# Raport for Flight Delay Predicion Project

Dominik Olejarz, Patryk Rybak, Maksymilian Wnuk

February 11, 2024

## 1 Problem Statement

Our goal is to predict flight delays and, in the future, expand the program to calculate the most probable delay values. Throughout our work, we will utilize a dataset containing flight information from 2017 to 2018, provided by the Bureau of Transportation Statistics in conjunction with the Weather API. The primary focus is to examine the correlation between weather conditions and flight delays.

Our approach centers around machine learning techniques, and we will employ statistical learning models using Python. The aim is to forecast flight delays and in future provide precise values of delays. We believe that our project can contribute to improving the travel experience by offering more accurate information about potential flight delays.

## 2 Data downloading

Getting data for our machine learning models requires two combined datasets, flights and weather:

### 2.1 Flights data

We are getting 2017-2018 flights data from this: `https://www.kaggle.com/datasets/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018` dataset from well known site Kaggle. It contains of columns:

- Airline - company name of airline operating flight
- Origin city - city of departure
- Destination city - city of arrival
- CRS dep time, CRS arr time - expected time of departure and arrive in (weirdly represented) unsigned integer in which least signitficant digits represent minutes and most significant represent hours
- Arr time - factual time of arrive

- Cancelled, Delay - boolean values signifying whether the flight was cancelled or (respectively) delayed
- Distance - distance between origin and destination
- Arrive delay - signed integer value of delay

## 2.2 Weather data

Fetching data for weather requires more work than just downloading a csv file. We will be downloading it by making api requests to `https://www.visualcrossing.com`. Thanks to Visual Crossing® company, we were able to make those requests in unlimited way. We contacted their support and then they gave us access to unlimited synchronous API queries.
However, we couldn't make 12 millions of api queries. Here we encounter our first problem.
**Solution:**
It wasn't that easy, we came up with idea of sending one request for one year for certain place. Api allowed us to make such queries. So in order to do this, we send request of getting timeline of whole year, given city, year (2017 and later 2018), state, starting and finishing date (1'st january to 31'st december). What we get is a whole timeline for the city. We can now easily merge data from flights with weathers, using matrix with weather for according place and time.

We get columns:

- Temperature - in fahrenheit scale
- Cloudcover - how much of the sky is covered in percentage
- Monphase - fractional portion through moon lunation cycle, 0-new moon to 0.5 ful moon and back to 1.0 next new moon
- Windspeed, windgust and winddir
- Snow and snowdepth - amount of snow that fell and depth of snow on the ground
- Dew - dew point temperature in fahrenheit
- Humidity - percentage of relative humidity
- Boolean represented variables:
  - ice
  - freezing rain
  - snow
  - rain

# 3 Data preprocessing

## 3.1 NaN's removal

Dataset contains of lots of NaN's. We removed all of them, by removing rows that contained them. Data loss was incredibly low, we lost about 10,000 of rows that contained at least one NaN, which is great. Additionally, we removed columns that contained A LOT of NaN's. Those columns were:

- Windgust -75.22% NaNs
- Snow - 8.13% NaNs
- Snowdepth - 8.13% NaNs
- ID,id - 55% and 44% respectively,no idea why so many nans, its just id's whose we don't even use in our model.

However, we are not sure if that was proper way of doing this, hence we ask ourselves: does snow affect flight delays? How about snowdepth and wingust? Later on we can always retrieve those columns and check correlation.

