



MANUAL BOOK
PRISKA
2025

ALVIN RYAN DANA
TRIMONO
MOHAMMAD IDHOM
DWI ARMAN PRASETYA
AMRI MUHAIMIN

Daftar Isi

Pengenalan	3
1. Home Page	3
2. Upload Data	4
3. Halaman Prediksi	7
4. Halaman VaR	10
Source Code Priska	14

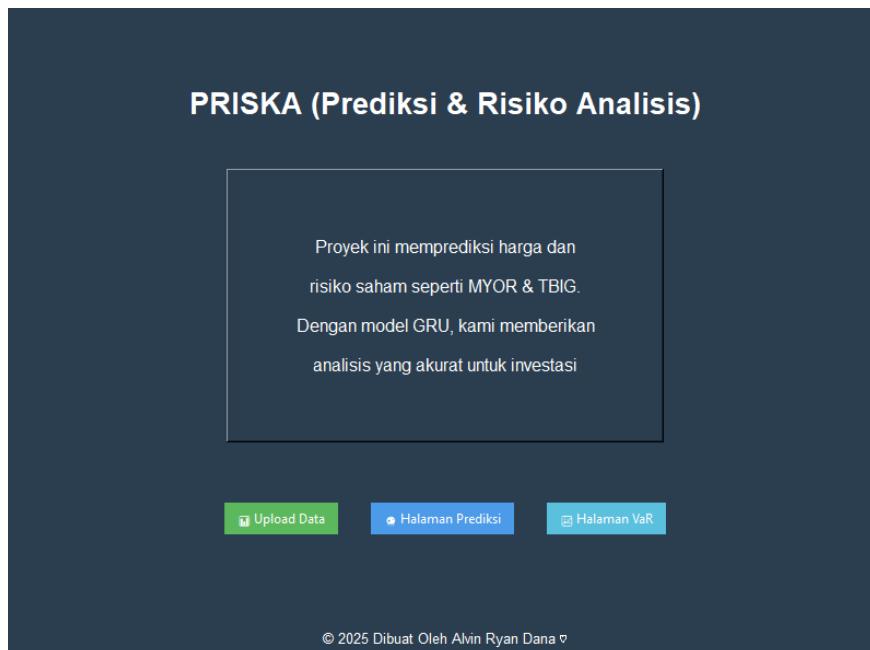
Pengenalan

Dalam dunia investasi, memahami pergerakan harga saham dan risikonya menjadi faktor penting dalam pengambilan keputusan yang tepat. PRISKA (Prediksi & Risiko Analisis) hadir sebagai solusi berbasis kecerdasan buatan yang mampu memprediksi harga serta risiko saham *blue chip*, seperti MYOR & TBIG. Dengan memanfaatkan model GRU (*Gated Recurrent Unit*), PRISKA mampu melakukan analisis data historis secara mendalam, sehingga menghasilkan prediksi yang akurat dan dapat diandalkan oleh investor.

Aplikasi ini dirancang untuk memberikan wawasan berbasis data, membantu investor dalam mengurangi ketidakpastian, serta meningkatkan strategi investasi mereka. Dengan antarmuka yang ramah pengguna, PRISKA memungkinkan siapa saja, baik investor pemula maupun profesional, untuk mengakses informasi dan analisis pasar dengan mudah.

Saat pertama kali mengakses PRISKA, pengguna akan diarahkan ke halaman *home page*. Halaman ini menampilkan gambaran umum aplikasi serta berbagai fitur utama yang dapat digunakan untuk melakukan analisis saham secara mendalam. Pengguna dapat mengunduh PRISKA dan mulai merasakan kemudahan dalam menganalisis saham dengan menggunakan link berikut : <https://bit.ly/4iHl227>

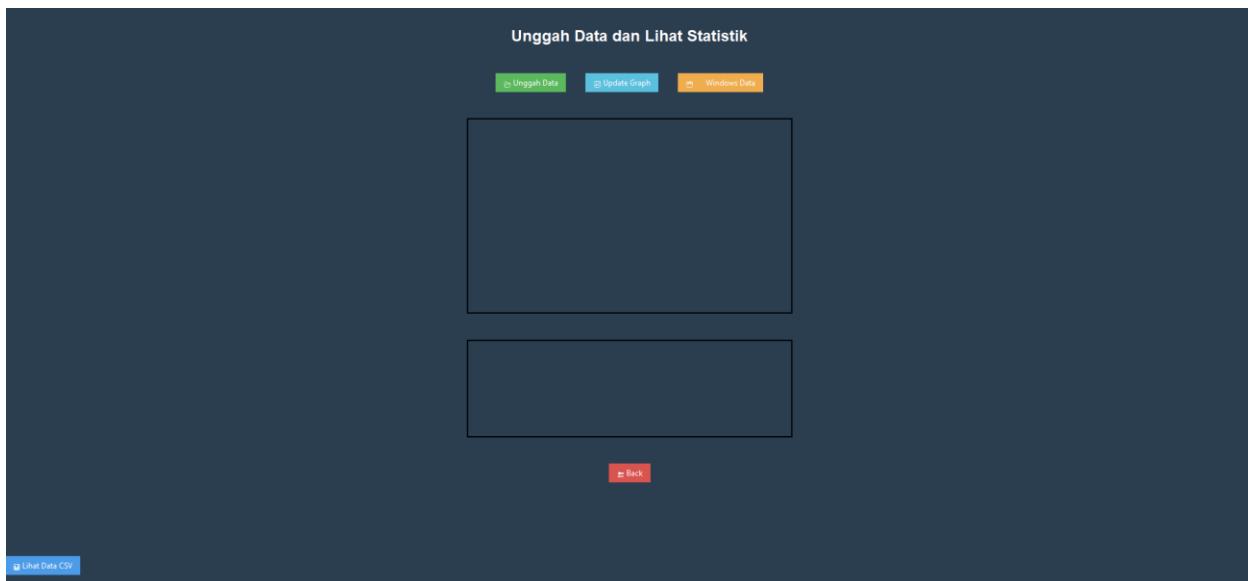
1. Home Page



Saat pertama kali mengakses aplikasi, pengguna akan diarahkan ke halaman *Home*, yang berisi gambaran umum tentang PRISKA serta tiga fitur utama yang dapat digunakan.  Fitur ini memungkinkan pengguna untuk mengunggah dan mempersiapkan data saham yang akan

digunakan dalam proses prediksi. Data yang diunggah akan diproses sehingga dapat dianalisis menggunakan model GRU untuk mendapatkan hasil prediksi yang optimal. [Halaman Prediksi](#)
Pada halaman ini, pengguna dapat melakukan prediksi harga saham di masa depan berdasarkan data yang telah diunggah sebelumnya. Model GRU akan menganalisis pola historis harga saham dan memberikan estimasi tren pergerakan harga ke depan. [Halaman VaR](#) Fitur ini berfungsi untuk memprediksi risiko kerugian dari data saham yang telah diunggah. Dengan menggunakan metode VaR, aplikasi dapat membantu pengguna memahami potensi risiko investasi mereka dalam periode waktu tertentu.

2. Upload Data

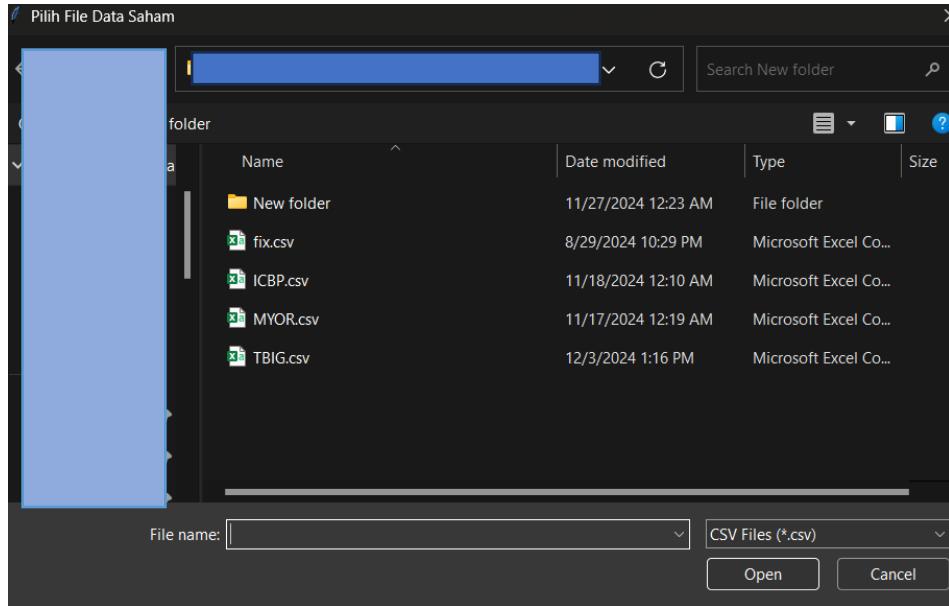


Fitur *Upload Data* pada PRISKA memungkinkan pengguna untuk mengunggah dataset yang akan digunakan dalam proses prediksi saham menggunakan model GRU. Berikut adalah langkah-langkah penggunaannya.

