Progress Report: {European Soccer}

Brian Pascente
Carson Holscher
Jacob Richardson
Justin O'Donnell

Dataset Overview:

- **Test.csv (csv)**: This is the merged csv dataset from last week. This is done by getting the player and the player attribute table from the SQLite dataset and merging it with the csv dataset based on the name of the player..
    - **Number of examples**: 1220 players with 72 columns

What We Are Predicting:

- **Target Variable**: Player performance (goals scored).
- **Problem Type**: This is a regression problem, as the goal is to predict continuous values (goals).

Summary:

Brian Pascente:

Our overall task this week was to explore different types of regression models for our dataset, and see which one works the best.

I was tasked to create a Lasso regression model for our dataset. I started off by getting rid of the outliers in the dataset since for my mini project I forgot to do that and it gave me bad results, so I thought it would be beneficial to do so in our project.

```
# Remove outliers from y using IQR method
Q1 = y.quantile(0.25)
Q3 = y.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
non_outliers = (y >= lower_bound) & (y <= upper_bound)
X = X[non_outliers]
y = y[non_outliers]
```

Lasso Regression (https://github.com/rhit-pascenby/CSSE415-EurpoeanSoccer/blob/main/Lasso.ipynb):
A GridSearchCV was used to search over different values of the regularization parameter alpha for Lasso Regression. This parameter controls how strongly the model penalizes large coefficients, helping to prevent overfitting and perform feature selection.

```
param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100],
}
```

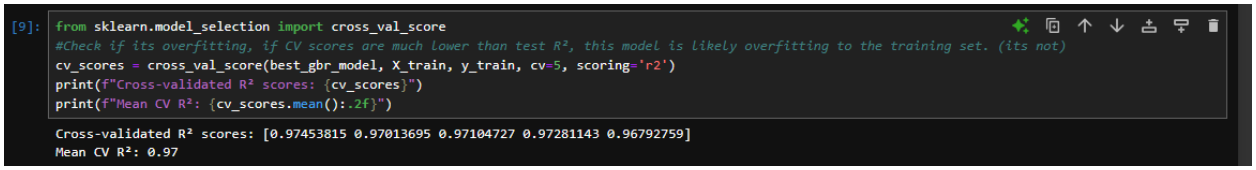
Results:
Best Alpha: 0.001
R^2: 0.61
MSE: 0.01
MAE: 0.06
Best Lasso Prediction: 0.07, Actual: 0.07
Worst Lasso Prediction: 0.05, Actual: 0.40
The Lasso regression model achieved a moderate R^2 of 0.61, indicating it explains 61% of the variance in the target variable, with low average errors (MSE: 0.01, MAE: 0.06). While it made some accurate predictions, the worst prediction error (0.35 difference) suggests limitations in this model.

Ridge Regression (https://github.com/rhit-pascenby/CSSE415-EurpoeanSoccer/blob/main/Ridge.ipynb):
I felt like I did not do enough work by just doing the Lasso, so I decided to do the Ridge Regression and gradient boosting as well. Ridge Regression was applied to predict the target variable (goals) using regularized linear modeling.. A GridSearchCV was used to tune the regularization strength (alpha) across several values, selecting the best one based on 5-fold cross-validated negative mean squared error.
Results:
Ridge Regression Performance (After Hyperparameter Tuning):
Best Alpha: 100
R^2: 0.56
MSE: 0.01
MAE: 0.07
Best Ridge Prediction: 0.03, Actual: 0.03
Worst Ridge Prediction: 0.04, Actual: 0.40
While the model performed well on some predictions (e.g., predicting 0.03 for an actual value of 0.03), its worst prediction (0.04 vs. 0.40) suggests limitations in capturing complex relationships in the data.

