Date: May 23, 2023
Author: Dan Segal
Github: @djsegal (https://github.com/djsegal/zelus_assessment)

Zelus Assessment

This assessment involves analysis of ball-by-ball data from professional cricket matches. The data include the overall match results and outcomes for every delivery for both teams.

The problems below showcase your ability to gather inferences from real-world data using methods in statistics and machine learning, as well as your ability to productionalize and deploy models for consumption. After the first question, there are no right or wrong answers; in fact, we have left questions quite open-ended as an opportunity for you to determine what assumptions are appropriate and operate within those. In addition to a sensible problem- solving strategy, we'll be looking for correct implementation and validation of the appropriate techniques you've chosen, as well as an understanding of their limitations.

Since the purpose of this assessment is to showcase your technical and problem-solving skills, please include clear, efficient, and well-organized code along with explanations on the justification for your problem-solving approach, its limitations, and the conclusions you're able to draw at each step. Please also include instructions on how to run your code, and structure it in a way that makes it as reproducible as possible.

Part 1: Data Analysis

To get started, download the One Day International match results and ball-by-ball innings data. These data were sourced from cricsheet.org and includes ball-by-ball summaries of ODIs from 2006-present for men and 2009-present, for women.

Question 0

We don't expect you to have any cricket knowledge and that isn't a requirement to ace this assessment.But we understand that familiarity with cricket may vary from one candidate to the next so we would like to know how you would rate your knowledge of cricket from 1 to 5, where 1 is basically no knowledge (like you had never seen or read anything about the sport until the days before this assessment) and 5 is highly knowledgeable (you watch matches regularly and have a jersey for the Rajasthan Royals in your closet, for example).

Answer 0

Prior to this assessment, my familiarity with cricket was limited, and I would have rated myself a 1. However, in preparation for this task, I have taken the time to watch the recommended Netflix Explained: Cricket video and read about the basic rules of the sport. While my practical knowledge is still developing, I am confident in my ability to apply my data analysis skills to this dataset, and look forward to the insights that this unique context will bring.

Question 1

Determine the win records (percentage win and total wins) for each team by year and gender, excluding ties, matches with no result, and matches decided by the DLS method in the event that, for whatever reason, the planned innings can’t be completed.

- Which male and female teams had the highest win percentages in 2019? Which had the highest total wins?
- Were these teams the same as those with the highest win percentages? Comment on why the leaders of these two stats might differ.

Answer 1

As observed from the tables at the end of this answer, **Australia** led in both total wins and win percentage in the women's 2019 season. For the men's teams, **Australia** had the most wins, while the **Netherlands**, despite playing fewer matches, had the highest win percentage.

In [1]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
```

In [2]:

```
match_results_filename = 'data/match_results.json'
match_data = pd.read_json(match_results_filename).rename(columns={"teams": "team"}

# only look at games with a definitive winner
match_data = match_data.dropna(subset="outcome.winner").reset_index(drop=True)

# require at least 1% of each column to be non-nan
match_data = match_data.dropna(thresh=len(match_data)/100, axis=1).copy()

# require more than one possible value (nan included)
for col in match_data.columns:
    if len(match_data[col].unique()) > 1: continue
    match_data.drop(col, inplace=True, axis=1)

match_data["year"] = pd.to_datetime(match_data.dates).dt.year
match_data["did_win"] = ( match_data["outcome.winner"] == match_data["team"] )

# each game has two rows. `team` column differs
assert match_data.matchid.value_counts().unique() == [2]
assert match_data.did_win.value_counts().nunique() == 1

print("\nAlternate Game Endings:", dict(match_data["outcome.method"].value_counts(
match_data = match_data[pd.isnull(match_data["outcome.method"])]

# assert games either end from runs or wickets
assert np.all(
    pd.isnull(match_data["outcome.runs"]) == ~pd.isnull(match_data["outcome.wicket
)

with pd.option_context('display.max_rows', 4,'display.max_columns', 6):
    display(match_data)
```

```
Alternate Game Endings: {'D/L': 370}
```

	city	dates	gender	...	match_type_number	year	did_win
0	Brisbane	2017-01-13	male	...	NaN	2017	True
1	Brisbane	2017-01-13	male	...	NaN	2017	False
...	...	...	...	...	...	...	...
4106	Edinburgh	2016-08-16	male	...	NaN	2016	True
4107	Edinburgh	2016-08-16	male	...	NaN	2016	False

3738 rows × 18 columns

In [3]:

```
# use bats_first boolean to replace toss logic
match_data["bat_first"] = (
    ( match_data.team == match_data["toss.winner"] ) ^
    ( match_data["toss.decision"] == "field" )
)

# drop unneeded columns
match_data = match_data.drop(columns=[
    "toss.decision", "toss.winner"
])

print("Bat First Win Rate: ", round(
    100*match_data[match_data["did_win"]]["bat_first"].value_counts()[True] / (len
), "%\n")

print("Variable Summary")