1. Mengunggah Data

- Pengguna menekan tombol [Unggah Data](#).
- Pilih file data yang ingin digunakan. Pastikan file dalam format CSV dan memiliki dua variabel utama, yaitu:
 - Date → Menunjukkan tanggal data saham.
 - Close → Harga penutupan saham pada tanggal tersebut.

- Jika format tidak sesuai, aplikasi mungkin tidak dapat memproses data dengan benar.



2. Memvisualisasikan Data

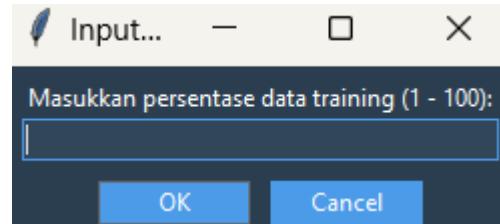
- Setelah data berhasil diunggah, pengguna dapat menekan tombol .
- Aplikasi akan menampilkan grafik visualisasi dari data yang telah diunggah serta analisis deskriptif untuk memahami pola harga saham.



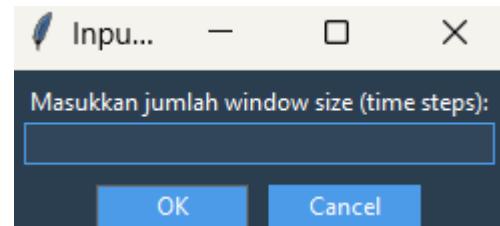
3. Menentukan Parameter Model

- Pengguna dapat menekan tombol , yang akan membuka dua kotak input untuk memasukkan parameter:

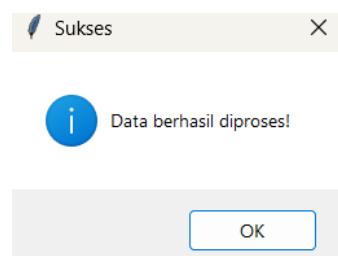
- Percent
ase Data Train → Persentase data yang digunakan untuk pelatihan model.



- Panjang Sequence Data → Jumlah data historis yang akan digunakan sebagai input dalam prediksi harga saham.



- Setelah parameter diatur, model siap untuk melakukan prediksi dengan GRU.



4. Melihat Data yang Diunggah

- Pengguna dapat menekan tombol **Lihat Data CSV** untuk menampilkan data yang telah diunggah ke dalam aplikasi.
- Ini berguna untuk memastikan data sudah benar sebelum melanjutkan ke tahap prediksi.

Data CSV

Date	Close
7/1/2019	770.0
7/2/2019	780.0
7/3/2019	795.0
7/4/2019	800.0
7/5/2019	828.0
7/8/2019	800.0
7/9/2019	805.0
7/10/2019	826.0
7/11/2019	850.0
7/12/2019	846.0
7/15/2019	846.0
7/16/2019	846.0
7/17/2019	836.0
7/18/2019	830.0
7/19/2019	874.0
7/22/2019	876.0
7/23/2019	874.0
7/24/2019	888.0
7/25/2019	896.0
7/26/2019	886.0

X Tutup

Dengan mengikuti alur ini, pengguna dapat dengan mudah menyiapkan data untuk dianalisis dan diprediksi menggunakan PRISKA

Sistem akan secara otomatis menyimpan data *train*, data *test*, dan data *sequence* yang telah ditentukan. Jika pengguna ingin mengubah data saham yang digunakan atau menyesuaikan parameter panjang *sequence*, mereka dapat mengunggah kembali file CSV baru dengan data saham yang berbeda. Selain itu, pengguna juga dapat menekan tombol  Windows Data untuk mengatur ulang persentase data *train* dan panjang *sequence* data tanpa harus mengganti dataset utama. Dengan fitur ini, pengguna memiliki fleksibilitas untuk melakukan eksperimen dengan berbagai skenario prediksi sesuai dengan kebutuhan investasi mereka.

3. Halaman Prediksi

Konfigurasi Dinamis Model Prediksi

Pilih Layer: Konfigurasi Layer

Pilih Optimizer:

Arsitektur Model

Buat Model Train Model Hapus Arsitektur Tempilk Grafik Hapus Prediksi

Prediksi Data Baru

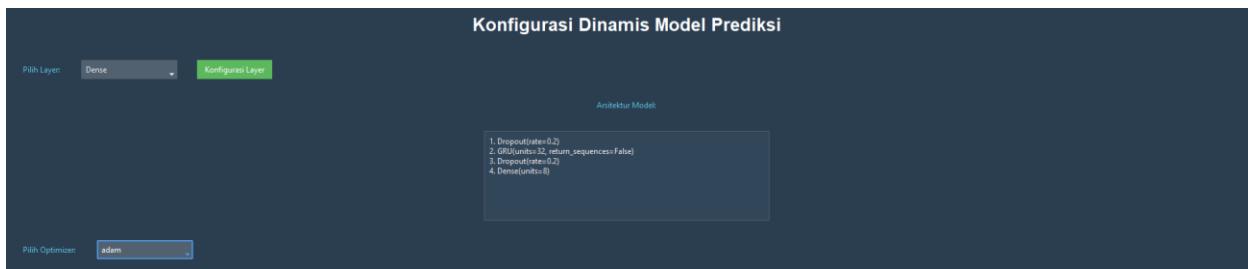
Jumlah langkah prediksi: Prediksi

Jumlah Epochs: Jumlah Unit GRU PerEpoch: Back

Pada halaman ini, pengguna dapat membangun arsitektur jaringan saraf tiruan berbasis GRU secara dinamis untuk melakukan prediksi harga saham. Secara *default*, sistem telah menetapkan *layer input* dengan jumlah unit GRU pertama yang dapat dikonfigurasi oleh pengguna melalui

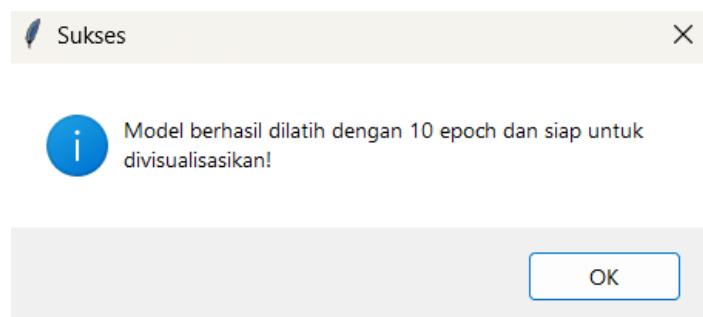
kolom **Jumlah Unit GRU Pertama:** di bagian bawah. Setelah itu, pengguna dapat menambahkan *layer* tambahan dengan memilih jenis layer yang tersedia, yaitu GRU, Dropout, atau Dense, sesuai dengan kebutuhan model mereka.

Setelah arsitektur selesai disusun, pengguna dapat memilih *optimizer* yang akan digunakan untuk pelatihan model. Sistem juga telah menetapkan secara *default layer output* menggunakan Dense dengan unit 1, yang berfungsi untuk menghasilkan prediksi harga saham.



Selanjutnya, pengguna dapat menekan tombol **Buat Model** untuk membangun arsitektur berdasarkan konfigurasi yang telah dibuat. Setelah itu, model dapat dilatih menggunakan tombol **Train Model**, yang akan memproses data dan menyesuaikan bobot jaringan saraf tiruan.

