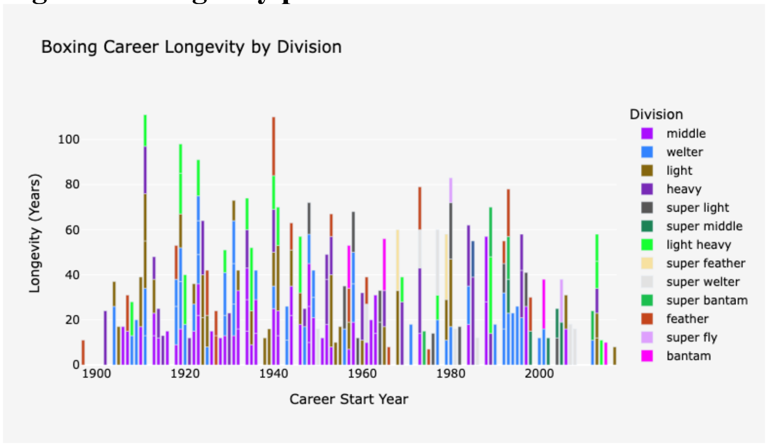
Jalen Buffert
Final Project Report
2 May 2025

Introduction

Professional boxing is a brutal sport of hit and don't get hit. Many athletes train their entire lives in pursuit of a top rank and a championship belt. Generally, a multitude of factors influence how successful a boxer's career will be. The bar chart below shows longevity and career start year color coded by division. This shows us which division of fighters have the longer career and what era they are from.

Figure 1: Longevity per Division



Often, boxers with similar fighting styles have completely different career outcomes. A boxer's starting age significantly impacts career outcomes, including career longevity, win percentage, and title attainment, with early starters showing advantages in technical skill development but disadvantages in longevity. BoxRec.com provides comprehensive rankings for all male professional boxers. I extracted data from the top 225 boxers of all time and applied elastic net regression and extreme gradient boosted trees to predict how win percentage is affected by a variety of factors. After inspecting the coefficients across both models, I found that while the coefficient for starting age was meaningful, it was smaller than those for beginning year and longevity. This suggests that although starting age has a positive effect on win percentage, win percentage is more strongly influenced by the era in which a fighter began his career.

Methods

The original dataset from BoxRec included the rank, name, division, outcome of the last six fights, number of wins, losses, and draws, career length, and residence. With the intention of predicting the effect that starting year had on career outcomes, I needed to retrieve additional data and transform some of the existing variables. Excel was used to split the wins-losses-draws column to calculate win percentage. Longevity was extracted from career length by calculating the difference between beginning and ending years. Beginning year was also extracted from the career length field. What was interesting was that BoxRec did not provide the birth year of the fighters, so I initially had no way of knowing the age at which a fighter's career started. I used the Wikipedia package in Python to extract the birth year of each fighter. Some values were blank or inaccurate, so I included more search patterns to improve the extraction process. Eventually, I implemented Google to correct missing or inconsistent birth years and then calculated each fighter's starting age. From there, I encoded the division column and dropped features that were irrelevant to the analysis. Below is a table containing the final dataset prior to encoding and training the models.

Table 1: Intro to Data

Division	Beginning Year	Longevity	Birth Year	Starting Age	Win %
middle	1940	25	1921	19	0.87437186
welter	1923	13	1907	16	0.79710145
welter	1996	21	1977	19	1
middle	1913	13	1894	19	0.88039867

The first model I used was elastic net, as I wanted to clearly understand the effect of each feature on win percentage. A grid search was conducted but did not significantly improve the results. To compare my findings, I introduced a gradient boosted model to evaluate whether starting age, along with other features, remained a strong predictor of win percentage within a more complex, nonlinear model.

