Flight cancellation prediction depending on weather conditions day before flight using Machine Learning Algorithms.

Author: Anthony Epshtein
Subject: Neural Network course
GitHub Link: https://github.com/e4nthony/flight_cancelation_predict_usingANN

Overview:

The idea of this project is to use Machine Learning to predict is the flight going to be canceled depending on only on weather conditions one day before flight.

The final goal in theory is to be able feed weather data of specific day and expect from model to predict is it will be flying weather the next day.

We will determine the not flying weather as weather in which the plane unsafe fly up or down, at the starting point and final point. We will suppose that pilots can take longer routes to avoid bad weather midair.

Dataset format:

Final shape of dataset will contain columns of numeric weather parameters and one column that will contain binary target value saying whether the weather in that specific row leads to flight cancellations next day.

Therefore, after processing datasets we have multiple columns containing different weather parameters and it called features set. And one target column containing binary value determining whether happened cancellation of flight a day after these weather conditions at features set.

To learn more, explore the sections: “The Dataset sources” & “Preprocessing Datasets”.

Tools:

I decided to use Python 3.10 to implement the project,
Pandas library for data preprocessing as it is open source data analysis and manipulation tool,
and sklearn library as source of AI algorithms and tool for separation datasets to training sets and test sets.

Research: Which algorithms we use to resolve this problem and why:

I suppose that dataset I'm planning to prepare couldn't be separated linearly, the weather pattern of day before flight cancellation is not obvious at first glance.

To overcome the complexity of our data I decided to use Machine Learning approach to process data and make algorithm find similar patterns in dataset I have. Because our weather numeric data is hard to understand by naked eye. Therefore, we have to automatize the process of finding pattern so computer will do that for us. In non-machine learning algorithms have to manually define patterns. And considering our dataset complexity it is not an option.

I'm implementing supervised learning in other words all data columns are 'labeled'.
Thought computer can't understand tittles of columns but it capable of find pattern in weather parameters which leads to flight cancellation.

Lest have a look on one of the simplest Machine learning algorithms that use Supervised Learning technique, the **Linear Regression**. And examine why we can't use it.

We have binary type target variable as shown on Illustration 1 bellow as y variable:

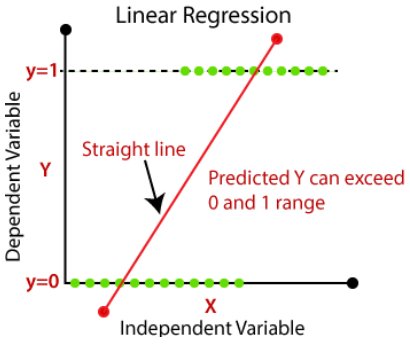


illustration 1 - Source[7]

In case of linear regression algorithm, it may produce inaccurate results because it built on assumption that relationship between the features and the target variable is linear, which is not true in our case.

It likely to predict some probabilities to be less than 0 or greater than 1, which are not meaningful in the case of binary classification as ours.

Methods like simple linear regression likely to fail to find the complex relationships present in the dataset. Therefore, we need more complicated algorithms that capable of separate the data non-linear way.

Logistic Regression Algorithm:

Next logical step to take is to examine **Logistic Regression Algorithm**.
Logistic Regression is a Supervised Learning algorithm used for binary classification tasks, where the target variable has binary values as 1 and 0 [8]. By description it seems to be the perfect solution to our problem.

According to description it seems to be specifically designed to work with Binary Classification problems such as ours.

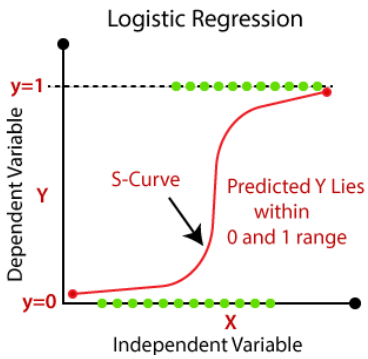


illustration 2 - Source[7]

It will use sigmoid function instead of linear one what will lead to outputs to be in boundaries of 1 and 0. And predictions will be in boundaries of true, false and something in between. As shown on Illustration 2.

In other words, it will provide predictions in range 0.0 to 1.0,
where for example value 0.85 will indicate that model 85% confident in prediction that it belongs to class 1 and there is 15% chance that it belongs to class 0. Therefore, one of its strengths is interpretability of its output, what means that it is easy for humans to understand the output.

Another advantage of algorithm that comes handy it is reliance on maximum likelihood estimation (MLE) as mentioned in source[8].
What means, that the algorithm can be 'tuned up' to prevent overfitting. Which is useful in our case because we are not checking the weather parameters that we get from source on the presentence of extreme values, as we cannot be sure where the limit of particular label lies.

Considering all advantages I decided to use it.

