F&B Process Anomaly Detection

Code:

```
import streamlit as st
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# --- Page Configuration ---
st.set_page_config(
    page_title="Wine Quality Predictor",
    page_icon="🍷",
    layout="wide"
)

# --- Model Training ---
@st.cache_data
def train_model(df):
    """Loads data and trains the Random Forest model."""
    df['quality_category'] = df['quality'].apply(lambda x: 1 if x >= 7 else 0)
    X = df.drop(['quality', 'quality_category'], axis=1)
    y = df['quality_category']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)
    model = RandomForestClassifier(n_estimators=200, random_state=42)
```

```
    model.fit(X_train, y_train)

    return model, X


# --- Load Data and Train Model ---

try:

    # --- THIS IS THE FINAL FIX ---

    # Based on your test, we are now using the correct comma separator.

    df = pd.read_csv('winequality-red.csv', sep=',')

    model, X = train_model(df)

except FileNotFoundError:

    st.error("The 'winequality-red.csv' file was not found. Please make sure it's in the same folder as this script.")

    st.stop()

except Exception as e:

    st.error(f"An error occurred during data loading or model training: {e}")

    st.stop()


# --- Dashboard Layout ---

st.title("🍷 F&B Process Anomaly Detection")

st.header("Wine Quality Prediction Dashboard")


# --- Sidebar for User Input ---

st.sidebar.header("Input Wine Properties")

st.sidebar.markdown("Use the sliders to input the chemical properties of a wine batch.")


user_inputs = {}

for feature in X.columns:

    min_val = float(X[feature].min())

    max_val = float(X[feature].max())
```

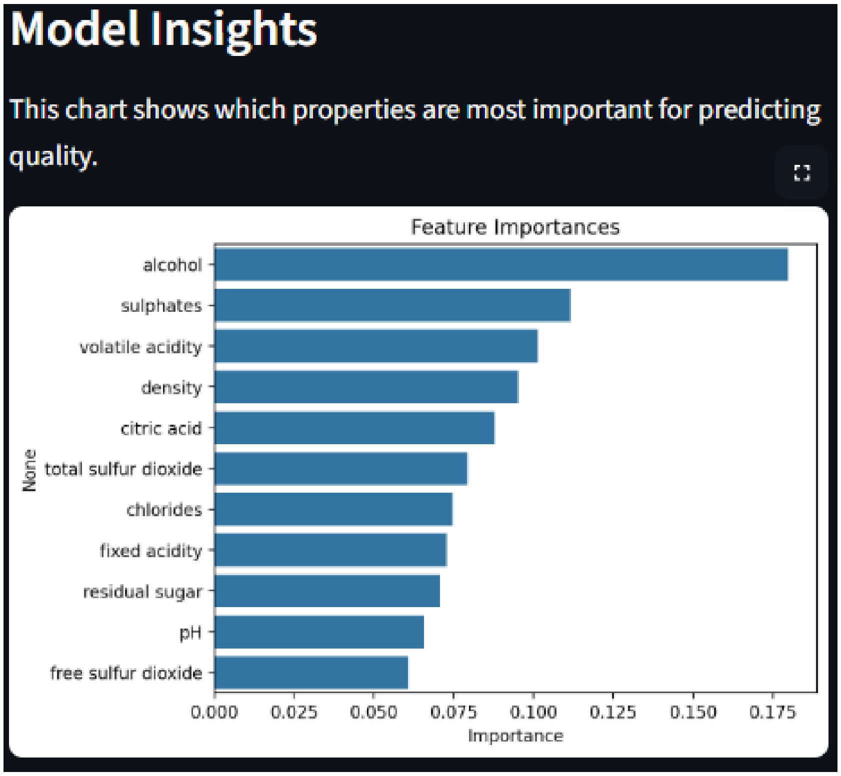
```
    mean_val = float(X[feature].mean())
    user_inputs[feature] = st.sidebar.slider(
        label=f"{feature}",
        min_value=min_val,
        max_value=max_val,
        value=mean_val,
        step=0.01
    )

# --- Main Panel ---
if st.sidebar.button("Predict Quality"):
    user_df = pd.DataFrame([user_inputs])
    prediction = model.predict(user_df)[0]
    prediction_proba = model.predict_proba(user_df)[0]

    st.subheader("Prediction Result")
    col1, col2 = st.columns(2)

    with col1:
        if prediction == 0: # Bad quality
            confidence = prediction_proba[0]
            st.error(f"🚨 QUALITY ALERT: Predicted Bad Quality")
            st.metric(label="Confidence", value=f"{confidence:.2%}")
            st.warning("This batch shows signs of a process anomaly.")
        else: # Good quality
            confidence = prediction_proba[1]
            st.success(f"Predicted Good Quality")
            st.metric(label="Confidence", value=f"{confidence:.2%}")
            st.info("This batch meets the quality standards.")
```

```python
  with col2:
    st.subheader("Model Insights")
    st.markdown("This chart shows which properties are most important for predicting quality.")

    feature_importances = pd.Series(model.feature_importances_, index=X.columns)
    feature_importances = feature_importances.sort_values(ascending=False)

    fig, ax = plt.subplots()
    sns.barplot(x=feature_importances, y=feature_importances.index, ax=ax)
    ax.set_title('Feature Importances')
    ax.set_xlabel('Importance')
    st.pyplot(fig)

# --- Instructions Section ---
st.markdown("<hr>", unsafe_allow_html=True)
with st.expander("ℹ️ How to Use This Dashboard"):
  st.markdown("""
  - 🎚️ **Adjust Sliders:** Use the sliders on the left to input your wine's properties.
  - 🖱️ **Predict:** Click the 'Predict Quality' button to see the result.
  - 📊 **View Result:** The dashboard will show the predicted quality (Good or Bad) and the model's confidence.
  - 📈 **Get Insights:** The 'Feature Importances' chart reveals which factors most influence the prediction.
  """)
```

Dataset link: [Red wine dataset]

<u>Why I chose Random Forest for this project?</u>

1. **Handles Non-Linearity Well**
    - The relationship between chemical properties (like acidity, alcohol, sulphates) and wine quality is not purely linear. Random Forest, being an ensemble of decision trees, can capture complex, non-linear relationships better than simple linear models.
2. **Robustness Against Overfitting**
    - Unlike a single decision tree, Random Forest builds multiple trees and averages their results. This reduces the risk of overfitting, which is important since the dataset is not very large.
3. **High Accuracy and Reliability**
    - Random Forest is known for strong predictive performance on classification tasks. In benchmark tests on the **Wine Quality dataset**, Random Forest often achieves higher accuracy compared to Logistic Regression or a single Decision Tree.
4. **Works Well with Imbalanced Data**
    - The dataset has more "average" wines and fewer "good" ones. Random Forest can handle imbalance better through class weighting and bootstrapping.
5. **Feature Importance Analysis**
    - Random Forest provides a measure of **feature importance**, which helps me explain which chemical properties (e.g., alcohol, sulphates, acidity) contribute most to wine quality. This interpretability is useful in the F&B industry.
6. **Scalability and Ease of Use**
    - It is efficient to train on medium-sized datasets and easy to implement with libraries like **scikit-learn**.



Dashboard to show results: