Predicting Olympic Medal Winners

A Data Mining Approach to Olympic Medal Winners

Project Introduction

Goal:

Predicting whether or not Olympians will win medal based on the historical dataset on the modern Olympic Games,including all the Games from Athens 1896 to Rio 2016.

Problem Statement:

Out of the various features provided in data set, project will use some or all to build a model which will predict if a Olympian will win a medal or not.

Project Workflow

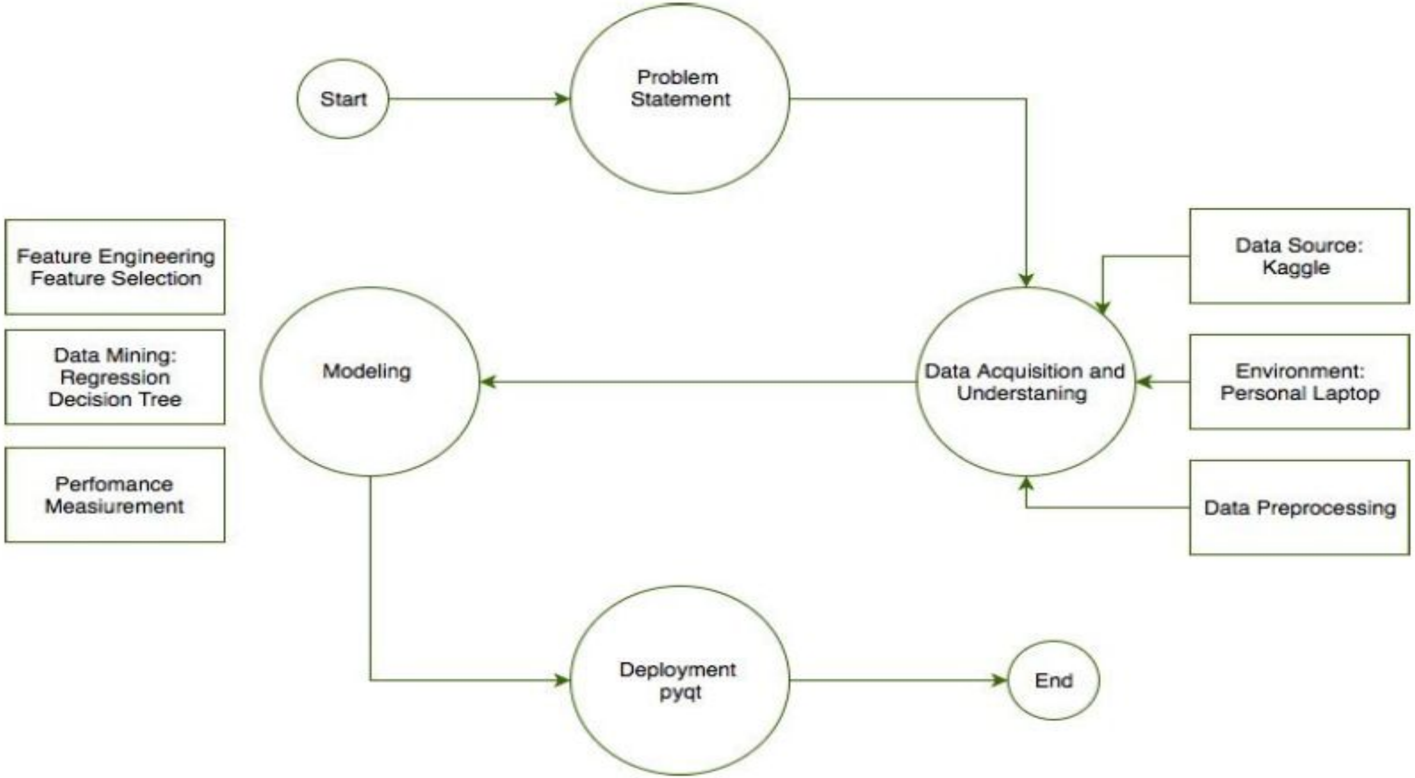


Figure 1: project workflow

Data

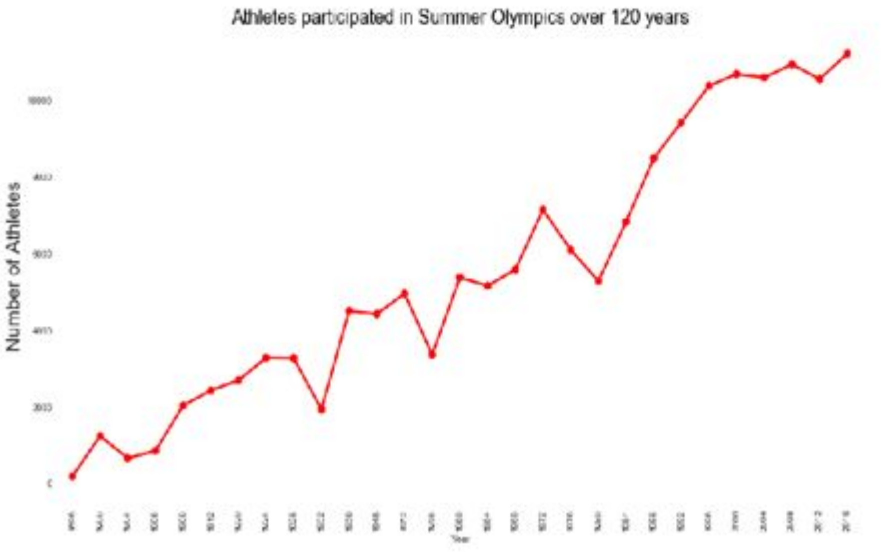
This is a historical dataset on the modern Olympic Games, including all the Games from Athens 1896 to Rio 2016. The columns are:

1. ID - Unique number for each athlete
2. Name - Athlete's name
3. Sex - M or F
4. Age - Integer
5. Height - In centimeters
6. Weight - In kilograms
7. Team - Team name
8. NOC - National Olympic Committee 3-letter code
9. Games - Year and season
10. Year - Integer
11. Season - Summer or Winter
12. City - Host city
13. Sport - Sport
14. Event - Event
15. Medal - Gold, Silver, Bronze, or NA

Data Insight -- Participation (3-1)