K-Nearest Neighbors Algorithm (KNN):

Among solutions using Machine Learning is hard to miss **K-Nearest Neighbors Algorithm**.
It is considered as effective approach for supervised learning problems,
while the concept of algorithm is relatively simple for humans to understand and that why it draw my attention.
I decided to compare its performance along with other solutions.

Among its advantages is that it is simple and can still make accurate predictions.
It has no actually training stage therefore we save lot of time on that stage comparing to other algorithms.
The training examples are vectors in a multidimensional feature space, each with a class label. [5]
The 'training' of KNN algorithm is actually comes down to just storing the feature vectors and labels of the training samples.

When algorithm need to determine how classify a new sample object, it looks for k nearest samples in the training set based on their labeled values. And depending on majority class among these nearest samples, the algorithm classifies the object as one of that particular class. That process called voting.

KNN capable of handling imbalanced datasets, it has characteristics that can help address class imbalance to some extent. Imbalanced datasets are these where one class is more prevalent than the other. It is a result of multiple features of that algorithm such as local decision making among k samples, what helps to handle imbalanced datasets. As in our case will be useful.
Because the vast majority of flights are accomplished successfully what means most of rows are containing target value 0 and relatively small number of weather samples of canceled flights (with target value 1).

There are multiple variants of it like KNN Classification, KNN Regression and more.

KNN Regression tries to predict the value of the output variable by using a local average.
And it does not address class imbalance issues, potentially leading to wrong predictions as they strive towards the majority class. We will not use it.

I decided to focus on KNN Classification because is it what we need.
We classify samples depending on their target values which is binary in out case.
We will use it.

Artificial Neural Network (ANN):

My initial impulse was to use ANN on this dataset because of the complexity of dataset I have.
But after research, I found better approaches that described above to find solutions which leads to slight better results. But out of curiosity and also research reasons I had to check how Artificial Neural Network will handle this task in comparison. We will compare its performance with other algorithms.

We will use multi-layer perceptron (MLP) as its core.

Among its advantages I can highlight few:
Its ability to find complex non-linear relationships that likely present in out dataset.
Also, as ANN robust to imbalanced datasets, as mentioned earlier that feature is important in our dataset.

I planning to compare results of different initial values of hidden layers.

Decision Trees Algorithm:

After research I decided to add to comparison Decision Trees Algorithm,
to check how it will handle the problem in comparison.
Because as mentioned in multiple sources of information, they suggest Decision Trees Algorithm to this kind of problem that we trying to solve.

If we try highlight its unique advantages we will mention that decision trees can handle missing values in the dataset.
Also it good at handling imbalanced datasets.

The Dataset sources:

To make predictions we need fit to each flight the weather info of previous day.

First, I decided to use prepared dataset that I found on internet but then I realized that data there isn't responding to my requirements. I wanted to be able to shift dates by one day what was impossible using prepared source. Also, other important columns that I needed was missing like info about destination of flight and more.

So decided to make my own dataset using original source.
I went to the source of flights data "Bureau of Transportation Statistics" [1] of United States.
And downloaded flights database with following parameters: FlightDate, Origin, Dest, CRSDepTime, CRSArrTime, Cancelled, CancellationCode. For period of 2023.01 - 2023.06 (6 months).

Next, I found database of worldwide weather data [3], where is possible to get historical weather records via API request for location and time period. It contains set of different parameters such as precipitation, temperature, wind, rain, snow etc. We will use in prediction model every parameter except irrelevant ones such as sun position, sunshine duration.

Also, I found airports dataset [4] where listed geolocation of each airport along with IATA codes.
It was necessary for connection between weather location and airports codes encoded in IATA formats.

Preprocessing Datasets:

I'm going to implement Supervised Machine Learning technique where the data is sorted in tables where each row represents the flight and for each flight the outcome is known (is flight canceled or not and why). And we going to focus only on one reason of cancellation, the weather.
It is presents as 'code b' in column of 'CancellationCode', it will be the target variable for the training our dataset.

It was important to decide how to feed data to out model and I thought that best way to do so is to represent one flight in two different rows. One row will contain data about origin of one flight and another the data about destination data of same flight.

This is because the flight might be cancelled as reason of destination bad weather or origin bad weather, which is two separate cases, and the program will examine them separately.

All the flights that canceled not because of weather are considered as accomplished flights.

After we process the data, we have only the weather parameters and one parameter from dataset 1 that says whether flight will be canceled. To do so we convert 'CancellationCode' to 'WeatherCancel' field that will contain '1' value if cancellation occurred as reason of bad weather and 0 if other reason affected cancellation or there wasn't cancellation at all. 'WeatherCancel' is the target value dataset in binary format.

I planned to use daily summary of weather the day before flight.
To do so we shift daily weather reports one day foward. That way the weather at date x-1 will point to flight at date x.

At this point I have two different datasets flights and weather and I had to merge them together.
I had idea to do that by using Geographic Coordinates (Latitude and Longitude) but they differ in both tables which leads to complicated solutions as finding most nearest coordinates and then merge them. Instead I chose to find dataset where presents IATA codes with their coordinates. That's why I needed source 4 to decode IATA codes.
After that I finally merged them using Latitude, Longitude and date.

