MANUAL BOOK KELOMPOK 6 (BUIN 3)

Langkah 1: Instalasi WSL dan Ubuntu 24.04.1 LTS (Windows)

- Aktifkan Fitur WSL di Windows:
- Tekan Win + R, ketik: optionalfeatures dan tekan Enter.
- Centang:
    - Windows Subsystem for Linux
    - Virtual Machine Platform
- Klik OK → Restart PC/Laptop.

Langkah 2: Install Miniconda di Ubuntu

- Download Miniconda: wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
- Cek integritas: sha256sum Miniconda3-latest-Linux-x86_64.sh
- Jalankan installer: bash Miniconda3-latest-Linux-x86_64.sh
- Aktivasi conda: source ~/.bashrc
  Jika tidak aktif: ~/miniconda3/bin/conda init bash && source ~/.bashrc

Langkah 3: Setup Environment Airflow

- Buat environment: conda create -n airflenv python=3.9
- Aktifkan environment: conda activate airflenv

Langkah 4: Install Apache Airflow

- Install airflow: pip install 'apache-airflow==3.0.1' --constraint "https://raw.githubusercontent.com/apache/airflow/constraints-3.0.1/constraints-3.9.txt"
- Setup airflow:
    - mkdir airflow
    - cd airflow
      airflow db migrate
- Jalankan airflow: airflow standalone
  Buka browser ke: http://localhost:8080 (Jika berhasil maka akan muncul menu login yang menampilkan username dan password.

Langkah 5: Install Django dan Setup Project

- Pastikan masih di airflenv: conda activate airflenv
- Install Django: pip install django
- Buat project: Django admin startproject app_query && cd app_query
- Buat app: python manage.py startapp bi
- Tambahkan ke INSTALLED_APPS di settings.py

Langkah 6: Install Django

- Lakukan di terminal virtual environment airflow “pip install django”
- django-admin startproject app_query
- cd app_query
- python manage.py startapp business_intelligence
- Jalankan djanggo dengan “python manage.py runserver (untuk awal akan berupa gambar roket karena data belum termigrasi)

Langkah 7 : Menyiapkan Dataset dan File ETL

- Setelah itu buat folder di airflow bernama dags untuk meletakkan data csv dan kode ETL (Python).
- Kode yang digunakan merupakan data Product Classification and Clustering (pricerunner_aggregate.csv) yang diambil dari UCI Machine Learning.

Langkah 8: Membuat Kode Untuk : ETL Airflow dan Integrasi Django

Saat env aktif dan berada directory app query ketikkan “code .” untuk membuka VScode yang menampilkan Airflow dan Django. Setelah itu masuk ke folder airflow maka akan menemui kode dags buat file etl python di dalamnya untuk melakukan etl. Program ini digunakan untuk melakukan proses ETL (Extract, Transform, Load) pada dataset pricerunner_aggregate.csv menggunakan Apache Airflow. Proses ini bertujuan membentuk skema data berbentuk Star Schema yang terdiri dari beberapa tabel dimensi dan tabel fakta, kemudian menyimpannya dalam format CSV.

Langkah pertama dilakukan melalui fungsi extract(), yaitu membaca file CSV mentah dan menyimpannya ulang sebagai raw_data.csv di folder star_output. Selanjutnya, fungsi transform() akan melakukan pemrosesan data. Pada tahap ini, data dibersihkan, kolom-kolom penting ditambahkan, lalu dibuatlah tujuh tabel dimensi, yaitu: dim_product, dim_category, dim_merchant, dim_cluster, dim_date, dim_offer_quality, dan dim_distribution_status. Selain itu, dihasilkan pula tiga tabel fakta, yaitu: fact_product_classification, fact_price_comparison, dan fact_product_distribution, yang mencerminkan aktivitas bisnis e-commerce dari berbagai sudut pandang. Setelah proses transformasi selesai, fungsi load() akan dijalankan untuk menampilkan notifikasi bahwa seluruh proses ETL berhasil dilakukan dan file hasil sudah disimpan di folder star_output.

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
import pandas as pd
import os

# Path setup
dag_path = os.path.dirname(__file__)
output_path = os.path.join(dag_path, 'star_output')
os.makedirs(output_path, exist_ok=True)


def extract():
    df = pd.read_csv(os.path.join(dag_path, 'pricerunner_aggregate.csv'))
    df.columns = df.columns.str.strip()
    df.to_csv(os.path.join(output_path, 'raw_data.csv'), index=False)


def transform():
    df = pd.read_csv(os.path.join(output_path, 'raw_data.csv'))
    df.columns = df.columns.str.strip()

    # Tambahkan ID dimensi dan dummy kolom
    df['product_id'] = df['Product ID']
    df['category_id'] = df['Category ID']
    df['merchant_id'] = df['Merchant ID']
    df['cluster_id'] = df['Cluster ID']
    df['date_id'] = 1
    df['status_id'] = 1
    df['offer_quality_id'] = 1

    # --- DIMENSI --- (dim_product, dim_category, dim_merchant, dim_cluster, dim_date, dim_offer_quality, dim_distribution_status)
    dim_product = df[['product_id', 'Product Title']].drop_duplicates()
    dim_product['product_description'] = 'N/A'
    dim_product['product_url'] = 'https://example.com/product'
    dim_product['image_url'] = 'https://example.com/image.jpg'
    dim_product['price'] = 100000.0
    dim_product['currency'] = 'IDR'
    dim_product['rating'] = 4.0
    dim_product['is_active'] = True
    dim_product['created_at'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    dim_product['updated_at'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    dim_product.columns = [
        'product_id', 'product_title', 'product_description', 'product_url',
        'image_url', 'price', 'currency', 'rating', 'is_active',
        'created_at', 'updated_at'
    ]
    dim_product.to_csv(os.path.join(output_path, 'dim_product.csv'), index=False)
```

```
    dim_category = df[['category_id', 'Category Label']].drop_duplicates()
    dim_category.columns = ['category_id', 'category_name']
    dim_category['parent_category_id'] = None
    dim_category['category_level'] = 1
    dim_category['category_path'] = dim_category['category_name']
    dim_category['is_standardized'] = True
    dim_category['created_at'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    dim_category['updated_at'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    dim_category.to_csv(os.path.join(output_path, 'dim_category.csv'), index=False)

    dim_merchant = df[['merchant_id']].drop_duplicates()
    dim_merchant['merchant_name'] = 'Merchant ' + dim_merchant['merchant_id'].astype(str)
    dim_merchant['merchant_rating'] = 4.5
    dim_merchant['merchant_type'] = 'resmi'
    dim_merchant['merchant_url'] = 'https://example.com/store'
    dim_merchant['merchant_location'] = 'Indonesia'
    dim_merchant['joined_date'] = datetime.now().strftime('%Y-%m-%d')
    dim_merchant['is_active'] = True
    dim_merchant.to_csv(os.path.join(output_path, 'dim_merchant.csv'), index=False)

# --- DIM CLUSTER (Pakai data mentah, jaminan benar) ---
    dim_cluster = df[['Cluster ID', 'Cluster Label']].drop_duplicates()
    dim_cluster.columns = ['cluster_id', 'cluster_name']

# Hindari pakai groupby-transform, langsung hitung size pakai dataframe terpisah
    cluster_sizes = df.groupby('cluster_id')['product_id'].count().reset_index()
    cluster_sizes.columns = ['cluster_id', 'cluster_size']

# Gabungkan ukuran cluster
    dim_cluster = dim_cluster.merge(cluster_sizes, on='cluster_id', how='left')

# Tambahkan kolom lain
    dim_cluster['cluster_description'] = 'Produk: ' + dim_cluster['cluster_name']
    dim_cluster['cluster_keywords'] = 'otomatis'
    dim_cluster['cluster_confidence'] = 95.0
    dim_cluster['created_at'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    dim_cluster['updated_at'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

# Simpan hasil
    dim_cluster.to_csv(os.path.join(output_path, 'dim_cluster.csv'), index=False)

    dim_date = pd.DataFrame([{
        'date_id': 1,
        'full_date': datetime.today().strftime('%Y-%m-%d'),
        'day_of_week': datetime.today().isoweekday(),
        'day_name': datetime.today().strftime('%A'),
        'month': datetime.today().month,
        'month_name': datetime.today().strftime('%B'),
        'quarter': (datetime.today().month - 1) // 3 + 1,
        'year': datetime.today().year,
        'is_weekend': datetime.today().isoweekday() in [6, 7]
    }])
```

