 -506.3 6.2
```

We see that some predictions show very large errors. In practice, this can be explained by the fact that during the separation in train and test sets, **data with no equivalent in the training set was put in the test data**. Thus, when calculating the prediction, the model has to **perform an extrapolation**. If the coefficients of the fit are large (which is often the case when overfitting), extrapolated values will show important values, as in the present case. In order to have a control over this phenomenon, we can use a **regularization method** which will put a penalty to the models whose coefficients are the most important:

Code

```
Out[68]:
Ridge(alpha=0.3, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=True, random_state=None, solver='auto', tol=0.001)
```

Now, if we calculate the score associated to the predictions made with a regularization technique, we have:

Code

```
Mean squared error =  65.4841834479
```

And we can see that we obtain a reasonnable score. Hence, with the current procedure, to determine the best model, we have two free parameters to adjust: the polynomial order and the α coefficient of the *'Ridge Regression'* :

Code

```
n=1 alpha=0 , MSE = 65.29
n=1 alpha=2 , MSE = 64.424
n=1 alpha=4 , MSE = 64.29
n=1 alpha=6 , MSE = 64.458
n=1 alpha=8 , MSE = 64.75
n=1 alpha=10 , MSE = 65.09
n=1 alpha=12 , MSE = 65.439
n=1 alpha=14 , MSE = 65.78
```

```
n=1 alpha=14 , MSE = 65.78
n=1 alpha=16 , MSE = 66.105
n=1 alpha=18 , MSE = 66.411
n=2 alpha=0 , MSE = 9.6708e+17
n=2 alpha=2 , MSE = 65.612
n=2 alpha=4 , MSE = 65.396
n=2 alpha=6 , MSE = 65.285
n=2 alpha=8 , MSE = 65.236
n=2 alpha=10 , MSE = 65.235
n=2 alpha=12 , MSE = 65.27
n=2 alpha=14 , MSE = 65.33
n=2 alpha=16 , MSE = 65.409
n=2 alpha=18 , MSE = 65.501
```

This grid search allows to find the best set of α and n parameters. Let us note, however, that for this model, the estimates obtained with a linear regression or a polynomial of order 2 are quite close. Now I use these parameters to test this template over the test set:

Code

```
54.9920975427
```

6.2.4 Testing the model: delays of end-january

At this stage, model predictions are tested against end-January data. These data are first extracted:

Code

then I convert them into a format suitable to perform the fit. At this stage, I manually do one-hot-encoding by re-using the labeling that had been established on the training data:

Code

I can then create the predictions

Code

```
Out[74]:
         'MSE = 59.36'
```

As before, assuming that the delay is independent of the point, this MSE score is equivalent to an average delay of:

Code

```
Out[75]:
         'Ecart = 7.70 min'
```

The current MSE score is calculated on all the airports served by *American Airlines*, whereas previously it was calculated on the data of a single airport. The current model is therefore more general. Moreover, considering the previous model, it is likely that predictions will be poor for airports with low statistics.

6.3 Model nº3: Accounting for destinations

In the previous model, I grouped the flights per departure time. Thus, flights with different destinations were grouped as soon as they leave at the same time. Now I make a model that accounts for both departure and arrival times:

Code

Code

```
Out[77]:
```

	heure_depart	heure_arrivee	ORIGIN_AIRPORT	DEPARTURE_DELAY	weekday
0	300	17640	LAX	2.133333	2.800000
1	300	17700	LAX	5.500000	3.750000
2	600	28200	LAX	-6.000000	3.250000
3	1200	29040	LAX	-4.117647	2.823529
4	1200	29100	LAX	0.800000	3.600000

Henceforth, regroupings are made on departure and arrival times, and the (specific) airports of origin and destination are implicitly taken into account. As before, I carry out the encoding of the airports:

Code

6.3.1 Choice of model parameters

As before, I will perform a regression with regularization and I will have to define the value to attribute to the parameter α. I therefore separate the data to train and then test the model to select the best value for α:

Code

Code

```
n=1 alpha=0.0 , MSE = 85.599
n=1 alpha=0.2 , MSE = 84.456
n=1 alpha=0.4 , MSE = 84.313
n=1 alpha=0.6 , MSE = 84.598
n=1 alpha=0.8 , MSE = 85.082
n=1 alpha=1.0 , MSE = 85.655
```

