The Ultimate Predictor

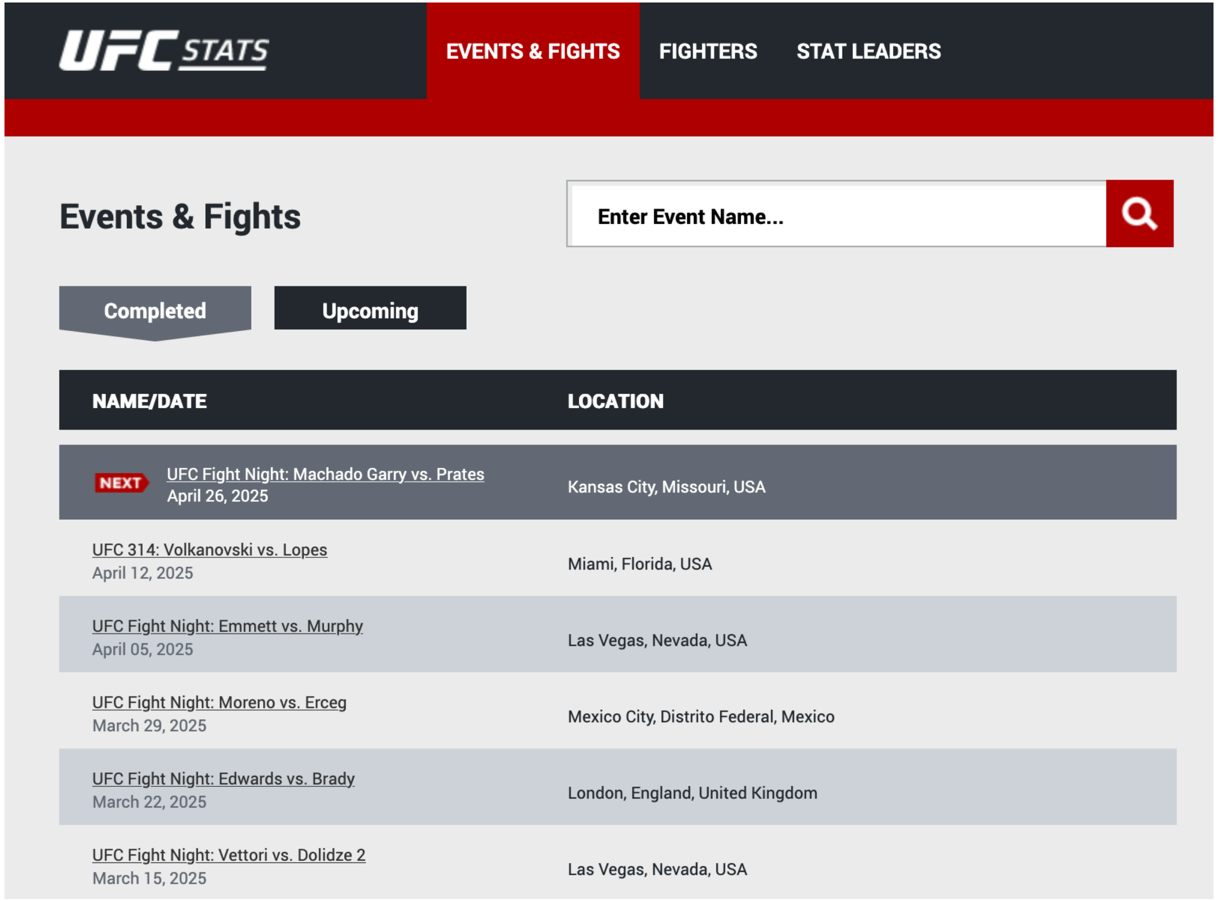
Cade Lueker CSCI5434

Analyzing UFC: Submission Insights and Predicting Fights

The Why

UFC Stats

UFC Stats

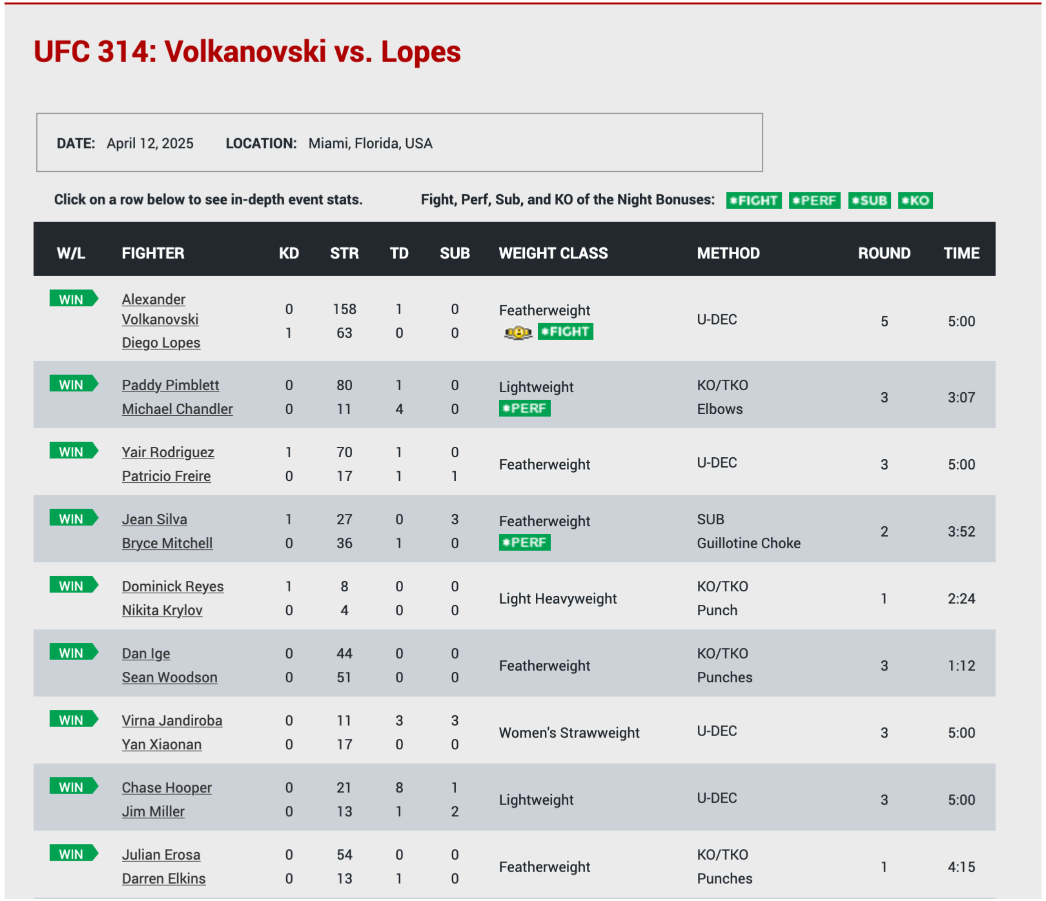


Events & Fights

Enter Event Name...