I want from model to predict outcome based only on weather data, that means I don't need to feed 'noise data' to the model. Such as department time, arrival time… etc. So when we finally combined the datasets, now we trim all fields from source 1 except the target variable, the 'WeatherCancel'.

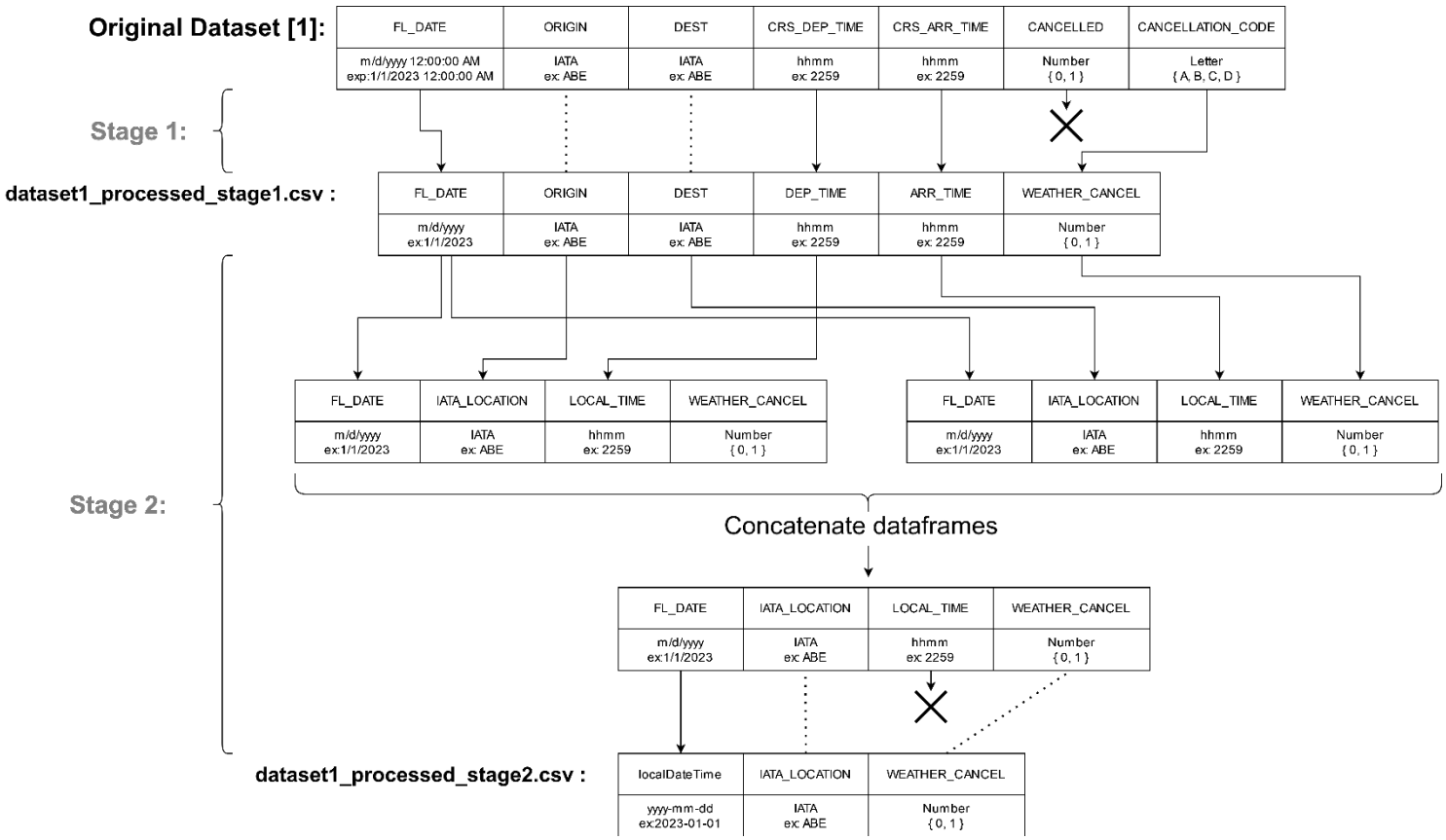
Now the data set was ready to be fed to model.