display(match_data.gender.value_counts())
display(match_data.city.value_counts().head(7))
```

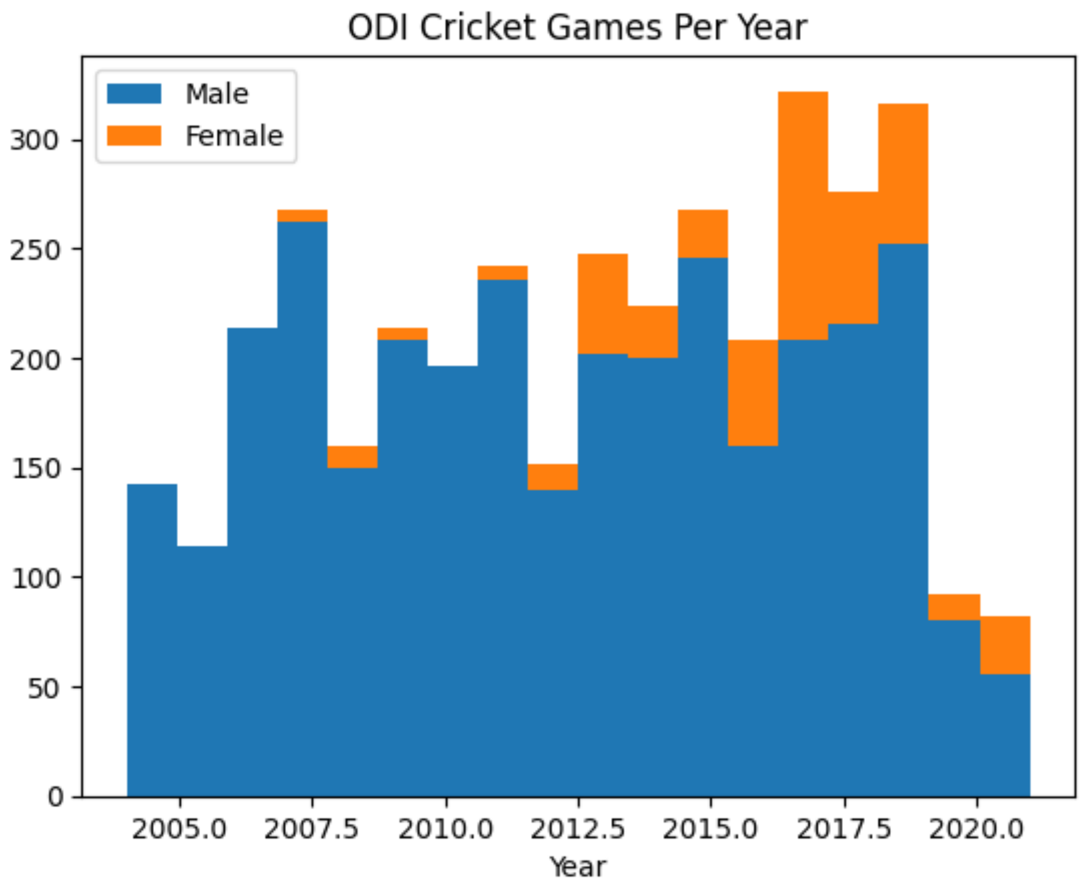
```
Bat First Win Rate:  48 %

Variable Summary

gender
male      3282
female     456
Name: count, dtype: int64

city
Mirpur       164
Colombo      156
London       126
Abu Dhabi     90
Bulawayo      78
Brisbane      66
Dublin        58
Name: count, dtype: int64
```

In [4]:

```
# todo: ask about difference in data set
#       2006 (male), 2009 (female)

print("\nFirst Game Dates (by gender)")
display(match_data.sort_values(by="year").drop_duplicates("gender")[["gender","dat
print()

plt.hist([
    match_data[match_data.gender == "male"].year,
    match_data[match_data.gender == "female"].year
], bins=match_data.year.nunique(), stacked=True, label=["Male", "Female"])

plt.title("ODI Cricket Games Per Year")
plt.xlabel("Year")
plt.legend();
```

```
First Game Dates (by gender)
```

	gender	dates
3481	male	2004-07-06
1765	female	2007-03-05



In [5]:

```
win_data = match_data.groupby(["gender","year","team"])['did_win'].agg(
    total_wins='sum', win_percentage='mean'
).reset_index()

win_data = win_data.sort_values(
    by=["total_wins","win_percentage"], ascending=False
).reset_index(drop=True)

match_data = pd.merge(
    match_data, win_data, on=["gender","year","team"]
)

# save match_results now that all features are added
match_data.to_csv("data/match_results.csv", index=False)

win_data = win_data[win_data.year == 2019]

win_data.head(6)
```

Out[5]:

	gender	year	team	total_wins	win_percentage
14	male	2019	Australia	16	0.695652
22	male	2019	India	15	0.652174
27	male	2019	England	14	0.736842
38	male	2019	New Zealand	13	0.684211
62	female	2019	Australia	10	1.000000
73	male	2019	West Indies	10	0.454545

```
In [6]:
```

```python
def print_header(input_header, cur_delim="-"):
    print()
    print(cur_delim * (len(input_header)+2))
    print(" " + input_header)
    print(cur_delim * (len(input_header)+2))

print_header("2019 Cricket Win Stats", "=")

print_header("Highest Win Percentages")
display(
    win_data[win_data.year == 2019].sort_values(by="win_percentage", ascending=Fal
)

print_header("Highest Total Wins")
display(
    win_data[win_data.year == 2019].sort_values(by="total_wins", ascending=False).
)
```

```
========================
 2019 Cricket Win Stats
========================

-------------------------
 Highest Win Percentages
-------------------------
```

	gender	year	team	total_wins	win_percentage
62	female	2019	Australia	10	1.0
261	male	2019	Netherlands	1	1.0

```
--------------------
 Highest Total Wins
--------------------
```

	gender	year	team	total_wins	win_percentage
62	female	2019	Australia	10	1.000000
14	male	2019	Australia	16	0.695652

As observed from the tables above, **Australia** led in both total wins and win percentage in the women's 2019 season. For the men's teams, **Australia** had the most wins, while the **Netherlands**, despite playing fewer matches, had the highest win percentage.

This illustrates how total wins and win percentages provide different perspectives on team performance. A team might play fewer matches but win a larger proportion, while another might play more matches, win many, but not most of them. Therefore, for a comprehensive performance evaluation, it's important to consider both these metrics.

Date: May 23, 2023
Author: Dan Segal
Github: @djsegal (https://github.com/djsegal/zelus_assessment)

Zelus Assessment

Part 1: Data Analysis (cont.)

Question 2

Setting aside individual batter production, cricket teams have two main 'resources' for producing runs:

- remaining overs and
- remaining wickets

The role resources have on run production is central to the statistical method known as 'DLS', which is used to award a winner in the case of incomplete/disrupted matches.

Part A

Use the ball-by-ball summaries under the innings descriptions of each men's match to make a dataset with the run and wicket outcomes for each delivery in a match, excluding matches with no result.

In [1]:

```
import pandas as pd
import numpy as np

from itertools import groupby, count

import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```python
innings_results_filename = 'data/innings_results.json'
innings_data = pd.read_json(innings_results_filename)

index_cols = ["matchid","innings","over"]
innings_data.sort_values(by=index_cols, inplace=True)

# remove rows with no matchid
innings_data = innings_data.dropna(subset="matchid")

# remove duplicated rows
innings_data = innings_data.drop_duplicates(subset=[*index_cols,"wicket.player_out

# remove games with very unusual events
bad_games = innings_data[
    ( innings_data["wicket.kind"] == "retired hurt"  ) |
    ( innings_data["wicket.kind"] == "obstructing the field"  )
].matchid

display(innings_data["wicket.kind"].value_counts())
innings_data = innings_data[~innings_data.matchid.isin(bad_games)]

# remove very rare fields
innings_data = innings_data.dropna(thresh=len(innings_data)/10000, axis=1).copy()

# require more than one possible value
for col in innings_data.columns:
    if len(innings_data[col].unique()) == 1:
        innings_data.drop(col, inplace=True, axis=1)

innings_data = innings_data.reset_index(drop=True)
```

```
wicket.kind
caught                   17339
bowled                    5621
lbw                       3436
run out                   2484
caught and bowled          955
stumped                    881
retired hurt                48
hit wicket                  27
obstructing the field        6
Name: count, dtype: int64
```

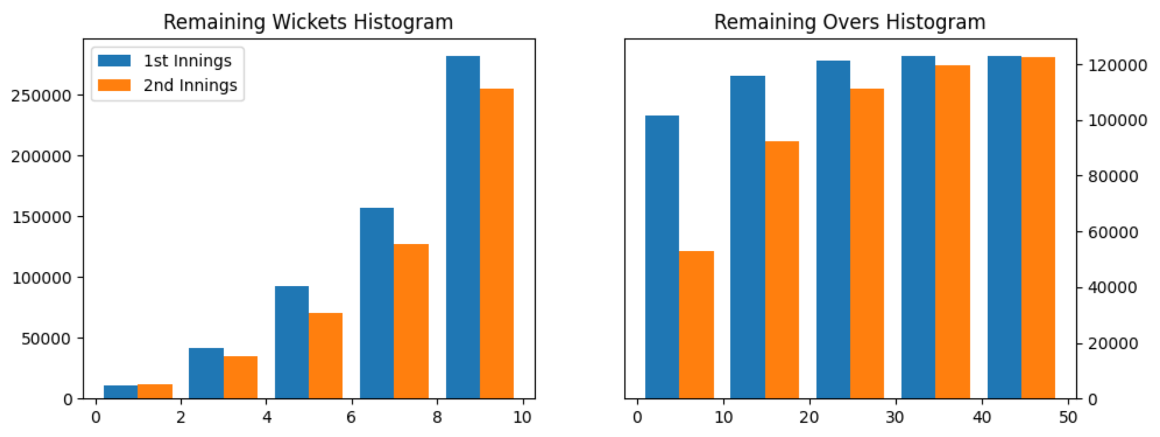
In [3]:

```
# remove wides and noballs to correct pitch count
innings_data = innings_data[pd.isnull(innings_data["wides"])]
innings_data = innings_data[pd.isnull(innings_data["noballs"])]

# add `overs` field that removes pitch component
innings_data["overs"] = np.floor(innings_data["over"]).astype(int)

innings_data["pitch"] = innings_data.groupby(["matchid","innings","overs"]).cumcou
assert np.max(innings_data["pitch"]) == 7

drop_indices = []

for cur_row in innings_data[innings_data["pitch"] == 7].itertuples():
    sub_table = innings_data[
        ( innings_data.innings == cur_row.innings ) &
        ( innings_data.matchid == cur_row.matchid ) &
        ( innings_data.overs == cur_row.overs )
    ]

    duplicated_indices = sub_table.drop(columns=["over","pitch"]).duplicated()[
        sub_table.drop(columns=["over","pitch"]).duplicated()
    ].index

    if len(duplicated_indices) > 0:
        c = count()
        val = max((list(g) for _, g in groupby(duplicated_indices, lambda x: x-nex

        drop_indices.append(val[len(val)//2])
        continue

    runless_indices = sub_table["runs.total"][1:-1][sub_table["runs.total"][1:-1]
    if len(runless_indices) > 0:
        assert len(runless_indices) < 3
        drop_indices.append(runless_indices[0])
        continue

    display(sub_table)

print("Dropped Pitch Count: ", len(drop_indices))
print("Total Pitch Count: ", "{:.1e}".format(len(innings_data)))

innings_data = innings_data[~innings_data.index.isin(drop_indices)].copy()

innings_data["pitch"] = innings_data.groupby(["matchid","innings","overs"]).cumcou
assert np.max(innings_data["pitch"]) == 6

print("\nPitch Mismatch [%]: ", int(np.round(100*(1-np.mean(
    innings_data["over"]  == ( innings_data["overs"].astype(str) + "." + innings_d
)))))

# todo: check this value against data set of final game runs
innings_data["runs"] = innings_data.groupby(["team","matchid"])["runs.total"].cums
```

```
Dropped Pitch Count:  41
Total Pitch Count:  1.1e+06

Pitch Mismatch [%]:  9
```

In [4]:

```
innings_data["is_wicket"] = ~pd.isnull(innings_data["wicket.kind"])
innings_data["remaining_wickets"] = 10 - innings_data.groupby(["team","matchid"]).

assert np.min(innings_data["remaining_wickets"]) == 0
assert np.max(innings_data["remaining_wickets"]) == 10

innings_data["remaining_overs"] = ((49 - innings_data["overs"]) + (6 - innings_dat

assert np.abs( np.min(innings_data["remaining_overs"]) - 0.0 ) < 1e-6
assert np.max(innings_data["remaining_overs"]) == 49.5