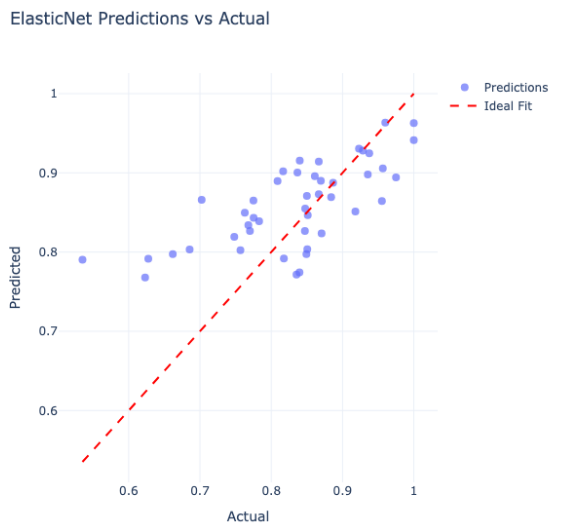
Results

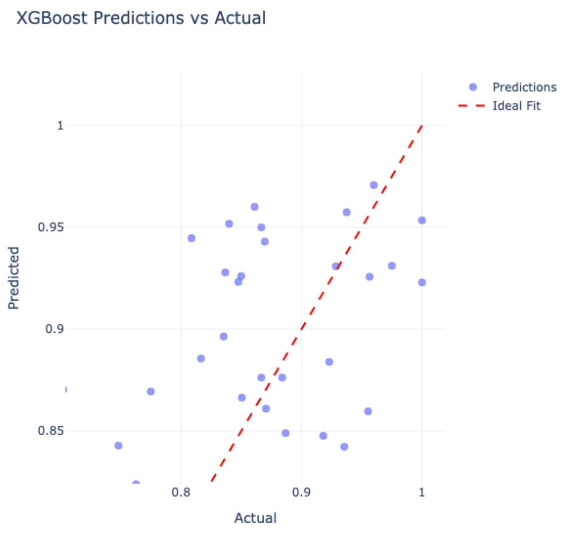
Table 2: Model Comparison

Model	R^2	RMSE	MAE
ElasticNet	0.391	0.079	0.061
XBGoost	0.294	0.085	0.068

The elastic net model outperformed the gradient boosted model across all evaluation metrics. With an R^2 of 0.391, the elastic net model may not appear to be a strong fit, but the gradient boosted model performed even worse. Gradient boosted models typically excel with large datasets, and this analysis was limited to only 225 observations. When comparing error metrics, the elastic net model also produced lower values, further indicating better overall performance.

Figure 2 and Figure 3: Scatter Plot of Predictions



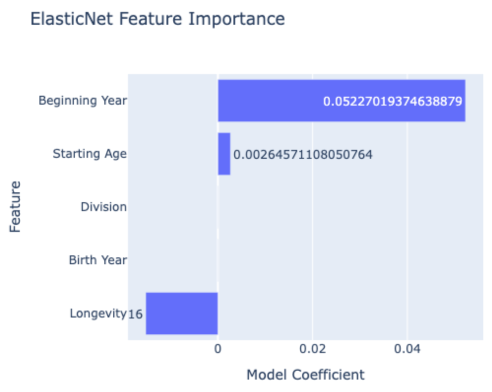


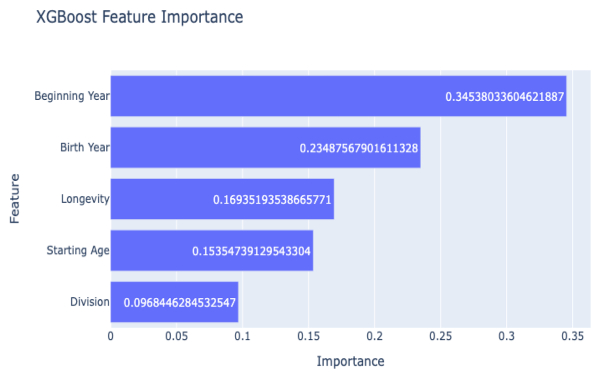
The scatter plot above shows the predicted values vs the actual values. The gradient boosted model has much more variance than the elastic net.