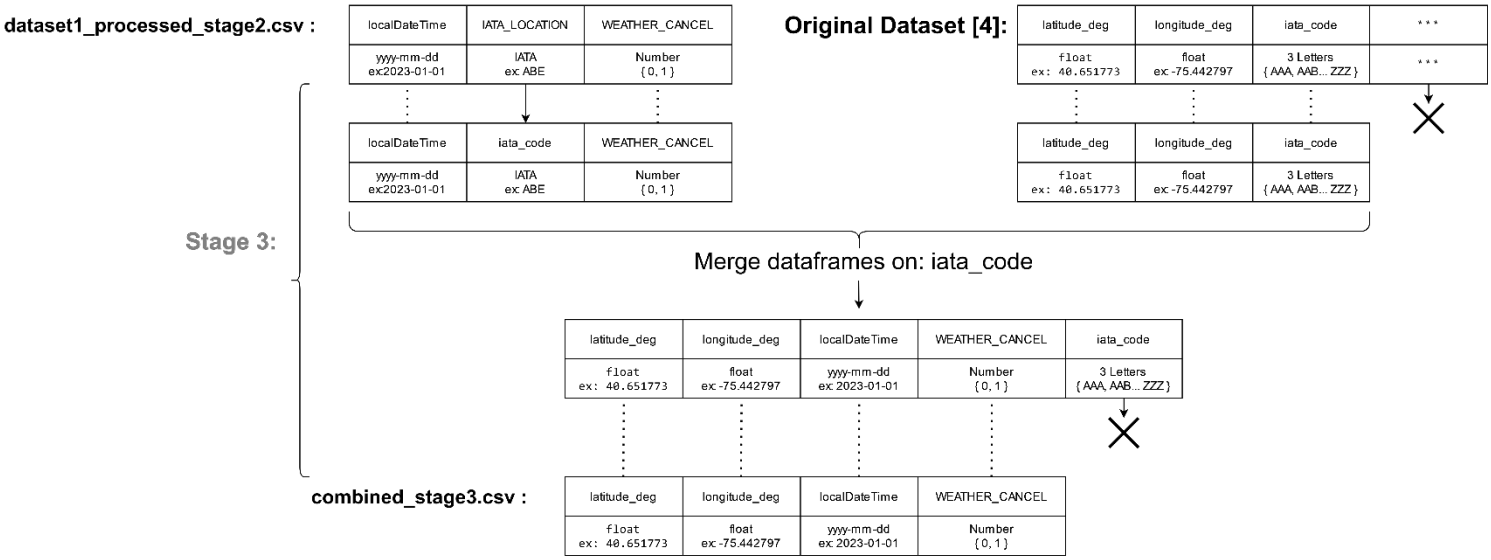
Note: We made our dataset to contain only numeric values which easier to algorithms to work with.

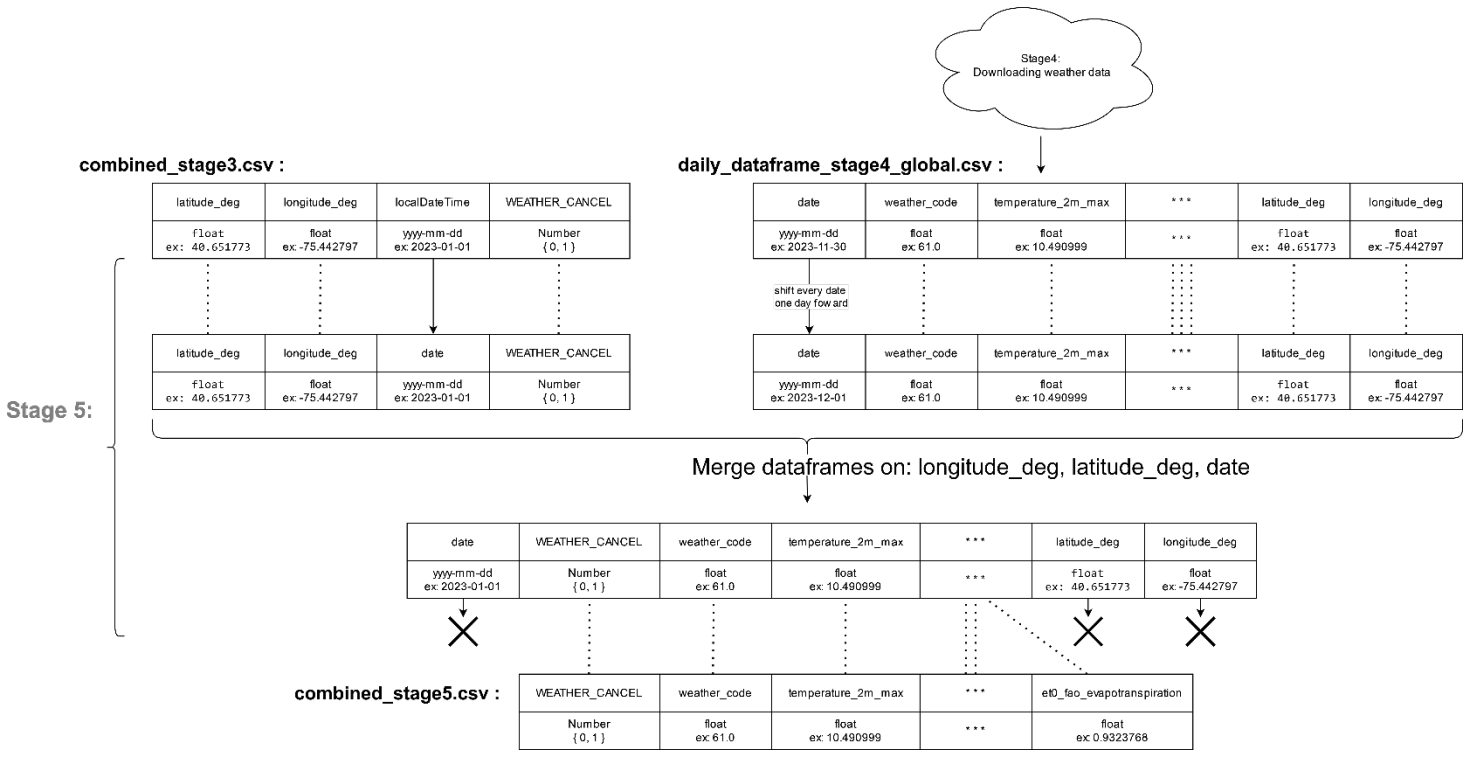
The data preprocessing part is implemented that way that allows to run it on new datasets in same format and it will automatically download necessary weather forecast using open-meteo.com API requests.
Each API request is asking for weather forecast of one airport at time for each day of period.
The period is set by global variable MONTHS, where listed months and years that we are processing and they are correspond with content of 'flight reports' directory .

