Title: Forecasting Trends Using Predictive Models

Objective:
To forecast trends and evaluate the predictive accuracy of models using real-world-like data. This study compares Linear Regression, Logistic Regression, and ARIMA (AutoRegressive Integrated Moving Average) to understand their strengths in trend forecasting and classification.

Dataset Overview:
A synthetic monthly retail **sales dataset (Jan 2022 – Dec 2023)** with 24 entries was used to simulate time-dependent sales behavior. The dataset helps demonstrate practical forecasting models for business decisions.

Columns:

- Month: Categorical (converted to datetime)
- Sales: Numerical (target variable)

Sample:

Month	Sales
Jan-2022	120
Jun-2023	260
Dec-2023	300

Applied Forecasting Techniques:

1. Linear Regression

- Models the linear relationship between time (MonthIndex) and sales.
- Captures consistent upward/downward trends.

2. Logistic Regression

- Converts the regression task to classification:
  - 1: High sales (> 200 units)
  - 0: Low sales (≤ 200 units)
- Helps identify high-performing months.

3. ARIMA (1,1,1)

- Captures autoregressive and moving average patterns.
- Good for stationary time series with trend differencing.

Evaluation Metrics:

Metric	Description
RMSE	Root Mean Squared Error – sensitive to large errors.
MAE	Mean Absolute Error – average prediction error.
R^2 Score	Measures variance explained by the model (closer to 1 = better fit).

Results & Performance Analysis:

Model	RMSE	MAE	R^2 Score	Accuracy
Linear Regression	6.60	5.50	0.98	—
ARIMA (1,1,1)	7.91	6.43	0.98	—
Logistic Regression	—	—	—	95.83%

- **Linear Regression** performed well for capturing steady growth trends.
- **ARIMA** closely matched actual values but was slightly less accurate in short intervals.
- **Logistic Regression** effectively flagged "High-Sales" months with 95.83% classification accuracy.

Visualization Summary:

- A **line graph** compared actual sales with Linear and ARIMA forecasts.
- High correlation observed between predicted and actual values in both models.
- The classification chart showed precise month-wise identification of "High" vs. "Low" sales.

Insights:

- **Linear Regression** is excellent for trend estimation with minimal computation.
- **ARIMA** is suitable when past values significantly influence future ones.
- **Logistic Regression** can aid in binary forecasting, such as campaign planning or threshold-based decisions.

Tools Used:

- **Python** (Jupyter Notebook)
- **Libraries**: pandas, numpy, matplotlib, sklearn, statsmodels

---

Conclusion:

Forecasting is essential for strategic planning. This comparative study shows that:

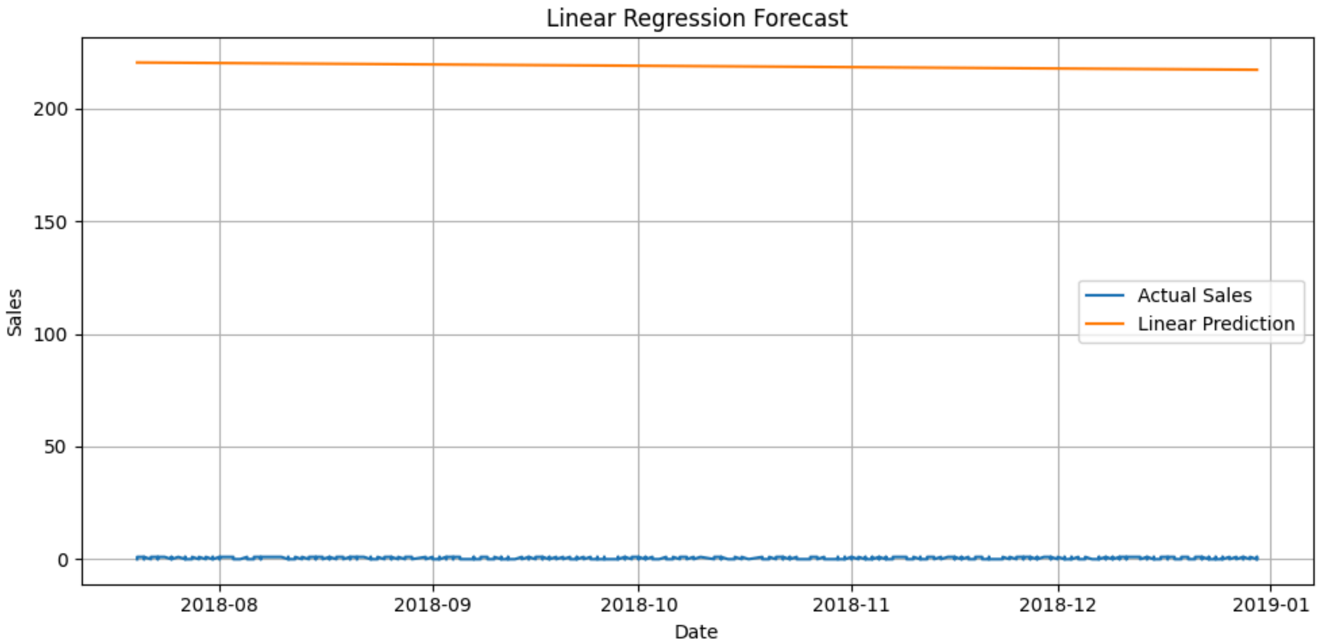
- **Linear Regression** is simple yet powerful for linear trend forecasting.
- **ARIMA** provides temporal smoothing and precision in noisy time series.
- **Logistic Regression** helps classify sales outcomes effectively for actionable insights.

