Capstone Project Report

I. Data Wrangling

1. Primary Exploratory Data Analysis

1.1. Dataset

I have chosen to work on an interesting dataset that I found on the NASA's website (https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh), encompassing the complete list of around 45000 meteorite landings on earth since the year 601. Each meteorite is further described by the 10 columns listed below:

- **name**: the name of the meteorite (typically a location, often modified with a number, year, composition, etc)
- **id**: a unique identifier for the meteorite
- **nametype**: one of: -- *valid*: a typical meteorite -- *relict*: a meteorite that has been highly degraded by weather on Earth
- **recclass**: the class of the meteorite; one of a large number of classes based on physical, chemical, and other characteristics
- **mass**: the mass of the meteorite, in grams
- **fall**: whether the meteorite was seen falling, or was discovered after its impact; one of: -- *Fell*: the meteorite's fall was observed -- *Found*: the meteorite's fall was not observed
- **year**: the year the meteorite fell, or the year it was found (depending on the value of fell)
- **reclat**: the latitude of the meteorite's landing
- **reclong**: the longitude of the meteorite's landing
- **GeoLocation**: a parentheses-enclose, comma-separated tuple that combines reclat and reclong

1.2. Data Exploration

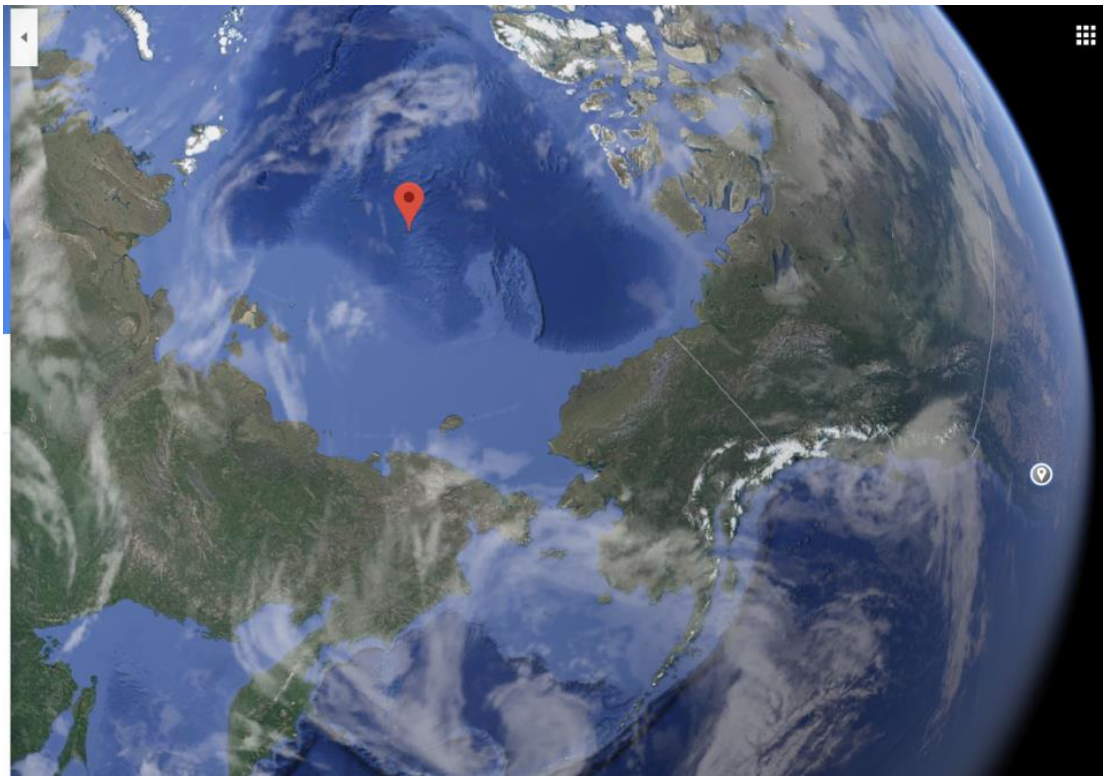
The row dataset is a csv file that I imported using the Jupyter Notebook. The imported file was quite clean, with, however, a lot of missing values (around 17%). The cleaning process is detailed in a separate ipynb file. Below is a preview of the steps that I took to make the data more readable:

- Used the head() method to briefly inspect the data
- Looked for missing values using the isnull() method
- Checked the unicity of the critical column ID
- Dropped duplicated rows
- Checked the percentage, per column, of the missing values
- Dropped the missing values that I could not retrieve
- Did some statistical analysis (mean, median, std...) to make sure that the data is accurate

- Looked for outliers by fact-checking the mass, the name and the geographical locations of certain meteorites
- Grouped the meteorites by classes

1.3. Looking for Outliers

In order to verify the accuracy of the data, I googled the names of random meteorites and checked if their mass or the geographical location of their landing matched the information provided by Wikipedia. For example: I checked the maximum longitude and saw whether the location existed or not. The result seemed logical.



Screenshot from Google Maps

I also selected the heaviest meteorite, Hoba, and verified all the information related to it, and once again, the results were completely satisfying. I couldn't, for obvious reasons, fact-check all the 45000 rows but for a primary analysis I think the dataset is for the most part accurate.

II. Data Visualization

1. Intro

The initial dataset from the NASA had 422 meteorite classes in it. A high number that did not necessarily reflect the distribution of meteorites. Indeed, as shown in the Phase-1 (Data Wrangling), the top ten meteorite classes encompassed approximately 80% of the total number of meteorites. For that reason, I decided to neglect – for now, the remaining 20% of the data – for a more concrete analysis.

2. Visualization

2.1. Scatter Plot

After extracting the needed segments, I used the Matplotlib Library to generate a Scatter Plot (meteorite classes on the x-axis and their mass on the y-axis) to see the distribution of the mass per class. Surprisingly, with the notable exception of a few outliers, the distribution was quite homogenous among the ten classes, with the majority of the meteorites having a mass range 0-500kg. Here are some conclusions:

- The heaviest meteorite belongs to the H5 class;
- The class L6 is more homogenous;
- The class CM2 has the lightest meteorites.

2.2. Seaborn lmplot

I used the Seaborn Library to visualize a sort of a time-series plot showing the distribution of the landings throughout the entire timescale of the dataset. Not surprisingly, most of the landings occurred after the 19th century, which can be explained by the restricted technological advances prior to that time, where only landings that were actually observed by humans were recorded.
The third plot shows the mass of the meteorites that landed throughout the timescale. Here are some conclusions:

- Accurate landing records really started in the 19th century;
- Recent meteorites seem to be heavier – which is probably due to the drastic increase of the landing records recently. More records => more meteorite landings => potentially heavier meteorites.

