TUGAS 5

Administrasi Basis Data (A)

Program Studi Sistem Informasi

Disusun Oleh :

Jonathan Cristopher Jetro (10231047)

SOAL

Visualisasikan Database Sales dari dari github Tugas 5 dengan data yang dibuat sendiri menggunakan conda dan library tambahan seperti streamlit dll.

JAWABAN

Visualisasi ini dibangun menggunakan **Streamlit**, **Plotly**, dan **PostgreSQL**

1. Pendahuluan

Dashboard *Jets Sales Analytics* merupakan sistem visualisasi data penjualan yang dirancang untuk memberikan wawasan terkait pelanggan, produk, dan performa penjualan secara keseluruhan. Sistem ini mengintegrasikan data dari PostgreSQL, kemudian divisualisasikan menggunakan Streamlit serta beberapa library pendukung seperti Plotly, NumPy, dan Pandas.

Dashboard ini terdiri dari empat modul utama:

1. **Overview**
2. **Customer Analytics**
3. **Product Analytics**
4. **Sales Analytics**

Setiap modul menyediakan analisis yang relevan melalui grafik interaktif, table data, dan ringkasan metrik agar bisnis dapat memahami performa secara cepat dan tepat.

2. Overview Dashboard

Halaman Overview memberikan gambaran umum performa bisnis secara menyeluruh. Tujuannya adalah menampilkan ringkasan *key performance indicators* (KPI) dari seluruh dataset.

2.1 Metrik Utama

Empat indikator utama ditampilkan pada bagian header:

- **Total Customers**: 100 pelanggan
  Menunjukkan skala basis pelanggan yang telah masuk ke sistem.
- **Total Products**: 100 produk
  Menggambarkan variasi produk yang tersedia di dalam inventori.

- **Total Revenue**: Rp 17.002.181
  Merupakan total pendapatan akumulatif berdasarkan seluruh transaksi.
- **Total Orders**: 68 pesanan
  Menunjukkan jumlah transaksi yang telah terjadi sepanjang periode analisis.

2.2 Revenue Trend Over Time

Grafik garis menampilkan pendapatan harian sepanjang tahun 2025.
Wawasan yang dapat diperoleh:

- Fluktuasi penjualan terlihat signifikan dari bulan ke bulan.
- Terdapat beberapa puncak pendapatan yang besar (sekitar >700K), mengindikasikan adanya pesanan besar.

2.3 Customer Age Distribution

Tersaji dalam bentuk pie chart berdasarkan "Age Group".
Temuan:

- Kelompok usia **41–50 tahun** mendominasi persentase pelanggan (sekitar 33%).
- Kelompok **30–40 tahun** dan **50–60 tahun** juga cukup besar.
- Data menunjukkan mayoritas pelanggan berada di usia produktif.

2.4 Top 10 Best Selling Products

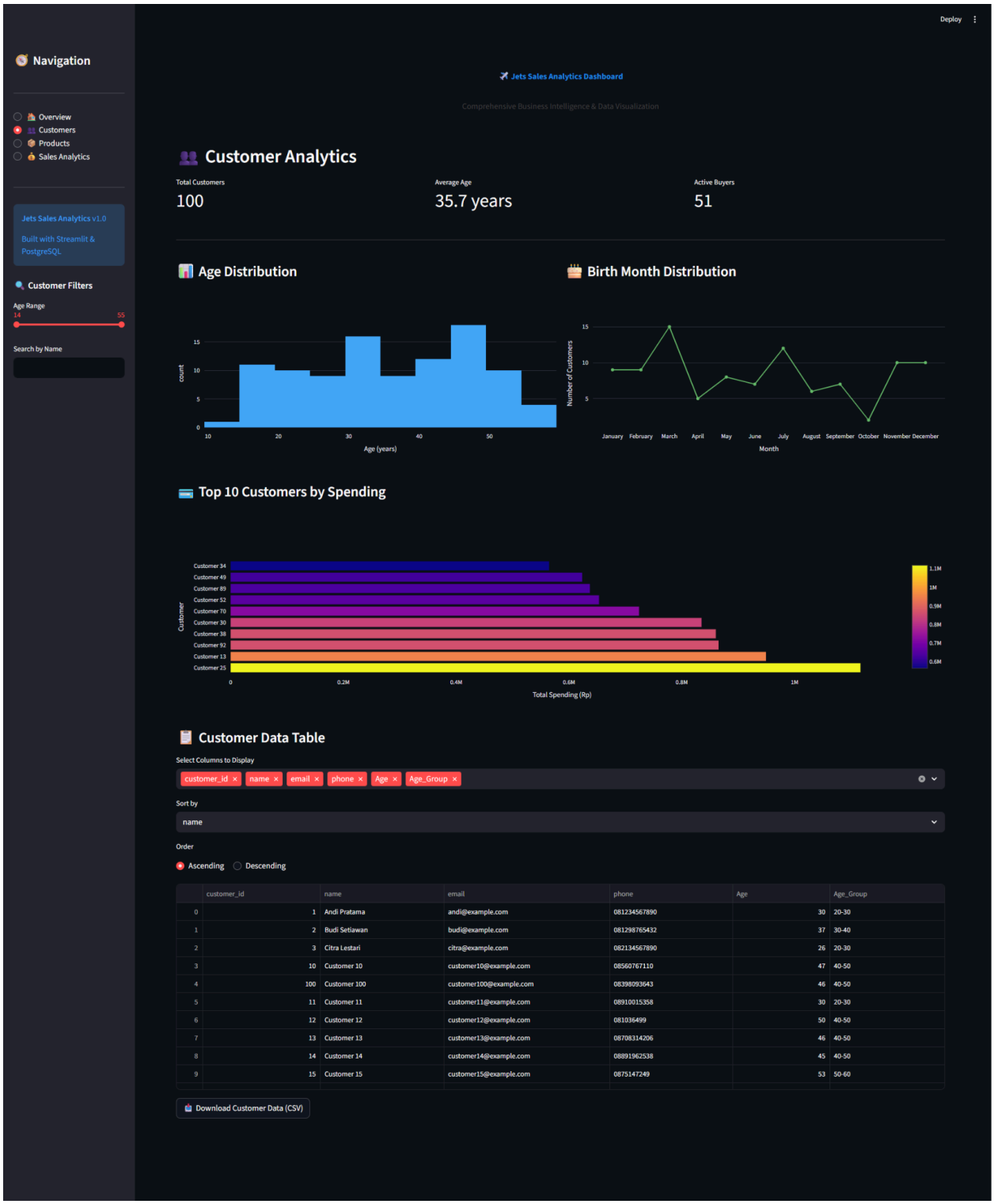
Grafik horizontal bar chart menampilkan produk dengan kuantitas terjual tertinggi.
Produk “Produk 86”, “Produk 100”, dan “Produk 58” berada pada posisi teratas.

