NATIONAL UNIVERSITY OF MODERN LANGUAGES ISLAMABAD

Data Mining (LAB)

Assignment: 01

Submitted to
Dr. Moiz Ullah Ghouri

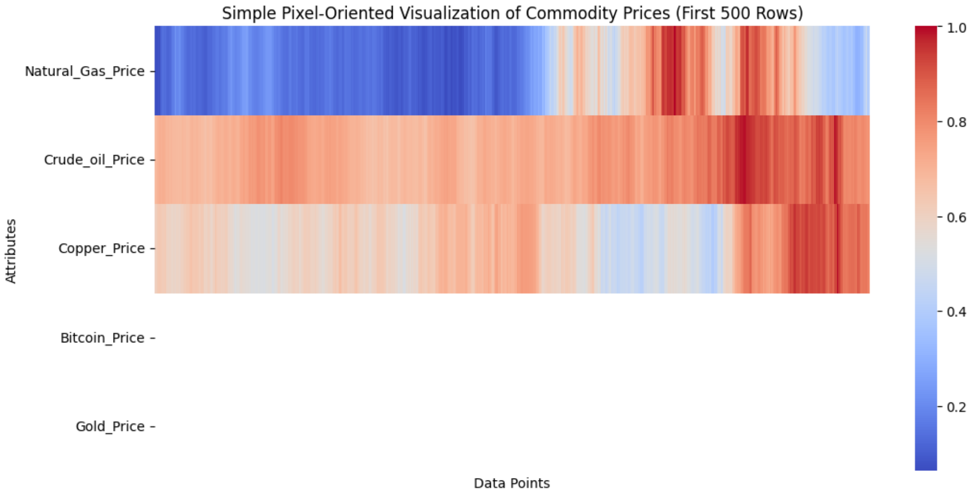
Submitted By
Junaid Asif
(BSAI-144)

Submission Date: October 04, 2024

Pixel-Oriented Visualization

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Load the dataset

data = pd.read_csv('Stock_Market_Dataset.csv')

data.sample(10)

data.columns


# Select relevant columns for visualization

subset_data = data[['Natural_Gas_Price', 'Crude_oil_Price', 'Copper_Price', 'Bitcoin_Price', 'Gold_Price']]


# Convert the selected columns to numeric (forcing errors to NaN)

subset_data = subset_data.apply(pd.to_numeric, errors='coerce')


# Handle NaN values (you can choose to drop them or fill them with a default value)

# Here, we'll fill NaN values with the column mean (you can also dropna if preferred)

subset_data = subset_data.fillna(subset_data.mean())


# Normalize the data for better pixel intensity visualization

normalized_data = (subset_data - subset_data.min()) / (subset_data.max() - subset_data.min())


# Take a sample of the first 500 rows for visualization (adjust as needed)
```

```
normalized_data = normalized_data.head(500)

# Set up the plot for pixel-oriented visualization
plt.figure(figsize=(12, 6))

# Create a heatmap using seaborn
sns.heatmap(normalized_data.T, cmap="coolwarm", cbar=True,
xticklabels=False,
yticklabels=subset_data.columns)

# Add title and axis labels
plt.title('Simple Pixel-Oriented Visualization of Commodity Prices (First 500
Rows)')
plt.xlabel('Data Points')
plt.ylabel('Attributes')