2.3. Plotly Library

2.3.1. Scatter Plot

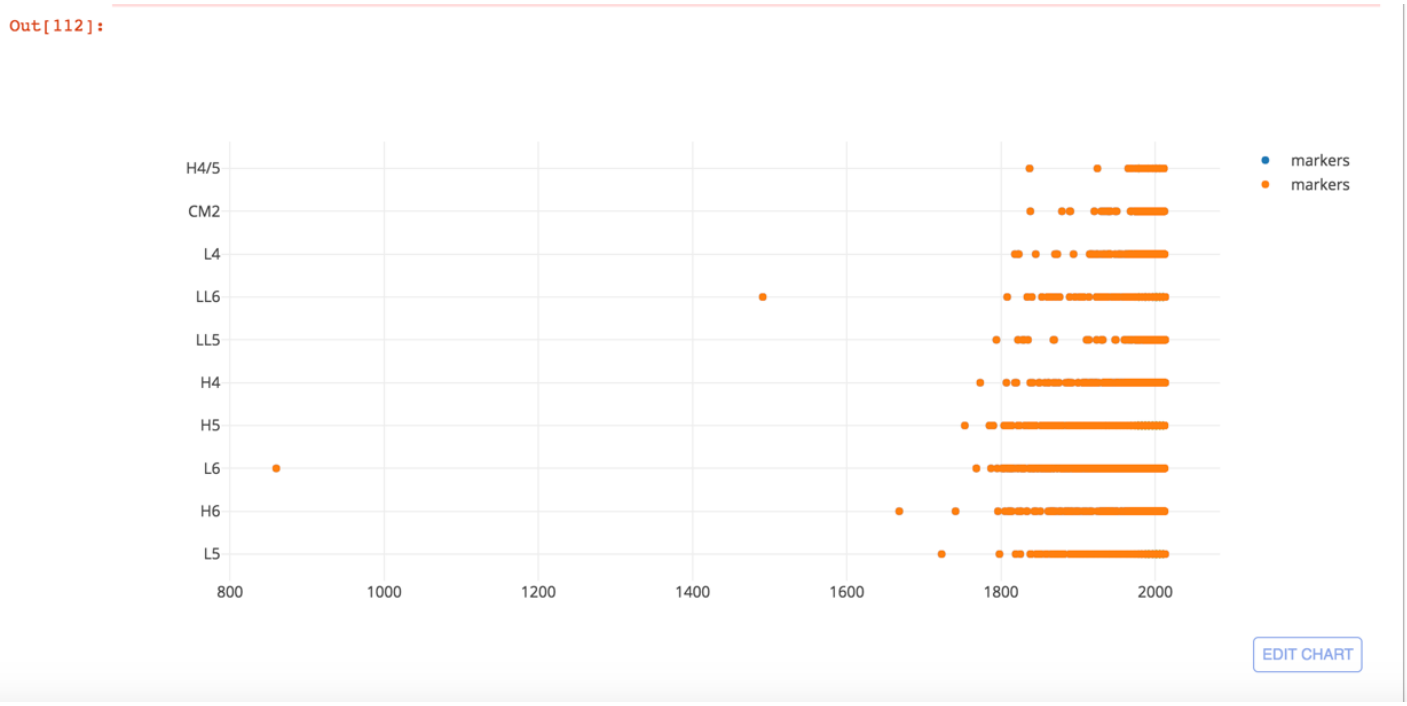
I opted for the Plotly Library for more esthetic plots. It's a bit slower and requires an API-Key to run adequately but it provides a variety of options that cannot be found elsewhere. The scatter plot shows the distribution of the top ten meteorite classes. We can learn, for example, that the L6 class is the oldest class observed and that the H4/5 class is contemporary.

2.3.2. Scatter Map Box

This is, I believe, the most interesting plot provided by the Plotly Library. I tried to plot the geo-location of about 35.000 the meteorite on the dataset, based on the latitude and the longitude of the places where they landed. Surprisingly, only a minority of the meteorites landed in the seas/oceans, despite them constituting around 70% of the planet.

Note:
Because the library requires an internet connection to display the graphs, I couldn't incorporate them in the notebook. However, I have taken a few screenshots to illustrate the results:

Out[112]:

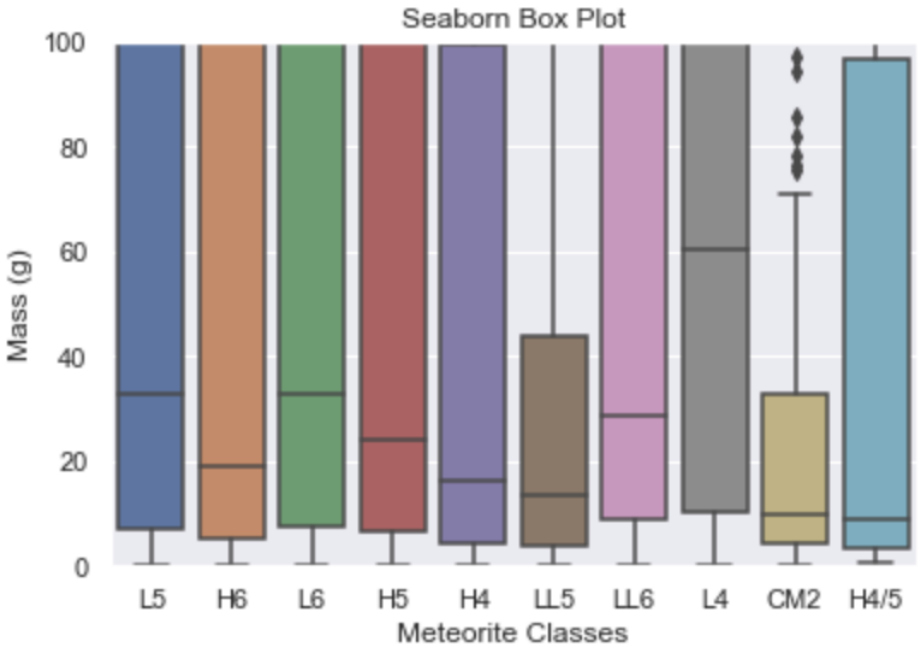


EDIT CHART

III. Statistical Thinking

Often, it is very useful to think "statistically". Numbers are the language of the universe, they help us understand things more clearly. Conducting a statistical analysis helped me to understand some "hidden" aspects of the dataset. In this section, I tried to perform a coherent statistical analysis. The dataset being very tedious, the results are quite restricted. However, they offer an interesting "plus-value".

As shown on the Jupyter Notebook, I focused on the “Mass” column due to its relevancy to the Machine Learning part, where I started by plotting a Box Plot for each class to see whether there outliers impacted significantly the mean. The results were note quite useful since the means of all the ten classes are very close to each other (all of them are between 15 and 70g).



Then I tried to plot the EDF - Empirical Distribution Function and the PDF – Probability Distribution Function, to learn more about the distribution of the data. Again, the results were unequivocal as shown on the Notebook: the data was not normally distributed.

For the sake of it, I also tried to perform a t-test to find out the true mean of the population. Unsurprisingly, the p-value was equal to 1, which indicates that the Null Hypothesis (the true mean is 1.534274e+03g) is true.

The dataset being very hard to explore because it is deeply lacking numerical features, the results of the inferential statistics part were very limited.

IV. Machine Learning

Indubitably the most entertaining part of the capstone project. After numerous fortuitous but captivating attempts to build a good model to predict something useful (I tried six supervised algorithms and k-means clustering) I determined that realistically, due to the circumstances, I could still predict something great: the *exact* emplacement of the next meteorite landings, based on their class, mass and the expected year of landing (all the three features can be evaluated via a telescope).

1. What to predict?

Features: mass, class and year

Independent variable: GeoLocation (latitude and longitude)

```
# Selecting the features and the target variable

X = df[['recclass', 'mass', 'year']] # Selecting the class of the meteorite, their mass and the predicted landing year
y = df[['reclat', 'reclong']] # The target variable represents the latitude and longitude of the expected landing
```

2. Which algorithm to use?

From my experiments, I have determined that Random Forest Regressor was the algorithm that gave the best results on the dataset.

```
In [67]: X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
         rfr = RandomForestRegressor().fit(X_train, y_train)

/Users/youcefdjeddar/anaconda3/envs/fastai-cpu/lib/python3.6/site-packages/sklearn/ensemble/forest.py:248: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

In [68]: print("Training set score: {:.2f}".format(rfr.score(X_train, y_train)))
         print("Test set score: {:.2f}".format(rfr.score(X_test, y_test)))
         %time rfr.fit(X_train, y_train)

Training set score: 0.92
Test set score: 0.53
CPU times: user 280 ms, sys: 9.85 ms, total: 290 ms
Wall time: 290 ms

Out[68]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                    oob_score=False, random_state=None, verbose=0, warm_start=False)
```

Clear overfitting. However, the score on the test set is very good. The algorithm can now predict, with 53% accuracy, where exactly the next meteorite landing will occur.

Random Forest has different parameters that we can explore to possibly enhance the score on the test set.