Aside from prediction accuracy, the goal was to identify which features were most influential in predicting win percentage. Below are the feature importance charts for each model. The most important feature across both models was beginning year, with a coefficient of 0.0522 in the elastic net model. Interestingly, longevity had a negative coefficient in elastic net, suggesting that longer careers may slightly reduce win percentage. The coefficient for starting age was 0.00264, indicating a minimal effect compared to beginning year.

Although the gradient boosted model performed worse than elastic net overall, its feature importance rankings still offered valuable insights. Birth year emerged as the second most important feature in the gradient boosted model, whereas starting age ranked among the least important features.

Figure 4 and Figure 5: Feature Importance Charts





Discussion

Given that both the elastic net and gradient boosted models identified starting age as an important feature, and the elastic net model showed a positive coefficient for starting age, it's clear that starting age does influence professional win percentage. The younger a fighter starts, the better chance they must develop the skills necessary to succeed at the professional level. However, according to this analysis, starting age is not the most important factor in predicting a successful boxing career. Since beginning year emerged as the most important feature across both models, the era in which a fighter competes appears to have a stronger effect on their win percentage. This may suggest that the overall skill level of boxers has risen over time, making it more difficult to remain undefeated.

Birth year, on the other hand, was significant in the gradient boosted model but not in the elastic net model, making it less reliable as a predictor overall. Longevity is particularly interesting, elastic net showed a negative effect of longevity on win percentage. This implies that the longer a professional career lasts, the lower a fighter's win percentage tends to be. There are intuitive explanations for this: the more fights a boxer has, the more opportunities there are to lose. Additionally, longer careers can lead to more wear and tear on the body, which may reduce performance over time.

There were several obstacles during this project, primarily related to transforming the data from BoxRec into a usable format. As mentioned earlier, extracting birth dates was a multistep process that involved using Wikipedia and manual correction to ensure accuracy. This was necessary to calculate each fighter's starting age. Figure 5 is a scatterplot of longevity (y-axis) versus starting age (x-axis), which not only shows a negative relationship between starting age and career length, but also helps identify outliers caused by incorrect birthdates scraped from Wikipedia.

The dataset used for this project consisted of all-time great fighters to evaluate what factors influence win percentage over time. However, a potential improvement would be to use a dataset of only active fighters to better control for the effects of era. Additionally, since these statistics are based solely on professional careers, it would be valuable to explore how starting age in boxing in amateur years affects long-term career outcomes.

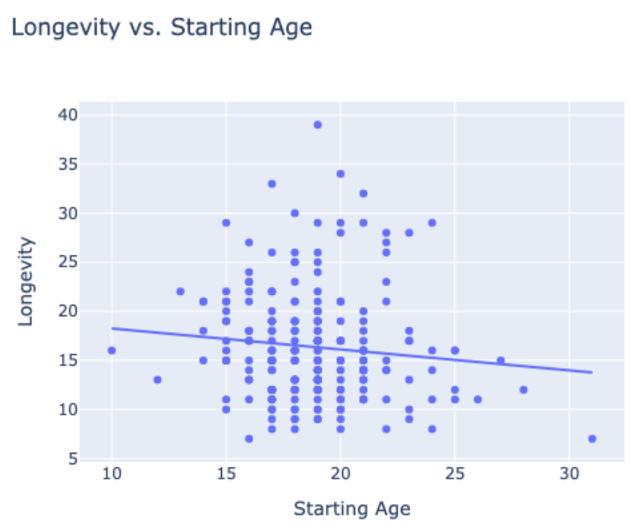
Citations

https://boxrec.com/en/ratings?r%5Brole%5D=box-pro&r%5Bsex%5D=M&r%5Bstatus%5D=&r%5Bdivision%5D=&r%5Bcountry%5D=&r_go=

I used ChatGPT to help me get the code to extract birth years from Wikipedia.

