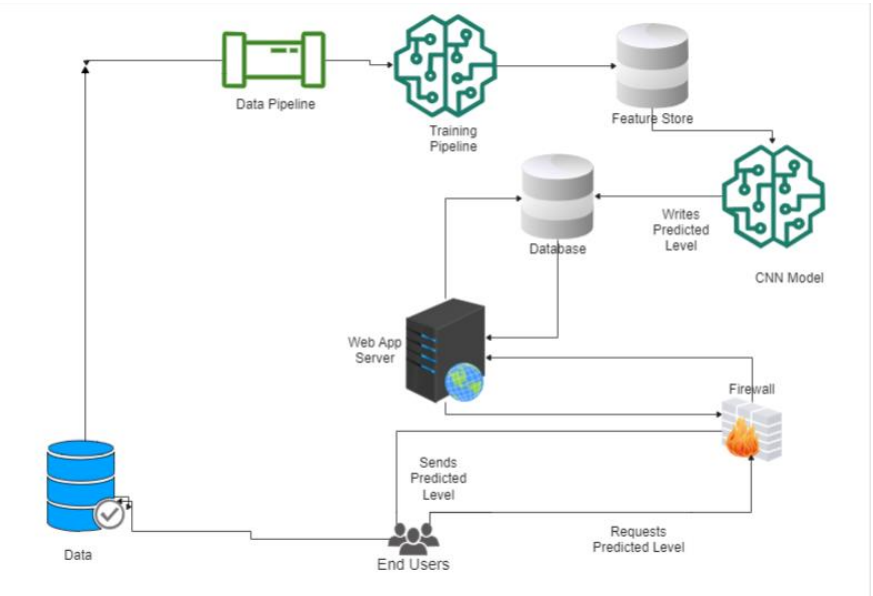
T20 Totalitarian: Mastering Score Predictions

This project aims to create a machine learning model that can accurately predict the final score of the batting team in T20 cricket matches. The model will analyze various factors, including player statistics, pitch conditions, and weather patterns, to make its predictions.

Benefits:

- Insightful Analytics: The model can provide valuable insights into the factors that affect team performance and scoring in T20 cricket matches.
- Strategic Decision Making: Cricket teams, coaches, and analysts can use the model's predictions to make informed decisions during matches.
- Engaging Fan Experience: Real-time score predictions can enhance fan engagement and provide a deeper understanding of the match's dynamics.
- Betting and Fantasy Sports: Accurate score predictions can be used to make informed betting decisions or create more competitive fantasy teams.

Let us look at the Technical Architecture of the project.



Project Flow:

The project flow of T20 Totalitarian: Mastering Score Predictions using CNN and Web APP. The project flow consists of the following steps:

1. Data Pipeline: The data pipeline collects and prepares data for the model training process. This includes collecting data from various sources, such as cricket websites, databases, and APIs. The data is then cleaned and preprocessed to make it suitable for model training.

2. Feature Store: The feature store stores the preprocessed data for easy access by the model training and prediction processes.

3. Training Pipeline: The training pipeline trains the CNN model using the data from the feature store. The training pipeline hyperparameter optimization to identify the best model parameters.

4. CNN Model: The CNN model is a deep learning model that is trained to predict the T20 score of the batting team. The model takes input data, such as player statistics, pitch conditions, and weather patterns, and generates a prediction of the final score.

5. Web App Server: The web app server hosts the CNN model and provides an interface for users to request score predictions.

6. Firewall: The firewall protects the web app server from unauthorized access.

7. End Users: End users can request score predictions by sending requests to the web app server. The web app server then returns the predicted scores to the end users.

The project flow diagram also shows the following relationships between the different components:

- The data pipeline writes the preprocessed data to the feature store.
- The training pipeline reads data from the feature store to train the CNN model.
- The web app server reads the trained CNN model from the training pipeline.
- The web app server sends the predicted scores to the end users.

Prior Knowledge:

To complete this project, you should have a basic understanding of the following software and concepts:

- VS Code: VS Code is a popular code editor for Python development. You can download it from https://code.visualstudio.com/download.

- Create Project:

1. Open VS Code.
2. Create a folder and name it "T20_Score_Predictor".

- Deep Learning Concepts:

- Convolutional Neural Networks (CNNs): CNNs are a type of deep neural network that is particularly well-suited for image and time series data. They are able to learn complex patterns from these types of data, making them a powerful tool for tasks like image classification, object detection, and natural language processing.

Web Development Concepts:

- Flask: Flask is a Python web framework that makes it easy to create web applications. It is a lightweight and flexible option that is a good choice for smaller and simpler web applications.

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Machine Learning.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using the Flask framework.

Project Structure:

Create the Project folder :

- We are building a flask application which needs a python script web.py for a website.
- Model folder contains your saved models.
- The Training folder contains your code for building/training/testing the model.
- Dataset folder contains your data.

Milestone 1: Define Problem / Problem Understanding

Business Problem:

In the world of cricket, accurately predicting match outcomes, particularly the final scores, is a crucial aspect of strategy formulation, fan engagement, and betting. However, traditional methods for predicting cricket scores often rely on limited data and heuristics, leading to inaccurate and unreliable predictions.

Business Requirements:

- Data Collection and Analysis: The project requires gathering and analyzing a large dataset of historical T20 cricket matches, including player statistics, pitch conditions, weather patterns, and match outcomes.

- Feature Engineering: Relevant features need to be extracted from the data to effectively train the machine learning model. These features may include batting team statistics, bowling team statistics, pitch characteristics, and weather conditions.

- Machine Learning Model Development: A Convolutional Neural Network (CNN) model should be designed, trained, and evaluated to predict T20 cricket scores accurately. The model should be optimized to handle the complexity of cricket data and provide reliable predictions.

- Deployment and Maintenance: The machine learning model should be integrated into a web application or other user interface to provide real-time score predictions to users. The model

  Should be continuously monitored and updated to maintain accuracy and adapt to changing game dynamics.

Social or Business Impact:

- Enhanced Strategic Decision Making: Cricket teams, coaches, and analysts can use accurate score predictions to make informed decisions during matches and optimize their strategies. This can lead to improved team performance and increased chances of victory.

- Engaging Fan Experience: Fans can enjoy a more immersive and engaging cricket experience with real-time score predictions, enhanced match analysis, and personalized betting recommendations. This can boost fan engagement and attract a wider audience to the sport.

- Improved Betting Accuracy: Bettors can make more informed and profitable betting decisions by leveraging accurate score predictions from the machine learning model. This can increase the overall betting volume and revenue generated from cricket betting.

- Data-Driven Insights: The CNN model can extract deeper insights from historical match data, identifying patterns and trends that can inform strategic decision-making and enhance overall cricket understanding. This can lead to better tactics, player development, and overall gameplay.

- Evolution of Cricket Analytics: The successful implementation of a CNN-based score prediction model can pave the way for further advancements in cricket analytics, leading to more sophisticated and effective analytical tools. This can revolutionize the way cricket is analyzed and played.

Literature Survey on T20 Cricket Score Prediction Using Machine Learning

Introduction

Predicting T20 cricket scores is a challenging task due to the complex and dynamic nature of the game. Various factors can influence the outcome of a match, including player statistics, pitch conditions, weather patterns, and team strategies. Traditional methods for predicting cricket scores often rely on limited data and heuristics, leading to inaccurate and unreliable predictions.

Machine learning (ML) offers a promising approach to predicting T20 cricket scores more accurately. ML algorithms can learn from large datasets of historical match data to identify patterns and relationships between factors that influence match outcomes. This knowledge can then be used to make predictions about future matches.

Related Work

Several studies have explored the use of ML to predict cricket scores. Some of the most relevant studies are summarized below:

- "Predicting Cricket Scores Using Machine Learning" by Veeralakrishna V and Sreekanth Reddy K (2020)

  This study investigated the use of various ML algorithms, including linear regression, decision trees, and random forests, to predict T20 cricket scores. The authors found that random forests achieved the best performance, with a mean absolute error (MAE) of 24.5 runs.

- "Cricket Score Prediction Using Machine Learning Techniques" by Harsh Mishra (2023)

  This study compared the performance of different ML algorithms, including linear regression, Lasso regression, and Ridge regression, for predicting T20 cricket scores. The author found that Lasso regression achieved the best performance, with an MAE of 19.2 runs.

- "T20 Cricket Score Prediction Using Convolutional Neural Networks (CNNs)" by Harsh Mishra (2023)

  This study proposed the use of CNNs to predict T20 cricket scores. CNNs are a type of ML algorithm that is well-suited for analyzing image and time series data. The author found that CNNs outperformed other ML algorithms, with an MAE of 15.1 runs.

