SOURCE CODE:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
    mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split


import spacy
import json
import time
import random
import hashlib
import re


from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import base64


# ---------------- Forecasting Model ----------------
def generate_sample_data():
    data = {
```

```
    'day': list(range(1, 31)),
    'sales': [50, 52, 54, 53, 55, 60, 58, 59, 62, 65, 67, 66, 70,
              72, 74, 73, 75, 77, 78, 80, 85, 87, 90, 88, 92, 95, 97,
 100, 102, 105]
  }
  return pd.DataFrame(data)

def train_forecasting_model():
  df = generate_sample_data()
  X = df[['day']]
  y = df['sales']
  X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, random_state=0)

  model = LinearRegression()
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)

  print("\n--- Forecasting Model Results ---")
  mse = mean_squared_error(y_test, y_pred)
  print("RMSE:", mse ** 0.5)
  print("MAE:", mean_absolute_error(y_test, y_pred))
```

```
    print("R² Score:", r2_score(y_test, y_pred))
    return model

# ---------------- Advanced Chatbot Interface ----------------
class ChatBot:
    def __init__(self):
        self.nlp = spacy.load("en_core_web_sm")

    def classify_intent(self, message):
        doc = self.nlp(message.lower())
        lemmas = [token.lemma_ for token in doc]

        if any(greet in lemmas for greet in ["hi", "hello", "hey"]):
            return "greeting"
        if "order" in lemmas and any(word in lemmas for word in
    ["track", "status", "check", "where"]):
            return "track_order"
        if "forecast" in lemmas or "demand" in lemmas:
            return "demand_forecast"
        if "product" in lemmas or "info" in lemmas or "price" in
    lemmas:
            return "product_info"
```

```
    if "bye" in lemmas or "exit" in lemmas or "quit" in
lemmas:
        return "goodbye"
    return "unknown"

def extract_order_id(self, message):
    match = re.search(r'\b[A-Z0-9]{4,}\b', message.upper())
    if match:
        return match.group()
    return None

def respond(self, message):
    intent = self.classify_intent(message)

    if intent == "greeting":
        return "Hello! How can I help you with your supply
chain today?"

    elif intent == "track_order":
        order_id = self.extract_order_id(message)
        if order_id:
```

```
        return f"Looking up order ID {order_id}... Status: In transit 🚚"
    else:
        return "Please provide your order ID so I can track it."

elif intent == "demand_forecast":
    return "This week's forecast: 📈 15% increase in demand expected due to seasonal trends."

elif intent == "product_info":
    return "We offer 500+ products. Please specify a product name or type for more details."

elif intent == "goodbye":
    return "Goodbye! Feel free to return if you have more questions."

else:
    return "I'm not sure I understood. Try asking about orders, products, or demand forecasts."
```

```
    def chat(self):
        print("\n--- AI Chatbot Ready --- (type 'exit' to quit)")
        while True:
            msg = input("You: ")
            if msg.strip().lower() in ["exit", "quit", "bye"]:
                print("Bot: Goodbye! 👋")
                break
            print("Bot:", self.respond(msg))

def run_chatbot():
    bot = ChatBot()
    bot.chat()

# ---------------- IoT Feed Simulation ----------------
def generate_sensor_feed():
    feed = {
        "timestamp": time.time(),
        "stock_level": random.randint(50, 150),
        "transit_status": random.choice(["in_transit",
    "delivered", "delayed"])
    }
```

```
    return json.dumps(feed)


def simulate_iot_feed(duration_seconds=20):
    print("\n--- IoT Feed Simulation ---")
    start = time.time()
    while time.time() - start < duration_seconds:
        feed = generate_sensor_feed()
        print(feed)
        time.sleep(5)

# ---------------- Blockchain Simulation ----------------
class Block:
    def __init__(self, index, previous_hash, data,
     timestamp=None):
        self.index = index
        self.timestamp = timestamp or time.time()
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.compute_hash()

    def compute_hash(self):
```

```
        block_string =
    f"{self.index}{self.timestamp}{self.data}{self.previous_hash}
    "
        return hashlib.sha256(block_string.encode()).hexdigest()

def simulate_blockchain():
    print("\n--- Blockchain Simulation ---")
    chain = []
    genesis_block = Block(0, "0", "Genesis Block")
    chain.append(genesis_block)

    for i in range(1, 4):
        block = Block(i, chain[-1].hash, f"Transaction {i}")
        chain.append(block)

    for block in chain:
        print(f"Block {block.index}: {block.hash}")

# ---------------- Security Framework ----------------
def pad(data):
    pad_length = AES.block_size - len(data) % AES.block_size
    return data + pad_length * chr(pad_length)
```

