HTML:

Base:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% block title %}IPL Predictor{% endblock %}</title>

<!-- Tailwind CSS -->

<script src="https://cdn.tailwindcss.com"></script>

<!-- Google Fonts -->

<link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&display=swap" rel="stylesheet">

<style>

body {

font-family: 'Poppins', sans-serif;

}

</style>

<!-- AOS Animations -->

<link href="https://unpkg.com/aos@2.3.1/dist/aos.css" rel="stylesheet">

<!-- Select2 Dropdown -->

<link href="https://cdn.jsdelivr.net/npm/select2@4.1.0-rc.0/dist/css/select2.min.css" rel="stylesheet" />

<!-- Chart.js -->

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>

<body class="bg-gradient-to-r from-blue-500 to-teal-400 min-h-screen flex flex-col" id="particles-js">

<!-- Navbar -->

<nav class="bg-white text-blue-600 p-4 shadow-lg">

<div class="container mx-auto flex justify-between items-center">

<h1 class="text-xl font-bold">🏏 IPL Predictor</h1>

<div class="space-x-4">

<a href="/" class="hover:underline">Home</a>

<a href="/prematch" class="hover:underline">Pre-Match</a>

<a href="/midmatch" class="hover:underline">Mid-Match</a>

</div>

</div>

</nav>

<!-- Main content -->

<main class="container mx-auto py-8 flex-grow">

{% block content %}{% endblock %}

</main>

<!-- Footer -->

<footer class="bg-white text-blue-600 p-4 text-center">

&copy; 2025 IPL Predictor

</footer>

<!-- JS Libraries -->

<script src="https://unpkg.com/aos@2.3.1/dist/aos.js"></script>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/select2@4.1.0-rc.0/dist/js/select2.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/particles.js"></script>

<!-- Init Scripts -->

<script>

// AOS animation init

AOS.init();

// Select2 with logos

$(document).ready(function () {

$('select').select2({

templateResult: formatTeam,

templateSelection: formatTeam

});

});

function formatTeam(team) {

if (!team.id) return team.text;

let imgPath = `/static/logos/${team.text.toLowerCase().replace(/ /g, '')}.png`;

return $(`<span><img src="${imgPath}" style="width:20px;height:20px;margin-right:8px;">${team.text}</span>`);

}

// Particles.js background

particlesJS("particles-js", {

particles: {

number: { value: 50 },

size: { value: 3 },

move: { speed: 1 }

}

});

</script>

</body>

</html>

Home:

{% extends "base.html" %}

{% block title %}Home - IPL Predictor{% endblock %}

{% block content %}

<!-- Fullscreen Background -->

<div

class="min-h-screen flex items-center justify-center bg-gradient-to-br from-blue-800 via-cyan-700 to-teal-800 px-4 relative overflow-hidden">

<!-- Glassmorphism Card -->

<div class="text-center space-y-8 py-12 px-8 rounded-2xl shadow-2xl backdrop-blur-md bg-white/10 border border-white/20 max-w-3xl w-full relative z-10"

data-aos="fade-up" data-aos-duration="1000">

<!-- Main Heading with Shimmer -->

<h2 class="text-5xl font-extrabold text-white drop-shadow-lg shimmer">

Welcome to the IPL Match Predictor

</h2>

<!-- Tagline -->

<p class="text-lg text-gray-200 max-w-2xl mx-auto leading-relaxed">

Harness the power of AI and data analytics to predict match outcomes with precision.

Make informed decisions <span class="font-semibold text-yellow-300">before</span> or

<span class="font-semibold text-yellow-300">during</span> the game.

</p>

<!-- Action Buttons -->

<div class="flex flex-wrap justify-center gap-6 mt-8">

<a href="/prematch"

class="bg-gradient-to-r from-blue-500 to-blue-700 text-white px-8 py-4 rounded-lg shadow-lg hover:scale-110 hover:shadow-blue-400/50 hover:from-blue-600 hover:to-blue-800 transition-transform duration-300 text-lg font-semibold">

Pre-match Prediction

</a>

<a href="/midmatch"

class="bg-gradient-to-r from-green-500 to-green-700 text-white px-8 py-4 rounded-lg shadow-lg hover:scale-110 hover:shadow-green-400/50 hover:from-green-600 hover:to-green-800 transition-transform duration-300 text-lg font-semibold">

Mid-match Prediction

</a>

<a href="/analysis"

class="bg-gradient-to-r from-purple-500 to-purple-700 text-white px-8 py-4 rounded-lg shadow-lg hover:scale-110 hover:shadow-purple-400/50 hover:from-purple-600 hover:to-purple-800 transition-transform duration-300 text-lg font-semibold">

Analysis

</a>

</div>

<!-- Small Tip Section -->

<div class="mt-6 text-sm text-gray-300 italic">

Tip: Stay updated with live match data for the most accurate predictions.