Figures

Figure 6:



Code

```
Import the necessary packages

# %%
import pandas as pd
import numpy as np
import os
import plotly.express as px
import plotly.graph_objects as go

# %% [markdown]
# Import csv containing boxing data that was cleaned on python and excel

# %%
box_d4 = pd.read_csv('box_d4.csv')
box_d4.dropna()


# %%
box_d4['Starting Age'].min(), box_d4['Starting Age'].max()

# %% [markdown]
# Initial Data Visualization using Plotly

# %%
fig = px.bar(
    box_d4,
    x="Beginning Year",
    y="Longevity",
    color="Division",
    hover_name="Name",
    title="Boxing Career Longevity by Division",
    labels={"Longevity": "Longevity (Years)", "Beginning Year": "Career Start Year"},
    color_discrete_sequence=px.colors.qualitative.Alphabet
)

fig.update_layout(
    plot_bgcolor='whitesmoke',
    paper_bgcolor='whitesmoke',
    font=dict(color='black'),
    bargap=0.3
)

fig.update_layout(bargap=0.3)
fig.show()

# %% [markdown]
# Using Wikipedia to find the birthdates of the fighters.

# %%
import wikipedia
import re

#Function for extracting birth year from Wikipedia summary
def get_birth_year(name):
    try:
        search_results = wikipedia.search(name)
        if not search_results:
            return None

        page_title = search_results[0]
        summary = wikipedia.summary(page_title, sentences=3, auto_suggest=False)

        # Comprehensive set of patterns to extract 4-digit birth years
        patterns = [
            r"\(born\s+[A-Za-z]+\s+\d{1,2},\s+(\d{4})",           # (born January 1,
1940)
            r"\(born\s+(\d{4})",                                   # (born 1940)
            r"\bborn\s+[A-Za-z]+\s+\d{1,2},\s+(\d{4})",           # born January 1,
1940
            r"\bborn\s+in\s+(\d{4})",                              # born in 1940
            r"\bborn\s+on\s+[A-Za-z]+\s+\d{1,2},\s+(\d{4})",      # born on January 1,
1940
            r"\(b\.\s+(\d{4})",                                    # (b. 1940)
            r"\(b\.\s+[A-Za-z]+\s+\d{1,2},\s+(\d{4})",            # (b. January 1,
1940)
            r"\bb\.\s+(\d{4})",                                    # b. 1940
            r"\bb\.\s+[A-Za-z]+\s+\d{1,2},\s+(\d{4})",            # b. January 1, 1940
            r"\*\s*(\d{4})",                                     # * 1940 (some lists
use this for birth)
            r"\b[Dd]ate\s+of\s+[Bb]irth.*?(\d{4})",              # Date of Birth ...
1940
            r"\(.*?(\d{4})\s*–",                                 # (1940 – 2021)
            r"\(.*?(\d{4})\)",                                   # (1940)
            r"\b(\d{4})\s*–\s*\d{4}",                            # 1940–2021
            r"\bborn.*?(\d{4})",                                 # loose fallback:
born ... 1940
            r"\b(\d{4})\s*–",                                    # 1940– (common in
intro)
        ]

        for pattern in patterns:
            match = re.search(pattern, summary)
            if match:
                year = match.group(1)
                if year.isdigit() and 1800 < int(year) < 2025:
                    return int(year)

        return None

    except Exception as e:
        print(f"{name}: {e}")
        return None

# %%
print(box_d4)



# %% [markdown]
# Encoding Features

# %%
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer

# Create a list of divisions
divisions = ['minimum', 'super fly','bantam','super bantam', 'feather', 'super
feather',
             'light', 'super light', 'welter','super welter', 'middle', 'super
middle', 'light heavy', 'heavy']
ord_features = ['Division']
ordEnc = OrdinalEncoder(categories = [divisions])


# Create a ColumnTransformer to apply the OrdinalEncoder to the specified columns
```

