Flight Delay Project

Description

In this notebook we will prepare airline and weather data in order to eventually create a model that can flag potential flights for a higher chance of being delayed. A flight will be considered delayed if it is more than 8 minutes past its expected arrival time.

Import Modules

- You will first need to install a number of modules in order to follow along with this notebook.
- Most of these packages, such as numpy and pandas, are available using Anaconda (https://conda.io/docs/user-guide/install/index.html).
- For the machine learning pipeline, we will be making use of the BigML Python bindings (https://bigml.readthedocs.io/en/latest/).

In [1092]:
```
import pandas as pd
import numpy as np

#This option lets us view all columns of our dataset
pd.set_option('display.max_columns', 500)

#We will ignore warning flags in the code
import warnings
warnings.filterwarnings('ignore')
```

In [1093]:
```
df = pd.read_csv("data/airline_data.csv")
```

We will remove columns which are not used in analysis and delete the categorized delay variables because many instances do not contain the data.

In [1094]:
```
df.drop(['ORIGIN_STATE_ABR','ORIGIN_AIRPORT_ID','DEST_AIRPORT_ID','DEST_STA
#Delete the categorized delay variables because less than 10% records have
df.drop(['Unnamed: 27','CARRIER_DELAY','WEATHER_DELAY','NAS_DELAY','SECURIT
```

We will drop the rows with missing values in important columns, such as departure times.

In [1095]:
```
#Drop rows with missing data in the important columns, i.e. the predictors
total_data_rows = len(df.index)
#Drop NaNs
df.dropna(subset = ['UNIQUE_CARRIER','ORIGIN','DEST','CRS_DEP_TIME','CRS_AR
data_retained = len(df.index)/total_data_rows
print('Data Retained: '+str(round(data_retained*100,2))+' %')
```

```
Data Retained: 98.93 %
```

In [1096]:
```
df.head(2)
```

Out[1096]:

	YEAR	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	FL_NUM	ORIGIN	DEST
0	2013	9	2	1	9E	3283	AUS	SLC
1	2013	9	3	2	9E	3283	BUF	JFK

We will filter on the top 50 airports (sorted by air traffic), contained in a given CSV file.

In [1097]:
```
top50_airport = pd.read_csv('data/top50airports.csv')['IATA'].tolist()
df = df[df['ORIGIN'].isin(top50_airport)]
df = df[df['DEST'].isin(top50_airport)]
```

In [ ]:

We will now join our weather data that was extracted from [www.ncdc.noaa.gov.in (http://www.ncdc.noaa.gov.in)](http://www.ncdc.noaa.gov.in)

In [1098]:
```
df_weather = pd.read_csv("data/860638.csv")
df_weather = df_weather.append(pd.read_csv('data/860640.csv'))
```

In [1099]:
```
df_weather.head(5)
```

Out[1099]:

	STATION	STATION_NAME	ELEVATION	LATITUDE	LONGITUDE	DATE	REPORTTPYE	HOURL
0	WBAN:12918	HOUSTON WILLIAM P HOBBY AIRPORT TX US	13.4	29.63806	-95.28194	2013-09-01 00:53	FM-15	
1	WBAN:12918	HOUSTON WILLIAM P HOBBY AIRPORT TX US	13.4	29.63806	-95.28194	2013-09-01 01:53	FM-15	
2	WBAN:12918	HOUSTON WILLIAM P HOBBY AIRPORT TX US	13.4	29.63806	-95.28194	2013-09-01 02:53	FM-15	
3	WBAN:12918	HOUSTON WILLIAM P HOBBY AIRPORT TX US	13.4	29.63806	-95.28194	2013-09-01 03:53	FM-15	
4	WBAN:12918	HOUSTON WILLIAM P HOBBY AIRPORT TX US	13.4	29.63806	-95.28194	2013-09-01 04:53	FM-15	

We will filter for fields that pertain to our project.

