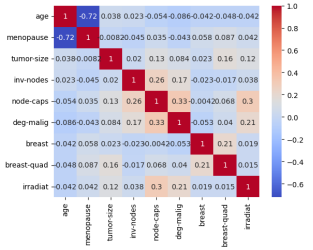
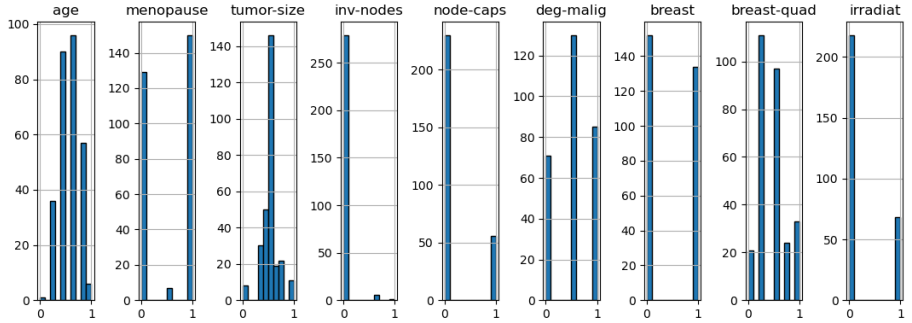
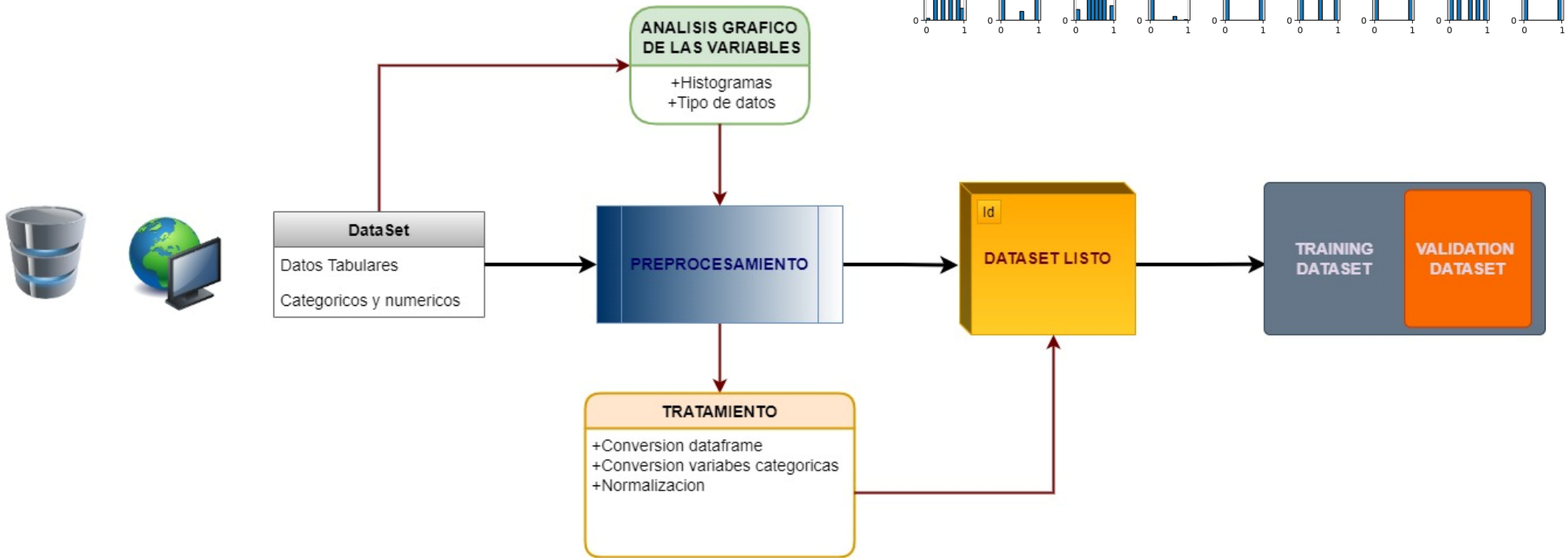
1.- PREPROCESAMIENTO








```python
# Conversión de variables categóricas a numéricas utilizando Label Encoding
label_encoder = LabelEncoder()
categorical_columns = ['node-caps', 'breast', 'irradiat', 'menopause', 'breast-quad']

for column in categorical_columns:
    df_combined[column] = label_encoder.fit_transform(df_combined[column])

# Normalización de variables numéricas entre 0 y 1
numeric_columns = df_combined.columns
#scaler = StandardScaler()
scaler = MinMaxScaler()
df_combined[numeric_columns] = scaler.fit_transform(df_combined[numeric_columns])
```