## 3.2 Normalization

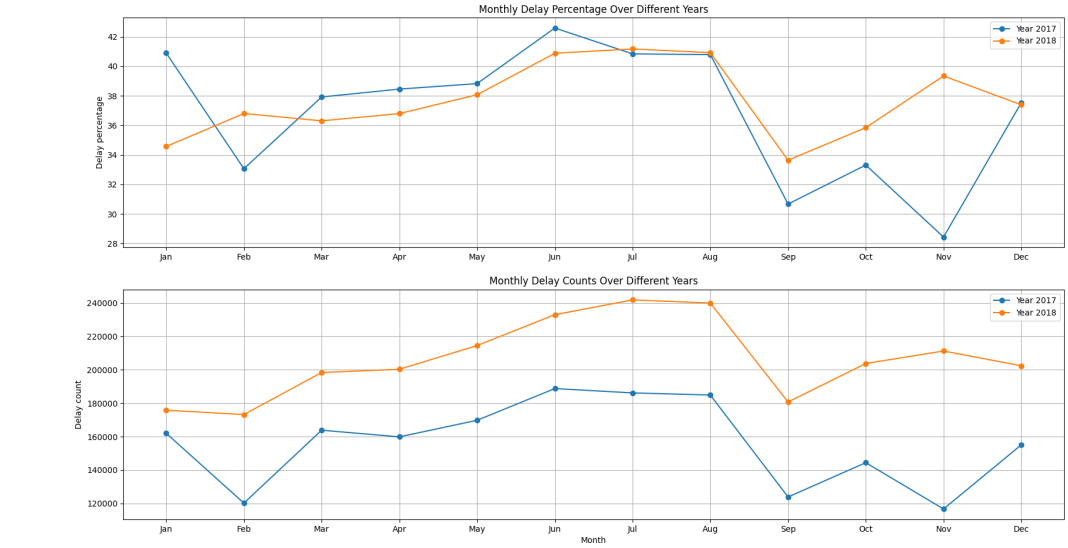
In order to normalize data, we will be using StandardScaler() class from sklearn.preprocessing library. It was vital to make our dataset have common scale in order to maybe speed up and (?)make models work better.