2.5 Orders by Day of Week

Analisis frekuensi pesanan berdasarkan hari dalam seminggu.
Insight:

- **Hari Jumat** memiliki jumlah pesanan tertinggi.
- Pola ini menunjukkan adanya kebiasaan pelanggan melakukan pemesanan menjelang akhir pekan.

3. Customer Analytics



Halaman ini fokus pada analisis perilaku dan karakteristik pelanggan.

3.1 Summary Metrics

- **Total Customers**: 100
- **Average Age**: 35.7 years
- **Active Buyers**: 51 (pelanggan yang pernah melakukan minimal 1 order)

3.2 Age Distribution Chart

Histogram ini menunjukkan distribusi usia pelanggan.
Wawasan:

- Rentang usia 30–50 tahun mendominasi data.
- Konsentrasi besar pelanggan berada pada zona usia produktif.

3.3 Birth Month Distribution

Line chart untuk melihat kelahiran pelanggan berdasarkan bulan.
Insight:

- Puncak jumlah kelahiran terlihat pada beberapa bulan tertentu (misal Januari dan Mei).
- Dapat digunakan untuk kampanye ulang tahun pelanggan tiap bulan.

3.4 Top 10 Customers by Spending

Grafik horizontal bar menampilkan pelanggan dengan total belanja terbesar.

Dari grafik terlihat beberapa pelanggan menghabiskan lebih dari Rp 1 juta. Ini membantu bisnis mengidentifikasi “high value customers” untuk program loyalitas.

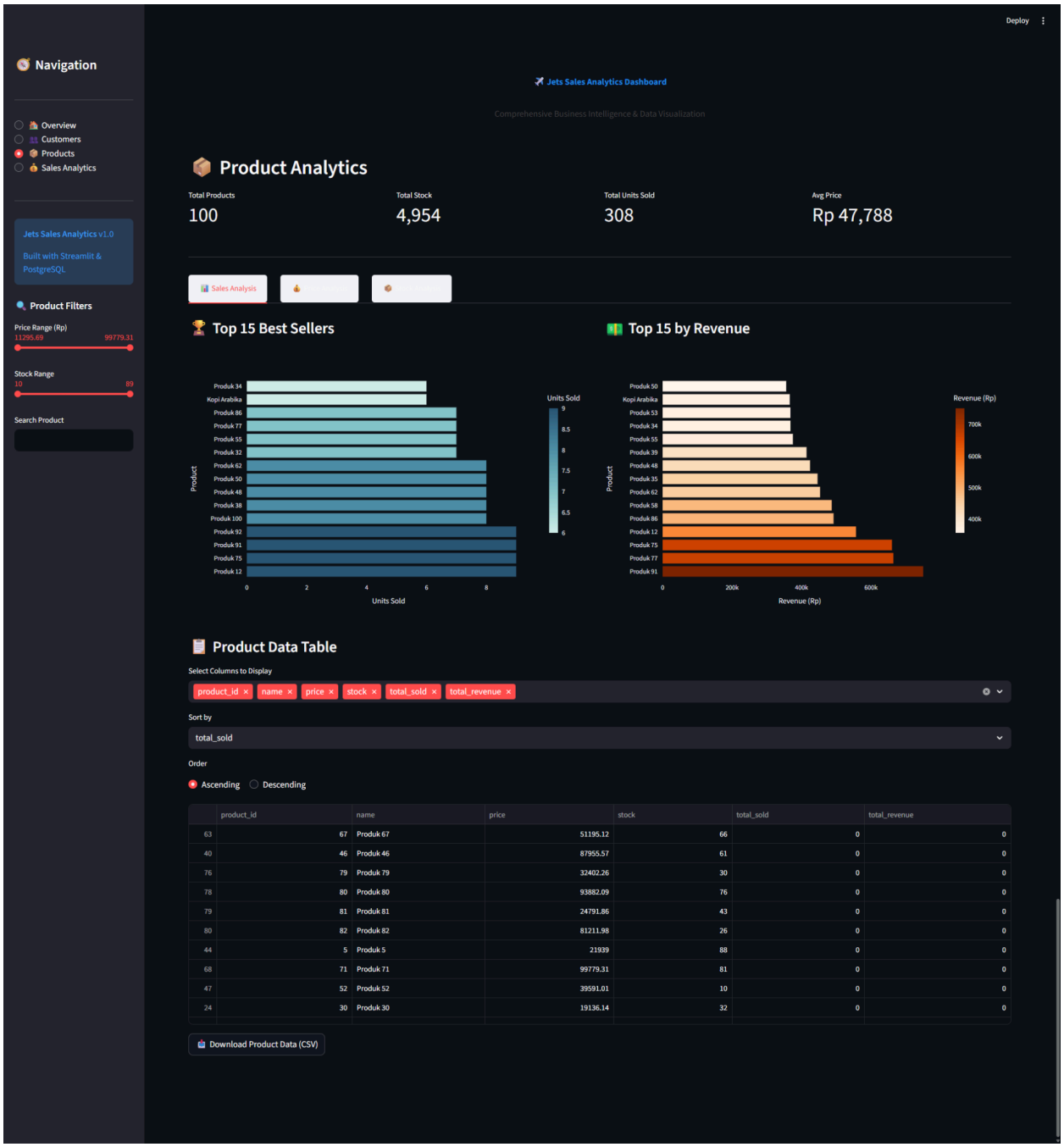
3.5 Customer Data Table

Tabel interaktif dengan berbagai fitur:

- Sorting
- Filtering berdasarkan usia
- Pencarian nama
- Download CSV

Memudahkan analisis lebih spesifik terkait karakter pelanggan.

