Final Project Report
Course: CS536
Name: Tathagata Dutta
netID: td425

Data Completion and Interpolation

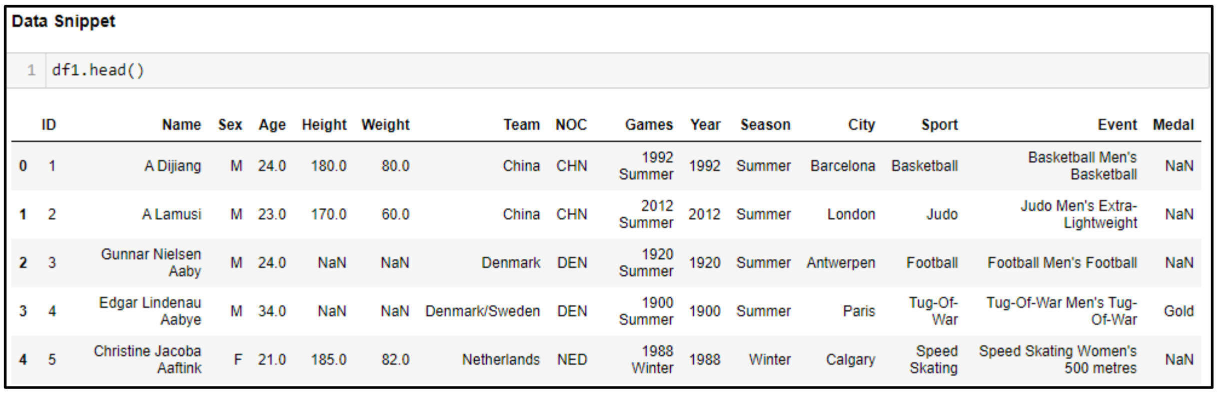
0. About the Dataset

For this project, I chose this dataset from Kaggle. It is a historical dataset on the modern Olympic Games, including all the Games from Athens 1896 to Rio 2016.

A brief description of the features is as follows:

1. ID – Unique identification number for each athlete
2. Name – Name of the Athlete
3. Sex – Male or Female
4. Age – in years
5. Height – in centimeters
6. Weight – in kilograms
7. Team – Name of team (generally refers to country)
8. NOC - National Olympic Committee 3-letter code (also mostly represents country)
9. Games - Year and season (eg. 2012 Summer)
10. Year – The year the Olympic took place
11. Season - Summer or Winter
12. City - Host city
13. Sport – Type of sport (eg. Basketball)
14. Event – Event (eg. Gymnastics Men's Horse Vault)
15. Medal - Gold, Silver, Bronze, or NA (NA = did not win)

Let's look at a snippet of the data:

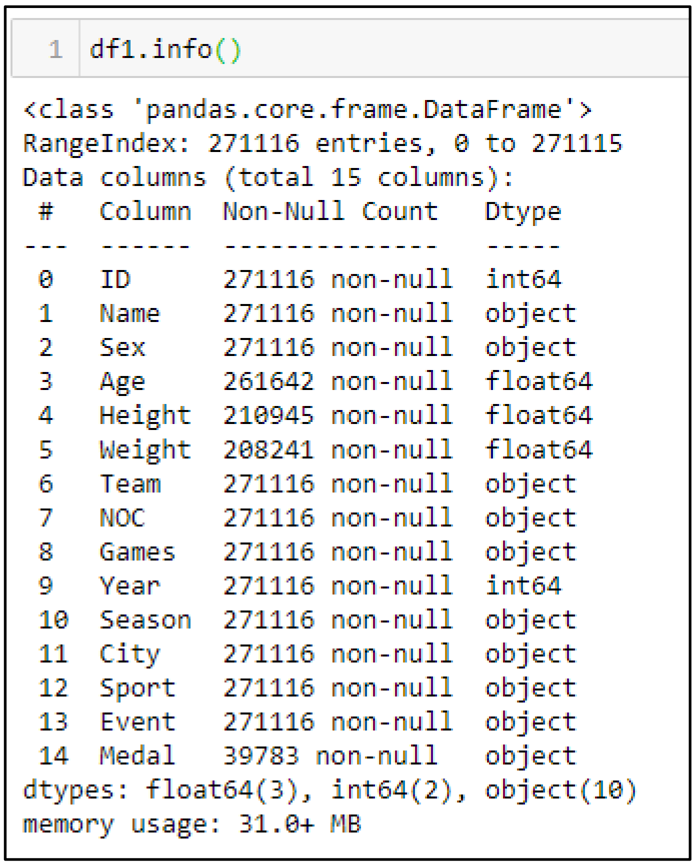


Data Snippet

```
df1.head()
```

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

And some general information and statistics about the data:

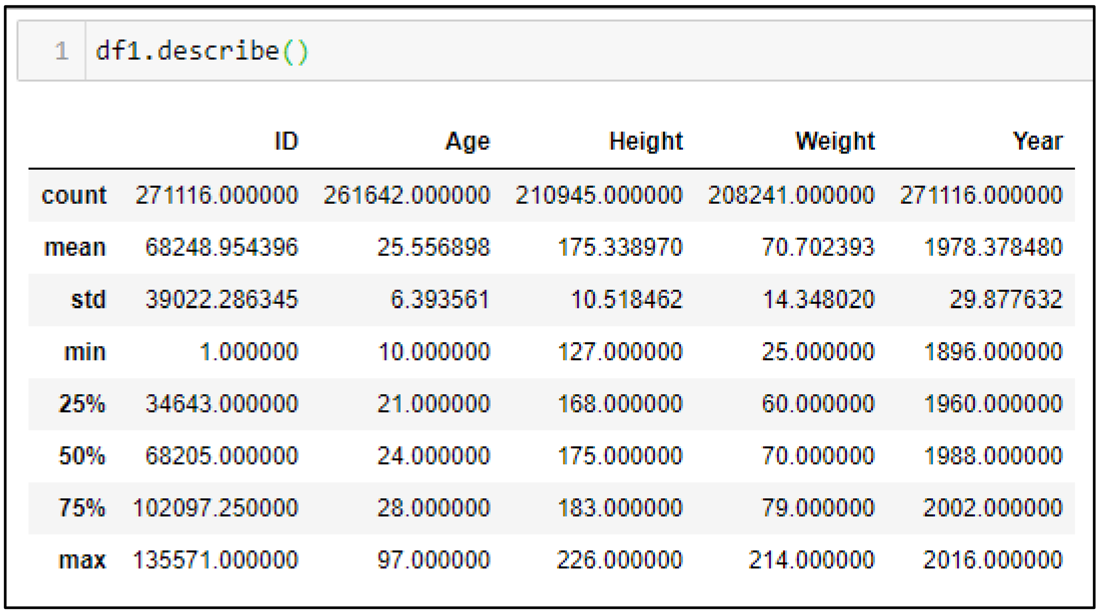


```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   ID      271116 non-null  int64  
 1   Name    271116 non-null  object 
 2   Sex     271116 non-null  object 
 3   Age     261642 non-null  float64
 4   Height  210945 non-null  float64
 5   Weight  208241 non-null  float64
 6   Team    271116 non-null  object 
 7   NOC     271116 non-null  object 
 8   Games   271116 non-null  object 
 9   Year    271116 non-null  int64  
 10  Season  271116 non-null  object 
 11  City    271116 non-null  object 
 12  Sport   271116 non-null  object 
 13  Event   271116 non-null  object 
 14  Medal   39783 non-null   object 
dtypes: float64(3), int64(2), object(10)
memory usage: 31.0+ MB
```

The dataset has **271,116 datapoints** including NaNs.

Some statistics about the numerical features:

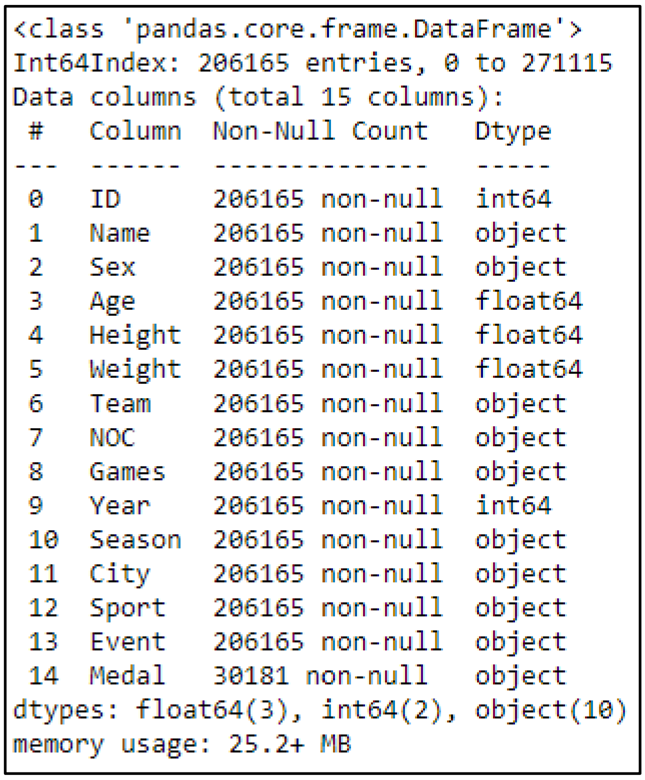


```
df1.describe()
```

	ID	Age	Height	Weight	Year
count	271116.000000	261642.000000	210945.000000	208241.000000	271116.000000
mean	68248.954396	25.556898	175.338970	70.702393	1978.378480
std	39022.286345	6.393561	10.518462	14.348020	29.877632
min	1.000000	10.000000	127.000000	25.000000	1896.000000
25%	34643.000000	21.000000	168.000000	60.000000	1960.000000
50%	68205.000000	24.000000	175.000000	70.000000	1988.000000
75%	102097.250000	28.000000	183.000000	79.000000	2002.000000
max	135571.000000	97.000000	226.000000	214.000000	2016.000000

1. Describe your Model:

The approach I took here is first I removed all datapoints if any feature is NaN. This still leaves us with **206,165 datapoints.** We will forcefully remove data and check model accuracy.