</div>

</div>

</div>

<!-- Shimmer Effect Styles -->

<style>

.shimmer {

background: linear-gradient(90deg, #fff, #ffeb3b, #fff);

background-size: 200% 100%;

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

animation: shimmer-move 3s infinite linear;

}

@keyframes shimmer-move {

0% {

background-position: -200% 0;

}

100% {

background-position: 200% 0;

}

}

</style>

{% endblock %}

Prematch:

<!DOCTYPE html>

<html>

<head>

<title>Pre-Match</title>

<style>

body {

font-family: Arial, sans-serif;

background: linear-gradient(rgba(0, 0, 0, 0.6), rgba(0, 0, 0, 0.6)),

url("{{ url\_for('static', filename='images/prematch-bg.png') }}") no-repeat center center fixed;

background-size: cover;

animation: zoomBg 25s ease-in-out infinite alternate;

color: white;

text-align: center;

padding-top: 30px;

}

@keyframes zoomBg {

0% {

background-size: 100% auto;

}

100% {

background-size: 110% auto;

}

}

h2 {

font-size: 32px;

margin-bottom: 20px;

color: #ffffff;

}

form {

background: rgba(255, 255, 255, 0.15);

padding: 20px;

border-radius: 10px;

display: inline-block;

backdrop-filter: blur(5px);

}

select,

input[type=submit] {

padding: 10px;

margin: 10px;

border: none;

border-radius: 5px;

font-size: 16px;

}

input[type=submit] {

background-color: #0ea5e9;

color: white;

font-weight: bold;

cursor: pointer;

transition: 0.3s;

}

input[type=submit]:hover {

background-color: #0369a1;

}

</style>

</head>

<body>

<h2>Pre-Match Prediction</h2>

<form method="post" action="/predict\_prematch">

<select name="batting\_team" required>

<option value="">--Select Batting Team--</option>

{% for t in teams %}<option value="{{ t }}">{{ t }}</option>{% endfor %}

</select><br>

<select name="bowling\_team" required>

<option value="">--Select Bowling Team--</option>

{% for t in teams %}<option value="{{ t }}">{{ t }}</option>{% endfor %}

</select><br>

<select name="toss\_winner" required>

<option value="">--Select Toss Winner--</option>

{% for t in teams %}<option value="{{ t }}">{{ t }}</option>{% endfor %}

</select><br>

<select name="toss\_decision" required>

<option value="">--Select Toss Decision--</option>

<option value="bat">bat</option>

<option value="field">field</option>

</select><br>

<select name="venue" required>

<option value="">--Select Venue--</option>

{% for v in venues %}<option value="{{ v }}">{{ v }}</option>{% endfor %}

</select><br>

<select name="city" required>

<option value="">--Select City--</option>

{% for c in cities %}<option value="{{ c }}">{{ c }}</option>{% endfor %}

</select><br>

<select name="season" required>

<option value="">--Select Season--</option>

{% for s in seasons %}<option value="{{ s }}">{{ s }}</option>{% endfor %}

</select><br>

<select name="match\_type" required>

<option value="">--Select Match Type--</option>

{% for mt in match\_types %}<option value="{{ mt }}">{{ mt }}</option>{% endfor %}

</select><br>

<input type="submit" value="Predict">

</form>

</body>

</html>

Midmatch:

<!DOCTYPE html>

<html>

<head>

<title>Mid-Match</title>

<style>

body {

font-family: Arial, sans-serif;

background: linear-gradient(rgba(0, 0, 0, 0.6), rgba(0, 0, 0, 0.6)),

url("{{ url\_for('static', filename='images/midmatch-bg.png') }}") no-repeat center center fixed;

background-size: cover;

animation: zoomBg 25s ease-in-out infinite alternate;

color: white;

text-align: center;

padding-top: 30px;

}

@keyframes zoomBg {

0% {

background-size: 100% auto;

}

100% {

background-size: 110% auto;

}

}

h2 {

font-size: 32px;

margin-bottom: 20px;

color: #ffffff;

}

form {

background: rgba(255, 255, 255, 0.15);

padding: 20px;

border-radius: 10px;

display: inline-block;

backdrop-filter: blur(5px);

}

select,

input {

padding: 10px;

margin: 10px;

border: none;

border-radius: 5px;

font-size: 16px;

}

input[type=submit] {

background-color: #0ea5e9;

color: white;

font-weight: bold;

cursor: pointer;

transition: 0.3s;