4. Product Analytics



Halaman Product Analytics fokus pada performa produk dan inventori.

4.1 Summary Metrics

- **Total Products**: 100
- **Total Stock**: 4.954 unit
- **Total Units Sold**: 308 unit
- **Average Price**: Rp 47.788

4.2 Top 15 Best Sellers

Grafik horizontal bar: produk dengan **unit terjual terbanyak**.
Manfaat:

- Mengidentifikasi produk yang diminati.
- Membantu penentuan restock.

4.3 Top 15 Products by Revenue

Grafik horizontal bar menampilkan produk dengan total pendapatan tertinggi.
Insight:

- Produk dengan harga lebih tinggi bisa masuk top revenue walaupun kuantitas terjualnya lebih rendah.

4.4 Product Data Table

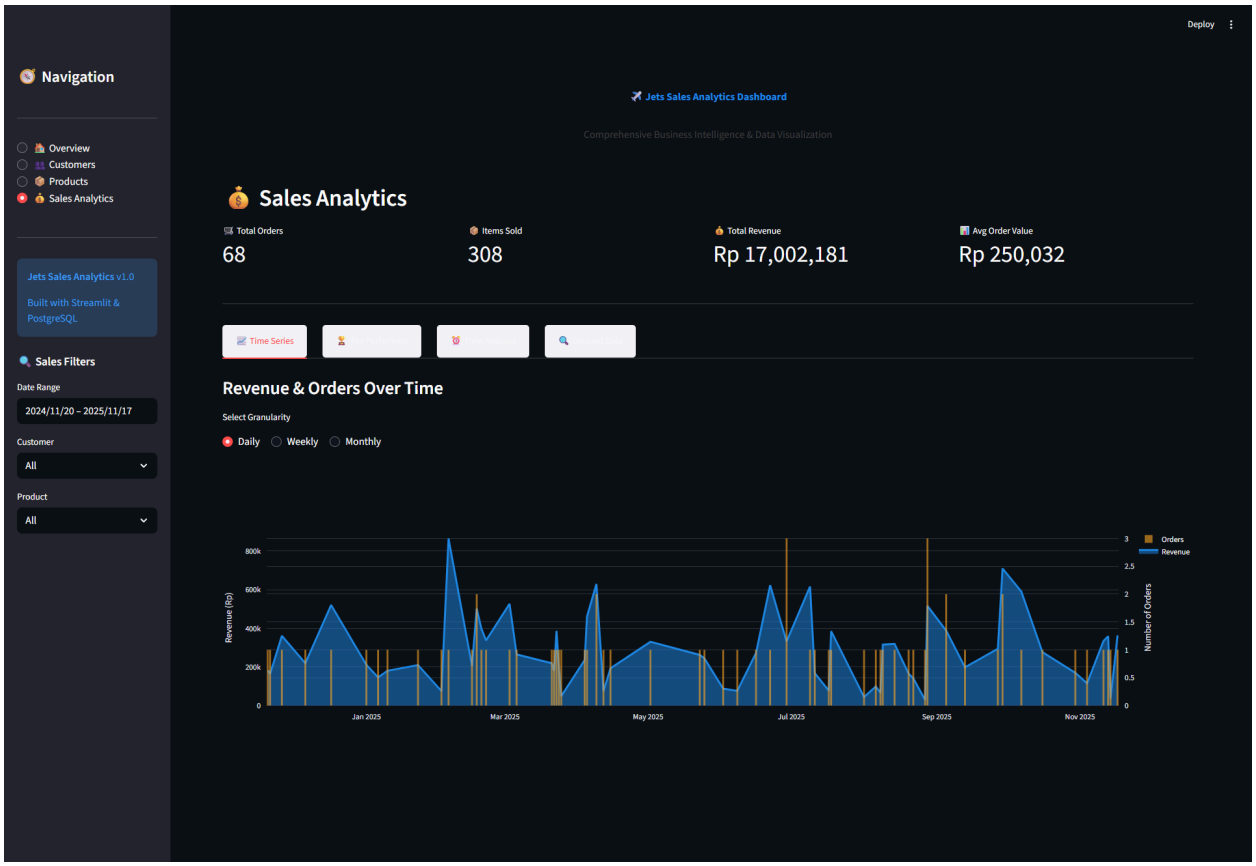
Tabel menampilkan:

- product_id
- name
- price
- stock
- total_sold
- total_revenue

Fitur:

- Sorting by specific metrics (misalnya total revenue)
- Download CSV
  Sangat berguna untuk analisis inventori dan pricing.

5. Sales Analytics



Halaman Sales Analytics merupakan modul yang paling fokus pada penjualan dan performa transaksi.

5.1 Summary Metrics

- **Total Orders**: 68
- **Items Sold**: 308
- **Total Revenue**: Rp 17.002.181
- **Average Order Value (AOV)**: Rp 250.032

5.2 Revenue & Orders Over Time

Chart ini adalah kombinasi:

- Grafik garis: Revenue
- Grafik batang: Jumlah order

Wawasan:

- Revenue tidak selalu linear dengan jumlah order (beberapa order bernilai sangat besar).
- Terlihat volatilitas penjualan harian cukup tinggi.

5.3 Sales Granularity Filter

Pengguna dapat memilih tampilan:

- Daily
- Weekly
- Monthly

Membantu memahami tren makro (monthly) dan mikro (daily).

5.4 Detailed Sales Table

Tersedia tabel penjualan yang dapat di-*export* sebagai CSV.
Data mencakup:

- order_id
- customer_id
- tanggal order
- total_amount
- jumlah item

Mendukung audit transaksi atau pengolahan lanjutan.

6. Kesimpulan

Dashboard Jets Sales Analytics berhasil menyajikan sistem visualisasi bisnis yang komprehensif dan informatif. Dengan pembagian laporan seperti Overview, Customer Analytics, Product Analytics, dan Sales Analytics, pengguna dapat:

- Memahami performa bisnis secara keseluruhan
- Menganalisis perilaku pelanggan
- Menilai performa produk
- Mengamati tren penjualan secara rinci

Sistem ini cocok digunakan untuk UMKM, toko online, hingga skala bisnis menengah yang memerlukan pemantauan penjualan real-time dan pengambilan keputusan berbasis data.