```
dim_date.to_csv(os.path.join(output_path, 'dim_date.csv'), index=False)

dim_offer_quality = pd.DataFrame([{
    'offer_quality_id': 1,
    'quality_level': 'Baik',
    'price_to_rating_ratio': 25000.0,
    'price_competitiveness': 'Kompetitif',
    'merchant_reliability_score': 90.0,
    'description': 'Penawaran baik dengan merchant terpercaya'
}])
dim_offer_quality.to_csv(os.path.join(output_path, 'dim_offer_quality.csv'), index=False)

dim_distribution_status = pd.DataFrame([{
    'status_id': 1,
    'status_name': 'Normal',
    'status_description': 'Distribusi produk stabil',
    'recommended_action': 'Tidak ada tindakan',
    'color_code': '#00FF00'
}])
dim_distribution_status.to_csv(os.path.join(output_path, 'dim_distribution_status.csv'), index=False)

# --- FAKTA 1: fact_product_classification ---
fact1 = df[['product_id', 'category_id', 'merchant_id', 'cluster_id', 'date_id']].copy()
fact1['product_classification_id'] = fact1.index + 1
fact1['classification_confidence'] = 100.0
fact1['is_validated'] = True
fact1['last_update_timestamp'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
fact1 = fact1[['product_classification_id', 'product_id', 'category_id', 'merchant_id', 'cluster_id', 'date_id',
               'classification_confidence', 'is_validated', 'last_update_timestamp']]
fact1.to_csv(os.path.join(output_path, 'fact_product_classification.csv'), index=False)

# --- FAKTA 2: fact_price_comparison ---
fact2 = df[['product_id', 'merchant_id', 'cluster_id', 'category_id', 'date_id', 'offer_quality_id']].copy()
fact2['price_comparison_id'] = fact2.index + 1
fact2['price'] = 100000.0
fact2['price_difference_from_avg'] = 0.0
fact2['price_difference_percentage'] = 0.0
fact2['is_best_offer'] = True
fact2['price_rank_in_cluster'] = 1
fact2['price_rank_in_category'] = 1
fact2['price_timestamp'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
fact2 = fact2[['price_comparison_id', 'product_id', 'merchant_id', 'cluster_id', 'category_id', 'date_id', 'offer_quality_id',
               'price', 'price_difference_from_avg', 'price_difference_percentage', 'is_best_offer',
               'price_rank_in_cluster', 'price_rank_in_category', 'price_timestamp']]
fact2.to_csv(os.path.join(output_path, 'fact_price_comparison.csv'), index=False)
```

```
    # --- FAKTA 3: fact_product_distribution ---
    grouped = df.groupby(['category_id', 'cluster_id', 'date_id', 'status_id']).agg(
        total_products=('Product ID', 'count'),
        total_merchants=('Merchant ID', pd.Series.nunique),
        avg_price=('Product ID', lambda x: 100000.0),
        min_price=('Product ID', lambda x: 95000.0),
        max_price=('Product ID', lambda x: 105000.0),
        price_std_deviation=('Product ID', lambda x: 2000.0),
        avg_rating=('Product ID', lambda x: 4.2),
        duplicates_detected=('Product ID', lambda x: 0),
        miscategorized_count=('Product ID', lambda x: 0)
    ).reset_index()
    grouped['distribution_id'] = grouped.index + 1
    grouped['last_updated'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    grouped = grouped[['distribution_id', 'category_id', 'cluster_id', 'date_id', 'status_id',
                       'total_products', 'total_merchants', 'avg_price', 'min_price', 'max_price',
                       'price_std_deviation', 'avg_rating', 'duplicates_detected',
                       'miscategorized_count', 'last_updated']]
    grouped.to_csv(os.path.join(output_path, 'fact_product_distribution.csv'), index=False)


def load():
    print("Star Schema ETL complete. Files saved in:", output_path)