- # of athletes participated over years

Summer

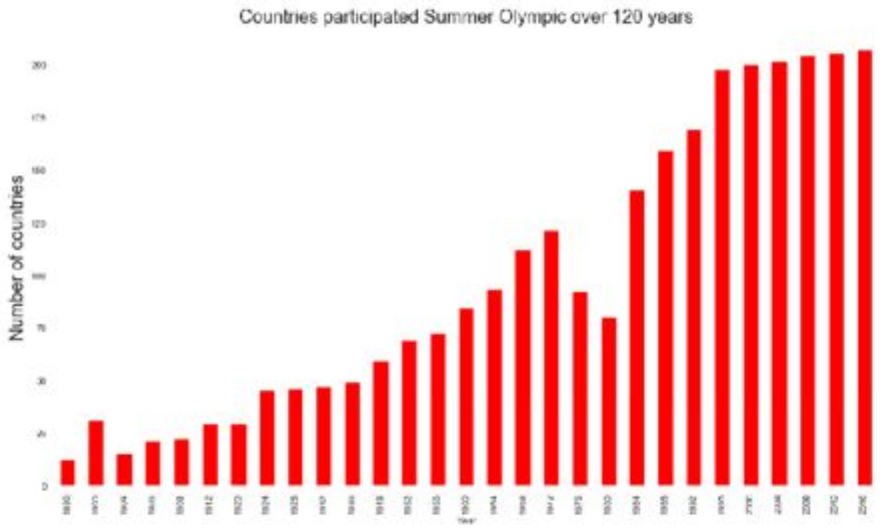


Winter

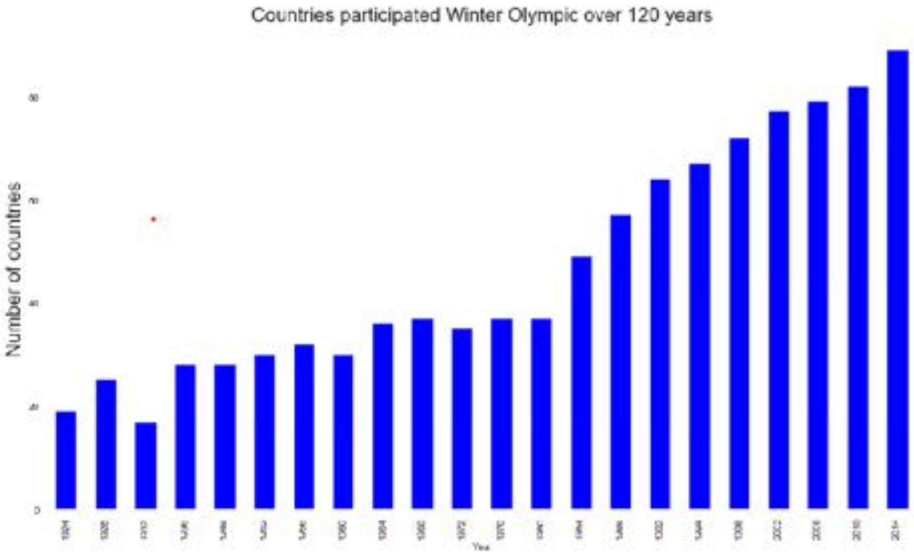
Data Insight -- Participation (3-2)

- # of countries participated over years

Summer



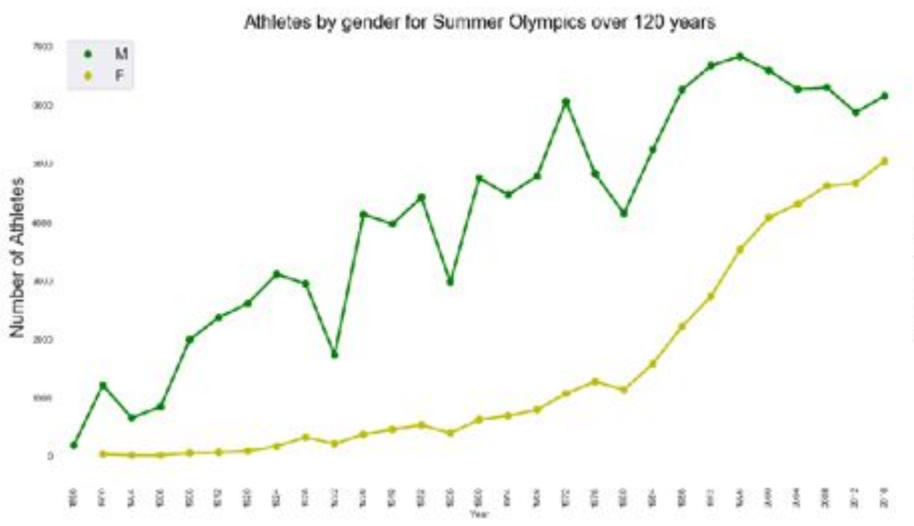
Winter



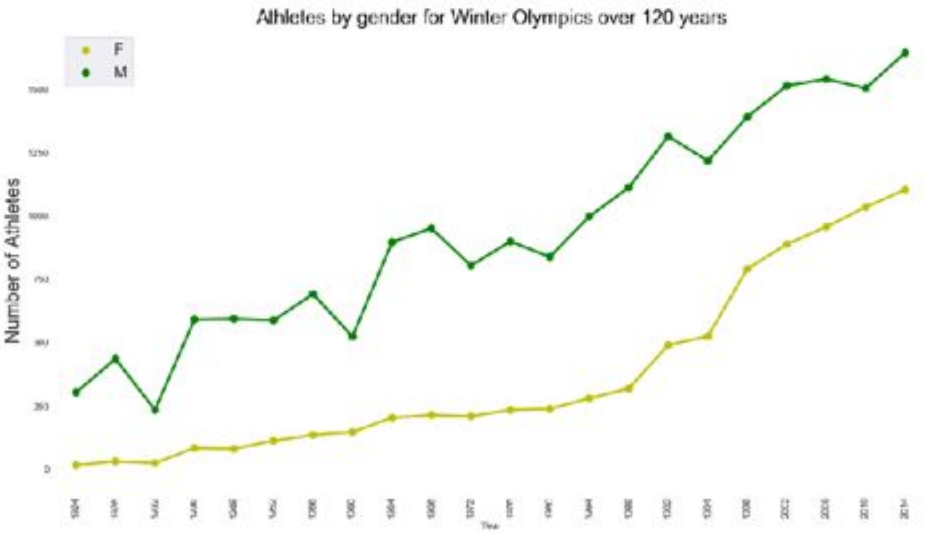
Data Insight -- Participation (3-3)

- Participation by gender

Summer

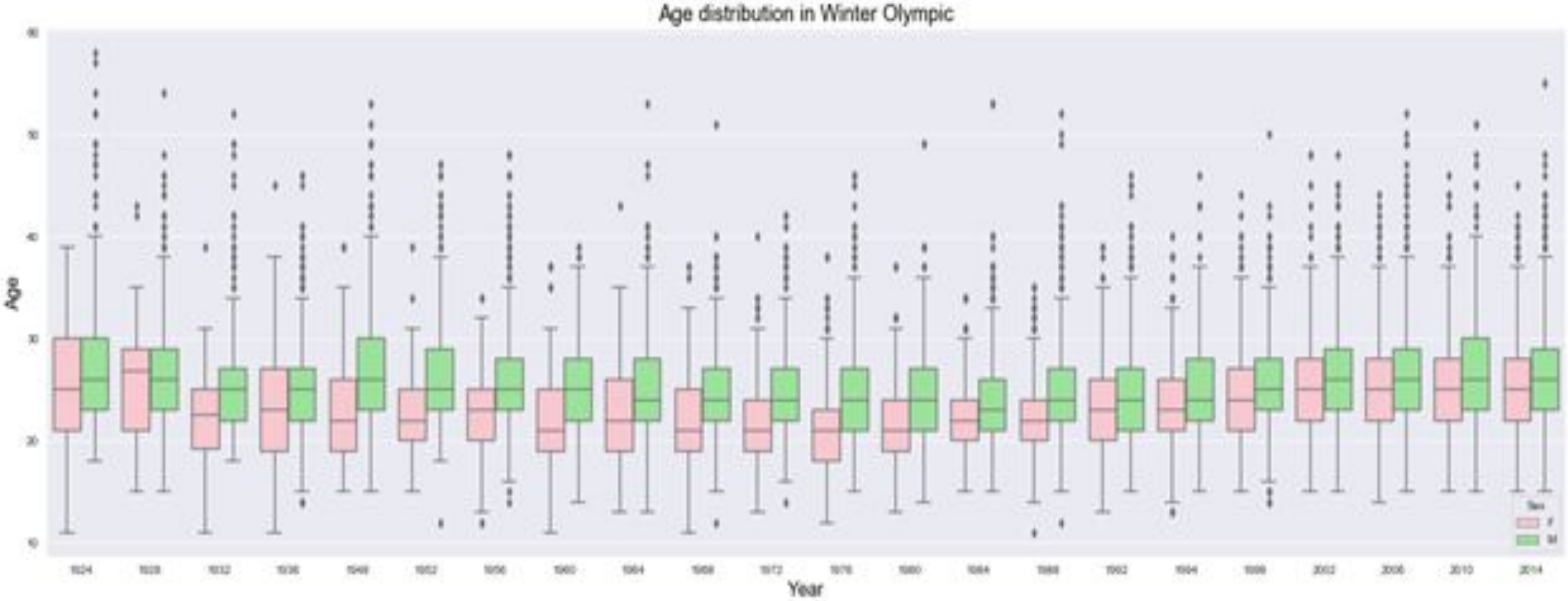


Winter



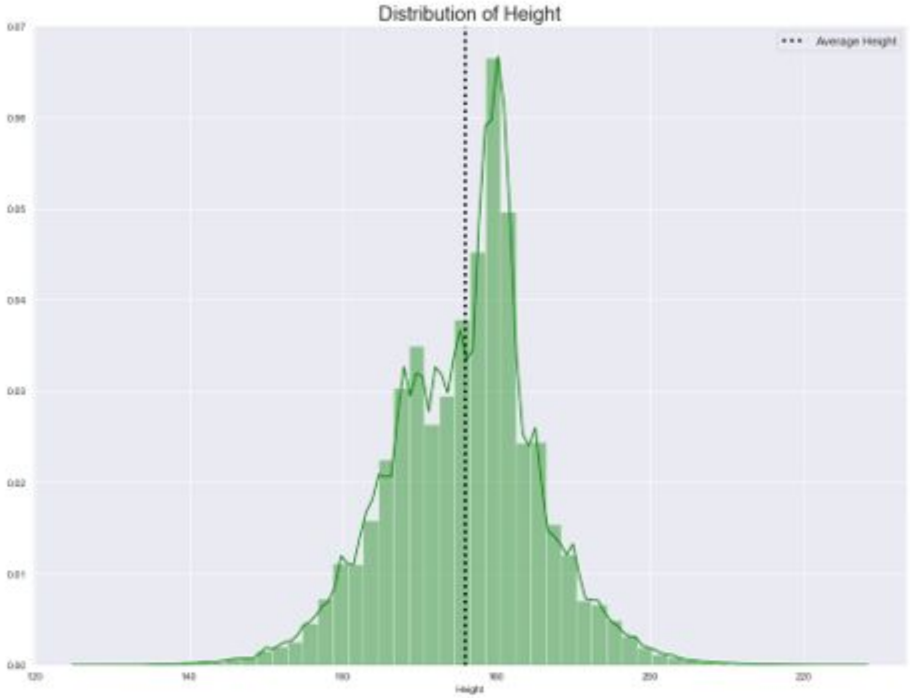
Data Insight -- Athletes (2-1)

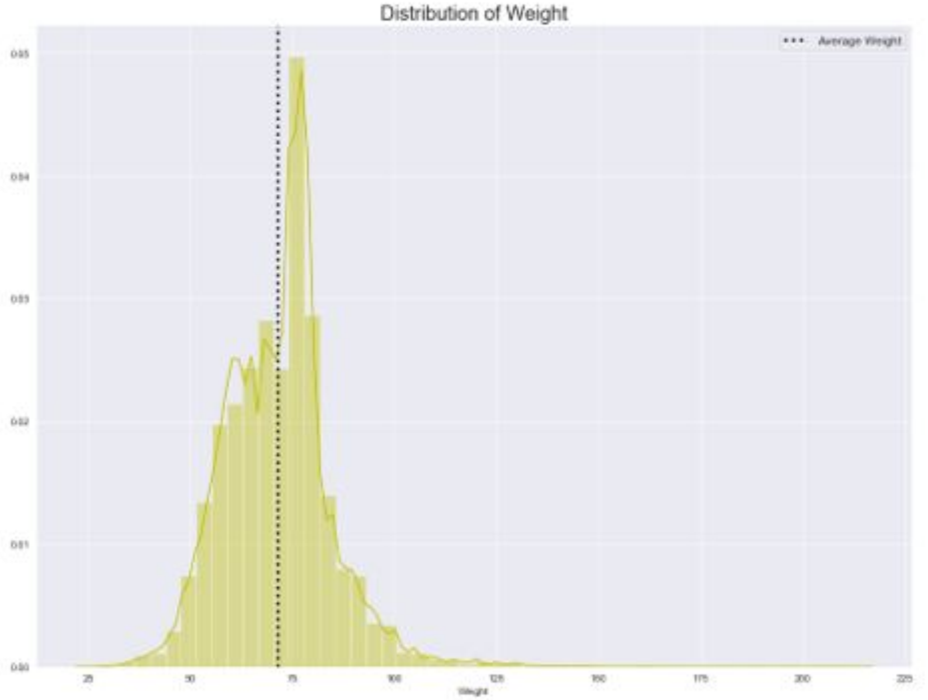
- Age Distribution



Data Insight -- Height and Weight (2-2)

- Height and Weight Distribution

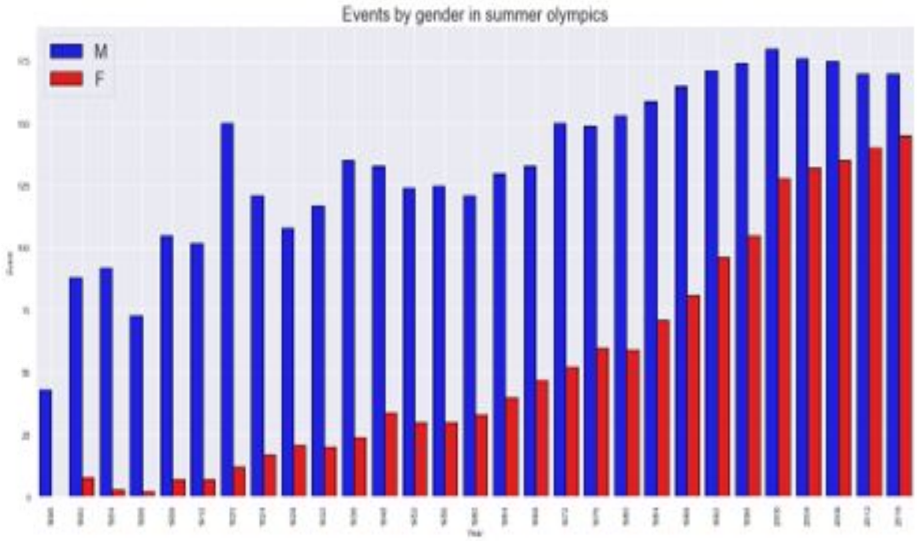




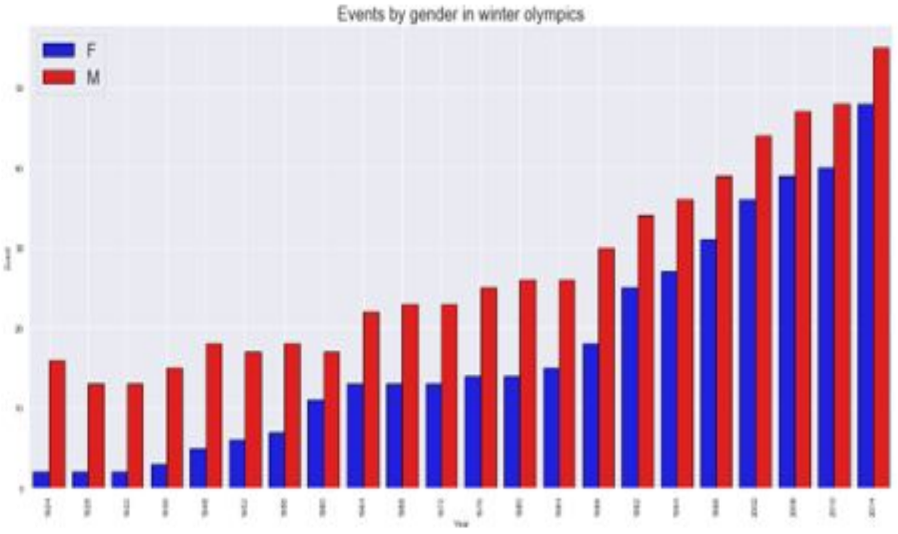
Data Insight -- Sport (2-1)