```python
coltrans = ColumnTransformer(
    transformers=[
        ("ord", ordEnc, ord_features),
        ],
    remainder = 'passthrough',
    verbose_feature_names_out=False)


X_trans = coltrans.fit_transform(box_d4)
X_trans

# %%
new_feature_names = coltrans.get_feature_names_out()
new_feature_names

# %% [markdown]
# Create a New Dataframe with the encoded features

# %%
boxing_d2 = pd.DataFrame(X_trans, columns = new_feature_names)
boxing_d2

# %%
print(boxing_d2)

# %% [markdown]
# Check the minimum and maximum values for outliers

# %%
boxing_d2['Starting Age'].min(), boxing_d2['Starting Age'].max()

# %% [markdown]
# Model validation/ Selection

# %% [markdown]
# Standardization and Scaling

# %%
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = boxing_d2.drop(columns=["Rank","Win %", "Name",
"Origin","Wins","Losses","Draws","Total Bouts"])  # replace with your actual target
y = boxing_d2["Win %"]

feature_names = X.columns
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X

# %% [markdown]
# ElasticNet

# %%
from sklearn.linear_model import ElasticNetCV
from sklearn.metrics import mean_squared_error, r2_score

enet = ElasticNetCV(cv=5, random_state=42)
enet.fit(X_train_scaled, y_train)

y_pred_enet = enet.predict(X_test_scaled)

print("ElasticNet R²:", r2_score(y_test, y_pred_enet))
print("ElasticNet RMSE:", mean_squared_error(y_test, y_pred_enet, squared=False))

for feature, coef in zip(feature_names, enet.coef_):
    print(f"Feature: {feature}, Coefficient: {coef}")

# %% [markdown]
# ElasticNet Feature Importance

# %%
import plotly.express as px
import pandas as pd

# Create a DataFrame of coefficients
coef_df = pd.Series(enet.coef_,
index=X.columns).sort_values(ascending=False).reset_index()
coef_df.columns = ['Feature', 'Coefficient']

# Plot with Plotly
fig = px.bar(
    coef_df,
    x='Coefficient',
    y='Feature',
    orientation='h',
    title='ElasticNet Feature Importance',
    labels={'Coefficient': 'Model Coefficient', 'Feature': 'Feature'},
    text='Coefficient'
)
```

```python
fig.update_layout(yaxis=dict(autorange="reversed"))

fig.show()

# %% [markdown]
# GridSearch to find best parameters

# %%
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV

# Define the grid
param_grid = {
    'alpha': [0.01, 0.1, 1.0, 10.0],
    'l1_ratio': [0.1, 0.5, 0.7, 0.9, 1.0]
}

# Create model
elastic_net = ElasticNet(max_iter=10000, random_state=42)

# Grid Search
grid_search_enet = GridSearchCV(
    estimator=elastic_net,
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    verbose=1
)

# Fit model
grid_search_enet.fit(X_train_scaled, y_train)

# Best model
best_enet = grid_search_enet.best_estimator_

print("Best Params:", grid_search_enet.best_params_)
print("Train R²:", grid_search_enet.best_score_)

# Evaluate on test set
from sklearn.metrics import r2_score, mean_squared_error

y_pred_enet = best_enet.predict(X_test_scaled)
print("Test R²:", r2_score(y_test, y_pred_enet))
print("Test RMSE:", mean_squared_error(y_test, y_pred_enet, squared=False))

# %% [markdown]
# Model Summary
# %%
# Print a summary of the best ElasticNet model
print("ElasticNet Model Summary:")
print(f"Alpha: {best_enet.alpha}")
print(f"L1 Ratio: {best_enet.l1_ratio}")
print(f"Coefficients: {best_enet.coef_}")
print(f"Intercept: {best_enet.intercept_}")

# %% [markdown]
# Model Summary with Features

# %%
for feature, coef in zip(feature_names, best_enet.coef_):
    print(f"Feature: {feature}, Coefficient: {coef}")

# %%
X = X.apply(pd.to_numeric, errors='ignore')

# %%
for col in X.columns:
    if X[col].dtype == 'object':
        try:
            X[col] = X[col].astype(float)
        except:
            pass