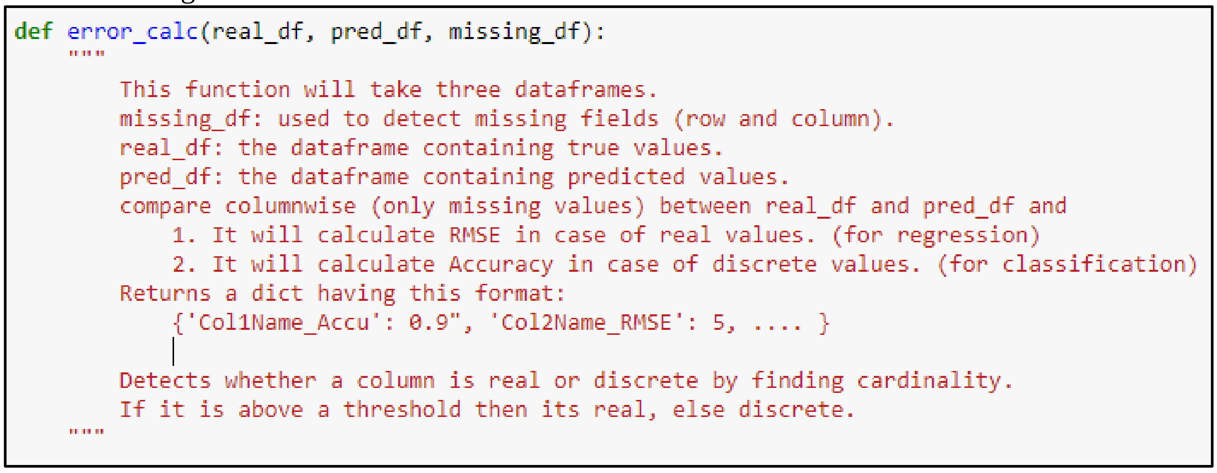
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 206165 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ID      206165 non-null  int64
 1   Name    206165 non-null  object
 2   Sex     206165 non-null  object
 3   Age     206165 non-null  float64
 4   Height  206165 non-null  float64
 5   Weight  206165 non-null  float64
 6   Team    206165 non-null  object
 7   NOC     206165 non-null  object
 8   Games   206165 non-null  object
 9   Year    206165 non-null  int64
 10  Season  206165 non-null  object
 11  City    206165 non-null  object
 12  Sport   206165 non-null  object
 13  Event   206165 non-null  object
 14  Medal   30181 non-null   object
dtypes: float64(3), int64(2), object(10)
memory usage: 25.2+ MB
```

The data was read into a dataframe. Various **cleaning** strategies were taken to clean the data. *Although I created a function to handle **preprocessing**/cleaning, it is the only function that is specific to the data and is not generalized.*

ID is a unique identification number for an athlete and can repeat in multiple datapoints. One hot encoding this would explore our feature space. Hence I left it as it is. *A point to note is, the regression/ classification algorithms which I wrote **mean centers and normalizes** the data within itself, so even high values will not have dangerous implications.* I dropped the Names feature because it is correlated to ID. I also dropped the Games feature as it just the concatenation of Season and Year which we already have. I could have one hot encoded the Year feature as well, but felt that increase in years bears some relevance and hence kept it as it is. I one hot encoded the Sex and Season columns. For the Medal feature, I replaced NaN with a string “DNW” (Did Not Win). Now it can have 4 possible values: Gold, Silver, Bronze, and DNW. Now I one hot encoded this feature to have 4 new features.

While predicting numerical features (regression), I used **RMSE** as the metric for error. And for categorical features (classification), I used **Accuracy** as the metric for error.

The following function finds the column-wise error and returns a dictionary of error for each feature that had missing data. See below:

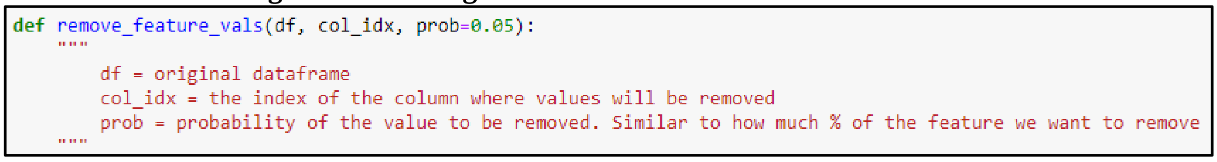


```
def error_calc(real_df, pred_df, missing_df):
    """
        This function will take three dataframes.
        missing_df: used to detect missing fields (row and column).
        real_df: the dataframe containing true values.
        pred_df: the dataframe containing predicted values.
        compare columnwise (only missing values) between real_df and pred_df and
            1. It will calculate RMSE in case of real values. (for regression)
            2. It will calculate Accuracy in case of discrete values. (for classification)
        Returns a dict having this format:
            {'Col1Name_Accu': 0.9", 'Col2Name_RMSE': 5, .... }

        Detects whether a column is real or discrete by finding cardinality.
        If it is above a threshold then its real, else discrete.
    """
```

The other features had very high cardinality so one hot encoding was not an option. Also, I checked with different prebuilt ML libraries if including them would reduce the error. Linear Regression gave very bad results, whereas RandomForests and NNs gave slight improvements (attached as a separate .ipynb file named "**Testing Data.ipynb**"). As the original dataset had missing values in 3 features only: Age, Height and Weight, I tested for RMSE and MAE by splitting the complete (206,165) dataset into train and test sets in the ratio 4:1. I feel that it would not have been possible to test for errors (RMSE) using an unoptimized self-written algorithm on such a vast number of features (after one-hot encoding) that can barely utilize even all CPU cores. Although I am not completely satisfied with this decision, but I dropped these features.

I created a function to **generate missing values**:



```
def remove_feature_vals(df, col_idx, prob=0.05):
    """
        df = original dataframe
        col_idx = the index of the column where values will be removed
        prob = probability of the value to be removed. Similar to how much % of the feature we want to remove
    """
```

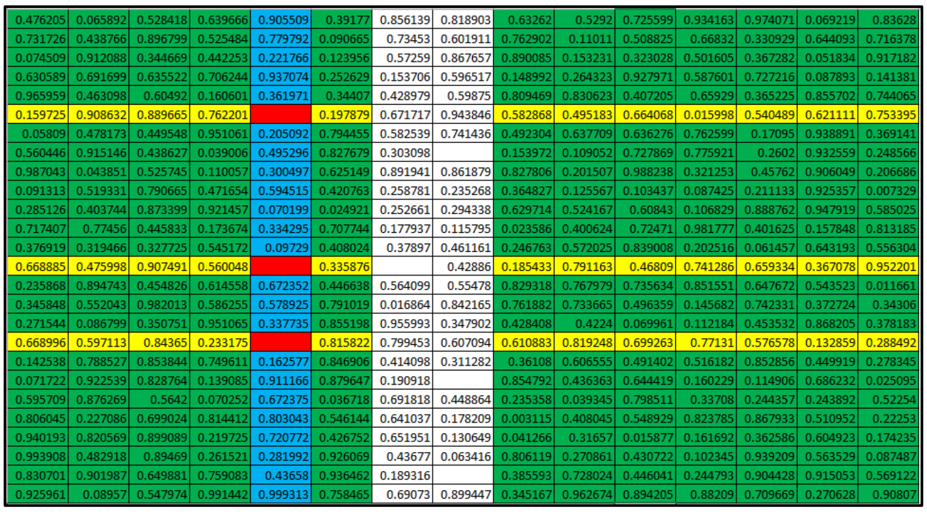
This way, I will have the **original dataset to check for errors**. The missing values in the original dataset can always be imputed but that doesn't allow us to check how accurate we were in predicting so.

2. Describe your Training Algorithm:

Apart from the basic completion agent, I implemented **two** different algorithms, one of them being a very naïve approach. For both the algorithms I used **Linear regression** for predicting numerical/real values and **Logistic regression** for predicting categorical/discrete values. Again using the same logic of both the algorithms, I used sk-learn's **RandomForestRegressor** and **RandomForestClassifier** to achieve better results.

Algorithm 1 (naïve method): Suppose our dataset has 'n' features that have missing values. We drop all records which have missing values. This way we are left with columns having no NaN values. We treat these columns as our X. Now we treat each column that had missing values one by one and treat them as Y and train our model. For training set X and Y, datapoints were taken for which Y has no missing values. And we use these models to predict the missing values of Y. After 1 such feature is predicted, we follow the same logic and predict for the rest 'n-1' features.

A sample spreadsheet is shown to explain (visualize) the algorithm.



0.476205	0.065892	0.528418	0.639666	0.905509	0.39177	0.856139	0.818903	0.63262	0.5292	0.725599	0.934163	0.974071	0.069219	0.83628
0.731726	0.438766	0.896799	0.525484	0.779792	0.090665	0.73453	0.601911	0.762902	0.11011	0.508825	0.66832	0.330929	0.644093	0.716378
0.074509	0.912088	0.344669	0.442253	0.221766	0.123956	0.57259	0.867657	0.890085	0.153231	0.323028	0.501605	0.367282	0.051834	0.917182
0.630589	0.691699	0.635522	0.706244	0.937074	0.252629	0.153706	0.596517	0.148992	0.264323	0.927971	0.587601	0.727216	0.087893	0.141381
0.965959	0.463098	0.60492	0.160601	0.361971	0.34407	0.428979	0.59875	0.809469	0.830623	0.407205	0.65929	0.365225	0.855702	0.744065
0.159725	0.908632	0.889665	0.762201	🟥	0.197879	0.671717	0.943846	0.582868	0.495183	0.664068	0.015998	0.540489	0.621111	0.753395
0.05809	0.478173	0.449548	0.951061	0.205092	0.794455	0.582539	0.741436	0.492304	0.637709	0.636276	0.762599	0.17095	0.938891	0.369141
0.560446	0.915146	0.438627	0.039006	0.495296	0.827679	0.303098		0.153972	0.109052	0.727869	0.775921	0.2602	0.932559	0.248566
0.987043	0.043851	0.525745	0.110057	0.300497	0.625149	0.891941	0.861879	0.827806	0.201507	0.988238	0.321253	0.45762	0.906049	0.206686
0.091313	0.519331	0.790665	0.471654	0.594515	0.420763	0.258781	0.235268	0.364827	0.125567	0.103437	0.087425	0.211133	0.925357	0.007329
0.285126	0.403744	0.873399	0.921457	0.070199	0.024921	0.252661	0.294338	0.629714	0.524167	0.60843	0.106829	0.888762	0.947919	0.585025
