PLAYER DELIVERIES ANALYSIS DATA USING PYTHON AND ML

MADHURA VIRKAR

AGENDA

1. FEATURE ENGINEERING / DATA PREPARATION
2. SUPERVISED LEARNING
3. EVALUATION & MODEL TUNING
4. TIME SERIES & SEQUENCE MODELING

FEATURE ENGINEERING / DATA PREPARATION

1. How would you preprocess this ball-by-ball data to predict the final score of an inning?

```
final_score_per_inning = df2.groupby(['match_id', 'inning', 'over']).agg(
    runs_per_over=('total_runs', 'sum'),
    wickets_per_over=('is_wicket', 'sum')
).reset_index()

final_score_per_inning.head(10)
```

match_id	inning	over	runs_per_over	wickets_per_over
335982	1	0	3	0
335982	1	1	18	0
335982	1	2	6	0
335982	1	3	23	0
335982	1	4	10	0
335982	1	5	1	1
335982	1	6	7	0
335982	1	7	5	0
335982	1	8	4	0
335982	1	9	10	0

2. Can you create features like “strike rate” or “bowler economy rate” from this data? How?

```
batting_score = df2.groupby('batter').agg(
    total_runs=('batsman_runs', 'sum'),
    balls_faced=('ball', 'count')  # assumes each row = 1 ball
).reset_index()
batting_score['strike_rate'] = batting_score['total_runs'] / (batting_score['balls_faced'] * 100)
batting_score.head()
```

```
bowler_performance = df2.groupby('bowler').agg(
    runs_conceded=('total_runs', 'sum'),
    balls_bowled=('ball', 'count')
).reset_index()

bowler_performance['overs'] = bowler_performance['balls_bowled'] / 6

bowler_performance['economy_rate'] = bowler_performance['runs_conceded'] / bowler_performance['overs']

bowler_performance.head()
```

	batter	total_runs	balls_faced	strike_rate
0	A Ashish Reddy	280	196	0.014286
1	A Badoni	634	505	0.012554
2	A Chandila	4	7	0.005714
3	A Chopra	53	75	0.007067
4	A Choudhary	25	20	0.012500

bowler	runs_conceded	balls_bowled	overs	economy_rate
A Ashish Reddy	400	270	45.000000	8.888889
A Badoni	37	25	4.166667	8.880000
A Chandila	245	234	39.000000	6.282051
A Choudhary	144	108	18.000000	8.000000
A Dananjaya	47	25	4.166667	11.280000

3. Build a dataset where each row represents an over. What features would you include?

```
columns = ['Bowler', 'Batsman', 'Over Number', 'Ball 1', 'Ball 2', 'Ball 3', 'Ball 4', 'Ball 5', 'Ball 6']
df = pd.DataFrame(columns=columns)
over_data = [
    ['Bowler A', 'Batsman X', 1, 1, 0, 4, 0, 1, 2],
    ['Bowler B', 'Batsman Y', 2, 1, 4, 0, 0, 6, 1],
    ['Bowler A', 'Batsman X', 3, 0, 1, 6, 4, 0, 1],
    ['Bowler B', 'Batsman Y', 4, 2, 0, 0, 6, 1, 2],
]

# Convert the list of lists to a DataFrame
df = pd.DataFrame(over_data, columns=columns)
df.head()
```

	Bowler	Batsman	Over Number	Ball 1	Ball 2	Ball 3	Ball 4	Ball 5	Ball 6
0	Bowler A	Batsman X	1	1	0	4	0	1	2
1	Bowler B	Batsman Y	2	1	4	0	0	6	1
2	Bowler A	Batsman X	3	0	1	6	4	0	1
3	Bowler B	Batsman Y	4	2	0	0	6	1	2