7. Lampiran Source Code

app.py

```
# Import library
import streamlit as st
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from datetime import datetime
import numpy as np

# Import fungsi dari config.py
from config import *

# ======================================================
```

```
# KONFIGURASI HALAMAN
# ========================================================
st.set_page_config("Jets Sales Analytics", page_icon="✈️",
layout="wide", initial_sidebar_state="expanded")

# Custom CSS untuk styling
st.markdown("""
<style>
    .main-header {
        font-size: 3rem;
        font-weight: bold;
        text-align: center;
        color: #1E88E5;
        padding: 1rem 0;
    }
    .sub-header {
        font-size: 1.5rem;
        color: #424242;
        text-align: center;
        padding-bottom: 2rem;
    }
    .metric-card {
        background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
        padding: 1rem;
        border-radius: 10px;
        color: white;
    }
    .stTabs [data-baseweb="tab-list"] {
        gap: 24px;
    }
    .stTabs [data-baseweb="tab"] {
        height: 50px;
        padding-left: 20px;
        padding-right: 20px;
        background-color: #f0f2f6;
        border-radius: 5px;
    }
</style>
""", unsafe_allow_html=True)
```

```
# ======================================================
# LOAD DATA DARI DATABASE
# ======================================================
@st.cache_data(ttl=300)
def load_data():
    """Load semua data dari database dengan caching"""
    # Data pelanggan
    result_customers = view_customers()
    df_customers = pd.DataFrame(result_customers, columns=[
        "customer_id", "name", "email", "phone", "address", "birthdate"
    ])

    # Data produk
    result_products = view_products()
    df_products = pd.DataFrame(result_products, columns=[
        "product_id", "name", "description", "price", "stock"
    ])

    # Data order details
    result_order_details = view_order_details_with_info()
    df_order_details = pd.DataFrame(result_order_details, columns=[
        "order_detail_id", "order_id", "order_date", "customer_id",
"customer_name",
        "product_id", "product_name", "unit_price", "quantity",
"subtotal",
        "order_total", "phone"
    ])

    return df_customers, df_products, df_order_details

df_customers, df_products, df_order_details = load_data()

# ======================================================
# PREPROCESSING DATA
# ======================================================
# Hitung usia dari birthdate
df_customers['birthdate'] = pd.to_datetime(df_customers['birthdate'])
df_customers['Age'] = (datetime.now() -
```

```
df_customers['birthdate']).dt.days // 365
df_customers['Age_Group'] = pd.cut(df_customers['Age'],
                                   bins=[0, 20, 30, 40, 50, 60, 100],
                                   labels=['<20', '20-30', '30-40',
'40-50', '50-60', '60+'])

# Proses order details
if not df_order_details.empty:
    df_order_details['order_date'] =
pd.to_datetime(df_order_details['order_date'])
    df_order_details['quantity'] =
pd.to_numeric(df_order_details['quantity'],
errors='coerce').fillna(0).astype(int)
    df_order_details['subtotal'] =
pd.to_numeric(df_order_details['subtotal'], errors='coerce').fillna(0)
    df_order_details['unit_price'] =
pd.to_numeric(df_order_details['unit_price'], errors='coerce').fillna(0)
    df_order_details['year'] = df_order_details['order_date'].dt.year
    df_order_details['month'] = df_order_details['order_date'].dt.month
    df_order_details['month_name'] =
df_order_details['order_date'].dt.strftime('%B')
    df_order_details['day_name'] =
df_order_details['order_date'].dt.strftime('%A')
    df_order_details['hour'] = df_order_details['order_date'].dt.hour

# ============================================================
# HEADER
# ============================================================
st.markdown('<p class="main-header">✈️ Jets Sales Analytics
Dashboard</p>', unsafe_allow_html=True)
st.markdown('<p class="sub-header">Comprehensive Business Intelligence &
Data Visualization</p>', unsafe_allow_html=True)

# ============================================================
# OVERVIEW DASHBOARD (Home)
# ============================================================
def show_overview():
    """Tampilan overview dengan KPI metrics"""
    st.header("📊 Business Overview")
```

```python
# KPI Metrics dalam kolom
col1, col2, col3, col4 = st.columns(4)

with col1:
    total_customers = len(df_customers)
    st.metric(
        label="👥 Total Customers",
        value=f"{total_customers:,}",
        delta="Active"
    )

with col2:
    total_products = len(df_products)
    st.metric(
        label="📦 Total Products",
        value=f"{total_products:,}",
        delta="In Stock"
    )

with col3:
    if not df_order_details.empty:
        total_revenue = df_order_details['subtotal'].sum()
        st.metric(
            label="💰 Total Revenue",
            value=f"Rp {total_revenue:,.0f}",
            delta="All Time"
        )
    else:
        st.metric(label="💰 Total Revenue", value="Rp 0")

with col4:
    if not df_order_details.empty:
        total_orders = df_order_details['order_id'].nunique()
        st.metric(
            label="🛒 Total Orders",
            value=f"{total_orders:,}",
            delta="Completed"
        )
```

```
        else:
            st.metric(label="🛒 Total Orders", value="0")

    st.markdown("---")

    # Grafik dalam dua kolom
    col_left, col_right = st.columns(2)

    with col_left:
        st.subheader("📈 Revenue Trend Over Time")
        if not df_order_details.empty:
            daily_revenue =
df_order_details.groupby(df_order_details['order_date'].dt.date)['subtot
al'].sum().reset_index()
            daily_revenue.columns = ['Date', 'Revenue']

            fig = px.area(daily_revenue, x='Date', y='Revenue',
                          title='Daily Revenue',
                          labels={'Revenue': 'Revenue (Rp)'},
                          color_discrete_sequence=['#1E88E5'])
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)
        else:
            st.info("No order data available")

    with col_right:
        st.subheader("👥 Customer Age Distribution")
        if not df_customers.empty:
            age_dist =
df_customers['Age_Group'].value_counts().sort_index()

            fig = px.pie(values=age_dist.values, names=age_dist.index,
                         title='Customer by Age Group',

color_discrete_sequence=px.colors.sequential.RdBu)
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)
        else:
            st.info("No customer data available")
```

```
    # Row kedua grafik
    col_left2, col_right2 = st.columns(2)

    with col_left2:
        st.subheader("🔥 Top 10 Best Selling Products")
        if not df_order_details.empty:
            top_products =
df_order_details.groupby('product_name')['quantity'].sum().sort_values(a
scending=False).head(10)

            fig = px.bar(x=top_products.values, y=top_products.index,
                         orientation='h',
                         labels={'x': 'Quantity Sold', 'y': 'Product'},
                         color=top_products.values,
                         color_continuous_scale='Viridis')
            fig.update_layout(height=400, showlegend=False)
            st.plotly_chart(fig, use_container_width=True)
        else:
            st.info("No sales data available")

    with col_right2:
        st.subheader("📅 Orders by Day of Week")
        if not df_order_details.empty:
            day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']
            orders_by_day =
df_order_details.groupby('day_name')['order_id'].nunique().reindex(day_o
rder, fill_value=0)

            fig = px.bar(x=orders_by_day.index, y=orders_by_day.values,
                         labels={'x': 'Day', 'y': 'Number of Orders'},
                         color=orders_by_day.values,
                         color_continuous_scale='Sunset')
            fig.update_layout(height=400, showlegend=False)
            st.plotly_chart(fig, use_container_width=True)
        else:
            st.info("No order data available")
```

```
# ====================================================
# CUSTOMER ANALYTICS
# ====================================================
def show_customers():
    """Analisis pelanggan dengan visualisasi interaktif"""
    st.header("👥 Customer Analytics")

    # Sidebar filters
    with st.sidebar:
        st.subheader("🔍 Customer Filters")

        # Age filter
        age_min = int(df_customers['Age'].min())
        age_max = int(df_customers['Age'].max())
        age_range = st.slider("Age Range", age_min, age_max, (age_min,
age_max))

        # Search by name
        search_name = st.text_input("Search by Name", "")

    # Apply filters
    filtered_customers =
df_customers[df_customers['Age'].between(*age_range)]
    if search_name:
        filtered_customers =
filtered_customers[filtered_customers['name'].str.contains(search_name,
case=False, na=False)]

    # Metrics
    col1, col2, col3 = st.columns(3)
    with col1:
        st.metric("Total Customers", len(filtered_customers))
    with col2:
        avg_age = filtered_customers['Age'].mean()
        st.metric("Average Age", f"{avg_age:.1f} years")
    with col3:
        if not df_order_details.empty:
            customers_with_orders =
df_order_details['customer_id'].nunique()
```

```
            st.metric("Active Buyers", f"{customers_with_orders:,}")
        else:
            st.metric("Active Buyers", "0")

    st.markdown("---")

    # Visualizations
    col_v1, col_v2 = st.columns(2)

    with col_v1:
        st.subheader("📊 Age Distribution")
        fig = px.histogram(filtered_customers, x='Age', nbins=20,
                           labels={'Age': 'Age (years)', 'count': 'Number
of Customers'},
                           color_discrete_sequence=['#42A5F5'])
        fig.update_layout(height=350)
        st.plotly_chart(fig, use_container_width=True)

    with col_v2:
        st.subheader("🎂 Birth Month Distribution")
        filtered_customers['birth_month'] =
filtered_customers['birthdate'].dt.month_name()
        month_order = ['January', 'February', 'March', 'April', 'May',
'June',
                       'July', 'August', 'September', 'October',
'November', 'December']
        birth_month_counts =
filtered_customers['birth_month'].value_counts().reindex(month_order,
fill_value=0)

        fig = px.line(x=birth_month_counts.index,
y=birth_month_counts.values,
                      markers=True,
                      labels={'x': 'Month', 'y': 'Number of Customers'},
                      color_discrete_sequence=['#66BB6A'])
        fig.update_layout(height=350)
        st.plotly_chart(fig, use_container_width=True)

    # Customer spending analysis
```

```
    if not df_order_details.empty:
        st.subheader("💳 Top 10 Customers by Spending")
        customer_spending = df_order_details.groupby(['customer_id',
'customer_name'])['subtotal'].sum().sort_values(ascending=False).head(10
)

        fig = go.Figure(data=[
            go.Bar(x=customer_spending.values, y=[name for _, name in
customer_spending.index],
                   orientation='h',
                   marker=dict(color=customer_spending.values,
                               colorscale='Plasma',
                               showscale=True))
        ])
        fig.update_layout(
            xaxis_title="Total Spending (Rp)",
            yaxis_title="Customer",
            height=400
        )
        st.plotly_chart(fig, use_container_width=True)

    # Data table
    st.subheader("📋 Customer Data Table")

    # Column selector
    available_cols = filtered_customers.columns.tolist()
    default_cols = ['customer_id', 'name', 'email', 'phone', 'Age',
'Age_Group']
    selected_cols = st.multiselect(
        "Select Columns to Display",
        available_cols,
        default=[col for col in default_cols if col in available_cols]
    )

    if selected_cols:
        # Sort options
        sort_col = st.selectbox("Sort by", selected_cols,
index=selected_cols.index('name') if 'name' in selected_cols else 0)
        sort_order = st.radio("Order", ["Ascending", "Descending"],
```

```
horizontal=True)

        display_df = filtered_customers[selected_cols].sort_values(
            by=sort_col,
            ascending=(sort_order == "Ascending")
        )

        st.dataframe(display_df, use_container_width=True, height=400)

        # Download button
        csv = display_df.to_csv(index=False).encode('utf-8')
        st.download_button(
            label="📥 Download Customer Data (CSV)",
            data=csv,
            file_name='customers_data.csv',
            mime='text/csv'
        )

# ========================================================
# PRODUCT ANALYTICS
# ========================================================
def show_products():
    """Analisis produk dengan visualisasi interaktif"""
    st.header("📦 Product Analytics")

    # Konversi tipe data produk
    df_products_work = df_products.copy()
    df_products_work['price'] = pd.to_numeric(df_products_work['price'],
errors='coerce').fillna(0)
    df_products_work['stock'] = pd.to_numeric(df_products_work['stock'],
errors='coerce').fillna(0).astype(int)

    # Calculate sales data
    if not df_order_details.empty:
        product_sales = df_order_details.groupby(['product_id',
'product_name']).agg({
            'quantity': 'sum',
            'subtotal': 'sum'
        }).reset_index()
```

```
        product_sales.columns = ['product_id', 'product_name',
'total_sold', 'total_revenue']

        # Konversi tipe data
        product_sales['total_sold'] =
pd.to_numeric(product_sales['total_sold'],
errors='coerce').fillna(0).astype(int)
        product_sales['total_revenue'] =
pd.to_numeric(product_sales['total_revenue'], errors='coerce').fillna(0)

        df_products_enhanced = df_products_work.merge(product_sales,
                                                      left_on=['product_id',
'name'],

right_on=['product_id', 'product_name'],
                                                      how='left')
        df_products_enhanced['total_sold'] =
pd.to_numeric(df_products_enhanced['total_sold'],
errors='coerce').fillna(0).astype(int)
        df_products_enhanced['total_revenue'] =
pd.to_numeric(df_products_enhanced['total_revenue'],
errors='coerce').fillna(0).astype(float)
    else:
        df_products_enhanced = df_products_work.copy()
        df_products_enhanced['total_sold'] = 0
        df_products_enhanced['total_revenue'] = 0.0

    # Sidebar filters
    with st.sidebar:
        st.subheader("🔍 Product Filters")

        # Price range
        price_min = float(df_products_enhanced['price'].min())
        price_max = float(df_products_enhanced['price'].max())
        price_range = st.slider("Price Range (Rp)", price_min,
price_max, (price_min, price_max))

        # Stock range
        stock_min = int(df_products_enhanced['stock'].min())
```

```
        stock_max = int(df_products_enhanced['stock'].max())
        stock_range = st.slider("Stock Range", stock_min, stock_max,
(stock_min, stock_max))

        # Search
        search_product = st.text_input("Search Product", "")

    # Apply filters
    filtered_products = df_products_enhanced[
        (df_products_enhanced['price'].between(*price_range)) &
        (df_products_enhanced['stock'].between(*stock_range))
    ]
    if search_product:
        filtered_products =
filtered_products[filtered_products['name'].str.contains(search_product,