}

input[type=submit]:hover {

background-color: #0369a1;

}

</style>

</head>

<body>

<h2>Mid-Match Prediction</h2>

<form method="post" action="/predict\_midmatch">

<select name="batting\_team" required>

<option value="">--Select Batting Team--</option>

{% for t in teams %}<option value="{{ t }}">{{ t }}</option>{% endfor %}

</select><br>

<select name="bowling\_team" required>

<option value="">--Select Bowling Team--</option>

{% for t in teams %}<option value="{{ t }}">{{ t }}</option>{% endfor %}

</select><br>

<select name="venue" required>

<option value="">--Select Venue--</option>

{% for v in venues %}<option value="{{ v }}">{{ v }}</option>{% endfor %}

</select><br>

<select name="city" required>

<option value="">--Select City--</option>

{% for c in cities %}<option value="{{ c }}">{{ c }}</option>{% endfor %}

</select><br>

Team runs: <input type="number" step="1" name="runs\_total" required><br>

Team wickets: <input type="number" name="team\_wicket" required><br>

Overs (decimal) or Team balls: <input type="number" step="0.1" name="overs"> OR <input type="number"

name="team\_balls"><br>

<input type="submit" value="Predict">

</form>

</body>

</html>

Analysis:

{% extends "base.html" %}

{% block content %}

<div class="container mt-4">

<h2 class="mb-4">IPL Data Analysis</h2>

<div class="row">

{% for plot in [

'top\_wicket\_takers.png', 'top\_run\_scorer.png',

'top\_batter\_strike\_rate.png', 'top\_economical\_bowler.png',

'consistency.png', 'toss\_impact\_venue.png', 'toss\_decision\_impact.png',

'top\_venues.png', 'top\_players.png', 'top\_team\_performance\_venue.png',

'matches\_per\_season.png', 'team\_game\_phase.png',

'top\_venue\_by\_chasing\_success.png'

] %}

<div class="col-md-6 mb-4">

<div class="card shadow-sm">

<div class="card-body text-center">

<h5>{{ plot.split('.')[0].replace('\_', ' ').title() }}</h5>

<img src="{{ url\_for('static', filename='plots/' + plot) }}" class="img-fluid" alt="{{ plot }}">

</div>

</div>

</div>

{% endfor %}

</div>

</div>

{% endblock %}

Result:

{% extends "base.html" %}

{% block title %}Prediction Result{% endblock %}

{% block content %}

<style>

body {

background: linear-gradient(rgba(0, 0, 0, 0.6), rgba(0, 0, 0, 0.6)),

url("{{ url\_for('static', filename='images/result-bg.png') }}") no-repeat center center fixed !important;

background-size: cover !important;

animation: zoomBg 25s ease-in-out infinite alternate;

}

@keyframes zoomBg {

0% {

background-size: 100% auto;

}

100% {

background-size: 110% auto;

}

}

</style>

<div class="max-w-md mx-auto text-center bg-white bg-opacity-80 rounded-lg p-6 shadow-lg backdrop-blur-sm"

data-aos="zoom-in">

<h2 class="text-3xl font-bold text-blue-600 mb-4">Prediction Result</h2>

<p class="text-2xl font-semibold text-green-600">{{ prediction }}</p>

<!-- Chart -->

<canvas id="probabilityChart" class="mt-4"></canvas>

<a href="/" class="mt-4 inline-block bg-blue-600 text-white px-6 py-2 rounded hover:bg-blue-800 transition">

Back to Home

</a>

</div>

<script>

var ctx = document.getElementById('probabilityChart').getContext('2d');

// Load team logo dynamically based on prediction

var logoImg = new Image();

logoImg.src = "/static/logos/{{ prediction | lower | replace(' ', '') }}.png";

// If image fails to load, use a default logo

logoImg.onerror = function () {

logoImg.src = "/static/logos/default.png";

};

// Plugin to draw logo inside chart

const logoPlugin = {

id: 'centerLogo',

afterDraw: (chart) => {

const { ctx, chartArea: { width, height, left, top } } = chart;

const xCenter = left + width / 2;

const yCenter = top + height / 2;

const imgSize = Math.min(width, height) / 2.5; // Bigger size for better fit

ctx.save();

ctx.globalAlpha = 1; // full opacity

ctx.drawImage(logoImg, xCenter - imgSize / 2, yCenter - imgSize / 2, imgSize, imgSize);

ctx.restore();

}

};

new Chart(ctx, {

type: 'doughnut',

data: {

labels: ['Win Probability', 'Lose Probability'],

datasets: [{

data: [75, 25], // Replace with real prediction values

backgroundColor: ['#22c55e', '#ef4444']

