Analytical Report

Analytical Report

1. Linear Regression

- Coefficients: [98.42373844 82.92962217 25.84208744]

- Intercept: -0.45

- MSE: 102.91

2. Logistic Regression

- Coefficients: [[-1.49386917 0.26932329 1.95268358 0.11398442]]

- Intercept: [1.21889719]

- Accuracy: 0.88

3. Time Series Analysis

- AIC: 5203.20

- AR1: -0.08, MA1: -0.72

4. Clustering

K-means

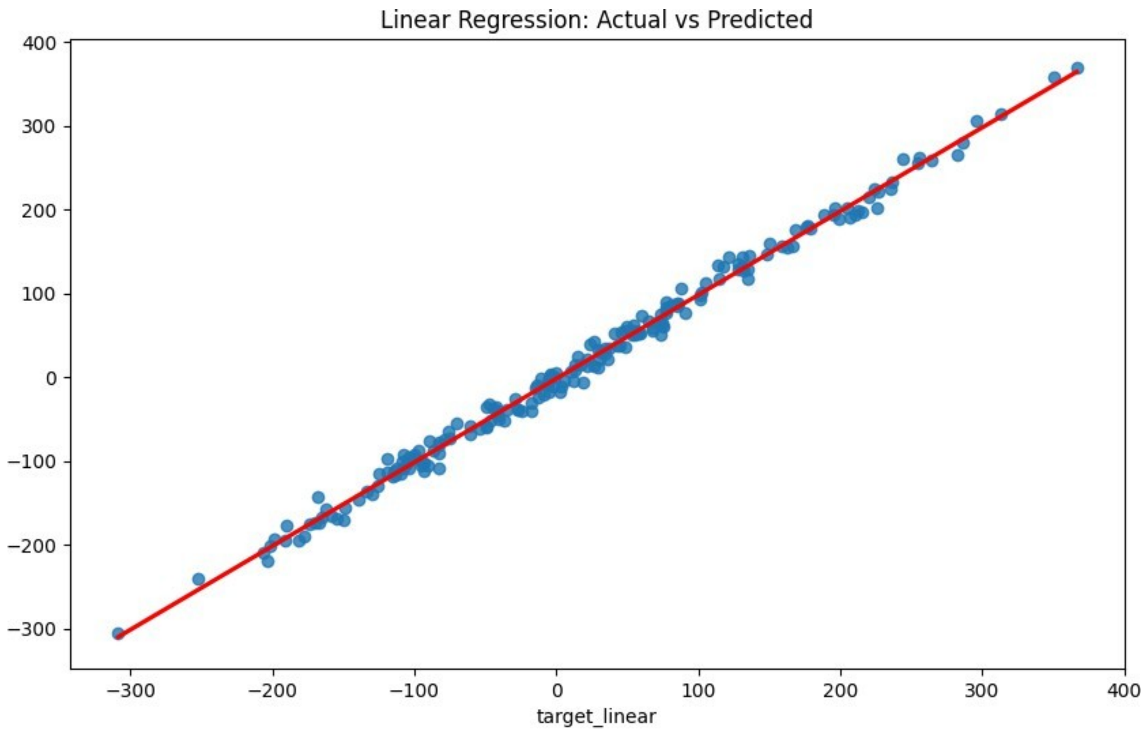
- Silhouette Score: 0.76

Hierarchical

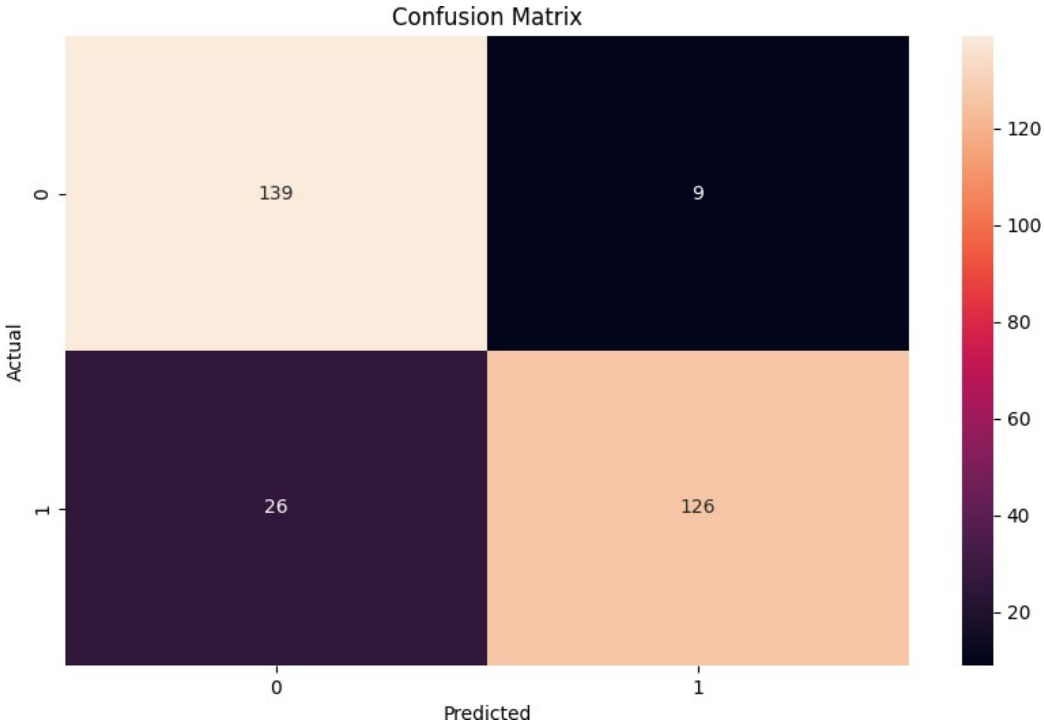
Analytical Report

- Silhouette Score: 0.76

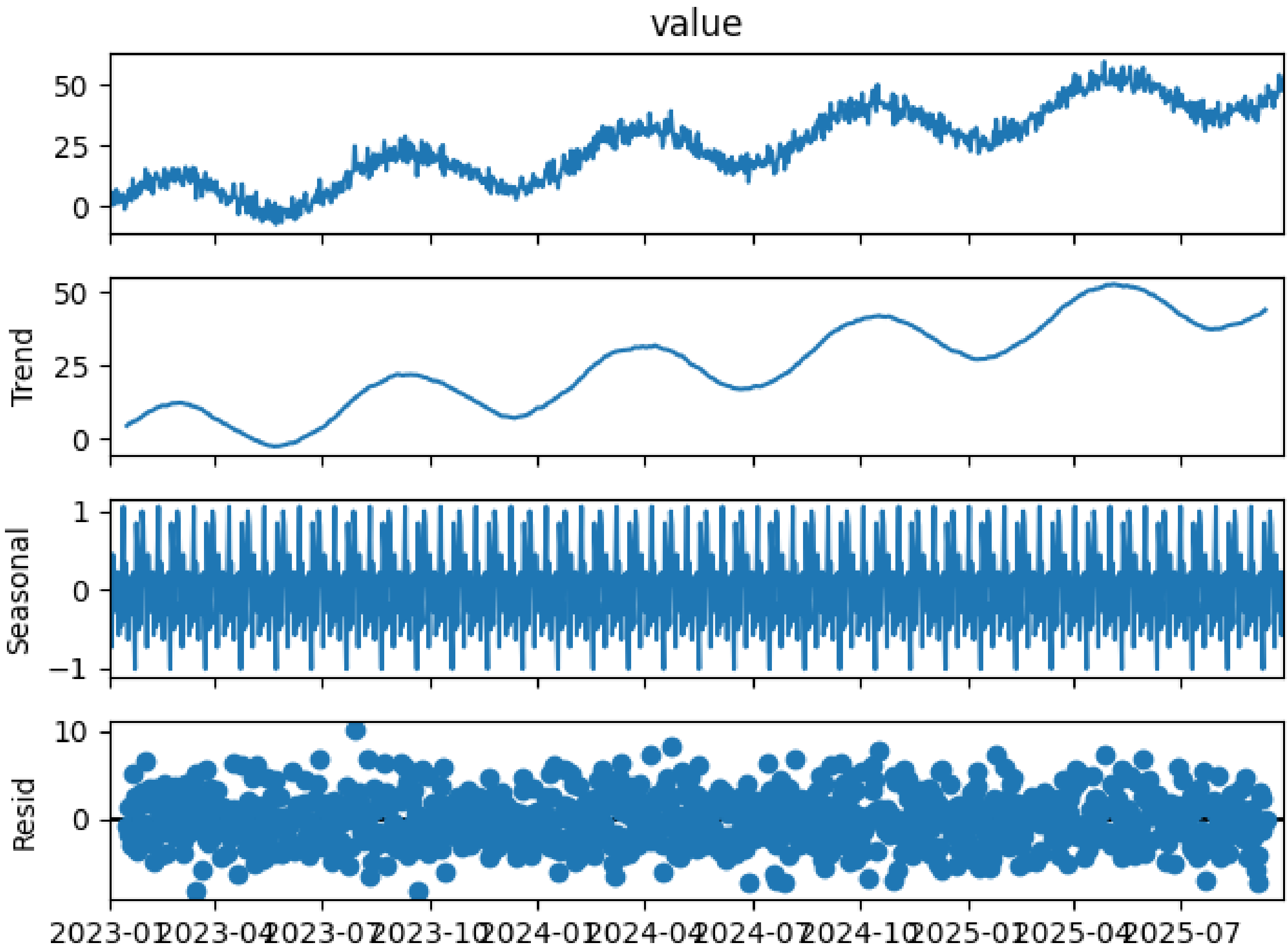
Analytical Report



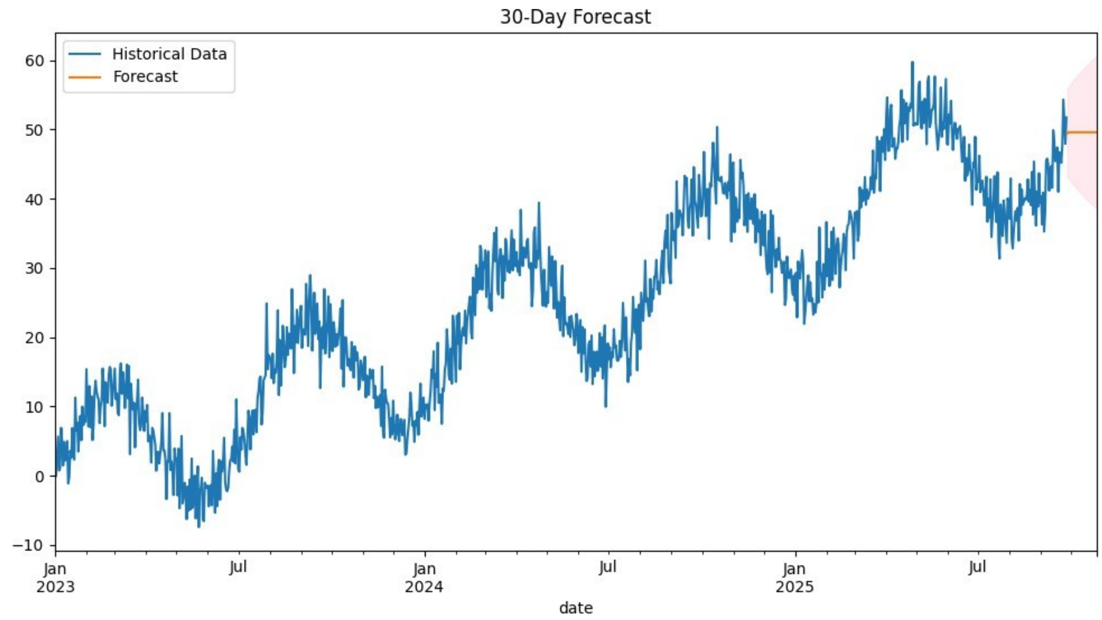
Analytical Report



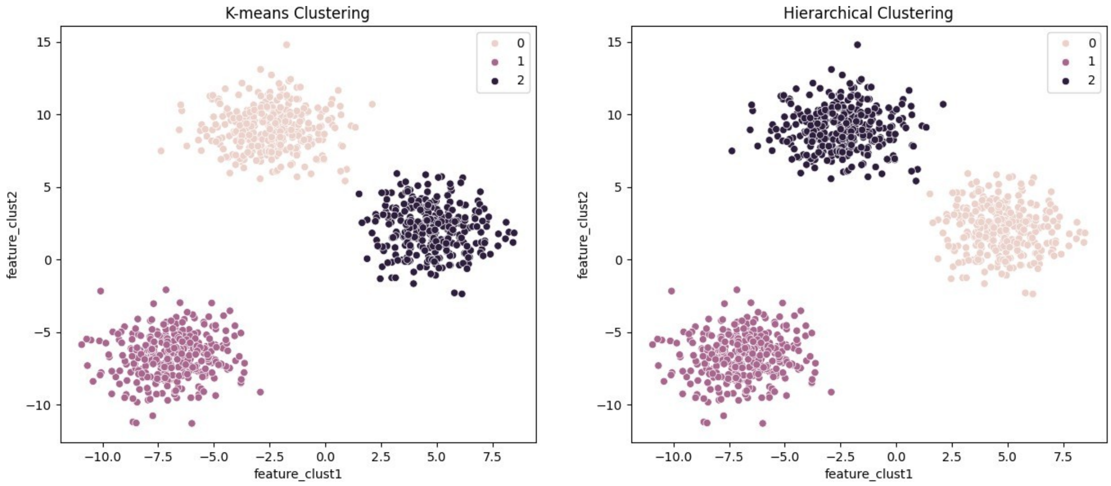
Analytical Report



Analytical Report



Analytical Report



Analytical Report

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import (mean_squared_error, accuracy_score,