```
n=1 alpha=1.0 , MSE = 85.655
n=1 alpha=1.2 , MSE = 86.26
n=1 alpha=1.4 , MSE = 86.867
n=1 alpha=1.6 , MSE = 87.46
n=1 alpha=1.8 , MSE = 88.03
n=2 alpha=0.0 , MSE = 9.4162e+12
n=2 alpha=0.2 , MSE = 86.377
n=2 alpha=0.4 , MSE = 85.564
n=2 alpha=0.6 , MSE = 85.185
n=2 alpha=0.8 , MSE = 84.98
n=2 alpha=1.0 , MSE = 84.874
n=2 alpha=1.2 , MSE = 84.837
n=2 alpha=1.4 , MSE = 84.85
n=2 alpha=1.6 , MSE = 84.901
n=2 alpha=1.8 , MSE = 84.982
```

Code

```
89.5503177274
```

6.3.2 Test of the model: late January delays

Now I test the quality of the predictions on the data of the last week of January. I first extract these data:

Code

```
Out[82]:
```

	heure_depart	heure_arrivee	ORIGIN_AIRPORT	DEPARTURE_DELAY	weekday
0	300	17640	LAX	-4.000	3.25
1	1200	29040	LAX	5.125	3.25
2	1800	20340	SFO	-6.750	3.25
3	2700	29340	LAS	-4.500	3.25
4	3900	31800	LAX	-4.875	3.25

Code

Code

```
MSE = 74.8
```

which corresponds to an average delay of:

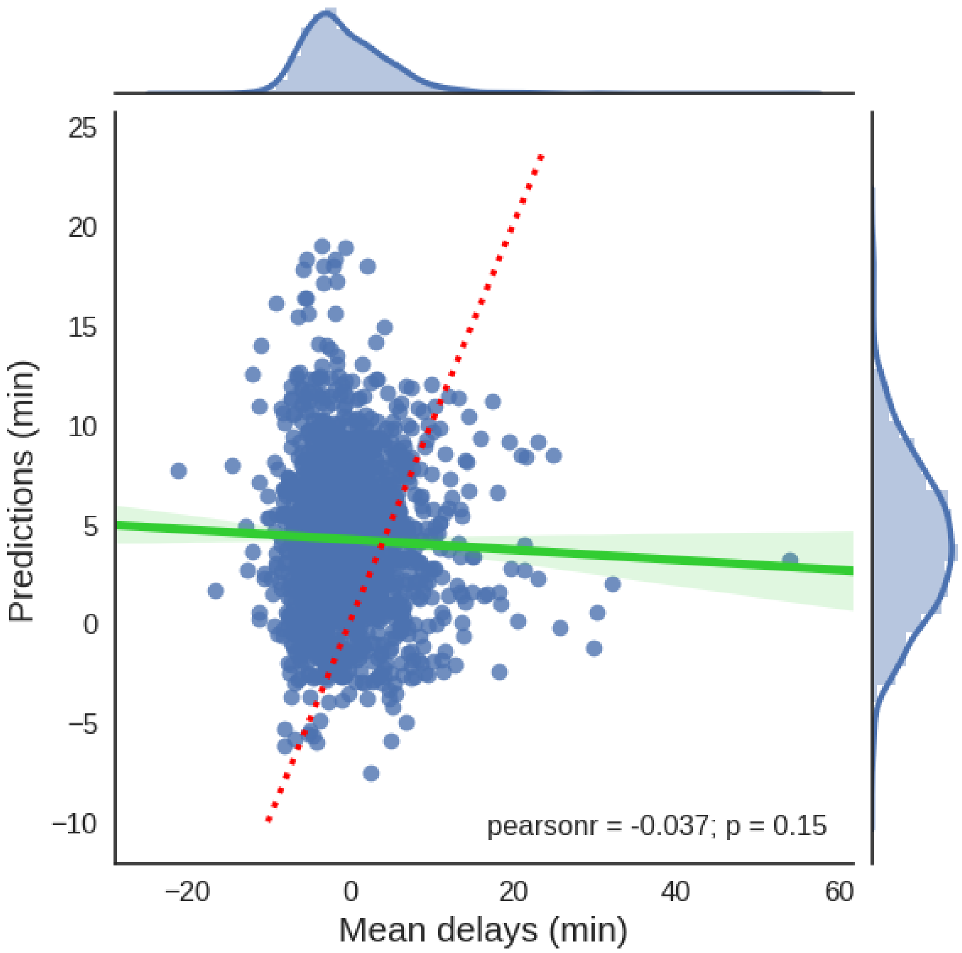
Code