case=False, na=False)]

    # Metrics
    col1, col2, col3, col4 = st.columns(4)
    with col1:
        st.metric("Total Products", len(filtered_products))
    with col2:
        total_stock = filtered_products['stock'].sum()
        st.metric("Total Stock", f"{total_stock:,}")
    with col3:
        total_sold = filtered_products['total_sold'].sum()
        st.metric("Total Units Sold", f"{int(total_sold):,}")
    with col4:
        avg_price = filtered_products['price'].mean()
        st.metric("Avg Price", f"Rp {avg_price:,.0f}")

    st.markdown("---")

    # Visualizations
    tab1, tab2, tab3 = st.tabs(["📊 Sales Analysis", "💰 Price
Analysis", "📦 Stock Analysis"])

    with tab1:
        col1, col2 = st.columns(2)
```

```
        with col1:
            st.subheader("🏆 Top 15 Best Sellers")
            top_sellers = filtered_products.nlargest(15, 'total_sold')

            fig = px.bar(top_sellers, x='total_sold', y='name',
                         orientation='h',
                         labels={'total_sold': 'Units Sold', 'name':
'Product'},
                         color='total_sold',
                         color_continuous_scale='Teal')
            fig.update_layout(height=500, showlegend=False)
            st.plotly_chart(fig, use_container_width=True)

        with col2:
            st.subheader("💵 Top 15 by Revenue")
            top_revenue = filtered_products.nlargest(15,
'total_revenue')

            fig = px.bar(top_revenue, x='total_revenue', y='name',
                         orientation='h',
                         labels={'total_revenue': 'Revenue (Rp)', 'name':
'Product'},
                         color='total_revenue',
                         color_continuous_scale='Oranges')
            fig.update_layout(height=500, showlegend=False)
            st.plotly_chart(fig, use_container_width=True)

    with tab2:
        col1, col2 = st.columns(2)

        with col1:
            st.subheader("💲 Price Distribution")
            fig = px.histogram(filtered_products, x='price', nbins=30,
                               labels={'price': 'Price (Rp)', 'count':
'Number of Products'},
                               color_discrete_sequence=['#7E57C2'])
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)
```

```
        with col2:
            st.subheader("📊 Price vs Sales Scatter")
            fig = px.scatter(filtered_products, x='price',
y='total_sold',
                             size='total_revenue', hover_data=['name'],
                             labels={'price': 'Price (Rp)', 'total_sold':
'Units Sold'},
                             color='total_revenue',
                             color_continuous_scale='Viridis')
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)

    with tab3:
        col1, col2 = st.columns(2)

        with col1:
            st.subheader("📦 Stock Distribution")
            fig = px.histogram(filtered_products, x='stock', nbins=25,
                               labels={'stock': 'Stock Quantity',
'count': 'Number of Products'},
                               color_discrete_sequence=['#26A69A'])
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)

        with col2:
            st.subheader("⚠️ Low Stock Alert (Stock < 20)")
            low_stock = filtered_products[filtered_products['stock'] <
20].nlargest(10, 'total_sold')

            if not low_stock.empty:
                fig = px.bar(low_stock, x='stock', y='name',
                             orientation='h',
                             labels={'stock': 'Stock Quantity', 'name':
'Product'},
                             color='stock',
                             color_continuous_scale='Reds')
                fig.update_layout(height=400, showlegend=False)
                st.plotly_chart(fig, use_container_width=True)
```

```
            else:
                st.success("✅ All products have sufficient stock!")

    # Data table
    st.subheader("📋 Product Data Table")

    available_cols = filtered_products.columns.tolist()
    default_cols = ['product_id', 'name', 'price', 'stock',
'total_sold', 'total_revenue']
    selected_cols = st.multiselect(
        "Select Columns to Display",
        available_cols,
        default=[col for col in default_cols if col in available_cols]
    )

    if selected_cols:
        sort_col = st.selectbox("Sort by", selected_cols,
                                index=selected_cols.index('total_sold')
if 'total_sold' in selected_cols else 0)
        sort_order = st.radio("Order", ["Ascending", "Descending"],
horizontal=True, key='prod_sort')

        display_df = filtered_products[selected_cols].sort_values(
            by=sort_col,
            ascending=(sort_order == "Ascending")
        )

        st.dataframe(display_df, use_container_width=True, height=400)

        csv = display_df.to_csv(index=False).encode('utf-8')
        st.download_button(
            label="📥 Download Product Data (CSV)",
            data=csv,
            file_name='products_data.csv',
            mime='text/csv'
        )

# ======================================================
# SALES ANALYTICS
```

```python
# ====================================================
def show_sales():
    """Analisis penjualan dengan visualisasi mendalam"""
    st.header("💰 Sales Analytics")

    if df_order_details.empty:
        st.warning("No sales data available")
        return

    # Sidebar filters
    with st.sidebar:
        st.subheader("🔍 Sales Filters")

        # Date range
        min_date = df_order_details['order_date'].min().date()
        max_date = df_order_details['order_date'].max().date()
        date_range = st.date_input(
            "Date Range",
            value=(min_date, max_date),
            min_value=min_date,
            max_value=max_date
        )

        # Customer filter
        all_customers = ['All'] +
sorted(df_order_details['customer_name'].unique().tolist())
        selected_customer = st.selectbox("Customer", all_customers)

        # Product filter
        all_products = ['All'] +
sorted(df_order_details['product_name'].unique().tolist())
        selected_product = st.selectbox("Product", all_products)

    # Apply filters
    filtered_sales = df_order_details[
        (df_order_details['order_date'].dt.date >= date_range[0]) &
        (df_order_details['order_date'].dt.date <= date_range[1])
    ]
```

```
    if selected_customer != 'All':
        filtered_sales = filtered_sales[filtered_sales['customer_name']
== selected_customer]

    if selected_product != 'All':
        filtered_sales = filtered_sales[filtered_sales['product_name']
== selected_product]

    # Key Metrics
    col1, col2, col3, col4 = st.columns(4)

    with col1:
        total_orders = filtered_sales['order_id'].nunique()
        st.metric("🛒 Total Orders", f"{total_orders:,}")

    with col2:
        total_items = filtered_sales['quantity'].sum()
        st.metric("📦 Items Sold", f"{int(total_items):,}")

    with col3:
        total_revenue = filtered_sales['subtotal'].sum()
        st.metric("💰 Total Revenue", f"Rp {total_revenue:,.0f}")

    with col4:
        avg_order_value =
filtered_sales.groupby('order_id')['subtotal'].sum().mean()
        st.metric("📊 Avg Order Value", f"Rp {avg_order_value:,.0f}")

    st.markdown("---")

    # Tabs untuk berbagai analisis
    tab1, tab2, tab3, tab4 = st.tabs(["📈 Time Series", "🏆 Top
Performers", "⏰ Time Analysis", "🔍 Detailed Data"])

    with tab1:
        st.subheader("Revenue & Orders Over Time")

        # Pilihan granularity
        granularity = st.radio("Select Granularity", ["Daily", "Weekly",
```

```
"Monthly"], horizontal=True)

        if granularity == "Daily":
            time_series =
filtered_sales.groupby(filtered_sales['order_date'].dt.date).agg({
                'subtotal': 'sum',