Untuk memantau hasil pelatihan, pengguna dapat menekan tombol **Tampilkan Grafik**, yang akan menampilkan metrik evaluasi model dalam bentuk visualisasi. Selain itu, pengguna juga dapat mengatur jumlah *epoch*, yaitu berapa kali model akan dilatih dengan data yang tersedia. Semakin besar nilai *epoch* yang dimasukkan, semakin lama model akan belajar dari data, yang dapat meningkatkan akurasi tetapi juga berisiko menyebabkan *overfitting*. Pengguna dapat memasukkan nilai epoch yang diinginkan pada kolom **Jumlah Epoch:** di bagian bawah sebelum menjalankan pelatihan model.

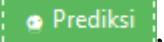


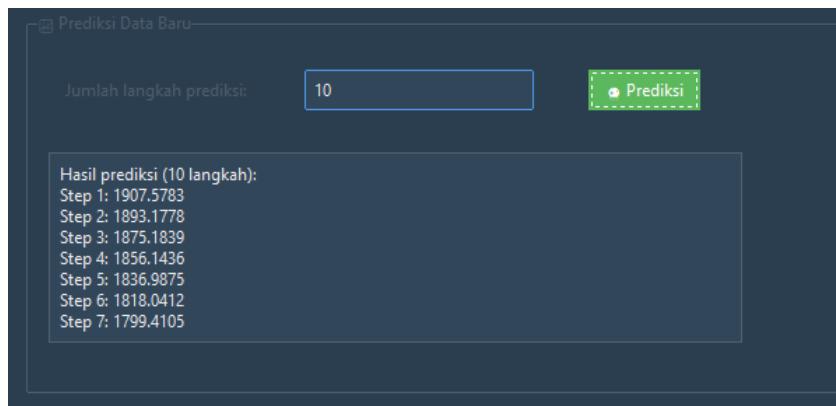
Setelah pengguna menekan tombol **Tampilkan Grafik**, sistem akan menampilkan dua grafik utama yang menggambarkan performa model. Grafik di sebelah kiri menunjukkan *Training*

& *Validation Loss*, di mana garis biru mewakili *training loss* dan garis oranye mewakili *validation loss*. Grafik ini membantu pengguna memahami seberapa baik model belajar dari data dan apakah terjadi *overfitting* atau *underfitting*.

Sementara itu, grafik di sebelah kanan menampilkan **Perbandingan Prediksi vs Data Asli**, di mana garis hijau menunjukkan data aktual, sedangkan garis ungu menunjukkan hasil prediksi model. Grafik ini memungkinkan pengguna mengevaluasi seberapa akurat model dalam memprediksi tren data.

Di bagian bawah, sistem juga menampilkan nilai **MAPE (Mean Absolute Percentage Error)** yang dalam kasus ini adalah 5.68%, menandakan seberapa besar rata-rata kesalahan prediksi dibandingkan dengan nilai sebenarnya. Semakin kecil nilai MAPE, semakin baik performa model dalam melakukan prediksi.

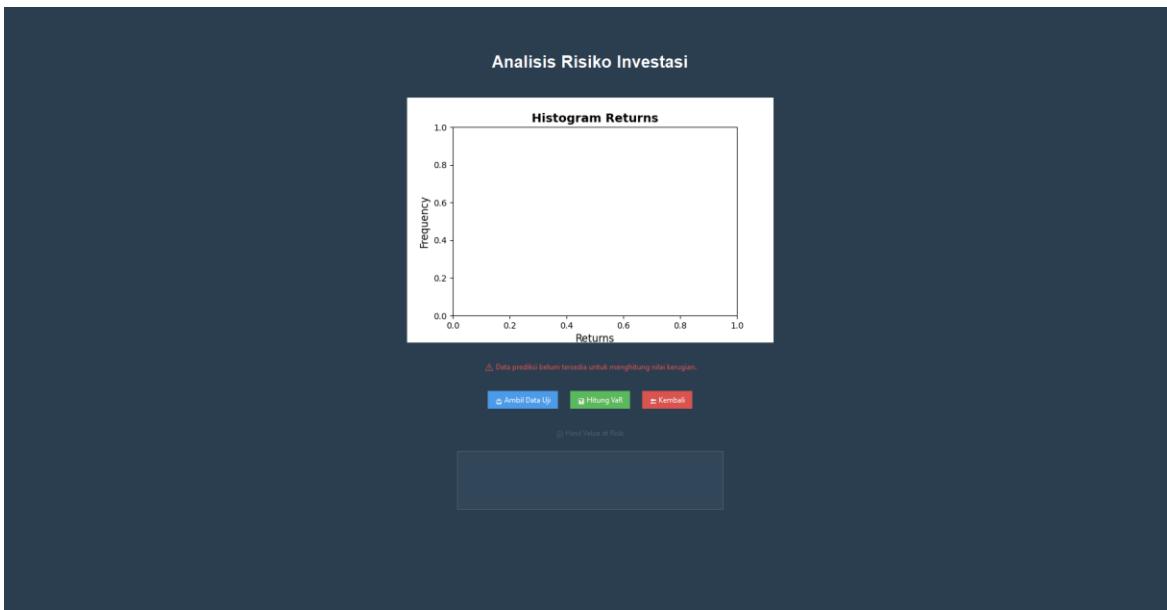
Setelah pengguna menekan tombol  **Prediksi**, sistem akan menghasilkan serangkaian nilai yang mewakili hasil prediksi berdasarkan model yang telah dilatih. Pada contoh berikut, pengguna telah memilih untuk memprediksi 10 langkah ke depan, dan sistem menampilkan hasil prediksi untuk setiap langkah secara berurutan.



Setiap *Step* menunjukkan nilai prediksi untuk titik waktu berikutnya, dengan angka yang semakin berkurang seiring waktu, yang mungkin menunjukkan tren penurunan dalam data yang dianalisis. Pengguna dapat mengubah jumlah langkah prediksi dengan memasukkan angka yang berbeda di kolom input sebelum menekan tombol  **Prediksi** kembali.

Prediksi ini dapat digunakan untuk menganalisis tren masa depan berdasarkan pola yang telah dipelajari oleh model. Jika hasil prediksi dirasa kurang akurat, pengguna dapat mencoba menyesuaikan arsitektur model, jumlah *epoch* saat pelatihan, atau melakukan tuning parameter lainnya untuk meningkatkan performa model.

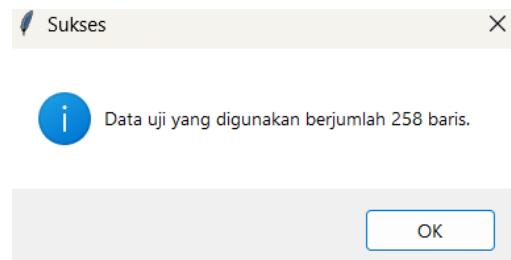
4. Halaman VaR



Fitur ini memungkinkan pengguna untuk menganalisis risiko investasi berdasarkan data prediksi yang telah dihasilkan sebelumnya. Berikut adalah tahapan penggunaannya:

1. Ambil Data Uji

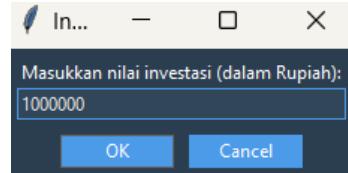
- Pengguna menekan tombol **Ambil Data Uji** untuk mengambil data uji yang tersimpan dalam sistem.
- Data ini mencakup data uji yang telah ditambahkan dengan hasil prediksi sebelumnya.



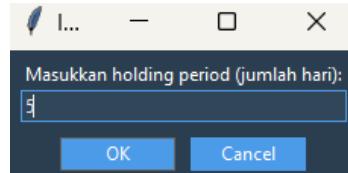
2. Hitung Value at Risk (VaR)

- Setelah data tersedia, pengguna dapat menekan tombol **Hitung VaR** untuk menghitung potensi risiko investasi.
- Sistem akan menampilkan jendela input yang meminta pengguna memasukkan beberapa parameter penting:

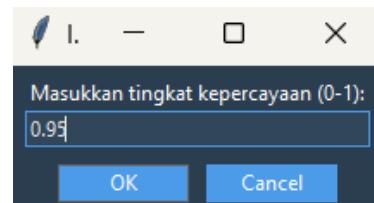
- Nilai Investasi (dalam Rupiah): Pengguna memasukkan jumlah modal yang diinvestasikan.



