FIFA 22 Players

Phase 1

The purpose of this project is to mine the players dataset from FIFA 22 and create a model for determining the best position for a player depending on the feat features.

The dataset contains information, that we can create a model for predicting growth of a player based on all the current values of the attributes/ skills.

The dataset was downloaded from https://www.kaggle.com/stefanoleone992/fifa-22-players-dataset

The data was scraped from sofifa.com. There are 110 features and 19239 registered players. So we can consider this dataset as the whole universe of players for the present.

The dataset contains attributes related to different soccer related abilities as well as some other features, i.e. value, wage, nationality, club, contract, etc. Some of the attributes are numerical, and some are categorical. Some of the attributes are ordinal, for example, the attribute "work_rate" contains equal to three values, low, medium, and high. Also, there are some missing values. The column "loan_date_end" is the only column with all the values are missing. Also, there are some attributes, such as goalkeeping attributes, that are available only for goalkeepers, or some of the attributes, such as defensive attributes and Attribute's score of zones from position related attributes.

Import libraries

```
# Import libraries
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_regression import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor, AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor, BaggingRegressor, VotingRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.model_selection import cross_val_score, GridSearchCV, RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics.cluster import silhouette_score, adjusted_rand_score
from sklearn.metrics.cluster import contingency_matrix
```

Import CSV file

```
df = pd.read_csv('./fifa22.csv')
df.shape, df.head(3).T, df.tail(3).T, df.sample(3).T
```

```
(19239, 110)
                  0                   1                   2  \
sofifa_id    158023              20801              190871
player_url  https://...  https://...  https://...
long_name   Lionel ...  Cristiano ...  Neymar da ...
...               ...                ...                ...
mentality_...       ...                ...                ...
mentality_...       ...                ...                ...
mentality_...       
```

```
# Variable and correlation
cor_target <- sort(cor(train[,-c(1,ncol(train))], train[,ncol(train)]),
                   decreasing = TRUE)
cor_target_df <- data.frame(Variable = names(cor_target),
                            Correlation = cor_target)
rownames(cor_target_df) <- NULL; cor_target_df
cor_target_df[cor_target_df$Correlation > 0.5, ]; cor_target_df[cor_target_df$Correlation < -0.5, ]
cor_target_df[abs(cor_target_df$Correlation) > 0.5, ]
```

	Variable	Correlation
1	OverallQual	0.8237819
2	GrLivArea	0.7156591
3	GarageCars	0.6404092
4	GarageArea	0.6233168
5	TotalBsmtSF	0.6135805
6	1stFlrSF	0.6058522
7	FullBath	0.5607150
8	TotRmsAbvGrd	0.5337232
9	YearBuilt	0.5228962
10	YearRemodAdd	0.5071010
11	GarageYrBlt	0.4866421
12	MasVnrArea	0.4772368
13	Fireplaces	0.4669288
14	BsmtFinSF1	0.3864198
15	LotFrontage	0.3351799
16	WoodDeckSF	0.3244134
17	OpenPorchSF	0.3158562
18	2ndFlrSF	0.3193042
19	HalfBath	0.2844076
20	LotArea	0.2638439
21	BsmtFullBath	0.2271226
22	BsmtUnfSF	0.2141676
23	BedroomAbvGr	0.1682865
24	ScreenPorch	0.1114037
25	PoolArea	0.0925594
26	MoSold	0.0464592
27	3SsnPorch	0.0446414
28	BsmtFinSF2	0.0050390
29	BsmtHalfBath	-0.0168215
30	MiscVal	-0.0206437
31	LowQualFinSF	-0.0256064
32	YrSold	-0.0289923
33	OverallCond	-0.0778558
34	MSSubClass	-0.0842848
35	EnclosedPorch	-0.1281993
36	KitchenAbvGr	-0.1359487