In [1100]:
```
#Select the weather parameters which affect flight status: Visibility, Temp
df_weather = df_weather[['STATION_NAME','DATE','HOURLYVISIBILITY','HOURLYDR
```

In [1101]:
```
df_weather.head(5)
```

Out[1101]:

	STATION_NAME	DATE	HOURLYVISIBILITY	HOURLYDRYBULBTEMPC	HOURLYWindSpeed	HOUR
0	HOUSTON WILLIAM P HOBBY AIRPORT TX US	2013-09-01 00:53	10.00	25	3	
1	HOUSTON WILLIAM P HOBBY AIRPORT TX US	2013-09-01 01:53	10.00	25	6	
2	HOUSTON WILLIAM P HOBBY AIRPORT TX US	2013-09-01 02:53	10.00	24.4	3	
3	HOUSTON WILLIAM P HOBBY AIRPORT TX US	2013-09-01 03:53	10.00	23.9	3	
4	HOUSTON WILLIAM P HOBBY AIRPORT TX US	2013-09-01 04:53	10.00	23.9	0	

Preparing the Weather dataset:

- Replace Long format station name with IATA codes #Need to fix an automated way to do this
- Fix incorrect and erroneous data, such as characters in temperature fields, etc
- Convert Timestamp into YEAR, MONTH, DAY_OF_MONTH and HOUR
- Remove duplicates from the dataset, i.e. multiple entries from same STATION for same HOUR on a particular Date
- Replace NaN with 0 in HOURLY_PRECIP
- Replace NaN with Mean Visibility in HOURLYVISIBILITY

We will first replace the long format names to abbreviations.