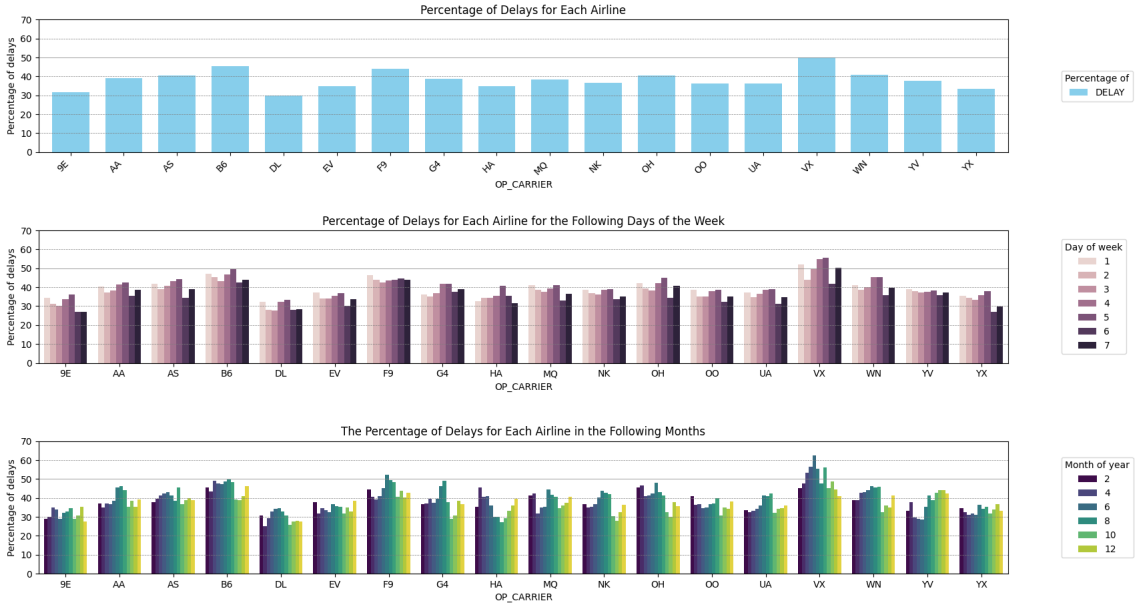
## 3.3 Mapping strings to integers

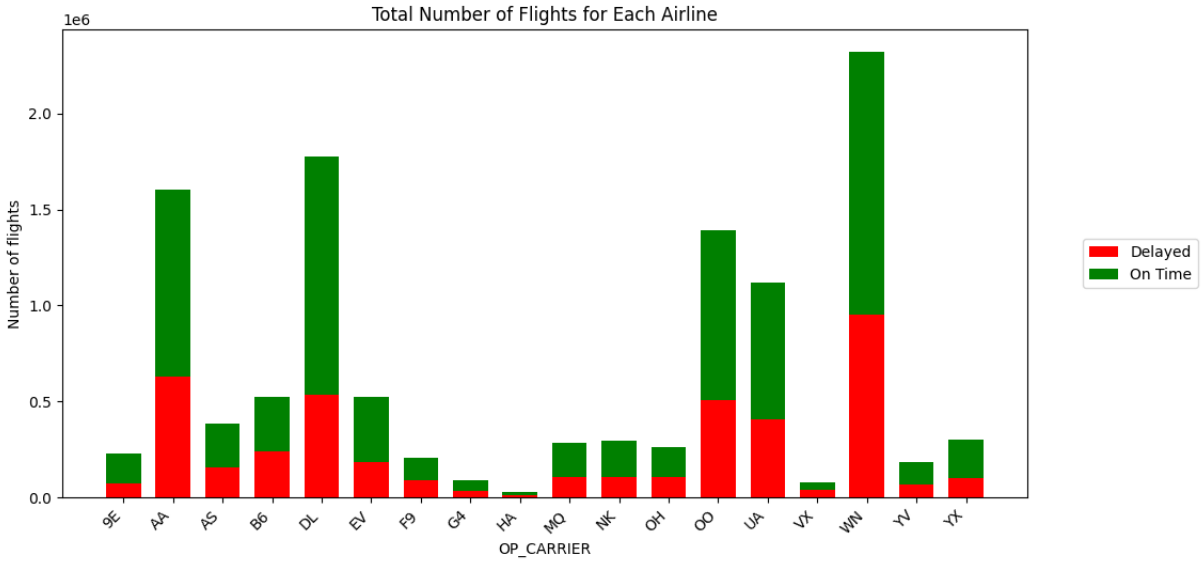
In machine learning we would like to represent strings as integers. We will do that, by mapping destination and origin cities to integers as json file. This file is in *Flight-Delay-Prediction/ml* folder, named mappings.json. Additionally, we will map some weather data. Under column conditions there are many string representations, but we will represent it as integers (also in file mappings.json).

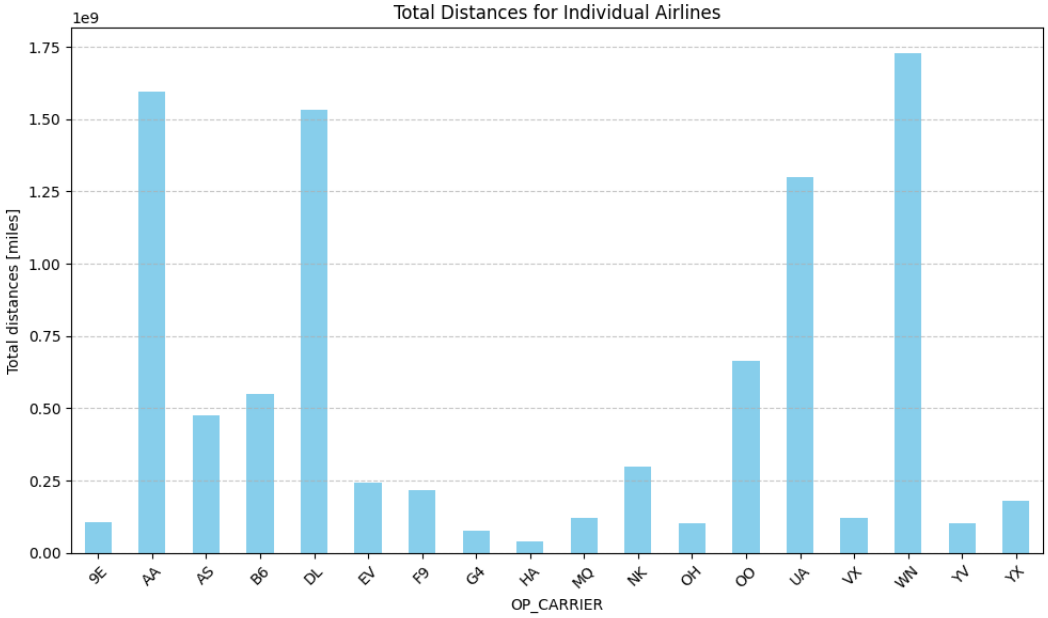
# 4 Explanatory Data Analysis (EDA) plots