# make histograms for remaining resources
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(11,4))

ax1.hist([
    innings_data[innings_data.innings==1].remaining_wickets,
    innings_data[innings_data.innings==2].remaining_wickets
],bins=5, stacked=False, label=["1st Innings","2nd Innings"]);

ax2.hist([
    innings_data[innings_data.innings==1].remaining_overs,
    innings_data[innings_data.innings==2].remaining_overs
],bins=5, stacked=False);

ax1.set_title("Remaining Wickets Histogram")
ax2.set_title("Remaining Overs Histogram")

ax2.yaxis.tick_right()
ax1.legend();
```



In [5]:

```
# merge with results from previous step
match_data = pd.read_csv("data/match_results.csv")
innings_data = pd.merge(innings_data, match_data, on=["matchid", "team"])

# restrict to male league games (as noted in document)
innings_data = innings_data[innings_data.gender == "male"]

# get runs_per_over using end of game results
end_of_game_data = innings_data.drop_duplicates(subset=["team","matchid"], keep="l
end_of_game_data["runs_per_over"] = end_of_game_data["runs"] / end_of_game_data["o

innings_data = pd.merge(
    innings_data, end_of_game_data[["matchid","team", "runs_per_over"]], on=["matc
)

# use weights to focus on rarer "remaining resource" pairings
weight_df = pd.DataFrame(
    1/np.log1p(innings_data[["innings","remaining_overs","remaining_wickets"]].val
).reset_index().rename(columns={"count":"weight"})

innings_data = pd.merge(
    innings_data, weight_df,
    on=["innings","remaining_overs","remaining_wickets"]
).sort_values(by=index_cols)

with pd.option_context('display.max_columns', 8,'display.max_rows', 6):
    display(innings_data[
        ( innings_data["overs"] == 49 ) &
        ( innings_data["pitch"] == 6 ) &
        ( innings_data["remaining_wickets"] == 0 ) &
        ( innings_data["innings"] == 2 )
    ])
```

	batsman	bowler	non_striker	runs.batsman	...	total_wins	win_percentage	runs_pe
858636	NW Bracken	HMCM Bandara	GB Hogg	0	...	17	0.708333	5.0
858637	RP Singh	MS Panesar	KD Karthik	0	...	15	0.500000	3.5
858638	WD Parnell	P Kumar	CK Langeveldt	1	...	11	0.733333	5.8
...	...	...	...	...	...	...	...	
858645	A Zampa	B Kumar	AT Carey	0	...	16	0.695652	6.1
858646	Sayed Shirzad	O Thomas	Mujeeb Ur Rahman	0	...	3	0.187500	5.5
858647	Kuldeep Yadav	MP Stoinis	JJ Bumrah	0	...	15	0.652174	4.6

12 rows × 46 columns

Part B

Develop a model to predict an average team's expected runs per over. Please state or include the assumptions/validation used to justify your model choice. A visualization prior to modelling could be helpful to justify your modelling decisions. Save your intermediate data with team, inning order, remaining overs, and remaining wickets to a JSON/CSV file for Q4. Summarize your conclusions.

In [6]:

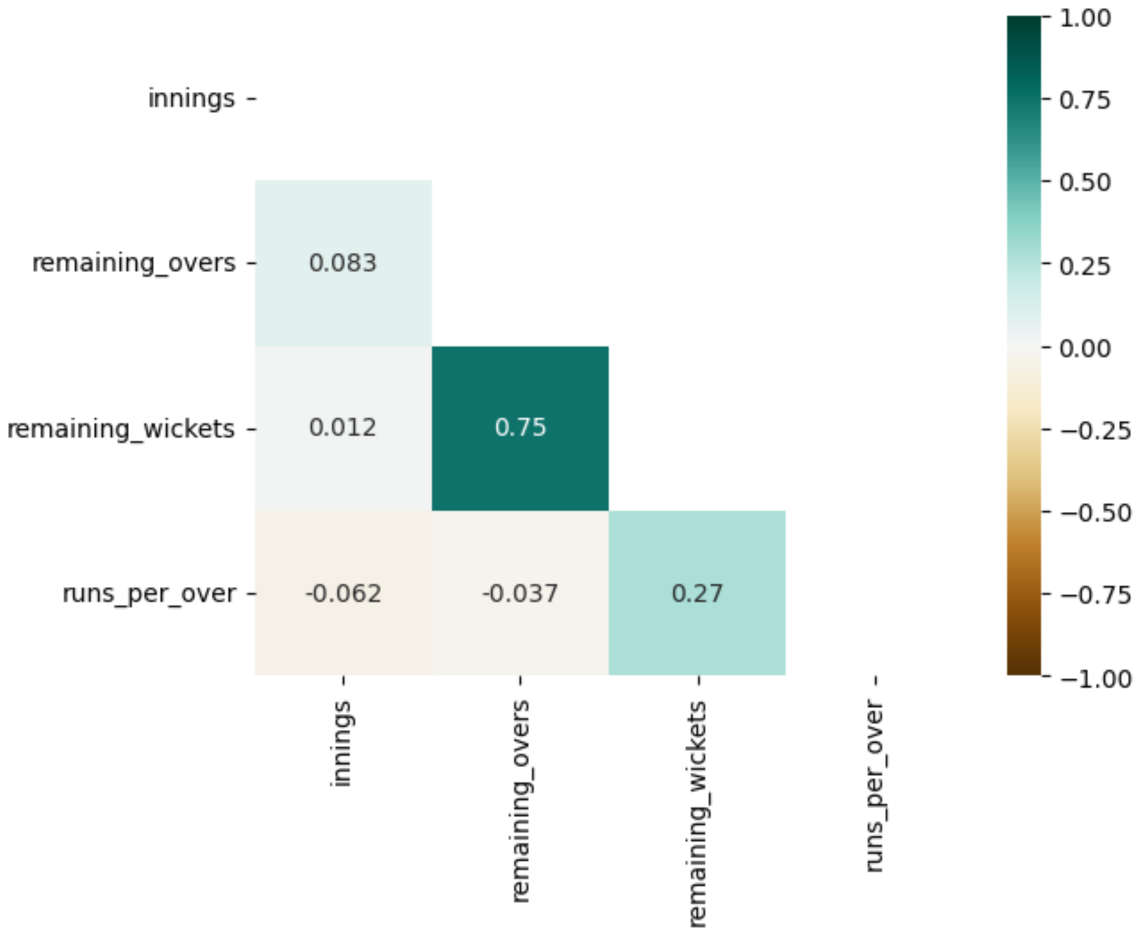
```
import optuna
import tensorflow as tf