In [1102]:
```
#Replacing Long Format Station Names with IATA Airport codes
df_weather['STATION_NAME'].replace('ATLANTA HARTSFIELD INTERNATIONAL AIRPOR
df_weather['STATION_NAME'].replace('CHICAGO OHARE INTERNATIONAL AIRPORT IL
df_weather['STATION_NAME'].replace('DAL FTW WSCMO AIRPORT TX US','DFW',inpl
df_weather['STATION_NAME'].replace('DENVER INTERNATIONAL AIRPORT CO US','DE
df_weather['STATION_NAME'].replace('LOS ANGELES INTERNATIONAL AIRPORT CA US
df_weather['STATION_NAME'].replace('SAN FRANCISCO INTERNATIONAL AIRPORT CA
df_weather['STATION_NAME'].replace('PHOENIX SKY HARBOR INTERNATIONAL AIRPOR
df_weather['STATION_NAME'].replace('HOUSTON INTERCONTINENTAL AIRPORT TX US'
df_weather['STATION_NAME'].replace('LAS VEGAS MCCARRAN INTERNATIONAL AIRPOR
df_weather['STATION_NAME'].replace('MINNEAPOLIS ST PAUL INTERNATIONAL AIRPO
df_weather['STATION_NAME'].replace('DETROIT METROPOLITAN AIRPORT MI US','DT
df_weather['STATION_NAME'].replace('SEATTLE TACOMA INTERNATIONAL AIRPORT WA
df_weather['STATION_NAME'].replace('ORLANDO INTERNATIONAL AIRPORT FL US','M
df_weather['STATION_NAME'].replace('BOSTON MA US','BOS',inplace=True)
df_weather['STATION_NAME'].replace('CHARLOTTE DOUGLAS AIRPORT NC US','CLT',
df_weather['STATION_NAME'].replace('NEWARK LIBERTY INTERNATIONAL AIRPORT NJ
df_weather['STATION_NAME'].replace('SALT LAKE CITY INTERNATIONAL AIRPORT UT
df_weather['STATION_NAME'].replace('LA GUARDIA AIRPORT NY US','LGA',inplace
df_weather['STATION_NAME'].replace('JFK INTERNATIONAL AIRPORT NY US','JFK',
df_weather['STATION_NAME'].replace('BALTIMORE WASHINGTON INTERNATIONAL AIRP
df_weather['STATION_NAME'].replace('CHICAGO MIDWAY AIRPORT IL US','MDW',inp
df_weather['STATION_NAME'].replace('MIAMI INTERNATIONAL AIRPORT FL US','MIA
df_weather['STATION_NAME'].replace('SAN DIEGO INTERNATIONAL AIRPORT CA US',
df_weather['STATION_NAME'].replace('WASHINGTON REAGAN NATIONAL AIRPORT VA U
df_weather['STATION_NAME'].replace('FORT LAUDERDALE HOLLYWOOD INTERNATIONAL
df_weather['STATION_NAME'].replace('PHILADELPHIA INTERNATIONAL AIRPORT PA U
df_weather['STATION_NAME'].replace('TAMPA INTERNATIONAL AIRPORT FL US','TPA
df_weather['STATION_NAME'].replace('DALLAS FAA AIRPORT TX US','DAL',inplace
df_weather['STATION_NAME'].replace('HOUSTON WILLIAM P HOBBY AIRPORT TX US',
df_weather['STATION_NAME'].replace('PORTLAND INTERNATIONAL AIRPORT OR US','
df_weather['STATION_NAME'].replace('NASHVILLE INTERNATIONAL AIRPORT TN US',
df_weather['STATION_NAME'].replace('ST LOUIS LAMBERT INTERNATIONAL AIRPORT
df_weather['STATION_NAME'].replace('WASHINGTON DULLES INTERNATIONAL AIRPORT
df_weather['STATION_NAME'].replace('HONOLULU INTERNATIONAL AIRPORT HI US','
df_weather['STATION_NAME'].replace('OAKLAND METROPOLITAN INTERNATIONAL AIRP
df_weather['STATION_NAME'].replace('AUSTIN BERGSTROM INTERNATIONAL AIRPORT
df_weather['STATION_NAME'].replace('KANSAS CITY INTERNATIONAL AIRPORT MO US
df_weather['STATION_NAME'].replace('NEW ORLEANS INTERNATIONAL AIRPORT LA US
df_weather['STATION_NAME'].replace('SAN JOSE CA US','SJC',inplace=True)
df_weather['STATION_NAME'].replace('SACRAMENTO METROPOLITAN AIRPORT CA US',
df_weather['STATION_NAME'].replace('SANTA ANA JOHN WAYNE AIRPORT CA US','SN
df_weather['STATION_NAME'].replace('CLEVELAND HOPKINS INTERNATIONAL AIRPORT
df_weather['STATION_NAME'].replace('RALEIGH AIRPORT NC US','RDU',inplace=Tr
df_weather['STATION_NAME'].replace('MILWAUKEE MITCHELL INTERNATIONAL AIRPOR
df_weather['STATION_NAME'].replace('SAN ANTONIO INTERNATIONAL AIRPORT TX US
df_weather['STATION_NAME'].replace('INDIANAPOLIS INTERNATIONAL AIRPORT IN U
df_weather['STATION_NAME'].replace('FORT MYERS SW FLORIDA REGIONAL AIRPORT
df_weather['STATION_NAME'].replace('PITTSBURGH ASOS PA US','PIT',inplace=Tr
df_weather['STATION_NAME'].replace('SAN JUAN L M MARIN INTERNATIONAL AIRPOR
df_weather['STATION_NAME'].replace('PORT COLUMBUS INTERNATIONAL AIRPORT OH
```

This function will be to fix incorrect data, such as characters in temperature fields.

In [1103]:
```
def convert(x):
        try:
            if str(x)[-1].isalpha():
                return(float(str(x)[:-1]))
            else:
                return(float(str(x)))
        except:
            return(np.nan)
```

In [1104]:
```
df_weather['HOURLYVISIBILITY'] = df_weather['HOURLYVISIBILITY'].apply(lambd
df_weather['HOURLYDRYBULBTEMPC'] = df_weather['HOURLYDRYBULBTEMPC'].apply(l
df_weather['HOURLYWindSpeed'] = df_weather['HOURLYWindSpeed'].apply(lambda
df_weather['HOURLYPrecip'] = df_weather['HOURLYPrecip'].apply(lambda x: con
```

We will extract the year, month, date, and hour from our weather dataframe.