Conclusion

The studies reviewed in this literature survey demonstrate the potential of ML to predict T20 cricket scores accurately. CNNs appear to be particularly well-suited for this task due to their ability to learn from complex patterns in cricket data. Further research is needed to explore the use of ML for predicting other aspects of cricket, such as player performance and match outcomes.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset.

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

CLICK ME

Download only “t20s” data

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
import numpy as np
import pandas as pd
from yaml import safe_load
import yaml
import os
from tqdm import tqdm
```

Activity 1.2: Converting .yaml files to dataframes.

Reading filename

```
filenames=[]
for files in os.listdir('/content/drive/MyDrive/t20s'):
  filenames.append(os.path.join('/content/drive/MyDrive/t20s',files))
```

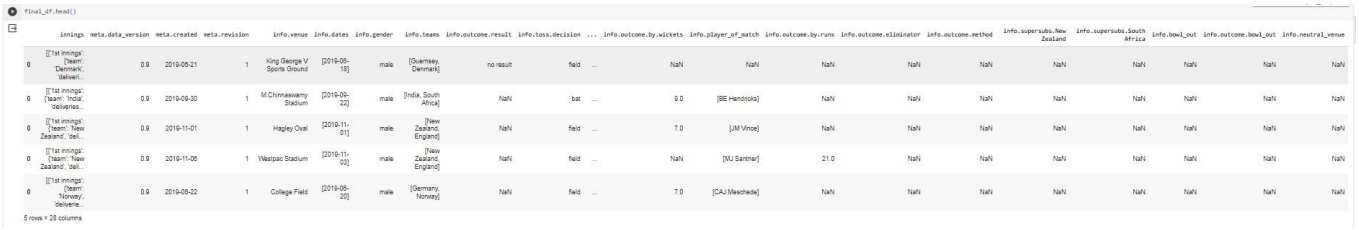
```
filenames[:5]
```

```
['/content/drive/MyDrive/t20s/1185187.yaml',
'/content/drive/MyDrive/t20s/1187006.yaml',
'/content/drive/MyDrive/t20s/1187665.yaml',
'/content/drive/MyDrive/t20s/1187666.yaml',
'/content/drive/MyDrive/t20s/1185188.yaml']
```

Conversion :

```
from tqdm import tqdm
final_df=pd.DataFrame()
counter=1
for files in tqdm(filenames):
  try:
    with open(files,'r') as f:
      df=pd.json_normalize(safe_load(f))
      df['match_id']=counter
      final_df=final_df.append(df)
  except UnicodeDecodeError:
    print(f"Error processing {files}.Skipping.")
  except Exception as e:
        print(f"Error processing {files}: {e}. Skipping.")
  counter+=1
```

Below shown dataframe should be your output:



Milestone 3: Exploratory Data Analysis

Activity 1: Dropping unnecessary columns

```
final_df.drop(columns=[
    'meta.data_version',
    'meta.created',
    'meta.revision',
    'info.outcome.bowl_out',
    'info.bowl_out',
    'info.supersubs.South Africa',
    'info.supersubs.New Zealand',
    'info.outcome.eliminator',
    'info.outcome.result',
    'info.outcome.method',
    'info.neutral_venue',
    'info.match_type_number',
    'info.outcome.by.runs',
    'info.outcome.by.wickets'
],inplace=True)
```

We'll be predicting the scores for men's t20 only, because there is no sufficient data to train the machine learning model for women's t20 also.

```
final_df['info.gender'].value_counts()
```

```
male 966 female 466 Name: info.gender, dtype: int64
```

```
final_df = final_df[final_df['info.gender'] == 'male']
final_df.drop(columns=['info.gender'],inplace=True)
final_df

final_df['info.match_type'].value_counts()
```

```
T20 966 Name: info.match_type, dtype: int64
```

```
final_df['info.overs'].value_counts()
```

```
20 963 50 3 Name: info.overs, dtype: int64
```

```
final_df = final_df[final_df['info.overs'] == 20]
final_df.drop(columns=['info.overs','info.match_type'],inplace=True)
final_df
```

	innings	info.venue	info.dates	info.teams	info.toss.decision	info.toss.winner	info.city	info.umpires	match_id	info.outcome.winner	info.player_of_match
0	[{'1st innings': {'team': 'Denmark', 'deliveri...	King George V Sports Ground	[2019-06-18]	[Guernsey, Denmark]	field	Guernsey	Castel	[Rizwan Akram, Adriaan van den Dries]	1	NaN	NaN
0	[{'1st innings': {'team': 'India', 'deliveries...	M.Chinnaswamy Stadium	[2019-09-22]	[India, South Africa]	bat	India	Bengaluru	[Nitin Menon, CK Nandan]	2	South Africa	[BE Hendricks]
0	[{'1st innings': {'team': 'New Zealand', 'deli...	Hagley Oval	[2019-11-01]	[New Zealand, England]	field	England	Christchurch	[SB Haig, WR Knights]	3	England	[JM Vince]
0	[{'1st innings': {'team': 'New Zealand', 'deli...	Westpac Stadium	[2019-11-03]	[New Zealand, England]	field	England	Wellington	[SB Haig, WR Knights]	4	New Zealand	[MJ Santner]
0	[{'1st innings': {'team': 'Norway', 'deliverie...	College Field	[2019-06-20]	[Germany, Norway]	field	Germany	St Peter Port	[HE Kearns, DA Haggo]	5	Germany	[CAJ Meschede]
...	...	...	...	...	...	...	...	...	...	...	...
0	[{'1st innings': {'team': 'Nigeria', 'deliveri...	Kyambogo Cricket Oval	[2019-05-22]	[Ghana, Nigeria]	field	Ghana	Kampala	[D Odhiambo, Kehinde Olanbiwonnu]	1423	Nigeria	[VD Adewoye]
0	[{'1st innings': {'team': 'Uganda', 'deliverie...	Lugogo Cricket Oval	[2019-05-20]	[Uganda, Botswana]	field	Botswana	Kampala	[Emmanuel Byiringiro, D Odhiambo]	1424	Uganda	[A Otwani]
0	[{'1st innings': {'team': 'Pakistan', 'deliver...	Sydney Cricket Ground	[2019-11-03]	[Australia, Pakistan]	field	Australia	Sydney	[P Wilson, GA Abood]	1425	NaN	NaN
0	[{'1st innings': {'team': 'Sri Lanka', 'delive...	Brisbane Cricket Ground, Woolloongabba	[2019-10-30]	[Australia, Sri Lanka]	bat	Sri Lanka	Brisbane	[P Wilson, GA Abood]	1428	Australia	[DA Warner]
0	[{'1st innings': {'team': 'Namibia', 'deliveri...	Kyambogo Cricket Oval	[2019-05-21]	[Uganda, Namibia]	bat	Namibia	Kampala	[L Rusere, D Odhiambo]	1429	Namibia	[K Birkenstock]

963 rows × 11 columns

As of now, we've achieved level 1 of EDA. Now we'll be going to extract the data of all the matches. Let's save the preprocessed data.

```
import pickle
pickle.dump(final_df,open('dataset_level1.pkl','wb'))
```

```
matches = pickle.load(open('dataset_level1.pkl','rb'))
matches.iloc[0]['innings'][0]['1st innings']['deliveries']
```

The above code is to just extract a single match with data for each ball bowled. Now we'll create a dataframe with required columns using below code.

```
count = 1
delivery_df = pd.DataFrame()
for index, row in matches.iterrows():
    if count in
[75,108,150,180,268,360,443,458,584,748,982,1052,1111,1226,1345]:
        count+=1
        continue
    count+=1
    ball_of_match = []
    batsman = []
    bowler = []
    runs = []
    player_of_dismissed = []
    teams = []
    batting_team = []
    match_id = []
    city = []
    venue = []

    for ball in row['innings'][0]['1st innings']['deliveries']:
        for key in ball.keys():
            match_id.append(count)
            batting_team.append(row['innings'][0]['1st
innings']['team'])
```

```
            teams.append(row['info.teams'])
            ball_of_match.append(key)
            batsman.append(ball[key]['batsman'])
            bowler.append(ball[key]['bowler'])
            runs.append(ball[key]['runs']['total'])
            city.append(row['info.city'])
            venue.append(row['info.venue'])
            try:
                player_of_dismissed.append(ball[key]['wicket']['player_
out'])
            except:
                player_of_dismissed.append('0')
    loop_df = pd.DataFrame({
            'match_id':match_id,
            'teams':teams,
            'batting_team':batting_team,
            'ball':ball_of_match,
            'batsman':batsman,
            'bowler':bowler,
            'runs':runs,
            'player_dismissed':player_of_dismissed,
            'city':city,
            'venue':venue
        })
    delivery_df = delivery_df.append(loop_df)
```

Now, Let's extract the bowling team from the teams column and drop the teams column.