                'order_id': 'nunique'
            }).reset_index()
            time_series.columns = ['Date', 'Revenue', 'Orders']
        elif granularity == "Weekly":
            time_series =
filtered_sales.groupby(filtered_sales['order_date'].dt.to_period('W').dt
.start_time).agg({
                'subtotal': 'sum',
                'order_id': 'nunique'
            }).reset_index()
            time_series.columns = ['Date', 'Revenue', 'Orders']
        else:  # Monthly
            time_series =
filtered_sales.groupby(filtered_sales['order_date'].dt.to_period('M').dt
.start_time).agg({
                'subtotal': 'sum',
                'order_id': 'nunique'
            }).reset_index()
            time_series.columns = ['Date', 'Revenue', 'Orders']

        # Dual axis chart
        fig = go.Figure()

        fig.add_trace(go.Scatter(
            x=time_series['Date'], y=time_series['Revenue'],
            name='Revenue',
            line=dict(color='#1E88E5', width=3),
            fill='tonexty'
        ))

        fig.add_trace(go.Bar(
            x=time_series['Date'], y=time_series['Orders'],
            name='Orders',
```

```
            marker_color='#FFA726',
            yaxis='y2',
            opacity=0.6
        ))

        fig.update_layout(
            yaxis=dict(title='Revenue (Rp)', side='left'),
            yaxis2=dict(title='Number of Orders', side='right',
overlaying='y'),
            hovermode='x unified',
            height=450
        )

        st.plotly_chart(fig, use_container_width=True)

    with tab2:
        col1, col2 = st.columns(2)

        with col1:
            st.subheader("🥇 Top 10 Products by Revenue")
            top_products_revenue =
filtered_sales.groupby('product_name')['subtotal'].sum().sort_values(asc
ending=False).head(10)

            fig = px.bar(x=top_products_revenue.values,
y=top_products_revenue.index,
                         orientation='h',
                         labels={'x': 'Revenue (Rp)', 'y': 'Product'},
                         color=top_products_revenue.values,
                         color_continuous_scale='Blues')
            fig.update_layout(height=450, showlegend=False)
            st.plotly_chart(fig, use_container_width=True)

        with col2:
            st.subheader("🥇 Top 10 Customers by Spending")
            top_customers =
filtered_sales.groupby('customer_name')['subtotal'].sum().sort_values(as
cending=False).head(10)
```

```
            fig = px.bar(x=top_customers.values, y=top_customers.index,
                         orientation='h',
                         labels={'x': 'Total Spending (Rp)', 'y':
'Customer'},
                         color=top_customers.values,
                         color_continuous_scale='Greens')
            fig.update_layout(height=450, showlegend=False)
            st.plotly_chart(fig, use_container_width=True)

        col3, col4 = st.columns(2)

        with col3:
            st.subheader("🔥 Most Popular Products (by Quantity)")
            top_quantity =
filtered_sales.groupby('product_name')['quantity'].sum().sort_values(asc
ending=False).head(10)

            fig = px.pie(values=top_quantity.values,
names=top_quantity.index,

color_discrete_sequence=px.colors.sequential.RdBu)
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)

        with col4:
            st.subheader("💎 Revenue Distribution by Product")
            revenue_by_product =
filtered_sales.groupby('product_name')['subtotal'].sum().sort_values(asc
ending=False).head(10)

            fig = px.pie(values=revenue_by_product.values,
names=revenue_by_product.index,

color_discrete_sequence=px.colors.sequential.Plasma)
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)

    with tab3:
        col1, col2 = st.columns(2)
```

```
        with col1:
            st.subheader("📅 Sales by Day of Week")
            day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']
            sales_by_day =
filtered_sales.groupby('day_name')['subtotal'].sum().reindex(day_order,
fill_value=0)

            fig = px.bar(x=sales_by_day.index, y=sales_by_day.values,
                         labels={'x': 'Day', 'y': 'Revenue (Rp)'},
                         color=sales_by_day.values,
                         color_continuous_scale='Sunset')
            fig.update_layout(height=400, showlegend=False)
            st.plotly_chart(fig, use_container_width=True)

        with col2:
            st.subheader("⏰ Sales by Hour of Day")
            sales_by_hour =
filtered_sales.groupby('hour')['subtotal'].sum()

            fig = px.line(x=sales_by_hour.index, y=sales_by_hour.values,
                          markers=True,
                          labels={'x': 'Hour of Day', 'y': 'Revenue
(Rp)'},
                          color_discrete_sequence=['#E91E63'])
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)

        st.subheader("🗓️ Sales Heatmap by Month and Day")
        if len(filtered_sales) > 0:
            heatmap_data = filtered_sales.groupby(['month_name',
'day_name'])['subtotal'].sum().reset_index()
            heatmap_pivot = heatmap_data.pivot(index='day_name',
columns='month_name', values='subtotal').fillna(0)

            # Reorder
            month_order = ['January', 'February', 'March', 'April',
'May', 'June',
```

```
                          'July', 'August', 'September', 'October',
'November', 'December']
            day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']

            heatmap_pivot = heatmap_pivot.reindex(index=day_order,
columns=[m for m in month_order if m in heatmap_pivot.columns],
fill_value=0)

            fig = px.imshow(heatmap_pivot,
                            labels=dict(x="Month", y="Day",
color="Revenue (Rp)"),
                            color_continuous_scale='YlOrRd',
                            aspect="auto")
            fig.update_layout(height=400)
            st.plotly_chart(fig, use_container_width=True)

    with tab4:
        st.subheader("📊 Detailed Sales Data")

        available_cols = filtered_sales.columns.tolist()
        default_cols = ['order_date', 'customer_name', 'product_name',
'quantity', 'unit_price', 'subtotal']
        selected_cols = st.multiselect(
            "Select Columns",
            available_cols,
            default=[col for col in default_cols if col in
available_cols]
        )

        if selected_cols:
            sort_col = st.selectbox("Sort by", selected_cols,

index=selected_cols.index('order_date') if 'order_date' in selected_cols
else 0)
            sort_order = st.radio("Order", ["Ascending", "Descending"],
horizontal=True, key='sales_sort')

            display_df = filtered_sales[selected_cols].sort_values(
```

```
                by=sort_col,
                ascending=(sort_order == "Ascending")
            )

            st.dataframe(display_df, use_container_width=True,
height=400)

            csv = display_df.to_csv(index=False).encode('utf-8')
            st.download_button(
                label="📥 Download Sales Data (CSV)",
                data=csv,
                file_name='sales_data.csv',
                mime='text/csv'
            )

# ============================================================
# MAIN NAVIGATION
# ============================================================
st.sidebar.title("🧭 Navigation")
st.sidebar.markdown("---")

page = st.sidebar.radio(
    "Select Page",
    ["🏠 Overview", "👥 Customers", "📦 Products", "💰 Sales
Analytics"],
    label_visibility="collapsed"
)

st.sidebar.markdown("---")
st.sidebar.info("**Jets Sales Analytics** v1.0\n\nBuilt with Streamlit &
PostgreSQL")

# Route to selected page
if page == "🏠 Overview":
    show_overview()
elif page == "👥 Customers":
    show_customers()
elif page == "📦 Products":
    show_products()
```

```
elif page == "💰 Sales Analytics":
    show_sales()
```

