```
!pip install catboost
!pip install lightgbm
!pip install xgboost
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.10/dist-packages (1.2.7)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from cat
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from c
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboo
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catbo
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: lightgbm in /usr/local/lib/python3.10/dist-packages (4.5.0)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lightg
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboos
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboos
```

```
import pandas as pd


df = pd.read_csv('merged_dataset.csv')


print(df.shape)
print(df.head)
print(df.dtypes)
print(df.describe())
df.isnull().sum()
df.dropna(inplace=True)
```

```
(9124, 8)
<bound method NDFrame.head of             Date          HomeTeam   AwayTeam  HomeOdds  DrawO
0     2024-12-06       fc volendam    jong az      1.45      4.95      5.18
1     2024-12-06           helmond  den bosch      2.80      3.52      2.27
2     2024-12-06        maastricht    vitesse      1.96      3.73      3.34
3     2024-12-06             venlo        oss      1.84      3.61      3.82
4     2024-12-06         jong ajax       roda      2.93      3.68      2.15
...          ...               ...        ...       ...       ...       ...
9119  2004-04-12           helmond  apeldoorn      1.68      3.40      4.28
9120  2004-04-12        maastricht    cambuur      1.89      3.35      3.40
```

```
9121  2004-04-12          sittard    telstar      2.70      3.25      2.23
9122  2004-04-12  sparta rotterdam        oss      1.40      3.80      5.70
9123  2004-04-12          heracles    veendam      1.45      3.60      5.40

      HomeGoals  AwayGoals
0           2.0        0.0
1           0.0        0.0
2           2.0        2.0
3           0.0        0.0
4           3.0        1.0
...         ...        ...
9119        3.0        0.0
9120        1.0        1.0
9121        1.0        1.0
9122        0.0        0.0
9123        3.0        2.0

[9124 rows x 8 columns]>
Date          object
HomeTeam      object
AwayTeam      object
HomeOdds     float64
DrawOdds     float64
AwayOdds     float64
HomeGoals    float64
AwayGoals    float64
dtype: object
          HomeOdds     DrawOdds     AwayOdds    HomeGoals    AwayGoals
count  9124.000000  9124.000000  9124.000000  9116.000000  9116.000000
mean      2.364929     3.864065     3.925253     1.722356     1.377139
std       1.438227     0.840965     2.324258     1.371774     1.232375
min       1.090000     2.450000     1.130000     0.000000     0.000000
25%       1.650000     3.350000     2.500000     1.000000     0.000000
50%       2.020000     3.610000     3.280000     1.000000     1.000000
75%       2.600000     4.030000     4.620000     2.000000     2.000000
max      22.200000     9.820000    20.770000     8.000000     6.000000
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import mean_squared_error

# Feature engineering
df['TotalGoals'] = df['HomeGoals'] + df['AwayGoals']
df['GoalDifference'] = df['HomeGoals'] - df['AwayGoals']
df['Over2.5'] = (df['TotalGoals'] > 2.5).astype(int)
df['MatchResult'] = df.apply(lambda row: 'Win' if row['HomeGoals'] > row['AwayGoals'] else ('Dra
df['Over3.5'] = (df['TotalGoals'] > 3.5).astype(int)

# Encode the MatchResult target variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['MatchResult'] = le.fit_transform(df['MatchResult'])
```

```
# Select features and target
features = ['HomeOdds', 'DrawOdds', 'AwayOdds']
target_score_home = 'HomeGoals'
target_score_away = 'AwayGoals'
target_over_2_5 = 'Over2.5'
target_match_result = 'MatchResult'
target_over_3_5 = 'Over3.5'
# Scaling features
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

# Split the data
X_train, X_test, y_train_home, y_test_home = train_test_split(df[features], df[target_score_home
_, _, y_train_away, y_test_away = train_test_split(df[features], df[target_score_away], test_siz
_, _, y_train_over_2_5, y_test_over_2_5 = train_test_split(df[features], df[target_over_2_5], te
_, _, y_train_Result, y_test_Result = train_test_split(df[features], df[target_match_result], te
X_train3_5, X_test3_5, y_train_over_3_5, y_test_over_3_5 = train_test_split(df[features], df[tar
```

```
df.head()
```

