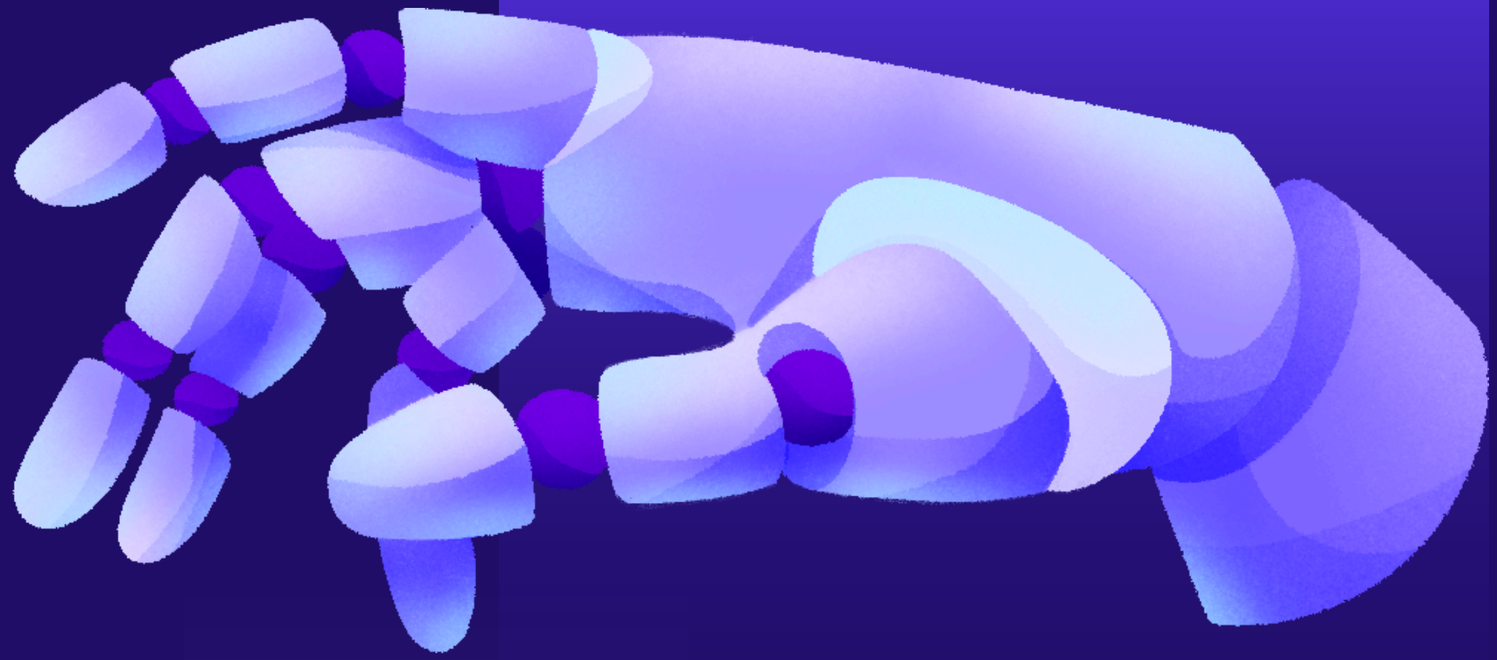


MARKET EYE AI-POWERED STOCK ANALYSIS SYSTEM

Team 1

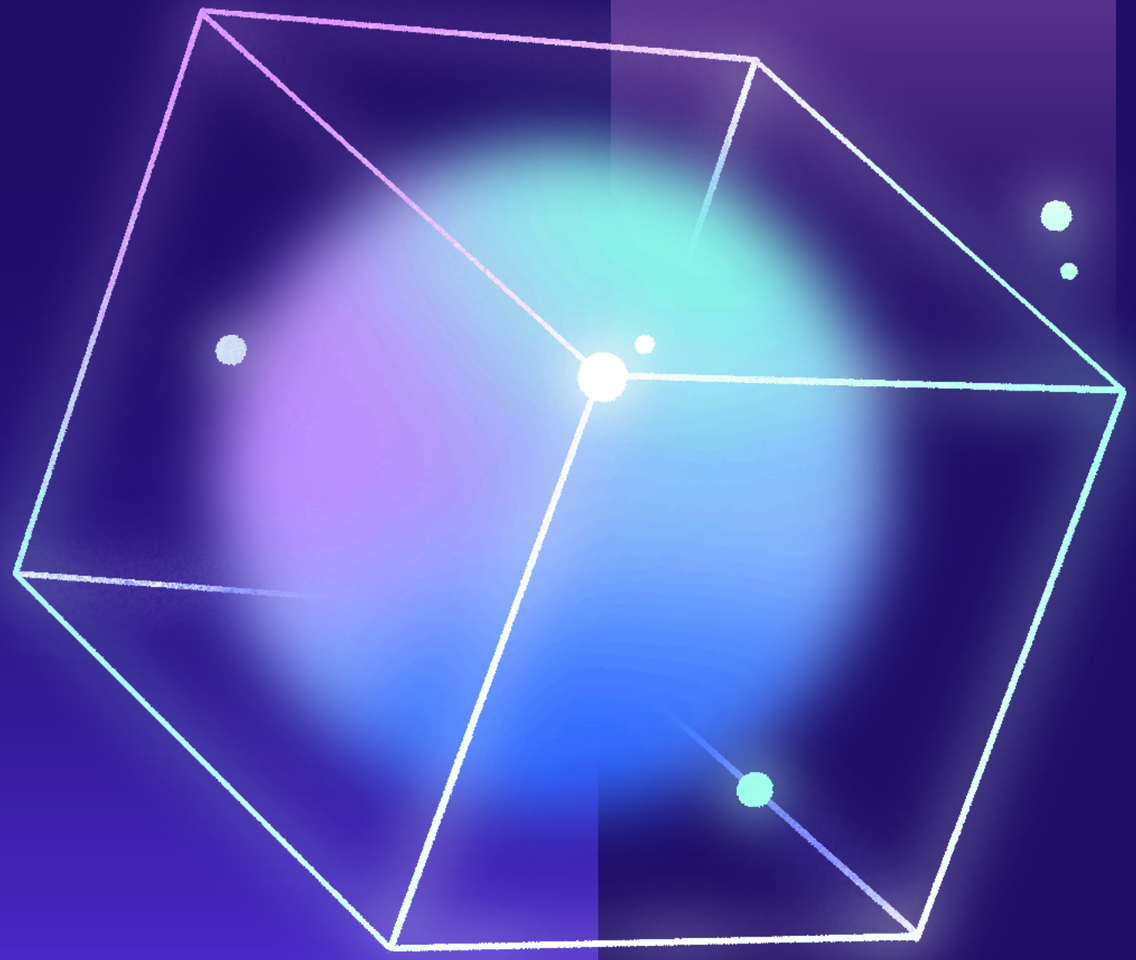
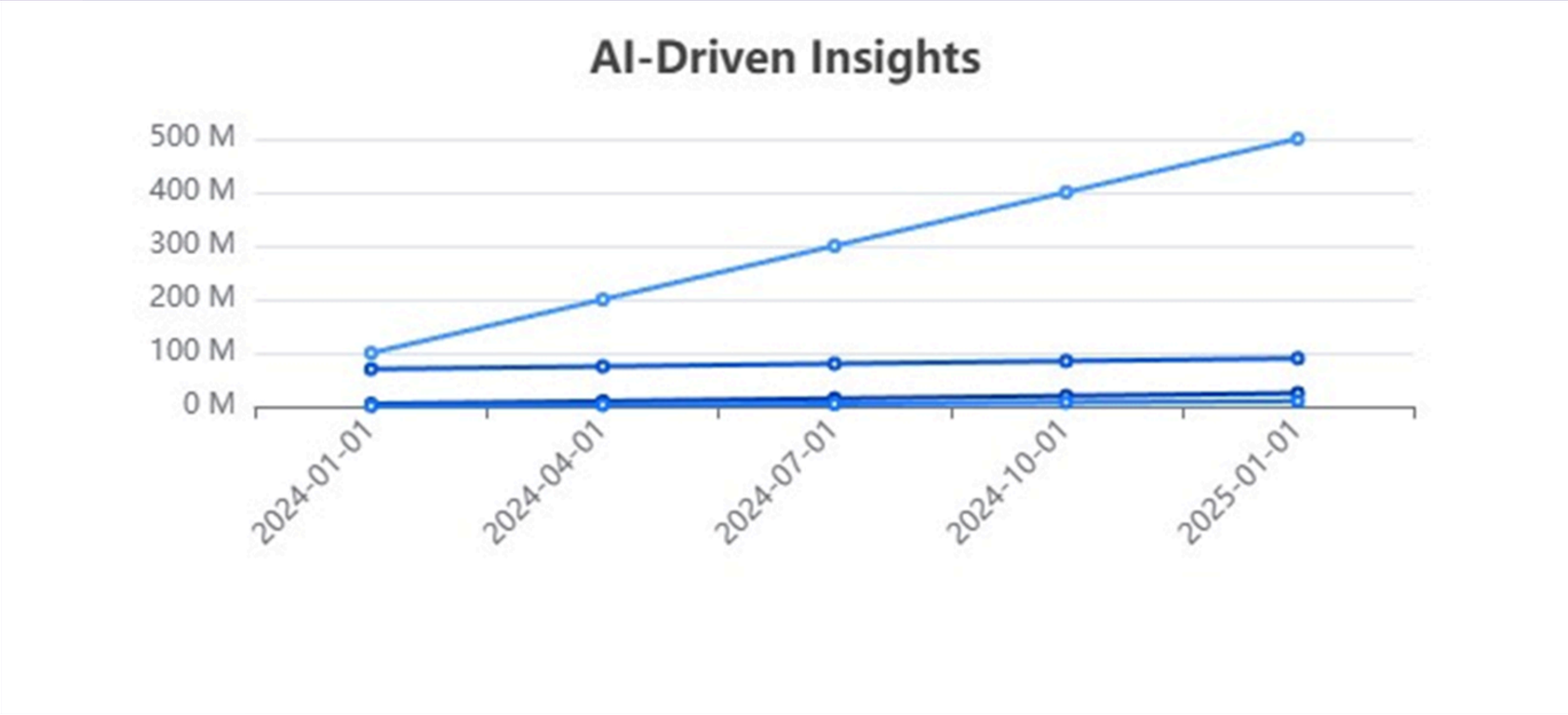


TABLE OF CONTENT:

SECTION 1
PROJECT OVERVIEW AND OBJECTIVES

INTRODUCTION TO MARKET EYE SYSTEM



KEY OBJECTIVES OF THE PROJECT

1 Automated Analysis Efficiency

Streamline stock market data analysis through automation, reducing manual effort and enhancing real-time investment insights for users.