Gradient Boosting Regression (https://github.com/rhit-pascenby/CSSE415-EurpoeanSoccer/blob/main/Gradient.ipynb):
A Gradient Boosting Regressor (GBR) was trained using scikit-learn's ensemble methods. A grid search was performed to tune three key hyperparameters: n_estimators (number of trees), learning_rate (contribution of each tree), max_depth (tree complexity).

```
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}
```

Results:
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
R^2: 0.97
MSE: 0.00
MAE: 0.01
Best GBR Prediction: -0.00, Actual: 0.00
Worst GBR Prediction: 0.33, Actual: 0.48
This model works very well with our data by looking at the results.
To confirm the model's generalization ability, 5-fold cross-validation was conducted on the training set using R^2 as the scoring metric. The cross-validated R^2 scores were very consistent, averaging 0.97, which closely matched the test R^2 score. This strongly suggests that the model is not overfitting and generalizes well across unseen data.



Carson Holscher:

I created a random forest regression model for our dataset

Random Forest Regression:
(https://github.com/rhit-pascenby/CSSE415-EurpoeanSoccer/blob/main/RandomForest.ipynb):
Grid search was used to find the optimal number of regressors to use in the model.
Cross validation was used to find the optimal number of features.

In [13]:

```
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

B = np.arange(10,200,10)
grid = {'n_estimators':B}

rf = GridSearchCV(RandomForestRegressor(),param_grid=grid,return_train_score=True,n_jobs=-1)
rf.fit(X_train,y_train)

cv_results = cross_validate(rf,X_train,y_train,return_train_score=True)
R2_trainCV = cv_results['train_score'].mean()
R2_valid   = cv_results['test_score'].mean()
predictions = rf.predict(X_test)

mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

print('train R2 (CV) =',R2_trainCV,'  valid R2 =',R2_valid)
print()
R2_train = rf.score(X_train,y_train)
R2_test  = rf.score(X_test,y_test)
print('     train R2 =',R2_train,'    test R2 =',R2_test)
print('mse = ' + str(mse))
print('mae = ' + str(mae))
```

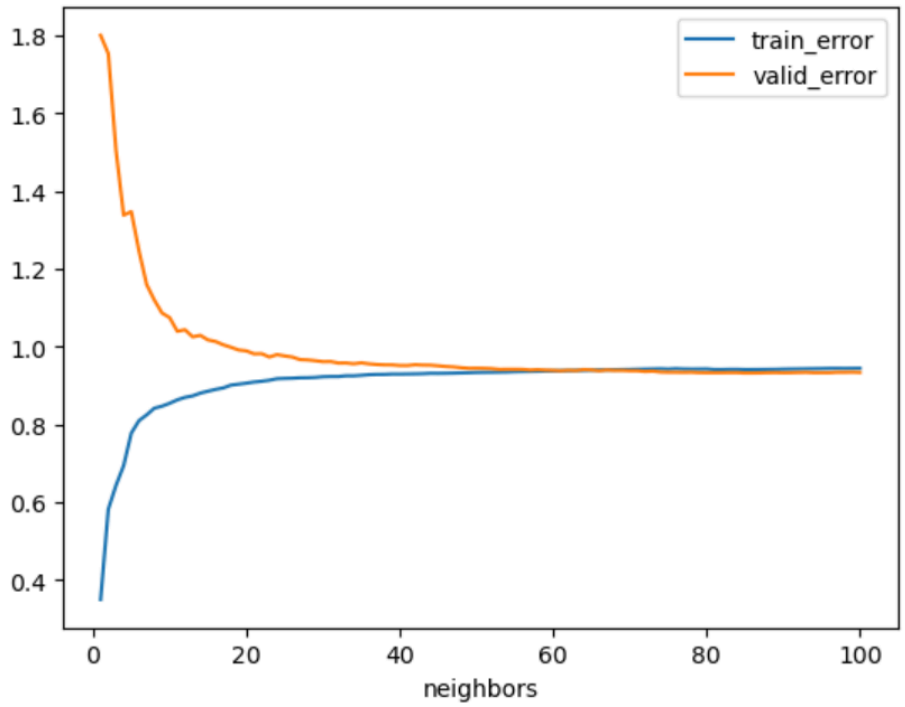
```
train R2 (CV) = 0.9850467657647355   valid R2 = 0.9006410241892511

     train R2 = 0.98388293913725     test R2 = 0.9227570587164161
mse = 0.07216240002290004
mae = 0.10240794986736662
```

