```python
import pandas as pd
import numpy as np
from sklearn.model_selection import TimeSeriesSplit, cross_val_score
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import lightgbm as lgb

# 1. Load Data
train = pd.read_csv('TRAIN.csv')
test = pd.read_csv('Test_final.csv')

train['Date'] = pd.to_datetime(train['Date'],format='%d-%m-%Y')
test['Date'] = pd.to_datetime(test['Date'],format='%Y-%m-%d')

for df in [train, test]:
    df['Year'] = df['Date'].dt.year
    df['Month'] = df['Date'].dt.month
    df['Day'] = df['Date'].dt.day
    df['DayOfWeek'] = df['Date'].dt.dayofweek
    df['IsWeekend'] = df['DayOfWeek'].isin([5,6]).astype(int)


# Encode categorical variables
cat_cols = ['Store_Type', 'Location_Type', 'Region_Code', 'Discount']
encoder = LabelEncoder()
for col in cat_cols:
    train[col] = encoder.fit_transform(train[col])
    test[col] = encoder.transform(test[col])

# Features and Target
X = train.drop(columns=['ID', 'Actual_Sales', 'Date','Data_type'])
y = train['Actual_Sales']
X_test = test.drop(columns=['ID', 'Date'])

lr = LinearRegression()


# -------------------------------
# 2. Custom Metrics
# -------------------------------
def safe_mape(y_true, y_pred):
    """Ignore zero actuals to avoid division by zero"""
    mask = y_true != 0
    if mask.sum() == 0:  # if all values are 0
        return np.nan
    return np.mean(np.abs((y_true[mask] - y_pred[mask]) /
y_true[mask])) * 100

def smape(y_true, y_pred):
    """Symmetric MAPE (bounded, stable)"""
    return 100 * np.mean(
        2 * np.abs(y_pred - y_true) / (np.abs(y_true) + np.abs(y_pred)
+ 1e-8)
    )

def wmape(y_true, y_pred):
    """Weighted MAPE (best for sales forecasting)"""
    return 100 * np.sum(np.abs(y_true - y_pred)) /
np.sum(np.abs(y_true))

# -------------------------------
# 3. Evaluation Function
# -------------------------------
def evaluate_model(model, X, y):
    tscv = TimeSeriesSplit(n_splits=5)
    rmse_scores, mae_scores, mape_scores, smape_scores,
wmape_scores,mse_scores = [], [], [], [], [],[]

    for train_idx, val_idx in tscv.split(X):
        X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_val)

        mse_scores.append(mean_squared_error(y_val, y_pred))
        rmse_scores.append(np.sqrt(mean_squared_error(y_val, y_pred)))
        mae_scores.append(mean_absolute_error(y_val, y_pred))
        mape_scores.append(safe_mape(y_val, y_pred))
        smape_scores.append(smape(y_val, y_pred))
        wmape_scores.append(wmape(y_val, y_pred))

    model_results = {
        "Model": model.__class__.__name__,
        "MSE":   np.mean(mse_scores),
        "RMSE":  np.mean(rmse_scores),
        "MAE":   np.mean(mae_scores),
        "MAPE":  np.nanmean(mape_scores),
        "sMAPE": np.mean(smape_scores),
        "wMAPE": np.mean(wmape_scores)}

    # □ Print
    print(f"Model: {model_results['Model']}")
    print(f"MSE:   {model_results['MSE']:.2f}")
    print(f"RMSE:  {model_results['RMSE']:.2f}")
    print(f"MAE:   {model_results['MAE']:.2f}")
    print(f"MAPE (safe):  {model_results['MAPE']:.2f}%")
    print(f"sMAPE: {model_results['sMAPE']:.2f}%")
    print(f"wMAPE: {model_results['wMAPE']:.2f}%")
    print("-"*40)

    # □ Save to CSV
    pd.DataFrame([model_results]).to_csv('model_metrics.csv',
index=False)
# --------------------------------
# 4. Run Evaluation
# --------------------------------
evaluate_model(lr, X, y)
```

```
Model: LinearRegression
MSE:   47558652.04
RMSE:  6580.21
MAE:   5145.57
MAPE (safe):  12.76%
sMAPE: 13.58%
wMAPE: 11.94%
----------------------------------------
```

```
lgb_model = lgb.LGBMRegressor(
    n_estimators=500,
    learning_rate=0.05,
    max_depth=10,
    num_leaves=31,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)

# --------------------------------
# 2. Custom Metrics
# --------------------------------
def safe_mape(y_true, y_pred):
    mask = y_true != 0
    if mask.sum() == 0:
        return np.nan
    return np.mean(np.abs((y_true[mask] - y_pred[mask]) /
y_true[mask])) * 100

def smape(y_true, y_pred):
    return 100 * np.mean(
        2 * np.abs(y_pred - y_true) / (np.abs(y_true) + np.abs(y_pred)
+ 1e-8)
    )

def wmape(y_true, y_pred):
    return 100 * np.sum(np.abs(y_true - y_pred)) /
np.sum(np.abs(y_true))