Después del preprocesamiento:

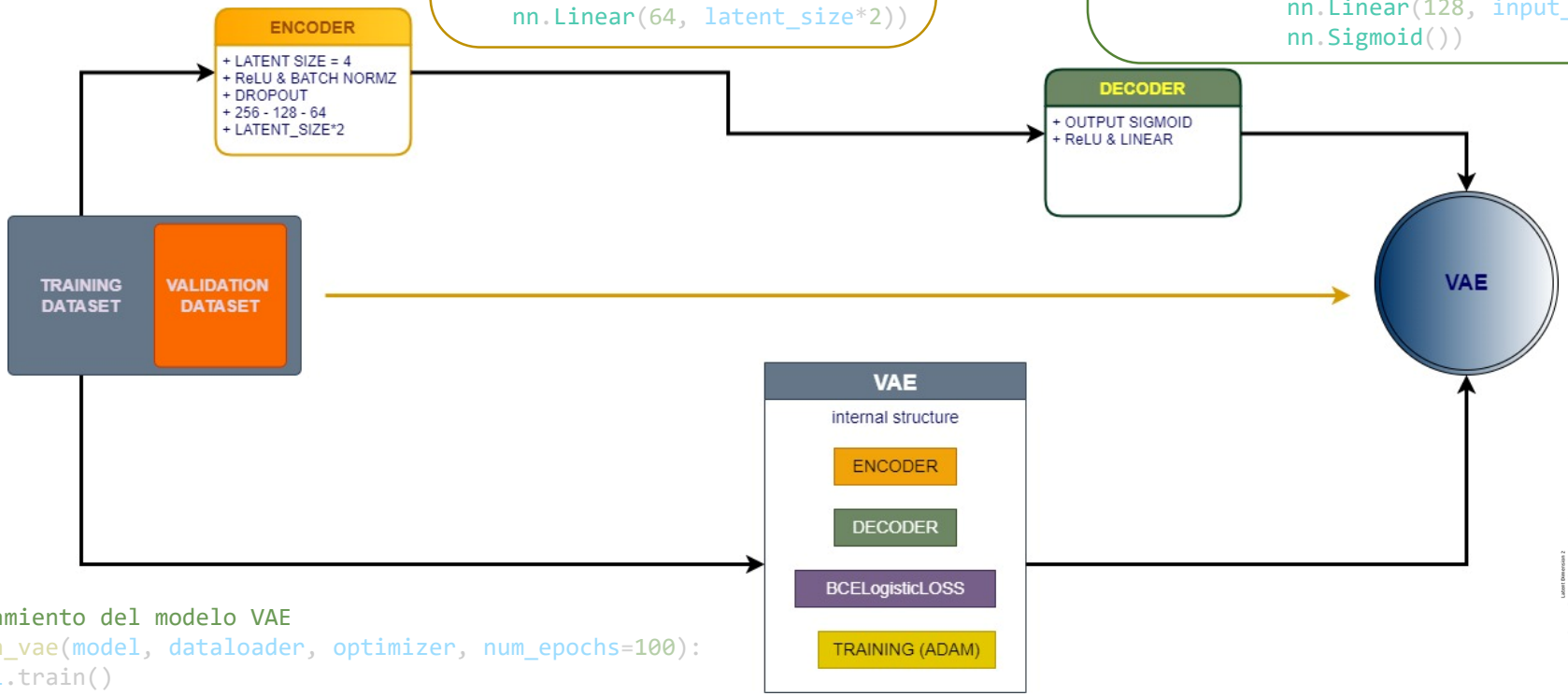
	age	menopause	tumor-size	inv-nodes	node-caps		
0	0.2	1.0	0.6	0.0	0.0	1.0	0.0
1	0.4	1.0	0.4	0.0	0.0	0.5	1.0
2	0.4	1.0	0.4	0.0	0.0	0.5	0.0
3	0.8	0.0	0.3	0.0	0.0	0.5	1.0
4	0.4	1.0	0.0	0.0	0.0	0.5	1.0