}]

},

options: {

responsive: true,

plugins: { legend: { position: 'bottom' } }

},

plugins: [logoPlugin]

});

</script>

{% endblock %}

[app.py](http://app.py):

from flask import Flask, render\_template, request

import pickle

import numpy as np

import os

app = Flask(\_\_name\_\_)

BASE = os.path.dirname(\_\_file\_\_)

def load\_package(path):

if not os.path.exists(path):

raise FileNotFoundError(f"Missing file: {path}. Run training first.")

with open(path, "rb") as f:

obj = pickle.load(f)

# Expect dictionary with keys: model, encoders, feature\_cols, target\_col

if not isinstance(obj, dict) or not all(

k in obj for k in ("model", "encoders", "feature\_cols", "target\_col")

):

raise ValueError(

f"{path} missing required keys (model, encoders, feature\_cols, target\_col)"

)

return obj["model"], obj["encoders"], obj["feature\_cols"], obj["target\_col"]

# Load both packages

prematch\_model, prematch\_encoders, PREMATCH\_FEATURES, PREMATCH\_TARGET = load\_package(

os.path.join(BASE, "prematch\_model.pkl")

)

midmatch\_model, midmatch\_encoders, MIDMATCH\_FEATURES, MIDMATCH\_TARGET = load\_package(

os.path.join(BASE, "midmatch\_model.pkl")

)

def transform\_with\_encoder(encoders, col, value):

if value is None:

return None

if col in encoders:

try:

return int(encoders[col].transform([str(value)])[0])

except Exception:

return None

# otherwise try numeric

try:

if "." in str(value):

return float(value)

return int(value)

except:

return None

def inverse\_label(encoders, target\_col, code):

if target\_col in encoders:

return encoders[target\_col].inverse\_transform([int(code)])[0]

return str(code)

def get\_dropdowns():

# Pull dropdown lists from encoders to ensure exact same strings

teams = (

list(prematch\_encoders["batting\_team"].classes\_)

if "batting\_team" in prematch\_encoders

else []

)

venues = (

list(prematch\_encoders["venue"].classes\_)

if "venue" in prematch\_encoders

else []

)

cities = (

list(prematch\_encoders["city"].classes\_) if "city" in prematch\_encoders else []

)

seasons = (

list(prematch\_encoders["season"].classes\_)

if "season" in prematch\_encoders

else []

)

match\_types = (

list(prematch\_encoders["match\_type"].classes\_)

if "match\_type" in prematch\_encoders

else []

)

toss\_decisions = ["bat", "field"]

return teams, venues, cities, seasons, match\_types, toss\_decisions

@app.route("/")

def home():

return render\_template("home.html")

@app.route("/prematch")

def prematch():

teams, venues, cities, seasons, match\_types, toss\_decisions = get\_dropdowns()

return render\_template(

"prematch.html",

teams=teams,

venues=venues,

cities=cities,

seasons=seasons,

match\_types=match\_types,

toss\_decisions=toss\_decisions,

)

@app.route("/midmatch")

def midmatch():

teams, venues, cities, \_, \_, \_ = get\_dropdowns()

return render\_template("midmatch.html", teams=teams, venues=venues, cities=cities)

@app.route("/predict\_prematch", methods=["POST"])

def predict\_prematch():

raw = {col: request.form.get(col) for col in PREMATCH\_FEATURES}

transformed = []

for col in PREMATCH\_FEATURES:

code = transform\_with\_encoder(prematch\_encoders, col, raw.get(col))

if code is None:

return render\_template(

"result.html",

prediction=f"Invalid or unseen value for '{col}': '{raw.get(col)}'. Please choose from dropdown.",

)

transformed.append(code)

X = np.array([transformed], dtype=float)

pred\_code = prematch\_model.predict(X)[0]

winner = inverse\_label(prematch\_encoders, PREMATCH\_TARGET, pred\_code)

return render\_template("result.html", prediction=winner)

@app.route("/predict\_midmatch", methods=["POST"])

def predict\_midmatch():

# categories

batting\_team = request.form.get("batting\_team")

bowling\_team = request.form.get("bowling\_team")

venue = request.form.get("venue")

city = request.form.get("city")

# numerics (runs/wickets/overs or team\_balls)

runs = request.form.get("runs\_total") or request.form.get("team\_runs")

wickets = request.form.get("team\_wicket")

overs\_input = request.form.get("overs")

team\_balls = request.form.get("team\_balls")

