# Formulas using Linear Programming:

Batsman Weights:

| Index | Feature | Optimized Weight |
|---|---|---|
| 0 | Impact_Score | 0.25 |
| 1 | Cost_efficiency_ratio | 0.10 |
| 2 | Replacement_value | 0.10 |
| 3 | Non_Boundary_Strike_Rate | 0.10 |
| 4 | Impact_Consistency | 0.15 |
| 5 | Experience_score | 0.10 |
| 6 | Boundary_percentage | 0.20 |

Bowler Weights:

| Index | Feature | Optimized Weight |
|---|---|---|
| 0 | Impact_Score | 0.25 |
| 1 | Cost_efficiency_ratio | 0.10 |
| 2 | Experience_score | 0.00 |
| 3 | Replacement_value | 0.22 |
| 4 | Pressure_Score_BO | 0.15 |
| 5 | Volume_Score_BO | 0.15 |
| 6 | Impact_Consistency | 0.10 |
| 7 | Bowler_Type_Spin_1 | 0.03 |

All - Rounder Weights:

| Index | Feature | Optimized Weight |
|---|---|---|
| 0 | Impact_Score | 0.25 |
| 1 | Non_Boundary_Strike_Rate | 0.00 |
| 2 | Boundary_percentage | 0.20 |
| 3 | Dot_ball_percentage | 0.15 |
| 4 | Impact_Consistency | 0.10 |
| 5 | Replacement_value | 0.02 |
| 6 | Cost_efficiency_ratio | 0.15 |
| 7 | Experience_score | 0.10 |
| 8 | Bowler_Type_Spin_1 | 0.03 |

Wicket keeper weights:

| Index | Feature | Optimized Weight |
|---|---|---|
| 0 | Final_Batsman_Score | 0.25 |
| 1 | Matches_As_WK | 0.20 |
| 2 | Dismissals_Per_Match | 0.25 |
| 3 | WK_Impact_Score | 0.30 |

## Selection of Full Squad:

```
# =========================
# FULL SQUAD SELECTION (18 PLAYERS)
# =========================

from pulp import (
   LpProblem, LpVariable, LpMaximize,
   lpSum, LpBinary, LpStatus
)

# -------------------------
# ASSUMPTION: df already exists
# Columns:
# Player_Name, type,
# Final_Batsman_Score, Final_Bowler_Score,
# Final_all_score, Final_wkt_score,
# Sold_Price
# -------------------------

# -------------------------
# STEP 1: Create capability flags (CRITICAL)
# -------------------------
df["Is_Batsman"] = df["Final_Batsman_Score"] > 0
df["Is_Bowler"] = df["Final_Bowler_Score"] > 0
df["Is_AllRounder"] = df["Final_all_score"] > 0
df["Is_WicketKeeper"] = df["Final_wkt_score"] > 0

# -------------------------
# STEP 2: Create unified player score
# -------------------------
df["Final_Player_Score"] = df[
   [
       "Final_Batsman_Score",
       "Final_Bowler_Score",
       "Final_all_score",
       "Final_wkt_score"
   ]
].fillna(0).max(axis=1)
```

```
# -------------------------
# STEP 3: Define LP model
# -------------------------
model = LpProblem(
   "IPL_Full_18_Squad_Selection",
   LpMaximize
)

players = df.index

# Decision variable: select player or not
select = {
   i: LpVariable(f"select_{i}", cat=LpBinary)
   for i in players
}

# -------------------------
# STEP 4: Objective function
# -------------------------
model += lpSum(
   select[i] * df.loc[i, "Final_Player_Score"]
   for i in players
)

# -------------------------
# STEP 5: Budget constraint (₹125 cr)
# -------------------------
model += lpSum(
   select[i] * df.loc[i, "Sold_Price"]
   for i in players
) <= 125

# -------------------------
# STEP 6: Squad size = 18
# -------------------------
model += lpSum(select[i] for i in players) == 18

# -------------------------
# STEP 7: Role constraints (FEASIBLE VERSION)
# -------------------------
# Wicket Keepers: exactly 2
model += lpSum(
   select[i] * df.loc[i, "Is_WicketKeeper"]
   for i in players
) == 2

# Batting-capable players: 4-6
model += lpSum(
   select[i] * df.loc[i, "Is_Batsman"]
   for i in players
) >= 4

model += lpSum(
   select[i] * df.loc[i, "Is_Batsman"]
   for i in players
) <= 6

# Bowling-capable players: 4-6
model += lpSum(
   select[i] * df.loc[i, "Is_Bowler"]
   for i in players
) >= 4

model += lpSum(
   select[i] * df.loc[i, "Is_Bowler"]
   for i in players
) <= 6

# All-rounders: 3-4
model += lpSum(
   select[i] * df.loc[i, "Is_AllRounder"]
   for i in players
) >= 3

model += lpSum(
   select[i] * df.loc[i, "Is_AllRounder"]
   for i in players
) <= 4
```

```
# -------------------------
# STEP 8: Solve the LP
# -------------------------
model.solve()

print("Full Squad Status:", LpStatus[model.status])

# -------------------------
# STEP 9: Extract the 18-player squad
# -------------------------
full_squad = df.loc[
   [i for i in players if select[i].value() == 1]
]

# -------------------------
# STEP 10: Sanity checks
# -------------------------
print("Total Players Selected:", len(full_squad))
print("Total Budget Used:", full_squad["Sold_Price"].sum())

print("\nRole Counts:")
print("Batsmen-capable:", full_squad["Is_Batsman"].sum())
print("Bowlers-capable:", full_squad["Is_Bowler"].sum())
print("All-rounders:", full_squad["Is_AllRounder"].sum())
print("Wicket Keepers:", full_squad["Is_WicketKeeper"].sum())

# -------------------------
# STEP 11: View final 18-man squad
# -------------------------
full_squad[
   ["Player_Name", "type", "Final_Player_Score", "Sold_Price"]
].sort_values(
   "Final_Player_Score", ascending=False
)
```

