# BrainDead 🧠 : The Ultimate Data Analysis & Machine Learning Challenge

Team name: ML Xtreme

Team Leader: Labanya Roy

Institution Name: Meghnad Saha Institute Of Technology

Problem Statement: 1

# Title: "ML-Based IPL 2025 Prediction - A Hackathon Project"

## Problem Statement

1. **What are we solving?**

   Predicting match outcomes, player performance, or team rankings for IPL 2025.

2. **Why is this important?**

   - Enhances team strategies, fantasy leagues, betting analysis, and fan engagement.

# Data Collection & Preprocessing

1. **Data Sources:** Kaggle, IPL Stats, Live APIs.

```python
# Load datasets
matches = pd.read_csv("matches.csv")
deliveries = pd.read_csv("deliveries.csv", on_bad_lines='skip')
```

2. **Preprocessing Steps:**
   - Handling missing data, feature scaling, and encoding categorical values.

```python
# Check for missing values
print("\nMissing Values in Matches Dataset:")
print(matches.isnull().sum())

print("\nMissing Values in Deliveries Dataset:")
print(deliveries.isnull().sum())

# Handle missing values
matches.fillna("Unknown", inplace=True)
deliveries.fillna("Unknown", inplace=True)

# Drop duplicates
matches.drop_duplicates(inplace=True)
deliveries.drop_duplicates(inplace=True)

# Convert 'over' and 'total_runs' columns to numeric to avoid errors
deliveries['over'] = pd.to_numeric(deliveries['over'], errors='coerce')
deliveries['total_runs'] = pd.to_numeric(deliveries['total_runs'], errors='coerce')

# Remove rows with NaN values in 'over' or 'total_runs' columns
deliveries.dropna(subset=['over', 'total_runs'], inplace=True)

# Standardizing team names
team_corrections = {
    "Delhi Daredevils": "Delhi Capitals",
    "Deccan Chargers": "Sunrisers Hyderabad"
}

matches.replace({"team1": team_corrections, "team2": team_corrections, "winner": team_corrections}, inplace=True)
deliveries.replace({"batting_team": team_corrections, "bowling_team": team_corrections}, inplace=True)
```

```
Matches Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1095 entries, 0 to 1094
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               1095 non-null   int64
 1   season           1095 non-null   object
 2   city             1044 non-null   object
 3   date             1095 non-null   object
 4   match_type       1095 non-null   object
 5   player_of_match  1090 non-null   object
 6   venue            1095 non-null   object
 7   team1            1095 non-null   object
 8   team2            1095 non-null   object
 9   toss_winner      1095 non-null   object
 10  toss_decision    1095 non-null   object
 11  winner           1090 non-null   object
 12  result           1095 non-null   object
 13  result_margin    1076 non-null   float64
 14  target_runs      1092 non-null   float64
 15  target_overs     1092 non-null   float64
 16  super_over       1095 non-null   object
 17  method           21 non-null     object
 18  umpire1          1095 non-null   object
 19  umpire2          1095 non-null   object
dtypes: float64(3), int64(1), object(16)
memory usage: 171.2+ KB
None
```