In [1105]:
```
df_weather['DATE'] = pd.to_datetime(df_weather['DATE'])

df_weather['YEAR']= df_weather['DATE'].apply(lambda time: time.year)
df_weather['MONTH']= df_weather['DATE'].apply(lambda time: time.month)
df_weather['DAY_OF_MONTH']= df_weather['DATE'].apply(lambda time: time.day)
df_weather['HOUR']= df_weather['DATE'].apply(lambda time: time.hour)
```

We will remove duplicates in our dataframe

In [1106]:
```
df_weather.drop_duplicates(['STATION_NAME','YEAR','MONTH','DAY_OF_MONTH','H
df_weather.drop('DATE',axis = 1,inplace=True)
```

We will replace NA values with 0 for hourly precipitation and replace NA values with the average in hourly visibility.

In [1107]:
```
df_weather['HOURLYPrecip'].fillna(value=0,inplace=True)

df_weather['HOURLYVISIBILITY'].fillna(df_weather['HOURLYVISIBILITY'].mean()
```

In [1108]:
```
df_weather.head(5)
```

Out[1108]:

	STATION_NAME	HOURLYVISIBILITY	HOURLYDRYBULBTEMPC	HOURLYWindSpeed	HOURLYPreci
0	HOU	10.0	25.0	3.0	0.
1	HOU	10.0	25.0	6.0	0.
2	HOU	10.0	24.4	3.0	0.
3	HOU	10.0	23.9	3.0	0.
4	HOU	10.0	23.9	0.0	0.

We will check for missing values

In [1109]:
```
df_weather.isnull().sum()
```

Out[1109]:
```
STATION_NAME          0
HOURLYVISIBILITY      0
HOURLYDRYBULBTEMPC    8
HOURLYWindSpeed      13
HOURLYPrecip          0
YEAR                  0
MONTH                 0
DAY_OF_MONTH          0
HOUR                  0
dtype: int64
```

Next, we will calculate the average weather values for each station, for example, the annual mean temperature. We will also create two dataframes, one for origin and one for the destination.

In [1110]:
```
df_avg_DEP = df_weather.groupby('STATION_NAME').mean()
df_avg_DEP.drop(['YEAR','MONTH','DAY_OF_MONTH','HOUR'],axis = 1,inplace=Tru
df_avg_DEP.reset_index(drop=False,inplace=True)
df_avg_DEP.rename(index=str, columns={"STATION_NAME": "ORIGIN"},inplace=Tru
df_avg_DEP.rename(index=str, columns={"HOURLYVISIBILITY": "DEP_AVG_HOURLYVI
df_avg_DEP.rename(index=str, columns={"HOURLYDRYBULBTEMPC": "DEP_AVG_HOURLY
df_avg_DEP.rename(index=str, columns={"HOURLYWindSpeed": "DEP_AVG_HOURLYWin
df_avg_DEP.rename(index=str, columns={"HOURLYPrecip": "DEP_AVG_HOURLYPrecip

df_avg_ARR = df_weather.groupby('STATION_NAME').mean()
df_avg_ARR.drop(['YEAR','MONTH','DAY_OF_MONTH','HOUR'],axis = 1,inplace=Tru
df_avg_ARR.reset_index(drop=False,inplace=True)
df_avg_ARR.rename(index=str, columns={"STATION_NAME": "DEST"},inplace=True)
df_avg_ARR.rename(index=str, columns={"HOURLYVISIBILITY": "ARR_AVG_HOURLYVI
df_avg_ARR.rename(index=str, columns={"HOURLYDRYBULBTEMPC": "ARR_AVG_HOURLY
df_avg_ARR.rename(index=str, columns={"HOURLYWindSpeed": "ARR_AVG_HOURLYWin
df_avg_ARR.rename(index=str, columns={"HOURLYPrecip": "ARR_AVG_HOURLYPrecip
```