NAME/DATE	LOCATION
NEXT UFC Fight Night: Machado Garry vs. Prates April 26, 2025	Kansas City, Missouri, USA
UFC 314: Volkanovski vs. Lopes April 12, 2025	Miami, Florida, USA
UFC Fight Night: Emmett vs. Murphy April 05, 2025	Las Vegas, Nevada, USA
UFC Fight Night: Moreno vs. Erceg March 29, 2025	Mexico City, Distrito Federal, Mexico
UFC Fight Night: Edwards vs. Brady March 22, 2025	London, England, United Kingdom
UFC Fight Night: Vettori vs. Dolidze 2 March 15, 2025	Las Vegas, Nevada, USA

UFC Stats



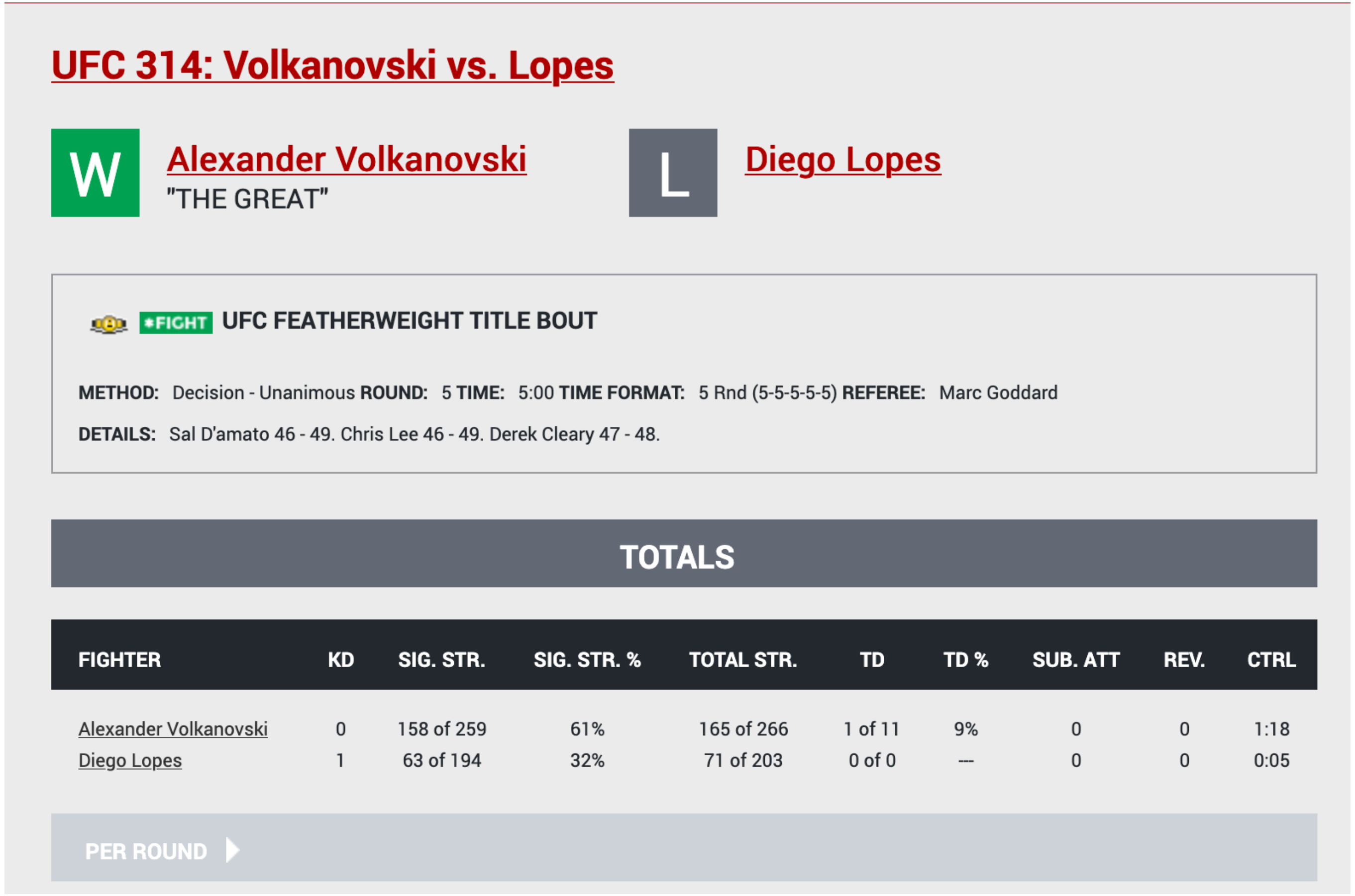
UFC 314: Volkanovski vs. Lopes

DATE: April 12, 2025 **LOCATION:** Miami, Florida, USA

Click on a row below to see in-depth event stats. **Fight, Perf, Sub, and KO of the Night Bonuses:** •FIGHT •PERF •SUB •KO

W/L	FIGHTER	KD	STR	TD	SUB	WEIGHT CLASS	METHOD	ROUND	TIME
WIN	Alexander Volkanovski	0	158	1	0	Featherweight •FIGHT	U-DEC	5	5:00
	Diego Lopes	1	63	0	0				
WIN	Paddy Pimblett	0	80	1	0	Lightweight •PERF	KO/TKO Elbows	3	3:07
	Michael Chandler	0	11	4	0				
WIN	Yair Rodriguez	1	70	1	0	Featherweight	U-DEC	3	5:00
	Patricio Freire	0	17	1	1				
WIN	Jean Silva	1	27	0	3	Featherweight •PERF	SUB Guillotine Choke	2	3:52
	Bryce Mitchell	0	36	1	0				
WIN	Dominick Reyes	1	8	0	0	Light Heavyweight	KO/TKO Punch	1	2:24
	Nikita Krylov	0	4	0	0				
WIN	Dan Ige	0	44	0	0	Featherweight	KO/TKO Punches	3	1:12
	Sean Woodson	0	51	0	0				
WIN	Virna Jandiroba	0	11	3	3	Women's Strawweight	U-DEC	3	5:00
	Yan Xiaonan	0	17	0	0				
WIN	Chase Hooper	0	21	8	1	Lightweight	U-DEC	3	5:00
	Jim Miller	0	13	1	2				
WIN	Julian Erosa	0	54	0	0	Featherweight	KO/TKO Punches	1	4:15
	Darren Elkins	0	13	1	0				

UFC Stats



UFC 314: Volkanovski vs. Lopes

W Alexander Volkanovski
"THE GREAT"

L Diego Lopes

FIGHT UFC FEATHERWEIGHT TITLE BOUT

METHOD: Decision - Unanimous **ROUND:** 5 **TIME:** 5:00 **TIME FORMAT:** 5 Rnd (5-5-5-5-5) **REFEREE:** Marc Goddard

DETAILS: Sal D'amato 46 - 49. Chris Lee 46 - 49. Derek Cleary 47 - 48.