# Show the plot
plt.show()
```

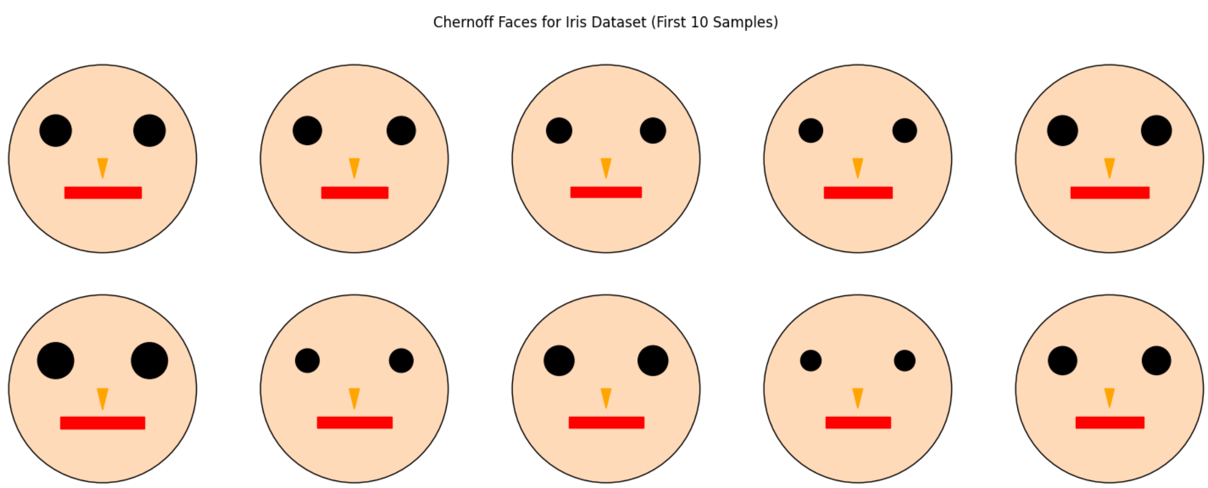


Chernoff Faces

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

# Load the dataset
data = pd.read_csv('iris_data.csv')

# Select relevant columns (for simplicity, using iris dataset features)
subset_data = data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]

# Normalize the data for better representation of facial features
normalized_data = (subset_data - subset_data.min()) / (subset_data.max() - subset_data.min())

# Function to draw a Chernoff face
def draw_face(ax, features):
    face_radius = 1
    # Draw the face (circle)
    face = plt.Circle((0, 0), face_radius, color='peachpuff', ec="black")
    ax.add_patch(face)

    # Eye parameters based on features
    eye_y = 0.3
    eye_x_offset = 0.5 * face_radius
    eye_radius = 0.1 + 0.3 * features[0]  # Sepal Length mapped to eye size
```

```
    # Draw eyes
    left_eye = plt.Circle((-eye_x_offset, eye_y), eye_radius, color='black')
    right_eye = plt.Circle((eye_x_offset, eye_y), eye_radius, color='black')
    ax.add_patch(left_eye)
    ax.add_patch(right_eye)

    # Mouth parameters based on features
    mouth_y = -0.3
    mouth_width = 0.5 + 0.5 * features[1]  # Sepal Width mapped to mouth width
    mouth_height = -0.1 - 0.2 * features[2]  # Petal Length mapped to mouth
curvature

    # Draw mouth
    mouth = plt.Rectangle((-mouth_width / 2, mouth_y), mouth_width,
mouth_height, color='red')
    ax.add_patch(mouth)

    # Nose size and position based on features
    nose_width = 0.1 + 0.1 * features[3]  # Petal Width mapped to nose size
    nose_height = 0.2 + 0.2 * features[3]

    # Draw nose
    nose = plt.Polygon([[-nose_width / 2, 0], [nose_width / 2, 0], [0, -
nose_height]], color='orange')
    ax.add_patch(nose)

# Plot Chernoff faces for the first 10 samples
fig, axs = plt.subplots(2, 5, figsize=(15, 6))
```

```
for i, ax in enumerate(axs.flatten()):
    features = normalized_data.iloc[i].values
    ax.set_xlim([-1.2, 1.2])
    ax.set_ylim([-1.2, 1.2])
    ax.set_aspect('equal')
    ax.axis('off')
    draw_face(ax, features)

plt.suptitle('Chernoff Faces for Iris Dataset (First 10 Samples)')
plt.tight_layout()
plt.show()
```