```
Out[85]:
         'Ecart = 8.65 min'
```

Code

```
ecarts > 15 minutes: 4.588%
```

Code



Conclusion

These notebook was two-fold. The first part dealt with an exploration of the dataset, with the aim of understanding some properties of the delays registered by flights. This exploration gave me the occasion of using various vizualization tools offered by python. The second part of the notebook consisted in the elaboration of a model aimed at predicting flight delays. For that purpose, I used polynomial regressions and showed the importance of regularisation techniques. In fact, I only used ridge regression but it is important to keep in mind that other regularisations techniques could be more appropriate (e.g Lasso or Elastic net).

If you see any kind of improvement, or mistakes, thanks in advance for telling me !!
If you liked this notebook, thanks for upvoting **:)**

Did you find this Kernel useful?
Show your appreciation with an upvote

180

Comments (62)

All Comments

Sort by Hotness

Please sign in to leave a comment.

DSEverything • Posted on Latest Version • 8 months ago • Options

2

Thanks FabienDaniel. Really awesome kernel. I will bookkeep some work here by borrowing a few your great stuffs to my kernel.

FabienDaniel Kernel Author • Posted on Latest Version • 8 months ago • Options

0

Cool, I'm happy you find this kernel useful !

Selfish Gene • Posted on Version 57 • 9 months ago • Options

1

This is very good.
I'm now following you @fabiendaniel!

FabienDaniel **Kernel Author** • Posted on Version 57 • 9 months ago • Options

0

Thanks !! I'm happy you like it :)

yyll008 • Posted on Version 56 • 9 months ago • Options

1

Awesome work ! 👍

yuyang • Posted on Version 52 • 10 months ago • Options

1

description of these variables

- **DEPARTURE_DELAY** and **ARRIVAL_TIME**: difference (in minutes) between planned and real times

ARIVAL_TIME should be ARRIVAL_DELAY

FabienDaniel **Kernel Author** • Posted on Version 52 • 10 months ago • Options

0

thanks :)

Shivendra Sharma • Posted on Version 52 • 10 months ago • Options

1

Should be tagged as one of the most comprehensive Python tutorials. This covers everything. I'm thinking of doing a similar thing through R.

A^b • Posted on Version 42 • 10 months ago • Options

1

Nice tutorial, lots of interesting stuff here!

FabienDaniel Kernel Author • Posted on Version 42 • 10 months ago • Options 0

thanks :)

Gregory Olson • Posted on Version 36 • 10 months ago • Options 1

Thanks for the great tutorial. Really great to see a practical example related to what I'm currently learning in an Andrew Ng course!

FabienDaniel Kernel Author • Posted on Version 36 • 10 months ago • Options 0

Thanks, I'm really happy you like it :)

Asura • Posted on Version 35 • 10 months ago • Options 1

Great notebook, thanks Fabien!

Ganesh Raskar • Posted on Version 31 • 10 months ago • Options 1

Great notebook, thanks Fabien!

geher • Posted on Version 29 • 10 months ago • Options 1

great kernels recently Fabien, many thanks

FabienDaniel Kernel Author • Posted on Version 29 • 10 months ago • Options 0

Thank you !! I'm happy you appreciate my kernels :)

kanavanand • Posted on Version 28 • 10 months ago • Options 1

Awesome!Thanks for sharing :)

Luiz Gustavo Mori • Posted on Version 26 • 10 months ago • Options 1

Nice.

FuLin • Posted on Version 26 • 10 months ago • Options 1

This is awesome. Thank you for sharing!

Fiona Wu • Posted on Version 21 • 10 months ago • Options 1

Great visualisations ! Thanks.

Evangelos Katsar... • Posted on Version 39 • 10 months ago • Options 2

Good job Fabien! Thanks for sharing. I 'll try to reproduce (for my own interest) a few of your findings in R. Thanks again!

> **FabienDaniel** Kernel Author • Posted on Version 39 • 10 months ago • Options 1
>
> Great ! I really look forward to see your notebooks. That's a good way to cross-check everything !!

Kueipo J. H. • Posted on Version 24 • 10 months ago • Options 2

great EDA. Does kaggle support mathjax or latex ?

> **FabienDaniel** Kernel Author • Posted on Version 24 • 10 months ago • Options 0
>
> Thanks ! I don't know for mathjax but yes for latex.

Asura • Posted on Version 35 • 10 months ago • Options 0

I got AxisError: axis 1 is out of bounds for array of dimension 1 in In [4] and RuntimeError: xdata and ydata must be the same length in In [29]. How to resolve it? I am using Python 2.7.13