with DAG(
    dag_id='etl_star_schema',
    start_date=datetime(2023, 1, 1),
    schedule='@daily',
    catchup=False,
    tags=['star_schema', 'etl', 'ecommerce']
) as dag:

    t1 = PythonOperator(
        task_id='extract',
        python_callable=extract
    )

    t2 = PythonOperator(
        task_id='transform',
        python_callable=transform
    )

    t3 = PythonOperator(
        task_id='load',
        python_callable=load
    )

    t1 >> t2 >> t3
```

Pada bagian INSTALLED_APPS dalam file settings.py, ditambahkan 'bi' sebagai salah satu aplikasi yang terdaftar di dalam proyek Django. Penambahan ini diperlukan agar Django dapat mengenali dan menjalankan modul aplikasi bernama bi yang berisi logika *Business Intelligence*, seperti model data OLAP, fungsi prediksi tren harga, serta konfigurasi tampilan dashboard pada browser. Tanpa mendaftarkan 'bi' di INSTALLED_APPS, Django tidak akan memproses file seperti models.py, views.py, dan templates yang berada di dalam folder aplikasi tersebut.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bi',
]
```

Selanjutnya kode pada file urls.py digunakan untuk mendefinisikan URL routing dalam aplikasi Django. Setiap baris path() menghubungkan URL tertentu ke fungsi tampilan (*view*) yang sesuai. Misalnya, saat pengguna mengakses /dashboard/, maka akan ditampilkan halaman utama dashboard. URL lain seperti /analysis/price/ dan /chart/avg-price-actual-predicted/ akan memuat hasil analisis harga dan grafik perbandingan nilai aktual dan prediksi. Semua rute ini berfungsi untuk menampilkan visualisasi dan analisis data OLAP melalui antarmuka web secara dinamis.

```
from django.urls import path
from .views import dashboard
from .views import analysis_price
from .views import chart_product_category_actual_predicted
from .views import chart_rating_actual_predicted
from .views import chart_avg_price_actual_predicted

urlpatterns = [
    path('dashboard/', dashboard, name='dashboard'),
    path('analysis/price/', analysis_price, name='analysis_price'),
    path('chart/category-actual-predicted/', chart_product_category_actual_predicted, name='category_actual_predicted'),
    path('chart/rating-actual-predicted/', chart_rating_actual_predicted, name='rating_actual_predicted'),
    path('chart/avg-price-actual-predicted/', chart_avg_price_actual_predicted, name='avg_price_actual_predicted'),
]
```

Kode pada file models.py mendefinisikan struktur tabel-tabel dalam basis data Django untuk membangun Star Schema. Tabel-tabel ini terbagi menjadi dua jenis:

1. Tabel Dimensi (*Product, Category, Merchant, Cluster, Date, OfferQuality*, dan *DistributionStatus*) berisi informasi deskriptif seperti nama produk, kategori, merchant, waktu, kualitas penawaran, dan status distribusi.

2. Tabel Fakta (*ProductClassification, PriceComparison, dan ProductDistribution*) berisi data hasil agregasi dan aktivitas bisnis yang menghubungkan dimensi-dimensi tersebut, seperti klasifikasi produk, perbandingan harga, dan distribusi produk.

