In [154…
```python
#Importando librerias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Carga y exploraciòn inicial

- Carga el archivo CSV en un DataFrame.
- Usa head(), info() y describe() para conocer el contenido.

In [155…
```python
df = pd.read_csv('../data/manzanas.csv')  #Iportando datos con pandas
print("HEAD")
#Mostrando las primeras 30 filas
df.head(50)
```

```
HEAD
```

Out[155...

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	
0	0	72.456908	225.980262	16.207798	6.493996	71.106940	5.098952	3.50
1	1	77.729087	225.457190	15.198435	69.756663	70.511971	5.260259	3.45543812
2	2	79.445155	227.837949	11.956943	6.828692	71.703181	4.988590	4.12432729
3	3	78.751328	226.365397	13.794924	6.951063	72.182782	3.975872	30.8577611
4	4	82.592012	227.925421	12.769205	6.723497	71.818525	4.608845	3.70
5	5	73.491740	227.745468	11.851893	6.722113	67.688157	5.574385	3.00
6	6	82.530051	232.617529	13.525585	6.161101	71.863807	4.445775	4.08
7	7	76.208622	229.313666	13.918386	6.628514	70.095300	5.292331	3.30
8	8	541.672903	224.024778	13.591857	6.396173	71.375724	6.224276	2.62
9	9	78.616833	229.291487	10.544666	7.298756	70.236229	5.486257	4.03
10	10	74.871261	227.872789	12.148896	6.687227	71.422645	6.020949	3.03
11	11	84.656824	229.097316	12.018976	7.471200	68.747610	5.364297	3.85
12	12	79.675458	227.012366	11.936893	8.206578	68.143267	4.812535	3.18
13	13	77.443492	227.402078	14.226486	7.877406	70.598540	5.130254	3.94
14	14	636.687044	229.448323	12.388122	66.652603	69.219963	5.174714	3.94
15	15	74.946214	225.526905	15.538242	7.348775	71.308547	4.973367	3.38327584
16	16	77.208992	226.879425	11.671376	7.454879	71.718669	6.189787	3.48
17	17	79.858697	222.456400	13.149861	8.467660	70.845853	4.206857	3.85
18	18	79.425508	232.759033	11.534598	8.732554	70.269875	4.977691	4.0
19	19	75.994705	230.570348	12.306284	9.163361	70.936926	3.610948	27.2985310
20	20	75.564280	225.290066	10.928423	7.381196	72.445877	6.903934	3.74
21	21	82.237426	NaN	12.167730	10.809926	70.641573	3.879558	4.12
22	22	75.394502	228.882399	13.087618	7.315053	71.794336	5.233842	
23	23	80.257857	228.793989	11.682312	7.519638	70.348323	5.068193	4.01
24	24	80.993625	227.715064	12.553889	7.893358	67.475474	4.452431	3.31
25	25	77.531010	224.392333	12.322359	7.277953	68.315470	5.514389	2.83
26	26	79.428673	229.178235	13.552604	7.689086	71.364848	5.223601	3.0
27	27	83.799679	231.071984	11.740230	8.322909	69.406302	5.111999	3.12
28	28	82.747499	232.650707	11.933488	7.854481	69.794998	4.603268	3.36
29	29	78.867932	232.455488	11.938409	7.332251	69.740275	5.240132	3.9

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	
30	30	80.779935	226.383536	10.998431	6.857397	68.359693	5.775796	
31	31	75.134344	229.845741	14.448264	7.659147	71.963013	4.460845	3.0
32	32	620.284014	225.229322	12.952117	6.526067	71.820888	4.787417	3.83
33	33	588.120779	228.204821	14.394577	7.155965	73.089244	3.994403	4.71
34	34	79.934701	227.867512	14.397461	8.431379	70.495144	5.318802	4.0
35	35	78.184736	226.062245	10.547706	7.419554	70.601930	6.492692	3.4
36	36	74.183786	231.339366	12.354017	7.762369	71.058484	4.983058	3.30
37	37	87.501416	227.385307	10.135732	7.965816	70.474324	5.474929	3.10
38	38	76.586235	224.449236	15.211262	6.917930	70.792179	5.444033	3.14
39	39	82.444802	226.112761	13.573680	7.738771	72.871825	3.985293	4.1
40	40	82.273436	227.683033	12.447090	8.002746	70.242845	4.789766	3.81
41	41	79.327038	226.761532	13.820191	8.616808	70.595601	5.457683	4.14
42	42	77.076647	225.603449	14.106970	7.245625	69.744511	5.377930	3.26
43	43	81.534923	231.204331	13.107536	7.725791	69.032884	4.167118	3.63698112
44	44	87.467687	234.407050	11.399230	7.714124	70.055657	5.328738	4.12
45	45	80.063595	229.929342	12.711279	9.468605	69.634076	4.030549	3.84
46	46	79.074003	228.769911	12.532461	8.012628	70.886756	5.080336	2.70
47	47	77.362819	234.381617	11.590470	7.476719	71.116429	5.695193	4.2
48	48	73.675714	NaN	12.145358	7.363693	70.946153	5.773341	3.73
49	49	77.896586	233.617917	11.418882	7.441132	69.985603	5.665430	4.17

In [156…