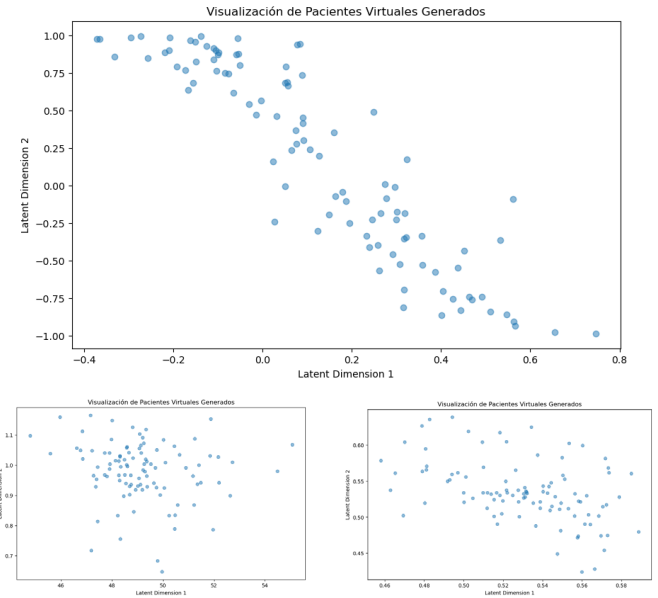
2.- GENERADOR DE DATOS

*** He probado distintas capas asi como activar o no la normalizacion (Batch) etc etc*

```python
# Encoder
self.encoder = nn.Sequential(
    nn.Linear(input_size, 128),
    nn.ReLU(),
    nn.BatchNorm1d(128),
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.BatchNorm1d(64),
    nn.Linear(64, latent_size*2))
```

```python
# Decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_size, 64),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.BatchNorm1d(128),
            nn.Linear(128, input_size),
            nn.Sigmoid())
```






```python
# Entrenamiento del modelo VAE
def train_vae(model, dataloader, optimizer, num_epochs=100):
    model.train()

    for epoch in range(num_epochs):
        for batch in dataloader:
            data = batch[0]
            optimizer.zero_grad()
            z_mean, z_log_var, z, reconstruction = model(data)
            # Calcular la pérdida VAE
            total_loss, reconstruction_loss, kl_loss = vae_loss(reconstruction,
data, z_mean, z_log_var)

            total_loss.backward()
            optimizer.step()
```

```python
# Función de pérdida VAE con BCEWithLogitsLoss
def vae_loss(reconstruction, data, z_mean, z_log_var):
    reconstruction_loss = nn.BCEWithLogitsLoss(reduction='sum')(reconstruction, data)
    # Pérdida KL
    kl_loss = -0.5 * torch.sum(1 + z_log_var - z_mean.pow(2) - z_log_var.exp())
    # Pérdida total
    total_loss = reconstruction_loss + kl_loss
    return total_loss, reconstruction_loss, kl_loss
```