Relasi antar tabel dibentuk menggunakan *ForeignKey*, sehingga dapat dilakukan analisis OLAP untuk kebutuhan visualisasi dan prediksi data pada dashboard Django.

```python
from django.db import models

# --- DIMENSI ---

class Product(models.Model):
    product_id = models.CharField(max_length=100, primary_key=True)
    product_title = models.CharField(max_length=255)
    product_description = models.TextField()
    product_url = models.URLField()
    image_url = models.URLField()
    price = models.FloatField()
    currency = models.CharField(max_length=10)
    rating = models.FloatField()
    is_active = models.BooleanField()
    created_at = models.DateTimeField()
    updated_at = models.DateTimeField()

    def __str__(self):
        return self.product_title

class Category(models.Model):
    category_id = models.CharField(max_length=100, primary_key=True)
    category_name = models.CharField(max_length=255)
    parent_category_id = models.CharField(max_length=100, null=True, blank=True)
    category_level = models.IntegerField()
    category_path = models.TextField()
    is_standardized = models.BooleanField()
    created_at = models.DateTimeField()
    updated_at = models.DateTimeField()

    def __str__(self):
        return self.category_name

class Merchant(models.Model):
    merchant_id = models.CharField(max_length=100, primary_key=True)
    merchant_name = models.CharField(max_length=255)
    merchant_rating = models.FloatField()
    merchant_type = models.CharField(max_length=50)
    merchant_url = models.URLField()
    merchant_location = models.CharField(max_length=255)
    joined_date = models.DateField()
    is_active = models.BooleanField()

    def __str__(self):
        return self.merchant_name

class Cluster(models.Model):
    cluster_id = models.CharField(max_length=100, primary_key=True)
    cluster_name = models.CharField(max_length=255)
    cluster_description = models.TextField()
    cluster_keywords = models.TextField()
    cluster_size = models.IntegerField()
    cluster_confidence = models.FloatField()
    created_at = models.DateTimeField()
    updated_at = models.DateTimeField()

    def __str__(self):
        return self.cluster_name

class Date(models.Model):
    date_id = models.IntegerField(primary_key=True)
    full_date = models.DateField()
    day_of_week = models.IntegerField()
    day_name = models.CharField(max_length=20)
    month = models.IntegerField()
    month_name = models.CharField(max_length=20)
    quarter = models.IntegerField()
    year = models.IntegerField()
    is_weekend = models.BooleanField()
```

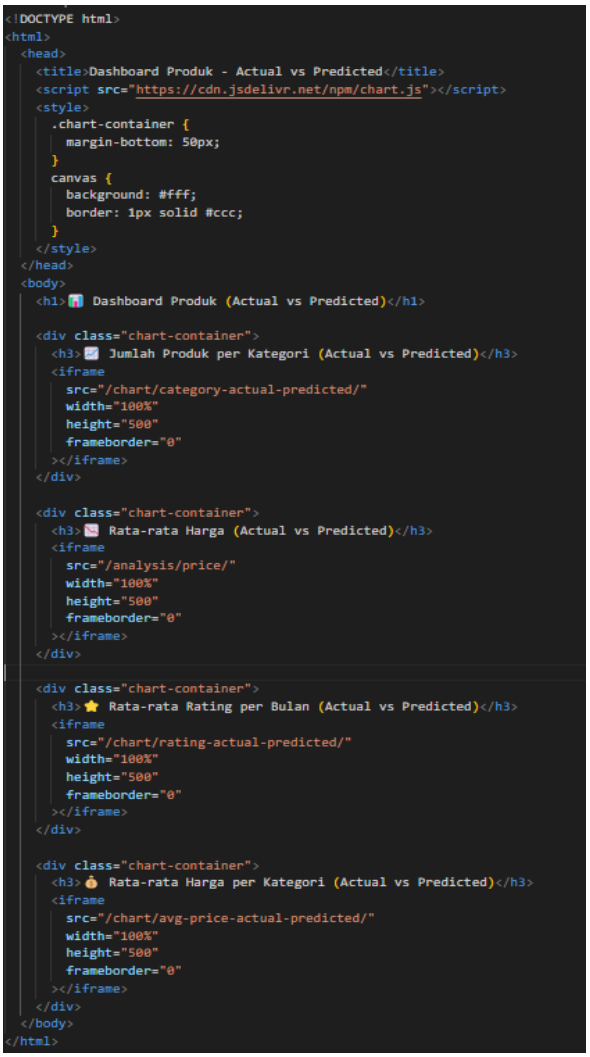
```python
    def __str__(self):
        return str(self.full_date)

class OfferQuality(models.Model):
    offer_quality_id = models.IntegerField(primary_key=True)
    quality_level = models.CharField(max_length=50)
    price_to_rating_ratio = models.FloatField()
    price_competitiveness = models.CharField(max_length=50)
    merchant_reliability_score = models.FloatField()
    description = models.TextField()

    def __str__(self):
        return self.quality_level

class DistributionStatus(models.Model):
    status_id = models.IntegerField(primary_key=True)
    status_name = models.CharField(max_length=50)
    status_description = models.TextField()
    recommended_action = models.TextField()
    color_code = models.CharField(max_length=10)

    def __str__(self):
        return self.status_name

# --- FAKTA ---

class ProductClassification(models.Model):
    product_classification_id = models.AutoField(primary_key=True)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    merchant = models.ForeignKey(Merchant, on_delete=models.CASCADE)
    cluster = models.ForeignKey(Cluster, on_delete=models.CASCADE)
    date = models.ForeignKey(Date, on_delete=models.CASCADE)
    classification_confidence = models.FloatField()
    is_validated = models.BooleanField()
    last_update_timestamp = models.DateTimeField()

class PriceComparison(models.Model):
    price_comparison_id = models.AutoField(primary_key=True)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    merchant = models.ForeignKey(Merchant, on_delete=models.CASCADE)
    cluster = models.ForeignKey(Cluster, on_delete=models.CASCADE)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    date = models.ForeignKey(Date, on_delete=models.CASCADE)
    offer_quality = models.ForeignKey(OfferQuality, on_delete=models.CASCADE)
    price = models.FloatField()
    price_difference_from_avg = models.FloatField()
    price_difference_percentage = models.FloatField()
    is_best_offer = models.BooleanField()
    price_rank_in_cluster = models.IntegerField()
    price_rank_in_category = models.IntegerField()
    price_timestamp = models.DateTimeField()

class ProductDistribution(models.Model):
    distribution_id = models.AutoField(primary_key=True)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    cluster = models.ForeignKey(Cluster, on_delete=models.CASCADE)
    date = models.ForeignKey(Date, on_delete=models.CASCADE)
    status = models.ForeignKey(DistributionStatus, on_delete=models.CASCADE)
    total_products = models.IntegerField()
    total_merchants = models.IntegerField()
    avg_price = models.FloatField()
    min_price = models.FloatField()
    max_price = models.FloatField()
    price_std_deviation = models.FloatField()
    avg_rating = models.FloatField()
    duplicates_detected = models.IntegerField()
    miscategorized_count = models.IntegerField()
    last_updated = models.DateTimeField()
```