TOTALS

FIGHTER	KD	SIG. STR.	SIG. STR. %	TOTAL STR.	TD	TD %	SUB. ATT	REV.	CTRL
Alexander Volkanovski	0	158 of 259	61%	165 of 266	1 of 11	9%	0	0	1:18
Diego Lopes	1	63 of 194	32%	71 of 203	0 of 0	---	0	0	0:05

PER ROUND

Web Scraping

```
class FightItem(scrapy.Item):
    """
    Information about a fight from a UFC event.
    """

    event_name = scrapy.Field()
    event_date = scrapy.Field()
    outcome = scrapy.Field()
    winner = scrapy.Field()
    loser = scrapy.Field()
    f1_name = scrapy.Field()
    f1_strikes = scrapy.Field()
    f1_td = scrapy.Field()
    f1_td_def = scrapy.Field()
    f2_name = scrapy.Field()
    f2_strikes = scrapy.Field()
    f2_td = scrapy.Field()
    f2_td_def = scrapy.Field()
    method = scrapy.Field()
    method_details = scrapy.Field()
    end_round = scrapy.Field()
    time = scrapy.Field()
    total_time = scrapy.Field()
    weight_class = scrapy.Field()
```

Web Scraping

```
class FighterItem(scrapy.Item):
    """
    Information about a fighter who has at least one fight in a UFC event.
    """

    name = scrapy.Field()
    height = scrapy.Field()
    reach = scrapy.Field()
    stance = scrapy.Field()
    dob = scrapy.Field()
```

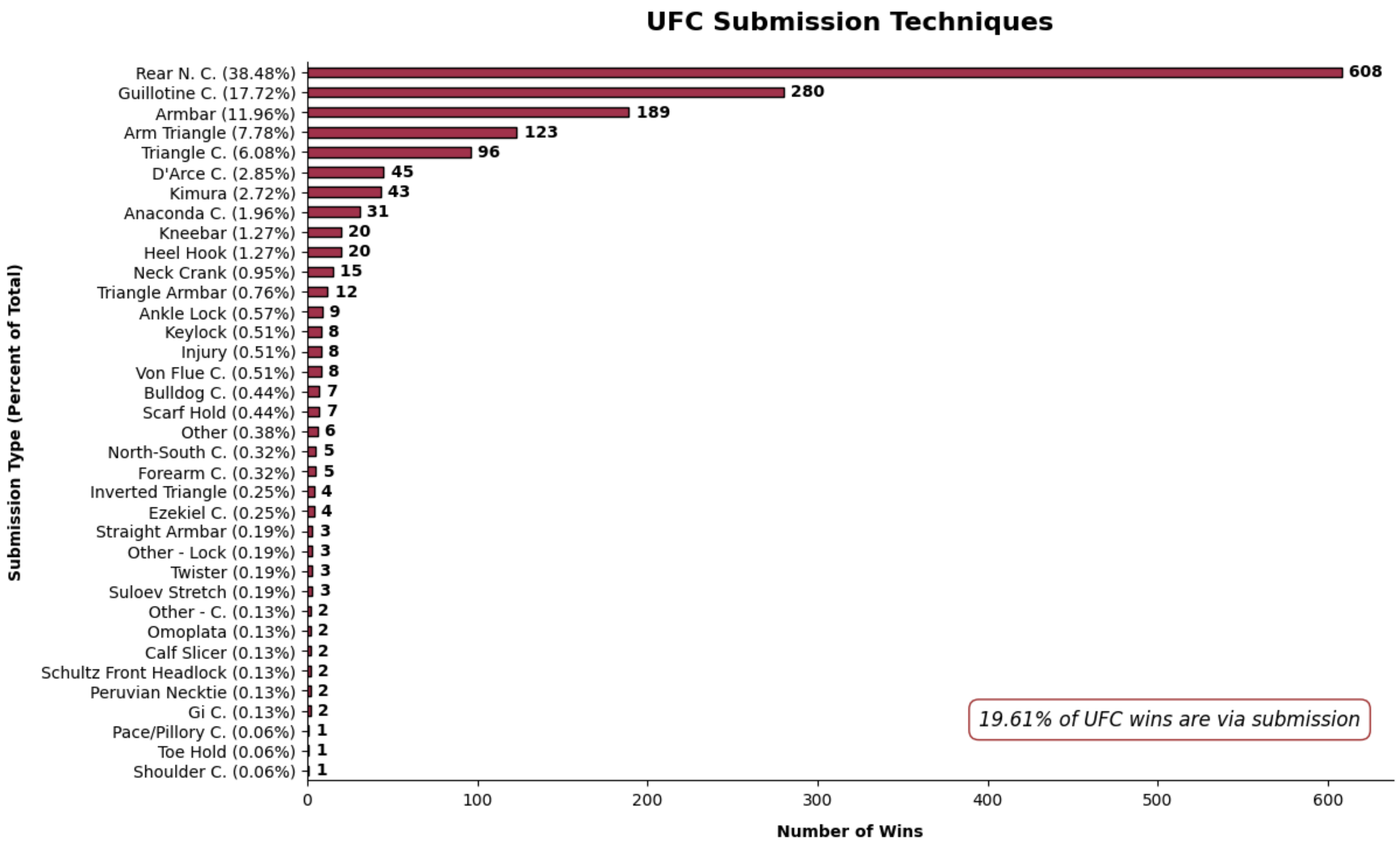
Web Scraping

```python
class TufSpider(scrapy.Spider):
    name = "tuf_spider"
    allowed_domains = ["ufcstats.com"]
    start_urls = [
        "http://ufcstats.com/statistics/events/completed?page=all",
    ]
    scraped_fighters = set()
    # --- helper methods ---
    def height_to_inches(self, height_str):  ...
    # parsing all events
    def parse(self, response):  ...
    # parse individual event
    def parse_event(self, response):  ...
    # if available parse fight details
    def parse_fight_details(self, response):  ...
    # parse individual fighter information
    def parse_fighter(self, response):  ...
```