# Machine Learning Models & Evaluation

**1. Models Used:** Random Forest (Baseline), XGBoost, LGBM Model.

**2. Metrics Used:** Accuracy, Precision, Recall, F1-Score, RMSE.

3. **Comparison of Models:** Show a table graph comparing performance.

```python
# Model Training
rf_model = RandomForestClassifier(n_estimators=200, max_depth=20, class_weight='balanced_subsample', random_state=42)

gb_model = GradientBoostingClassifier(n_estimators=200, max_depth=7, learning_rate=0.05, random_state=42)

xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=10,
    learning_rate=0.05,
    objective='multi:softmax',  # For classification output
    num_class=len(y_train.cat.categories),  # Number of classes
    eval_metric='mlogloss',
    random_state=42
)

lgbm_optimized = LGBMClassifier(
    n_estimators=296,
    max_depth=19,
    learning_rate=0.021,
    num_leaves=22,
    colsample_bytree=0.581,
    subsample=0.632,
    random_state=42
)
```

```python
# Model Evaluation
y_pred_rf = rf_model.predict(X_test)
y_pred_gb = gb_model.predict(X_test)
y_pred_xgb = xgb_model.predict(X_test)
y_pred_lgbm = lgbm_optimized.predict(X_test)

results = pd.DataFrame({
    'Model': ['Random Forest', 'Gradient Boosting', 'XGBoost', 'LightGBM'],
    'Accuracy': [accuracy_score(y_test, y_pred_rf), accuracy_score(y_test, y_pred_gb), accuracy_score(y_test, y_pred_xgb), accuracy_score(y_test, y_pred_lgbm)]
})
```

# RESULTS & INSIGHTS

**Final Model Accuracy:** 60% (Highlight best model's performance).

**Predictions Example:**

Match outcome predictions.
Player performance forecasts.

**Business Use Case:**

Impact on IPL teams, betting companies, and fantasy leagues.

| | Model | Accuracy |
|---|---|---|
| 0 | Random Forest | 0.515982 |
| 1 | Gradient Boosting | 0.547945 |
| 2 | XGBoost | 0.557078 |
| 3 | LightGBM | 0.579909 |

```
results = pd.DataFrame({
    'Model': ['Random Forest', 'Gradient Boosting', 'XGBoost', 'LightGBM'],
    'Accuracy': [accuracy_score(y_test, y_pred_rf), accuracy_score(y_test, y_pred_gb), accuracy_score(y_test, y_pred_xgb), accuracy_score(y_test, y_pred_lgbm)]
})
```

## Future Scope

- Real-time predictions .
- Using ML models for strategy optimization.

```
Random Forest Prediction: Sunrisers Hyderabad
Gradient Boosting Prediction: Sunrisers Hyderabad
XGBoost Prediction: Rajasthan Royals
LightGBM Prediction: Sunrisers Hyderabad

🎯 Final Predicted Winner (by Voting): Sunrisers Hyderabad
```

## Conclusion

**“ Machine Learning meets Cricket – The Future of IPL Predictions! “**

## Problem Statement : 2

# Research Article Summarization using Advanced NLP Techniques

### Problem Statement

**Objective:** Develop an advanced summarization model to generate concise, informative summaries.

**Complexities:**

1. Structured format (Introduction, Methods, Results, Discussion).
2. Citation dependencies.
3. Tables, figures, and domain-specific knowledge retention.

# SOLUTION & DATASET OVERVIEW

**Approach:** Hybrid extractive-abstractive summarization using Large Language Models (LLMs).

**Key Features:**

1. Summarize single and multi-document research papers.

2. Handles long-document summarization efficiently.

**Datasets Used:**

1. CompScholar (370 research articles across domains).

2. PubMed (Millions of biomedical research articles).

3. arXiv (1.7M scientific papers across disciplines).

```
# Define dataset path
dataset_path = "/content/drive/MyDrive/Brain_Dead_CompScholar_Dataset.csv"

# Load dataset
dataset = load_dataset("csv", data_files=dataset_path, split="train")
print("Dataset Columns:", dataset.column_names)

# Split dataset into training and validation sets
dataset = dataset.train_test_split(test_size=0.1)
train_dataset, val_dataset = dataset["train"], dataset["test"]
```

# Data Preprocessing & Handling

**Preprocessing Steps:**

1. Tokenization, stop-word removal, and stemming.
2. Handling missing values and OCR text.
3. Encoding citation dependencies and references.