Kode pada file views.py berisi kumpulan fungsi tampilan (*views*) dalam Django yang digunakan untuk mengambil dan mengolah data CSV hasil ETL lalu mengirimkan data tersebut ke template HTML untuk divisualisasikan dalam bentuk grafik. Beberapa fungsi seperti analysis_price(), chart_rating_actual_predicted(), dan chart_avg_price_actual_predicted() menggunakan model regresi linier dari library scikit-learn (LinearRegression) untuk memprediksi tren data ke depan, seperti prediksi harga rata-rata, prediksi rating bulanan, dan prediksi rata-rata harga per kategori. Data-data yang digunakan dalam prediksi ini berasal dari file dim_product.csv hasil ETL, dan dianalisis berdasarkan waktu (created_at) atau berdasarkan kategori produk. Setiap fungsi mengembalikan hasil analisis berupa *label* (misalnya bulan atau kategori), nilai aktual, dan nilai prediksi, yang dikirim ke HTML melalui objek context. Dengan demikian, kode ini sepenuhnya memanfaatkan pendekatan regresi linier sederhana untuk tujuan prediktif, tanpa menggunakan metode atau model lain.

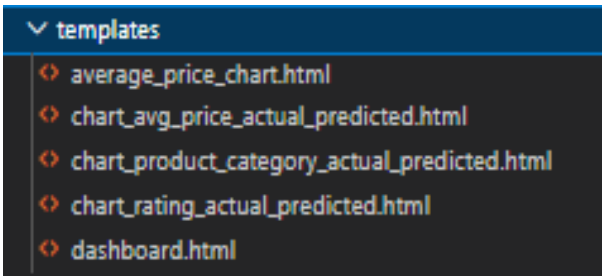
```python
from django.shortcuts import render
import pandas as pd
from django.conf import settings
import os
import numpy as np
from sklearn.linear_model import LinearRegression

def analysis_price(request):
    path = os.path.join(settings.BASE_DIR, 'star_output', 'dim_product.csv')
    df = pd.read_csv(path)

    # Pastikan kolom waktu dan harga valid
    df['created_at'] = pd.to_datetime(df['created_at'], errors='coerce')
    df = df.dropna(subset=['created_at', 'price'])

    # Buat kolom bulanan
    df['year_month'] = df['created_at'].dt.to_period('M').astype(str)

    # Hitung rata-rata harga per bulan
    df_summary = df.groupby('year_month')['price'].mean().reset_index()

    # Konversi index jadi fitur numerik untuk regresi
    df_summary['time_index'] = np.arange(len(df_summary)).reshape(-1, 1)

    # Siapkan data X dan y
    X = df_summary['time_index'].values.reshape(-1, 1)
    y = df_summary['price'].values

    # Model linear regression
    model = LinearRegression()
    model.fit(X, y)

    # Prediksi
    df_summary['predicted_price'] = model.predict(X)

    context = {
        'labels': df_summary['year_month'].tolist(),
        'actual': df_summary['price'].round().astype(int).tolist(),
        'predicted': df_summary['predicted_price'].round().astype(int).tolist()
    }

    return render(request, 'average_price_chart.html', context)

def chart_product_category_actual_predicted(request):
    path = os.path.join(settings.BASE_DIR, 'star_output', 'dim_product.csv')
    df = pd.read_csv(path)

    # Pastikan kolom ada
    if 'category_name' not in df.columns:
        return render(request, 'bi/error.html', {'message': 'Kolom category_name tidak ditemukan'})

    # Hitung jumlah produk aktual per kategori
    actual_df = df.groupby('category_name')['product_id'].count().reset_index()
    actual_df.columns = ['category_name', 'actual_total']

    # Urutkan berdasarkan jumlah produk terbanyak
    actual_df = actual_df.sort_values(by='actual_total', ascending=False).reset_index(drop=True)

    # Ambil top 10 kategori
    top_n = actual_df.head(10).copy()

    # Buat time_index (0 - 9) untuk prediksi
    top_n['index'] = np.arange(len(top_n)).reshape(-1, 1)

    # Siapkan model regresi