3. Build a dataset where each row represents an over. What features would you include?

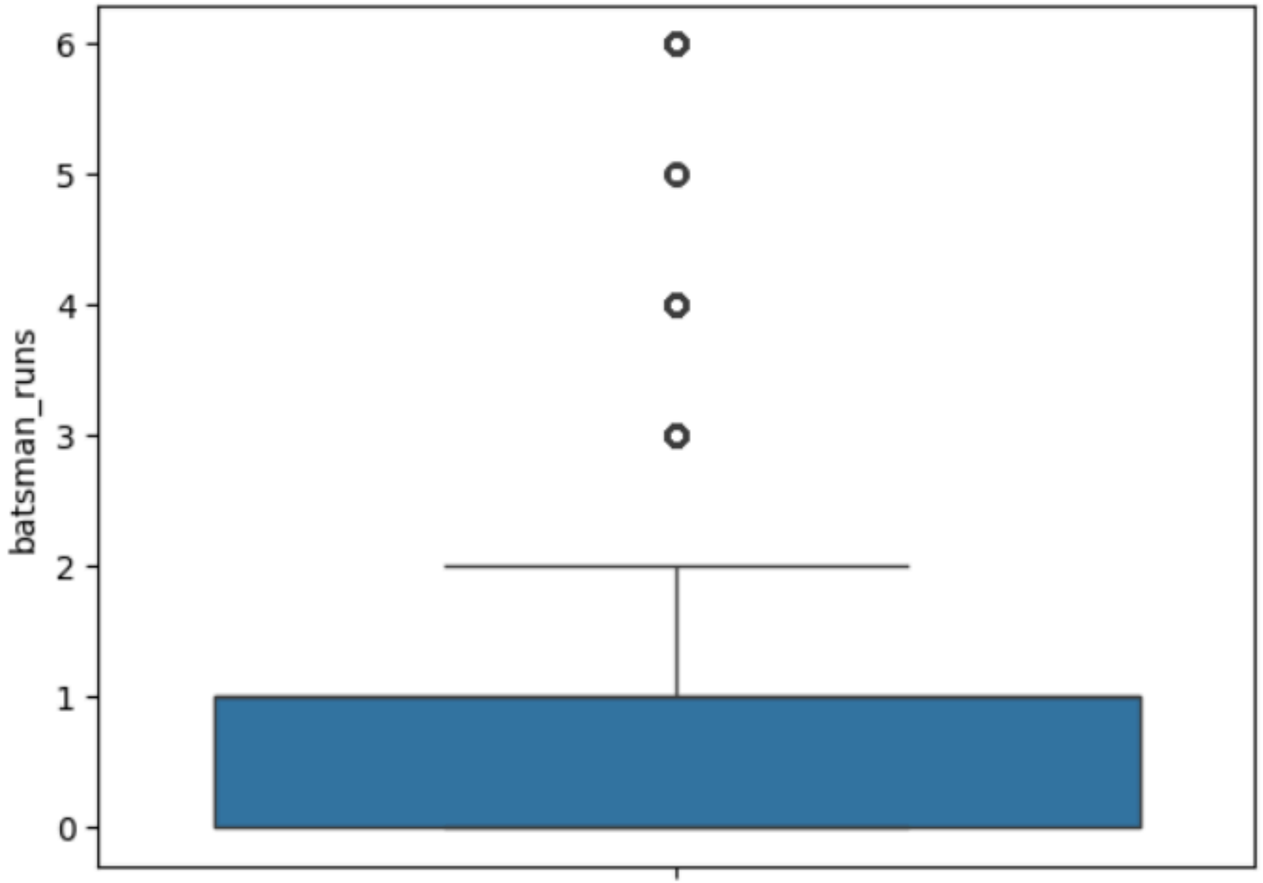
```
df['Runs_per_over'] = df['Ball 1'] + df['Ball 2'] + df['Ball 3'] + df['Ball 4'] + df['Ball 5'] + df['Ball 6']
df.head()
```

	Bowler	Batsman	Over Number	Ball 1	Ball 2	Ball 3	Ball 4	Ball 5	Ball 6	Runs_per_over
0	Bowler A	Batsman X	1	1	0	4	0	1	2	8
1	Bowler B	Batsman Y	2	1	4	0	0	6	1	12
2	Bowler A	Batsman X	3	0	1	6	4	0	1	12
3	Bowler B	Batsman Y	4	2	0	0	6	1	2	11

4. How would you detect and handle outliers in batsman_runs or total_runs?

```
sns.boxplot(df2['batsman_runs'])
```

```
<Axes: ylabel='batsman_runs'>
```