- Events by gender over years

Summer



Winter



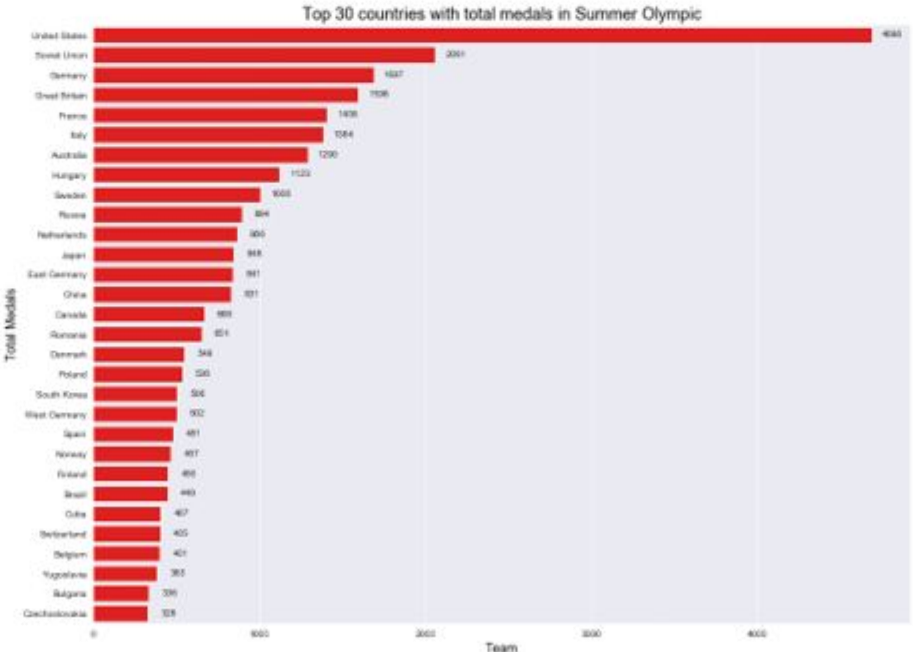
Data Insight -- Sport (2-1)

- Events by sport over years

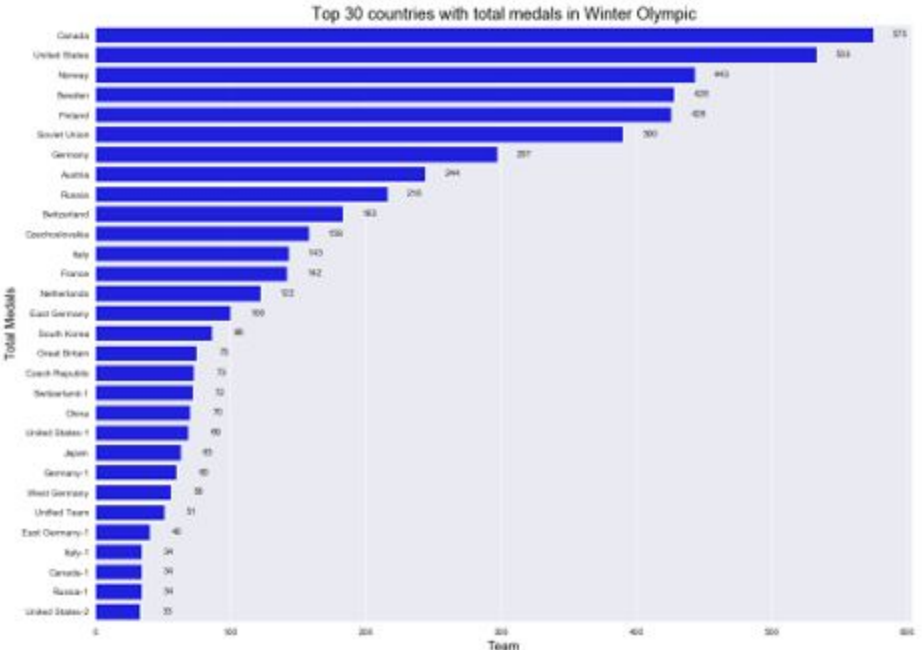
Data Insight -- Medal (2-1)

- Top 30 countries with the maximum number of medals

Summer

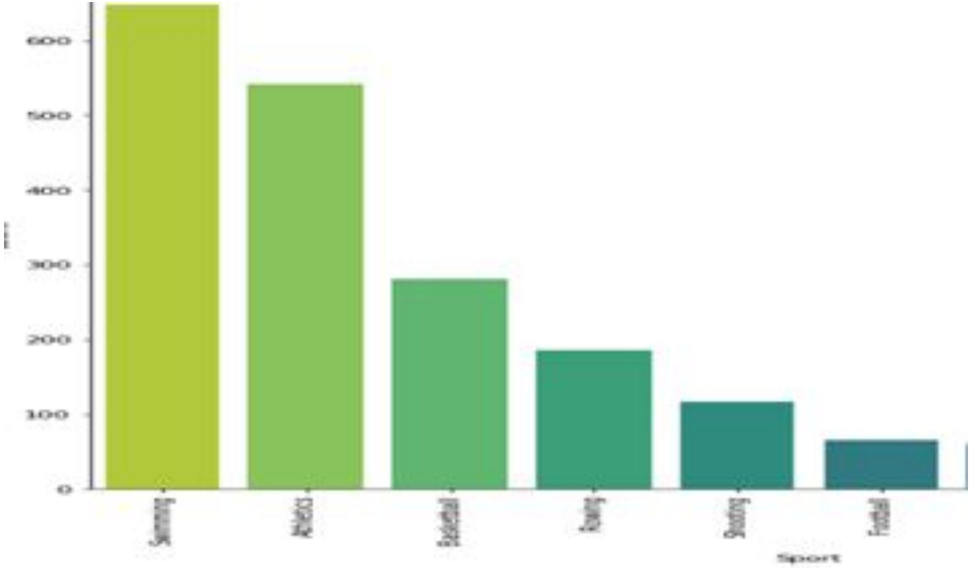


Winter



Data Insight -- Medal (2-2)