                             silhouette_score, adjusted_rand_score)
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy.cluster.hierarchy import dendrogram, linkage
from fpdf import FPDF
from PIL import Image
import os
from datetime import datetime, timedelta
from sklearn.datasets import make_classification, make_regression, make_blobs
import warnings

warnings.filterwarnings('ignore')

# 1. Generate dataset
np.random.seed(42)
n_samples = 1000

# Linear regression data
X_lin, y_lin = make_regression(n_samples=n_samples, n_features=3, noise=10, random_state=42)
df_lin = pd.DataFrame(X_lin, columns=['feature_lin1', 'feature_lin2', 'feature_lin3'])
df_lin['target_linear'] = y_lin

# Logistic regression data
X_log, y_log = make_classification(n_samples=n_samples, n_features=4, n_classes=2,
                                   n_clusters_per_class=1, random_state=42)
df_log = pd.DataFrame(X_log, columns=['feature_log1', 'feature_log2', 'feature_log3', 'feature_log4'])
df_log['target_logistic'] = y_log

# Time series data
dates = [datetime(2023, 1, 1) + timedelta(days=i) for i in range(n_samples)]
trend = np.linspace(0, 50, n_samples)
seasonality = 10 * np.sin(np.linspace(0, 10*np.pi, n_samples))
noise = np.random.normal(0, 3, n_samples)
time_series = trend + seasonality + noise
```

```python
df_time = pd.DataFrame({
    'date': dates,
    'value': time_series,
    'moving_avg': pd.Series(time_series).rolling(window=7).mean(),
    'lag1': pd.Series(time_series).shift(1)
})

# Clustering data
X_clust, y_clust = make_blobs(n_samples=n_samples, centers=3, n_features=2,
                              cluster_std=1.5, random_state=42)
df_clust = pd.DataFrame(X_clust, columns=['feature_clust1', 'feature_clust2'])
df_clust['cluster_true'] = y_clust

# Combine all data
df = pd.concat([df_lin, df_log, df_time, df_clust], axis=1)
df.to_csv('synthetic_dataset.csv', index=False)

# 2. Analysis
report_text = "# Analytical Report\n\n"

# Linear Regression
X_lin = df[['feature_lin1', 'feature_lin2', 'feature_lin3']]
y_lin = df['target_linear']
X_train, X_test, y_train, y_test = train_test_split(X_lin, y_lin, test_size=0.2, random_state=42)

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

report_text += "## 1. Linear Regression\n\n"
report_text += f"- Coefficients: {lin_reg.coef_}\n"
report_text += f"- Intercept: {lin_reg.intercept_:.2f}\n"
report_text += f"- MSE: {mse:.2f}\n\n"

plt.figure(figsize=(10, 6))
sns.regplot(x=y_test, y=y_pred, line_kws={'color': 'red'})
plt.title('Linear Regression: Actual vs Predicted')
plt.savefig('linear_regression.png')
plt.close()