```
print("INFO")
df.info() # Verificando la informacion general
print("DISCRIBE")
df.describe() #Estadisrticas basicas
```

```
INFO
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   A_id         4000 non-null   int64
 1   Size         3880 non-null   float64
 2   Weight       3880 non-null   float64
 3   Sweetness    4000 non-null   float64
 4   Crunchiness  3880 non-null   float64
 5   Juiciness    4000 non-null   float64
 6   Ripeness     4000 non-null   float64
 7   Acidity      3881 non-null   object
 8   Quality      4000 non-null   object
dtypes: float64(6), int64(1), object(2)
memory usage: 281.4+ KB
DISCRIBE
```

Out[156...

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	R
count	4000.000000	3880.000000	3880.000000	4000.000000	3880.000000	4000.000000	4000
mean	1999.500000	94.515423	286.188355	12.717713	9.280662	70.307271	5
std	1154.844867	87.595191	327.681811	1.166064	10.257560	1.158171	0
min	0.000000	66.411764	218.560244	8.863309	3.972471	66.422862	3
25%	999.750000	76.616864	226.867309	11.956945	7.052908	69.519229	4
50%	1999.500000	79.152324	228.518195	12.697145	7.528237	70.320531	5
75%	2999.250000	81.867842	230.267036	13.481153	7.995647	71.101586	5
max	3999.000000	640.367782	2233.299942	16.824949	83.198105	74.418642	7

Analisis

- Contamos 9 columnas: Ai_id, Size, Weight, Sweetness, Crunchiess, Juiciness, Ripeness, Acidity, Quality.
- Encontramos que las variables Crunchines (120), Weight (120), Size (120), Acidity (119) cuentan con valorees nulos
- Todas la variables son de tipo númerico a excepción de Acidity y Quality que son categoricas. Mientras qie el A_id es entero.
- La variable Acidity no corresponde con su tipo a los datos que contiene

2. Limpieza de datos

- Verifica la existencia de valores nulos o duplicados.
- Asegurate de que los tipos de datos sean correctos.
- Trata los outliers si se identifican.

In [157…
```
#Convirtiendo Acidity a numerico
df['Acidity'] = pd.to_numeric(df['Acidity'], errors='coerce')
```

In [158…
```
#Encontrando valores nulos
df.isnull().sum()
```

Out[158…
```
A_id            0
Size          120
Weight        120
Sweetness       0
Crunchiness   120
Juiciness       0
Ripeness        0
Acidity       120
Quality         0
dtype: int64
```

In [159…
```
#Verificando diferencias en variables categóricas

# 1. Ver valores únicos actuales (para diagnosticar)
print("Valores únicos en 'Calidad' ANTES de la corrección:")
print(df['Quality'].unique())

# 2. Estandarizar a minúsculas (o mayúsculas) y eliminar espacios
df['Quality'] = df['Quality'].str.strip().str.lower()  # " GOOD " -> "good"

# 3. Opcional: Mapear a nombres consistentes (ej: "buena"/"mala")
# df['Calidad'] = df['Calidad'].map({'good': 'buena', 'bad': 'mala'})

# 4. Verificar después de la corrección
print("\nValores únicos en 'Calidad' DESPUÉS de la corrección:")
print(df['Quality'].unique())

# 5. Contar frecuencias
print("\nConteo de valores:")
print(df['Quality'].value_counts())
```

```
Valores únicos en 'Calidad' ANTES de la corrección:
['good' 'bad' 'GOOD' 'BAD']

Valores únicos en 'Calidad' DESPUÉS de la corrección:
['good' 'bad']

Conteo de valores:
Quality
good    2004
bad     1996
Name: count, dtype: int64
```

In [160…
```
#Encontrando valores duplicados
df.duplicated().sum()  # Verificando si hay filas duplicadas
```

Out[160… 0

In [161…
```
#Eliminar filas con valores nulos
df_cleaned = df.dropna()
```

In [162…
```
#Eliminando columna A_id
df_cleaned = df_cleaned.drop(columns=['A_id'])
```

In [163…
```
#Encontrando valores nulos
print(df_cleaned.isnull().sum())
print(df_cleaned.info())
```