0
0
0

Country
Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Cluster 7
Cluster 8
Cluster 9
Cluster 10
Cluster 11
Cluster 12
Cluster 13
Cluster 14
Cluster 15
Cluster 16
Cluster 17
Cluster 18
Cluster 19
Cluster

```
# Set features, this will vary by year or topic.
features = ['Tournament', 'Seed', 'Conference', 'Tournament']
# Create a copy of the dataset for ease of use.
df = df.dropna(subset=['Tournament'])
print(df.shape, df.dtypes)
df.head()
```

```
# Split the data into training and testing groups and print them.
X_train, X_test, y_train, y_test = train_test_split(df, df, random_state=1)
```

```
# Set a baseline accuracy.
y = df.Tournament.value_counts().sort_values(ascending=False)
baseline = y[1]/y.sum()
print('Baseline accuracy:', baseline)
```

```
Baseline accuracy: 0.5000000000000000
```

```
# Import the Decision Tree classifier.
from sklearn.tree import DecisionTreeClassifier
```

```
# Instantiate the classifier.
dt = DecisionTreeClassifier(max_depth=None,
                            min_samples_split=2,
                            min_weight_fraction_leaf=0.0,
                            max_features=None,
                            random_state=None,
                            max_leaf_nodes=None,
                            min_impurity_decrease=0.0,
                            class_weight=None,
                            ccp_alpha=0.0)
# Fit the classifier to the training data.
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None,
                       max_features=None, 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, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# Print the feature importances.
for zip_object in zip(X_train.columns, dt.feature_importances_):
    print(zip_object)
```

```
('Tournament', 0.0)
('Seed', 0.0)
('Conference', 0.0)
('Tournament', 0.0)
```

```
# Create the tree export function.
from sklearn.tree import plot_tree
plot_tree(decision_tree=dt,
          feature_names=X_train.columns,
          class_names=['Tournament', 'Seed', 'Conference', 'Tournament'],
          filled=True,
          rounded=True,
          max_depth=None,
          precision=3)
```

```
[Text(0.500, 0.500, 'Tournament <= 0.0\n gini = 0.667\n samples = 3\n value = [1, 1, 1]\n class = Tournament')]
```



```
# Import the Random Forest classifier.
from sklearn.ensemble import RandomForestClassifier
```

```
# Instantiate the classifier.
rf = RandomForestClassifier(n_estimators=100,
                            criterion='gini',
                            max_depth=None,
                            min_samples_split=2,
                            min_weight_fraction_leaf=0.0,
                            max_features='auto',
                            max_leaf_nodes=None,
                            min_impurity_decrease=0.0,
                            min_impurity_split=None,
                            bootstrap=True,
                            oob_score=False,
                            n_jobs=None,
                            random_state=None,
                            verbose=0,
                            warm_start=False,
                            class_weight=None,
                            ccp_alpha=0.0,
                            max_samples=None)
# Fit the classifier to the training data.
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None,
                       max_features='auto', max_leaf_nodes=None,
                       max_samples=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=100, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
```

```
# Print the feature importances.
for zip_object in zip(X_train.columns, rf.feature_importances_):
    print(zip_object)
```

```
('Tournament', 0.0)
('Seed', 0.0)
('Conference', 0.0)
('Tournament', 0.0)
```

```
# Create the tree export function.
from sklearn.tree import plot_tree
plot_tree(decision_tree=rf[0],
          feature_names=X_train.columns,
          class_names=['Tournament', 'Seed', 'Conference', 'Tournament'],
          filled=True,
          rounded=True,
          max_depth=None,
          precision=3)
```

```
[Text(0.500, 0.500, 'Tournament <= 0.0\n gini = 0.667\n samples = 2\n value = [1, 1, 1]\n class = Tournament')]
```



```
# Import the Gradient Boosting classifier.
from sklearn.ensemble import GradientBoostingClassifier
```

```
# Instantiate the classifier.
gb = GradientBoostingClassifier(loss='deviance',
                                learning_rate=0.1,
                                n_estimators=100,
                                subsample=1.0,
                                criterion='friedman_mse',
                                min_samples_split=2,
                                min_weight_fraction_leaf=0.0,
                                max_depth=3,
                                min_impurity_decrease=0.0,
                                min_impurity_split=None,
                                init=None,
                                random_state=None,
                                max_features=None,
                                verbose=0,
                                max_leaf_nodes=None,
                                warm_start=False,
                                presort='deprecated',
                                validation_fraction=0.1,
                                n_iter_no_change=None,
                                tol=0.0001)
# Fit the classifier to the training data.
gb.fit(X_train, y_train)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse',
                           init=None, learning_rate=0.1, loss='deviance',
                           max_depth=3, max_features=None,
                           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=100, n_iter_no_change=None,
                           presort='deprecated', random_state=None,
                           subsample=1.0, tol=0.0001, validation_fraction=0.1,
                           verbose=0, warm_start=False)
```

```
# Print the feature importances.
for zip_object in zip(X_train.columns, gb.feature_importances_):
    print(zip_object)
```

```
('Tournament', 0.0)
('Seed', 0.0)
('Conference', 0.0)
('Tournament', 0.0)
```

```
# Create the tree export function.
from sklearn.tree import plot_tree
plot_tree(decision_tree=gb[0],
          feature_names=X_train.columns,
          class_names=['Tournament', 'Seed', 'Conference', 'Tournament'],
          filled=True,
          rounded=True,
          max_depth=None,
          precision=3)
```

```
[Text(0.500, 0.500, 'Tournament <= 0.0\n friedman_mse = 0.000\n samples = 3\n value = [0.000, 0.000, 0.000]\n class = Tournament')]
```