# --------------------------------
# 3. Evaluation Function
# --------------------------------
def evaluate_model(model, X, y, save_path="metrics.csv"):
    tscv = TimeSeriesSplit(n_splits=5)
    lg_mse_scores, lg_rmse_scores, lg_mae_scores, lg_mape_scores,
lg_smape_scores, lg_wmape_scores = [], [], [], [], [], []

    for train_idx, val_idx in tscv.split(X):
        X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_val)

        lg_mse_scores.append(mean_squared_error(y_val, y_pred))
        lg_rmse_scores.append(np.sqrt(mean_squared_error(y_val,
y_pred)))
        lg_mae_scores.append(mean_absolute_error(y_val, y_pred))
        lg_mape_scores.append(safe_mape(y_val, y_pred))
        lg_smape_scores.append(smape(y_val, y_pred))
        lg_wmape_scores.append(wmape(y_val, y_pred))

    # □ Collect results
    results = {
        "Model": model.__class__.__name__,
        "MSE":   np.mean(lg_mse_scores),
        "RMSE":  np.mean(lg_rmse_scores),
        "MAE":   np.mean(lg_mae_scores),
        "MAPE":  np.nanmean(lg_mape_scores),
        "sMAPE": np.mean(lg_smape_scores),
        "wMAPE": np.mean(lg_wmape_scores)
    }

    # □ Print
    print(f"Model: {results['Model']}")
    print(f"MSE:   {results['MSE']:.2f}")
    print(f"RMSE:  {results['RMSE']:.2f}")
    print(f"MAE:   {results['MAE']:.2f}")
    print(f"MAPE (safe):  {results['MAPE']:.2f}%")
    print(f"sMAPE: {results['sMAPE']:.2f}%")
    print(f"wMAPE: {results['wMAPE']:.2f}%")
    print("-"*40)

    # □ Save to CSV
    pd.DataFrame([results]).to_csv(save_path, index=False)
    print(f"□ Metrics saved to {save_path}")

# --------------------------------
# 4. Run Evaluation
# ------------------------------
evaluate_model(lgb_model, X, y)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.001564 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 528
[LightGBM] [Info] Number of data points in the train set: 31390,
number of used features: 11
[LightGBM] [Info] Start training from score 42597.300640
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.002579 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 543
[LightGBM] [Info] Number of data points in the train set: 62780,
number of used features: 11
[LightGBM] [Info] Start training from score 43039.218252
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Warning] No further splits with positive gain, best gain:
-inf
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.003548 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 573
[LightGBM] [Info] Number of data points in the train set: 94170,
number of used features: 11
[LightGBM] [Info] Start training from score 43205.629550
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.006272 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 578
[LightGBM] [Info] Number of data points in the train set: 125560,
number of used features: 11
[LightGBM] [Info] Start training from score 42241.985609
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.006907 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 584
[LightGBM] [Info] Number of data points in the train set: 156950,
number of used features: 12
[LightGBM] [Info] Start training from score 42607.227616
Model: LGBMRegressor
MSE:   17658368.97
RMSE:  3999.36
```

```
MAE:   2974.33
MAPE (safe):  7.28%
sMAPE: 7.18%
wMAPE: 6.89%
----------------------------------------
□ Metrics saved to metrics.csv
```

```python
best_model = lgb_model
best_model.fit(X, y)

# Align columns in test
X_test = X_test.reindex(columns=X.columns, fill_value=0)

# □ Predict
test['Sales'] = best_model.predict(X_test)

test.to_csv('Visualization_test')
train.to_csv('visualization_train')
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.011211 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 584
[LightGBM] [Info] Number of data points in the train set: 188340,
number of used features: 12
[LightGBM] [Info] Start training from score 42784.327981
```

```python
import joblib
joblib.dump(best_model, "sales_forecast_model.pkl")

print("Min Sales:", train["Actual_Sales"].min())
print("Max Sales:", train["Actual_Sales"].max())
print("Mean Sales:", train["Actual_Sales"].mean())
```

```
Min Sales: 0.0
Max Sales: 247215.0
Mean Sales: 42784.327981522765
```