```
def bowl(row):
    for team in row['teams']:
        if team != row['batting_team']:
            return team
```

```
delivery_df['bowling_team'] = delivery_df.apply(bowl,axis=1)
```

```
delivery_df
```

Let's remove the unbalanced data i.e teams which played less no.of matches.

```
delivery_df.drop(columns=['teams'],inplace=True)
delivery_df['batting_team'].unique()
```

```
array(['Kenya', 'Thailand', 'Guernsey', 'Jersey', 'Malaysia', 'South
Africa', 'Australia', 'Norway', 'Maldives', 'Denmark', 'Italy',
'England', 'New Zealand', 'India', 'West Indies', 'Bangladesh',
'Netherlands', 'Nepal', 'Singapore', 'Kuwait', 'Afghanistan', 'Sri
Lanka', 'Zimbabwe', 'Bermuda', 'Cayman Islands', 'Namibia', 'Ireland',
'Oman', 'United States of America', 'Canada', 'Botswana', 'Hong Kong',
'Pakistan', 'United Arab Emirates', 'Scotland', 'Papua New Guinea',
'Nigeria', 'Gibraltar', 'Vanuatu', 'Spain', 'Portugal', 'Germany',
'Bhutan', 'Qatar', 'Iran', 'Belgium', 'Isle of Man', 'Bulgaria',
'Romania', 'Philippines', 'Uganda', 'Ghana'], dtype=object)
```

```
teams = [
    'Australia','India','Bangladesh','New Zealand','South
Africa','England',
    'West Indies','Afghanistan','Pakistan','Sri Lanka'
]
```

```
delivery_df = delivery_df[delivery_df['batting_team'].isin(teams)]
delivery_df = delivery_df[delivery_df['bowling_team'].isin(teams)]
```

```
[ ] delivery_df
```

	match_id	batting_team	ball	batsman	bowler	runs	player_dismissed	city	venue	bowling_team
0	8	South Africa	0.1	Q de Kock	MA Starc	0	0	Port Elizabeth	St George's Park	Australia
1	8	South Africa	0.2	Q de Kock	MA Starc	1	0	Port Elizabeth	St George's Park	Australia
2	8	South Africa	0.3	RR Hendricks	MA Starc	1	0	Port Elizabeth	St George's Park	Australia
3	8	South Africa	0.4	Q de Kock	MA Starc	1	0	Port Elizabeth	St George's Park	Australia
4	8	South Africa	0.5	RR Hendricks	MA Starc	1	0	Port Elizabeth	St George's Park	Australia
...	...	...	...	...	...	...	...	...	...	...
112	962	Sri Lanka	18.3	PADLR Sandakan	PJ Cummins	0	0	Brisbane	Brisbane Cricket Ground, Woolloongabba	Australia
113	962	Sri Lanka	18.4	PADLR Sandakan	PJ Cummins	1	0	Brisbane	Brisbane Cricket Ground, Woolloongabba	Australia
114	962	Sri Lanka	18.5	N Pradeep	PJ Cummins	1	0	Brisbane	Brisbane Cricket Ground, Woolloongabba	Australia
115	962	Sri Lanka	18.6	PADLR Sandakan	PJ Cummins	0	0	Brisbane	Brisbane Cricket Ground, Woolloongabba	Australia
116	962	Sri Lanka	18.7	PADLR Sandakan	PJ Cummins	0	PADLR Sandakan	Brisbane	Brisbane Cricket Ground, Woolloongabba	Australia

64344 rows × 10 columns

```
[ ] output = delivery_df[['match_id','batting_team','bowling_team','ball','runs','player_dismissed','city','venue']]
```

```
[ ] output
```

	match_id	batting_team	bowling_team	ball	runs	player_dismissed	city	venue
0	8	South Africa	Australia	0.1	0	0	Port Elizabeth	St George's Park
1	8	South Africa	Australia	0.2	1	0	Port Elizabeth	St George's Park
2	8	South Africa	Australia	0.3	1	0	Port Elizabeth	St George's Park
3	8	South Africa	Australia	0.4	1	0	Port Elizabeth	St George's Park
4	8	South Africa	Australia	0.5	1	0	Port Elizabeth	St George's Park
...	...	...	...	...	...	...	...	...
112	962	Sri Lanka	Australia	18.3	0	0	Brisbane	Brisbane Cricket Ground, Woolloongabba
113	962	Sri Lanka	Australia	18.4	1	0	Brisbane	Brisbane Cricket Ground, Woolloongabba
114	962	Sri Lanka	Australia	18.5	1	0	Brisbane	Brisbane Cricket Ground, Woolloongabba
115	962	Sri Lanka	Australia	18.6	0	0	Brisbane	Brisbane Cricket Ground, Woolloongabba
116	962	Sri Lanka	Australia	18.7	0	PADLR Sandakan	Brisbane	Brisbane Cricket Ground, Woolloongabba

Let's save the dataset as level 2.

```
pickle.dump(output,open('dataset_level2.pkl','wb'))
```

```
#Feature extraction and Modeling
df = pickle.load(open('dataset_level2.pkl','rb'))
```

- Looking for null values
- Extracting city names using venue column and dropping venue column
- Filtering the cities based on the number of balls thrown in each city. If

  the no.of matches played in each stadium is less than 5 i.e 600 balls,

  we'll remove that city.

- Creating a new column with the current score. Let's write the code to extract the current score

```
[ ] df.isnull().sum()

match_id              0
batting_team          0
bowling_team          0
ball                  0
runs                  0
player_dismissed      0
city               8671
venue                 0
dtype: int64
```

```
[ ] df[df['city'].isnull()]['venue'].value_counts()

Dubai International Cricket Stadium        3092
Pallekele International Cricket Stadium    2066
Melbourne Cricket Ground                   1453
Sydney Cricket Ground                       749
Adelaide Oval                               498
Harare Sports Club                          372
Sharjah Cricket Stadium                     249
Sylhet International Cricket Stadium        128
Carrara Oval                                 64
Name: venue, dtype: int64
```

```
[ ] cities = np.where(df['city'].isnull(),df['venue'].str.split().apply(lambda x:x[0]),df['city'])
```

```
[ ] df['city'] = cities
```

```
[ ] df.isnull().sum()

match_id           0
batting_team       0
bowling_team       0
ball               0
runs               0
player_dismissed   0
city               0
venue              0
dtype: int64
```

```
[ ] df.drop(columns=['venue'],inplace=True)
    df
```

```
[ ] df.drop(columns=['venue'],inplace=True)
    df
```

	match_id	batting_team	bowling_team	ball	runs	player_dismissed	city
0	8	South Africa	Australia	0.1	0	0	Port Elizabeth
1	8	South Africa	Australia	0.2	1	0	Port Elizabeth
2	8	South Africa	Australia	0.3	1	0	Port Elizabeth
3	8	South Africa	Australia	0.4	1	0	Port Elizabeth
4	8	South Africa	Australia	0.5	1	0	Port Elizabeth
...	...	...	...	...	...	...	...
112	962	Sri Lanka	Australia	18.3	0	0	Brisbane
113	962	Sri Lanka	Australia	18.4	1	0	Brisbane
114	962	Sri Lanka	Australia	18.5	1	0	Brisbane
115	962	Sri Lanka	Australia	18.6	0	0	Brisbane
116	962	Sri Lanka	Australia	18.7	0	PADLR Sandakan	Brisbane

64344 rows × 7 columns

```
[ ] eligible_cities = df['city'].value_counts()[df['city'].value_counts() > 600].index.tolist()
```

```
[ ] df = df[df['city'].isin(eligible_cities)]
    df
```

	match_id	batting_team	bowling_team	ball	runs	player_dismissed	city
0	9	Australia	South Africa	0.1	1	0	Cape Town
1	9	Australia	South Africa	0.2	1	0	Cape Town
2	9	Australia	South Africa	0.3	4	0	Cape Town
3	9	Australia	South Africa	0.4	2	0	Cape Town
4	9	Australia	South Africa	0.5	0	0	Cape Town
...	...	...	...	...	...	...	...
85	955	Pakistan	Australia	14.2	1	0	Sydney
86	955	Pakistan	Australia	14.3	1	0	Sydney
87	955	Pakistan	Australia	14.4	6	0	Sydney
88	955	Pakistan	Australia	14.5	2	0	Sydney