2 Enhanced Prediction Accuracy

Aim for a minimum of 85% accuracy in stock price forecasts by employing advanced algorithms and diverse forecasting techniques.

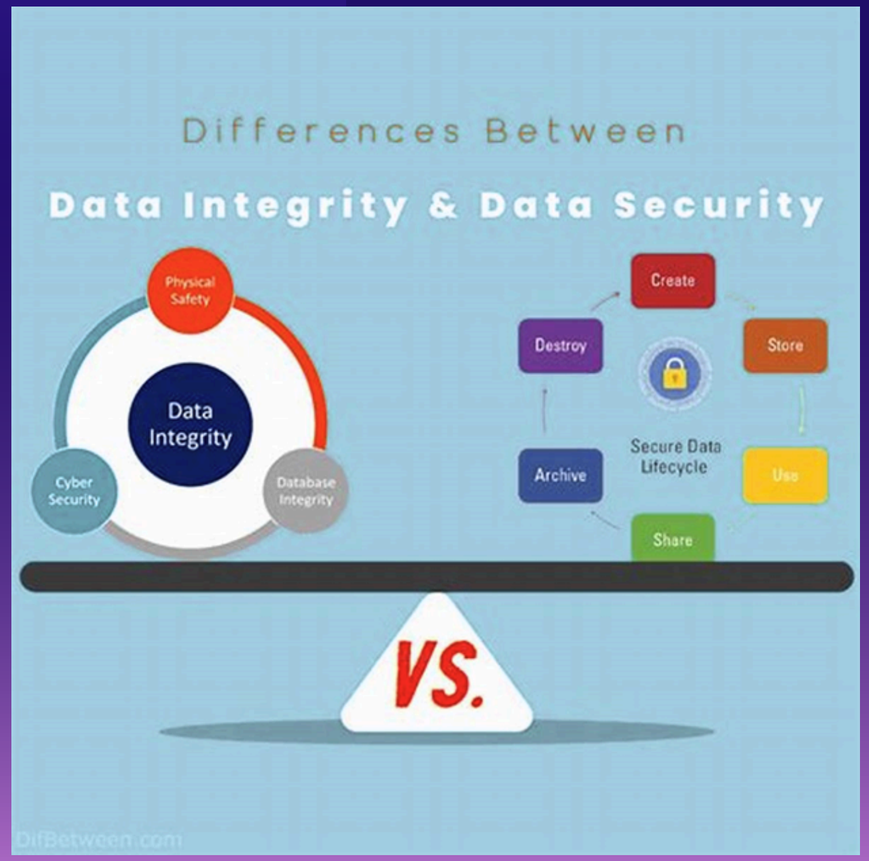
3 Intuitive User Experience

Design a user-friendly interface that simplifies access to analytical tools, catering to investors with varying levels of expertise.

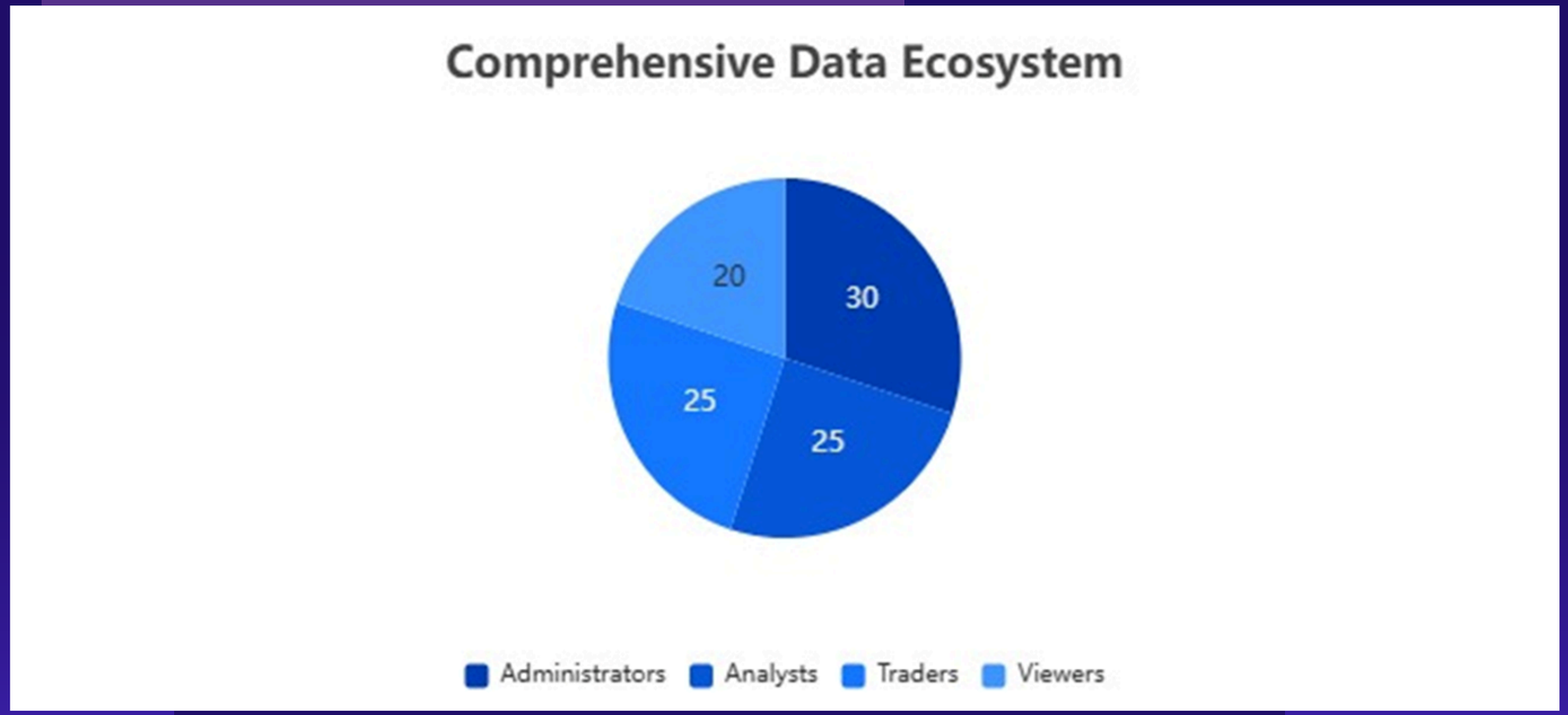
IMPORTANCE OF SYSTEM SECURITY AND DATA INTEGRITY