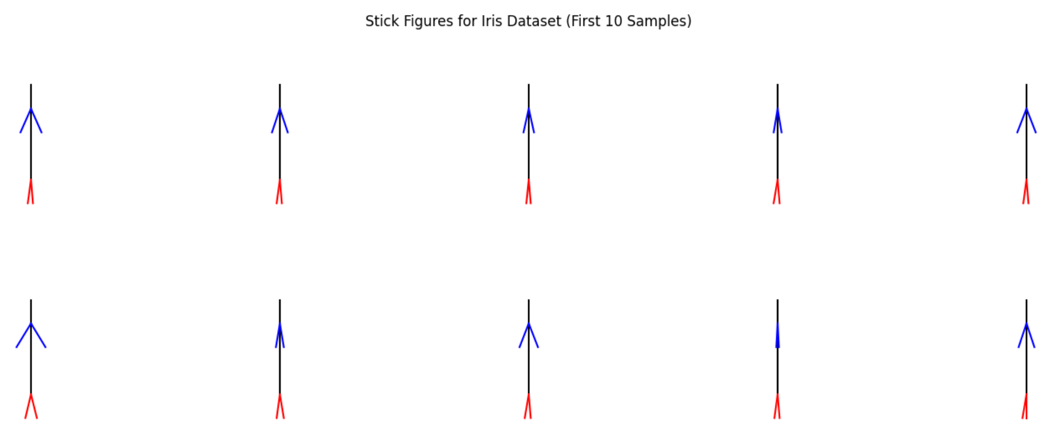
Stick Figures

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

```
# Load the dataset
data = pd.read_csv('iris_data.csv')

# Select relevant columns (for simplicity, using iris dataset features)
subset_data = data[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]

# Normalize the data for better representation
normalized_data = (subset_data - subset_data.min()) / (subset_data.max() -
subset_data.min())

# Function to draw a stick figure
def draw_stick_figure(ax, features):
    # Define the coordinates of the figure's body parts based on features
    # Head to foot is vertical line, arms and legs based on feature values
    head = (0, 1)
    body_top = (0, 0.8)
    body_bottom = (0, 0.2)

    # Arm positions based on the first and second feature
    left_arm = (-0.4 * features[0], 0.6)
    right_arm = (0.4 * features[0], 0.6)

    # Leg positions based on the third and fourth feature
    left_leg = (-0.4 * features[2], 0)
    right_leg = (0.4 * features[3], 0)

    # Plot head, body, arms, and legs
```

```
    ax.plot([head[0], body_top[0]], [head[1], body_top[1]], color='black')  # Head to body
    ax.plot([body_top[0], body_bottom[0]], [body_top[1], body_bottom[1]], color='black')  # Body

    # Arms
    ax.plot([body_top[0], left_arm[0]], [body_top[1], left_arm[1]], color='blue')  # Left arm
    ax.plot([body_top[0], right_arm[0]], [body_top[1], right_arm[1]], color='blue')  # Right arm

    # Legs
    ax.plot([body_bottom[0], left_leg[0]], [body_bottom[1], left_leg[1]], color='red')  # Left leg
    ax.plot([body_bottom[0], right_leg[0]], [body_bottom[1], right_leg[1]], color='red')  # Right leg

    # Set limits and aspect ratio
    ax.set_xlim([-1, 1])
    ax.set_ylim([-0.5, 1.2])
    ax.set_aspect('equal')
    ax.axis('off')

# Plot stick figures for the first 10 rows
fig, axs = plt.subplots(2, 5, figsize=(15, 6))

for i, ax in enumerate(axs.flatten()):
    features = normalized_data.iloc[i].values
    draw_stick_figure(ax, features)
```

```
plt.suptitle('Stick Figures for Iris Dataset (First 10 Samples)')
plt.tight_layout()
plt.show()
```



Tree Map

```
import pandas as pd
import matplotlib.pyplot as plt
import squarify

# Load the dataset
data = pd.read_csv('sales_data.csv')

# Group data by product category and sum the amount
grouped_data = data.groupby('product_category')['amount'].sum().reset_index()

# Define sizes and labels for the tree map
sizes = grouped_data['amount']
labels = grouped_data['product_category']
```

```
# Create the tree map
plt.figure(figsize=(12, 8))
squarify.plot(sizes=sizes, label=labels, alpha=.8)

# Set title and display the plot
plt.title('Tree Map of Product Categories by Amount Sold')
plt.axis('off')
plt.show()
```