For better understanding of data pre-processing process here is diagram:

This diagram made to visually represent the pre-processing steps that were performed on datasets:




dataset1_processed_stage2.csv :
localDateTime
IATA_LOCATION
WEATHER_CANCEL
yyyy-mm-dd ex:2023-01-01
IATA ex: ABE
Number {0, 1}
localDateTime
iata_code
WEATHER_CANCEL
yyyy-mm-dd ex:2023-01-01
IATA ex: ABE
Number {0, 1}
Original Dataset [4]:
latitude_deg
longitude_deg
iata_code
* * *
float ex: 40.651773
float ex: -75.442797
3 Letters { AAA, AAB... ZZZ }
* * *
latitude_deg
longitude_deg
iata_code
float ex: 40.651773
float ex: -75.442797
3 Letters { AAA, AAB... ZZZ }
Stage 3:
Merge dataframes on: iata_code
latitude_deg
longitude_deg
localDateTime
WEATHER_CANCEL
iata_code
float ex: 40.651773
float ex: -75.442797
yyyy-mm-dd ex: 2023-01-01
Number {0, 1}
3 Letters { AAA, AAB... ZZZ }
combined_stage3.csv :
latitude_deg
longitude_deg
localDateTime
WEATHER_CANCEL
float ex: 40.651773
float ex: -75.442797
yyyy-mm-dd ex: 2023-01-01
Number {0, 1}


Stage4: Downloading weather data
combined_stage3.csv :
daily_dataframe_stage4_global.csv :
latitude_deg
longitude_deg
localDateTime
WEATHER_CANCEL
float ex: 40.651773
float ex: -75.442797
yyyy-mm-dd ex: 2023-01-01
Number {0, 1}
latitude_deg
longitude_deg
date
WEATHER_CANCEL
float ex: 40.651773
float ex: -75.442797
yyyy-mm-dd ex: 2023-01-01
Number {0, 1}
date
weather_code
temperature_2m_max
* * *
latitude_deg
longitude_deg
yyyy-mm-dd ex: 2023-11-30
float ex: 61.0
float ex: 10.490999
* * *
float ex: 40.651773
float ex: -75.442797
shift every date one day foward
date
weather_code
temperature_2m_max
* * *
latitude_deg
longitude_deg
yyyy-mm-dd ex: 2023-12-01
float ex: 61.0
float ex: 10.490999
* * *
float ex: 40.651773
float ex: -75.442797
Stage 5:
Merge dataframes on: longitude_deg, latitude_deg, date
date
WEATHER_CANCEL
weather_code
temperature_2m_max
* * *
latitude_deg
longitude_deg
yyyy-mm-dd ex: 2023-01-01
Number {0, 1}
float ex: 61.0
float ex: 10.490999
* * *
float ex: 40.651773
float ex: -75.442797
combined_stage5.csv :
WEATHER_CANCEL
weather_code
temperature_2m_max
* * *
et0_fao_evapotranspiration
Number {0, 1}
float ex: 61.0
float ex: 10.490999
* * *
float ex: 0.9323768

Running Data Preprocessing before we can start AI Model:

Firstly we run our program with these global parameters:

```
MONTHS = ['2023-01', '2023-02', '2023-03', '2023-04', '2023-05', '2023-06']
REPROCESS_FLIGHT_DATA = True
REREQUEST_WEATHER_DATA = True
```

That will trigger to program to start processing datasets.
Every dataset in our case is a monthly report of flights, and they are listed in variable MONTHS accordingly to content of folder '..\\assets\\flight_reports' .
There is sub folders named accordingly in format YYYY-MM.
What means that there are 6 folders in our '..\\assets\\flight_reports' directory each containing csv file with reports of particular month.