**Challenges:**

1. Large context Length.

```python
# Load tokenizers
bert_tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
bart_tokenizer = AutoTokenizer.from_pretrained("facebook/bart-large-cnn")

# Tokenization function for BERT (Extractive)
def extractive_preprocess_function(examples):
    inputs = bert_tokenizer(examples["Document"], padding="max_length", truncation=True, max_length=512)
    labels = [1 if "important" in doc.lower() else 0 for doc in examples["Document"]]  # Sample label strategy
    inputs["labels"] = labels
    return inputs

# Tokenization function for BART (Abstractive)
def abstractive_preprocess_function(examples):
    inputs = bart_tokenizer(examples["Document"], padding="max_length", truncation=True, max_length=1024)
    labels = bart_tokenizer(examples["Summary"], padding="max_length", truncation=True, max_length=256)
    inputs["labels"] = labels["input_ids"]
    return inputs

# Apply tokenization
train_dataset = train_dataset.map(extractive_preprocess_function, batched=True)
val_dataset = val_dataset.map(extractive_preprocess_function, batched=True)
train_dataset_bart = train_dataset.map(abstractive_preprocess_function, batched=True)
val_dataset_bart = val_dataset.map(abstractive_preprocess_function, batched=True)
```

## Model Selection & Architecture

**Extractive Summarization Models:**

BERTSUM.

**Abstractive Summarization Models:**

BART

**Hybrid Approach:**

1. Combination of extractive and abstractive techniques.

2. Fine-tuning pre-trained LLMs for domain-specific summarization.

```
sample_text = """
Artificial intelligence is transforming industries worldwide.
It has applications in healthcare, finance, and transportation.
Recent advancements in deep learning have improved AI performance.
"""

print("Extractive Summary:\n", extractive_summary(sample_text))
print("Abstractive Summary:\n", abstractive_summary(sample_text))
print("Hybrid Summary:\n", hybrid_summary(sample_text))
```

```
Extractive Summary:

Artificial intelligence is transforming industries worldwide.
It has applications in healthcare, finance, and transportation.
Recent advancements in deep learning have improved AI performance.

Abstractive Summary:
 , healthcare, finance, finance, and transportation are just some of the industries where artificial intelligence has applications in healthcare, finance
Hybrid Summary:
 , healthcare, finance, finance, and transportation are just some of the industries where artificial intelligence has applications in healthcare, finance
```

```python
def abstractive_summary(text, window_size=1024, stride=512):
    sentences = text.split(". ")
    summaries = []

    for i in range(0, len(sentences), stride):
        chunk = ". ".join(sentences[i:i + window_size])
        inputs = bart_tokenizer(chunk, return_tensors="pt", truncation=True, max_length=1024).to(device)
        summary_ids = abstractive_model.generate(
            inputs["input_ids"], max_length=200, min_length=50, num_beams=5, early_stopping=True
        )
        summaries.append(bart_tokenizer.decode(summary_ids[0], skip_special_tokens=True))

    return " ".join(summaries)
```

```python
def hybrid_summary(text):
    extracted_text = extractive_summary(text)  # Extract key sentences
    return abstractive_summary(extracted_text) if extracted_text else abstractive_summary(text)
```

