```python
import numpy as np
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
```

```python
# Data Loading
matches_raw = pd.read_csv('matches.csv')
deliveries_raw = pd.read_csv('deliveries.csv')

# Define acive teams as older teams are merged or not currently playing
active_teams = [
    'Sunrisers Hyderabad', 'Mumbai Indians', 'Royal Challengers Bangalore',
    'Kolkata Knight Riders', 'Kings XI Punjab', 'Chennai Super Kings',
    'Rajasthan Royals', 'Delhi Capitals'
]
```

```python
# first innings
inning_one_scores = deliveries_raw.groupby(['match_id', 'inning']).sum()['total_runs'].reset_index()
inning_one_scores = inning_one_scores[inning_one_scores['inning'] == 1]

# Merge target scores with match details
match_summary = matches_raw.merge(inning_one_scores[['match_id', 'total_runs']], left_on='id', right_on='match_id')

# some team's name changed overtime
team_map = {
    'Delhi Daredevils': 'Delhi Capitals',
    'Deccan Chargers': 'Sunrisers Hyderabad'
}
match_summary['team1'] = match_summary['team1'].replace(team_map)
match_summary['team2'] = match_summary['team2'].replace(team_map)

# filter for active teams
match_summary = match_summary[match_summary['team1'].isin(active_teams) & match_summary['team2'].isin(active_teams)]
match_summary = match_summary[match_summary['dl_applied'] == 0]


match_summary = match_summary[['match_id', 'city', 'winner', 'total_runs']]

# 2nd innings
chase_data = match_summary.merge(deliveries_raw, on='match_id')
chase_data = chase_data[chase_data['inning'] == 2]
```

```python
# Calculate basic metrics
chase_data['score_accumulated'] = chase_data.groupby('match_id')['total_runs_y'].cumsum()
chase_data['runs_needed'] = chase_data['total_runs_x'] - chase_data['score_accumulated']
chase_data['balls_remaining'] = 126 - (chase_data['over'] * 6 + chase_data['ball'])

# Calculate Wickets Remaining
chase_data['is_wicket'] = chase_data['player_dismissed'].apply(lambda x: 0 if pd.isna(x) else 1)
wickets_fallen = chase_data.groupby('match_id')['is_wicket'].cumsum().values
chase_data['wickets_in_hand'] = 10 - wickets_fallen

# Calculate Run Rates
# CRR = Current Run Rate, RRR = Required Run Rate
chase_data['current_run_rate'] = (chase_data['score_accumulated'] * 6) / (120 - chase_data['balls_remaining'])
chase_data['required_run_rate'] = (chase_data['runs_needed'] * 6) / chase_data['balls_remaining']
```

```python
# Define Target Variable (Win/Loss)
def determine_outcome(row):
    return 1 if row['batting_team'] == row['winner'] else 0

chase_data['outcome'] = chase_data.apply(determine_outcome, axis=1)
```

```
# Renaming cols
dataset = chase_data[[
    'batting_team', 'bowling_team', 'city', 'runs_needed',
    'balls_remaining', 'wickets_in_hand', 'total_runs_x',
    'current_run_rate', 'required_run_rate', 'outcome'
]]


dataset.columns = [
    'batting_squad', 'bowling_squad', 'venue_city', 'runs_needed',
    'balls_remaining', 'wickets_remaining', 'target_score',
    'cur_run_rate', 'req_run_rate', 'is_win'
]
```

```
# Dropping NaNs and impossible scenarios (0 balls left)
dataset.dropna(inplace=True)
dataset = dataset[dataset['balls_remaining'] != 0]
```

```
/tmp/ipython-input-2575709879.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  dataset.dropna(inplace=True)
```

```
# training
X_features = dataset.iloc[:, :-1]
y_labels = dataset.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X_features, y_labels, test_size=0.2, random_state=1)
```

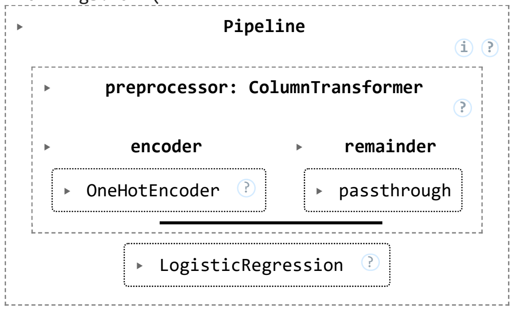
```
preprocessor = ColumnTransformer([
    ('encoder', OneHotEncoder(sparse_output=False, drop='first'),
     ['batting_squad', 'bowling_squad', 'venue_city'])
], remainder='passthrough')

win_predictor = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(solver='liblinear'))
])

win_predictor.fit(X_train, y_train)
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in ColumnTransformer.transformers_ will change in version 1.7 to ma
At the moment the remainder columns are stored as indices (of type int). With the same ColumnTransformer configuration, in t
To use the new behavior now and suppress this warning, use ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```



```
predictions = win_predictor.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Refactored Model Accuracy: {accuracy:.2%}")

# Save model
```

```
pickle.dump(win_predictor, open('win_predictor.pkl', 'wb'))
print("Model saved as 'win_predictor.pkl'")
```

```
Refactored Model Accuracy: 80.66%
Model saved as 'win_predictor.pkl'
```

```
# Load model
model = pickle.load(open('win_predictor.pkl', 'rb'))



input_data = pd.DataFrame({
    'batting_squad': ['Chennai Super Kings'],
    'bowling_squad': ['Kolkata Knight Riders'],
    'venue_city': ['Kolkata'],
    'runs_needed': [80],
    'balls_remaining': [36],
    'wickets_remaining': [7],
    'target_score': [200],
    'cur_run_rate': [8.57],
    'req_run_rate': [13.33]
})

# Predict
probability = model.predict_proba(input_data)

win_chance = probability[0][1]
lose_chance = probability[0][0]

print("\n--- Match Simulation Result ---")
print(f"Batting Team: {input_data['batting_squad'][0]}")
print(f"Bowling Team: {input_data['bowling_squad'][0]}")
print(f"Win Probability: {win_chance * 100:.2f}%")
print(f"Loss Probability: {lose_chance * 100:.2f}%")
```

```

--- Match Simulation Result ---
Batting Team: Chennai Super Kings
Bowling Team: Kolkata Knight Riders
Win Probability: 60.02%
Loss Probability: 39.98%
```

Start coding or generate with AI.