- Holding Period (jumlah hari): Periode waktu investasi yang ingin dianalisis.

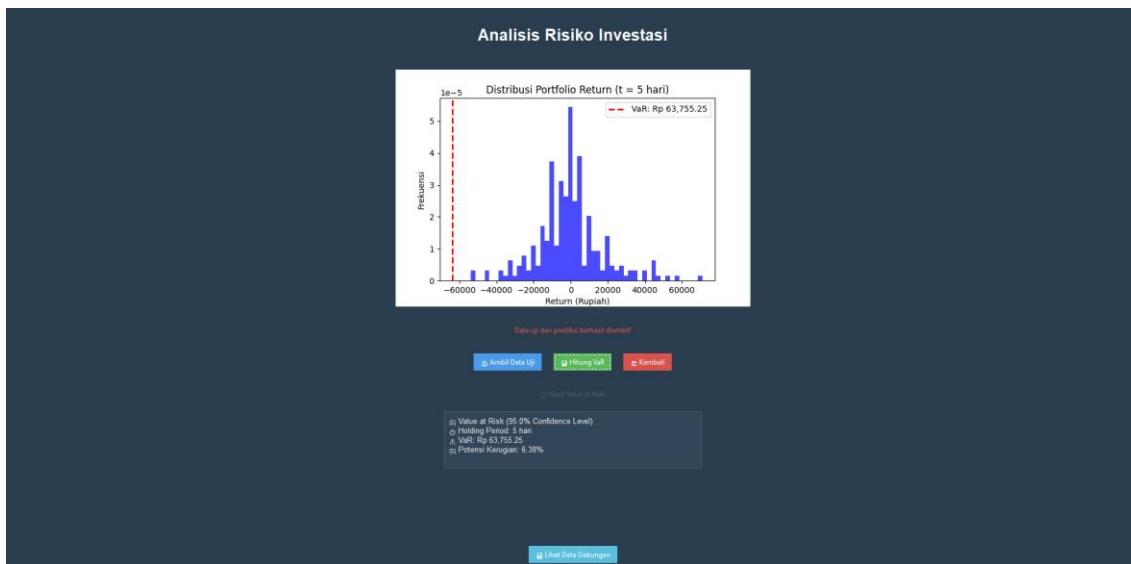


- Tingkat Kepercayaan (0-1): Probabilitas dalam menentukan batas kerugian maksimal yang dapat diterima.



3. Hasil Analisis Risiko

- Setelah semua parameter diinput, sistem akan menghitung Value at Risk (VaR) dan menampilkan hasilnya.



- Hasil ini membantu pengguna memahami kemungkinan kerugian dalam investasi mereka dalam periode yang telah ditentukan

4. Lihat Data Gabungan

- Setelah mendapatkan hasil analisis, pengguna akan melihat tombol **Lihat Data Gabungan**.
- Dengan menekan tombol ini, pengguna dapat melihat tabel yang menampilkan data tes dan data prediksi secara bersamaan.



The screenshot shows a modal window titled "Data Uji dan Prediksi". The window has a dark blue header bar with the title and standard close/minimize/maximize buttons. Below the header is a table with two columns: "Index" and "Close Price". The table contains 15 rows of data. At the bottom of the window is a red "Tutup" button. The table data is as follows:

Index	Close Price
0	1960.0
1	1955.0
2	1960.0
3	1980.0
4	2020.0
5	2040.0
6	1975.0
7	1910.0
8	1890.0
9	1930.0
10	1875.0
11	1900.0
12	1900.0
13	1950.0
14	1950.0

- Hal ini memungkinkan pengguna untuk membandingkan nilai aktual dengan hasil prediksi guna memahami pola pergerakan data lebih baik.



SOURCE CODE
PRISKA
2025

ALVIN RYAN DANA
TRIMONO
MOHAMMAD IDHOM
DWI ARMAN PRASETYA
AMRI MUHAIMIN

Source Code Priska

```
import tkinter as tk
from tkinter import ttk, filedialog
from tkinter.filedialog import askopenfilename
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dropout, Dense
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tkinter import ttk, messagebox
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tkinter import ttk
from tkinter import Frame
import tkinter as tk
from ttkbootstrap import Style
from matplotlib.figure import Figure
from tkinter import simpledialog

class AplikasiSaham(tk.Tk):
    def __init__(self):
        super().__init__()
        self.prediksi_terakhir = []
        self.title("Aplikasi Prediksi Saham dan Risiko Investasi")
        self.geometry("800x600") # Ukuran Window
        self.center_window(800, 600) # Pusatkan Window

        # Gaya Bootstrap
        self.style = Style(theme="superhero")

        self.min_values = None
        self.max_values = None
        # Frame Halaman
```

```

self.halaman_utama = HalamanUtama(self)
self.halaman_unggah = HalamanUnggah(self)
self.halaman_prediksi = HalamanPrediksi(self)
self.halaman_risiko = HalamanRisiko(self)

self.halaman_utama.pack(fill="both", expand=True)

def center_window(self, width, height):
    screen_width = self.winfo_screenwidth()
    screen_height = self.winfo_screenheight()
    x = (screen_width // 2) - (width // 2)
    y = (screen_height // 2) - (height // 2)
    self.geometry(f"{width}x{height}+{x}+{y}")

def buka_halaman_unggah(self):
    self.halaman_utama.pack_forget()
    self.halaman_prediksi.pack_forget()
    self.halaman_risiko.pack_forget()
    self.halaman_unggah.pack(fill="both", expand=True)

def kembali_ke_halaman_utama(self):
    self.halaman_unggah.pack_forget()
    self.halaman_prediksi.pack_forget()
    self.halaman_risiko.pack_forget()
    self.halaman_utama.pack(fill="both", expand=True)

def buka_halaman_prediksi(self):
    self.halaman_unggah.pack_forget()
    self.halaman_utama.pack_forget()
    self.halaman_risiko.pack_forget()
    self.halaman_prediksi.pack(fill="both", expand=True)

def buka_halaman_risiko(self):
    self.halaman_utama.pack_forget()
    self.halaman_unggah.pack_forget()
    self.halaman_prediksi.pack_forget()

```

```

self.halaman_risiko.pack(fill="both", expand=True)

class HalamanUtama(Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg=parent.style.colors.bg)

    tk.Label(
        self, text="Aplikasi Prediksi Saham dan Risiko Investasi",
        font=("Helvetica", 20, "bold"), # Font lebih besar dan tebal
        bg=parent.style.colors.bg,
        fg=parent.style.colors.primary
    ).pack(pady=(40, 20))

    frame_gambar = tk.Frame(self, width=400, height=250, bg="white", relief="raised",
                           borderwidth=2)
    frame_gambar.pack(pady=20)

    teks_proyek = tk.Label(
        frame_gambar,
        text="Proyek ini memprediksi harga dan\nrisiko saham blue chip seperti MYOR & TBIG.\nDengan model GRU, kami memberikan\nanalisis yang akurat untuk investasi",
        font=("Arial", 12),
        bg="white",
        justify="center"
    )
    teks_proyek.place(relx=0.5, rely=0.5, anchor="center")

    # Frame untuk tombol dengan spacing
    button_frame = tk.Frame(self, bg=parent.style.colors.bg)
    button_frame.pack(pady=30)

    btn_unggah = ttk.Button(button_frame, text=" <img alt='Upload icon' data-bbox='558 858 578 878' style='vertical-align: middle; height: 1em;"/> Upload Data",
                           style="success.TButton",

```

```

        command=self.parent.buka_halaman_unggah)
btn_unggah.pack(side="left", padx=15, pady=5)

btn_prediksi = ttk.Button(button_frame, text="  Halaman Prediksi",
                         style="primary.TButton",
                         command=self.parent.buka_halaman_prediksi)
btn_prediksi.pack(side="left", padx=15, pady=5)

btn_var = ttk.Button(button_frame, text="  Halaman VaR",
                     style="info.TButton",
                     command=self.parent.buka_halaman_risiko)
btn_var.pack(side="left", padx=15, pady=5)

# Tambahan footer atau copyright
tk.Label(
    self, text="© 2025 Dibuat Oleh Alvin Ryan Dana ♡",
    font=("Arial", 10),
    bg=parent.style.colors.bg,
    fg=parent.style.colors.secondary
).pack(side="bottom", pady=10)

class HalamanUnggah(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.file_path = None
        self.data_ready = False

        style = Style(theme="darkly")

        tk.Label(self, text="Unggah Data dan Lihat Statistik", font=("Helvetica", 18, "bold"), fg="white", bg="black").pack(pady=20)

        # Frame untuk tombol dengan padding
        button_frame = tk.Frame(self, bg="black")