> **FabienDaniel** Kernel Author • Posted on Version 35 • 10 months ago • Options 0
>
> I've never used python 2.7, so I'm not sure I can help much ... why don't you use python 3 ?
> For the cell [4], you should try to identify which array (or list) you are out of bounds, then printing

the size of the array (with the len() funntion or np.shape() if this is a numpy array).
Cell [29] is also quite big: you should identify at which point of the cell the error occur.

SakshamMalhotra • Posted on Version 59 • 9 months ago • Options 0

You present everything so nicely. Really informative tutorial

SakshamMalhotra • Posted on Version 59 • 9 months ago • Options 0

You present everything so nicely. Really informative tutorial

SakshamMalhotra • Posted on Version 59 • 9 months ago • Options 0

[Deleted User] • Posted on Version 59 • 9 months ago • Options 0

Great kernel

Miklós Barsy • Posted on Version 60 • 9 months ago • Options 0

This is veri good turtorial. It is good to see a practical example that can be utilized in practice in our work.

WendiLi • Posted on Version 60 • 9 months ago • Options 0

nice script !:)

JayaSingh • Posted on Version 60 • 9 months ago • Options 0

very good for beginner.

Astandri K • Posted on Latest Version • 8 months ago • Options 0

very good tutorial. thanks a lot

Shahroz.Nadeem • Posted on Latest Version • 7 months ago • Options 0

The effort you put in making this notebook is apparent. As a fellow Data scientist i know how hectic visualizations can be sometimes. But nevertheless GREAT JOB !!

Pranalli • Posted on Latest Version • 6 months ago • Options 0

Which version of python will be apt for this?

Pranalli • Posted on Latest Version • 5 months ago • Options 0

Can someone please suggest the exact version of Python to be used for this one? It will be of great help. In advance, thanks a lot :) Please help.

FabienDaniel Kernel Author • Posted on Latest Version • 5 months ago • Options 1

I used python 3.x to create this notebook.

Pranalli • Posted on Latest Version • 5 months ago • Options 0

Thank-you so much, Fabein! This notebook is just beyond awesome and helpful. Great work. And, will it better to install anaconda and then proceed or just python 3 will be okay? Thanks in advance :)

FabienDaniel Kernel Author • Posted on Latest Version • 5 months ago • Options 1

Thanks :)
Installing anaconda will make your life much easier since all the packages you need will be installed directly. Otherwise, you'll have to install the packages separately.
You may want to use Kaggle's kernels also, since you'll have everything already installed for you, with good computing capabilities.

Pranalli • Posted on Latest Version • 5 months ago • Options 0

Okay, I'll go with either Anaconda(which will basically serve the purpose of Python 3 with all the packages possible? Right?) or Kaggle kernels. Thank-you so much, for your help! :)

Suyash Khemka • Posted on Latest Version • 5 months ago • Options 0

Have an error when I compute the map for the first time. Kindly help :)

ImportError Traceback (most recent call last) ~\Anaconda3\lib\site-packages\mpl_toolkits\basemap_init_.py in warpimage(self, image, scale, **kwargs) 4025 try: -> 4026 from PIL import Image 4027 except ImportError:

~\Anaconda3\lib\site-packages\PIL\Image.py in () 55 # and should be considered private and subject to change. ---> 56 from . import _imaging as core 57 if PILLOW_VERSION != getattr(core, 'PILLOW_VERSION', None):

ImportError: DLL load failed: The specified module could not be found.

During handling of the above exception, another exception occurred:

ModuleNotFoundError Traceback (most recent call last) ~\Anaconda3\lib\site-packages\mpl_toolkits\basemap_init_.py in warpimage(self, image, scale, **kwargs) 4028 try: -> 4029 import Image 4030 except ImportError:

ModuleNotFoundError: No module named 'Image'

During handling of the above exception, another exception occurred:

ImportError Traceback (most recent call last) in () 6 labels.append("{} <.< {}".format(size_limits[i], size_limits[i+1])) 7 map = Basemap(resolution='i',llcrnrlon=-180, urcrnrlon=-50, llcrnrlat=10, urcrnrlat=75, lat_0=0, lon_0=0,) ----> 8 map.shadedrelief() 9 map.drawcoastlines() 10 map.drawcountries(linewidth = 3)