```
Size           0
Weight         0
Sweetness      0
Crunchiness    0
Juiciness      0
Ripeness       0
Acidity        0
Quality        0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
Index: 3543 entries, 0 to 3999
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Size         3543 non-null   float64
 1   Weight       3543 non-null   float64
 2   Sweetness    3543 non-null   float64
 3   Crunchiness  3543 non-null   float64
 4   Juiciness    3543 non-null   float64
 5   Ripeness     3543 non-null   float64
 6   Acidity      3543 non-null   float64
 7   Quality      3543 non-null   object
dtypes: float64(7), object(1)
memory usage: 249.1+ KB
None
```

In [ ]:

In [164…

```python
# 1. Copiar el DataFrame original
df_cleane = df_cleaned.copy()

# 2. Seleccionar columnas numéricas
numeric_cols = df_cleaned.select_dtypes(include=['float64', 'int64']).columns

# 3. Función para reemplazar outliers por la mediana
def replace_outliers_with_median(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Calcular la mediana de la columna (sin outliers)
        median = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)][col].media

        # Reemplazar outliers
        df[col] = np.where((df[col] < lower_bound) | (df[col] > upper_bound), media
    return df

# 4. Aplicar la función
df_replaced = replace_outliers_with_median(df_cleaned, numeric_cols)

# 5. Verificar resultados
print("Resumen de cambios:")
print(f"- Outliers reemplazados en: {list(numeric_cols)}")
print(f"- Ejemplo en 'Peso': Mediana = {df_cleaned['Weight'].median():.2f}")
```

```
Resumen de cambios:
- Outliers reemplazados en: ['Size', 'Weight', 'Sweetness', 'Crunchiness', 'Juicines
s', 'Ripeness', 'Acidity']
- Ejemplo en 'Peso': Mediana = 228.43
```

In [165…

```python
print(df_cleaned.info())
print(df_cleaned.head())
print(df_cleaned.describe)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3543 entries, 0 to 3999
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Size         3543 non-null   float64
 1   Weight       3543 non-null   float64
 2   Sweetness    3543 non-null   float64
 3   Crunchiness  3543 non-null   float64
 4   Juiciness    3543 non-null   float64
 5   Ripeness     3543 non-null   float64
 6   Acidity      3543 non-null   float64
 7   Quality      3543 non-null   object 
dtypes: float64(7), object(1)
memory usage: 249.1+ KB
None
        Size      Weight  Sweetness  Crunchiness  Juiciness  Ripeness  \
0  72.456908  225.980262  12.688757     6.493996  71.106940  5.098952   
1  77.729087  225.457190  15.198435     7.497543  70.511971  5.260259   
2  79.445155  227.837949  11.956943     6.828692  71.703181  4.988590   
3  78.751328  226.365397  13.794924     6.951063  72.182782  3.975872   
4  82.592012  227.925421  12.769205     6.723497  71.818525  4.608845   

    Acidity Quality  
0  3.501682    good  
1  3.455438    good  
2  4.124327     bad  
3  3.604264    good  
4  3.700397    good  
<bound method NDFrame.describe of            Size      Weight  Sweetness  Crunchines
s  Juiciness  Ripeness  \
0     72.456908  225.980262  12.688757     6.493996  71.106940  5.098952   
1     77.729087  225.457190  15.198435     7.497543  70.511971  5.260259   
2     79.445155  227.837949  11.956943     6.828692  71.703181  4.988590   
3     78.751328  226.365397  13.794924     6.951063  72.182782  3.975872   
4     82.592012  227.925421  12.769205     6.723497  71.818525  4.608845   
...         ...         ...        ...          ...        ...       ...   
3994  82.816765  225.870110  12.815867     7.763938  70.633817  5.768249   
3995  80.112834  228.292147  10.771271     7.236526  71.018792  5.673216   
3997  74.994421  226.578805  11.535723     7.328611  71.319825  6.429158   
3998  72.384793  227.153061  14.419838     6.899835  71.296861  5.064347   
3999  80.529225  227.255192  13.072730     6.422963  70.760006  4.767029   

       Acidity Quality  
0     3.501682    good  
1     3.455438    good  
2     4.124327     bad  
3     3.604264    good  
4     3.700397    good  
...        ...     ...  
3994  3.354149    good  
3995  3.627557     bad  
3997  3.333078     bad  
3998  3.154056    good  
3999  3.919959    good
```