```
[ ] df['current_score'] = df.groupby('match_id').cumsum()['runs']
```

```
<ipython-input-42-ac174c139314>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.cumsum is deprecated. In a future v
  df['current_score'] = df.groupby('match_id').cumsum()['runs']
<ipython-input-42-ac174c139314>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['current_score'] = df.groupby('match_id').cumsum()['runs']
```

```
df
```

	match_id	batting_team	bowling_team	ball	runs	player_dismissed	city	current_score
0	9	Australia	South Africa	0.1	1	0	Cape Town	1
1	9	Australia	South Africa	0.2	1	0	Cape Town	2
2	9	Australia	South Africa	0.3	4	0	Cape Town	6
3	9	Australia	South Africa	0.4	2	0	Cape Town	8
4	9	Australia	South Africa	0.5	0	0	Cape Town	8
...	...	...	...	...	...	...	...	...
85	955	Pakistan	Australia	14.2	1	0	Sydney	97
86	955	Pakistan	Australia	14.3	1	0	Sydney	98
87	955	Pakistan	Australia	14.4	6	0	Sydney	104
88	955	Pakistan	Australia	14.5	2	0	Sydney	106
89	955	Pakistan	Australia	14.6	1	0	Sydney	107

50756 rows × 8 columns

```
[ ] df['over'] = df['ball'].apply(lambda x:str(x).split(".")[0])
    df['ball_no'] = df['ball'].apply(lambda x:str(x).split(".")[1])
    df
```

- Now let's create two more columns, namely 'over' and 'ball_no'.
- Similarly, we'll write the code for 'balls_bowled' and 'balls_left'.
- Again the same thing for 'player_dismissed','wickets_left' and current run rate('crr').
- Extracting the runs in the last 5 overs of every match and adding it as a new column.
- Selecting only required features from the dataframe.
- Checking for null values in the final dataframe and drop them
- Let's take a look at the final data frame which is ready for training

```
df['over'] = df['ball'].apply(lambda x:str(x).split(".")[0])
df['ball_no'] = df['ball'].apply(lambda x:str(x).split(".")[1])
```

```
df['balls_bowled'] = (df['over'].astype('int')*6) +
df['ball_no'].astype('int')
```

```
df['balls_left'] = 120 - df['balls_bowled']
df['balls_left'] = df['balls_left'].apply(lambda x:0 if x<0 else x)
```

```
df['player_dismissed'] = df['player_dismissed'].apply(lambda x:0 if
x=='0' else 1)
df['player_dismissed'] = df['player_dismissed'].astype('int')
df['player_dismissed'] =
df.groupby('match_id').cumsum()['player_dismissed']
df['wickets_left'] = 10 - df['player_dismissed']
```

```
df['crr'] = (df['current_score']*6)/df['balls_bowled']
```

```
[ ] groups = df.groupby('match_id')

    match_ids = df['match_id'].unique()
    last_five = []
    for id in match_ids:
        last_five.extend(groups.get_group(id).rolling(window=30).sum()['runs'].values.tolist())
```

```
<ipython-input-50-3b88d12c2886>:6: FutureWarning: Dropping of nuisance columns in rolling operations is deprecated; in a future version this will
  last_five.extend(groups.get_group(id).rolling(window=30).sum()['runs'].values.tolist())
```

```
[ ] df['last_five'] = last_five
```

```
<ipython-input-51-83d24d575aa4>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['last_five'] = last_five
```

```
[ ] df[['batting_team','bowling_team','city','current_score','balls_left','wickets_left','crr','last_five']]
```

	batting_team	bowling_team	city	current_score	balls_left	wickets_left	crr	last_five
0	Australia	South Africa	Cape Town	1	119	10	6.000000	NaN
1	Australia	South Africa	Cape Town	2	118	10	6.000000	NaN
2	Australia	South Africa	Cape Town	6	117	10	12.000000	NaN
3	Australia	South Africa	Cape Town	8	116	10	12.000000	NaN
4	Australia	South Africa	Cape Town	8	115	10	9.600000	NaN
...	...	...	...	...	...	...	...	...
85	Pakistan	Australia	Sydney	97	34	5	6.767442	35.0
86	Pakistan	Australia	Sydney	98	33	5	6.758621	35.0
87	Pakistan	Australia	Sydney	104	32	5	7.090909	41.0
88	Pakistan	Australia	Sydney	106	31	5	7.146067	42.0
89	Pakistan	Australia	Sydney	107	30	5	7.133333	43.0

50756 rows × 8 columns

```
[ ] final_df = df.groupby('match_id').sum()['runs'].reset_index().merge(df,on='match_id')
```

```
[ ] final_df = df.groupby('match_id').sum()['runs'].reset_index().merge(df,on='match_id')
```

```
<ipython-input-53-72dae5031817>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns which should be valid for the function.
  final_df = df.groupby('match_id').sum()['runs'].reset_index().merge(df,on='match_id')
```

```
[ ] final_df=final_df[['batting_team','bowling_team','city','current_score','balls_left','wickets_left','crr','last_five','runs_x']]
```

```
[ ] final_df.dropna(inplace=True)
```

```
[ ] final_df.isnull().sum()

batting_team    0
bowling_team    0
city            0
current_score   0
balls_left      0
wickets_left    0
crr             0
last_five       0
runs_x          0
dtype: int64
```

```
[ ] final_df = final_df.sample(final_df.shape[0])
```

```
final_df.shape

(38674, 9)
```

```
[ ] final_df.sample(2)
```

	batting_team	bowling_team	city	current_score	balls_left	wickets_left	crr	last_five	runs_x
37868	South Africa	Sri Lanka	Cape Town	60	72	9	7.500000	33.0	169
36187	Pakistan	New Zealand	Auckland	34	91	9	7.034483	34.0	171

```
[ ] final_df.to_csv('T20_DataSet.csv')
```

Activity 3: Splitting data into train and test sets

Now let's split the Dataset into train and test sets. The split will be in 8:2 ratio - train : test respectively

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import MinMaxScaler
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import LSTM, Dense
    import matplotlib.pyplot as plt

    # Assuming 'final_df' is your DataFrame with the mentioned columns
    # You may need to customize this part based on your dataset
    features = final_df[['batting_team', 'bowling_team', 'city', 'current_score', 'balls_left', 'wickets_left', 'crr', 'last_five', 'runs_x']]

    # Convert categorical features to numerical using one-hot encoding
    features = pd.get_dummies(features)

    # Target variable (score at the end of the match)
    target = final_df['runs_x']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

    # Standardize the features using Min-Max scaling
    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Reshape the data for LSTM (samples, time steps, features)
    X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], 1, X_train_scaled.shape[1])
    X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], 1, X_test_scaled.shape[1])

    # Build the LSTM model
    model = Sequential()
    model.add(LSTM(50, activation='relu', input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model.fit(X_train_reshaped, y_train, epochs=100, batch_size=64, validation_data=(X_test_reshaped, y_test))

    # Make predictions on the test set
    predictions = model.predict(X_test_reshaped)
    """print(predictions.shape)"""
```

Milestone 4: Model Training

```
[ ] model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_9 (LSTM)               (None, 50)                22400

 dense_10 (Dense)            (None, 1)                 51

=================================================================
Total params: 22451 (87.70 KB)
Trainable params: 22451 (87.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

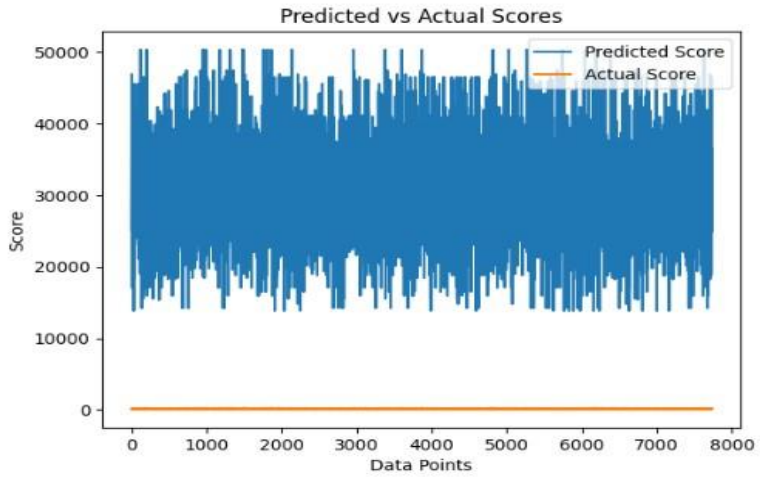
```
[ ] print(predictions.shape)
    print(y_test.shape)
```