Next these datasets are combined together to one big dataset (called df) and that dataset will be processed in stages 1-2 described earlier. This occurring in data_preprocessing.py file.

Next, program will access folder '..\\assets\\airports_data\\source4' and open dataset of artports from there.
It will be merged with global df dataset.
As result we are getting ‘combined_stage3.csv’ file as final version of dataframe on stage 3.

Next started stage 4 occurring in weather.py file, during which we sending multiple requests using api of open-meteo.com for each airport location.
As result, we get answer from server with daily weather summary for each day during period.
The period is calculated between first_day of our first month in variable MONTHS and next day after the last month in variable MONTHS.

It sends multiple requests as count of different airports that are present at our dataset.
There is no other option I found.
Approx. 350 requests, to not exceed max limit of requests per minute I set sleep(seconds=4) function. So program need time at this point.

After we getting all answers from we combine them as file ‘daily_dataframe_stage4_global.csv’ .

And we ready to start stage 5, where we combining weather (from stage 4) and dataframe (from stage 3) and trim the irellevant columns to stay only with weather parameters and target variable.
Program saving it ‘combined_stage5.csv’ file as last version of dataframe.

Now a new dataset ready for our AI Algorithms to process.

From now there is no need to preprocess data again as long as we don’t provide a new monthly flight reports.
So we will disable following global variables:

```
REPROCESS_FLIGHT_DATA = False
REREQUEST_WEATHER_DATA = False
```

Implementation of Logistic Regression Model*:*

Our processed dataset will use only numeric values what means the Logistic Regression Algorithm can efficiently use these values as is.

First we need import relevant modules from sklearn library:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

We are selecting features and target_variable by splitting the dataset by columns labels:

```
features = df.drop(columns=['WEATHER_CANCEL'])
target_variable = df['WEATHER_CANCEL']
```

Next, we need to split each of them once again to training sets and testing sets.
We will use train_test_split(..) to do so:

```
features_train_set, features_test_set, target_train_set, target_test_set =
train_test_split(features, target_variable, test_size=0.5, random_state=50)
```

test_size variable will define the percentage of dataset that will be saved as testing set for later.

We are using random_state variable to provide random number for splitting algorithm so splitting will be unpredictable and fair.

Next, we use LogisticRegression() function from sklearn to create Logistic Regression classifier object.

```
logistic_regression = LogisticRegression(max_iter=1000000)
```

We apply on that object fit() function which is actually 'training' our model using training examples.

```
logistic_regression.fit(features_train_set, target_train_set)
```

Next we run predict() function on the same object, what will return us a predictions set made by our model.

```
predictions = logistic_regression.predict(features_test_set)
```

now we just using accuracy_score() func to calculate accuracy of our algorithm and printing result

```
accuracy = accuracy_score(target_test_set, predictions)
print("Accuracy score:", accuracy)
print("Time [sec]:", time.time() - start_time)
```

Also I wanted to calculate the Mean Squared Error to see It on live example and compare to accuracy score.For this I needed

```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(target_test_set,
predictions)

print("Mean Squared Error:", mse)
```

The output example:

```
Started LogisticRegression_Model...
Accuracy score: 0.9910247892073691
Time [sec]: 144.70487141609192

Process finished with exit code 0
```

As we can see the accuracy of predictions is very promising.

Implementation of DecisionTrees Model:

First we need import relevant modules from sklearn library, this time we using KNeighborsClassifier by sklearn library:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

At first we need to separate dataset to features and target variable.

```
features = df.drop(columns=['WEATHER_CANCEL'])
target_variable = df['WEATHER_CANCEL']
```

Next, we need to split each of them once again to training sets and testing sets.

```
features_train_set, features_test_set, target_train_set, target_test_set =
train_test_split(features, target_variable, test_size=0.5, random_state=50)
```

train , predict and print:

```
decision_tree.fit(features_train_set, target_train_set)

predictions = decision_tree.predict(features_test_set)

accuracy = accuracy_score(target_test_set, predictions)
print("Accuracy:", accuracy)

from sklearn.metrics import mean_squared_error
mse = mean_squared_error(target_test_set, predictions)
print("Mean Squared Error:", mse)
print("Time [sec]:", time.time() - start_time)
```

The output example:

```
Started DecisionTrees_Model...
Accuracy: 0.9914810691930401
Mean Squared Error: 0.008518930806959878
Time [sec]: 102.66975355148315

Process finished with exit code 0
```

This is impressive result.

Implementation of K-Nearest Neighbors Model:

As we use k-nearest neighbors algorithm. We will feed it our pre-processed dataset,

First we need import relevant modules from sklearn library, this time we using KNeighborsClassifier by sklearn library:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

At first we need to separate dataset to features and target variable.
The target variable we know as column 'WeatherCancel' in dataset we built and features is weather data of particular date and location in other words all the columns that are not 'WeatherCancel'.