Robust Security Measures

Implementing comprehensive security protocols, including encryption and access controls, is vital for protecting sensitive financial data, ensuring data integrity, and maintaining user trust in the Market Eye AI-Powered Stock Analysis System.



OVERVIEW OF DATA COLLECTION AND ANALYSIS TECHNIQUES



SECTION 2
FEATURES AND TECHNICAL IMPLEMENTATION

RECOMMENDATION GENERATION PROCESS

Systematic Data Processing

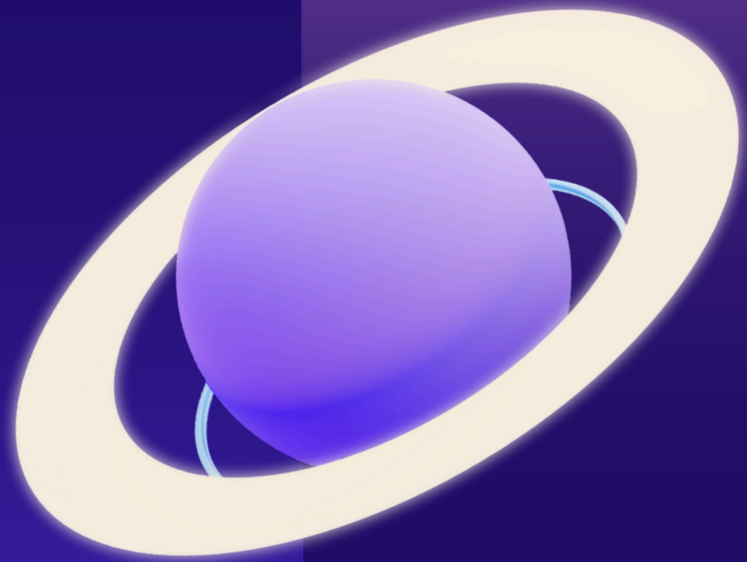
Employs rigorous data cleaning and preprocessing to ensure high-quality input for analysis.

Advanced Predictive Models

Utilizes cutting-edge algorithms like ARIMA and LSTM for accurate stock price forecasting.

Actionable Investment Insights

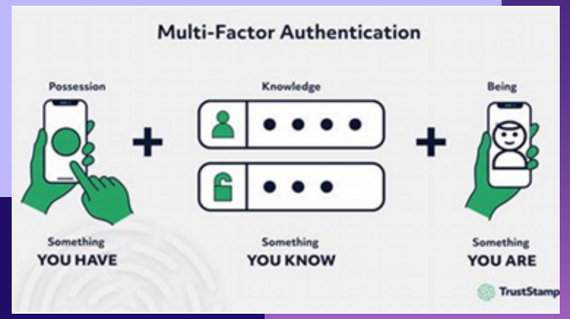
Generates tailored recommendations, including 'Buy', 'Hold', or 'Sell', based on predictive outcomes.



USER AUTHENTICATION AND ACCESS CONTROL

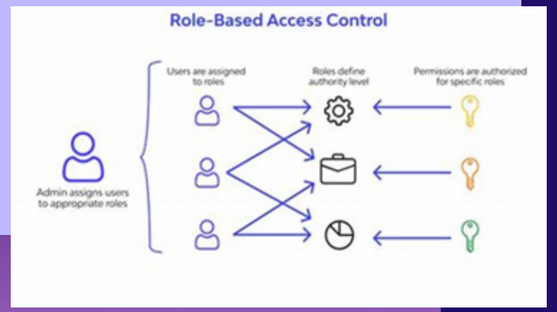
Multi-Factor Authentication (MFA)

The implementation of MFA significantly enhances security by requiring users to verify their identity through multiple methods, reducing the risk of unauthorized access.