Runs more than 2 per over are outliers

5. Engineer a feature that captures momentum (e.g., runs scored in the last 2 overs). How would you implement it in Pandas?

```python
# Function to calculate runs in last 2 overs
def calculate_runs_last_2_overs(df2):
    df2['last_2_overs_runs'] = 0
    innings = df2['inning'].unique()
    for inning in innings:
        inning_data = df2[df2['inning'] == inning]
        max_over = inning_data['over'].max()
        last_2_overs_data = inning_data[(inning_data['over'] >= max_over - 1) & (inning_data['over'] <= max_over)]
        df2.loc[df2['inning'] == inning, 'last_2_overs_runs'] = last_2_overs_data['total_runs'].sum()
    return df2

# Apply the function
df2 = calculate_runs_last_2_overs(df2)
print(df2)
```

```
        last_2_overs_runs
0                   23621
1                   23621
2                   23621
3                   23621
4                   23621
...                   ...
260915              13220
260916              13220
260917              13220
260918              13220
260919              13220
```

SUPERVISED LEARNING

6. Build a regression model to predict the final inning total using data available till a certain over. Which regression algorithms would you try (Linear Regression, XGBoost, etc.)?

```
#Linear regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_pred = lr_model.predict(X_test)

print("Linear Regression:", mean_absolute_error(y_test, lr_pred))
```

```
Linear Regression: 17.383214836970904
```

```
#XGBoost model
xgb = XGBRegressor()
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)
print("XGBoost:", mean_absolute_error(y_test, xgb_pred))
```

```
XGBoost: 17.33625602722168
```

Coparatively XGBoost is giving lesser errors. If we have more data then results will be better.

7. Can you train a classification model to predict whether a team will score more than 180 runs in an inning? What would your target and features be?

```
model_data['score_above_180'] = model_data['final_total'].apply(lambda x: 1 if x > 180 else 0)

X = model_data[['over', 'cum_runs', 'cum_wickets']]
y = model_data['score_above_180']
```

```
from sklearn.metrics import precision_recall_fscore_support as score

clf = LogisticRegression(max_iter=10000, random_state=0)
clf.fit(X_train, y_train)

acc = accuracy_score(y_test, clf.predict(X_test)) * 100
print(f"Logistic Regression model accuracy: {acc:.2f}%")

precision, recall, fscore, support = score(y_test, clf.predict(X_test))
print('precision: {}'.format(precision))
print('recall: {}'.format(recall))
print('fscore: {}'.format(fscore))
```

```
Logistic Regression model accuracy: 81.63%
precision: [0.83071796 0.73247381]
recall: [0.94741844 0.42716165]
fscore: [0.88523862 0.539626  ]
```

Accuracy, Precision showing almost same result whereas for recall there can be overfitting.

8. Train a model to predict whether a wicket will fall on a given ball. Which features are most predictive?

```
#Random forest
model = RandomForestClassifier(random_state=23, class_weight='balanced')
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=23)
```

```
# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_rep)
```

Result shows it is a A perfect model would have zero false positives and zero false negatives and therefore an accuracy of 1.0, or 100%

```
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     49679
           1       1.00      1.00      1.00      2505

    accuracy                           1.00     52184
   macro avg       1.00      1.00      1.00     52184
weighted avg       1.00      1.00      1.00     52184
```

Very high training accuracy in a random forest is normal and does not indicate that the random forest is overfitted

9. Create a model to predict the next ball's runs (0, 1, 2, 4, 6). Which model performs best: Random Forest, Logistic Regression, or Gradient Boosting?

```
nextBall=df2[['match_id','inning','over','ball','total_runs']]
nextBall=nextBall.sort_values(by=['match_id','inning','over','ball']).reset_index(drop=True)
nextBall

nextBall['next_ball_run'] = (nextBall['total_runs'].shift(-1))
nextBall.head()
```

	match_id	inning	over	ball	total_runs	next_ball_run
0	335982	1	0	0	1	0.0
1	335982	1	0	1	0	1.0
2	335982	1	0	2	1	0.0
3	335982	1	0	3	0	0.0
4	335982	1	0	4	0	0.0

10. Using past data, predict the “Player of the Match” using features like runs scored, wickets taken, strike rate, etc.

```
df2['strike_rate1'] = batting_score['total_runs'] / (batting_score['balls_faced'] * 100)
data = df2[['match_id','batter','non_striker','total_runs', 'is_wicket', 'strike_rate1']]
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Logistic Regression model
model1 = LogisticRegression()
model1.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model1.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# Example prediction for a new match
new_match_data = df2[['match_id','batter','non_striker','total_runs', 'is_wicket', 'strike_rate1']]
new_match_data = new_match_data.reindex(columns=X.columns, fill_value=0)
predicted_player = model1.predict(new_match_data)
print(f'Predicted Player of the Match: {predicted_player}')
```

```
Accuracy: 1.0
Predicted Player of the Match: [1 1 1 ... 1 1 1]
```

EVALUATION & MODEL TUNING

11. What evaluation metrics would you use for your regression and classification models? Why?

For regression models, I would use Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (Coefficient of Determination).