In [1111]:
```
df_weather_origin = df_weather.copy()
df_weather_dest = df_weather.copy()
del df_weather

#Rename the Columns, add DEP_ to each column name and STATION_NAME to ORIGI
df_weather_origin.rename(index=str, columns={"STATION_NAME": "ORIGIN"},inpl
df_weather_origin.rename(index=str, columns={"HOURLYVISIBILITY": "DEP_HOURL
df_weather_origin.rename(index=str, columns={"HOURLYDRYBULBTEMPC": "DEP_HOU
df_weather_origin.rename(index=str, columns={"HOURLYWindSpeed": "DEP_HOURLY
df_weather_origin.rename(index=str, columns={"HOURLYPrecip": "DEP_HOURLYPre
df_weather_origin.rename(index=str, columns={"HOUR": "DEP_HOUR"},inplace=Tr

#Rename the Columns, add ARR_ to each column name and STATION_NAME to DEST
df_weather_dest.rename(index=str, columns={"STATION_NAME": "DEST"},inplace=
df_weather_dest.rename(index=str, columns={"HOURLYVISIBILITY": "ARR_HOURLYV
df_weather_dest.rename(index=str, columns={"HOURLYDRYBULBTEMPC": "ARR_HOURL
df_weather_dest.rename(index=str, columns={"HOURLYWindSpeed": "ARR_HOURLYWi
df_weather_dest.rename(index=str, columns={"HOURLYPrecip": "ARR_HOURLYPreci
df_weather_dest.rename(index=str, columns={"HOUR": "ARR_HOUR"},inplace=True
```

We will create four new columns from the time columns, for departure and arrival of actual hours and computer reservation hours (CRS).

In [1112]:
```
df["DEP_HOUR"] = df["DEP_TIME"].apply(lambda x:x//100)
df =df[np.isfinite(df['DEP_HOUR'])]
df["DEP_HOUR"]=df["DEP_HOUR"].astype('int64')

df["ARR_HOUR"] =  df["ARR_TIME"].apply(lambda x: x//100)
df =df[np.isfinite(df["ARR_HOUR"])]
df["ARR_HOUR"]=df["ARR_HOUR"].astype('int64')

df["CRS_DEP_HOUR"] = df["CRS_DEP_TIME"].apply(lambda x:x//100)
df =df[np.isfinite(df['CRS_DEP_HOUR'])]
df["CRS_DEP_HOUR"]=df["CRS_DEP_HOUR"].astype('int64')

df["CRS_ARR_HOUR"] =  df["CRS_ARR_TIME"].apply(lambda x: x//100)
df =df[np.isfinite(df["CRS_ARR_HOUR"])]
df["CRS_ARR_HOUR"]=df["CRS_ARR_HOUR"].astype('int64')
```

In [1113]:
```
print(df_weather_origin.shape)
print(df.shape)
```

```
(99351, 9)
(299011, 22)
```

We can see that our weather location is only linked to 4 cities, so we will filter our airline dataframe for only the labeled sities.

In [1114]:
```
df_weather_origin.groupby("ORIGIN").count()
```

Out[1114]:

ORIGIN	DEP_HOURLYVISIBILITY	DEP_HOURLYDRYBULBTEMPC	DEP_HOURLYWindSpeed	DEP_HOU
BNA	20442	20442	20441	
DAL	26303	26303	26299	
HOU	26304	26303	26301	
STL	26302	26295	26297	

In [1115]:
```
df=df[df["ORIGIN"].apply(lambda x: x in ["BNA","DAL","HOU","STL"])]
df=df[df["DEST"].apply(lambda x: x in ["BNA","DAL","HOU","STL"])]
```

In [1116]:
```
print(df["YEAR"].dtype,df["YEAR"].dtype,df["DAY_OF_MONTH"].dtype,df["DEP_HO
print(df_weather_origin["YEAR"].dtype,df_weather_origin["YEAR"].dtype,df_we
```

```
int64 int64 int64 int64
int64 int64 int64 int64
```

We will join the weather for origin and destination Airports for each flight in the data frame

In [1117]:
```
df= pd.merge(df, df_weather_origin, on=['ORIGIN','YEAR','MONTH','DAY_OF_MON
df = pd.merge(df, df_weather_dest, on=['DEST','YEAR','MONTH','DAY_OF_MONTH'
```