Without any giving parameters, RFR gave a score of 92% on the training set but only 53% on the test set. I was clearly overfitting. To remediate to this situation, I tried to incorporate new parameters:

3. What parameters?

a. **n_estimators** = number of trees in the foreset

b. **max_depth** = max number of levels in each decision tree

c. **min_samples_leaf** = min number of data points allowed in a leaf node

d. **n_jobs** = The number of jobs to run in parallel for both fit and predict. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors.

The results were better but the model was still overfitting. After a Cross Validation I conducted a Grid Search to find the best possible combination of the parameters:

4. Cross Validation

Cross Validation

In [70]:
```
rfr = RandomForestRegressor(n_estimators = 20, min_samples_leaf = 3, max_features = 0.5, n_jobs =
1).fit(X_train, y_train)
print("Training set score: {:.2f}".format(rfr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(rfr.score(X_test, y_test)))
%time rfr.fit(X_train, y_train)
```

```
Training set score: 0.77
Test set score: 0.58
CPU times: user 245 ms, sys: 6.8 ms, total: 252 ms
Wall time: 253 ms
```

Out[70]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
            max_features=0.5, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=3, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
            oob_score=False, random_state=None, verbose=0, warm_start=False)
```

5. Grid Search

Grid Search

In [77]:
```
# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, random_state=0)
# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X_trainval, y_trainval, random_state=1)
print("Size of training set: {}    size of validation set: {}     size of test set:"
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))
best_score = 0
for n_estimators in [1, 10, 20,  50, 100, 200]:
    for min_samples_leaf in [1, 3, 10, 20, 50, 100]:
        for max_depth in [1, 2, 3, 4, 5, 10, 15]:
            for n_jobs in [-1, 1, 2, 5, 10]:
                rf = RandomForestRegressor(n_estimators = n_estimators,min_samples_leaf = min_samp
les_leaf,
                                           max_depth = max_depth, n_jobs = n_jobs )
                rf.fit(X_train, y_train)
                score = rf.score(X_valid, y_valid)
                if score > best_score:
                    best_score = score
                    best_parameters = {'n_estimators': n_estimators, 'min_samples_leaf': min_sampl
es_leaf,
                                       'max_depth': max_depth,'n_jobs': n_jobs}
                    rf = RandomForestRegressor(**best_parameters)
                    rf.fit(X_trainval, y_trainval)
                    test_score = rf.score(X_test, y_test)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

```
Size of training set: 17058    size of validation set: 5687     size of test set: 7582

Best score on validation set: 0.63
Best parameters:  {'n_estimators': 100, 'min_samples_leaf': 10, 'max_depth': 15, 'n_jobs': -1}
Test set score with best parameters: 0.63
```

Notice that I had to create a validation set to avoid the test set from being involved in parameters tuning.

6. Final results

Using Random Forest Regressor, the algorithm was able to determine, with 62% accuracy, where the next meteorite landings will occur, based only on the class, the mass and the expected landing year of the meteorites. Very encouraging results.

V. Capstone Project Summary

During this interesting journey, the following objectives were reached:

1. The dataset was properly cleaned;
2. The dataset was separated into different segments, each segment having a specific purpose;
3. Important segments were adequately visualized via scatter plots, histograms, box plots and map boxes;
4. Interesting statistical analysis were conducted on the dataset (t-test, EDF, PDF...);
5. The dataset was correctly encoded to be ready for the machine learning part;
6. After experimenting different machine learning algorithms (supervised and unsupervised) and combinations (classification and regression), Random Forest Regressor was always giving satisfying results;
7. After a multitude of tests, determining where the next meteorite landings will occur was the most realistic prediction;
8. With only three features that can be observed and determined via telescopes, the algorithm was giving a fair score;
9. A cross validation and a grid search were performed on the dataset to enhance the test score and to reduce over-fitting, successfully.

VI. What now?

The dataset needs more details, like the chemical components of each meteorite, to determine which ones are going to reach planet Earth and which ones are going to get disintegrated while entering the atmosphere. In addition, the dataset has a lot of missing values that I think considerably altered the results of some statistical analysis.