config.py

```
import psycopg2

# Koneksi ke database PostgreSQL
conn = psycopg2.connect(
    host="localhost",
    port="5432",          # port default PostgreSQL
    user="postgres",      # ganti sesuai user PostgreSQL kamu
    password="putu2520",  # ganti sesuai password PostgreSQL kamu
    dbname="jets"         # nama database
)

print("Koneksi PostgreSQL berhasil!")

# Membuat cursor
c = conn.cursor()

# ============================
# Fungsi ambil data dari tabel
# ============================

def view_customers():
    query = '''
        SELECT customer_id, name, email, phone, address, birthdate
        FROM customers
        ORDER BY name ASC
    '''
    c.execute(query)
    return c.fetchall()
```

```
def view_orders_with_customers():
    query = '''
        SELECT
            o.order_id,
            o.order_date,
            o.total_amount,
            c.name AS customer_name,
            c.phone
        FROM orders o
        JOIN customers c ON o.customer_id = c.customer_id
        ORDER BY o.order_date DESC
    '''
    c.execute(query)
    return c.fetchall()

def view_products():
    query = '''
        SELECT product_id, name, description, price, stock
        FROM products
        ORDER BY name ASC
    '''
    c.execute(query)
    return c.fetchall()

def view_order_details_with_info():
    query = '''
        SELECT
            od.order_detail_id,
            o.order_id,
            o.order_date,
            c.customer_id,
            c.name AS customer_name,
            p.product_id,
            p.name AS product_name,
            p.price AS unit_price,
            od.quantity,
            od.subtotal,
            o.total_amount AS order_total,
```

```
            c.phone
        FROM order_details od
        JOIN orders o ON od.order_id = o.order_id
        JOIN customers c ON o.customer_id = c.customer_id
        JOIN products p ON od.product_id = p.product_id
        ORDER BY o.order_date DESC
    '''
    c.execute(query)
    return c.fetchall()


    # Tutup koneksi
    c.close()
    conn.close()
```