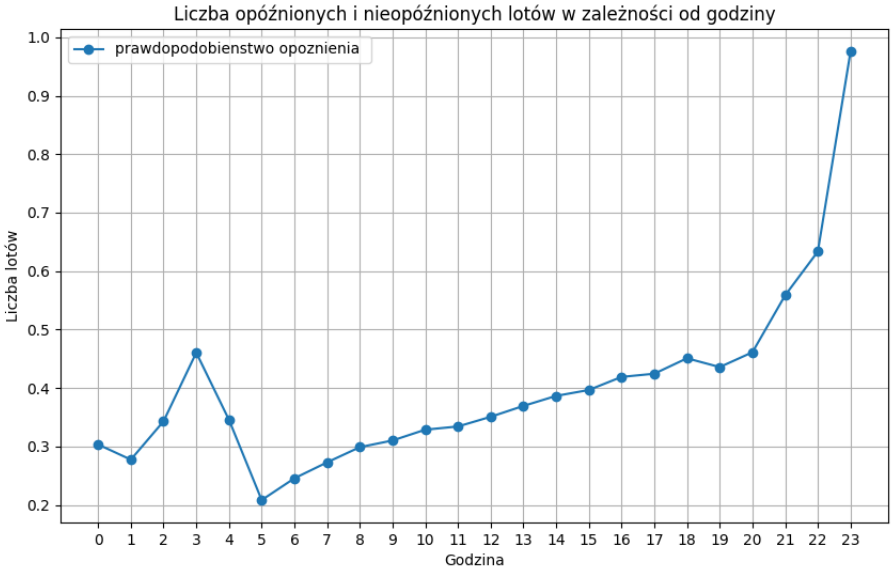
In this section we will look at plots describing data in our dataset.
Those plots are constructed in python' file *EDA_plottings.py*