- Gold received USA



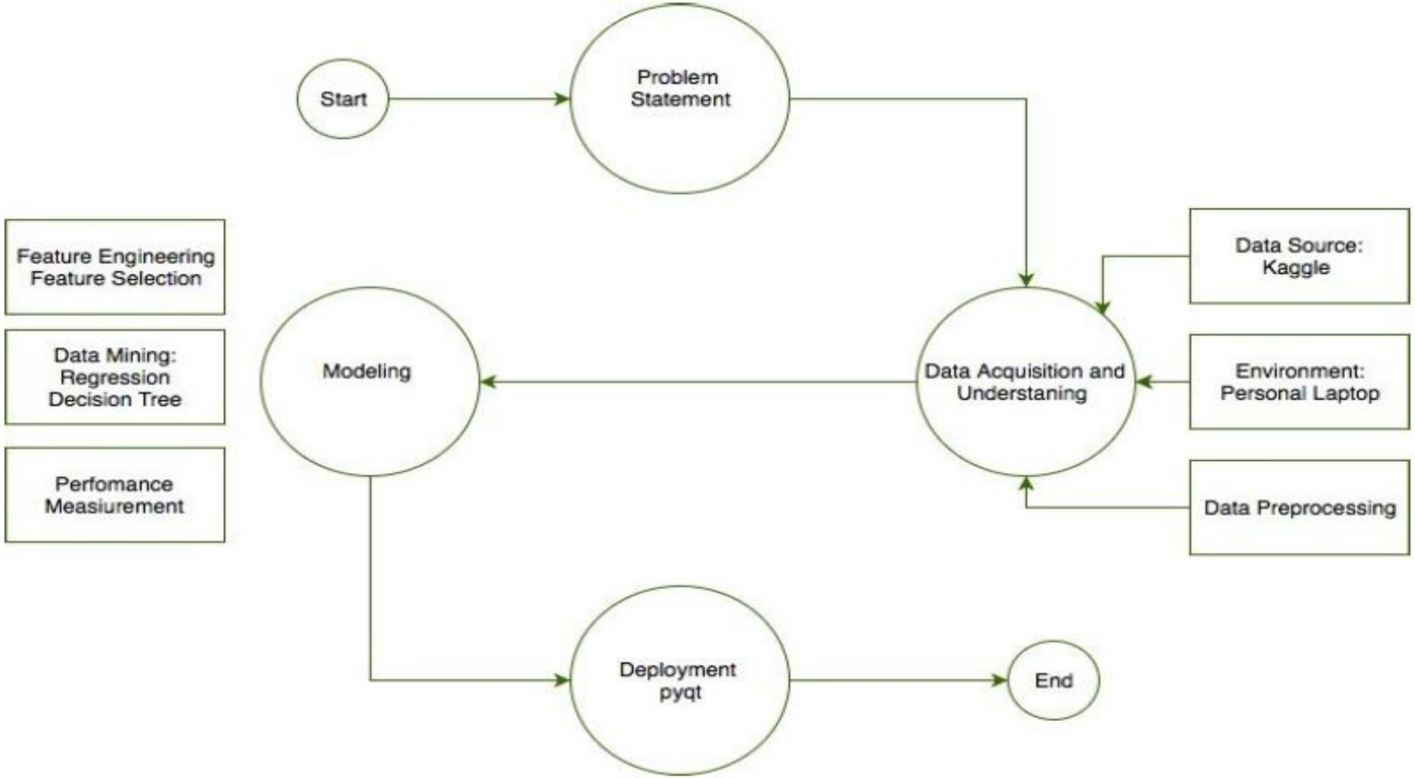
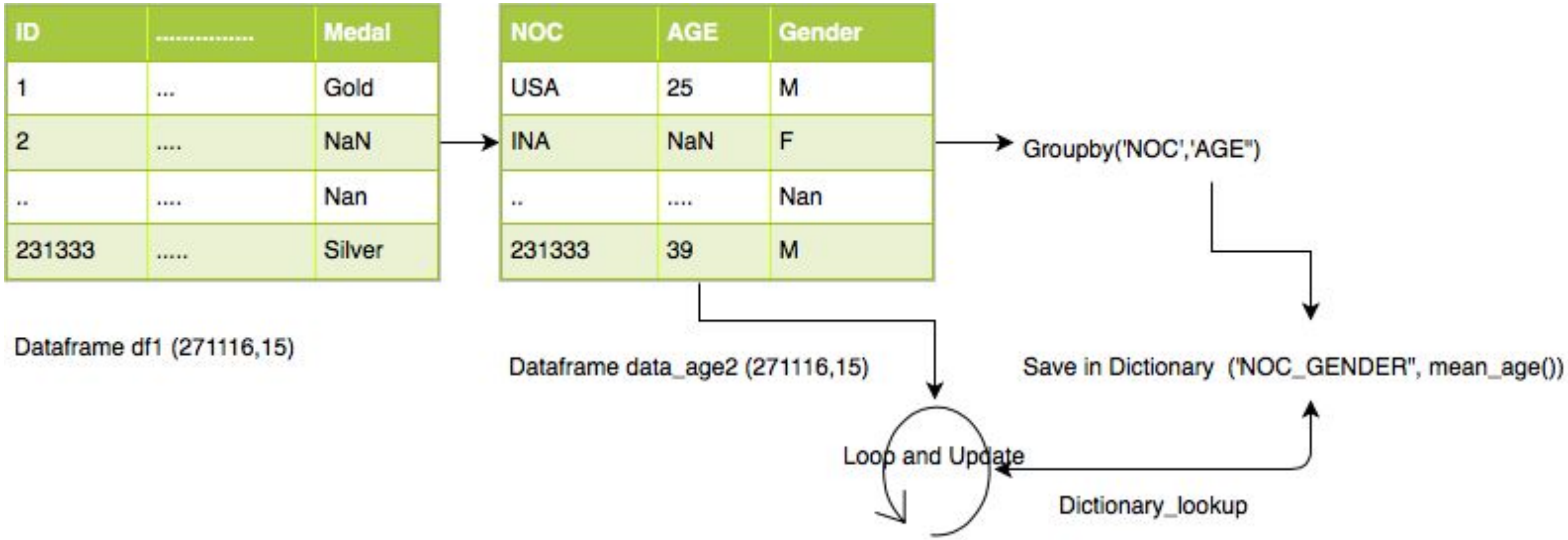


Figure 1: project workflow

Missing Value:

Shape	271116	15
	Missing Data	Percentage
ID	0	0
Name	0	0
Sex	0	0
Age	9474	3.49444518
Height	60171	22.1938211
Weight	62875	23.1911802
Team	0	0
NOC	0	0
Games	0	0
Year	0	0
Season	0	0
City	0	0
Sport	0	0
Event	0	0
Medal	231333	85.3262072

Data pre-processing: Age



Data pre-processing: Height and Weight

```
#Height Update
Similar process as age:
data_height = df1[['NOC','Height','Sex']].copy()
data_height = data_height.groupby(['NOC','Sex']).mean().reset_index()

d_height = defaultdict()
for index, row in data_height.iterrows():
    d_height[row['combined']]=row['Height']

#Weight Update:

data_weight = df1[['NOC','Weight','Sex']].copy()
data_weight = data_weight.groupby(['NOC','Sex']).mean().reset_index()

d_weight = defaultdict()
for index, row in data_weight.iterrows():
    d_weight[row['combined']]=row['Weight']
```

Result of all Update (column and Missing Value) :
Age: 13
Height 108
weight : 257

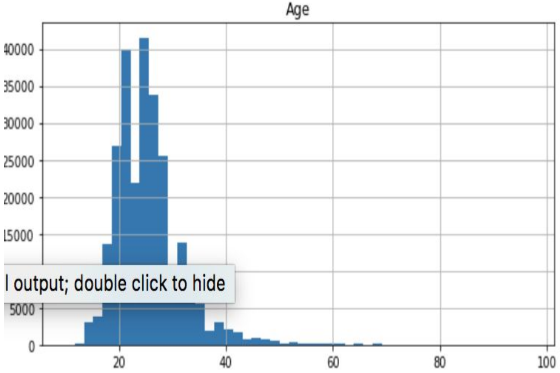
Final Preprocessing:

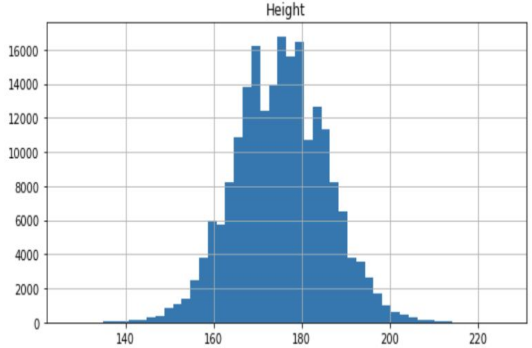
```
#find average of updated data and update the rest of NaN value

mean_age = df1.Age.mean()

mean_height = df1.Height.mean()

mean_weight = df1.Weight.mean()

print(mean_age,mean_height,mean_weight)
```