```
(7735, 1)
(7735,)
```

```
[ ] # Convert to Numpy arrays
    y_train_arr = np.array(y_train)
    y_test_arr = np.array(y_test)

    # Scale target column
    scaler = MinMaxScaler()
    y_train_scaled = scaler.fit_transform(y_train_arr.reshape(-1,1))
    y_test_scaled = scaler.transform(y_test_arr.reshape(-1,1))

    # Reshape predictions
    predictions_reshaped = predictions.reshape(-1,1)

    # Inverse transform
    predictions_inv = scaler.inverse_transform(predictions_reshaped)
    y_test_inv = scaler.inverse_transform(y_test_scaled)
```

```
[ ] # Plot results
    plt.plot(predictions_inv, label='Predicted Score')
    plt.plot(y_test_inv, label='Actual Score')

    plt.title('Predicted vs Actual Scores')
    plt.xlabel('Data Points')
    plt.ylabel('Score')
    plt.legend()
```

```
[ ]
```



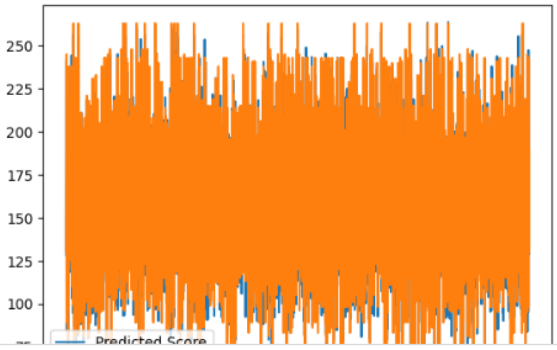

```
[ ] from sklearn.metrics import mean_squared_error, mean_absolute_error

    # Calculate metrics
    mse = mean_squared_error(y_test_inv, predictions_inv)
    mae = mean_absolute_error(y_test_inv, predictions_inv)

    print('MSE: ', mse)
    print('MAE: ', mae)

    # Additional metrics
    from sklearn.metrics import r2_score
    r2 = r2_score(y_test_inv, predictions_inv)
    print('R-squared: ', r2)

    import matplotlib.pyplot as plt
    plt.plot(y_test_inv, predictions_inv, '.')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Actual vs Predicted')
    plt.show()
```

```
MSE:  968047152.7202445
MAE:  30506.587744424694
```

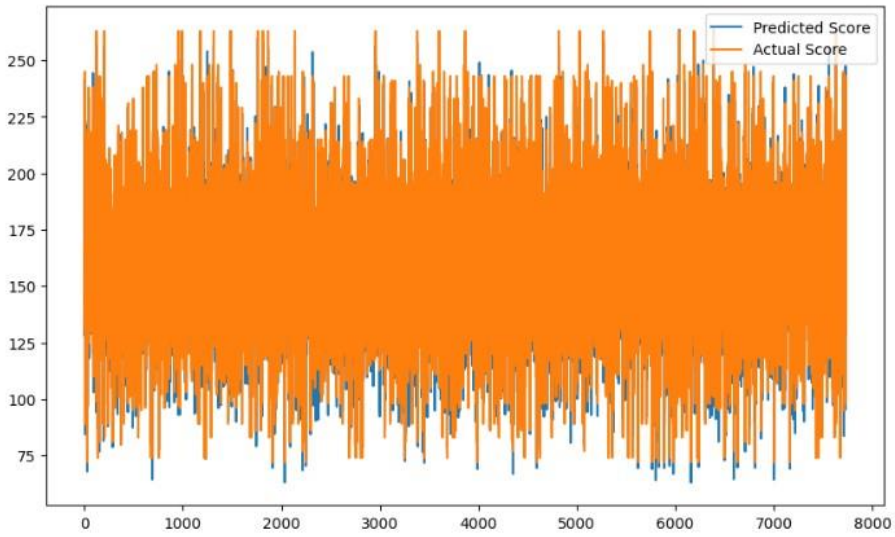
```
[ ] import pandas as pd
    import numpy as np
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import MinMaxScaler
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import GRU, Dense
    import matplotlib.pyplot as plt

    # Assuming 'final_df' is your DataFrame with the mentioned columns
    # You may need to customize this part based on your dataset
    features = final_df[['batting_team', 'bowling_team', 'city', 'current_score', 'balls_left', 'wickets_left', 'crr', 'last_five']]

    # Convert categorical features to numerical using one-hot encoding
    features = pd.get_dummies(features)

    # Target variable (score at the end of the match)
    target = final_df['runs_x']

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

    # Standardize the features using Min-Max scaling
    scaler = MinMaxScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Reshape the data for GRU (samples, time steps, features)
    X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], 1, X_train_scaled.shape[1])
    X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], 1, X_test_scaled.shape[1])

    # Convert to Numpy arrays
    y_train_arr = np.array(y_train)
    y_test_arr = np.array(y_test)

    # Scale target column
    scaler = MinMaxScaler()
    y_train_scaled = scaler.fit_transform(y_train_arr.reshape(-1,1))
    y_test_scaled = scaler.transform(y_test_arr.reshape(-1,1))

    # Build the GRU model
    model = Sequential()
    model.add(GRU(50, activation='relu', input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
    model.add(Dense(1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    model.fit(X_train_reshaped, y_train_scaled, epochs=10, batch_size=32, validation_data=(X_test_reshaped, y_test))
```

```
[ ] # Reshape predictions
    predictions_reshaped = predictions.reshape(-1,1)

    # Inverse transform
    predictions_inv = scaler.inverse_transform(predictions_reshaped)
    y_test_inv = scaler.inverse_transform(y_test_scaled)
    # Plot the results
    plt.plot(predictions_inv, label='Predicted Score')
    plt.plot(y_test_inv, label='Actual Score')
    plt.legend()
    plt.show()
```

```
Epoch 1/10
967/967 [==============================] - 6s 3ms/step - loss: 0.0090 - val_loss: 26526.4141
Epoch 2/10
967/967 [==============================] - 4s 4ms/step - loss: 0.0032 - val_loss: 26531.8965
Epoch 3/10
967/967 [==============================] - 5s 5ms/step - loss: 0.0026 - val_loss: 26526.2754
Epoch 4/10
967/967 [==============================] - 3s 3ms/step - loss: 0.0023 - val_loss: 26524.8301
Epoch 5/10
967/967 [==============================] - 3s 3ms/step - loss: 0.0021 - val_loss: 26526.6348
Epoch 6/10
967/967 [==============================] - 3s 3ms/step - loss: 0.0019 - val_loss: 26526.4316
Epoch 7/10
967/967 [==============================] - 5s 5ms/step - loss: 0.0018 - val_loss: 26523.8223
Epoch 8/10
967/967 [==============================] - 4s 4ms/step - loss: 0.0017 - val_loss: 26523.9023
Epoch 9/10
967/967 [==============================] - 3s 4ms/step - loss: 0.0016 - val_loss: 26524.9199
Epoch 10/10
967/967 [==============================] - 3s 3ms/step - loss: 0.0016 - val_loss: 26530.6602
242/242 [==============================] - 1s 2ms/step
```



```
[ ] plt.figure(figsize=(10,6))
    plt.plot(predictions_inv, label='Predicted Score')
    plt.plot(y_test_inv, label='Actual Score')
    plt.legend()
    plt.show()
```




```
[ ] from sklearn.metrics import mean_squared_error, mean_absolute_error

    # Calculate metrics
    mse = mean_squared_error(y_test_inv, predictions_inv)
    mae = mean_absolute_error(y_test_inv, predictions_inv)

    print('MSE: ', mse)
    print('MAE: ', mae)

    # Additional metrics
    from sklearn.metrics import r2_score
    r2 = r2_score(y_test_inv, predictions_inv)
    print('R-squared: ', r2)
```

```
MSE:  63.77712735042881
```

```
[ ] predictions_inv
```

```
array([[131.0782 ],
       [169.57573],
       [166.1321 ],
       ...,
       [151.2154 ],
       [130.70163],
       [146.46645]], dtype=float32)
```