	Date	HomeTeam	AwayTeam	HomeOdds	DrawOdds	AwayOdds	HomeGoals	AwayGoals	TotalGoals
0	2024-12-06	fc volendam	jong az	-0.635959	1.290464	0.539372	2.0	0.0	2.0
1	2024-12-06	helmond	den bosch	0.302338	-0.409684	-0.712223	0.0	0.0	0.0
2	2024-12-06	maastricht	vitesse	-0.281491	-0.160012	-0.252014	2.0	2.0	4.0
3	2024-12-06	venlo	oss	-0.364895	-0.302681	-0.045566	0.0	0.0	0.0
4	2024-12-06	jong ajax	roda	0.392692	-0.219457	-0.763835	3.0	1.0	4.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Predict Home Goals
rf_regressor_home = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor_home.fit(X_train, y_train_home)
y_pred_home = rf_regressor_home.predict(X_test)
print(f"RMSE for Home Goals: {mean_squared_error(y_test_home, y_pred_home)}")

# Predict Away Goals
rf_regressor_away = RandomForestRegressor(n_estimators=100, random_state=42)
```

```
rf_regressor_away.fit(X_train, y_train_away)
y_pred_away = rf_regressor_away.predict(X_test)
print(f"RMSE for Away Goals: {mean_squared_error(y_test_away, y_pred_away)}")
```

```
RMSE for Home Goals: 0.027322762842173557
RMSE for Away Goals: 0.006427528631383909
```

```
import lightgbm as lgb

# Predict Over 2.5 Goals
lgb_classifier = lgb.LGBMClassifier(n_estimators=100, max_depth=6, learning_rate=0.2, random_sta
lgb_classifier.fit(X_train, y_train_over_2_5)
y_pred_over_2_5_lgb = lgb_classifier.predict(X_test)
print(f"Accuracy for Over 2.5 Goals (LightGBM): {accuracy_score(y_test_over_2_5, y_pred_over_2_5_
print(f"F1 Score for Over 2.5 Goals (LightGBM): {f1_score(y_test_over_2_5, y_pred_over_2_5_lgb)}
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Accuracy for Over 2.5 Goals (LightGBM): 0.9566885964912281
F1 Score for Over 2.5 Goals (LightGBM): 0.9636112390603409
```

```
import xgboost as xgb

# Predict Over 2.5 Goals
xgb_classifier = xgb.XGBClassifier(n_estimators=100, max_depth=6, learning_rate=0.1, random_stat
xgb_classifier.fit(X_train, y_train_over_2_5)
y_pred_over_2_5_xgb = xgb_classifier.predict(X_test)
print(f"Accuracy for Over 2.5 Goals (XGBoost): {accuracy_score(y_test_over_2_5, y_pred_over_2_5_
print(f"F1 Score for Over 2.5 Goals (XGBoost): {f1_score(y_test_over_2_5, y_pred_over_2_5_xgb)}"
```

```
Accuracy for Over 2.5 Goals (XGBoost): 0.9237938596491229
F1 Score for Over 2.5 Goals (XGBoost): 0.9375841939829367
```

```
import lightgbm as lgb
#predict over 3.5 Goals
lgb3_classifier = lgb.LGBMClassifier(n_estimators=100, max_depth=6, learning_rate=0.15, random_s
lgb3_classifier.fit(X_train3_5, y_train_over_3_5)
y_pred_over_3_5_lgb = lgb3_classifier.predict(X_test3_5)
print(f"Accuracy for Over 3.5 Goals (LightGBM): {accuracy_score(y_test_over_3_5, y_pred_over_3_5_
print(f"F1 Score for Over 3.5 Goals (LightGBM): {f1_score(y_test_over_3_5, y_pred_over_3_5_lgb)}
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Accuracy for Over 3.5 Goals (LightGBM): 0.9188299817184643
F1 Score for Over 3.5 Goals (LightGBM): 0.8825396825396825
```

```
from catboost import CatBoostClassifier
from sklearn.metrics import accuracy_score, f1_score

cb_classifier = CatBoostClassifier(iterations=100, depth=7, learning_rate=0.4, random_seed=52, v
cb_classifier.fit(X_train, y_train_over_2_5)
y_pred_over_2_5 = cb_classifier.predict(X_test)

print(f"Accuracy for Over 2.5 Goals: {accuracy_score(y_test_over_2_5, y_pred_over_2_5)}")
print(f"F1 Score for Over 2.5 Goals: {f1_score(y_test_over_2_5, y_pred_over_2_5)}")
```

```
Accuracy for Over 2.5 Goals: 0.9254385964912281
F1 Score for Over 2.5 Goals: 0.9376718606782768
```

Start coding or generate with AI.

Generate print hello world using rot13 Close

```
import lightgbm as lgb
from sklearn.metrics import accuracy_score, f1_score

# Initialize the LightGBM model
lgb_match_result = lgb.LGBMClassifier(n_estimators=100, max_depth=6, learning_rate=0.1, random_s