0.717407	0.77456	0.445833	0.173674	0.334295	0.707744	0.177937	0.115795	0.023586	0.400624	0.72471	0.981777	0.401625	0.157848	0.813185
0.376919	0.319466	0.327725	0.545172	0.09729	0.408024	0.37897	0.461161	0.246763	0.572025	0.839008	0.202516	0.061457	0.643193	0.556304
0.668885	0.475998	0.907491	0.560048	🟥	0.335876		0.42886	0.185433	0.791163	0.46809	0.741286	0.659334	0.367078	0.952201
0.235868	0.894743	0.454826	0.614558	0.672352	0.446638	0.564099	0.55478	0.829318	0.767979	0.735634	0.851551	0.647672	0.543523	0.011661
0.345848	0.552043	0.982013	0.586255	0.578925	0.791019	0.016864	0.842165	0.761882	0.733665	0.496359	0.145682	0.742331	0.372724	0.34306
0.271544	0.086799	0.350751	0.951065	0.337735	0.855198	0.955993	0.347902	0.428408	0.4224	0.069961	0.112184	0.453532	0.868205	0.378183
0.668996	0.597113	0.84365	0.233175	🟥	0.815822	0.799453	0.607094	0.610883	0.819248	0.699263	0.77131	0.576578	0.132859	0.288492
0.142538	0.788527	0.853844	0.749611	0.162577	0.846906	0.414098	0.311282	0.36108	0.606555	0.491402	0.516182	0.852856	0.449919	0.278345
0.071722	0.922539	0.828764	0.139085	0.911166	0.879647	0.190918		0.854792	0.436363	0.644419	0.160229	0.114906	0.686232	0.025095
0.595709	0.876269	0.5642	0.070252	0.672375	0.036718	0.691818	0.448864	0.235358	0.039345	0.798511	0.33708	0.244357	0.243892	0.52254
0.806045	0.227086	0.699024	0.814412	0.803043	0.546144	0.641037	0.178209	0.003115	0.408045	0.548929	0.823785	0.867933	0.510952	0.22253
0.940193	0.820569	0.899089	0.219725	0.720772	0.426752	0.651951	0.130649	0.041266	0.31657	0.015877	0.161692	0.362586	0.604923	0.174235
0.993908	0.482918	0.89469	0.261521	0.281992	0.926069	0.43677	0.063416	0.806119	0.270861	0.430722	0.102345	0.939209	0.563529	0.087487
0.830701	0.901987	0.649881	0.759083	0.43658	0.936462	0.189316		0.385593	0.728024	0.446041	0.244793	0.904428	0.915053	0.569122
0.925961	0.08957	0.547974	0.991442	0.999313	0.758465	0.69073	0.899447	0.345167	0.962674	0.894205	0.88209	0.709669	0.270628	0.90807

(a) Green columns is our X; blue column is our first Y (white ones are next)
(b) Fit a model using X and Y
(c) Predict red values (one at a time) using rows with yellow values (one row at a time)
(d) Continue for the remaining white columns.

Algorithm 2:

(a) Calculate the standard deviation of each column/feature (say n) with missing values.
(b) Take the least 'n-1' columns and fill them with the mean/mode (like we did in the basic completion agent, *all the while keeping a record of the addresses of each missing value.*
(c) Train a model using 'k-1' filled columns as X and the other one as Y (k being the total number of columns).
(d) Predict the last unfilled column using the model, *all the while keeping a record of the addresses of each missing value.*
    a. Up to this step is epoch 0.
    b. Save RMSE/Accuracy for each feature that had missing values in a list of dict.
(e) Now run an iterative process to predict the initial n features (which had missing values) one by one, taking the rest 'k-1' columns as X and the other one as Y
    a. Repeat any number of epochs over all Ys.