```
[3543 rows x 8 columns]>
```

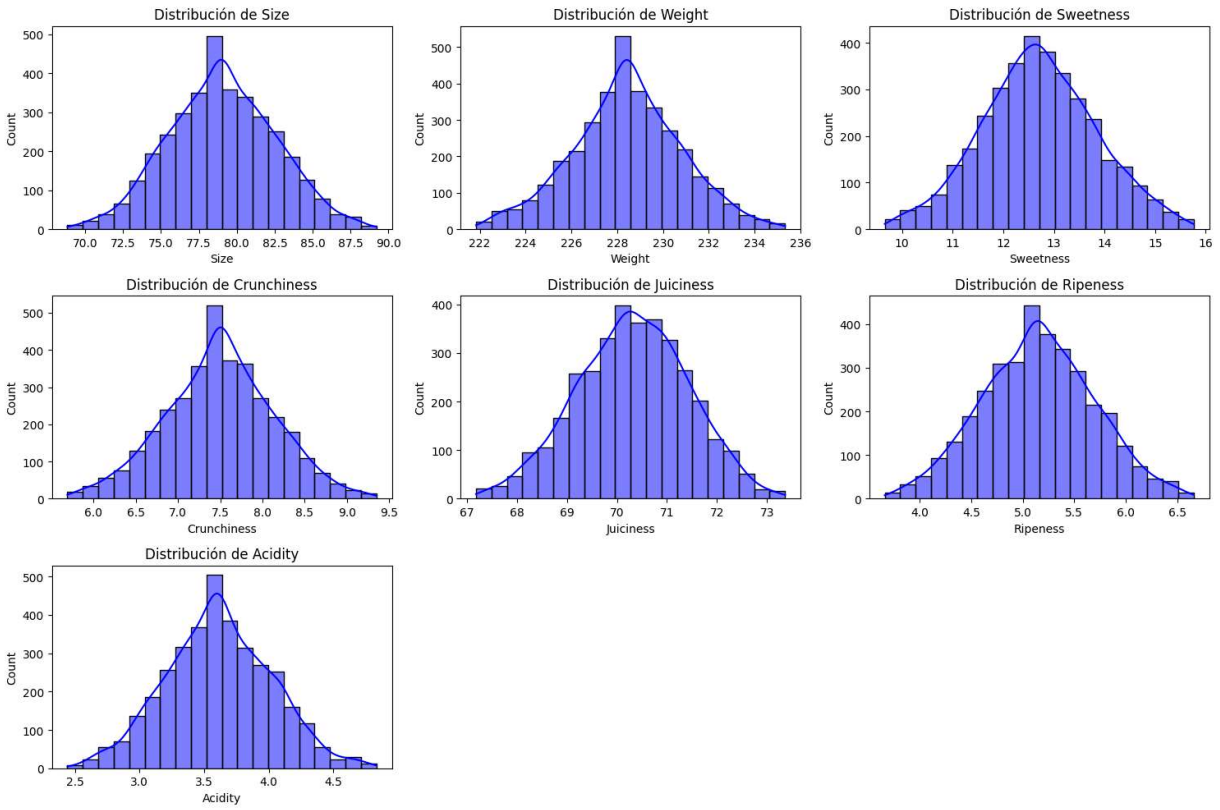
- Se elimino la columna A_id ya que no tenia utilidad en nuestro analisis
- Se eliminaron las filas que contenian NaN ya que correspondian a menos del 12% del total del df
- Se estandarizaron los outliners usando la mediana de cada variable

3. Analisis exploratorio de datos (EDA)

- Analiza las distribuciones de cada variable (histogramas, etc).
- Usa graficas de dispersin y boxplots para comparar variables con la calidad.
- Muestra la matriz de correlacion y un mapa de calor