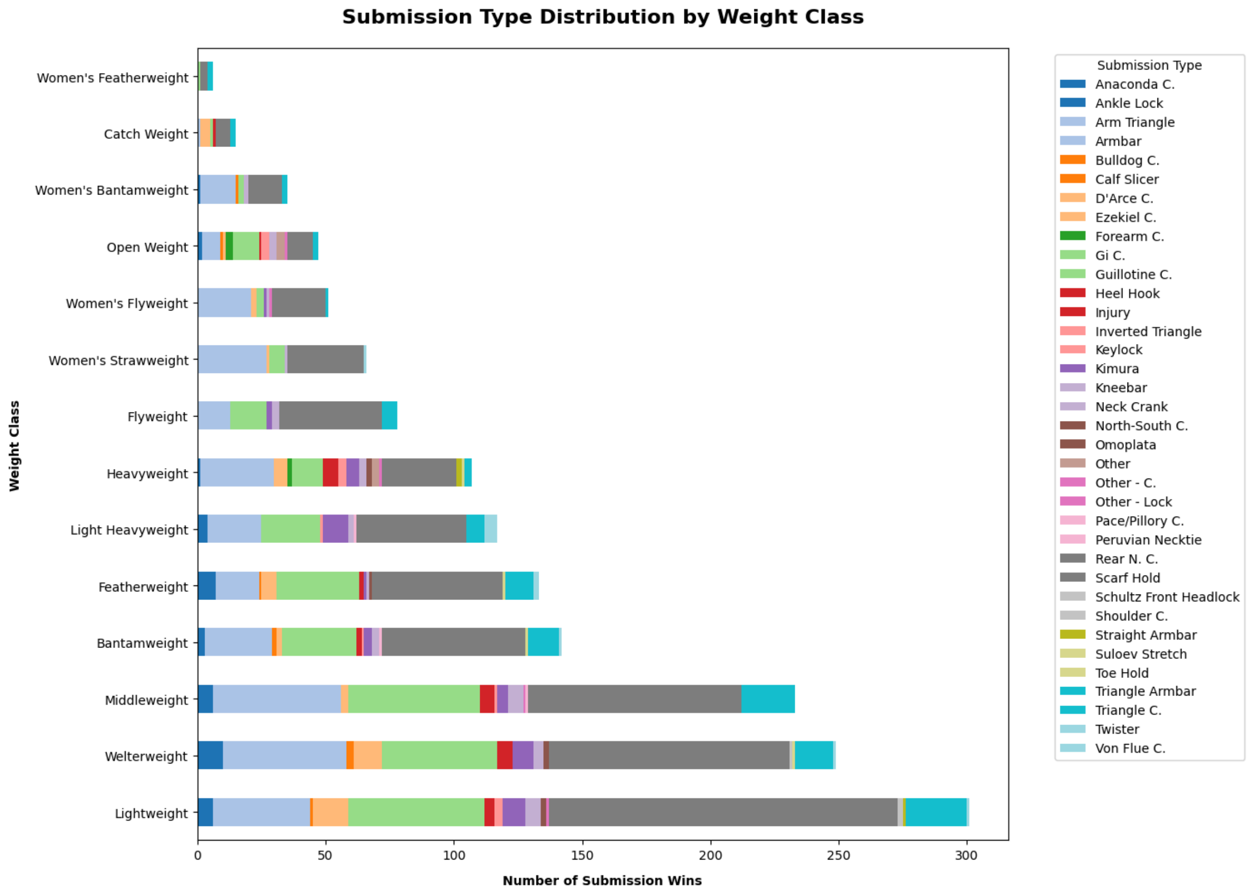
Cleaning and Combining the Data

```python
for fighter_num in ['f1', 'f2']:
    fights = fights.merge(
        fighters[['name', 'height', 'reach', 'stance', 'dob']],
        left_on=f'{fighter_num}_name',
        right_on='name',
        how='left',
        suffixes=('', f'_{fighter_num}')
    )
    fights = fights.rename(columns={
        'height': f'{fighter_num}_height',
        'reach': f'{fighter_num}_reach',
        'stance': f'{fighter_num}_stance',
        'dob': f'{fighter_num}_dob'
    })
    fights[f'{fighter_num}_age'] = fights.apply(
        lambda x: calculate_age(x[f'{fighter_num}_dob'], x['event_date']),
        axis=1
    )
    fights.drop(columns=['name'], inplace=True) # already have their names

fights['reach_diff'] = (fights['f1_reach'] - fights['f2_reach']).abs()
fights['height_diff'] = (fights['f1_height'] - fights['f2_height']).abs()
fights['age_diff'] = (fights['f1_age'] - fights['f2_age']).abs()
```