- Evaluation metrics quantify how well a regression model performs on unseen data
- Different metrics capture different aspects of model performance (error, variance explained, etc.)
- Interpreting multiple metrics provides a comprehensive understanding of a model’s strengths and limitations
- Regression metrics are essential tools for evaluating the performance of predictive models, helping to quantify accuracy and guide improvements.

11. What evaluation metrics would you use for your regression and classification models? Why?

For classification models, I would use Accuracy, Precision, Recall, F1-score, and ROC-AUC.

- True Positive and True Negative values mean the predicted value matches the actual value.
- The matrix helps in understanding where the model has gone wrong and gives guidance to correct the path
- A Type I Error happens when the model makes an incorrect prediction, as in, the model predicted positive for an actual negative value.
- A Type II Error happens when the model makes an incorrect prediction of an actual positive value as negative.

12. How would you use cross-validation to improve model reliability?

```python
from sklearn.model_selection import KFold
kf = KFold(n_splits=2, shuffle=True, random_state=42) # shuffle and random_state are optional

# Create Logistic Regression model
model_reg = LogisticRegression()

X = np.random.rand(100, 10)
y = np.random.randint(0, 2, 100)
# Perform cross-validation
#X = df2[['match_id','batter','non_striker','total_runs', 'is_wicket', 'strike_rate1']]
cv_results = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model
    model_reg.fit(X_train, y_train)

    # Make predictions
    y_pred = model_reg.predict(X_test)

    # Evaluate performance
    accuracy = accuracy_score(y_test, y_pred)
    cv_results.append(accuracy)

print(cv_results) # Output will be a list of accuracies for each fold
print(np.mean(cv_results)) # Output will be the average accuracy across all folds
```

```
[0.32, 0.46]
0.39
```

Accuracy 0.39 need more data

13. What methods would you use to handle class imbalance if wickets happen in <10% of deliveries?

Synthetic Minority Oversampling Technique

```
X = df2.drop('is_wicket', axis=1)
y = df2['is_wicket']


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

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train a model (e.g., Logistic Regression) on the resampled data
model = LogisticRegression()
model.fit(X_train_resampled, y_train_resampled)

# Evaluate the model on the test set
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.989709489498697
```

TIME SERIES & SEQUENCE MODELING

14. Can you model run scoring as a time series (over-wise or ball-wise)? Try using LSTM or ARIMA.

```
X = df2['total_runs']  # Time series data
X_train, X_test = train_test_split(X, test_size=0.2, shuffle=False)  # Ensure time series is not shuffled
model = sm.tsa.arima.ARIMA(X_train, order=(5, 1, 0))  # Example order
model_fit = model.fit()
predictions = model_fit.predict(start=len(X_train), end=len(X) - 1)

#  Forecasting and Evaluation
rmse = mean_squared_error(X_test, predictions)
print(f'RMSE: {rmse}')
```

```
RMSE: 3.1932117918334
```

15. How would you implement a rolling average or exponential moving average of batsman performance for use as features in a predictive model?

YOUR PARAGRAPH TEXT

```
scaler = MinMaxScaler()
scaled_array = scaler.fit_transform(Batsman_df)

# Convert back to DataFrame
Batsman_df = pd.DataFrame(scaled_array, columns=Batsman_df.columns, index=Batsman_df.index)

Batsman_df.head()
# Features and target
X = Batsman_df[['rolling_avg_3', 'ema_runs']]
y = Batsman_df['target_runs_next_match']

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model
model_player_avg = RandomForestRegressor()
model_player_avg.fit(X_train, y_train)

# Predict
y_pred = model_player_avg.predict(X_test)

# 5-Fold Cross Validation on Regression Problem
scores = cross_val_score(model_player_avg, X, y, cv=5, scoring='neg_mean_absolute_error')
print("Neg Mean absolute error after cross validation : "+str(-scores))
# Evaluate
print("MAE:", mean_absolute_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
Neg Mean absolute error after cross validation : [0.0998249  0.09782205 0.09348002 0.09864181 0.09987807]
MAE: 0.09531194486510462
R² Score: -0.15571002482360918
```

THANK YOU