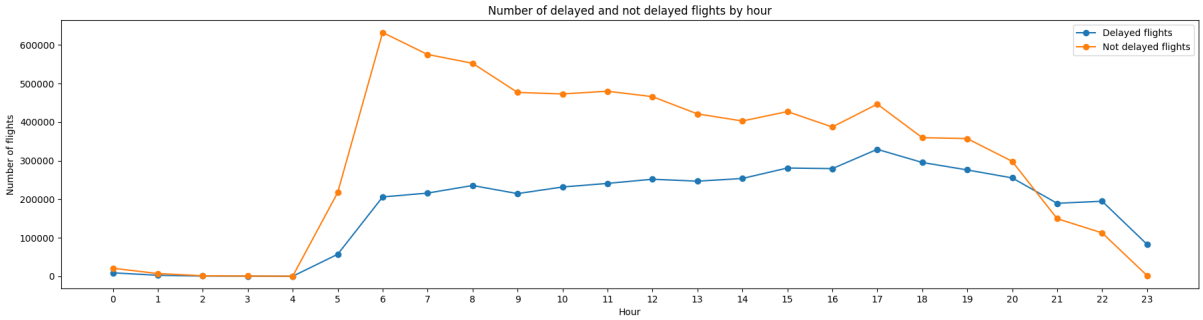


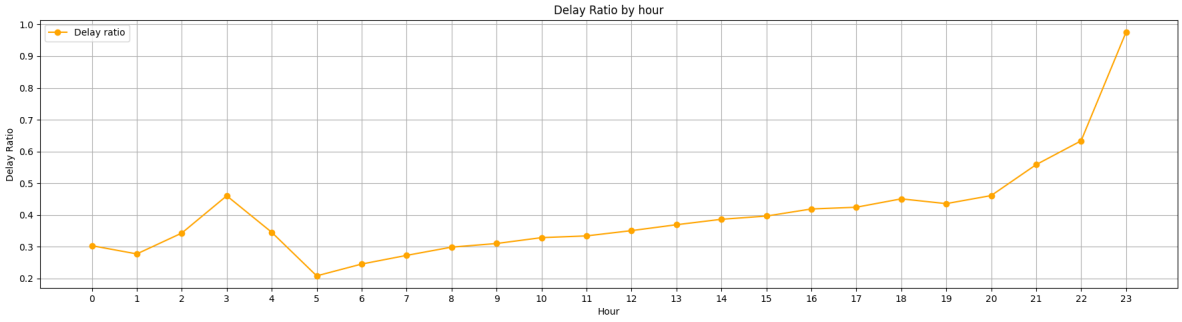




Total Number of Flights for Each Airline
1e6
Number of flights
0.0
0.5
1.0
1.5
2.0
9E
AA
AS
B6
DL
EV
F9
G4
HA
MQ
NK
OH
OO
UA
VX
WN
YV
YX
OP_CARRIER
Delayed
On Time


Total Distances for Individual Airlines
1e9
Total distances [miles]
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
9E
AA
AS
B6
DL
EV
F9
G4
HA
MQ
NK
OH
OO
UA
VX
WN
YV
YX
OP_CARRIER


Liczba opóźnionych i nieopóźnionych lotów w zależności od godziny
prawdopodobienstwo opoznienia
Liczba lotów
Godzina


Number of delayed and not delayed flights by hour
Delayed flights
Not delayed flights
Number of flights
Hour


Delay Ratio by hour
Delay ratio
Delay Ratio
Hour

# 5 Machine learning part

In this section we will consider machine learning algorithms. All of them are sklearn library based classes.

## 5.1 Decision tree

We ran Decision Trees with gini criterion from max_depth ranging from 4 to 20. From our experience, the best results we achieved with depth equal to 17. We achived 66% accuracy. It's the best accuracy we could score in our models.

## 5.2 Naive Bayes

Our model was trained on GaussianNB() classifier. Furthermore, we used GridSearchCV() class, which helped us to choose best var_smoothing parameter (portion of the largest variance of all features that is added to variances for calculation stability [from documentation]). It turned out that best accuracy was achieved with smoothing equal to $10^{-9}$ and returning 63% of accuracy. Testing lower smoothing values resulted in same accuracy.

## 5.3 Miscellaneous models

- ADABoostClassifier with SAMME algorithm and 100 estimators: 57%
- Random Forest Classifier with 10_000 min_leafs(minimum samples required to be a leaf): 64%
- SGD Classifier with best alpha equal to 0.0001 and max_iter equal to 3000 gives 62.5% accuracy

## 5.4 4fun models

In this section we considered models that were extremely bad or did not fall under machine learning algorithms category (neural networks).

### 5.4.1 Online model

Accuracy - around 33%. We tried to implement using divided datasets and train model by SGD classifier (stochastic gradient descent). It was bad.

### 5.4.2 Neural network

Accuracy - around 67%. It was kind of good result. I don't know what this code does. help

# 6 Conclusion

Even though we think 67% accuracy might be too low, project taught us numerous important things. First and foremost, data exploration, that is working with huge datasets and handling large api queries. Then we had to process data to work with it. Problems started when we started machine learning section. One thing that could be the cause of failure might be too much of data and lack of knowledge. Maybe we could have used time series? We could easily see the correlation between time and delay of flights.