```
[ ] y_test_inv
```

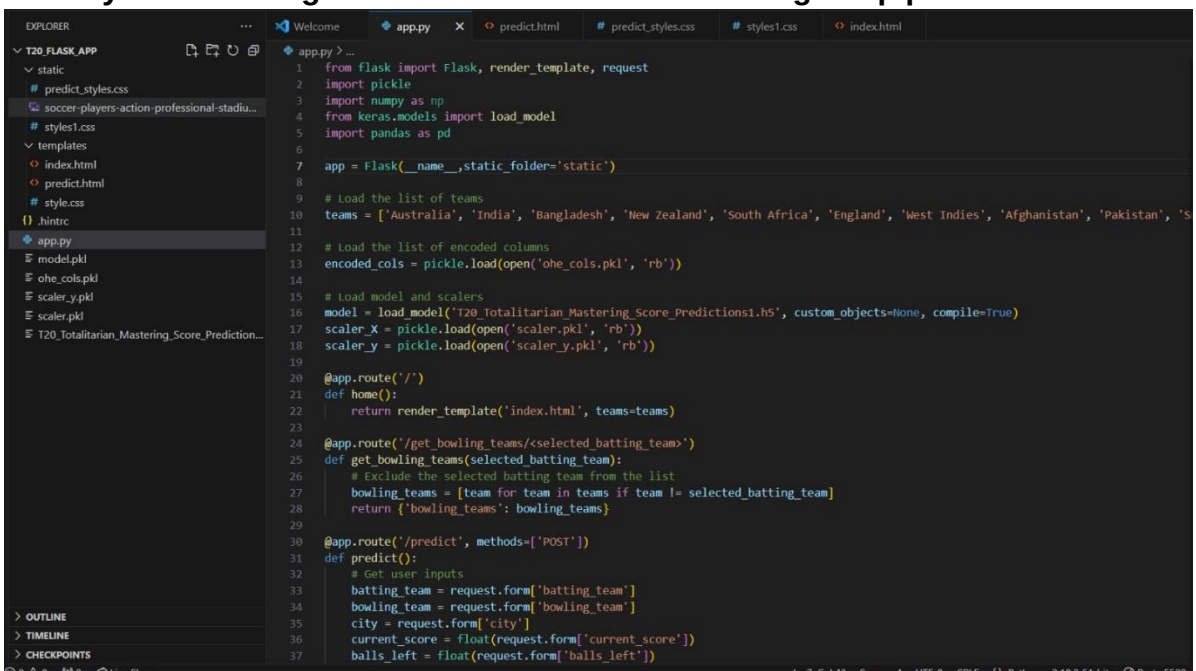
```
array([[131.],
       [169.],
       [165.],
       ...,
       [151.],
       [130.],
       [146.]])
```

Milestone 5: Model Deployment

- **Activity 1: Save the best model**
- **Activity 2: Integrate with Web Framework**

Activity 2.1: Create a web.py file and import necessary packages:

Activity 2.2: Defining teams/cities names and loading the pipeline:



```
EXPLORER
T20_FLASK_APP
  static
    predict_styles.css
    soccer-players-action-professional-stadiu...
    styles1.css
  templates
    index.html
    predict.html
    style.css
  .hintrc
  app.py
  model.pkl
  ohe_cols.pkl
  scaler_y.pkl
  scaler.pkl
  T20_Totalitarian_Mastering_Score_Prediction...
OUTLINE
TIMELINE
CHECKPOINTS
```

```
Welcome | app.py | predict.html | predict_styles.css | styles1.css | index.html
app.py > ...
from flask import Flask, render_template, request
import pickle
import numpy as np
from keras.models import load_model
import pandas as pd

app = Flask(__name__,static_folder='static')

# Load the list of teams
teams = ['Australia', 'India', 'Bangladesh', 'New Zealand', 'South Africa', 'England', 'West Indies', 'Afghanistan', 'Pakistan', 'S

# Load the list of encoded columns
encoded_cols = pickle.load(open('ohe_cols.pkl', 'rb'))

# Load model and scalers
model = load_model('T20_Totalitarian_Mastering_Score_Predictions1.h5', custom_objects=None, compile=True)
scaler_X = pickle.load(open('scaler.pkl', 'rb'))
scaler_y = pickle.load(open('scaler_y.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html', teams=teams)

@app.route('/get_bowling_teams/<selected_batting_team>')
def get_bowling_teams(selected_batting_team):
    # Exclude the selected batting team from the list
    bowling_teams = [team for team in teams if team != selected_batting_team]
    return {'bowling_teams': bowling_teams}

@app.route('/predict', methods=['POST'])
def predict():
    # Get user inputs
    batting_team = request.form['batting_team']
    bowling_team = request.form['bowling_team']
    city = request.form['city']
    current_score = float(request.form['current_score'])
    balls_left = float(request.form['balls_left'])
```

```
⊗0 △0  0  0  Live Share   Ln 7, Col 43  Spaces: 4  UTF-8  CRLF  Python  3.10.2 64-bit  Port : 5500
```

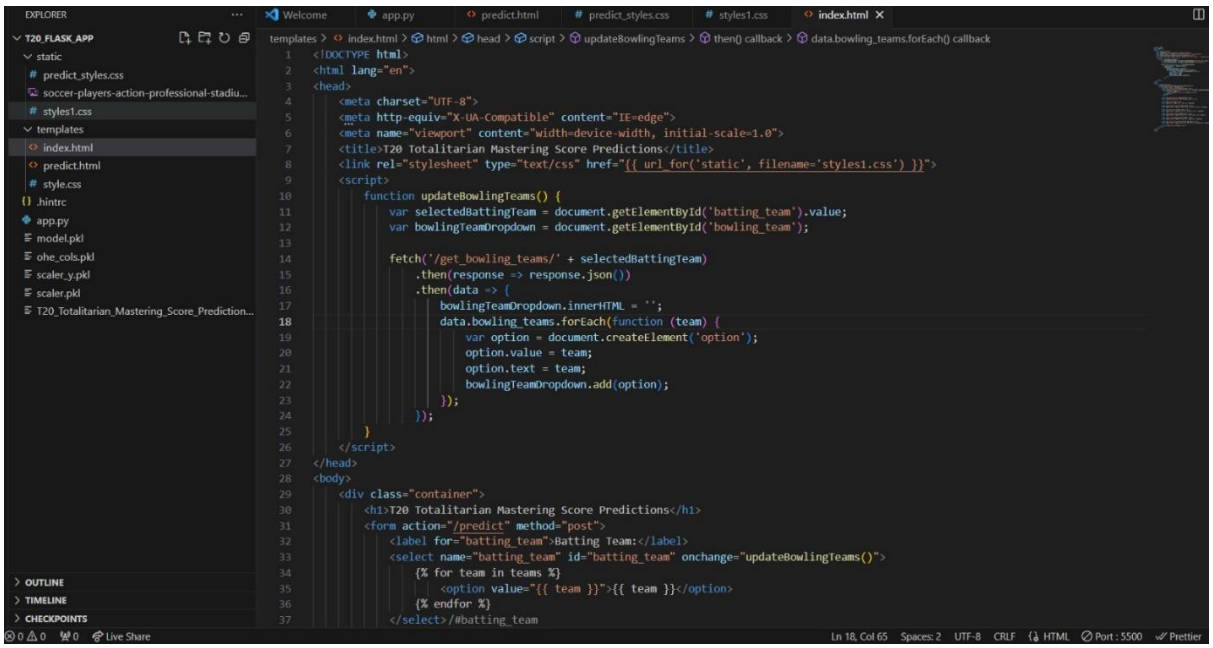
```
    wickets_left = float(request.form['wickets_left'])
    crr = float(request.form['crr'])
    last_five = float(request.form['last_five'])

    # Create DataFrame with the input data
    input_data = [[batting_team, bowling_team, city, current_score, balls_left, wickets_left, crr, last_five]]
    input_df = pd.DataFrame(input_data, columns=['batting_team', 'bowling_team', 'city', 'current_score', 'balls_left', 'wickets_le

    # Apply one-hot encoding based on the loaded columns
    input_df_encoded = pd.get_dummies(input_df, columns=['batting_team', 'bowling_team', 'city'])

    # Ensure the columns match the ones used during training
    input_df_encoded = input_df_encoded.reindex(columns=encoded_cols, fill_value=0)