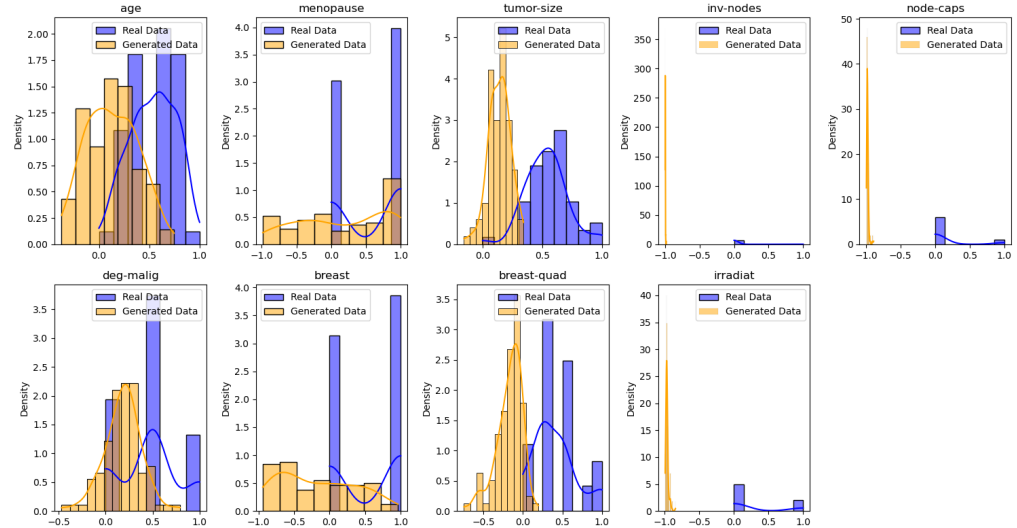
3.- VALIDACION PACIENTES VIRTUALES

3A VALIDACION CON DISTRIBUCION MARGINAL

```
def compare_marginal_distributions(real_data, generated_data, feature_names):
    # Crear subgráficos para cada covariable
    fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(15, 8))

    # Iterar sobre las covariables
    for i, ax in enumerate(axes.flatten()):
        # Verificar si hay más covariables a mostrar
        if i < len(feature_names):
            # Visualizar histogramas y densidades para pacientes reales y virtuales
            sns.histplot(real_data[:, i], kde=True, color='blue', ax=ax, label='Real Data', stat='density')
            sns.histplot(generated_data[:, i], kde=True, color='orange', ax=ax, label='Generated Data', stat='density')
            ax.set_title(feature_names[i])  # Establecer el título de la covariable
            ax.legend()  # Mostrar leyenda

        else:
            # Si no hay más covariables, desactivar los subgráficos restantes
            ax.axis('off')
```



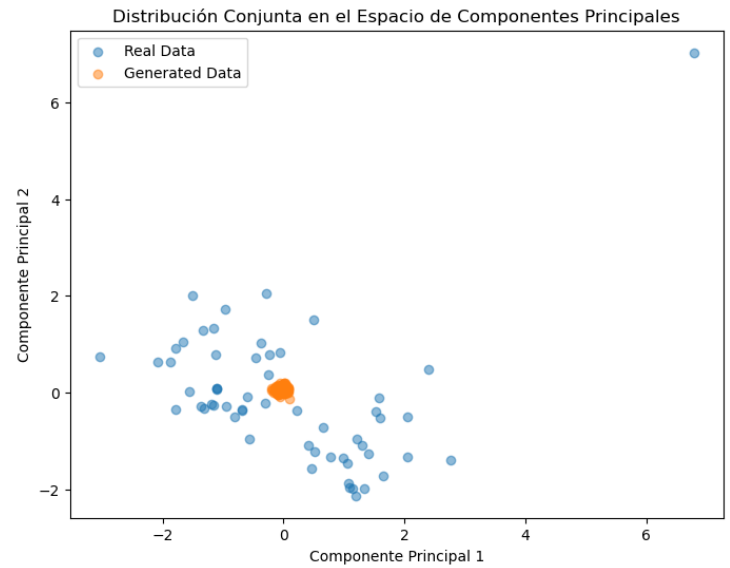
3B

```
# Función para validar usando un clasificador con distribución conjunta
def validate_with_joint_distribution(real_data, generated_data):
    # Etiquetas para clasificación (1 para real, 0 para virtual)
    y_real = np.ones(len(real_data))
    y_generated = np.zeros(len(generated_data))

    # Combinar datos reales y virtuales
    X = np.vstack([real_data, generated_data])
    y = np.concatenate([y_real, y_generated])

    # Entrenar un clasificador (Random Forest en este caso)
    classifier = RandomForestClassifier(n_estimators=100, random_state=42)
    classifier.fit(X, y)

    # Hacer predicciones y calcular métricas
    y_pred = classifier.predict(X)
    accuracy = accuracy_score(y, y_pred)
    report = classification_report(y, y_pred)
```



3.- VALIDACION PACIENTES VIRTUALES

Prueba de Mann-Whitney para inv-nodes:
Estadística de la prueba U: 5800.0
Valor p: 8.774845425833869e-27
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para node-caps:
Estadística de la prueba U: 5800.0
Valor p: 2.1684685102850063e-26
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para deg-malig:
Estadística de la prueba U: 4316.0
Valor p: 2.9129139740596515e-07
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para breast:
Estadística de la prueba U: 4786.0
Valor p: 7.632893105569979e-12
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para breast-quad:
Estadística de la prueba U: 5728.0
Valor p: 1.5513375604734579e-24
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para irradiat:
Estadística de la prueba U: 5800.0
Valor p: 4.6411411517614344e-26
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para age:
Estadística de la prueba U: 5085.0
Valor p: 2.9637570293685484e-15
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para menopause:
Estadística de la prueba U: 4425.0
Valor p: 3.102764863108536e-08
Resultado: Diferencia significativa entre las distribuciones.

Prueba de Mann-Whitney para tumor-size:
Estadística de la prueba U: 5689.0
Valor p: 7.5670290477332e-24
Resultado: Diferencia significativa entre las distribuciones.