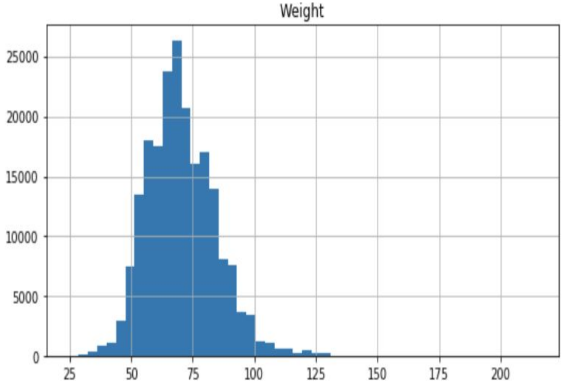
Loop and update (expensive process)

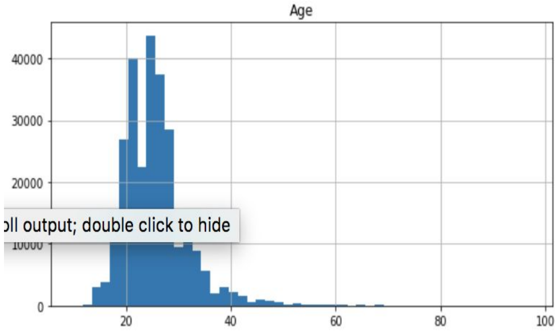
Result of Final Update (column and Missing Value) :
Age: 0
Height 0
weight : 0

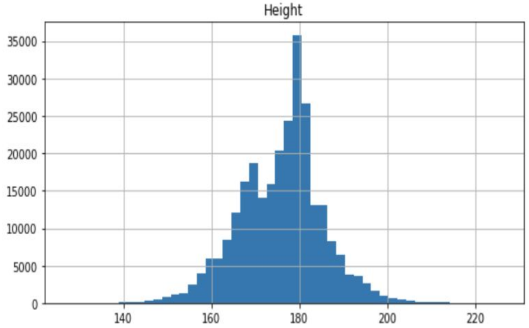
Age, Height, Weight Distribution

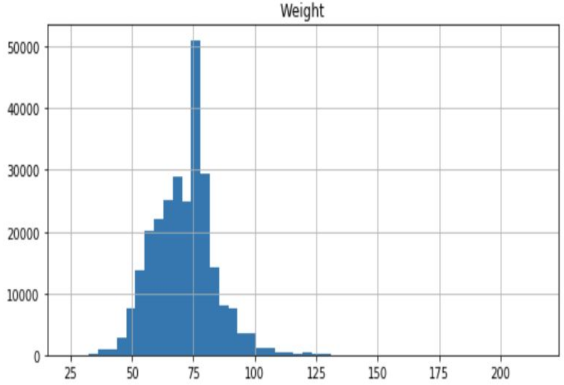












Feature Selection

```
df1 = df1.drop(['ID', 'Name','Games','Sport'], axis=1)

#removed features that were not relevant
```

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.000000	80.000000	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	0
1	2	A Lamusi	M	23.0	170.000000	60.000000	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	0
2	3	Gunnar Nielsen Aaby	M	24.0	181.527607	77.607492	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	0
3	4	Edgar Lindenau Aabye	M	34.0	181.527607	77.607492	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	1
4	5	Christine Jacoba Aaftink	F	21.0	185.000000	82.000000	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	0

Encoding (for categorical Data)

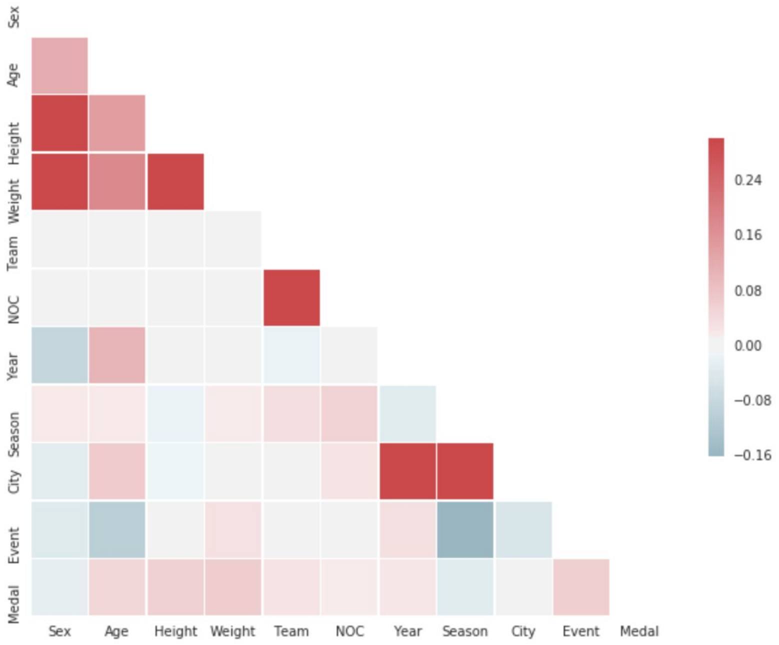
```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

# LabelEncoder
le = LabelEncoder()
# apply "le.fit_transform"
df_encoded = df1.apply(le.fit_transform)
```

Result:

```
print(df_endcoded.head(10))
```

	Sex	Age	Height	Weight	Team	NOC	Year	Season	City	Event	Medal
0	1	21	353	413	60	39	0	0	3	69	0
1	1	16	185	148	60	39	10	0	6	237	0
6	0	30	398	427	222	138	0	1	0	357	0
7	0	30	398	427	222	138	0	1	0	353	0
8	0	40	398	427	222	138	1	1	5	357	0
9	0	40	398	427	222	138	1	1	5	353	0
10	1	45	402	339	342	203	0	1	0	124	0
11	1	45	402	339	342	203	0	1	0	131	0
12	1	45	402	339	342	203	0	1	0	126	0
13	1	45	402	339	342	203	0	1	0	130	0

Correlation



TakeAway:
Medal is highly depended on :
Age
Height
Weight
Team
NOC
Year
Event

Model

1st Run:

LogisticRegression

Predicted values were only Zero

`print(np.unique(Y_predict)) [0]`

Issue:

Imbalanced Data Set

Fix: Upsamling and Downsampling

Upscaling

```
print(df_encoded.Medal.value_counts())
from sklearn.utils import resample
df_majority = df_encoded[df_encoded.Medal==0]
df_minority = df_encoded[df_encoded.Medal==1]

# Upsample minority class

df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=105435,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class

df_upsampled = pd.concat([df_majority, df_minority_upsampled])
```

LogisticRegression - Upscaling

```
0    105435
1     16781
Name: Medal, dtype: int64
1    105435
0    105435
Name: Medal, dtype: int64
0.5842733166676829
             precision    recall  f1-score   support

          0       0.58      0.58      0.58     31441
          1       0.58      0.59      0.58     31820

avg / total       0.58      0.58      0.58     63261

[0 1]
```

Downscaling

```
#Downscaling
print(df_encoded.Medal.value_counts())

from sklearn.utils import resample
df_majority = df_encoded[df_encoded.Medal==0]
df_minority = df_encoded[df_encoded.Medal==1]

# Upsample minority class
df_majority_downsampled = resample(df_majority,
                                 replace=True,      # sample with replacement
                                 n_samples=16781,    # to match majority class
                                 random_state=123) # reproducible results
# Combine majority class with upsampled minority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])
```

LogisticRegression - Downscaling

```
0    105435
1     16781
Name: Medal, dtype: int64
1    16781
0    16781
Name: Medal, dtype: int64
0.5817477546503214
             precision    recall  f1-score   support

          0       0.59      0.59      0.59      5008
          1       0.59      0.59      0.59      5061

avg / total       0.59      0.59      0.59     10069

[0 1]
```