from joblib import dump, load
from xgboost import XGBRegressor

from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler

from sklearn.compose import TransformedTargetRegressor
from sklearn.compose import make_column_transformer

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNetCV

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import StackingRegressor

from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score

from keras.models import Sequential
from keras.models import load_model

from keras.layers import Dense
from keras.layers.experimental.preprocessing import Normalization
```

In [7]:

```
input_fields = ["innings","remaining_overs","remaining_wickets"]

X = innings_data[["team", *input_fields, "weight"]]
y = innings_data["runs_per_over"]

innings_data.to_csv('data/innings_results.csv', index=False)
X.drop(columns="weight").to_csv('data/intermediate_data.csv', index=False)
```

In [8]:

```
corr_df = innings_data[["innings","remaining_overs","remaining_wickets","runs_per_

mask = np.triu(np.ones_like(corr_df, dtype=bool))
heatmap = sns.heatmap(corr_df, mask=mask, vmin=-1, vmax=1, annot=True, cmap='BrBG'
```



In [9]:

```
# Make sure some games do not appear in training set

exclude_frac = 0.1
train_test_frac = 0.2

included_matches = innings_data.matchid.drop_duplicates().sample(
    frac=1-exclude_frac, random_state=42
)

included_df = innings_data[ innings_data.matchid.isin(included_matches)]
excluded_df = innings_data[~innings_data.matchid.isin(included_matches)]

def get_X_and_y(input_df):
    X = input_df[[*input_fields, "weight"]]
    y = input_df["runs_per_over"]

    return X, y

X_train, X_test, y_train, y_test = train_test_split(
    *get_X_and_y(included_df), stratify=np.log1p(included_df["runs_per_over"]),
    test_size=(train_test_frac-exclude_frac)/(1-exclude_frac), random_state=42

)

X_excluded, y_excluded = get_X_and_y(excluded_df)
X_test = pd.concat([X_test, X_excluded])
y_test = pd.concat([y_test, y_excluded])

assert np.abs( ( len(X_test) / len(innings_data) ) - ( train_test_frac ) ) < 3e-3
assert np.abs( len(excluded_df) / len(innings_data) - ( exclude_frac ) ) < 3e-3
```

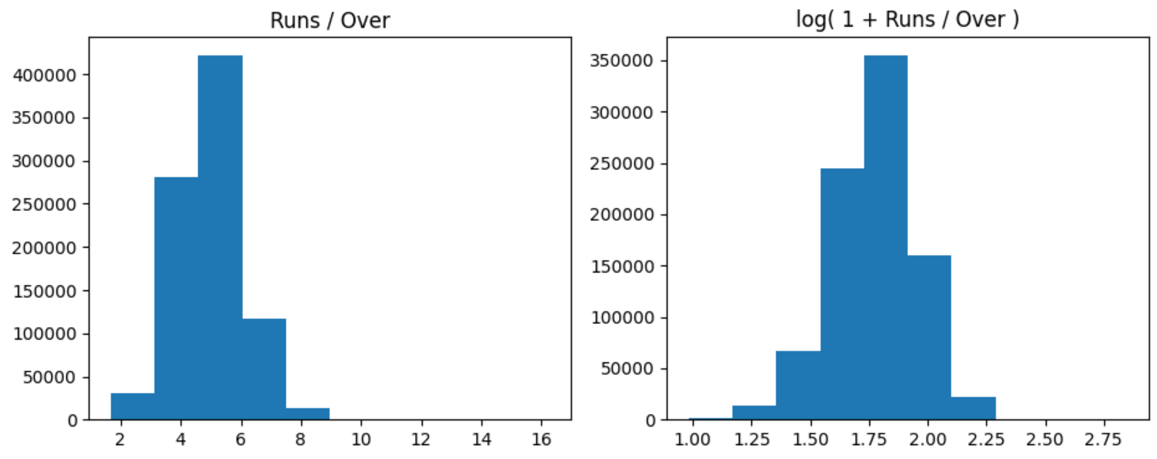
In [10]:

```
# First do Linear Regression as Benchmark

least_squares_model = LinearRegression()
least_squares_model.fit(X_train.drop(columns="weight"), y_train)

y_pred = least_squares_model.predict(X_test.drop(columns="weight"))
score = r2_score(y_test, y_pred)

print("R^2 score for Linear Regression: ", round(score,3))
```

```
R^2 score for Linear Regression:  0.196
```

In [11]:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(11,4))

ax1.hist([
    innings_data["runs_per_over"]
], stacked=False)

ax2.hist([
    np.log1p(innings_data["runs_per_over"])
], stacked=False)

ax1.set_title("Runs / Over")
ax2.set_title("log( 1 + Runs / Over )");
```