    # Scale features
    X_scaled = scaler_X.transform(input_df_encoded)

    # Reshape for the model
    X_reshaped = X_scaled.reshape(1, 1, X_scaled.shape[1])

    # Make prediction
    y_pred = model.predict(X_reshaped)

    # Inverse transform prediction
    y_pred = scaler_y.inverse_transform(y_pred)

    return render_template('predict.html', prediction=y_pred[0][0])

if __name__ == '__main__':
    app.run(debug=True)
```

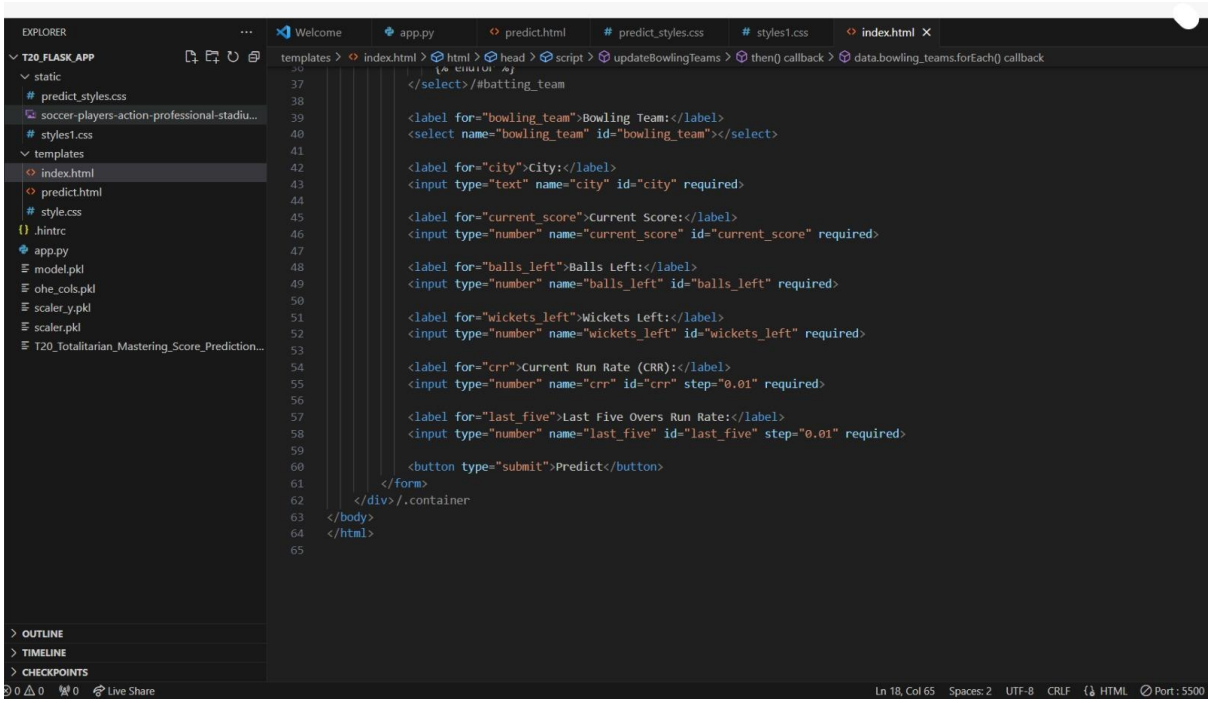
teams = ['Australia',
'India',
'Bangladesh',
'New Zealand',
'South Africa', 'England',
'West Indies', 'Afghanistan',
'Pakistan',
'Sri Lanka']

```
cities = ['Colombo ', 'Mirpur',
'Johannesburg',
'Dubai',
'Auckland', 'Cape Town',
'London', 'Pallekele',
'Barbados',
'Sydney',
'Melbourne',
'Durban',
'St Lucia',
'Wellington', 'Lauderhill',
'Hamilton', 'Centurion', 'Manchester', 'Abu Dhabi',
'Mumbai', 'Nottingham', 'Southampton',
'Mount Maunganui',
'Chittagong', 'Kolkata',
'Lahore',
'Delhi',
'Nagpur',
'Chandigarh',
'Adelaide',
'Bangalore',
'St Kitts',
'Cardiff',
'Christchurch', 'Trinidad']
```

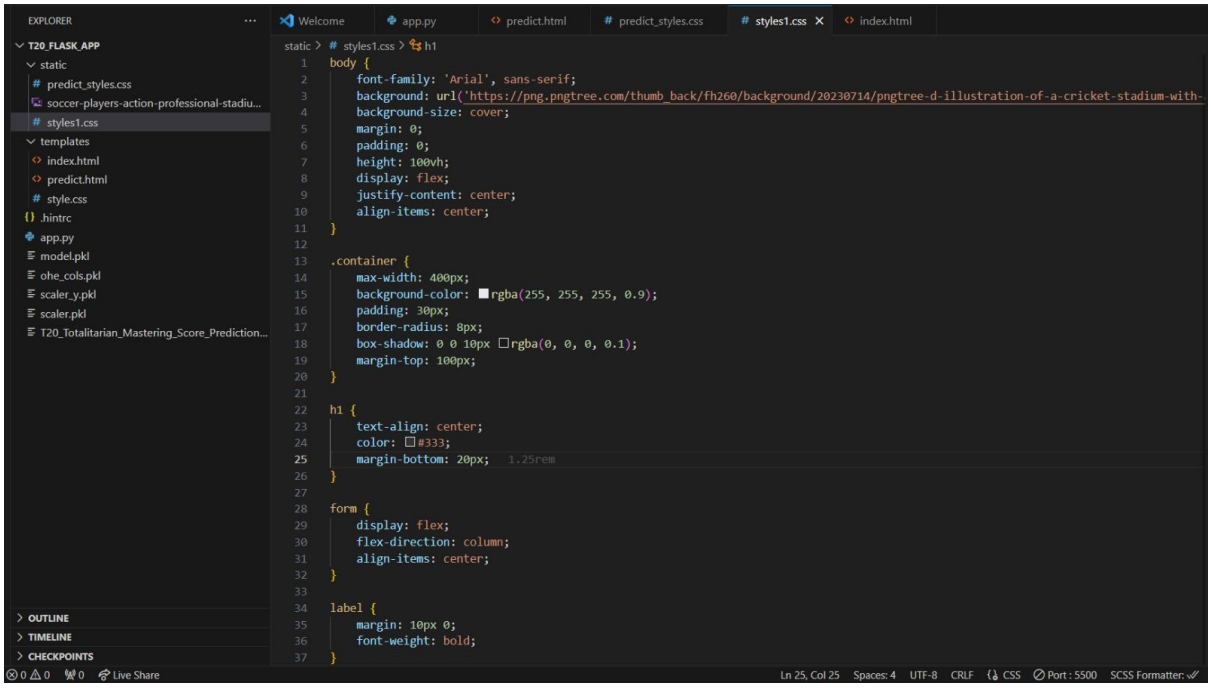
Activity 2.3: Accepting the input from user and prediction

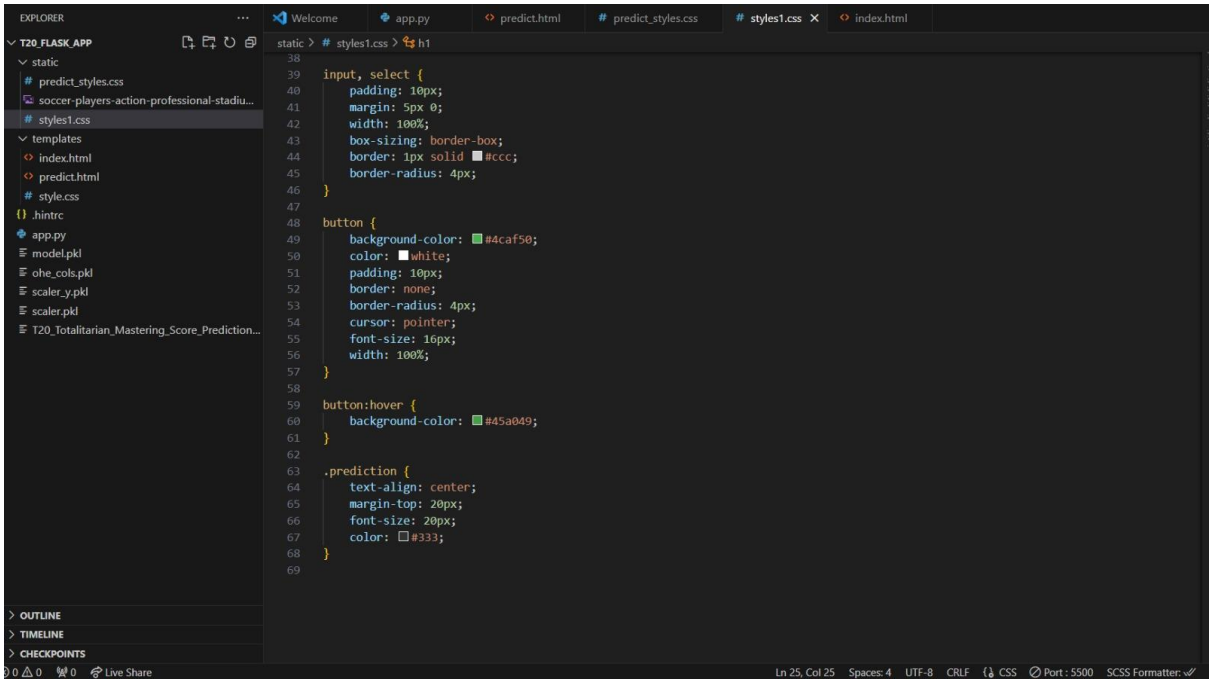
Index.html





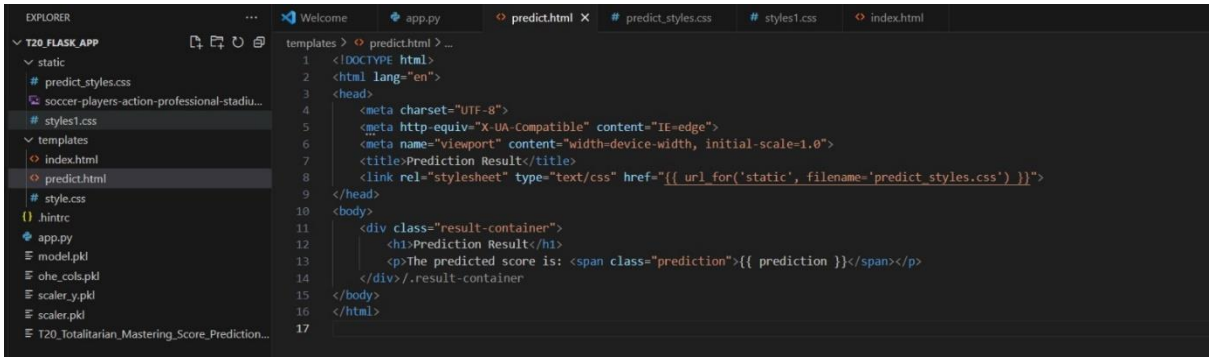
The css files for the index.html file4



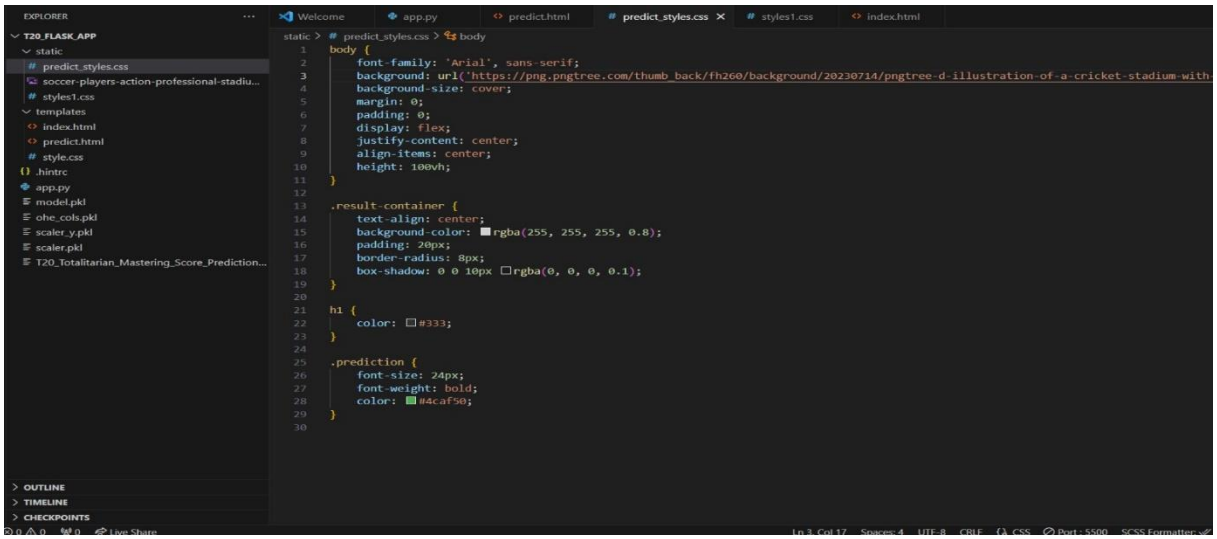