# Train the model
lgb_match_result.fit(X_train, y_train_Result)

# Make predictions
y_pred_match_result = lgb_match_result.predict(X_test)

# Evaluate the model
accuracy_match_result = accuracy_score(y_test_Result, y_pred_match_result)
f1_match_result = f1_score(y_test_Result, y_pred_match_result, average='weighted')
print(f"Accuracy for Match Result (LightGBM): {accuracy_match_result}")
print(f"F1 Score for Match Result (LightGBM): {f1_match_result}")
```

```
tGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000432 s
an set `force_col_wise=true` to remove the overhead.
tGBM] [Info] Total Bins 749
tGBM] [Info] Number of data points in the train set: 7292, number of used features: 3
tGBM] [Info] Start training from score -1.485397
tGBM] [Info] Start training from score -1.159100
tGBM] [Info] Start training from score -0.776922
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
tGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import StratifiedKFold

# Assuming you have already prepared your dataset
# X, y = ...

# Split the data into training and test sets
# The original code used features, which is a list of column names
# Instead, use df[features] to select the actual data from the dataframe
X_train, X_test, y_train, y_test = train_test_split(df[features], df[target_match_result], test_si

# Initialize the Gradient Boosting Classifier with reduced complexity
gb_classifier = GradientBoostingClassifier(n_estimators=200, max_depth=5, learning_rate=0.3, rando

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

# Predictions
y_pred_train = gb_classifier.predict(X_train)
y_pred_test = gb_classifier.predict(X_test)

# Evaluate the model
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)
train_f1 = f1_score(y_train, y_pred_train, average='weighted')
test_f1 = f1_score(y_test, y_pred_test, average='weighted')

print(f"Training Accuracy: {train_accuracy}")
print(f"Test Accuracy: {test_accuracy}")
print(f"Training F1 Score: {train_f1}")
```

```
print(f"Test F1 Score: {test_f1}")

# Cross-validation to check for overfitting
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(gb_classifier, df[features], df[target_match_result], cv=cv, scoring='
print(f"Cross-validation Accuracy Scores: {cv_scores}")
print(f"Mean Cross-validation Accuracy: {cv_scores.mean()}")

# Hyperparameter tuning with Grid Search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.05, 0.1]
}

grid_search = GridSearchCV(estimator=gb_classifier, param_grid=param_grid, cv=cv, scoring='accurac
grid_search.fit(X_train, y_train)

print(f"Best Parameters from Grid Search: {grid_search.best_params_}")
best_model = grid_search.best_estimator_

# Evaluate the best model
y_pred_best = best_model.predict(X_test)
best_accuracy = accuracy_score(y_test, y_pred_best)
best_f1 = f1_score(y_test, y_pred_best, average='weighted')
print(f"Test Accuracy (Best Model): {best_accuracy}")
print(f"Test F1 Score (Best Model): {best_f1}")
```

```
Training Accuracy: 0.9928688974218322
Test Accuracy: 0.9890350877192983
Training F1 Score: 0.9928662388650514
Test F1 Score: 0.9890660418007817
Cross-validation Accuracy Scores: [0.98903509 0.98738343 0.98902907 0.98793198 0.98957762]
Mean Cross-validation Accuracy: 0.9885914388274581
Best Parameters from Grid Search: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300
Test Accuracy (Best Model): 0.9884868421052632
Test F1 Score (Best Model): 0.9885068151561498
```

```
cv_scores = cross_val_score(gb_classifier, df[features], df[target_match_result], cv=cv, scoring='


print(f"Cross-validation Accuracy Scores: {cv_scores}")
print(f"Mean Cross-validation Accuracy: {cv_scores.mean()}")
```

```
Cross-validation Accuracy Scores: [0.98903509 0.98738343 0.98902907 0.98793198 0.98957762]
Mean Cross-validation Accuracy: 0.9885914388274581
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

*Later tests *

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer


# Feature engineering
df['TotalGoals'] = df['HomeGoals'] + df['AwayGoals']
df['Over2.5'] = (df['TotalGoals'] > 2.5).astype(int)
df['Result'] = df.apply(lambda row: 'Win' if row['HomeGoals'] > row['AwayGoals'] else ('Draw' if r

# Select features and target
features = ['Date', 'HomeTeam', 'AwayTeam','HomeOdds', 'DrawOdds', 'AwayOdds']
Result = 'Result'
target_over_2_5 = 'Over2.5'

# Scaling features
scaler = StandardScaler()
df[features] = scaler.fit_transform(df['HomeOdds', 'DrawOdds', 'AwayOdds'])

# Split the data
X_train, X_test, y_train_Result, y_test_Result = train_test_split(df[features], df[Result], test_s
_, _, y_train_over_2_5, y_test_over_2_5 = train_test_split(df[features], df[target_over_2_5], test
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: ('HomeOdds', 'DrawOdds', 'AwayOdds')

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                  2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
```