# %% [markdown]
# XGBoost Model

# %%
from xgboost import XGBRegressor

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

xgb = XGBRegressor(n_estimators=100, max_depth=4, learning_rate=0.1, random_state=42)
xgb.fit(X_train, y_train)

y_pred_xgb = xgb.predict(X_test)

print("XGBoost R²:", r2_score(y_test, y_pred_xgb))
print("XGBoost RMSE:", mean_squared_error(y_test, y_pred_xgb, squared=False))

# %% [markdown]
# XGBoost Feature Importance

# %%
```

```python
importance_df = pd.Series(xgb.feature_importances_,
index=X.columns).sort_values(ascending=False).reset_index()
importance_df.columns = ['Feature', 'Importance']
# Plot feature importances
fig_importance = px.bar(
    importance_df,
    x='Importance',
    y='Feature',
    orientation='h',
    title="XGBoost Feature Importance",
    text='Importance'
)
fig_importance.update_layout(yaxis=dict(autorange="reversed"))
fig_importance.show()


# %% [markdown]
# XGBoost GridSearch

# %%
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor

# Define parameter grid
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Initialize base model
xgb = XGBRegressor(random_state=42)

# Grid search with 5-fold CV
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    cv=5,
    scoring='r2',  # You can change this to 'neg_mean_squared_error', etc.
    verbose=1,
    n_jobs=-1
)

# Fit search
grid_search.fit(X_train, y_train)
```

```
# Best model
best_model = grid_search.best_estimator_

print("Best Parameters:", grid_search.best_params_)
print("Best R² Score on Training:", grid_search.best_score_)

# Evaluate on test set
from sklearn.metrics import mean_squared_error, r2_score

y_pred = best_model.predict(X_test)
print("Test R²:", r2_score(y_test, y_pred))
print("Test RMSE:", mean_squared_error(y_test, y_pred, squared=False))

# %% [markdown]
# Assessment

# %%
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def evaluate_model(name, y_true, y_pred):
    print(f"🔍 {name} Evaluation")
    print(f"R² Score: {r2_score(y_true, y_pred):.3f}")
    print(f"RMSE: {mean_squared_error(y_true, y_pred, squared=False):.3f}")
    print(f"MAE: {mean_absolute_error(y_true, y_pred):.3f}")
    print('-' * 40)

print(evaluate_model("ElasticNet", y_test, y_pred_enet))
print(evaluate_model("XGBoost", y_test, y_pred_xgb))

# %% [markdown]
# Plotting Predictions vs actual for both models

# %%
import plotly.graph_objects as go
import numpy as np

def plot_predictions_plotly(y_true, y_pred, model_name):
    fig = go.Figure()

    # Scatter plot: predicted vs actual
    fig.add_trace(go.Scatter(
        x=y_true,
        y=y_pred,
        mode='markers',
        name='Predictions',
        marker=dict(size=8, opacity=0.7)
    ))
    # Diagonal reference line (perfect prediction)
    min_val = np.min([y_true.min(), y_pred.min()])
    max_val = np.max([y_true.max(), y_pred.max()])
    fig.add_trace(go.Scatter(
        x=[min_val, max_val],
        y=[min_val, max_val],
        mode='lines',
        line=dict(dash='dash', color='red'),
        name='Ideal Fit'
    ))

    fig.update_layout(
        title=f"{model_name} Predictions vs Actual",
        xaxis_title="Actual",
        yaxis_title="Predicted",
        width=600,
        height=600,
        template="plotly_white"
    )

    fig.show()

# Usage
plot_predictions_plotly(y_test, y_pred_enet, "ElasticNet")
plot_predictions_plotly(y_test, y_pred_xgb, "XGBoost")

# %% [markdown]
# Scatter of Longevity vs Starting Age

# %%
fig_tradeoff = px.scatter(
    boxing_d2,
    x='Starting Age',
    y='Longevity',
    trendline='ols',
    title='Longevity vs. Starting Age',
    labels={'Starting Age': 'Starting Age', 'Win %': 'Win Percentage'}
)
fig_tradeoff.show()
```