The random forest regression model achieved a decent testing R^2 of 0.92, indicating it explains 92% of the variance in the target variable, with low average errors (MSE: 0.07, MAE: 0.1). Given the general accuracy of the predictions, I would say that random forests are a strong candidate for our model of choice

Jacob Richardson:

I created a KNN regression model for our dataset using grid search
Cross validation was used to find the best KNN hyperparameter



The R2 value for the optimal neighbors (K=85) was 0.093, which indicates KNN may not be the best predictor of goals scored from the features used:

```
features = ['height', 'age', 'assists', 'games_injured', 'award', 'current_value', 'highest_value']
target = 'goals'
```

```
K = np.arange(100)+1 #Grid Search
grid = {'n_neighbors':K}
```

```
knnCV.fit(X_train, y_train)
```

```
#Pull information regarding best parameter
ix = results['valid_error'].idxmin()
results.iloc[ix]
```

```
neighbors      85.000000
train_error     0.941191
valid_error     0.932369
Name: 84, dtype: float64
```

```
# Compute Test Results - R2
knn = KNeighborsRegressor(n_neighbors=85)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

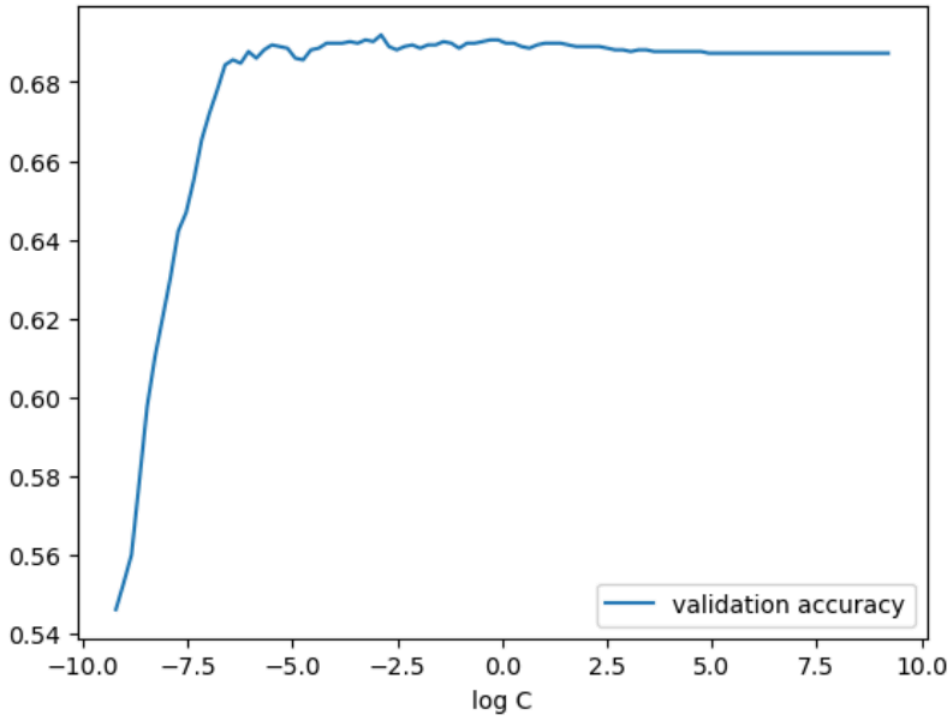
```
0.09321628929867187
```

Justin O’Donnell:

Coefficient Factor:
(https://github.com/rhit-pascenby/CSSE415-EurpoeanSoccer/blob/main/CoefficientFactorsGoals.ipynb)

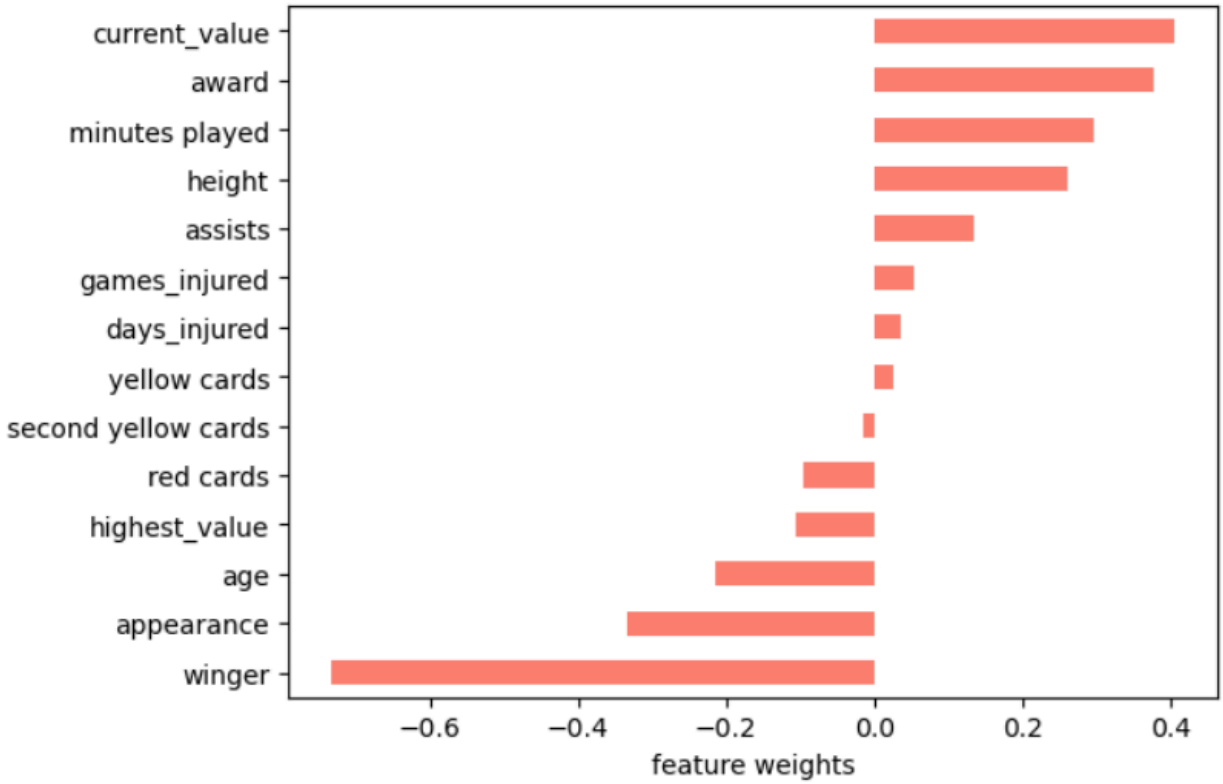
I focused on coefficient factors for goals. This is different from the other models as this is not a predictor but more an analysis of the statistics and how helpful they are for prediction. Especially, because we plan to use this for wins. Goals were a nice start as many of these coefficient factors should make sense. Such as winger being the largest negative coefficient factor as wingers play farther from the goal. I only analyzed ‘Attackers’.

We are also thinking about using this to analyze how teams should spend their money. For example, if we find a stronger correlation with stadium size than average team price to wins, which is doubtful but possible. Then the owners would know to prioritize making a bigger stadium over signing expensive players. For now, I am just focusing on the effects a player's statistics have on the number of goals they score.



This is the validation accuracy, which is helpful to know how quickly the model learned and determined the coefficient factors. After this I was able to obtain this graph of coefficient factors for attackers. I used a rating split of 0.33 goals which is a goal every three games, which turned out to be close to about a 60% bad and 40% good split. Which makes sense as there are much less bad strikers than there are good. The graph below shows the values for the coefficient

factors of each statistic.



You can see that the highest value is current_value, which is extremely intuitive because goals very directly correlate with wins and the number of wins a team has the player will be more valuable. Additionally, it is surprising that red and yellow cards have such a low correlation as they often are associated with sloppy play or being a bad player. Appearance having such a low correlation makes sense as the more games they play in it is harder to consistently make an impact. But because the best players play the most I would expect this value to be less negative.