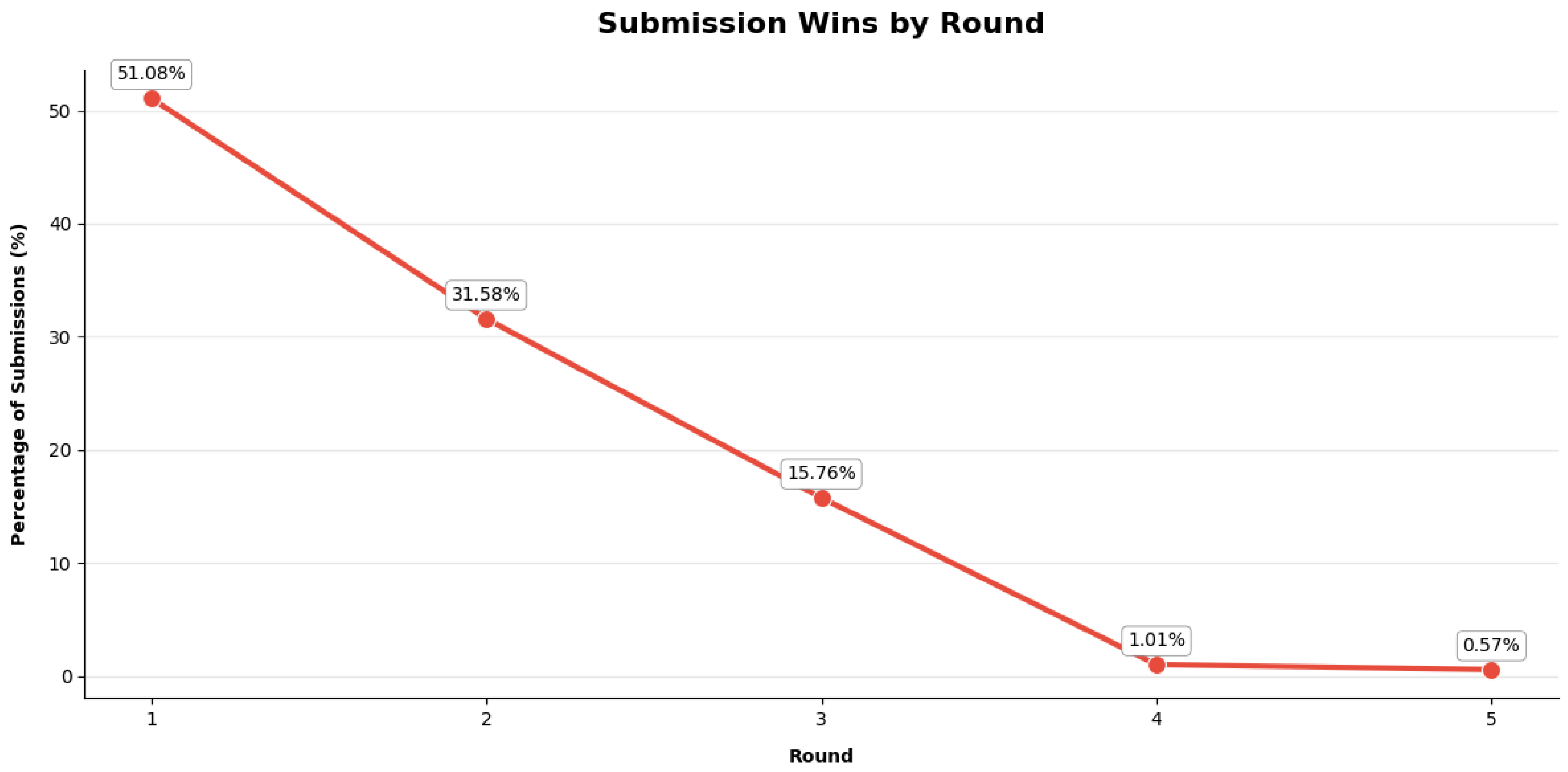
Submission rates



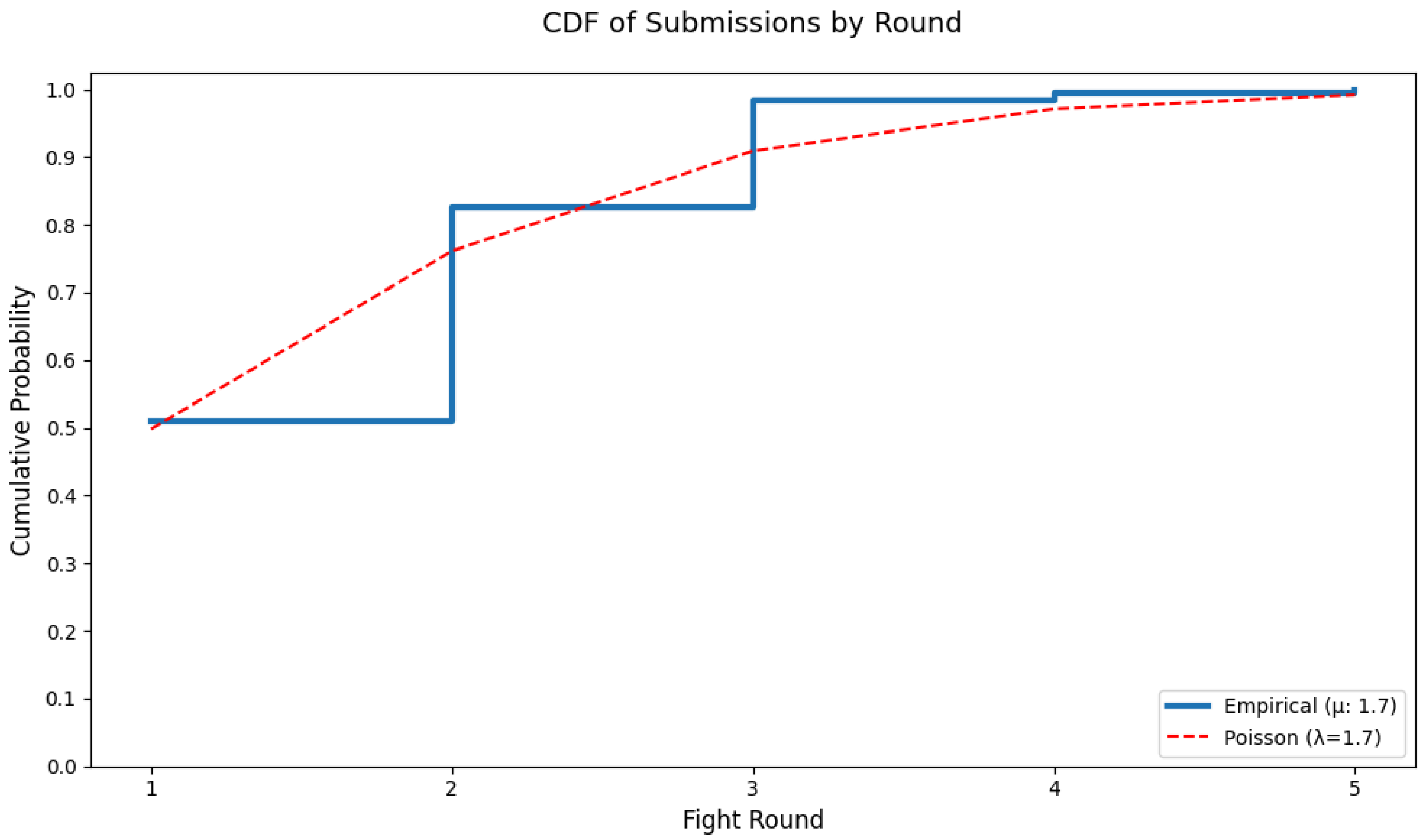
Submission Rates by Weightclass



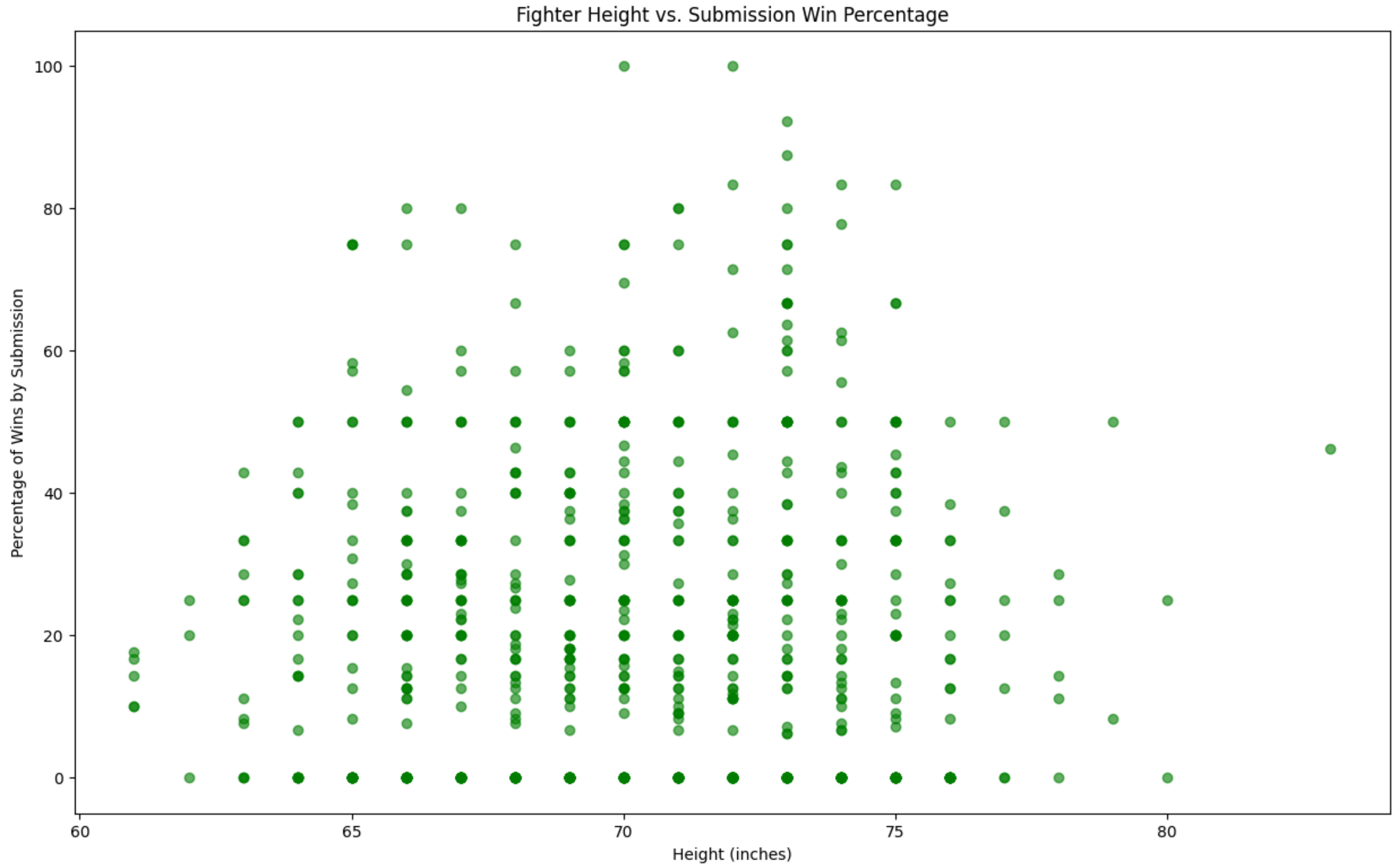
Submissions by round



Submissions by round CDF



Submissions to Height



Normalizing Heights and Linear Regression

- a short heavyweight might be a tall lightweight so normalizing by weightclass gives better insight.