In [166…

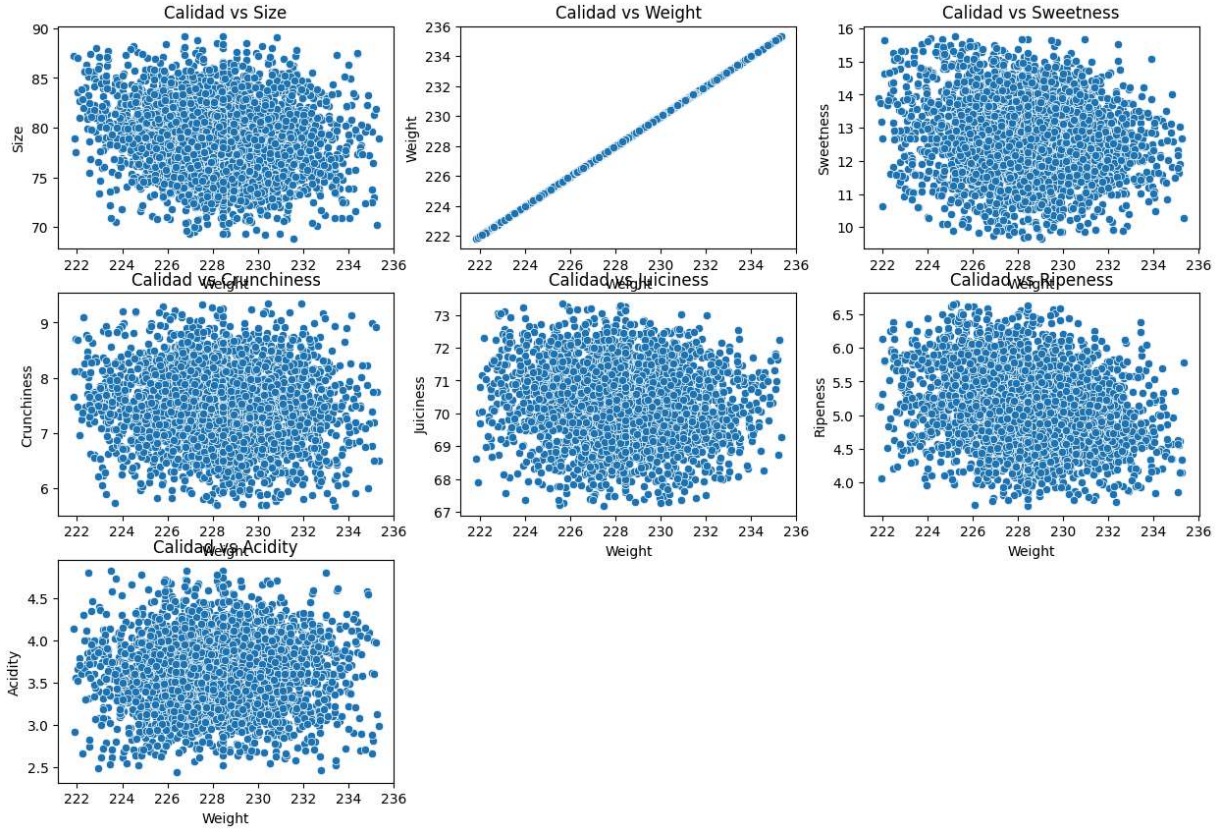
```
#Generando histograma para todo el df_cleaned
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(df_cleaned[col], kde=True, bins=20, color='blue')
    plt.title(f'Distribución de {col}')
plt.tight_layout()
plt.show()
```



In [167…

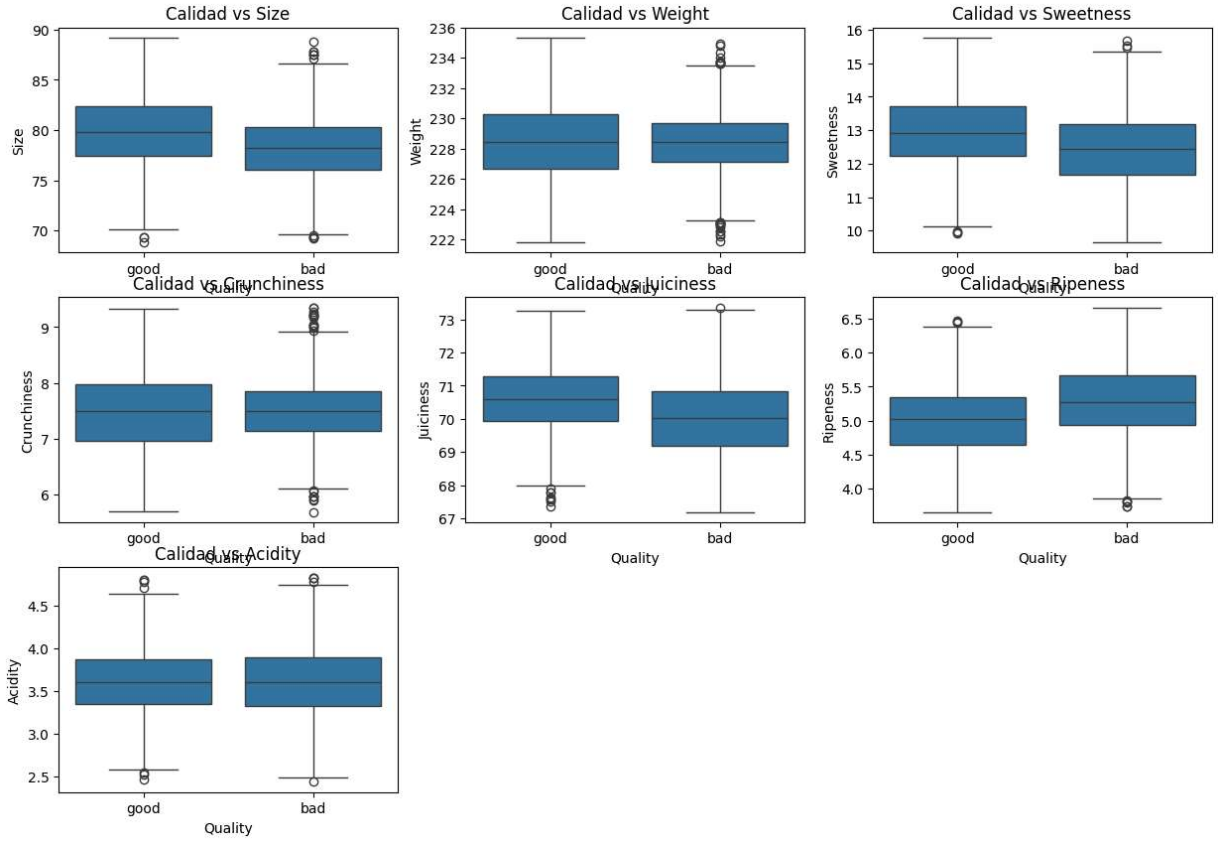
```
# Graficas de dispersiom para comparar las variables
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols, 1):
    if col != 'Quality':  # Evitar comparar con la columna 'Weight'
```