# Model Training & Evaluation

**Performance Metrics:**

1. ROUGE-1, ROUGE-2, ROUGE-L (content overlap).

2. BLEU Score (text fluency and coherence).

3. Summarization length and readability.

4. Computational efficiency (training time, inference speed).

```python
# Training Arguments for Extractive Model
training_args = TrainingArguments(
    output_dir="./bert_results",
    evaluation_strategy="epoch",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    save_total_limit=2,
    save_strategy="epoch",
    fp16=True,
    gradient_accumulation_steps=4,
    logging_dir="./logs",
)

# Train Extractive Model
bert_trainer = Trainer(
    model=extractive_model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=bert_tokenizer,
    data_collator=data_collator,
)
bert_trainer.train()
extractive_model.save_pretrained("./fine_tuned_bert")

# Training Arguments for Abstractive Model
bart_training_args = TrainingArguments(
    output_dir="./bart_results",
```

```
!zip -r fine_tuned_models.zip fine_tuned_bart fine_tuned_bert

  adding: fine_tuned_bart/ (stored 0%)
  adding: fine_tuned_bart/config.json (deflated 61%)
  adding: fine_tuned_bart/model.safetensors (deflated 7%)
  adding: fine_tuned_bart/generation_config.json (deflated 47%)
  adding: fine_tuned_bert/ (stored 0%)
  adding: fine_tuned_bert/config.json (deflated 49%)
  adding: fine_tuned_bert/model.safetensors (deflated 7%)

from google.colab import files
files.download("fine_tuned_models.zip")
```

```python
from rouge_score import rouge_scorer
from sacrebleu import corpus_bleu
from bert_score import score as bert_score

def evaluate_summary(reference, generated):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
    return scorer.score(reference, generated)

def evaluate_bleu(reference, generated):
    return corpus_bleu([generated], [[reference]]).score

def evaluate_bert_score(reference, generated):
    P, R, F1 = bert_score([generated], [reference], lang="en")
    return {"Precision": P.mean().item(), "Recall": R.mean().item(), "F1": F1.mean().item()}
```

# Results & Insights

**Example Summaries:** Before vs. After Summarization.

**Key Observations:**

1. Improved readability while retaining key insights.
2. Efficient summarization of long documents.

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.d
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.d
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

===== Fine-Tuned Models =====
Extractive Summary:
Artificial intelligence is transforming industries worldwide.
It has applications in healthcare, finance, and transportation.
Recent advancements in deep learning have improved AI performance.

Abstractive Summary: , healthcare, finance, finance, and transportation are just some of the industries where artificial intelligence has applications in healthcare, f
ROUGE Extractive: {'rouge1': Score(precision=0.43478260869565216, recall=0.7142857142857143, fmeasure=0.5405405405405405), 'rouge2': Score(precision=0.0454545454545454
ROUGE Abstractive: {'rouge1': Score(precision=0.14285714285714285, recall=0.2857142857142857, fmeasure=0.19047619047619047), 'rouge2': Score(precision=0.0, recall=0.0,
BLEU Score: 2.6227235705350953
BERTScore: {'Precision': 0.8509353399276733, 'Recall': 0.8890441656112671, 'F1': 0.8695724010467529}

===== Pre-Trained Models (Without Fine-Tuning) =====
Extractive Summary:
Artificial intelligence is transforming industries worldwide.
It has applications in healthcare, finance, and transportation.
Recent advancements in deep learning have improved AI performance.

Abstractive Summary: Artificial intelligence is transforming industries worldwide. It has applications in healthcare, finance, and transportation. Recent advancements
ROUGE Extractive: {'rouge1': Score(precision=0.43478260869565216, recall=0.7142857142857143, fmeasure=0.5405405405405405), 'rouge2': Score(precision=0.0454545454545454
ROUGE Abstractive: {'rouge1': Score(precision=0.23809523809523808, recall=0.7142857142857143, fmeasure=0.35714285714285715), 'rouge2': Score(precision=0.02439024390243
BLEU Score: 3.06569275855315
BERTScore: {'Precision': 0.8883844614028931, 'Recall': 0.9423068165779114, 'F1': 0.9145514965057373}
```

## Challenges Faced

**1. Data Limitations:** huge dataset taking time to complete the task.

**2. External Factors:** Injuries, real-time pitch conditions.

**3. Computational Constraints:** Training time, real-time processing issues.

## Future Scope

1. Real-time predictions using live APIs.

2. Using Reinforcement Learning for strategy optimization.

3. Building a web/app interface for user-friendly predictions.

# Thank You

## Team Members

1. Labanya Roy
2. Priyanka Shaw
3. Dipayan Paul
4. Sampurna Mallick
5. Snehal Banerjee

## Git hub

https://github.com/labanya-1/XTREAME_Brain_dead