In [1118]:
```
df.head(5)
```

Out[1118]:

	YEAR	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	FL_NUM	ORIGIN	DEST
0	2013	9	1	7	WN	60	BNA	HOU
1	2013	9	1	7	WN	64	BNA	HOU
2	2013	9	1	7	WN	912	BNA	HOU
3	2013	9	1	7	WN	54	BNA	STL
4	2013	9	1	7	WN	19	DAL	HOU

In [1119]:
```
df.shape
```

Out[1119]: (2354, 30)

We will also join the average for origin and destination Airports for each flight in the data frame

In [1120]:
```
df = pd.merge(df,df_avg_DEP,how='left',on='ORIGIN')
df = pd.merge(df,df_avg_ARR,how='left',on='DEST')
```

In [1121]:
```
df.shape
```

Out[1121]:
```
(2354, 38)
```

In [1122]:
```
df.head(5)
```

Out[1122]:

	YEAR	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	FL_NUM	ORIGIN	DEST
0	2013	9	1	7	WN	60	BNA	HOU
1	2013	9	1	7	WN	64	BNA	HOU
2	2013	9	1	7	WN	912	BNA	HOU
3	2013	9	1	7	WN	54	BNA	STL
4	2013	9	1	7	WN	19	DAL	HOU

In [1123]:
```
df.head(2)
```

Out[1123]:

	YEAR	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	FL_NUM	ORIGIN	DEST
0	2013	9	1	7	WN	60	BNA	HOU
1	2013	9	1	7	WN	64	BNA	HOU

In our new data frame, we will create a target column that will label any flight as True that has passed 8 minutes on their expected arrival time.

In [1055]:
```
df["ARRIVIAL_DELAYED"] =  df["ARR_DELAY"].apply(lambda x: "YES" if x > 8 el
```

In [1056]:
```
df["ARRIVIAL_DELAYED"].value_counts()
```

Out[1056]:
```
NO      1694
YES      660
Name: ARRIVIAL_DELAYED, dtype: int64
```

We will then drop the ARR_Delay column as well as any column that involves any air or arrival time data. We will remove these additional data because our model would rely to heavily on the data for prediction, and because some of the information is to closely related to the target column.

In [1057]:
```
df.drop(['ARR_DELAY'],axis=1,inplace=True)
```

In [1058]:
```
df.drop(['ARR_TIME','AIR_TIME','ACTUAL_ELAPSED_TIME','ARR_HOUR'],axis=1,inp
```

Save the DataFrames as a .csv file in order to import to BigML

In [1124]:
```
df.to_csv('data/Airline+Weather_data.csv',index=False)
```

Save our BigML Username and Api Key to our environment to access the API

In [1125]:
```
import os
os.environ['BIGML_USERNAME'] = "EFETOROS"
os.environ['BIGML_API_KEY'] = "7e5fc6a649fd0f8517fc8ecf2ebd30151c5d4fb4"
```

Creat our main API object that all the main functions will utilize.

In [1126]:
```
from bigml.api import BigML
api = BigML()
```

Importing Data to BigML

In order to start a BigML workflow, a source object has to be created. The API function that creates a source is `create_source` . The method's inputs will be a file path to the csv it will be converting. The source will be created from the csv files written by `to_csv` from before.

In [1127]:
```
source = api.create_source('data/Airline+Weather_data.csv')
```

BigML's `ok` method is called in order to assure that an object is created and will wait if it is not done being completed.

In [1063]:
```
api.ok(source)
```

Out[1063]:
```
True
```

Creating a Dataset

BigML will use the newly created source to create datasets which will enable the API to perform many more operations. In order to create a dataset, the API calls the function `create_dataset` . The method will take the source created by the API as an input.

In [1064]:
```
origin_dataset = api.create_dataset(source)
```

Test-Train Split

Since we want our data to stay in the form of BigML's datasets, the test-train split of the data will be done through BigML's API. This form will allow for the API's computations. The test-train split will be created by the function `create_dataset` mentioned before. However, it will take advantage of the more available inputs of the function. Many BigML API functions take in a dictionary with many fields as an additional input. These fields allow for much manipulation of the original function's outcome. In a test-train split, the field of sample_rate will allow for the choosing of the percentage of data being sampled. The train dataset will have out_of_bag field set to False and the test dataset will have it set to True. Since the test out_of_bag is set to True, its size will be 20% when its sample rate is 80%.