In [12]:

```
# Make a more sophisticated ElasticNetCV model

# excluding weights from transformations
elastic_net_scaler = make_column_transformer(
    (StandardScaler(), input_fields),
    (PolynomialFeatures(2), input_fields),
    remainder='passthrough'
)

elastic_net_pipeline = make_pipeline(
    elastic_net_scaler, ElasticNetCV(cv=5, random_state=42)
)

# fitting our model in log space
elastic_net_model = TransformedTargetRegressor(
    regressor=elastic_net_pipeline,
    func=np.log1p, inverse_func=np.expm1
)

weight_kwargs = {elastic_net_pipeline.steps[-1][0] + '__sample_weight': X_train["w
elastic_net_model.fit(X_train.drop(columns="weight"), y_train, **weight_kwargs)

y_pred = elastic_net_model.predict(X_test.drop(columns="weight"))
score = r2_score(y_test, y_pred)

print("R^2 score for ElasticNetCV: ", round(score,3))
```

```
R^2 score for ElasticNetCV:  0.204
```

In [13]:

```
# Try out Random Forest to investigate non-linearities

base_model = RandomForestRegressor(n_jobs=-1)

random_forest_model = TransformedTargetRegressor(
    regressor=base_model, func=np.log1p, inverse_func=np.expm1
)

random_forest_model.fit(X_train.drop(columns="weight"), y_train, sample_weight=X_t

y_pred = random_forest_model.predict(X_test.drop(columns="weight"))
score = r2_score(y_test, y_pred)

print("R^2 score for RandomForest: ", round(score,3))
```

```
R^2 score for RandomForest:  0.213
```

In [14]:

```python
# Run optimization study on xgboost parameters (no log scaling)

def objective(trial):
    n_estimators = trial.suggest_int('n_estimators', 4, 60, log=True)
    max_depth = trial.suggest_int('max_depth', 4, 12)

    model = XGBRegressor(
        n_jobs=-1, n_estimators=n_estimators, max_depth=max_depth
    )

    model.fit(X_train.drop(columns="weight"), y_train, sample_weight=X_train["weig
    y_pred = model.predict(X_test.drop(columns="weight"))

    return r2_score(y_test, y_pred)

max_trials = 8

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=max_trials)

xg_boost_model = XGBRegressor(**study.best_params)
xg_boost_model.fit(X_train.drop(columns="weight"), y_train, sample_weight=X_train[

y_pred = xg_boost_model.predict(X_test.drop(columns="weight"))
score = r2_score(y_test, y_pred)

print("R^2 score for XGBoost: ", round(score,3))
```

```
[I 2023-05-25 04:09:38,794] A new study created in memory with name:
no-name-ab873769-d871-474a-93fc-990db5a49627
[I 2023-05-25 04:09:39,102] Trial 0 finished with value: -0.775520051
4454283 and parameters: {'n_estimators': 4, 'max_depth': 7}. Best is
trial 0 with value: -0.7755200514454283.
[I 2023-05-25 04:09:39,507] Trial 1 finished with value: -0.775309281
793914 and parameters: {'n_estimators': 4, 'max_depth': 10}. Best is
trial 1 with value: -0.775309281793914.
[I 2023-05-25 04:09:45,097] Trial 2 finished with value: 0.2193062720
8151798 and parameters: {'n_estimators': 57, 'max_depth': 11}. Best i
s trial 2 with value: 0.21930627208151798.
[I 2023-05-25 04:09:46,312] Trial 3 finished with value: 0.2222997522
9987764 and parameters: {'n_estimators': 22, 'max_depth': 6}. Best is
trial 3 with value: 0.22229975229987764.
[I 2023-05-25 04:09:47,372] Trial 4 finished with value: 0.2222431530
6195797 and parameters: {'n_estimators': 19, 'max_depth': 6}. Best is
trial 3 with value: 0.22229975229987764.
[I 2023-05-25 04:09:47,687] Trial 5 finished with value: -0.271057844
2876326 and parameters: {'n_estimators': 5, 'max_depth': 6}. Best is
trial 3 with value: 0.22229975229987764.
[I 2023-05-25 04:09:48,133] Trial 6 finished with value: -0.269425641
4638567 and parameters: {'n_estimators': 5, 'max_depth': 9}. Best is
trial 3 with value: 0.22229975229987764.
[I 2023-05-25 04:09:48,665] Trial 7 finished with value: 0.1626766128
2694945 and parameters: {'n_estimators': 8, 'max_depth': 7}. Best is
trial 3 with value: 0.22229975229987764.

R^2 score for XGBoost:  0.222
```

In [15]:

```
# Train a simple neural net model

normalizer = Normalization()
normalizer.adapt(X_train.drop(columns="weight"))

neural_net_model = Sequential()
neural_net_model.add(normalizer)

neural_net_model.add(Dense(64, activation='relu', input_shape=(X_train.drop(column
neural_net_model.add(Dense(32, activation='relu'))
neural_net_model.add(Dense(1))

neural_net_model.compile(optimizer="adam", loss="mse")

max_epochs = 8

neural_net_model.fit(
    X_train.drop(columns="weight"), y_train, verbose=1,
    sample_weight=X_train["weight"], epochs=max_epochs, batch_size=32
)

y_pred = neural_net_model.predict(X_test.drop(columns="weight"))

score = r2_score(y_test, y_pred)

print("R^2 score for NeuralNets: ", round(score,3))
```

```
Epoch 1/8
21644/21644 [==============================] - 9s 382us/step - loss:
0.1614
Epoch 2/8
21644/21644 [==============================] - 8s 358us/step - loss:
0.1527
Epoch 3/8
21644/21644 [==============================] - 8s 380us/step - loss:
0.1526
Epoch 4/8
21644/21644 [==============================] - 8s 372us/step - loss:
0.1524
Epoch 5/8
21644/21644 [==============================] - 8s 375us/step - loss:
0.1522
Epoch 6/8
21644/21644 [==============================] - 8s 365us/step - loss:
0.1521
Epoch 7/8
21644/21644 [==============================] - 8s 358us/step - loss:
0.1521
Epoch 8/8
21644/21644 [==============================] - 7s 342us/step - loss:
0.1520
5346/5346 [==============================] - 1s 215us/step
R^2 score for NeuralNets:  0.215
```