```
        plt.subplot(3, 3, i)
        sns.scatterplot(data=df_cleaned, x='Weight', y=col)
        plt.title(f'Calidad vs {col}')
```



In [168…

```
# boxplor para comparar las variables con quality
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols, 1):
    if col != 'Quality':  # Evitar comparar con la columna 'Weight'
        plt.subplot(3, 3, i)
        sns.boxplot(data=df_cleaned, x='Quality', y=col)
        plt.title(f'Calidad vs {col}')
```



In [171…
```
#Transformando la columna Quality a numérica
df_cleaned['Quality'] = df_cleaned['Quality'].map({'good': 1, 'bad': 0})
# Verificando la transformación
print("\nValores únicos en 'Quality' DESPUÉS de la transformación:")
print(df_cleaned['Quality'].unique())
```

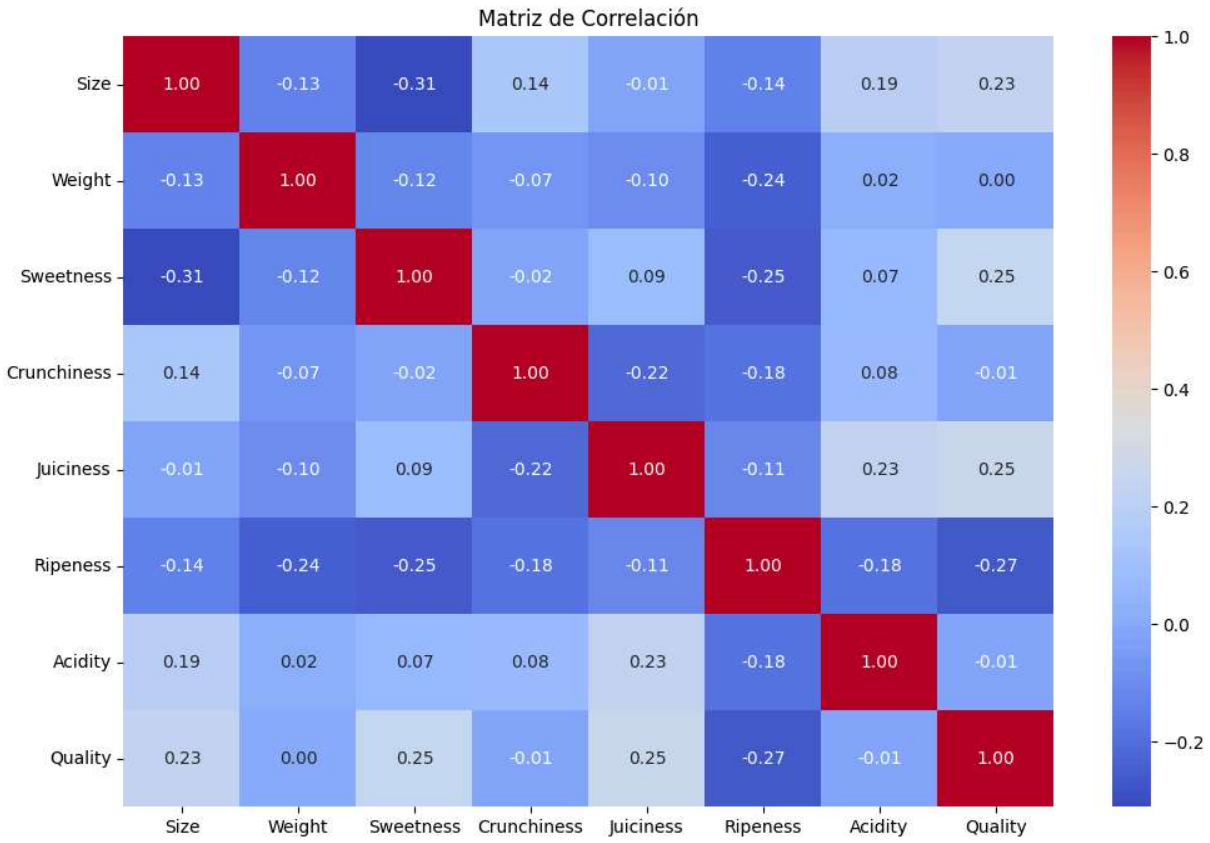
```
Valores únicos en 'Quality' DESPUÉS de la transformación:
[1 0]
```

In [173…
```
#Revisando tipos de datos
df_cleaned.dtypes
```

Out[173…
```
Size           float64
Weight         float64
Sweetness      float64
Crunchiness    float64
Juiciness      float64
Ripeness       float64
Acidity        float64
Quality          int64
dtype: object
```

In [172…
```
#Generando matriz de correlación
plt.figure(figsize=(12, 8))
sns.heatmap(df_cleaned.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matriz de Correlación')
```

Out[172…
```
Text(0.5, 1.0, 'Matriz de Correlación')
```



4. Preparacion para el modelado

- Codifica la variable calidad como binaria (en caso de ser necesario).
- Aplica normalizaci´ on o estandarizaci´ on si es necesario.
- Divide los datos en conjuntos de entrenamiento y prueba (80/20)

In [176…
```
#Separando variables independientes y dependientes
X = df_cleaned.drop(columns=['Quality'])
y = df_cleaned['Quality']
```

In [177…
```
#Estandarizando las variables independientes
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