Model evaluation

```
# Import the 3 ML's Cross Score Model, Cross Score Model, Cross Score Model, Cross Score Model.
from sklearn.model_selection import cross_val_score, cross_val_score, cross_val_score, cross_val_score
```

```
# Instantiate dt, rf, gb, dummy.
dummy = DummyClassifier(strategy='most_frequent',
                        random_state=None,
                        constant=None)
```

```
# Set 10 folds.
folds = 10
```

```
# Print the accuracy.
print('Dummy Score:', cross_val_score(dummy, X, y, cv=folds).mean())
print('Decision Tree Score:', cross_val_score(dt, X, y, cv=folds).mean())
print('Random Forest Score:', cross_val_score(rf, X, y, cv=folds).mean())
print('Gradient Boosting Score:', cross_val_score(gb, X, y, cv=folds).mean())
```

```
Dummy Score: 0.5000000000000000
Decision Tree Score: 0.5000000000000000
Random Forest Score: 0.5000000000000000
Gradient Boosting Score: 0.5000000000000000
```

Hyperparameter Tuning

No need to run the next 4 blocks as it takes nearly 30 mins to execute!!!

```
# Import GridSearchCV.
# CV.fit_get_params().
```

```
# Create an instantiated grid.
grid = {'n_estimators': [100, 200, 300, 500],
        'criterion': ['gini', 'entropy', 'log2'],
        'max_depth': [None, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'max_features': ['auto', 'sqrt', 'log2']}
```

```
# Create an instantiated GridSearchCV object.
gscv = GridSearchCV(estimator=rf,
                    param_grid=grid,
                    scoring=None,
                    n_jobs=None,
                    iid=False,
                    refit=True,
                    cv=None,
                    verbose=0,
                    pre_dispatch='2*n_jobs',
                    error_score=nan,
                    return_train_score=False)
```

```
# Fitting the grid to the training data.
gscv.fit(X_train, y_train, groups=None, **fit_params)
```

```
# Print the best parameters.
print('Best Parameters:', gscv.fit(X_train, y_train).best_params_)
```

```
Best Parameters: {'criterion': 'log2', 'max_features': 'sqrt', 'max_depth': 8, 'n_estimators': 300}
```

Run from here

```
# Instantiate model using the found parameters.
rf = RandomForestClassifier(n_estimators=300, criterion='log2',
                            max_depth=8, max_features='sqrt', random_state=1)
```

```
# Fit the classifier to the training data.
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(ccp_alpha=0.0, class_weight=None,
                       criterion='log2', max_depth=8,
                       max_features='sqrt', max_leaf_nodes=None,
                       max_samples=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=300, n_jobs=None, oob_score=False,
                       random_state=1, verbose=0, warm_start=False)
```

```
# Print the feature importances.
for zip_object in zip(X_train.columns, rf.feature_importances_):
    print(zip_object)
```

```
('Tournament', 0.0)
('Seed', 0.0)
('Conference', 0.0)
('Tournament', 0.0)
```

```
# Create the tree export function.
from sklearn.tree import plot_tree
plot_tree(decision_tree=rf[0],
          feature_names=X_train.columns,
          class_names=['Tournament', 'Seed', 'Conference', 'Tournament'],
          filled=True,
          rounded=True,
          max_depth=None,
          precision=3)
```

```
[Text(0.500, 0.500, 'Tournament <= 0.0\n gini = 0.667\n samples = 2\n value = [1, 1, 1]\n class = Tournament')]
```


Model evaluation

```
# Import the 3 ML's Cross Score Model, Cross Score Model, Cross Score Model, Cross Score Model.
from sklearn.model_selection import cross_val_score, cross_val_score, cross_val_score, cross_val_score
```

```
# Instantiate dt, rf, gb, dummy.
dummy = DummyClassifier(strategy='most_frequent',
                        random_state=None,
                        constant=None)
```

```
# Set 10 folds.
folds = 10
```

```
# Print the accuracy.
print('Dummy Score:', cross_val_score(dummy, X, y, cv=folds).mean())
print('Decision Tree Score:', cross_val_score(dt, X, y, cv=folds).mean())
print('Random Forest Score:', cross_val_score(rf, X, y, cv=folds).mean())
print('Gradient Boosting Score:', cross_val_score(gb, X, y, cv=folds).mean())
```

```
Dummy Score: 0.5000000000000000
Decision Tree Score: 0.5000000000000000
Random Forest Score: 0.5000000000000000
Gradient Boosting Score: 0.5000000000000000
```

Dummy Score:	0.5000
Decision Tree Score:	0.5000
Random Forest Score:	0.5000
Gradient Boosting Score:	0.5000

```
print('Dummy Score:', cross_val_score(dummy, X, y, cv=folds).mean())
print('Decision Tree Score:', cross_val_score(dt, X, y, cv=folds).mean())
print('Random Forest Score:', cross_val_score(rf, X, y, cv=folds).mean())
print('Gradient Boosting Score:', cross_val_score(gb, X, y, cv=folds).mean())
```

The model has only been already scored after the division into classes.

The model has only been already scored after the division into classes.