    *b. *only predict values that were initially missing.*
    c. Save RMSE/Accuracy for each feature that had missing values in a list of dict.

View the list of dict containing RMSE and Accuracy and choose the best number of epochs to prevent overfitting and rerun on the original dataset (having missing features) or future datapoints.

A **third** algorithm I thought of (which is similar to the above one, but I felt would yield better results) but couldn't implement due to shortage of time (already being late):

(a) Train and predict the values for which rows have only 1 missing value.
    a. The training set will be the rows with non-missing values
(b) Now all rows which have missing values have at least 2 missing values.
    a. Fill 1 value with mean/mode (preferably the one which has lower SD)
    b. Predict the remaining using the logic we used in Algorithm 2
(c) Now all rows which have missing values have at least 3 missing values.
    a. Fill 2 value with mean/mode (preferably the ones which has least SDs)
    b. Predict the remaining using the logic we used in Algorithm 2
    keep a record of the addresses of each missing value in (a), (b), and (c)
(d) Repeat until all values are imputed.
(e) Use the (e)th step as in Algorithm 2.

The 2nd (and 3rd?) algorithm may not be able to train over all features at one go. However, in steps, it can train over all (but one) and predict all missing values.

For the 2nd algorithm, it took time to train over multiple epochs. A way to settle them was taking a subset of the entire dataset and training of that, although the model would not be as strong. I have ran it on the full dataset and the 2nd algorithm with random forests took ages to train and predict. For handling bugs, I sometimes used a subset for faster (but bad) results.
Furthermore, in the gradient descent portion of linear and logistic regression training, it has a hard cap (customizable), but may terminate early if weights don't change. This might reduce some training burden.

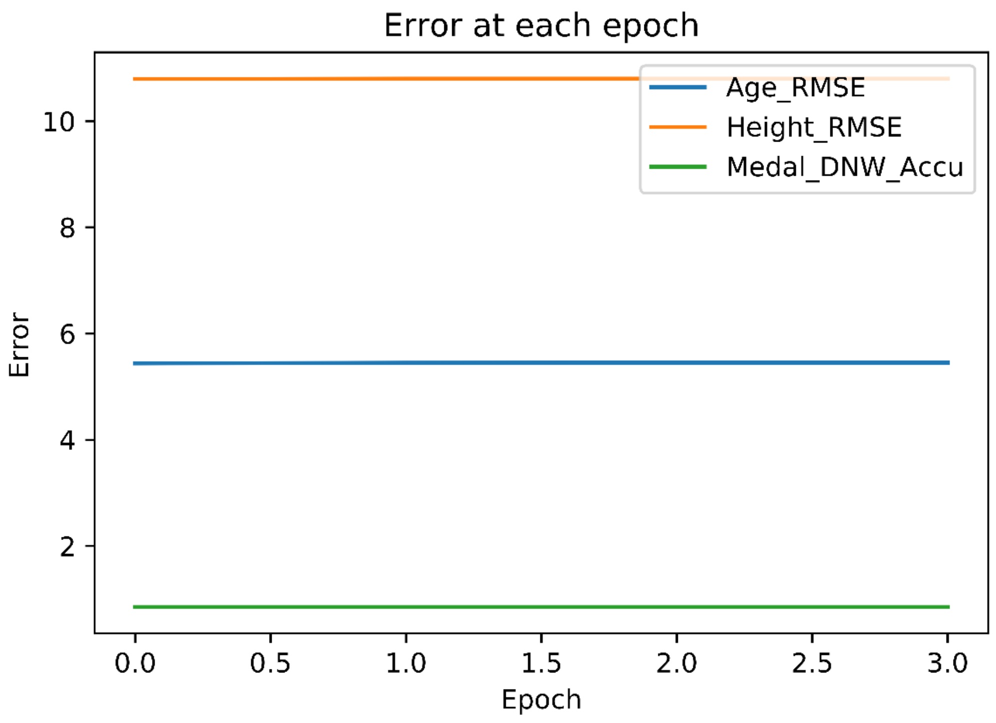
3. Describe your Model Validation:

The entire data (almost) is our canvas. We can use the entire data (wherever possible) for training and then predict the missing values.

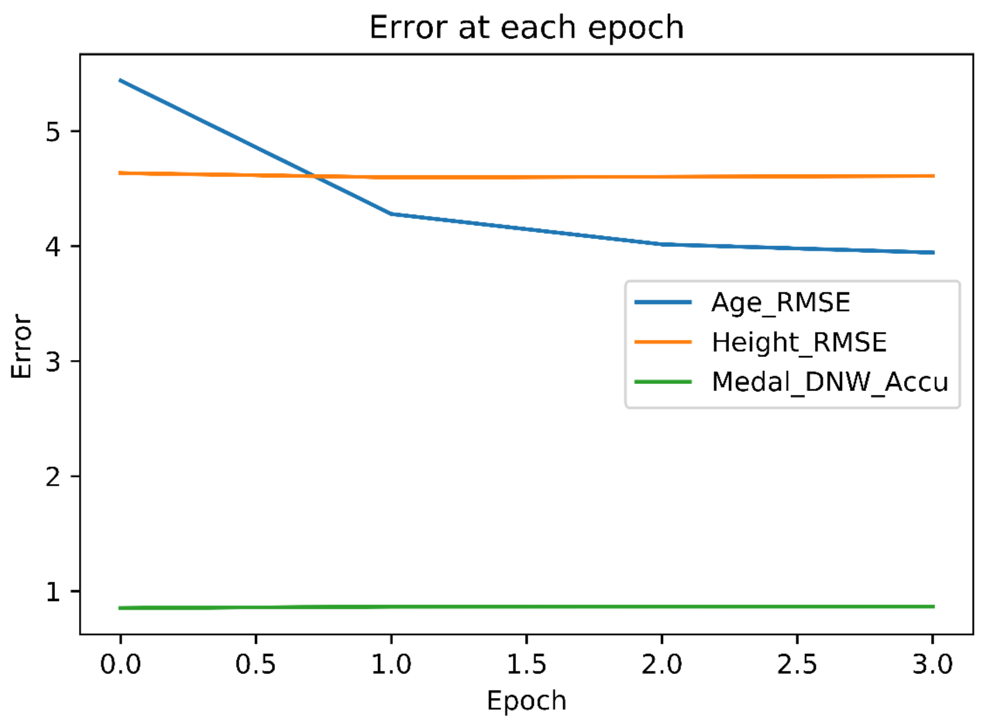
The second algorithm stores the RMSE/Accuracy of each feature at each epoch (in a list of dict). One can refer to that to check when the errors are increasing and stop early (to prevent overfitting).

I had to play with ~200k datapoints, which is not large if real word situations are considered, but even then it took some time for training as most handwritten algorithms cant even utilize the full power of CPU, let alone the GPU.

RMSE/Accuracy of 2nd Algorithm with Linear and Logistic regression after each epoch:



For 2nd algorithm with Linear and Logistic Regression, RMSE/Accuracy didn't change (change is there but very low) after every iteration.
RMSE/Accuracy of 2nd Algorithm with RandomForests after each epoch:



For 2nd algorithm with RandomForests, RMSE for the feature Age got better, whereas the other two features didn't see any change (very small change).