    model = LinearRegression()
    model.fit(top_n['index'].values.reshape(-1, 1), top_n['actual_total'].values)

    # Prediksi jumlah produk
    top_n['predicted_total'] = model.predict(top_n['index'].values.reshape(-1, 1)).round().astype(int)

    context = {
        'labels': top_n['category_name'].tolist(),
        'actual': top_n['actual_total'].tolist(),
        'predicted': top_n['predicted_total'].tolist()
    }

    return render(request, 'chart_product_category_actual_predicted.html', context)
```

```python
def chart_rating_actual_predicted(request):
    path = os.path.join(settings.BASE_DIR, 'star_output', 'dim_product.csv')
    df = pd.read_csv(path)

    # Pastikan kolom waktu dan rating valid
    df['created_at'] = pd.to_datetime(df['created_at'], errors='coerce')
    df = df.dropna(subset=['created_at', 'rating'])

    # Ambil bulan dari created_at
    df['year_month'] = df['created_at'].dt.to_period('M').astype(str)

    # Hitung rata-rata rating per bulan
    df_summary = df.groupby('year_month')['rating'].mean().reset_index()
    df_summary.columns = ['month', 'avg_rating']

    # Ubah 'month' menjadi fitur numerik (jumlah bulan sejak bulan pertama)
    df_summary['month_num'] = pd.to_datetime(df_summary['month']).dt.month + 12 * (pd.to_datetime(df_summary['month']).dt.year - pd.to_datetime(df_summary['month']).dt.year.min())

    # Model regresi linear untuk prediksi
    X = df_summary['month_num'].values.reshape(-1, 1)  # Menggunakan bulan sebagai fitur numerik
    y = df_summary['avg_rating'].values

    model = LinearRegression()
    model.fit(X, y)

    # Prediksi
    df_summary['predicted_rating'] = model.predict(X).round(2)

    context = {
        'labels': df_summary['month'].tolist(),
        'actual': df_summary['avg_rating'].round(2).tolist(),
        'predicted': df_summary['predicted_rating'].tolist(),
    }

    return render(request, 'chart_rating_actual_predicted.html', context)

def chart_avg_price_actual_predicted(request):
    path = os.path.join(settings.BASE_DIR, 'star_output', 'dim_product.csv')
    df = pd.read_csv(path)

    # Validasi data
    if 'category_name' not in df.columns or 'price' not in df.columns:
        return render(request, 'bi/error.html', {'message': 'Kolom category_name atau price tidak ditemukan'})

    # Grouping harga rata-rata aktual per kategori
    df_actual = df.groupby('category_name')['price'].mean().reset_index()
    df_actual.columns = ['category', 'actual_avg_price']

    # Sort dan buat indeks untuk regresi
    df_actual = df_actual.sort_values(by='actual_avg_price', ascending=False).reset_index(drop=True)
    df_actual['index'] = np.arange(len(df_actual))

    # Linear Regression untuk prediksi harga
    model = LinearRegression()
    X = df_actual['index'].values.reshape(-1, 1)
    y = df_actual['actual_avg_price'].values
    model.fit(X, y)
    df_actual['predicted_avg_price'] = model.predict(X).round()

    context = {
        'labels': df_actual['category'].tolist(),
        'actual': df_actual['actual_avg_price'].astype(int).tolist(),
        'predicted': df_actual['predicted_avg_price'].astype(int).tolist(),
    }

    return render(request, 'chart_avg_price_actual_predicted.html', context)

def dashboard(request):
    return render(request, 'dashboard.html')
```