~\Anaconda3\lib\site-packages\mpl_toolkits\basemap_init_.py in shadedrelief(self, ax, scale, *kwargs) 3977 return self.warpimage(image='shadedrelief',ax=ax,scale=scale,*kwargs) 3978 else: -> 3979 return self.warpimage(image='shadedrelief',scale=scale,*kwargs) 3980 3981 def etopo(self,ax=None,scale=None,*kwargs):

~\Anaconda3\lib\site-packages\mpl_toolkits\basemap_init_.py in warpimage(self, image, scale, **kwargs) 4029 import Image 4030 except ImportError: -> 4031 raise ImportError('warpimage method requires PIL (http://www.pythonware.com/products/pil)') 4032 4033 from matplotlib.image import pil_to_array

ImportError: warpimage method requires PIL (http://www.pythonware.com/products/pil)

Suyash Khemka • Posted on Latest Version • 5 months ago • Options 0

Someone please explain this portion: isize = [i for i, val in enumerate(size_limits) if val < count_flights[code]] ind = isize[-1]

Thanks!!

I,Coder • Posted on Latest Version • 5 months ago • Options 0

Great one !!

kevserşimşek • Posted on Latest Version • 5 months ago • Options 0

very good and covers everything. Thanks !

Subham • Posted on Latest Version • 4 months ago • Options 0

Thanks Fabien Daniel. Very Useful kernel to learn from for a beginner like me.

Gábor Forgács • Posted on Latest Version • 4 months ago • Options

0

Thanks for the awesome notebook. It'll be most valuable for my learning journey.

Wejdan • Posted on Latest Version • 3 months ago • Options

0

I tried use convert DEPARTURE_TIME and ARRIVAL_TIME values to 'HHMM' string to datetime.time but this error appear 'invalid literal for int() with base 10: '23:54:00'' Do you know how to fix this problem??...plz hlep me

shivani • Posted on Latest Version • 3 months ago • Options

0

df_train = df[df['SCHEDULED_DEPARTURE'].apply(lambda x:x.date()) < datetime.date(2015, 1, 23)] File "C:\Users\u\PycharmProjects\Pandas\venv1\lib\site-packages\pandas\core\series.py", line 2551, in apply mapped = lib.map_infer(values, f, convert=convert_dtype) File "pandas/_libs/src\inference.pyx", line 1521, in pandas._libs.lib.map_infer File "C:/Users/u/PycharmProjects/Pandas/start.py", line 670, in df_train = df[df['SCHEDULED_DEPARTURE'].apply(lambda x:x.date()) < datetime.date(2015, 1, 23)] AttributeError: 'long' object has no attribute 'date'

Can you please help me to resolve this error??

RuYu • Posted on Latest Version • 3 months ago • Options

0

Your idea is wonderful!I want to refer to your some ideas to complete my assignment. Can I?

Rajpal Kulhari • Posted on Latest Version • 2 months ago • Options

0

Awesome kernel FabienDaniel, thanks for putting it publicly. I learned a lot from this!

Kritika Singh • Posted on Latest Version • 2 months ago • Options

0

File "", line 6, in format_heure chaine = "{0.04d}".format(int(chaine))

AttributeError: 'int' object has no attribute '04d'

How can I resolve this error.

> **FabienDaniel** Kernel Author • Posted on Latest Version • 2 months ago • Options
>
> 1
>
> should be: `0:04d` rather than `0.04d`

Kritika Singh • Posted on Latest Version • 2 months ago • Options 0

How can we solve this error?

Figure style object has no attribute pos_update

Sridhar Venkatar... • Posted on Latest Version • 2 months ago • Options 0

Excellent..

Sridhar Venkatar... • Posted on Latest Version • 2 months ago • Options 0

Sridhar Venkatar... • Posted on Latest Version • 2 months ago • Options 0

Biswamitra Bisw... • Posted on Latest Version • a month ago • Options 0

Great tutorial. Thanks!

Robin Lemke • Posted on Latest Version • a month ago • Options 0

Great work, great tutorial. Thanks!

Tiago Schuster • Posted on Latest Version • 24 days ago • Options 0

Great kernel!

Jack • Posted on Latest Version • 20 days ago • Options 0

Really awesome detailed explanation, very helpful in learning data visualization in python. List titles under paragraph "6.Predicting flight delays" have typos, it should be "6.1" and so on. May I known why you prefer OneHotEncoder instead of LabelEncoder() in this case?

harshitha • Posted on Latest Version • 20 days ago • Options 0

I was confused about not understanding how to describe my findings in any kernel. The description u wrote in this gave me a good insight into it. Thanks, FabienDaniel.

Similar Kernels

Statistics And EDA Tutorial For Beginners

Data ScienceTutorial For Beginners

Home Credit : Complete EDA + Feature Importance ✓✓

Titanic Survival Prediction End To End ML Pipeline

Home Credit Default Risk Extensive EDA

Our Team Terms Privacy Contact/Support