In [16]:

```
# Train another weak learner (R^2 ~ 0.09)
simple_knn_model = KNeighborsRegressor()

simple_knn_model.fit(
    X_train.drop(columns="weight"), y_train
)

estimator_list = [
    ('ls', least_squares_model),
    ('en', elastic_net_model),
    ('rn', random_forest_model),
    ('sk', simple_knn_model),
    ('xg', xg_boost_model)
]

stack_ensemble_regressor = StackingRegressor(
    estimator_list, cv='prefit', n_jobs=-1, verbose=1,
    final_estimator=ElasticNetCV(cv=5, random_state=42)
)

stack_ensemble_regressor.fit(
    X_train.drop(columns="weight"), y_train,
    sample_weight=X_train["weight"]
)

y_pred = stack_ensemble_regressor.predict(X_test.drop(columns="weight"))
score = r2_score(y_test, y_pred)

print("R^2 score for StackedEnsemble: ", round(score,3))
```

```
R^2 score for StackedEnsemble:  0.218
```

In [17]:

```
# Approximate the model we will use in command line

neural_net_model.save("neural_net")
dump(stack_ensemble_regressor, 'data/stack_model.joblib')

loaded_nn_model = load_model('neural_net')
loaded_se_model = load('data/stack_model.joblib')

y_pred = 0.5 * (
    loaded_se_model.predict(X_test.drop(columns="weight")) +
    loaded_nn_model.predict(X_test.drop(columns="weight")).transpose()[0]
)

score = r2_score(y_test, y_pred)

print("R^2 score for FullModel: ", round(score,3))
```

```
INFO:tensorflow:Assets written to: neural_net/assets

INFO:tensorflow:Assets written to: neural_net/assets

5346/5346 [==============================] - 1s 253us/step
R^2 score for FullModel:  0.22
```

Discussion

In this analysis, we developed a model to predict an average cricket team's expected runs per over. Our exploration of various regression techniques - Linear Regression, Elastic Net CV, Random Forest, XGBoost, Neural Networks, and Stacked Ensembles - showcased the merits of each approach. We found that all models significantly outperformed a naive mean-based prediction, with the best model, a combination of Stacked Ensembles and Neural Networks, achieving an R^2 score of 0.22. These results suggest that there are meaningful patterns in the data that our models are beginning to capture.

However, it should be noted that due to the inherent complexity and unpredictability of cricket matches, achieving a much higher R^2 might not be feasible. Further improvements may come from feature engineering or exploring more complex models, but these would also increase the risk of overfitting. The data used for modeling, including team, inning order, remaining overs, and remaining wickets, has been saved in CSV files for further analysis and future use. Our study provides a foundation for future research in this domain and paves the way for developing more sophisticated models for cricket analytics.

Date: May 24, 2023
Author: Dan Segal
Github: @djsegal (https://github.com/djsegal/zelus_assessment)

Zelus Assessment

Part 1: Data Analysis (cont.)

Question 3

More generally and unrelated to cricket or the previous questions, model deployment in a production environment is an important aspect of an engineer's toolkit. Describe a scalable architecture (a diagram may be helpful) that would be appropriate for deploying a model that predicts frame-level play values into a cloud environment with the following assumptions:

- Spatial temporal high frame-rate data (~1 GB per game)
- Play-values are predicted at each frame of a game
- Delivery of game predictions are expected to be delivered overnight
- 500 games per season with 50 games a day
- 5 seasons of existing data
- Model training resources: 8 hour runtime with multiple cores (8) and large memory usage
- Model prediction resources: 60 min runtime per game with a single CPU and 4 GB of memory usage

List out the services, tooling, and reasoning for the choices of architecture. For example, a LAMP stack could be appropriate for an internal home network webpage on a Raspberry Pi.

Answer 3

Architecture

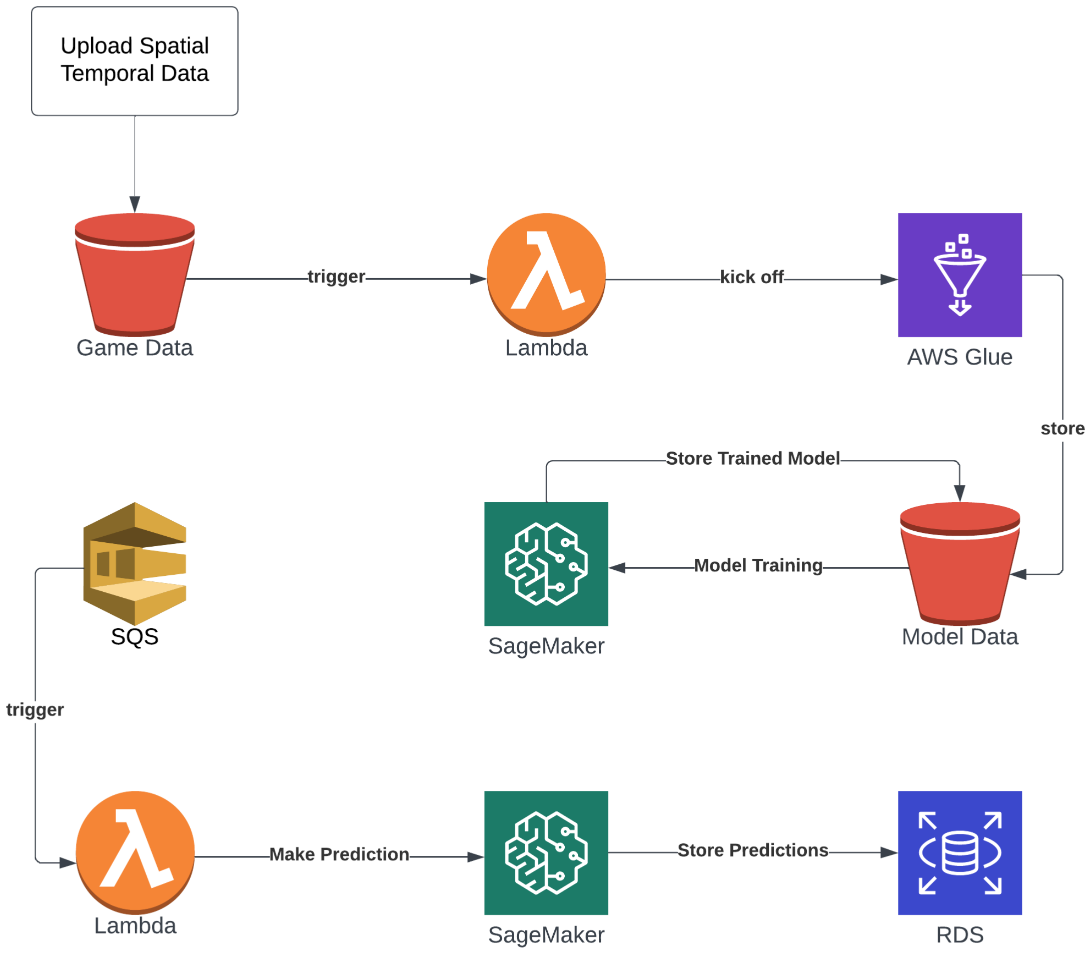
Services:

- Amazon S3 (Simple Storage Service): For storage of game data and model parameters.
- Amazon SageMaker: For model training and prediction.
- AWS Lambda: For event-driven processing.
- Amazon SQS (Simple Queue Service): For task queuing and managing game processing.
- Amazon CloudWatch: For monitoring services.
- Amazon RDS (Relational Database Service): For storing metadata and results.
- AWS Glue: For data extraction, transformation, and load operations.

Architecture Diagram:

In [1]:

```
from IPython.display import Image, display

display(Image(filename='architecture_diagram.png',width=800))
```



Workflow:

1. **Game Data Storage (Amazon S3)**: The high frame-rate data for each game is stored in an Amazon S3 bucket.
2. **ETL (AWS Glue)**: AWS Glue is used to extract the game data from S3, transform it into a suitable format for model training and load it into SageMaker.
3. **Model Training (Amazon SageMaker)**: Amazon SageMaker is used for initial model training. The trained model parameters are stored in Amazon S3.
4. **Game Processing Queue (Amazon SQS)**: Once the game data is available, a message is pushed to an Amazon SQS queue, which triggers the prediction service.
5. **Model Prediction (Amazon SageMaker, AWS Lambda)**: A Lambda function triggers a SageMaker prediction job when a new game is queued. The trained model parameters are loaded from Amazon S3 for each prediction job.
6. **Results and Metadata Storage (Amazon RDS)**: The results of the model prediction along with any relevant metadata are stored in an Amazon RDS database.
7. **Monitoring (Amazon CloudWatch)**: Amazon CloudWatch is used to monitor the SageMaker jobs, SQS queue, and Lambda function. It ensures that any issues are quickly identified and addressed.

Reasoning

The architecture proposed is designed to be scalable, robust, and effective in addressing the given problem. The choice of Amazon S3 as a storage solution for game data and model parameters is influenced by its efficiency, cost-effectiveness, high durability, and availability. This ensures that the large volume of high frame-rate data per game is securely and readily available for processing.

To handle the processing of this data, AWS Glue is selected for its capabilities to perform extract, transform, and load (ETL) operations. It effectively prepares the raw game data for the subsequent model training phase in a format that the model can ingest.

For model training and prediction, Amazon SageMaker, a fully-managed service, is utilized. SageMaker provides a comprehensive suite of machine learning capabilities, allowing for distributed training and real-time predictions. With SageMaker, the burden of managing the underlying infrastructure is removed, enabling the focus to remain on the development and optimization of the model.

The workflow includes Amazon SQS which acts as a buffer, managing the flow of games for prediction and ensuring the prediction service is not overwhelmed. Additionally, AWS Lambda is employed to facilitate event-driven processing, thereby minimizing the delay between when game data is available and when the prediction commences.

In terms of storage, Amazon RDS offers a structured way to store and retrieve data. It provides the needed functionality to house the results of the model prediction and any relevant metadata.

Lastly, the monitoring of the entire architecture is crucial to the prompt identification and resolution of any issues. Amazon CloudWatch provides the necessary services to ensure that the operation of SageMaker jobs, SQS queue, and the Lambda function are closely observed.

The selected services are all part of AWS, ensuring seamless interoperability and reducing complexity compared to a multi-cloud approach. AWS services are known for their robust support for automation, which is an essential factor for a system processing high volumes of data. This combination of services provides a cohesive, efficient, and scalable solution to meet the requirements of the task.

Date: May 24, 2023
Author: Dan Segal
Github: @djsegal (https://github.com/djsegal/zelus_assessment)

Zelus Assessment

Part 2: Deployment

This part of the assessment showcases your ability to deploy a model in a local environment. Please complete Part 2 of the assessment by packaging your working directory in a zip file with the intermediate data file from Q2, model files, and any necessary scripts.

Question 4

Save your model from Q2 into a file and create a packaged solution for being able to build, deploy and run your model locally. We are expecting a solution where local runs can be initiated from the command line, not an API-style deployment. As a way to test your package, create a shell script that takes data saved from Q2, filters for the first 5 Ireland overs, sends them to your model, and displays the model results to `stdout`.

Answer 4

In [1]:

```
import argparse
import pandas as pd

from joblib import dump, load
from keras.models import load_model

# parser = argparse.ArgumentParser(description='Make predictions using trained moc
# parser.add_argument('input_file', type=str, help='Input CSV file')

# args = parser.parse_args()
# input_data = pd.read_csv(args.input_file)

input_data = pd.read_csv("data/intermediate_data.csv")

loaded_nn_model = load_model('neural_net')
loaded_se_model = load('data/stack_model.joblib')

y_pred = 0.5 * (
    loaded_se_model.predict(input_data.drop(columns="team")) +
    loaded_nn_model.predict(input_data.drop(columns="team"), verbose=False).transp
)

# print(list(y_pred))
list(y_pred)
```