```

```

button_frame.pack(pady=20)

# Tombol Unggah dengan ikon dan padding
    btn_unggah_file = ttk.Button(button_frame, text="📁 Unggah Data",
command=self.unggah_file, style="success.TButton")
    btn_unggah_file.pack(side="left", padx=15)

# Tombol Update Graph dengan ikon dan padding
    btn_update = ttk.Button(button_frame, text="📈 Update Graph",
command=self.update_graph, style="info.TButton")
    btn_update.pack(side="left", padx=15)

# Tombol Windows Data dengan ikon dan padding
    btn_windows_data = ttk.Button(button_frame, text="📁 Windows Data",
command=self.process_data, style="warning.TButton")
    btn_windows_data.pack(side="left", padx=15)

# Placeholder untuk Grafik dengan border dan warna modern
    self.canvas_frame = tk.Frame(self, width=500, height=300, bg="white",
relief="solid", borderwidth=2)
    self.canvas_frame.pack(pady=20)

# Placeholder untuk Statistik Deskriptif dengan border dan warna modern
    self.stats_frame = tk.Frame(self, width=500, height=150, bg="white", relief="solid",
borderwidth=2)
    self.stats_frame.pack(pady=20)

# Tombol Back dengan padding
    btn_back = ttk.Button(self, text="⬅ BACK",
command=self.parent.kembali_ke_halaman_utama, style="danger.TButton")
    btn_back.pack(pady=20)

    btn_view_csv = ttk.Button(self, text="📊 Lihat Data CSV",
command=self.open_csv_window, style="primary.TButton")
    btn_view_csv.pack(side="left", pady=20)

```

```

def unggah_file(self):
    self.file_path = filedialog.askopenfilename(
        title="Pilih File Data Saham",
        filetypes=((("CSV Files", "*.csv"), ("All Files", "*.*")))
    )
    if self.file_path:
        data = pd.read_csv(self.file_path)
        #self.parent.min_values = data['Close'].min()
        #self.parent.max_values = data['Close'].max()

def update_graph(self):
    if not self.file_path:
        messagebox.showerror("Error", "Silakan unggah file terlebih dahulu.")
        return

    try:
        data = pd.read_csv(self.file_path)
        if 'Date' not in data.columns or 'Close' not in data.columns:
            messagebox.showerror("Error", "File harus memiliki kolom 'Date' dan 'Close'.")
            return

        data['Date'] = pd.to_datetime(data['Date'])
        data.sort_values('Date', inplace=True)

        for widget in self.canvas_frame.winfo_children():
            widget.destroy()
        for widget in self.stats_frame.winfo_children():
            widget.destroy()

        fig, ax = plt.subplots(figsize=(6, 4))
        ax.plot(data['Date'], data['Close'], label="Close Price", color='blue')
        ax.set_title("Closing Prices Over Time", fontsize=14)
        ax.set_xlabel("Date", fontsize=12)
        ax.set_ylabel("Price", fontsize=12)
        ax.legend()
    
```

```

    canvas = FigureCanvasTkAgg(fig, master=self.canvas_frame)
    canvas_widget = canvas.get_tk_widget()
    canvas_widget.pack(fill="both", expand=True)
    canvas.draw()

    stats = data['Close'].describe()
    stats_text = (
        f"Statistika Deskriptif:\n"
        f"- Count: {stats['count']}\n"
        f"- Mean: {stats['mean']:.2f}\n"
        f"- Std: {stats['std']:.2f}\n"
        f"- Min: {stats['min']:.2f}\n"
        f"- 25%: {stats['25%']:.2f}\n"
        f"- 50% (Median): {stats['50%']:.2f}\n"
        f"- 75%: {stats['75%']:.2f}\n"
        f"- Max: {stats['max']:.2f}"
    )
    stats_label = tk.Label(self.stats_frame, text=stats_text, font=("Arial", 12),
justify="left", bg="white")
    stats_label.pack(anchor="w", padx=10, pady=10)

except Exception as e:
    messagebox.showerror("Error", f"Terjadi kesalahan saat memproses file:\n{e}")

def process_data(self):
    if not self.file_path:
        messagebox.showerror("Error", "Silakan unggah file terlebih dahulu.")
        return
    try:
        data = pd.read_csv(self.file_path)
        if 'Close' not in data.columns:
            messagebox.showerror("Error", "File harus memiliki kolom 'Close'.")
            return
        train_percentage = simpledialog.askinteger(
            "Input Persentase",

```

```

    "Masukkan persentase data training (1 - 100):",
    minvalue=1, maxvalue=100
)

if train_percentage is None: # Jika user menekan "Cancel"
    return

time_steps = simpledialog.askinteger(
    "Input Window Size",
    "Masukkan jumlah window size (time steps):",
    minvalue=1, maxvalue=100
)
if time_steps is None: # Jika user menekan "Cancel"
    return

train_ratio = train_percentage / 100
features = ['Close']
df = data[features]
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)

def create_sequences(data, window_size=6):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i + window_size])
        y.append(data[i + window_size][0])
    return np.array(X), np.array(y)

self.X_train_value = time_steps
X, y = create_sequences(scaled_data, time_steps)

split = int(train_ratio * len(X))
self.X_train, self.X_test = X[:split], X[split:]
self.y_train, self.y_test = y[:split], y[split:]

```

```

        self.train_var, self.test_var = train_test_split(df['Close'], test_size=1-train_ratio,
shuffle=False)

        self.data_ready = True
        self.parent.min_values = scaler.data_min_
        self.parent.max_values = scaler.data_max_
        messagebox.showinfo("Sukses", "Data berhasil diproses!")

    except Exception as e:
        messagebox.showerror("Error", f"Terjadi kesalahan saat memproses data:\n{e}")

def open_csv_window(self):
    if not self.file_path:
        messagebox.showerror("Error", "Silakan unggah file CSV terlebih dahulu!")
        return

    import pandas as pd
    data = pd.read_csv(self.file_path)

    # Membuat jendela baru dengan tema gelap
    new_window = tk.Toplevel(self)
    new_window.title("Data CSV")
    new_window.geometry("800x500")
    new_window.configure(bg="#2c2f33") # Warna latar belakang gelap

    # Judul jendela yang diperbesar
    tk.Label(new_window, text="Data CSV", font=("Helvetica", 20, "bold"), fg="white",
bg="#2c2f33").pack(pady=10)

    # Frame untuk Treeview dengan padding dan warna gelap
    frame_treeview = tk.Frame(new_window, bg="#2c2f33")
    frame_treeview.pack(expand=True, fill="both", padx=20, pady=20)

    # Menambahkan Scrollbar horizontal dan vertikal
    tree_scroll_y = ttk.Scrollbar(frame_treeview, orient="vertical")
    tree_scroll_y.pack(side="right", fill="y")

```

```

tree_scroll_x = ttk.Scrollbar(frame_treeview, orient="horizontal")
tree_scroll_x.pack(side="bottom", fill="x")

# Membuat Treeview untuk menampilkan seluruh data dengan scrollbar
tree = ttk.Treeview(frame_treeview, yscrollcommand=tree_scroll_y.set,
xscrollcommand=tree_scroll_x.set)
tree.pack(expand=True, fill="both")

tree_scroll_y.config(command=tree.yview)
tree_scroll_x.config(command=tree.xview)

# Menambahkan kolom dan header dengan gaya bold
tree["columns"] = list(data.columns)
tree["show"] = "headings"

# Menambahkan gaya kustom untuk header
style = ttk.Style()
    style.configure("Treeview.Heading", font=("Helvetica", 12, "bold"),
background="#7289da", foreground="white")
    style.configure("Treeview", font=("Helvetica", 10), background="#23272a",
foreground="white", fieldbackground="#2c2f33")

for col in data.columns:
    tree.heading(col, text=col)
    tree.column(col, anchor="center")

# Menambahkan seluruh data ke Treeview
for index, row in data.iterrows():
    tree.insert("", "end", values=list(row))

# Tombol keluar dengan warna merah
    btn_close = ttk.Button(new_window, text="✖ Tutup",
command=new_window.destroy, style="danger.TButton")
    btn_close.pack(pady=10)