```
input, select {
    padding: 10px;
    margin: 5px 0;
    width: 100%;
    box-sizing: border-box;
    border: 1px solid #ccc;
    border-radius: 4px;
}

button {
    background-color: #4caf50;
    color: white;
    padding: 10px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
    width: 100%;
}

button:hover {
    background-color: #45a049;
}

.prediction {
    text-align: center;
    margin-top: 20px;
    font-size: 20px;
    color: #333;
}
```

The html and css file for the predict page



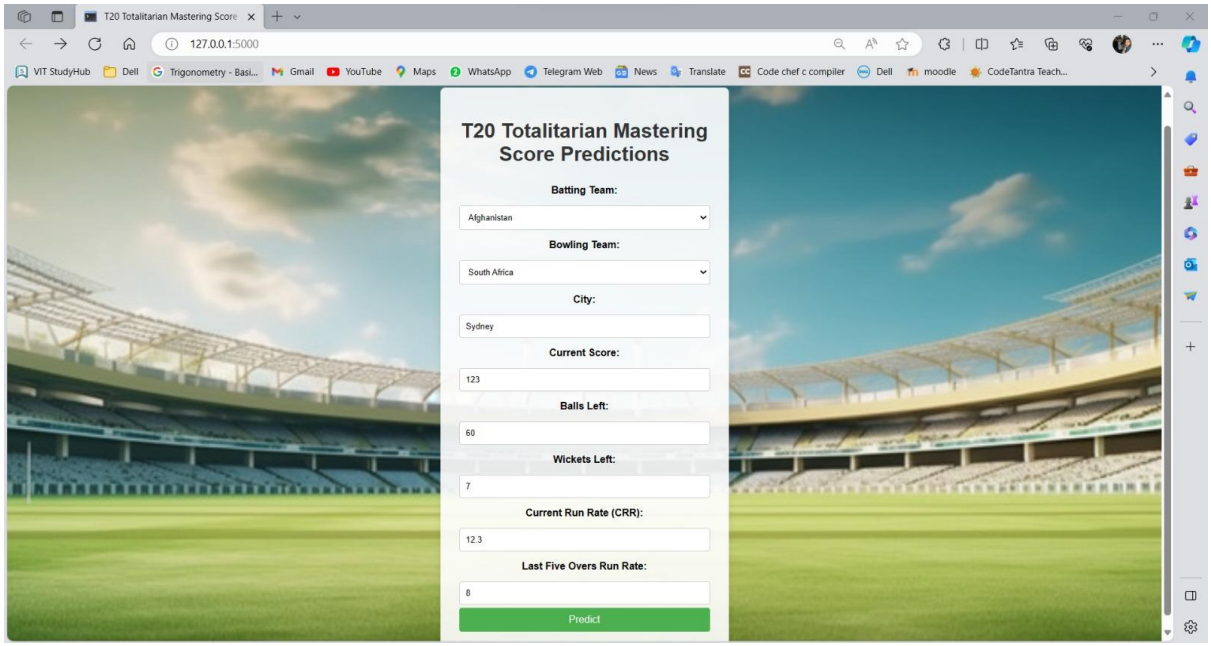
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prediction Result</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='predict_styles.css') }}">
</head>
<body>
    <div class="result-container">
        <h1>Prediction Result</h1>
        <p>The predicted score is: <span class="prediction">{{ prediction }}</span></p>
    </div>/.result-container
</body>
</html>
```



```
body {
    font-family: 'Arial', sans-serif;
    background: url('https://png.pngtree.com/thumb_back/fh260/background/20230714/pngtree-d-illustration-of-a-cricket-stadium-with
    background-size: cover;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.result-container {
    text-align: center;
    background-color: rgba(255, 255, 255, 0.8);
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
    color: #333;
}

.prediction {
    font-size: 24px;
    font-weight: bold;
    color: #4caf50;
}
```

How Does Website Look Like ?

Before predicting the data



After predicting the data