Sedangkan untuk file html pastikan untuk membuat folder “templates” di dalam folder bi terlebih dahulu dan dapat diisikan kode HTML, ini merupakan template tampilan utama dashboard yang menampilkan hasil visualisasi perbandingan antara data aktual dan hasil prediksi. Dashboard ini terdiri dari empat bagian, masing-masing ditampilkan dalam bentuk iframe yang mengambil data dari URL Django seperti /chart/category-actual-predicted/, /analysis/price/, dan lainnya. Setiap bagian menampilkan grafik interaktif dengan Chart.js yang membandingkan data aktual dan prediksi untuk jumlah produk per kategori, rata-rata harga per bulan, rating produk, dan harga per kategori. Tampilan ini memudahkan pengguna dalam menganalisis performa produk secara visual.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Dashboard Produk - Actual vs Predicted</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <style>
      .chart-container {
        margin-bottom: 50px;
      }
      canvas {
        background: #fff;
        border: 1px solid #ccc;
      }
    </style>
  </head>
  <body>
    <h1>📊 Dashboard Produk (Actual vs Predicted)</h1>

    <div class="chart-container">
      <h3>📈 Jumlah Produk per Kategori (Actual vs Predicted)</h3>
      <iframe
        src="/chart/category-actual-predicted/"
        width="100%"
        height="500"
        frameborder="0"
      ></iframe>
    </div>

    <div class="chart-container">
      <h3>💸 Rata-rata Harga (Actual vs Predicted)</h3>
      <iframe
        src="/analysis/price/"
        width="100%"
        height="500"
        frameborder="0"
      ></iframe>
    </div>

    <div class="chart-container">
      <h3>⭐ Rata-rata Rating per Bulan (Actual vs Predicted)</h3>
      <iframe
        src="/chart/rating-actual-predicted/"
        width="100%"
        height="500"
        frameborder="0"
      ></iframe>
    </div>

    <div class="chart-container">
      <h3>🏷️ Rata-rata Harga per Kategori (Actual vs Predicted)</h3>
      <iframe
        src="/chart/avg-price-actual-predicted/"
        width="100%"
        height="500"
        frameborder="0"
      ></iframe>
    </div>
  </body>
</html>
```



Langkah 9 : Migrasi Data dan Jalankan Django

Setelah semua konfigurasi selesai lakukan migrasi pada django di terminal pastikan posisi berada dalam virtual environment aktif dan berada dalam app_query lakukan migrasi dengan

- python manage.py makemigrations
- python manage.py migrate
- Lalu jalankan djanggo dengan python manage.py runserver

Buka browser dengan alamat http://127.0.0.1:8000

Maka tampilan dashboard akan muncul