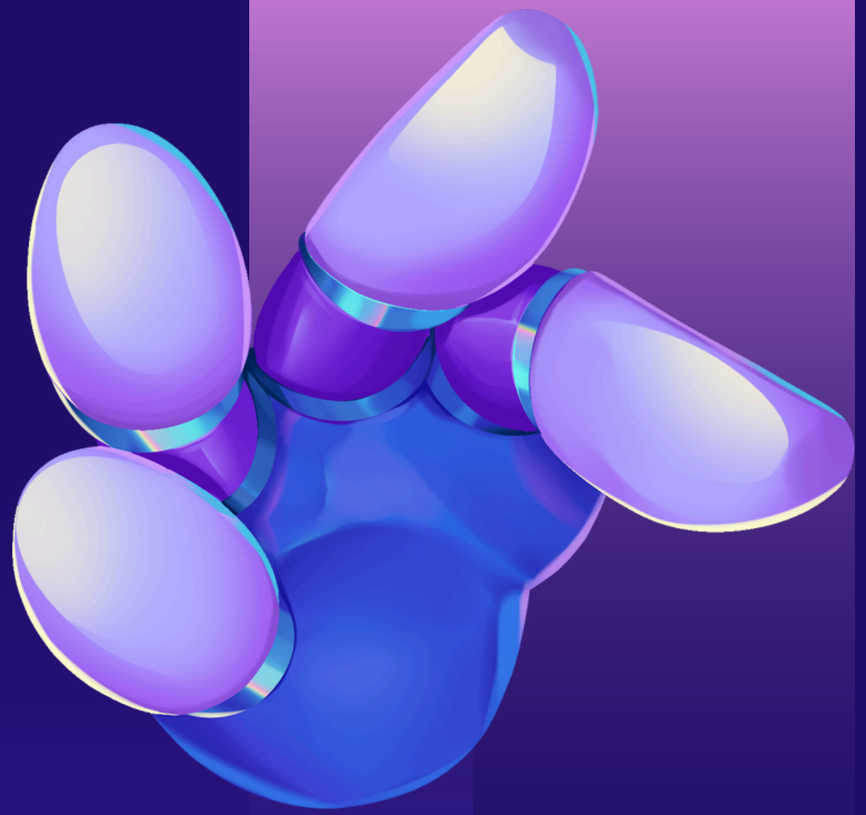
Role-Based Access Control (RBAC)

Aim for a minimum of 85% accuracy in stock price forecasts by employing advanced algorithms and diverse forecasting techniques.



Data Encryption Standards

Design a user-friendly interface that simplifies access to analytical tools, catering to investors with varying levels of expertise.



DATA COLLECTION METHODS AND SOURCES

1 Diverse Data Acquisition Techniques

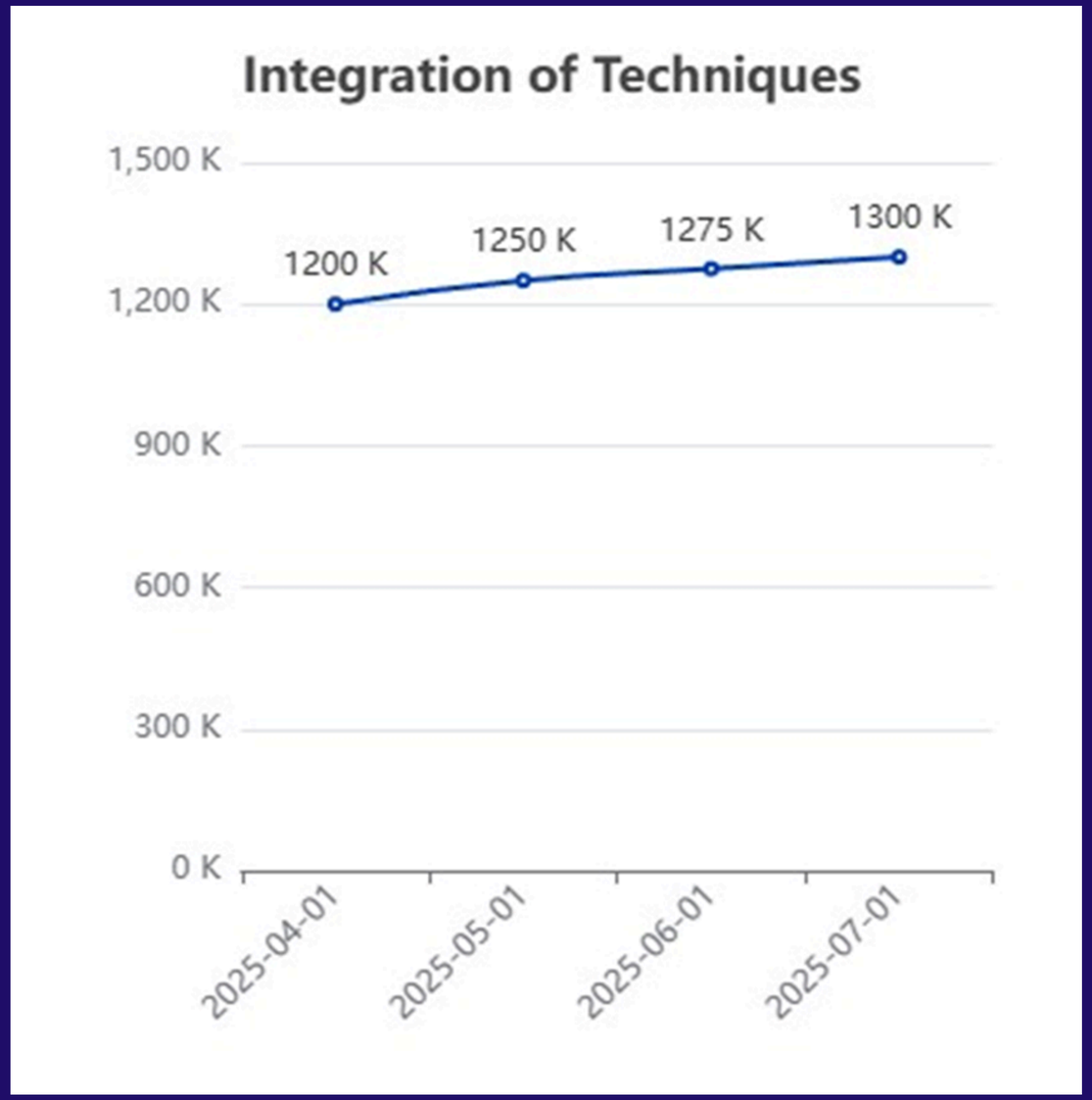
Employing a combination of web scraping, API integration, and user-reported data ensures a comprehensive dataset for accurate market analysis and investment insights.