In [1065]:
```
origin_dataset = api.create_dataset(source)
train_dataset = api.create_dataset(
    origin_dataset, {"name": "Flight Delay | Training",
                     "sample_rate": 0.8, "seed": "my seed"})
test_dataset = api.create_dataset(
    origin_dataset, {"name": "Flight Delay | Test",
                     "sample_rate": 0.8, "seed": "my seed",
                     "out_of_bag": True})
api.ok(train_dataset)
api.ok(test_dataset)
```

Out[1065]:
```
True
```

Creating Ensembles

BigML's API allows for the creation of many models. For this dataset, Ensembles will be used. The BigML API will use the method `create_ensemble` . As stated before, the method takes in additional inputs for different use cases and manipulation of the function. We will create to ensembles.

1) In the first ensemble, we will balance the weights of the objective field, since there are more flights not delayed than delayed.

In [1066]:
```
ensemble_1 = api.create_ensemble(train_dataset, { \
    "balance_objective" : True})
api.ok(ensemble_1)
```

Out[1066]:
```
True
```

2) In the second ensemble, we will also balance the weights of the objective field, but we will also choose to exclude any information that involves exact departure time. For example, we will leave in computer reservation system departure time (CRS_DEP_TIME), since this data is the scheduled time and will be known even before take off, but we will drop departure delay (DEP_DELAY), which will only be known in the exact moment or very close to take off. This model will lose performance, however, it can be used to make predictions if a flight is going to be delayed much before the flight has taken off, on the other hand, the first model heavily relies on departure delay data to identify arrival delay, which might not be very useful.

In [1082]:
```
ensemble_2 = api.create_ensemble(train_dataset, { \
#     "objective_weights": [["YES", 8], ["NO", 1]],
    "balance_objective" : True,
    "excluded_fields": ["DEP_HOUR","DEP_TIME","DEP_DELAY"]})
api.ok(ensemble_2)
```

Out[1082]:
```
True
```

This function will be used to retrieve field names from IDs.

In [1083]:
```
def names(field_importance):
    names_of_important_fields = {}
    for keys in field_importance.keys():
        names_of_important_fields[train_dataset['object']["fields"][keys]["
    sorted_values = sorted(names_of_important_fields.items(), key=lambda kv
    return sorted_values
```

Below will be a list of ordered field importance for both models. We can see that the first model put a lot of weight on departure delay.

BigML API objects, such as ensemble_1, are nested dictionaries and will have many information within.

In [1084]:
```
field_importance_ensemble_1 = ensemble_1["object"]["importance"]
```

In [1085]:
```
names(field_importance_ensemble_1)
```

Out[1085]:
```
[('CRS_ARR_HOUR', 0.00019),
 ('DEST', 0.00027),
 ('DEP_AVG_HOURLYPrecip', 0.0003),
 ('DEP_AVG_HOURLYWindSpeed', 0.00031),
 ('CRS_DEP_HOUR', 0.00065),
 ('ARR_AVG_HOURLYWindSpeed', 0.00082),
 ('ORIGIN', 0.00289),
 ('DEP_HOUR', 0.00294),
 ('ARR_AVG_HOURLYDRYBULBTEMPC', 0.0038),
 ('DEP_HOURLYPrecip', 0.00492),
 ('ARR_AVG_HOURLYVISIBILITY', 0.00512),
 ('DEP_AVG_HOURLYDRYBULBTEMPC', 0.0059),
 ('DISTANCE', 0.00743),
 ('DEP_TIME', 0.01548),
 ('DAY_OF_WEEK', 0.01807),
 ('CRS_ELAPSED_TIME', 0.02178),
 ('DEP_HOURLYVISIBILITY', 0.02414),
 ('ARR_HOURLYPrecip', 0.02463),
 ('CRS_ARR_TIME', 0.02849),
 ('ARR_HOURLYWindSpeed', 0.03141),
 ('ARR_HOURLYVISIBILITY', 0.03163),
 ('FL_NUM', 0.03833),
 ('DEP_HOURLYWindSpeed', 0.04145),
 ('DAY_OF_MONTH', 0.04541),
 ('CRS_DEP_TIME', 0.0483),
 ('ARR_HOURLYDRYBULBTEMPC', 0.05175),
 ('DEP_HOURLYDRYBULBTEMPC', 0.07302),
 ('DEP_DELAY', 0.47056)]
```