RandomForestClassifier: downscaling

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Train model
clf_4 = RandomForestClassifier()
clf_4.fit(X_train_std, y_train)

# Predict on training set
pred_y_4 = clf_4.predict(X_train_std)

# Is our model still predicting just one class?
print( np.unique( pred_y_4 ) )

# How's our accuracy?
print( accuracy_score(y_train, pred_y_4) )

# What about AUROC?
prob_y_4 = clf_4.predict_proba(X_train_std)
prob_y_4 = [p[1] for p in prob_y_4]

print( roc_auc_score(y_train, prob_y_4) )
predict_random = clf_4.predict(X_test_std)
print(classification_report(y_test,predict_random))
```

```
[0 1]
0.49597775350084417
0.816217959583181
             precision    recall  f1-score   support

          0       0.73      0.78      0.75      5008
          1       0.77      0.71      0.74      5061

avg / total       0.75      0.75      0.74     10069
```

Conclusion

Best Result with:

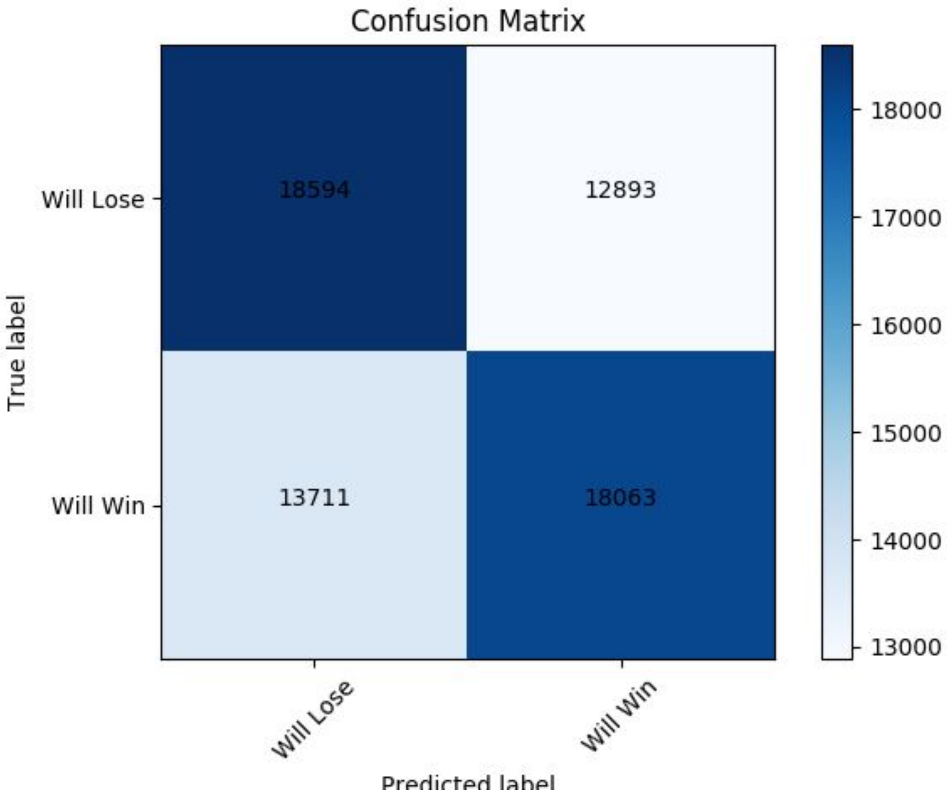
- StandardScaler
- Upsampling
- Random Forest Classifier

Improvements/Next Step

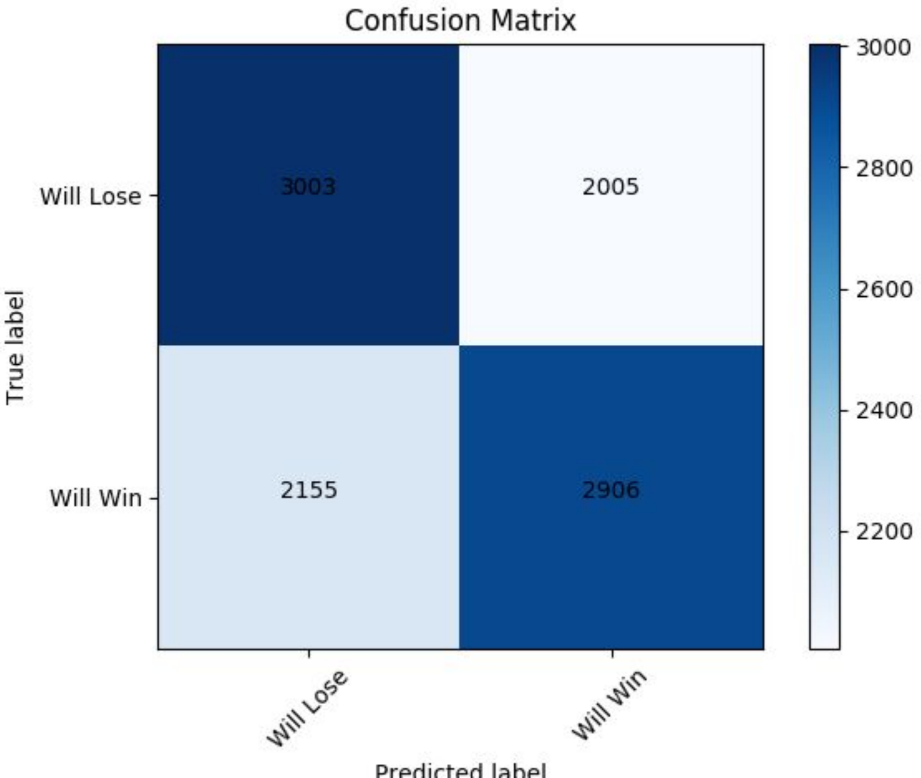
- Make loops efficient if possible
- Filter data to most recent events
- Categorical Target

Questions?

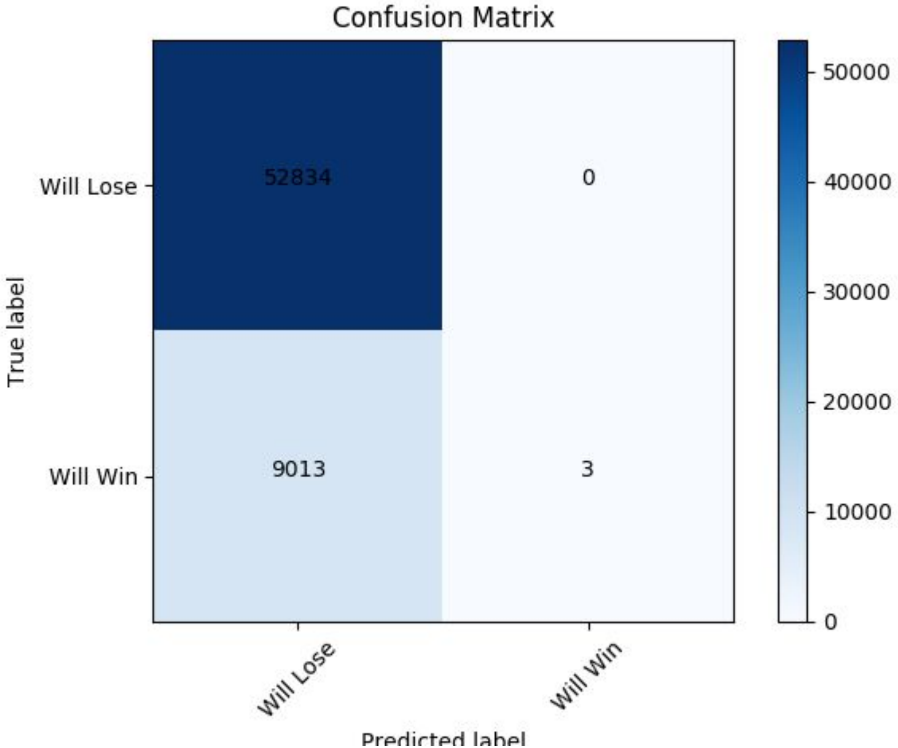
Logistic Regression w/ Preprocessing



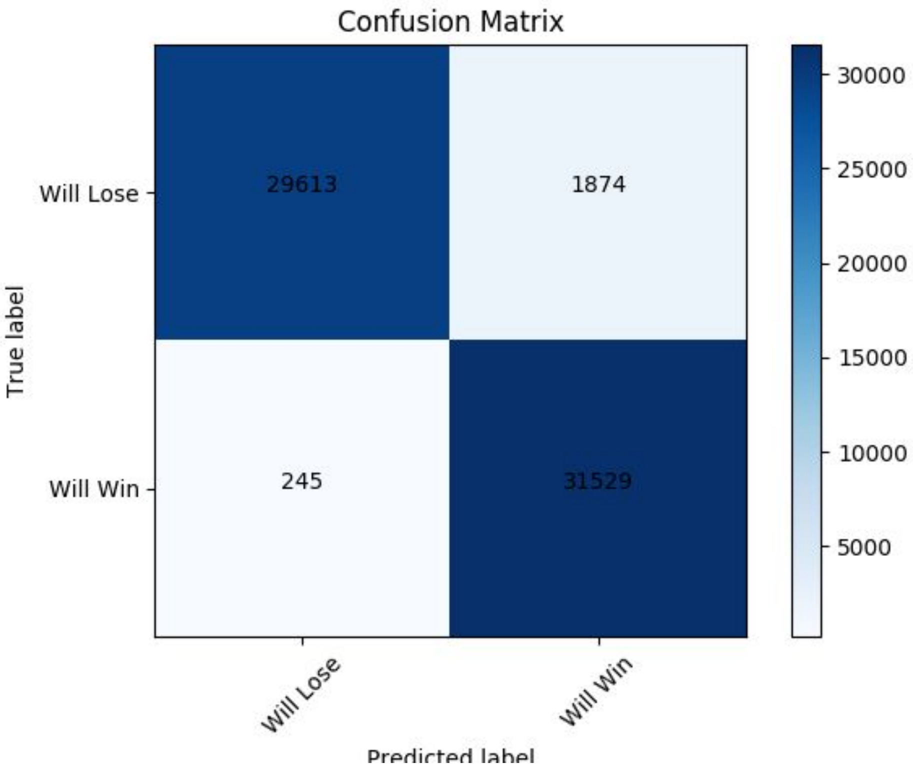
Logistic Regression w/ Preprocessing (Downsample)



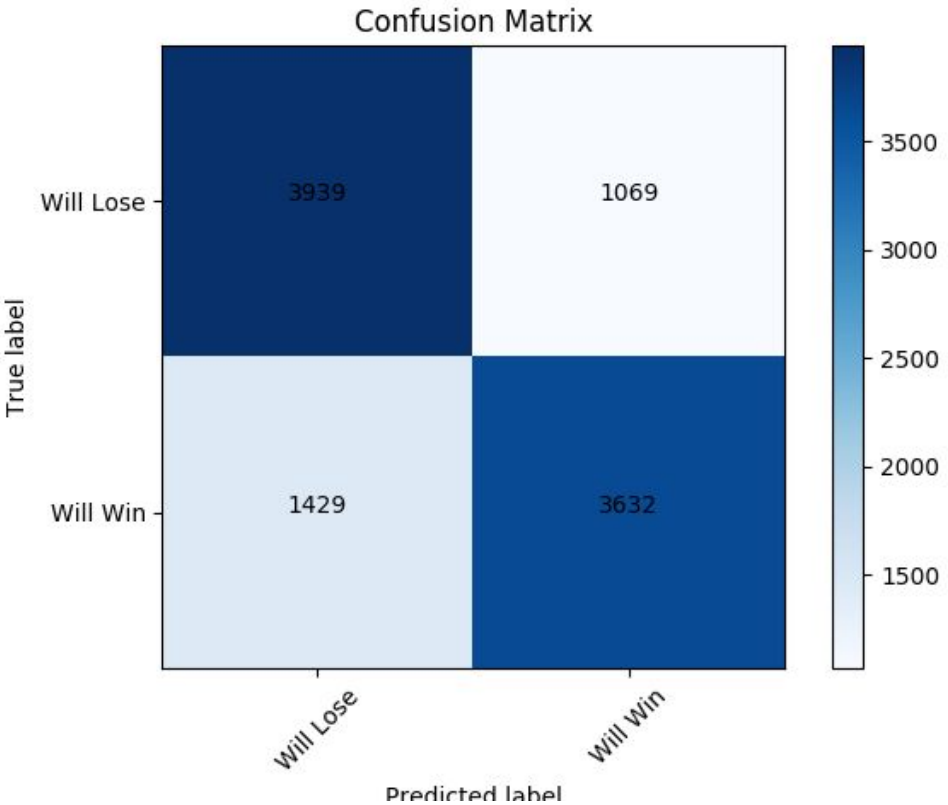
Logistic Regression w/ no Preprocessing



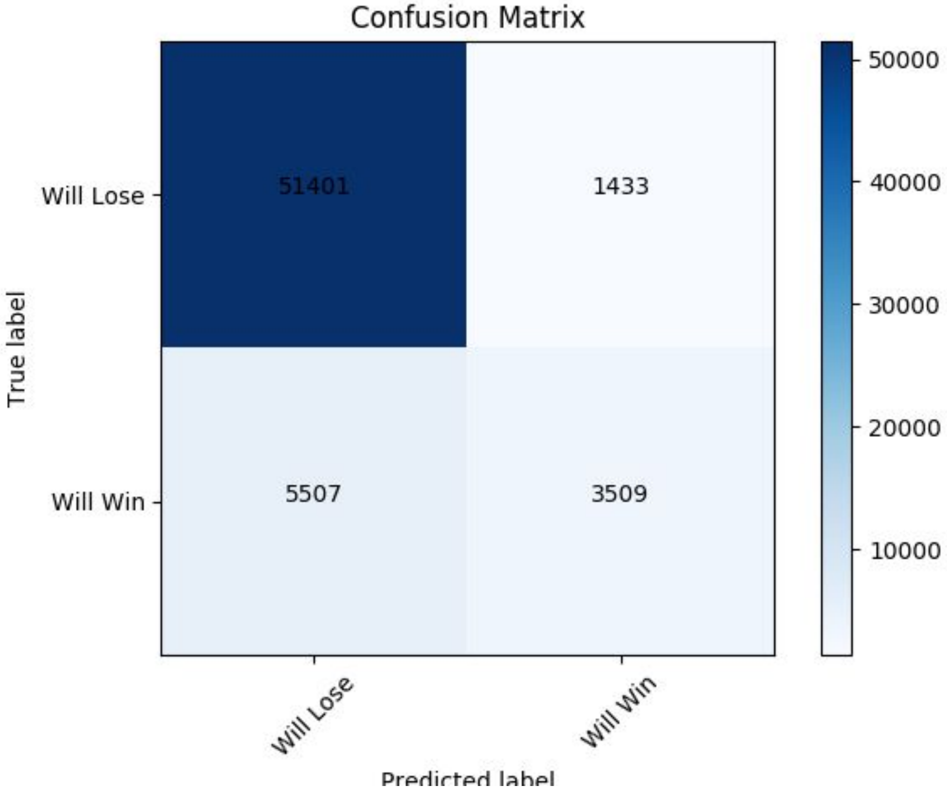
RF w/ Preprocessing



RF w/ Preprocessing (Down)



RF w/ no Preprocessing



Questions ???