2 Quality Assurance from Sources

Utilizing reputable financial exchanges, news aggregators, and public databases enhances data reliability, fostering informed decision-making and effective stock performance forecasting.

Integration of Techniques

COMBINING TIME SERIES ANALYSIS, REGRESSION ANALYSIS, AND MACHINE LEARNING ALGORITHMS ENHANCES PREDICTIVE ACCURACY BY LEVERAGING THE STRENGTHS OF EACH METHOD TO ANALYZE COMPLEX MARKET BEHAVIORS AND TRENDS.



FORECASTING TECHNIQUES AND ALGORITHMS

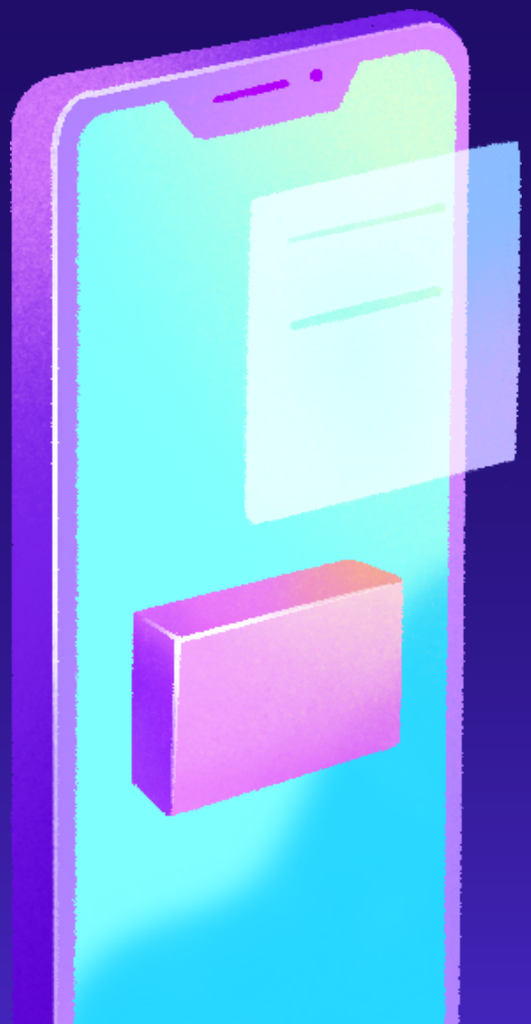
SECTION 3
CHALLENGES, OUTCOMES, AND FUTURE DIRECTIONS

CHALLENGES FACED DURING DEVELOPMENT

INTEGRATION OF DATA SOURCES

The complexity of merging diverse data collection methods posed significant challenges, necessitating advanced algorithms to ensure accurate forecasting and seamless operation, ultimately impacting project timelines and requiring iterative refinements.

ACHIEVEMENTS AND RESULTS OF THE PROJECT



USER GROWTH MILESTONE

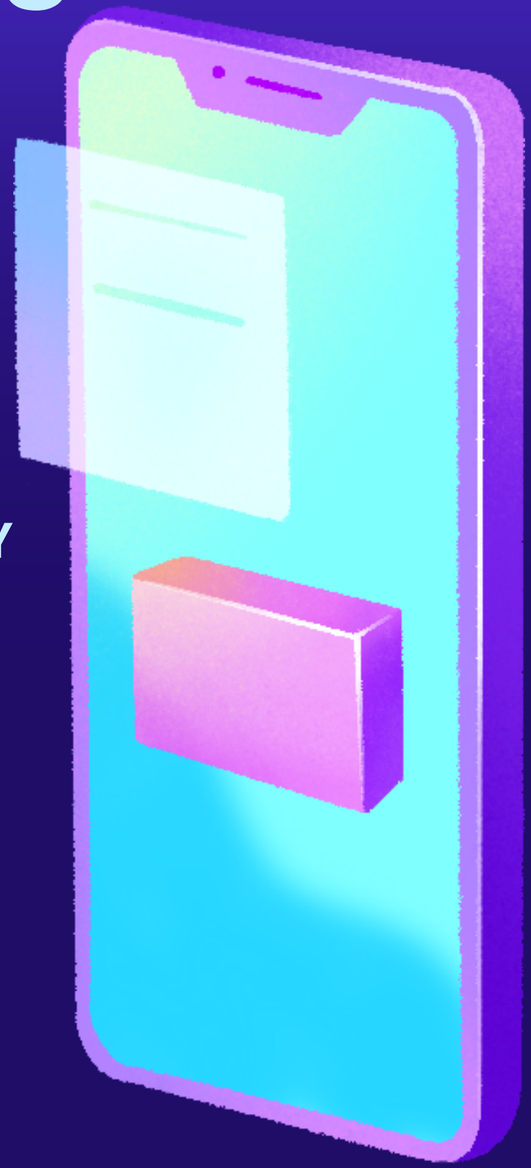
Achieved a 150% increase in user registrations within six months post-launch.