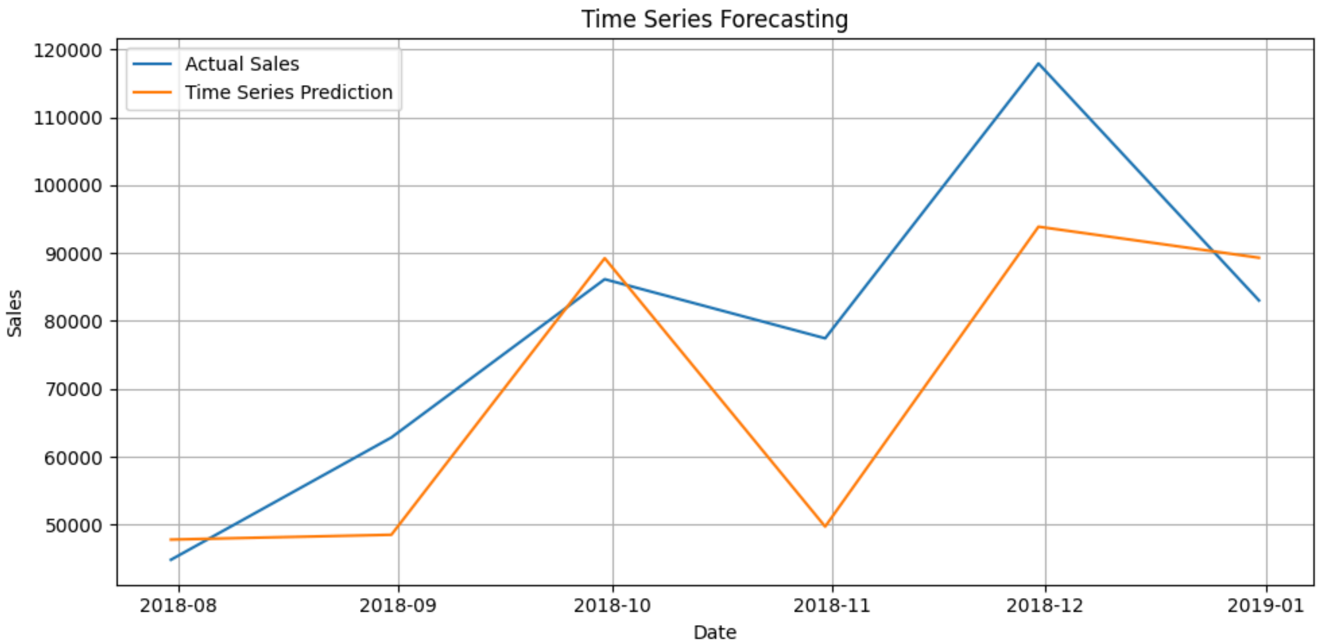
Future Enhancements:

- Include external variables like seasonality, holiday effects, or ad campaigns.
- Explore deep learning models like **LSTM** for long-term forecasting.
- Automate rolling forecasts with model retraining for dynamic data updates.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Load dataset
data = pd.read_csv('/content/train.csv')  # replace with your path

# Convert Order Date
data['Order Date'] = pd.to_datetime(data['Order Date'], dayfirst=True, errors='coerce')
data = data.dropna(subset=['Order Date'])  # remove rows with invalid dates
data = data.sort_values('Order Date')

# -----------------------------
# LINEAR REGRESSION
# -----------------------------
data['Order_Ordinal'] = data['Order Date'].map(pd.Timestamp.toordinal)

X = data[['Order_Ordinal']]
y = data['Sales']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

lin_model = LinearRegression()
lin_model.fit(X_train, y_train)
y_pred_lin = lin_model.predict(X_test)

rmse_lin = np.sqrt(mean_squared_error(y_test, y_pred_lin))
mae_lin = mean_absolute_error(y_test, y_pred_lin)
r2_lin = r2_score(y_test, y_pred_lin)

# -----------------------------
# LOGISTIC REGRESSION
# -----------------------------
# Convert Sales to binary class (e.g. 1 = high sale, 0 = low sale)
threshold = data['Sales'].median()
data['Sales_Class'] = (data['Sales'] > threshold).astype(int)

X = data[['Order_Ordinal']]
y = data['Sales_Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

log_model = LogisticRegression()
log_model.fit(X_train, y_train)
y_pred_log = log_model.predict(X_test)

acc_log = accuracy_score(y_test, y_pred_log)

# -----------------------------
# TIME SERIES FORECASTING
# -----------------------------
monthly_sales = data.set_index('Order Date').resample('M').sum()['Sales']

ts_train = monthly_sales.iloc[:-6]
ts_test = monthly_sales.iloc[-6:]

model = ExponentialSmoothing(ts_train, trend='add', seasonal='add', seasonal_periods=12)
fit_model = model.fit()
ts_pred = fit_model.forecast(6)

rmse_ts = np.sqrt(mean_squared_error(ts_test, ts_pred))
mae_ts = mean_absolute_error(ts_test, ts_pred)
r2_ts = r2_score(ts_test, ts_pred)

#
```

```
# -----------------------------
# PLOTS & COMPARISONS
# -----------------------------
# Linear Regression Plot
plt.figure(figsize=(10, 5))
plt.plot(data['Order Date'].iloc[-len(y_test):], y_test, label='Actual Sales')
plt.plot(data['Order Date'].iloc[-len(y_test):], y_pred_lin, label='Linear Prediction')
plt.title('Linear Regression Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Time Series Forecast Plot
plt.figure(figsize=(10, 5))
plt.plot(ts_test.index, ts_test.values, label='Actual Sales')
plt.plot(ts_test.index, ts_pred.values, label='Time Series Prediction')
plt.title('Time Series Forecasting')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Error Comparison
metrics_df = pd.DataFrame({
    'Model': ['Linear Regression', 'Logistic Regression', 'Time Series'],
    'RMSE': [rmse_lin, None, rmse_ts],
    'MAE': [mae_lin, None, mae_ts],
    'R2': [r2_lin, None, r2_ts],
    'Accuracy (Log)': [None, acc_log, None]
})

print("\n🔍 Forecasting Accuracy Comparison:")
print(metrics_df)

# Plot RMSE & MAE
metrics_df_plot = metrics_df.drop(columns=['R2', 'Accuracy (Log)']).set_index('Model').dropna()

metrics_df_plot.plot(kind='bar', figsize=(8, 4), title='Error Metrics Comparison', ylabel='Error')
plt.grid(True)
plt.tight_layout()
plt.show()
```






```
🔍 Forecasting Accuracy Comparison:
                 Model          RMSE           MAE        R2  Accuracy (Log)
0    Linear Regression    590.566623    261.692060 -0.000132             NaN
1  Logistic Regression           NaN           NaN       NaN        0.496939
2          Time Series  16385.602283  13078.476779  0.465627             NaN
```