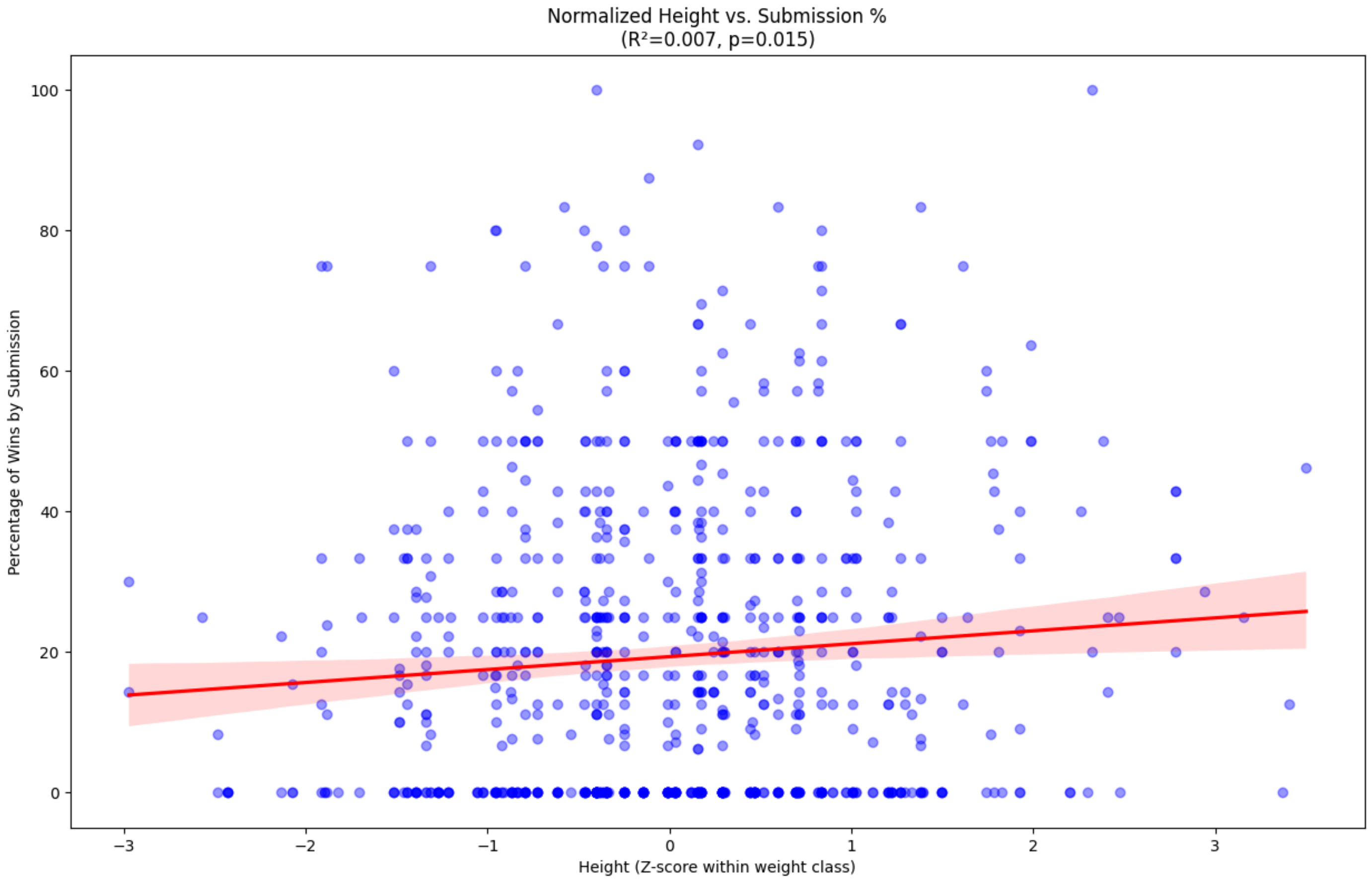
```
# add heights normalized by weightclass
fights['f1_norm_height'] = fights.groupby('weight_class')['f1_height'].transform(
    lambda x: (x - x.mean()) / x.std()
)
fights['f2_norm_height'] = fights.groupby('weight_class')['f2_height'].transform(
    lambda x: (x - x.mean()) / x.std()
)
```