In [1086]:
```
field_importance_ensemble_2 = ensemble_2["object"]["importance"]
```

In [1087]:
```
names(field_importance_ensemble_2)
```

Out[1087]:
```
[('DEP_AVG_HOURLYWindSpeed', 2e-05),
 ('DEST', 0.00026),
 ('DEP_AVG_HOURLYPrecip', 0.00032),
 ('ARR_AVG_HOURLYVISIBILITY', 0.00125),
 ('ARR_AVG_HOURLYDRYBULBTEMPC', 0.00141),
 ('DEP_AVG_HOURLYDRYBULBTEMPC', 0.00151),
 ('CRS_DEP_HOUR', 0.00349),
 ('ARR_AVG_HOURLYWindSpeed', 0.00411),
 ('ARR_AVG_HOURLYPrecip', 0.00574),
 ('ORIGIN', 0.00623),
 ('DEP_AVG_HOURLYVISIBILITY', 0.00724),
 ('ARR_HOURLYVISIBILITY', 0.01168),
 ('DISTANCE', 0.01937),
 ('DEP_HOURLYVISIBILITY', 0.01975),
 ('CRS_ELAPSED_TIME', 0.02997),
 ('ARR_HOURLYPrecip', 0.03154),
 ('DEP_HOURLYPrecip', 0.03274),
 ('ARR_HOURLYWindSpeed', 0.04471),
 ('FL_NUM', 0.05451),
 ('DEP_HOURLYDRYBULBTEMPC', 0.06314),
 ('DAY_OF_MONTH', 0.06618),
 ('DEP_HOURLYWindSpeed', 0.07475),
 ('DAY_OF_WEEK', 0.08143),
 ('CRS_DEP_TIME', 0.09465),
 ('CRS_ARR_HOUR', 0.09597),
 ('ARR_HOURLYDRYBULBTEMPC', 0.11033),
 ('CRS_ARR_TIME', 0.13771)]
```

Creating Evaluations

BigML's API allows for the creation of evaluations for specific models. The BigML API will use the method `create_evaluation` , and will take in the model of interest and the test_dataset.

In [1088]:
```
evaluation_1 = api.create_evaluation(ensemble_1, test_dataset)
api.ok(evaluation_1)
```

Out[1088]:
```
True
```

In [1089]:
```
evaluation_2 = api.create_evaluation(ensemble_2, test_dataset)
api.ok(evaluation_2)
```

Out[1089]:
```
True
```

Below are performance metrics for both models, as seen, the second model is not as accurate, however it may prove to be more useful. For example, an airline can mark a plane that has a higher chance of being delayed before departure, and allocate resources to hopefully save further delay expensives.

In [1090]:
```
print("Accuracy: ",evaluation_1["object"]["result"]["model"]["accuracy"])
print("Recall: ",evaluation_1["object"]["result"]["model"]["average_recall"
print("Precision: ",evaluation_1["object"]["result"]["model"]["average_prec
```

```
Accuracy:  0.87686
Recall:  0.83418
Precision:  0.87724
```

In [1091]:
```
print("Accuracy: ",evaluation_2["object"]["result"]["model"]["accuracy"])
print("Recall: ",evaluation_2["object"]["result"]["model"]["average_recall"
print("Precision: ",evaluation_2["object"]["result"]["model"]["average_prec
```

```
Accuracy:  0.75584
Recall:  0.64719
Precision:  0.75952
```