We are selecting features and target_variable by splitting the dataset by columns labels:

```
features = df.drop(columns=['WEATHER_CANCEL'])
target_variable = df['WEATHER_CANCEL']
```

Next, we need to split each of them once again to training sets and testing sets.
We will use train_test_split(..) to do so:

```
features_train_set, features_test_set, target_train_set, target_test_set =
train_test_split(features, target_variable, test_size=0.5, random_state=50)
```

test_size variable will define the percentage of dataset that will be saved as testing set for later.

We are using random_state variable to provide random number for splitting algorithm so splitting will be unpredictable and fair.

Next, we use KNeighborsClassifier() function from sklearn to create K-Neighbors Classifier object, n_neighbors variable is actually how many neighbors will be selected for voting around testing sample.

```
knn_classifier = KNeighborsClassifier(n_neighbors=10)
```

We apply on that object fit() function which is actually 'training' what just stores the features vectors and their target values (rows).

```
knn_classifier.fit(features_train_set, target_train_set)
```

Next we run predict() function on the same object, what will return us a predictions set made by our model.

```
predictions = knn_classifier.predict(features_test_set)
```

Now we using accuracy_score() func to calculate accuracy of our algorithm and printing result

```
accuracy = accuracy_score(target_test_set, predictions)
print("Accuracy score:", accuracy)
```

Also I wanted to calculate the Mean Squared Error to see It on live example and compare to accuracy score.
For this I needed

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(target_test_set, predictions)
print("Mean Squared Error:", mse)
```

Also I wanted to compare the time that each algorithm requires so I calculated it

The output example :

```
Started KNeighborsClassifier_Model... With k= 4
Training process started...
predict()
Accuracy score: 0.991241349253304
Mean Squared Error: 0.008758650746696031
Time [sec]: 451.98271894454956

Process finished with exit code 0
```

As we can see the accuracy of predictions of KNN is very promising too.

Implementation of ANN Model:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

We are going to use Multi-Layer Perceptron (MLP) as core to Artificial Neural Network.
For that we will need:

```
from sklearn.neural_network import MLPClassifier
```

Next, we are selecting features and target_variable by splitting the dataset by columns labels:

```
features = df.drop(columns=['WEATHER_CANCEL'])
target_variable = df['WEATHER_CANCEL']
```

Next, we need to split each of them once again to training sets and testing sets.
We will use train_test_split(..) to do so:

```
features_train_set, features_test_set, target_train_set, target_test_set =
train_test_split(features, target_variable, test_size=0.3, random_state=30)
```

We cannot directly say how many neurons be used in input layer or output layer, but we can choose the hidden layers numbers and count per layer:

```
# Create ANN using multi-layer perceptron (MLP)
ann_classifier = MLPClassifier(hidden_layer_sizes=neurons, max_iter=5000,
random_state=50)
```

We train model by using fit()

```
ann_classifier.fit(features_train_set, target_train_set)
```

Next we run predict() function on the same object, what will return us a predictions set made by our model.

```
predictions = ann_classifier.predict(features_test_set)
```

Now we using accuracy_score() func to calculate accuracy of our algorithm and printing result

```
accuracy = accuracy_score(target_test_set, predictions)
print("Accuracy score:", accuracy)
```

added mean_squared_error:

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(target_test_set, predictions)
print("Mean Squared Error:", mse)
```

and time:

```
print("Time [sec]:", time.time() - start_time)
```

example of output:

```
ANN_Model... With hiddden_layer_neurons:[5, 5, 5] (sizes in each hidden layer)
Accuracy score: 0.9911225428959288
Mean Squared Error: 0.008877457104071129
Time [sec]: 370.7881212234497

Process finished with exit code 0
```

Comparing results and Conclusion:

Result of running Logistic Regression:

```
Started LogisticRegression_Model...
Accuracy score: 0.9910247892073691
Time [sec]: 144.70487141609192

Process finished with exit code 0
```

Result of running DecisionTrees_Model:

```
Started DecisionTrees_Model...
Accuracy: 0.9914810691930401
Mean Squared Error: 0.008518930806959878
Time [sec]: 102.66975355148315

Process finished with exit code 0
```

Result of running KNN with k = 10 :

```
Started KNeighborsClassifier_Model... With k= 10
Training process started...
predict()
Accuracy score: 0.9913126336017575
Mean Squared Error: 0.008687366398242495
Time [sec]: 531.344128370285

Process finished with exit code 0
```

Result of running KNN with k = 4 :

```
Started KNeighborsClassifier_Model... With k= 4
Training process started...
predict()
Accuracy score: 0.991241349253304
Mean Squared Error: 0.008758650746696031
Time [sec]: 451.98271894454956

Process finished with exit code 0
```

As we can see the accuracy score indeed were lower with less neighbors at voting.