```

```

class HalamanPrediksi(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.model = None
        self.layer_config = []

        # Gunakan style tkinterbootstrap
        style = Style(theme="superhero")

        # Frame utama dengan padding dan tata letak rapi
        container = ttk.Frame(self, padding=20)
        container.pack(fill="both", expand=True)

        self.notifikasi = tk.Label(self.parent, text="", font=("Arial", 10), fg="red")
        self.notifikasi.pack(pady=5)

        # Judul
        ttk.Label(container, text="Konfigurasi Dinamis Model Prediksi",
            font=("Helvetica", 20, "bold")).pack(pady=20)

        # Pilihan Layer dengan spacing yang lebih baik
        layer_frame = ttk.Frame(container)
        layer_frame.pack(fill="x", pady=15)
        ttk.Label(layer_frame, text="Pilih Layer:", style="info.TLabel").grid(row=0,
        column=0, padx=15, sticky="e")

        self.layer_choice = ttk.Combobox(layer_frame, values=["GRU", "Dropout",
        "Dense"], state="readonly")
        self.layer_choice.grid(row=0, column=1, padx=15, sticky="w")
        ttk.Button(layer_frame, text="Konfigurasi Layer", style="success.TButton",
            command=self.konfigurasi_layer).grid(row=0, column=2, padx=15,
        sticky="w")

        # Preview Layer dengan style modern

```

```

ttk.Label(container, text="Arsitektur Model:", style="info.TLabel").pack(pady=15)
    self.layer_preview = tk.Text(container, width=70, height=8, state="disabled",
wrap="word", bg="#f8f9fa",
                    relief="solid", borderwidth=2)
    self.layer_preview.pack(padx=20, pady=15)

# Pilihan Optimizer dengan penataan yang lebih rapi
optimizer_frame = ttk.Frame(container)
optimizer_frame.pack(fill="x", pady=15)
    ttk.Label(optimizer_frame,      text="Pilih      Optimizer:",
style="info.TLabel").grid(row=0, column=0, padx=15, sticky="e")
    self.optimizer_choice = ttk.Combobox(optimizer_frame, values=["adam", "sgd",
"rmsprop"], state="readonly")
    self.optimizer_choice.grid(row=0, column=1, padx=15, sticky="w")

# Tombol dengan ikon dan padding
button_frame = ttk.Frame(container)
button_frame.pack(fill="x", pady=20)
    ttk.Button(button_frame, text="🛠️ Buat Model", style="success.TButton",
command=self.buat_model).grid(row=0, column=0, padx=10, pady=5)
    ttk.Button(button_frame, text="📊 Train Model", style="info.TButton",
command=self.train_model).grid(row=0, column=1, padx=10, pady=5)
    ttk.Button(button_frame, text="🗑️ Hapus Arsitektur", style="danger.TButton",
command=self.hapus_arsitektur).grid(row=0, column=2, padx=10, pady=5)
    ttk.Button(button_frame, text="📈 Tampilkan Grafik", style="info.TButton",
command=self.open_loss_comparison_window).grid(row=0,     column=3,     padx=10,
pady=5)
    ttk.Button(button_frame, text="ลบ Hapus Prediksi", style="danger.TButton",
command=self.hapus_prediksi).grid(row=0, column=4, padx=10, pady=5)

# Prediksi Data Baru dengan frame yang lebih modern
prediksi_frame = ttk.Labelframe(container, text="📈 Prediksi Data Baru",
padding=15, style="secondary.TLabelframe")
prediksi_frame.pack(fill="both", expand=True, padx=20, pady=20)

```

```

        ttk.Label(prediksi_frame,   text="Jumlah langkah prediksi:",  

style="secondary.TLabel").grid(row=0, column=0, sticky="w", padx=10, pady=10)  

        self.input_steps = ttk.Entry(prediksi_frame, width=25)  

        self.input_steps.grid(row=0, column=1, sticky="w", padx=10, pady=10)  

        ttk.Button(prediksi_frame,  text="🔮 Prediksi", style="success.TButton",  

command=self.predict_future).grid(row=0, column=2, padx=10, pady=10, sticky="w")  
  

        self.prediksi_output = tk.Text(prediksi_frame, height=8, width=80, state="disabled",  

wrap="word", bg="#f8f9fa",  

                relief="solid", borderwidth=2)  

        self.prediksi_output.grid(row=1, column=0, columnspan=3, pady=20)  

epoch_frame = ttk.Frame(container)  

epoch_frame.pack(fill="x", pady=15)  
  

        ttk.Label(epoch_frame, text="Jumlah Epoch:", style="info.TLabel").grid(row=0,  

column=0, padx=15, sticky="e")  
  

        self.epoch_input = ttk.Entry(epoch_frame, width=10)  

self.epoch_input.grid(row=0, column=1, padx=15, sticky="w")  
  

gru_frame = ttk.Frame(container)  

gru_frame.pack(fill="x", pady=15)  
  

        ttk.Label(gru_frame,   text="Jumlah Unit GRU Pertama:",  

style="info.TLabel").grid(row=0, column=0, padx=15, sticky="e")  

        self.btn_hapus_prediksi = ttk.Button(self, text="ลบ Hapus Prediksi",  

                command=self.hapus_prediksi, style="danger.TButton")  

self.btn_hapus_prediksi.pack(pady=10)  
  

# Entry untuk jumlah unit GRU pertama  

        self.gru_units_input = ttk.Entry(gru_frame, width=10)  

self.gru_units_input.grid(row=0, column=1, padx=15, sticky="w")  

self.gru_units_input.insert(0, "50") # Nilai default  

# Tombol untuk menampilkan grafik Loss dan Prediksi  

# Tombol Kembali dipastikan berada di dalam layar

```

```

        ttk.Button(gru_frame, text="⬅️ Back", style="danger.TButton",
command=self.parent.kembali_ke_halaman_utama).grid(row=0, column=2, padx=20,
sticky="e")

def check_data(self):
    messagebox.showinfo("Data Check", "Fitur ini belum dikonfigurasi sepenuhnya.")

def konfigurasi_layer(self):
    layer_type = self.layer_choice.get()
    if not layer_type:
        messagebox.showerror("Error", "Pilih jenis layer terlebih dahulu.")
        return
    if layer_type == "GRU":
        self.konfigurasi_gru()
    elif layer_type == "Dropout":
        self.konfigurasi_dropout()
    elif layer_type == "Dense":
        self.konfigurasi_dense()

def konfigurasi_gru(self):
    def tambahkan_gru():
        try:
            units = int(entry_units.get())
            return_sequences = var_return_sequences.get()
            self.layer_config.append({"type": "GRU", "units": units, "return_sequences": return_sequences})
            self.update_preview()
            window.destroy()
        except ValueError:
            messagebox.showerror("Error", "Masukkan jumlah unit GRU yang valid.")

    window = tk.Toplevel(self)
    window.title("Konfigurasi GRU")
    tk.Label(window, text="Jumlah Unit:").pack(pady=5)
    entry_units = tk.Entry(window, width=10)
    entry_units.pack(pady=5)

```

```

var_return_sequences = tk.BooleanVar(value=False)
ttk.Checkbutton(window,      text="Return Sequences",
variable=var_return_sequences).pack(pady=5)

ttk.Button(window, text="Tambahkan", command=tambahkan_gru).pack(pady=10)

def konfigurasi_dropout(self):
    def tambahkan_dropout():
        try:
            rate = float(entry_rate.get())
            if not (0 <= rate <= 1):
                raise ValueError
            self.layer_config.append({"type": "Dropout", "rate": rate})
            self.update_preview()
            window.destroy()
        except ValueError:
            messagebox.showerror("Error", "Masukkan dropout rate yang valid (0-1.)")

window = tk.Toplevel(self)
window.title("Konfigurasi Dropout")
tk.Label(window, text="Dropout Rate (0-1):").pack(pady=5)
entry_rate = tk.Entry(window, width=10)
entry_rate.pack(pady=5)
ttk.Button(window,      text="Tambahkan",
command=tambahkan_dropout).pack(pady=10)

def konfigurasi_dense(self):
    def tambahkan_dense():
        try:
            units = int(entry_units.get())
            self.layer_config.append({"type": "Dense", "units": units})
            self.update_preview()
            window.destroy()
        except ValueError:
            messagebox.showerror("Error", "Masukkan jumlah unit Dense yang valid.")

window = tk.Toplevel(self)