HIGH PREDICTIVE ACCURACY

Forecasting algorithms reached a 92% accuracy rate in stock price predictions.

REAL-TIME DATA PROCESSING

Processed over 10 million data points daily, improving efficiency by 75%.



FUTURE IMPROVEMENTS AND ENHANCEMENTS

1.MACHINE LEARNING INTEGRATION

Future enhancements will incorporate state-of-the-art machine learning techniques to refine predictive models, aiming for a 15% increase in forecasting accuracy by Q1 2025.

2.USER-CENTRIC DESIGN UPDATES

Planned UI upgrades will feature customizable dashboards and real-time data visualizations, enhancing user engagement and satisfaction by mid-2025.

SECTION 4
MARKET EYE AI STOCK SYSTEM: CODE, PURPOSE & OUTPUTS

CORE IMPLEMENTATION DETAILS OF THE MARKET EYE AI-POWERED STOCK ANALYSIS SYSTEM: CODE, PURPOSE, AND OUTPUTS

THIS BLOCK-BY-BLOCK BREAKDOWN COVERS THE KEY TASKS OF THE MARKET EYE PROJECT AS FOLLOWING:

1.detailing the inputs.
2.Reasons
3.Code
4.File location
5.Outputs.

THE IMPLEMENTATION ENSURES:

1. Secure user access
2.Automated data processing
3.Insightful analytics
4.Interactive visualization
5.Report generation

BLOCK 1: USER AUTHENTICATION SETUP

1.Input

•Create a secure login and signup system for users.

•Store user credentials in a database and log user activities for session management.

2. Why?

•To ensure user security by hashing passwords and managing user sessions, a core project requirement.

•To enable personalized dashboards by tracking user activities (e.g., login, report downloads).

3. **Where:** backend/auth.py

4. **Output :** A secure login and signup system was established, storing user credentials in users.db.

For example, a user "algha" can register with a password, which is hashed using SHA-256 for security.

Upon login, the system verifies the credentials and logs the action (e.g., "Logged in as algha").

This enables session management, allowing users to access personalized dashboards with their activity tracked, such as viewing stock data or downloading reports.

```python
import sqlite3
import hashlib

def register_user(username, password):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    c.execute('CREATE TABLE IF NOT EXISTS users (username TEXT, password TEXT)')
    password = hashlib.sha256(password.encode()).hexdigest()
    c.execute('INSERT INTO users VALUES (?, ?)', (username, password))
    conn.commit()
    conn.close()
    return True, "User registered successfully!"

def login_user(username, password):
    conn = sqlite3.connect('users.db')
    c = conn.cursor()
    password = hashlib.sha256(password.encode()).hexdigest()
    c.execute('SELECT * FROM users WHERE username = ? AND password = ?', (username, pass
    user = c.fetchone()
    conn.close()
    if user:
        return True, user[0], "Logged in successfully!"
    return False, None, "Invalid username or password."
```

BLOCK 2: AUTOMATED DATA COLLECTION

1. Input (What did we need to do)

•Fetch and preprocess stock data from 2000 to 2024, and evaluation data for January 2025.

•Filter the data by specific tickers (e.g., AAPL, GOOGL, MSFT, AMZN, TSLA, JPM, NKE).

2. Why?

•To automate dataset integration, ensuring the app has up-to-date stock data for analysis and forecasting.

•A key project requirement to enable historical analysis and future predictions.

3. Where? backend/collect_stock_data.py.

4. Output: The script processed the World-Stock-Prices-Dataset.csv file, extracting data for 7 tickers (AAPL, GOOGL, MSFT, AMZN, TSLA, JPM, NKE) from January 2000 to December 2024 for historical analysis, and January 2025 for evaluation. The output includes processed CSV files (e.g., AAPL_stock_data.csv) saved in the data/ directory, each containing columns like date, open, high, low, close, and volume, totaling 40,441 rows for historical data and 24 rows per ticker for January 2025.

```python
import pandas as pd
import logging

def collect_stock_data(tickers, full_historical=True):
    cutoff_date = pd.to_datetime("2024-12-31", utc=True) if full_historical else pd.to_
    start_date = pd.to_datetime("2000-01-03", utc=True) if full_historical else pd.to_da
    logging.info(f"Loading Kaggle dataset from {KAGGLE_DATA_PATH} for {'historical analy
    df = pd.read_csv(KAGGLE_DATA_PATH)
    df['date'] = pd.to_datetime(df['date'], utc=True, errors='coerce')
    df = df[df['ticker'].isin(tickers)]
    df = df[(df['date'] >= start_date) & (df['date'] <= cutoff_date)]
    df = df.sort_values('date')
    logging.info(f"Collected {len(df)} rows for tickers {tickers} between {start_date} a
    return df
```

BLOCK 3: HISTORICAL ANALYSIS AND FORECASTING

1.Input :

•Compute historical metrics such as volatility, RSI, and growth percentages (e.g., 2020 growth).

•Forecast stock prices for January 2025 using an LSTM model.

2. Why?

•To provide investors with historical insights (e.g., trends, risk metrics) and future price predictions, which are core requirements for analysis and forecasting.

•To enable data-driven decision-making through actionable metrics and forecasts.

3. **Where?** agents/stock_analysis_agents.py.

4**. Output:** The code computed historical metrics for 7 tickers:

• Volatility (e.g., AAPL: 0.25 annualized, indicating moderate price fluctuations).