# Logistic Regression
X_log = df[['feature_log1', 'feature_log2', 'feature_log3', 'feature_log4']]
```

```
y_log = df['target_logistic']
X_train, X_test, y_train, y_test = train_test_split(X_log, y_log, test_size=0.3, random_state=42)

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

report_text += "## 2. Logistic Regression\n\n"
report_text += f"- Coefficients: {log_reg.coef_}\n"
report_text += f"- Intercept: {log_reg.intercept_}\n"
report_text += f"- Accuracy: {accuracy:.2f}\n\n"

conf_matrix = pd.crosstab(y_test, y_pred,
                          rownames=['Actual'],
                          colnames=['Predicted'])
plt.figure(figsize=(10, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.savefig('logistic_regression.png')
plt.close()

# Time Series Analysis
ts_data = df.set_index('date')['value']
decomposition = seasonal_decompose(ts_data, period=30)

plt.figure(figsize=(12, 8))
decomposition.plot()
plt.savefig('time_series_decomposition.png')
plt.close()

try:
    model = ARIMA(ts_data, order=(1, 1, 1))
    results = model.fit()
    report_text += "## 3. Time Series Analysis\n\n"
    report_text += f"- AIC: {results.aic:.2f}\n"
    report_text += f"- AR1: {results.arparams[0]:.2f}, MA1: {results.maparams[0]:.2f}\n\n"

    forecast = results.get_forecast(steps=30)
    plt.figure(figsize=(12, 6))
    ts_data.plot(label='Historical Data')
    forecast.predicted_mean.plot(label='Forecast')
    plt.fill_between(forecast.conf_int().index,
```

```python
# Clustering
X_clust = df[['feature_clust1', 'feature_clust2']]
true_labels = df['cluster_true']

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_clust)
silhouette_kmeans = silhouette_score(X_clust, kmeans_labels)

agg_clust = AgglomerativeClustering(n_clusters=3)
agg_labels = agg_clust.fit_predict(X_clust)
silhouette_agg = silhouette_score(X_clust, agg_labels)

report_text += "## 4. Clustering\n\n"
report_text += "### K-means\n"
report_text += f"- Silhouette Score: {silhouette_kmeans:.2f}\n\n"
report_text += "### Hierarchical\n"
report_text += f"- Silhouette Score: {silhouette_agg:.2f}\n\n"

plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
sns.scatterplot(data=df, x='feature_clust1', y='feature_clust2', hue=kmeans_labels)
plt.title('K-means Clustering')

plt.subplot(1, 2, 2)
sns.scatterplot(data=df, x='feature_clust1', y='feature_clust2', hue=agg_labels)
plt.title('Hierarchical Clustering')
plt.savefig('clustering_results.png')
plt.close()

# 3. Create PDF
class PDF(FPDF):
    def header(self):
        self.set_font('Arial', 'B', 12)
        self.cell(0, 10, 'Analytical Report', 0, 1, 'C')

    def footer(self):
        self.set_y(-15)
        self.set_font('Arial', 'I', 8)
        self.cell(0, 10, f'Page {self.page_no()}', 0, 0, 'C')
```

```
pdf = PDF()
pdf.add_page()
pdf.set_font('Arial', '', 12)
pdf.multi_cell(0, 10, report_text)

image_files = ['linear_regression.png', 'logistic_regression.png',
               'time_series_decomposition.png', 'time_series_forecast.png',
               'clustering_results.png']

for img in image_files:
    if os.path.exists(img):
        pdf.add_page()
        try:
            pdf.image(img, x=10, w=180)
        except:
            pdf.cell(0, 10, f"Failed to load image: {img}", 0, 1)

pdf.output('analytical_report.pdf')
print("Analysis complete! Created:")
print("- synthetic_dataset.csv")
print("- analytical_report.pdf")
print("- Analysis plots (.png)")
```

```
Analysis complete! Created:
- synthetic_dataset.csv
- analytical_report.pdf
- Analysis plots (.png)
<Figure size 1200x800 with 0 Axes>
```