```
def unpad(data):
    pad_length = ord(data[-1])
    return data[:-pad_length]

def encrypt_message(message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    padded_message = pad(message)
    encrypted_bytes =
     cipher.encrypt(padded_message.encode('utf-8'))
    return base64.b64encode(encrypted_bytes).decode('utf-8')

def decrypt_message(encrypted_message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    encrypted_bytes =
     base64.b64decode(encrypted_message)
    decrypted_padded =
     cipher.decrypt(encrypted_bytes).decode('utf-8')
    return unpad(decrypted_padded)

def run_security_demo():
    print("\n--- Security Framework ---")
```

```
    key = get_random_bytes(16)
    message = "Sensitive Supply Chain Data"
    encrypted = encrypt_message(message, key)
    print("Encrypted:", encrypted)
    decrypted = decrypt_message(encrypted, key)
    print("Decrypted:", decrypted)

# ---------------- Main Menu ----------------
def main():
    print("\nAI-Powered Supply Chain Management System")
    while True:
        print("\nSelect an option:")
        print("1. Run Forecasting Model")
        print("2. Start Chatbot")
        print("3. Simulate IoT Feed")
        print("4. Simulate Blockchain Ledger")
        print("5. Run Security Encryption")
        print("6. Exit")

        choice = input("Enter your choice: ")
        if choice == "1":
```

```
            train_forecasting_model()
        elif choice == "2":
            run_chatbot()
        elif choice == "3":
            simulate_iot_feed()
        elif choice == "4":
            simulate_blockchain()
        elif choice == "5":
            run_security_demo()
        elif choice == "6":
            print("Exiting system...")
            break
        else:
            print("Invalid choice. Try again.")

if __name__ == "__main__":
    main()
```

```
Select an option:
1. Run Forecasting Model
2. Start Chatbot
3. Simulate IoT Feed
4. Simulate Blockchain Ledger
5. Run Security Encryption
6. Exit
Enter your choice: 1

--- Forecasting Model Results ---
RMSE: 1.6649197472469235
MAE: 1.3114919354838754
R² Score: 0.9908783839550439
```

```
AI-Powered Supply Chain Management System

Select an option:
1. Run Forecasting Model
2. Start Chatbot
3. Simulate IoT Feed
4. Simulate Blockchain Ledger
5. Run Security Encryption
6. Exit
Enter your choice: 1

--- Forecasting Model Results ---
RMSE: 1.6649197472469235
MAE: 1.3114919354838754
R² Score: 0.9908783839550439
```

```
Select an option:
1. Run Forecasting Model
2. Start Chatbot
3. Simulate IoT Feed
4. Simulate Blockchain Ledger
5. Run Security Encryption
6. Exit
Enter your choice: 2

--- AI Chatbot Ready --- (type 'exit' to quit)
You: hi
Bot: Hello! How can I help you with your supply chain today?
You: can u track the order 12980
Bot: Looking up order ID TRACK... Status: In transit 🚚
You: another order 89084 track this also
Bot: Looking up order ID ANOTHER... Status: In transit 🚚
You: exit
Bot: Goodbye! 👋
```

```
Select an option:
1. Run Forecasting Model
2. Start Chatbot
3. Simulate IoT Feed
4. Simulate Blockchain Ledger
5. Run Security Encryption
6. Exit
Enter your choice: 3

--- IoT Feed Simulation ---
{"timestamp": 1746598800.1504974, "stock_level": 131, "transit_status": "delivered"}
{"timestamp": 1746598805.1536047, "stock_level": 73, "transit_status": "delivered"}
{"timestamp": 1746598810.1546526, "stock_level": 133, "transit_status": "in_transit"}
{"timestamp": 1746598815.1559014, "stock_level": 141, "transit_status": "delivered"}
```

```
Select an option:
1. Run Forecasting Model
2. Start Chatbot
3. Simulate IoT Feed
4. Simulate Blockchain Ledger
5. Run Security Encryption
6. Exit
Enter your choice: 4

--- Blockchain Simulation ---
Block 0: d850b91527b88329c0ddc5f1328892b3b531b5838f942c94f7992b221ccdbc6e
Block 1: 890f798d7beef2326984f2060d4e4a73e93766d6d34ad94ed14f390f30853706
Block 2: 7399e1a9b5f1d8ad32210fa076c47520b9e5d19d68d6f33b181741a78ccee5dc
Block 3: c8ce3bcc98fb6a0f40f2aca0a5076140fa1bbffcd7c73d50694f346a8af51c5d
```

```
Select an option:
1. Run Forecasting Model
2. Start Chatbot
3. Simulate IoT Feed
4. Simulate Blockchain Ledger
5. Run Security Encryption
6. Exit
Enter your choice: 5

--- Security Framework ---
Encrypted: Hh32RuYhSwujaHSodDNSWDEDAL9R/JjdCKDE+Ic6c0g=
Decrypted: Sensitive Supply Chain Data
```

```
Select an option:
1. Run Forecasting Model
2. Start Chatbot
3. Simulate IoT Feed
4. Simulate Blockchain Ledger
5. Run Security Encryption
6. Exit
Enter your choice: 6
Exiting system...
```