---

4. Evaluate your Model:

My model is better at predicting numerical data (regression) than categorical data (classification).

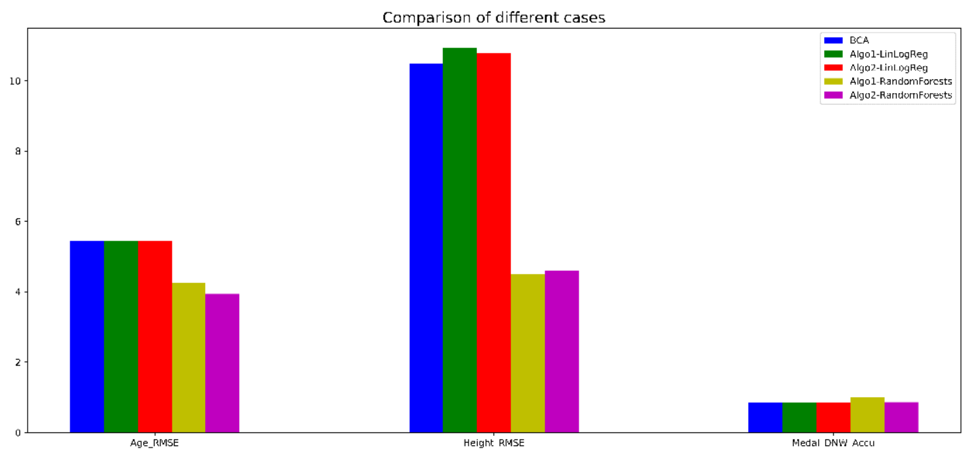
Although there is no specific requirement of having a minimum number of features in any of the algorithms I discussed, they will provide better results when more features are available.

The algorithms are fully generalized to handle any number of missing features, any percentage of missing values in a specific feature, and both numerical and categorical data. One restriction is that it can only handle binary classification. However, all categorical data is one-hot encoded, so I didn't have to worry about this issue.

Age, Height, Weight, and Season are the most important features in the data set and it is easier to predict other features if all or most of these features are present. Unfortunately, in the original dataset, Age, Height, and Weight are the features that have missing values.

The reason that these are the most important features is that they are quite varied and are the most important features (not in ML terms) of an individual. Also, Season is an important feature (ML terminology) because the sports being played in summer are quite different from winter.

Comparison of different algorithms with different ML techniques and its corresponding errors:



BCA, and both algorithms with Linears and Logistic Regression perform similarly.

For Age and Height, both algorithms with RandomForests perform better.

---

5. Analyze the Data:

Age, Height, Weight, and Season are the most important and relevant features in the data which can be confirmed from weights after training/fitting a model.

ID and Name are correlated, hence either one can be dropped while keeping the other. I chose to drop Name to prevent an extra step of Label Encoding. I also dropped the Games feature as it just the concatenation of Season and Year which we are using in the dataset.

Some features like location, sport/event, which had very high cardinality. One hot encoding them would drastically increase the feature space. I checked with fast prebuilt ML libraries and found that these features didn't reduce the error that much compared to the added complexity.

For this dataset, the conclusion one can draw after experiencing sports events over the ages. Based on the dataset, its inference, and personal expertise, the relevant features can be deduced for the best outcome for regression or classification.

---

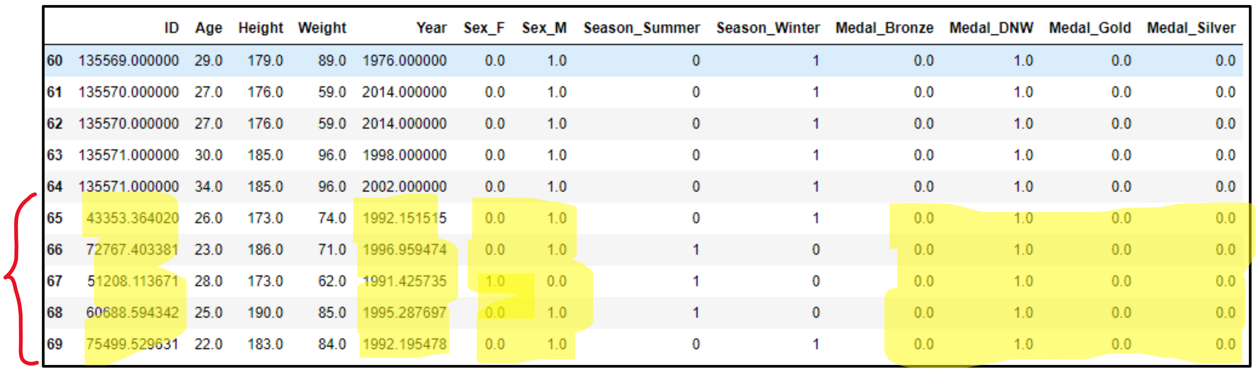
6. Generate Data:

Known:

- The system/algorithm takes in a dataframe containing missing values and trains on itself to predict missing values.
- In the last answer, we claimed Age, Height, Weight, and Season as the most valuable/relevant features

So, say if we want to say generate 5 new datapoints, we add new rows at the end of the existing dataframe (fully filled), and only fill in values of Age, Height, Weight, and Season (chosen randomly from the existing dataframe itself). Then we pass this dataframe to our system to get a newly filled dataframe. The last 5 rows are now synthetically generated.

