Python Code:

```
# Demand Forecasting in Agriculture

# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings('ignore')

# Load dataset (Example dataset - you can replace with actual dataset)
df = pd.read_csv('df.csv')

# Preprocessing
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')

# Define independent and dependent variables for demand forecasting
X = df[['Sales (tons)', 'Price ($/ton)', 'Temp (°C)', 'Rainfall (mm)',
'Consumer_Trend_Index']]
y_demand = df['Demand_Forecast (tons)']

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_demand, test_size=0.3,
random_state=42)

### Question 1: Time-Series Demand Forecasting ###

# Time-Series Forecasting using ARIMA model
# Aggregate sales data to visualize time-series behavior
sales_ts = df['Sales (tons)'].resample('D').sum()

# Fit ARIMA model on sales time series
model_arima = ARIMA(sales_ts, order=(5, 1, 0))  # Adjust ARIMA parameters as needed
arima_fit = model_arima.fit()

# Forecast future demand (next 30 days as an example)
forecast_arima = arima_fit.forecast(steps=30)

# Plot actual sales and ARIMA forecast
plt.figure(figsize=(10,6))
plt.plot(sales_ts, label='Actual Sales')
plt.plot(pd.date_range(start=sales_ts.index[-1], periods=30, freq='D'),
forecast_arima, label='Forecasted Demand', color='red')
plt.xlabel('Date')
plt.ylabel('Sales (tons)')
plt.title('Time-Series Forecast of Agricultural Demand (ARIMA)')
plt.legend()
plt.show()
```

```
### Question 2: Regression Model for Identifying Key Drivers of Demand ###

# Train Random Forest Regressor to identify key drivers of demand
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate model performance
print(f"Random Forest Regression R²: {r2_score(y_test, y_pred_rf):.2f}")
print(f"Mean Absolute Error (MAE): {mean_absolute_error(y_test, y_pred_rf):.2f}")
print(f"Root Mean Squared Error (RMSE): {np.sqrt(mean_squared_error(y_test,
y_pred_rf)):.2f}")

# Feature importance
importances = rf_model.feature_importances_
feature_names = X.columns
sorted_indices = np.argsort(importances)[::-1]

# Plot feature importance
plt.figure(figsize=(10,6))
plt.title('Feature Importance in Agricultural Demand Forecasting (Random Forest)')
sns.barplot(x=importances[sorted_indices], y=feature_names[sorted_indices])
plt.show()

### Question 3: Optimization of Planting Schedules ###

# Simulated planting schedule optimization using Reinforcement Learning concept

def planting_schedule_optimization(crop_demand_forecast, planting_window,
growth_cycle):
    # Example optimization rule: Maximize crops that have high forecast demand and
fit the planting window
    score = crop_demand_forecast / growth_cycle * planting_window
    return score

# Simulated crop demand forecast and parameters
crop_demand_forecast = np.array([500, 620, 470, 550, 500])  # Example demand
forecast for 5 crops
planting_window = np.array([60, 80, 75, 50, 65])  # Days available for planting
growth_cycle = np.array([120, 100, 130, 140, 90])  # Growth cycle length in days

# Calculate planting schedule scores
schedule_scores = planting_schedule_optimization(crop_demand_forecast,
planting_window, growth_cycle)

# Output optimized planting schedule
best_crop_index = np.argmax(schedule_scores)
print(f"Best crop to plant based on demand forecast and planting schedule: Crop
{best_crop_index + 1} with score {schedule_scores[best_crop_index]:.2f}")

### Question 4: Integration with Digital Platforms ###

# Conceptual Example: Integrating demand forecasts into digital platforms

def provide_demand_forecast_to_farmers(demand_forecast,
platform='AgricultureDigitalPlatform'):
```

```
    """
    Simulate the process of providing demand forecasts to farmers via a digital
platform.
    This can be implemented using APIs and platform integrations in real-world
scenarios.
    """
    print(f"Providing demand forecast to farmers via {platform}:")
    print(demand_forecast)

# Example of providing demand forecast to digital platform
provide_demand_forecast_to_farmers(forecast_arima)

### Question 5: Real-Time Data Integration and Dynamic Forecasting ###

# Example: Using a Recurrent Neural Network (RNN) for dynamic demand forecasting
from keras.models import Sequential
from keras.layers import Dense, LSTM

# Reshape data for LSTM model
X_lstm = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))

# LSTM Model Definition
model_lstm = Sequential()
model_lstm.add(LSTM(50, activation='relu', input_shape=(X_train.shape[1], 1)))
model_lstm.add(Dense(1))

# Compile model
model_lstm.compile(optimizer='adam', loss='mse')

# Train LSTM model
model_lstm.fit(X_lstm, y_train, epochs=10, batch_size=32, verbose=1)

# Predict with LSTM on test set (real-time forecasting)
X_test_lstm = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1))
y_pred_lstm = model_lstm.predict(X_test_lstm)

# Evaluate LSTM model
print(f"LSTM R²: {r2_score(y_test, y_pred_lstm):.2f}")
print(f"Mean Absolute Error (MAE) for LSTM: {mean_absolute_error(y_test,
y_pred_lstm):.2f}")

# Plot actual vs predicted using LSTM model
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_lstm, alpha=0.7)
plt.xlabel('Actual Demand (tons)')
plt.ylabel('Predicted Demand (tons)')
plt.title('LSTM Model - Actual vs Predicted Agricultural Demand')
plt.show()
```