This model uses binary linear programming to select an 18-member IPL squad under a fixed ₹125 crore budget. Players are evaluated using role-specific performance scores, consolidated into a unified score. Role constraints are applied using capability flags rather than rigid labels, allowing realistic overlap between batsmen, bowlers, all-rounders, and wicket keepers.

## Selection of Dream 11 Players:

```
# =========================
# DREAM 11 SELECTION (FROM FULL SQUAD)
# =========================

from pulp import (
   LpProblem, LpVariable, LpMaximize,
   lpSum, LpBinary, LpStatus
)

# -------------------------
# ASSUMPTION:
# full_squad already exists from previous step
# Columns present:
# Player_Name, type,
# Final_Player_Score,
# Final_Batsman_Score, Final_Bowler_Score,
# Final_all_score, Final_wkt_score
# -------------------------

# -------------------------
# STEP 1: Create Dream 11 LP model
# -------------------------
xi_model = LpProblem(
   "Dream_11_Playing_XI_Selection",
   LpMaximize
)

players = full_squad.index

# Decision variable: play in XI or not
play = {
   i: LpVariable(f"play_{i}", cat=LpBinary)
   for i in players
}

# -------------------------
# STEP 2: Objective function
# Maximize on-field performance
# -------------------------
xi_model += lpSum(
   play[i] * full_squad.loc[i, "Final_Player_Score"]
   for i in players
)

# -------------------------
# STEP 3: Playing XI size = 11
# -------------------------
xi_model += lpSum(play[i] for i in players) == 11

# -------------------------
# STEP 4: Dream 11 role constraints
# (Realistic IPL balance)
# -------------------------

# At least 1 Wicket Keeper
xi_model += lpSum(
   play[i] * (full_squad.loc[i, "Final_wkt_score"] > 0)
   for i in players
) >= 1

# At least 3 batsmen
xi_model += lpSum(
   play[i] * (full_squad.loc[i, "Final_Batsman_Score"] > 0)
   for i in players
) >= 3

# At least 2 all-rounders
xi_model += lpSum(
   play[i] * (full_squad.loc[i, "Final_all_score"] > 0)
   for i in players
) >= 2

# At least 3 bowlers
```

```
xi_model += lpSum(
   play[i] * (full_squad.loc[i, "Final_Bowler_Score"] > 0)
   for i in players
) >= 3

# -------------------------
# STEP 5: Solve Dream 11 LP
# -------------------------
xi_model.solve()

print("Dream 11 Status:", LpStatus[xi_model.status])

# -------------------------
# STEP 6: Extract Dream 11
# -------------------------
dream_11 = full_squad.loc[
   [i for i in players if play[i].value() == 1]
]

# -------------------------
# STEP 7: Sanity checks
# -------------------------
print("Total Players in Dream 11:", len(dream_11))

print("\nRole Counts in Dream 11:")
print("Batsmen:", (dream_11["Final_Batsman_Score"] > 0).sum())
print("Bowlers:", (dream_11["Final_Bowler_Score"] > 0).sum())
print("All-rounders:", (dream_11["Final_all_score"] > 0).sum())
print("Wicket Keepers:", (dream_11["Final_wkt_score"] > 0).sum())

# -------------------------
# STEP 8: View Final Dream 11
# -------------------------
dream_11[
   ["Player_Name", "type", "Final_Player_Score"]
].sort_values(
   "Final_Player_Score", ascending=False
)
```

After selecting the optimal 18-member squad under budget constraints, a second binary linear program was formulated to select the best playing XI. The model maximizes on-field performance while enforcing realistic role balance using capability-based constraints, allowing wicket keeper–batsmen and all-rounders to fulfill multiple roles.