•RSI (e.g., AAPL: 65, suggesting an overbought condition).

•2020 growth (e.g., AAPL: 78.24%, showing strong performance).

The LSTM model forecasted January 2025 prices (e.g., AAPL: $180 ± $9).

•Providing investors with a predicted price range.

•These results were saved in forecasts/ (e.g., AAPL_jan2025_forecast.csv) and analytics_and_forecasts.json, enabling the dashboard to display trends and predictions.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input

# Calculate Volatility
def calculate_volatility(df, window=60):
    returns = df['close'].pct_change().fillna(0)
    volatility = returns.rolling(window=window).std() * np.sqrt(252)
    volatility = volatility.fillna(0)
    return volatility

# Calculate RSI
def calculate_rsi(df, periods=14):
    delta = df['close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=periods).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=periods).mean()
    rs = gain / loss
    rsi = 100 - (100 / (1 + rs))
    rsi = rsi.fillna(0)
    return rsi

# Compute Growth and Forecast with LSTM
def process_data_and_forecast(combined_data):
    tickers = combined_data['ticker'].unique()
    analytics = {}
    forecasts = {}
    for ticker in tickers:
        ticker_data = combined_data[combined_data['ticker'] == ticker]
        # Compute 2020 Growth
        year_2020_data = ticker_data[ticker_data['date'].dt.year == 2020]
        if not year_2020_data.empty:
            start_price = year_2020_data.iloc[0]['close']
            end_price = year_2020_data.iloc[-1]['close']
            growth_2020 = ((end_price - start_price) / start_price) * 100 if
```

BLOCK 4: INTERACTIVE DASHBOARD

1.Input :

•Display interactive charts (candlestick, volatility, RSI) for historical data.

•Show January 2025 forecasts and sector comparisons (Tech, Finance, Sportswear).

2. Why?

•To provide a user-friendly interface for visualizing stock data, trends, and predictions, meeting the Streamlit frontend requirement.

•To enable investors to explore data interactively and make informed decisions.

3. Where? frontend/app.py.

4. Output The dashboard displays interactive visualizations:

• Candlestick charts showing price trends (e.g., AAPL's overall trend from 2000-2024),

•Volatility charts (e.g., 60-day annualized fluctuations),

• RSI charts (e.g., 14-day, with overbought/oversold thresholds),

•January 2025 forecasts (e.g., AAPL: $180).

•Sector comparisons highlight trends,

•Tech's average 2020 growth of 243.22%.

•Users can access this at http://localhost:8501 locally or via the deployed URL (e.g., https://nora140321-marketeye.streapp), providing an intuitive interface for exploring stock insights.amlit.

```python
import streamlit as st
import plotly.graph_objects as go
import plotly.express as px

# Candlestick Chart
fig = go.Figure(data=[go.Candlestick(
    x=df['date'],
    open=df['open'],
    high=df['high'],
    low=df['low'],
    close=df['close']
)])
fig.update_layout(
    title=f"{ticker} Overall Candlestick Chart",
    yaxis_title="Price",
    xaxis_title="Date",
    template="plotly_dark"
)
st.plotly_chart(fig)

# Sector Comparison Bar Chart
fig_growth = px.bar(
    sector_df,
    x="Sector",
    y="Average 2020 Growth",
    title="Average 2020 Growth by Sector",
    text="Average 2020 Growth",
    template="plotly_dark"
)
fig_growth.update_traces(texttemplate='%{text}', textposition='auto')
st.plotly_chart(fig_growth)
```

BLOCK 5: PDF REPORT GENERATION

1.Input:

•Generate a PDF report containing historical data, metrics, forecasts, and recommendations.

•Include visualizations and key insights for user download.

2. Why?

•To provide a downloadable summary of stock analysis, fulfilling the PDF report generation requirement.

•To enhance user experience by offering a detailed, portable report for offline use.

3. Where: frontend/app.py.

4. Output

The PDF report (e.g., AAPL_report.pdf) includes historical data for the last 60 day average price: $233.37, RSI: 65), January 2025 forecasts (e.g., $180), and recommendations (e.g., "Hold AAPL due to stability"). The report is generated via the dashboard and saved in the reports/ directory, downloadable by users for offline reference, providing a comprehensive summary of stock insightsays (e.g., AAPL's closing prices and volumes), key metrics (e.g.,

```python
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from reportlab.platypus import SimpleDocTemplate, Table, Paragraph

def generate_pdf_report(ticker, historical_df, summary_metrics, recommendation, forecast
    buffer = BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=letter)
    elements = []
    elements.append(Paragraph(f"Market Eye Report for {ticker}", title_style))
    historical_data = historical_df.tail(60)[['date', 'close', 'volume']].copy()
    historical_data['date'] = historical_data['date'].apply(lambda x: x.strftime("%d-%m-
    data = [['Date', 'Close', 'Volume']] + historical_data.values.tolist()
    table = Table(data)
    elements.append(table)
    summary_text = (
        f"- Average Closing Price: ${summary_metrics['avg_close']:.2f}\n"
        f"- Recent Volatility: {summary_metrics['recent_volatility']:.2f}\n"
        f"- Current RSI: {summary_metrics['current_rsi']:.2f}\n"
    )
    elements.append(Paragraph(summary_text, normal_style))
    elements.append(Paragraph(recommendation, normal_style))
    doc.build(elements)
    buffer.seek(0)
    return buffer
```

THANK YOU!