print("Datos estandarizados:")
print(X_scaled.head())
```

```
Datos estandarizados:
       Size    Weight  Sweetness  Crunchiness  Juiciness  Ripeness   Acidity
0 -1.866793 -1.024784  -0.014789    -1.553219   0.715551 -0.093476 -0.273529
1 -0.370827 -1.243653   2.203673     0.006988   0.181402  0.202378 -0.386566
2  0.116102 -0.247472  -0.661685    -1.032869   1.250842 -0.295891  1.248449
3 -0.080769 -0.863632   0.963022    -0.842621   1.681416 -2.153321 -0.022781
4  1.009014 -0.210871   0.056325    -1.196415   1.354394 -0.992382  0.212204
```

In [178…
```
#Dividiendo los datos en entrenamiento y prueba
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ran

print("\nTamaños de los conjuntos:")
print(f"- Entrenamiento: {X_train.shape}, {y_train.shape}")
print(f"- Prueba: {X_test.shape}, {y_test.shape}")
```

```
Tamaños de los conjuntos:
- Entrenamiento: (2834, 7), (2834,)
- Prueba: (709, 7), (709,)
```

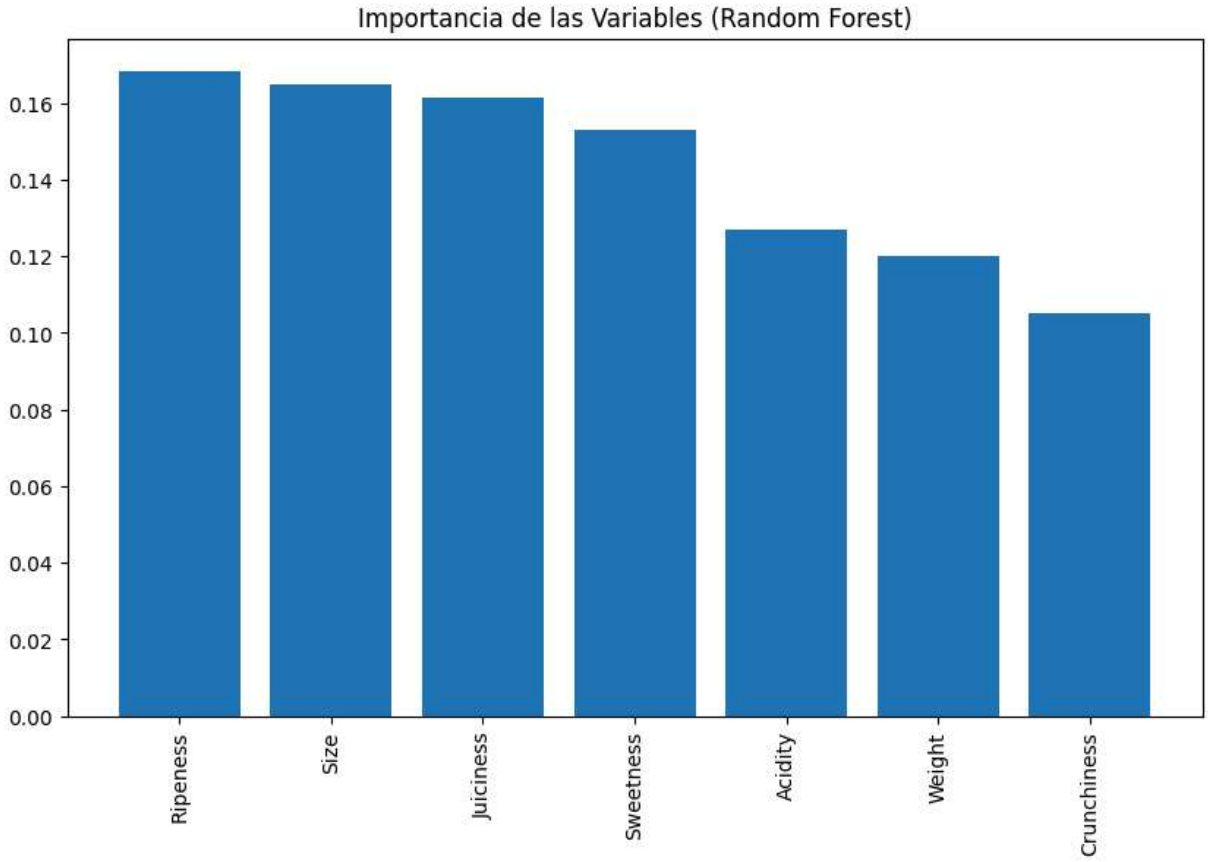
5. Modelado predictivo

- Entrena al menos dos modelos de clasificacion supervisada entre los siguientes:
- Regresión
- Arboles de Decision
- Random Forest
- SVM
- Evalua cada modelo utilizando:
- Accuracy
- Matriz de confusion
- Precision, Recall y F1-Score

In [187…
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Modelos (configuración básica)
models = {
    "Regresión Logística": LogisticRegression(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM": SVC(random_state=42)
}

# Entrenamiento y evaluación
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"\n--- {name} ---")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))

#Graficando la importancia de las variables
plt.figure(figsize=(10, 6))
importances = models["Random Forest"].feature_importances_
indices = np.argsort(importances)[::-1]
plt.bar(range(X_train.shape[1]), importances[indices], align='center')
plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
plt.title('Importancia de las Variables (Random Forest)')
```

```
--- Regresión Logística ---
Accuracy: 0.7545839210155149
              precision    recall  f1-score   support

           0       0.75      0.76      0.75       353
           1       0.76      0.75      0.75       356

    accuracy                           0.75       709
   macro avg       0.75      0.75      0.75       709
weighted avg       0.75      0.75      0.75       709


--- Random Forest ---
Accuracy: 0.8688293370944993
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       353
           1       0.86      0.88      0.87       356

    accuracy                           0.87       709
   macro avg       0.87      0.87      0.87       709
weighted avg       0.87      0.87      0.87       709


--- SVM ---
Accuracy: 0.8575458392101551
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       353
           1       0.87      0.85      0.86       356

    accuracy                           0.86       709
   macro avg       0.86      0.86      0.86       709
weighted avg       0.86      0.86      0.86       709
```

Out[187… Text(0.5, 1.0, 'Importancia de las Variables (Random Forest)')



6. Conclusiones

Redacta una seccion con respuestas a:

- ¿Que modelo funciono mejor y por que?
  - Considero que el modelo que tuvo el mejor resultado fue Random forest, ya que accuarcy esta en 87%
- ¿Que variables fueron mas relevantes?
  - Las variables mas importantes fueron el tamaño y la madurez.
- ¿Que limitaciones tuvo el analisis?
  - Casi ningun variable destaca por arriba de otra al momento de evaluar la calidad.
- ¿Como se podria mejorar el modelo con mas datos?
  - Pues probablemente con opiniones de los consumidores en forma de categorias.