```

```

window.title("Konfigurasi Dense")
tk.Label(window, text="Jumlah Unit:").pack(pady=5)
entry_units = tk.Entry(window, width=10)
entry_units.pack(pady=5)
ttk.Button(window, text="Tambahkan", command=tambahkan_dense).pack(pady=10)

def update_preview(self):
    self.layer_preview.config(state="normal")
    self.layer_preview.delete("1.0", "end")
    for i, layer in enumerate(self.layer_config):
        if layer["type"] == "GRU":
            text = f"{i + 1}. GRU(units={layer['units']},"
return_sequences={layer['return_sequences']}\\n"
        elif layer["type"] == "Dropout":
            text = f"{i + 1}. Dropout(rate={layer['rate']})\\n"
        elif layer["type"] == "Dense":
            text = f"{i + 1}. Dense(units={layer['units']})\\n"
            self.layer_preview.insert("end", text)
    self.layer_preview.config(state="disabled")

def tambahkan_layer_input(self):
    """
    Tambahkan layer pertama ke model
    """
    timesteps = int(self.gru_units_input.get())
    X_train2 = self.parent.halaman_unggah.X_train_value
    self.model.add(GRU(units=timesteps, return_sequences="TRUE",
input_shape=(X_train2, 1))

def tambahkan_layer_output(self):
    """
    Tambahkan layer terakhir ke model (Dense dengan 1 unit dan linear activation).
    """
    self.model.add(Dense(units=1, activation='linear'))

```

```

def buat_model(self):
    if not self.layer_config:
        messagebox.showerror("Error", "Tambahkan setidaknya satu layer.")
        return

    optimizer_name = self.optimizer_choice.get()
    if optimizer_name not in ["adam", "sgd", "rmsprop"]:
        tk.messagebox.showerror("Error", "Pilih optimizer yang valid.")
        return

    try:
        try:
            gru_units = int(self.gru_units_input.get())
            if gru_units <= 0:
                raise ValueError
            except ValueError:
                tk.messagebox.showerror("Error", "Jumlah unit GRU harus berupa angka positif!")
        return

        # Hapus model lama jika ada
        if self.model is not None:
            del self.model
            self.model = None
            gc.collect() # Bersihkan memori

        self.model = Sequential()

        self.tambahkan_layer_input()

        # Tambahkan semua layer yang dikonfigurasi
        for layer in self.layer_config:
            if layer["type"] == "GRU":
                self.model.add(GRU(units=layer["units"]),
                return_sequences=layer["return_sequences"]))
            elif layer["type"] == "Dropout":
                self.model.add(Dropout(rate=layer["rate"]))

```

```

elif layer["type"] == "Dense":
    self.model.add(Dense(units=layer["units"], activation='relu'))

# Tambahkan layer output secara eksplisit
self.tambahkan_layer_output()

# Pilih dan kompilasi optimizer
optimizer = {"adam": Adam(), "sgd": SGD(), "rmsprop": RMSprop()}[optimizer_name]
self.model.compile(optimizer=optimizer, loss="mse", metrics=["mae"])

print("\n==== Arsitektur Model Setelah Training ===")
self.model.summary()

tk.messagebox.showinfo("Sukses", "Model berhasil dibuat dan dikompilasi!")
except Exception as e:
    tk.messagebox.showerror("Error", f"Terjadi kesalahan: {e}")

def open_loss_comparison_window(self):
    if not hasattr(self, 'history') or not hasattr(self, 'y_test_predict'):
        tk.messagebox.showerror("Error", "Model belum dilatih atau data prediksi tidak tersedia.")
    return

    mape = np.mean(np.abs((self.y_test_predict - self.test_predict[:, 0]) / self.y_test_predict)) * 100

# Membuat jendela baru
new_window = tk.Toplevel(self)
new_window.title("Grafik Loss dan Perbandingan Prediksi")
new_window.geometry("900x600")
new_window.configure(bg="#2c2f33")

# Membuat figure matplotlib
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
fig.patch.set_facecolor('#2c2f33')

```

```

# Plot Loss Training dan Validasi
train_loss = self.history.history['loss']
val_loss = self.history.history['val_loss']

axes[0].plot(train_loss, label='Training Loss', color='cyan')
axes[0].plot(val_loss, label='Validation Loss', color='orange')
axes[0].set_title('Training & Validation Loss', fontsize=14, color='white')
axes[0].set_xlabel('Epochs')
axes[0].set_ylabel('Loss')
axes[0].legend()
axes[0].grid(True)
axes[0].set_facecolor('#23272a')
axes[0].tick_params(colors='white')

# Plot Perbandingan Prediksi dan Data Asli
axes[1].plot(self.y_test_predict, label='Actual Data', color='lime')
axes[1].plot(self.test_predict[:, 0], label='Predictions', color='magenta')
axes[1].set_title('Perbandingan Prediksi vs Data Asli', fontsize=14, color='white')
axes[1].set_xlabel('Data Index')
axes[1].set_ylabel('Price')
axes[1].legend()
axes[1].grid(True)
axes[1].set_facecolor('#23272a')
axes[1].tick_params(colors='white')

# Menampilkan figure di Tkinter dengan Matplotlib Canvas
canvas = FigureCanvasTkAgg(fig, master=new_window)
canvas.draw()
canvas.get_tk_widget().pack(expand=True, fill="both")

    mape_label = ttk.Label(new_window, text=f"MAPE: {mape:.2f}%", font=("Helvetica", 14, "bold"), foreground="white", background="#2c2f33")
    mape_label.pack(pady=10)
    # Tombol tutup

```

```

        ttk.Button(new_window, text="✖ Tutup", command=new_window.destroy,
style="danger.TButton").pack(pady=10)

def train_model(self):
    if self.model is None:
        tk.messagebox.showerror("Error", "Model belum dibuat.")
        return

        if not hasattr(self.parent.halaman_unggah, 'data_ready') or not
self.parent.halaman_unggah.data_ready:
            tk.messagebox.showerror("Error", "Data belum siap! Silakan proses data di
halaman unggah.")
        return

    X_train = self.parent.halaman_unggah.X_train
    y_train = self.parent.halaman_unggah.y_train
    X_test = self.parent.halaman_unggah.X_test
    y_test = self.parent.halaman_unggah.y_test

    try:
        epochs = int(self.epoch_input.get())
        if epochs <= 0:
            raise ValueError
    except ValueError:
        tk.messagebox.showerror("Error", "Jumlah epoch harus berupa angka positif!")
        return

# Latih model dengan jumlah epoch yang dipilih user
    self.history = self.model.fit(X_train, y_train, epochs=epochs, batch_size=32,
verbose=1, validation_data=(X_test, y_test))
    self.test_predict = self.model.predict(X_test)
    self.y_test_predict = y_test

    tk.messagebox.showinfo("Sukses", f"Model berhasil dilatih dengan {epochs} epoch
dan siap untuk divisualisasikan!")