Normalizing Heights and Linear Regression

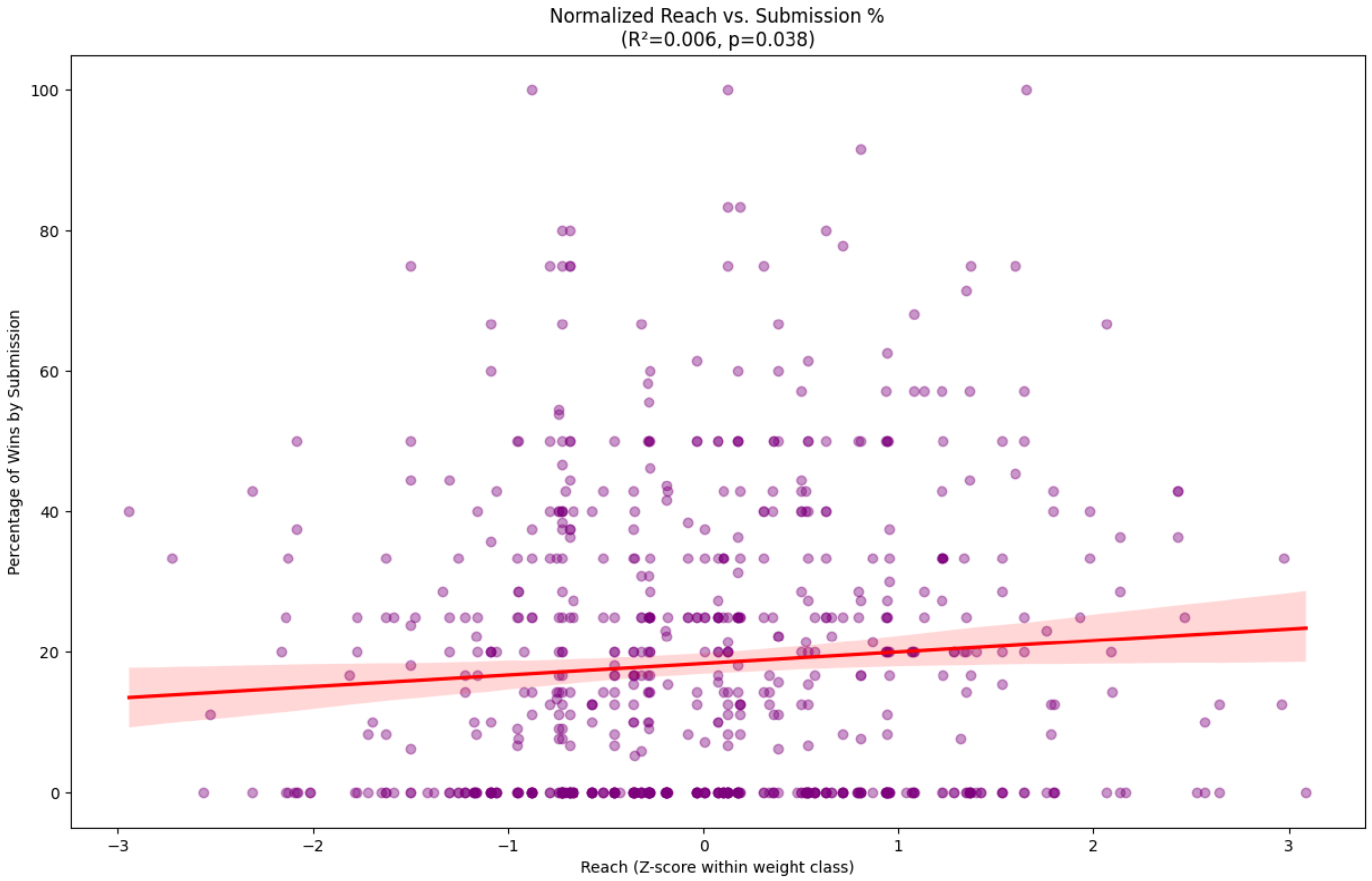
- a short heavyweight might be a tall lightweight so normalizing by weightclass gives better insight.

```
# regression values
slope, _, r_value, p_value, _ = stats.linregress(
    sub_df['norm_height'], sub_df['sub_percent']
)
```