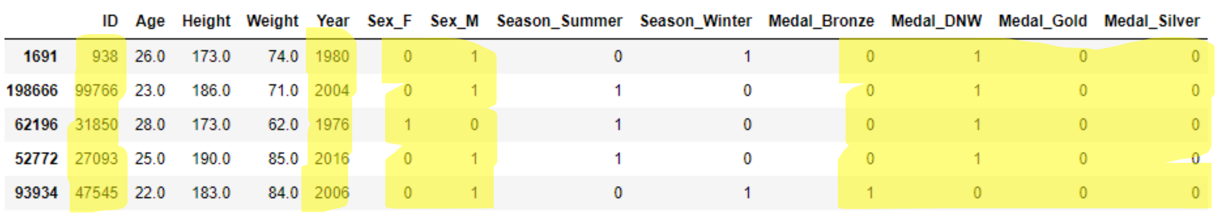
Let us view last 10 rows of this dataframe to see how the first 5 rows (true values) compare to the last 5 rows (generated).



	ID	Age	Height	Weight	Year	Sex_F	Sex_M	Season_Summer	Season_Winter	Medal_Bronze	Medal_DNW	Medal_Gold	Medal_Silver
60	135569.000000	29.0	179.0	89.0	1976.000000	0.0	1.0	0	1	0.0	1.0	0.0	0.0
61	135570.000000	27.0	176.0	59.0	2014.000000	0.0	1.0	0	1	0.0	1.0	0.0	0.0
62	135570.000000	27.0	176.0	59.0	2014.000000	0.0	1.0	0	1	0.0	1.0	0.0	0.0
63	135571.000000	30.0	185.0	96.0	1998.000000	0.0	1.0	0	1	0.0	1.0	0.0	0.0
64	135571.000000	34.0	185.0	96.0	2002.000000	0.0	1.0	0	1	0.0	1.0	0.0	0.0
65	43353.364020	26.0	173.0	74.0	1992.151515	0.0	1.0	0	1	0.0	1.0	0.0	0.0
66	72767.403381	23.0	186.0	71.0	1996.959474	0.0	1.0	1	0	0.0	1.0	0.0	0.0
67	51208.113671	28.0	173.0	62.0	1991.425735	1.0	0.0	1	0	0.0	1.0	0.0	0.0
68	60688.594342	25.0	190.0	85.0	1995.287697	0.0	1.0	1	0	0.0	1.0	0.0	0.0
69	75499.529631	22.0	183.0	84.0	1992.195478	0.0	1.0	0	1	0.0	1.0	0.0	0.0

Apart from Year and ID (which require a little bit of post processing), the other features highlighted in yellow seem legit.
From a neutral standpoint its hard to outright claim that these data were generated synthetically.

Let us compare with the original values.



	ID	Age	Height	Weight	Year	Sex_F	Sex_M	Season_Summer	Season_Winter	Medal_Bronze	Medal_DNW	Medal_Gold	Medal_Silver
1691	938	26.0	173.0	74.0	1980	0	1	0	1	0	1	0	0
198666	99766	23.0	186.0	71.0	2004	0	1	1	0	0	1	0	0
62196	31850	28.0	173.0	62.0	1976	1	0	1	0	0	1	0	0
52772	27093	25.0	190.0	85.0	2016	0	1	1	0	0	1	0	0
93934	47545	22.0	183.0	84.0	2006	0	1	0	1	1	0	0	0

We don't want the values to be exactly equal, then there is no point of generating new data (and not just copying entire datapoints). We want it to be slightly different, but not so much.

Apart from ID and Year (which didn't come out correct maybe due to lack of encoding, the rest of the values are are predicted mostly correct.

Link to video(youtube): https://youtu.be/heyQle3TwKg

Link to video(Google Drive):
https://drive.google.com/file/d/1y3N7WWg0WaDcsevN6BA4wF9VRkpz5AoP/view?usp=sharing

Shortcomings:

1. In the 2nd algorithm, we use SD as a measure of which columns to fill in first. Most of the time it will choose categorical features first (as they have lower SD). We should give more weightage to numerical features compared categorical features.
2. The algorithms does not support multi class classifiers.

3. Data cleaning/ preprocessing is not generalized.
4. Does not perform dimensionality reduction and cross validation.
5. No post processing.

Merits:

1. The algorithms are generalized and can distinguish between categorical and numerical features and apply the desired method of training, much like what Auto ML does.
2. Decent performance on missing values.
3. Generalized calculation of RMSE/ Accuracy depending of feature type and whether it contains missing values.
4. Can generate (semi decent) synthetic data.

***