```

```

def predict_future(self):
    if self.model is None:
        tk.messagebox.showerror("Error", "Model belum dibuat atau dilatih.")
        return

    try:
        steps = int(self.input_steps.get())
        if steps <= 0:
            raise ValueError

        test_data = self.parent.halaman_unggah.X_test
        x_input2 = test_data[-1:] # Ambil baris terakhir dari data uji
        min_values = self.parent.min_values
        max_values = self.parent.max_values

        predicted_values = []
        for _ in range(steps):
            prediction = self.model.predict(x_input2)
            predicted_value = prediction[0][0]

            # Update sequence dengan prediksi terbaru
            new_sequence = np.copy(x_input2)
            new_sequence[:, :-1] = x_input2[:, 1:]
            new_sequence[:, -1] = predicted_value

            predicted_values.append(predicted_value)
            x_input2 = new_sequence
            ""

        denormalized_y_test = [
            val * (max_values - min_values) + min_values
            for val in self.parent.halaman_unggah.y_test
        ]
        ""

        # Denormalisasi prediksi
        denormalized_predictions = [
            val * (max_values[0] - min_values[0]) + min_values[0]

```

```

        for val in predicted_values
    ]
    test_var_data = self.parent.halaman_unggah.test_var
    # **Gabungkan Data Uji dengan Prediksi dan Simpan**
    y_test_extended = np.append(
        test_var_data,
        denormalized_predictions
    )
    self.parent.halaman_unggah.y_test_extended = y_test_extended

    self.prediksi_output.config(state="normal")
    self.prediksi_output.delete("1.0", "end")
    self.prediksi_output.insert("end", f"Hasil prediksi ({steps} langkah):\n")
    self.prediksi_output.insert(
        "end",
        "\n".join(
            [f"Step {i+1}: {val:.4f}" for i, val in enumerate(denormalized_predictions)]
        )
    )
    self.prediksi_output.config(state="disabled")

except ValueError:
    tk.messagebox.showerror("Error", "Masukkan jumlah langkah prediksi yang valid.")

def hapus_arsitektur(self):
    """Menghapus seluruh arsitektur model yang sudah dikonfigurasi"""
    self.layer_config.clear() # Hapus semua layer yang sudah dimasukkan
    self.layer_preview.config(state="normal") # Aktifkan Text widget
    self.layer_preview.delete("1.0", tk.END) # Hapus isi preview
    self.layer_preview.config(state="disabled") # Nonaktifkan kembali
    tk.messagebox.showinfo("Info", "Arsitektur model berhasil dihapus!")

def hapus_prediksi(self):
    # Pastikan data uji tersedia
    if not hasattr(self.parent.halaman_unggah, 'y_test_extended'):

```

```

tk.messagebox.showinfo("Info", "Tidak ada data prediksi untuk dihapus.")
return

try:
    # Kembalikan data hanya ke y_test
    self.parent.halaman_unggah.y_test_extended = self.parent.halaman_unggah.y_test
    self.test_data2 = pd.DataFrame(self.parent.halaman_unggah.y_test,
columns=['Close']).reset_index(drop=True)

    self.notifikasi.config(text="Data prediksi berhasil dihapus! Hanya data uji yang tersisa.")
    tk.messagebox.showinfo("Sukses", "Data prediksi telah dihapus. Hanya data uji yang digunakan.")

except Exception as e:
    tk.messagebox.showerror("Error", f"Terjadi kesalahan saat menghapus data prediksi:\n{e}")

class HalamanRisiko(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent

        style = Style(theme="superhero")

        container = ttk.Frame(self, padding=20)
        container.pack(fill="both", expand=True)

        ttk.Label(container, text="Analisis Risiko Investasi",
            font=("Helvetica", 20, "bold")).pack(pady=20)

        self.fig, self.ax = plt.subplots(figsize=(6, 4))
        self.ax.set_title("Histogram Returns", fontsize=14, weight="bold")
        self.ax.set_xlabel("Returns", fontsize=12)
        self.ax.set_ylabel("Frequency", fontsize=12)

```

```

self.canvas = FigureCanvasTkAgg(self.fig, master=container)
self.canvas.get_tk_widget().pack(pady=20)

self.notifikasi = ttk.Label(
    container,
    text="⚠ Data prediksi belum tersedia untuk menghitung nilai kerugian.",
    style="danger.TLabel"
)
self.notifikasi.pack(pady=10)

button_frame = ttk.Frame(container)
button_frame.pack(pady=20)

ttk.Button(button_frame, text="⬇️ Ambil Data Uji", command=self.ambil_data_uji,
style="primary.TButton").pack(side="left", padx=10)
ttk.Button(button_frame, text="📊 Hitung VaR", command=self.hitung_var,
style="success.TButton").pack(side="left", padx=10)
ttk.Button(button_frame, text="⬅️ Kembali",
command=self.parent.kembali_ke_halaman_utama,
style="danger.TButton").pack(side="left", padx=10)

ttk.Label(container, text="📈 Hasil Value at Risk:", style="secondary.TLabel").pack(pady=10)
self.hasil_var = tk.Text(container, height=5, width=60, state="disabled",
bg="#f8f9fa",
relief="solid", borderwidth=2, font=("Arial", 10))
self.hasil_var.pack(pady=10)

def ambil_data_uji(self):
    # Pastikan data uji tersedia
    if not hasattr(self.parent.halaman_unggah, 'test_var'):
        tk.messagebox.showerror("Error", "Data uji belum tersedia!")
    return

```

```

try:
    # Cek apakah data prediksi sudah tersedia dan gunakan yang terbaru
    if hasattr(self.parent.halaman_unggah, 'y_test_extended'):
        combined_data = self.parent.halaman_unggah.y_test_extended
        self.notifikasi.config(text="Data uji dan prediksi berhasil diambil!")
    else:
        combined_data = self.parent.halaman_unggah.test_var
        self.notifikasi.config(text="Data uji berhasil diambil!")

    # Simpan data uji dalam DataFrame
    self.test_data2      = pd.DataFrame(combined_data,
columns=['Close']).reset_index(drop=True)

    if not hasattr(self, 'btn_lihat_data'):
        self.btn_lihat_data = ttk.Button(self, text=" 

```

```

data_window = tk.Toplevel(self)
data_window.title("Data Uji dan Prediksi")
data_window.geometry("600x400")

# Judul
    ttk.Label(data_window, text="Data Uji dan Prediksi", font=("Helvetica", 16, "bold")).pack(pady=10)

# Membuat Treeview untuk menampilkan data dalam bentuk tabel
tree = ttk.Treeview(data_window, columns=("Index", "Close"), show="headings")
tree.heading("Index", text="Index")
tree.heading("Close", text="Close Price")
tree.pack(expand=True, fill="both", padx=20, pady=20)

# Menambahkan data ke dalam Treeview
for index, value in self.test_data2.iterrows():
    tree.insert("", "end", values=(index, value['Close']))

# Tombol Tutup Window
    ttk.Button(data_window, text="Tutup", command=data_window.destroy, style="danger.TButton").pack(pady=10)

def hitung_var(self):
    if not hasattr(self, 'test_data2') or self.test_data2.empty:
        tk.messagebox.showerror("Error", "Data uji belum tersedia. Klik 'Ambil Data Uji' terlebih dahulu.")
    return

try:
    # **Meminta Input dari User**
    portfolio_value = simpledialog.askfloat("Input", "Masukkan nilai investasi (dalam Rupiah):", minvalue=1)
    holding_period = simpledialog.askinteger("Input", "Masukkan holding period (jumlah hari):", minvalue=1)
    confidence_level = simpledialog.askfloat("Input", "Masukkan tingkat kepercayaan (0-1):", minvalue=0, maxvalue=1)

```

```

if portfolio_value is None or holding_period is None:
    tk.messagebox.showerror("Error", "Input tidak valid atau dibatalkan.")
    return

# **Menghitung Return**
test_close_prices = self.test_data2['Close']
returns = (test_close_prices - test_close_prices.shift(1)) / test_close_prices.shift(1)
returns = returns.dropna() # Hapus nilai NaN dari hasil shift

alpha = 1 - confidence_level # 5% Significance Level
P_alpha = np.percentile(returns, alpha * 100)

# **Menghitung VaR dalam Rupiah**
VaR_rupiah = -portfolio_value * P_alpha * np.sqrt(holding_period)

# **Menghitung Persentase Risiko**
percentage_loss = (VaR_rupiah / portfolio_value) * 100

# **Menampilkan Hasil pada GUI**
self.hasil_var.config(state="normal")
self.hasil_var.delete("1.0", "end")
    self.hasil_var.insert("end", f" 

```

```
self.ax.legend()
self.ax.set_title(f'Distribusi Portfolio Return (t = {holding_period} hari)')
self.ax.set_xlabel("Return (Rupiah)")
self.ax.set_ylabel("Frekuensi")

# **Refresh Canvas**
self.canvas.draw()

except Exception as e:
    tk.messagebox.showerror("Error", f"Terjadi kesalahan: {e}")

# Menjalankan aplikasi
if __name__ == "__main__":
    app = AplikasiSaham()
    app.mainloop()
```