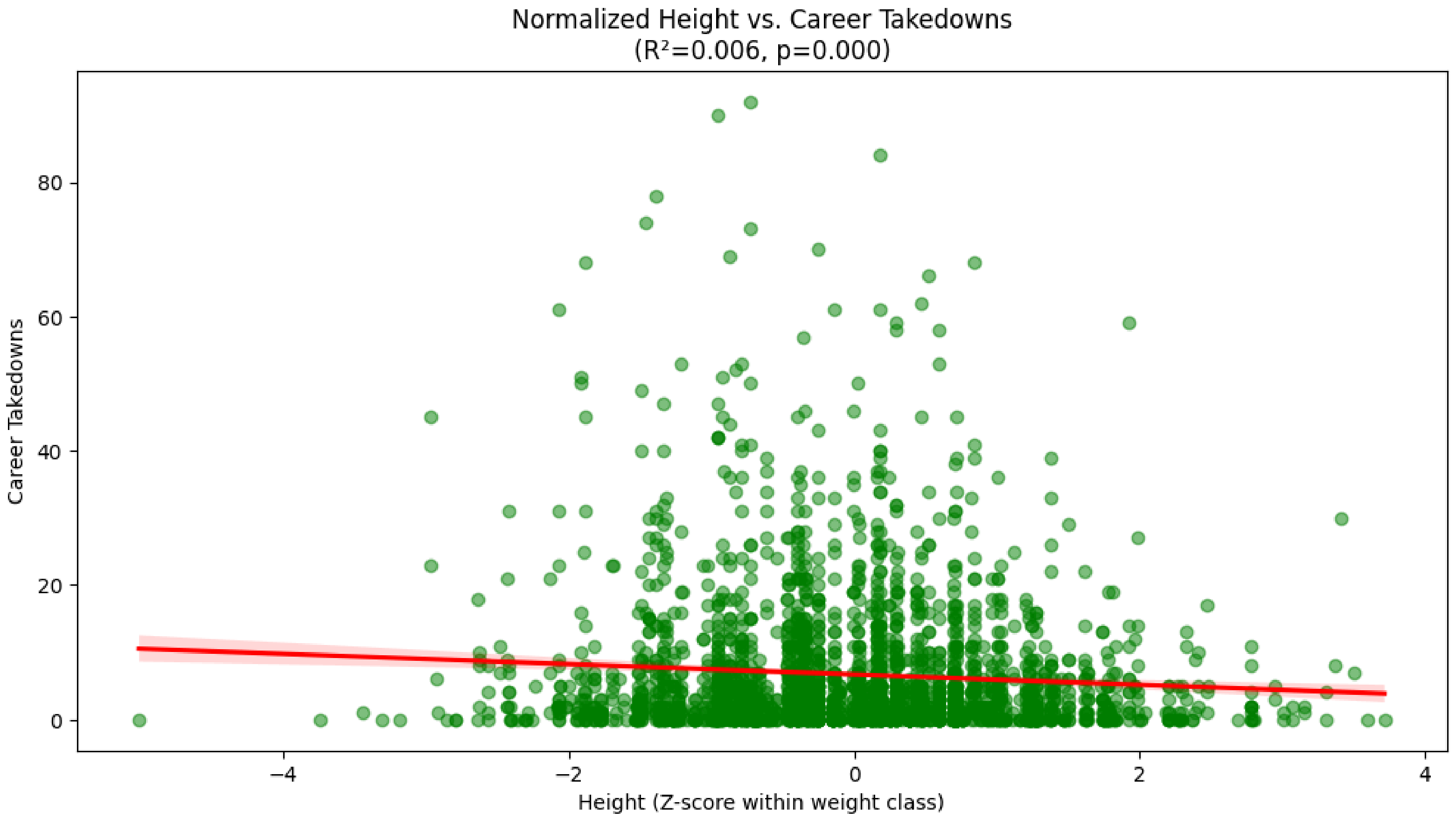
Submissions to Normalized Height



Submissions to Normalized Reach



Takedowns to Normalized Height

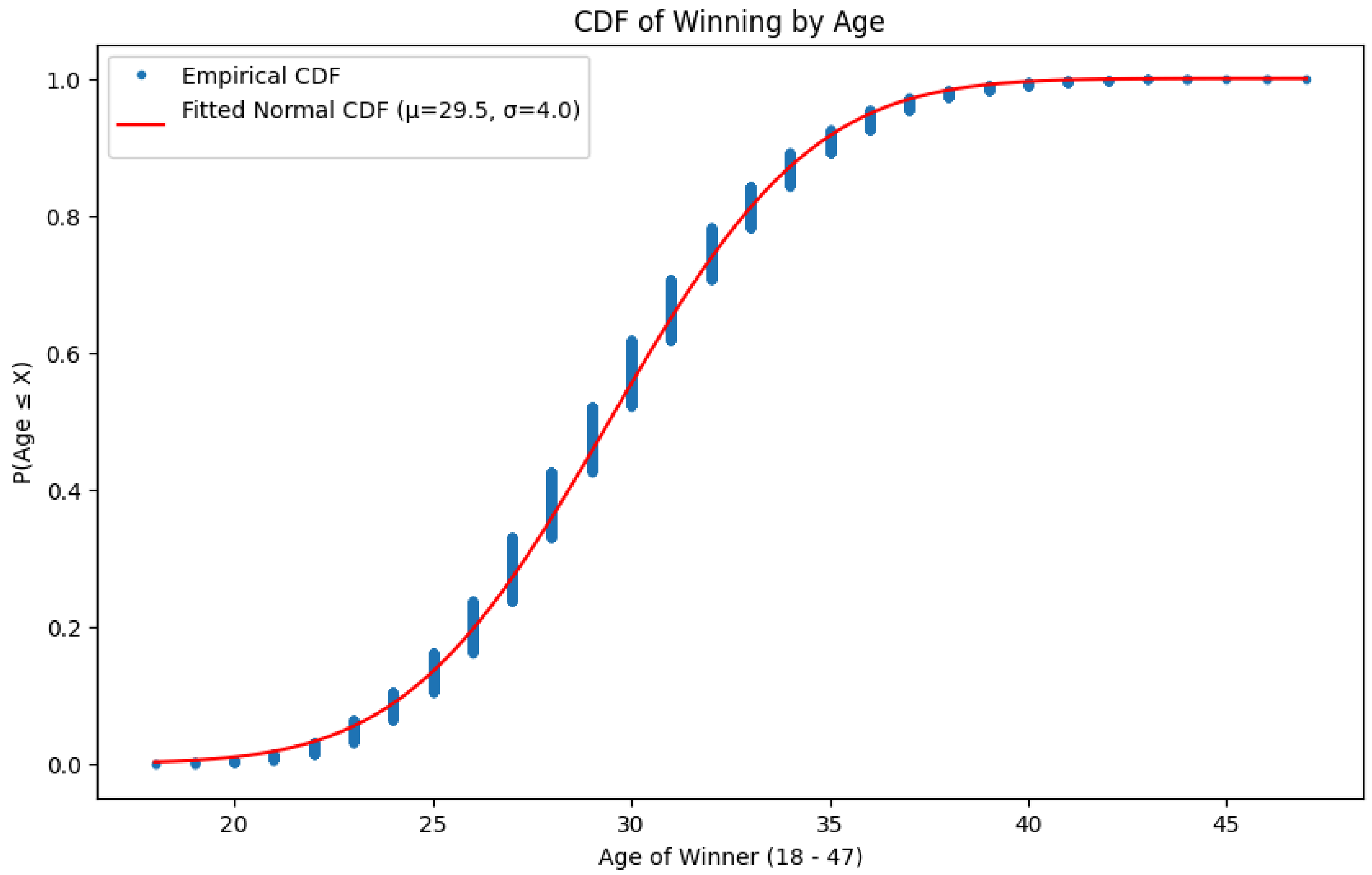


Age of Winning Fighter CDF

Code to calculate CDF

```python
# convert into df for better manipulation
win_ages = pd.DataFrame(winners_age, columns=['age'])

# sort ages and remove nonexistant data (fighters pre 2000 sometimes don't have their ages recorded)
sorted_ages = np.sort(win_ages['age'].dropna())

# normalize the data
mu, sigma = stats.norm.fit(win_ages['age'].dropna())

# get points for cdf
x = np.linspace(min(sorted_ages), max(sorted_ages), 100)
y = np.arange(1, len(sorted_ages) + 1) / len(sorted_ages)

# get CDF
cdf_fitted = stats.norm.cdf(x, mu, sigma)
```

Age of Winning CDF Plotted



Prediction

- creating a new dataframe with historical data for each fighter

```
def build_fighter_history(fights):
    rows = []

    for _, row in fights.iterrows():
        event_date = row['event_date']
        previous_fights = fights[fights['event_date'] < event_date]

        # fighter 1 stats from previous fights
        past_f1 = previous_fights[(previous_fights['f1_name'] == row['f1_name']) | (previous_fights['f2_name'] == row['f
        # fighter 2 stats from previous fights
        past_f2 = previous_fights[(previous_fights['f1_name'] == row['f2_name']) | (previous_fights['f2_name'] == row['f

        # fighter 1
        # strikes
        avg_f1_strikes = past_f1['f1_strikes'].mean()
        # takedowns
        avg_f1_td = past_f1['f1_td'].mean()
        avg_f1_td_def = past_f1['f1_td_def'].mean()
        avg_f1_td_rate = past_f1['f1_td_rate'].mean()
        avg_f1_td_def_rate = past_f1['f1_td_def_rate'].mean()
        ...
```

All possible features

```
features = [
    # strikes
    # 'strikes',
    # takedown
    # 'td', # 'td_def', # 'td_rate', # 'td_def_rate',
    # metrics
    # 'height', # 'reach', # 'age', # 'stance',
    # record
    'last',
    'last_3',
    'record',
    'ko_loss',
    'ko_rate',
    # 'already_beat', # 'opp_strikes', # 'opp_td', # 'opp_td_def',
    # 'opp_td_rate', # 'opp_td_def_rate', # 'opp_height', # 'opp_reach',
    # 'opp_age', # 'opp_stance', # opponent record
    'opp_last',
    'opp_last_3',
    'opp_record',
    'opp_ko_loss',
    'opp_ko_rate',
    # 'opp_already_beat',
    # differentials
```

Splitting the data

- because each fight contains historical data we need to split by date otherwise we could see leakage in our model

```
# start with oldest now
df = df.sort_values('event_date', ascending=True)
print(df.head(1))

# 80% train, 20% test
split_idx = int(len(df) * 0.8)
train_df = df.iloc[:split_idx]
test_df = df.iloc[split_idx:]
```

xgboost

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

xgb_model = xgb.XGBClassifier(
    objective='binary:logistic',
    eval_metric='auc',
    max_depth=3,
    learning_rate=0.01,
    n_estimators=1000,
    subsample=0.7,
    colsample_bytree=0.7,
    reg_alpha=0.5,
    reg_lambda=0.5,
    early_stopping_rounds=20,
)

xgb_model.fit(
    X_train_scaled, y_train,
    eval_set=[(X_train_scaled, y_train), (X_test_scaled, y_test)],
)
```

Random Forest

```
forest_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=5,
    random_state=5
)

forest_model.fit(X_train, y_train)
preds = forest_model.predict(X_test)
pred_probs = forest_model.predict_proba(X_test)[:, 1]
```

Results

- **xgboost**
  - Accuracy: *0.6139*
  - ROC AUC: *0.6475*
- **Random Forest**
  - Accuracy: *0.6094*
  - ROC AUC: *0.6463*