# determine overs

if overs\_input and overs\_input.strip() != "":

try:

overs = float(overs\_input)

except:

return render\_template("result.html", prediction="Invalid overs value")

elif team\_balls and team\_balls.strip() != "":

try:

overs = int(team\_balls) / 6.0

except:

return render\_template("result.html", prediction="Invalid team\_balls value")

else:

return render\_template("result.html", prediction="Provide overs or team\_balls")

# encode categories

encoded = []

for col, val in [

("batting\_team", batting\_team),

("bowling\_team", bowling\_team),

("venue", venue),

("city", city),

]:

code = transform\_with\_encoder(midmatch\_encoders, col, val)

if code is None:

return render\_template(

"result.html",

prediction=f"Invalid or unseen value for '{col}': '{val}'. Please choose from dropdown.",

)

encoded.append(code)

# numeric casting

try:

team\_runs = float(runs)

team\_wicket = int(wickets)

except:

return render\_template(

"result.html", prediction="Invalid numeric input for runs or wickets"

)

X = np.array([encoded + [team\_runs, team\_wicket, overs]], dtype=float)

pred\_code = midmatch\_model.predict(X)[0]

winner = inverse\_label(midmatch\_encoders, MIDMATCH\_TARGET, pred\_code)

return render\_template("result.html", prediction=winner)

# ✅ New Analysis Route

@app.route("/analysis")

def analysis():

return render\_template("analysis.html")

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

Train prematch:

# train\_prematch\_model.py

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

import pickle

# Load dataset (make sure IPL.csv is in same folder)

df = pd.read\_csv("IPL.csv", low\_memory=False)

# Features & target (must match the app)

feature\_cols = [

"batting\_team",

"bowling\_team",

"toss\_winner",

"toss\_decision",

"venue",

"city",

"season",

"match\_type",

]

target\_col = "match\_won\_by"

# Keep only rows that have all required fields

df = df[feature\_cols + [target\_col]].dropna()

# Strip strings to avoid whitespace mismatches

for c in df.select\_dtypes(include="object").columns:

df[c] = df[c].astype(str).str.strip()

# Fit label encoders for categorical columns (including target)

encoders = {}

for col in feature\_cols + [target\_col]:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col].astype(str))

encoders[col] = le

# Prepare X and y

X = df[feature\_cols]

y = df[target\_col]

# Train-test split (optional)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

# Train model (RandomForest for stability)

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Save model + encoders + metadata

with open("prematch\_model.pkl", "wb") as f:

pickle.dump(

{

"model": model,

"encoders": encoders,

"feature\_cols": feature\_cols,

"target\_col": target\_col,

},

f,

)

print("Saved prematch\_model.pkl (model + encoders).")

Train midmatch:

# train\_midmatch\_model.py

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

import pickle

# Load dataset

df = pd.read\_csv("IPL.csv", low\_memory=False)

# Mid-match features and target

feature\_cols = [

"batting\_team",

"bowling\_team",

"venue",

"city",

"team\_runs",

"team\_wicket",

"overs",

]

target\_col = "match\_won\_by"

# Keep only rows with required data

df = df[feature\_cols + [target\_col]].dropna()

# Clean string columns

for c in df.select\_dtypes(include="object").columns:

df[c] = df[c].astype(str).str.strip()

# Ensure numeric columns are numeric

df["team\_runs"] = pd.to\_numeric(df["team\_runs"], errors="coerce")

df["team\_wicket"] = pd.to\_numeric(df["team\_wicket"], errors="coerce")

df["overs"] = pd.to\_numeric(df["overs"], errors="coerce")

df = df.dropna(subset=["team\_runs", "team\_wicket", "overs", target\_col])

# Fit LabelEncoders for categorical columns (and target)

encoders = {}

categorical\_cols = ["batting\_team", "bowling\_team", "venue", "city", target\_col]

for col in categorical\_cols:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col].astype(str))

encoders[col] = le

# Prepare X and y

X = df[feature\_cols]

y = df[target\_col]

# Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

# Train model

model = RandomForestClassifier(n\_estimators=200, random\_state=42)

model.fit(X\_train, y\_train)

# Save model + encoders + metadata

with open("midmatch\_model.pkl", "wb") as f:

pickle.dump(

{

"model": model,

"encoders": encoders,

"feature\_cols": feature\_cols,

"target\_col": target\_col,

},

f,

)

print("Saved midmatch\_model.pkl (model + encoders).")

encoders prematch and midmatch:  
from google.colab import files

files.download('prematch\_model.pkl')

files.download('encoders.pkl')

from google.colab import files

files.download('midmatch\_model.pkl')

files.download('midmatch\_encoders.pkl')