Result of running ANN with neurons = [5, 5, 5] :

```
ANN_Model... With hiddden_layer_neurons:[5, 5, 5] (sizes in each hidden layer)
Accuracy score: 0.9911225428959288
Mean Squared Error: 0.008877457104071129
Time [sec]: 370.7881212234497

Process finished with exit code 0
```

Result of running ANN with neurons = [5, 5] :

```
ANN_Model... With hiddden_layer_neurons:[5, 5] (sizes in each hidden layer)
Accuracy score: 0.9910859482622061
Mean Squared Error: 0.008914051737793935
Time [sec]: 187.87066960334778

Process finished with exit code 0
```

Result of running ANN with neurons = [20, 20] :

```
ANN_Model... With hiddden_layer_neurons:[20, 20] (sizes in each hidden layer)
Accuracy score: 0.9912047554977171
Mean Squared Error: 0.008795244502282903
Time [sec]: 387.197553396225

Process finished with exit code 0
```

Result of running ANN with neurons = [2, 2] :

```
ANN_Model... With hiddden_layer_neurons:[2, 2] (sizes in each hidden layer)
Accuracy score: 0.9908919465738399
Mean Squared Error: 0.00910805342616005
Time [sec]: 205.53613209724426

Process finished with exit code 0
```

As we can see the accuracy is dropped as we removed one hidden layer,
also we can see that it is increased when we added 15 neurons to each hidden layer.
As conclusion we can see that despite more neurons at model [20, 20] the model [5, 5, 5] got to better results even thought it have less neurons in total,
and from that we learn about importance of third hidden is obvious and as we add layer we add more steps in processing each sample what leads to more accurate results.

(Additionally I checked [2, 2], and we see that [2, 2] took more time than [5, 5] .)

But in comparison to other algorithms ANN based on multi-layer perceptron (MLP) getting to less impressive accuracy.

Winners:

We see that out of limited iterations we performed winner is DecisionTrees algorithm !
Both by time and accuracy what wasn't obvious at first for me.

The second most fastest were: Logistic Regression and ANN with neurons = [5, 5] .
Ther accuracy wasn't best but they provided answer at fastest time

The second most accuracy was: KNN with k = 10.
It wasn't fastest but provided second best accuracy.

Limitations:

- To keep things simple we have to not account mid flight weather because I have no access to global weather dataset, only to specific locations.
  Also I have limited number of daily API requests to weather dataset.

- Another limitation: I must request daily weather data rather than hourly because total dataset will be over 3 million rows and that will be difficult to computer to process in short period of time.

- When we separate cancelled flight record into two separate cases of origin and destination, we get one of the new rows potentially have wrong target value. For example, if flight was cancelled because of origin bad weather (we saved it in first row), there is potentially good weather at destination point (we saved it in second row) with value 1, what means this actually good weather leads to flight cancellation too.
  Right now, it is our noise data that we can't get rid of.
  The model will be better if we could clean up these wrong rows, but I don't have solution to that problem because there is not enough data in source to do so.

The problems I faced:

1. There are multiple different identificators to same airports.

   The IATA (International Air Transport Association) codes represented by 3 letters.
   The ICAO (International Civil Aviation Organization) codes represented by 3-4 alphabetic characters.
   Also there is LID "local identifier" means its local to the country in which they are assigned.

   The US seems to be using mix of identification systems for airports, so they have an identification number assigned by US DOT" which can be same as IATA or ICAO and sometimes this make confusion when type isn't specified.

   I had to suppose that OIGIN and DEST fields at dataset is using IATA standard (isn't specified in documentation). Also, its more standardized and more world recognizable.

2. Additionally, there is confusion with time zones. They change according to destination country by arrival and be better if they considered in computation too. But the weather source is not

sending the time zones of locations when requesting daily weather, which makes difficult to process hours and compute them accordingly to. I didn't find solution to build dataset hourly yet as reason of multiple difficulties.

3. I decided to include the geolocation coordinates for every airport as well.
   This required to find additional data source [4] to correspond with weather station locations.

4. I struggled to find a complete set of worldwide weather historical data that will be free.
   but I have found API that can return the data by location and date, so I used it.

Datasets Sources:

[1] Bureau of Transportation Statistics "Reporting Carrier On-Time Performance"
https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr

[2] ~~Location of Airports Worldwide~~. (It uses ICAO formats, failed to connect to other datasets.)
~~https://catalog.data.gov/dataset/airports-5e97a~~

[3] Historical Weather Records API
https://open-meteo.com/en/docs/historical-weather-api#

[4] Location of Airports Worldwide.
https://ourairports.com/data/

Information Sources:

[5] K-Nearest Neighbors Algorithm
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[6] Artificial Neural Network
https://en.wikipedia.org/wiki/Neural_network_(machine_learning)

[7] Comparison of Linear Regression vs Logistic Regression
https://www.javatpoint.com/linear-regression-vs-logistic-regression-in-machine-learning

[8] Logistic Regression
https://en.wikipedia.org/wiki/Logistic_regression

[9] Decision Tree Learning
https://en.wikipedia.org/wiki/Decision_tree_learning