1. Evaluación de los resultados respecto a la problemática definida:

In [1]:
```
# Recordatorio del objetivo inicial: Identificación de factores que influyen
import pandas as pd

# Cargar el conjunto de datos
data = pd.read_csv('heart_disease_uci.csv')

# Ver las primeras filas
data.head()

# Filtrar las columnas más relevantes para la evaluación del problema
# Ejemplo: 'trestbps', 'chol', 'num' fueron clave en el análisis previo
relevant_columns = ['trestbps', 'chol', 'num']
filtered_data = data[relevant_columns].dropna()

# Estadísticas descriptivas
mean_trestbps = filtered_data['trestbps'].mean()
std_trestbps = filtered_data['trestbps'].std()
mean_chol = filtered_data['chol'].mean()
std_chol = filtered_data['chol'].std()

print(f"Media Presión Arterial: {mean_trestbps}, Desviación Estándar: {std_t
print(f"Media Colesterol: {mean_chol}, Desviación Estándar: {std_chol}")
```

```
Media Presión Arterial: 132.08992805755395, Desviación Estándar: 19.077093
087446126
Media Colesterol: 200.0335731414868, Desviación Estándar: 110.527172108274
61
```

2. Evaluación del rendimiento del modelo:

Se sugiere comparar métricas de rendimiento del modelo como el coeficiente de determinación (R^2) para ver si el modelo explica suficientemente bien la variabilidad de los datos:

In [2]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Variables: X = trestbps, y = chol
X = filtered_data['trestbps'].values.reshape(-1, 1)
y = filtered_data['chol'].values

# Crear y entrenar el modelo
model = LinearRegression()
model.fit(X, y)

# Predicción
y_pred = model.predict(X)

# Calcular R²
r2 = r2_score(y, y_pred)
print(f"Coeficiente de Determinación R²: {r2}")
```

```
Coeficiente de Determinación R²: 0.008621722952801059
```

3. Procesamiento adicional o recopilación de más datos:

Se puede probar diferentes transformaciones o agregar nuevas variables como la edad (age), el sexo (sex), o variables categóricas como cp (tipo de dolor en el pecho):

In [3]:
```
# Agregar nuevas variables al análisis
new_columns = ['age', 'sex', 'cp']
new_filtered_data = data[new_columns + relevant_columns].dropna()

# Visualizar correlaciones entre variables
correlation_matrix = new_filtered_data.corr()
print(correlation_matrix)
```

```
               age  trestbps      chol       num
age       1.000000  0.253467 -0.088470  0.334203
trestbps  0.253467  1.000000  0.092853  0.128628
chol     -0.088470  0.092853  1.000000 -0.251563
num       0.334203  0.128628 -0.251563  1.000000
```

4. Experimento con hiperparámetros:

El uso de modelos más complejos como el de regresión de Lasso y Ridge que incluyen regularización podría ser útil si hay sobreajuste:

In [4]:
```
from sklearn.linear_model import Lasso, Ridge

# Entrenamiento con Lasso (Regularización L1)
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X, y)

# Entrenamiento con Ridge (Regularización L2)
ridge_model = Ridge(alpha=0.1)
ridge_model.fit(X, y)
```

Out[4]:
```
Ridge(alpha=0.1)
```

5. Creación de nuevas características:

Se pueden crear nuevas variables combinando variables existentes, como ratios:

In [5]:
```
# Creación de nuevas características
data['chol_trestbps_ratio'] = data['chol'] / data['trestbps']
```

MEJORAMIENTO DEL MODELO

1. Importación de bibliotecas y carga de datos:

Comenzamos cargando las bibliotecas necesarias y los datos.

Desde Jupyter Notebook: Utiliza el comando mágico ! para ejecutar comandos de shell directamente en una celda en este caso especifico para instalar Dash:

In [7]:
```
# Importar bibliotecas necesarias
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

# Cargar los datos
data = pd.read_csv('heart_disease_uci.csv')

# Vista preliminar de los datos
data.head()
```

Out[7]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False

2. Definir visualizaciones:

utilizamos Plotly para crear gráficos que nos ayudarán a visualizar las variables clave del conjunto de datos. En este caso, se genera un histograma para observar la distribución de la presión arterial en reposo y un gráfico de dispersión para analizar la relación entre la presión arterial y el colesterol. Estas visualizaciones nos permiten identificar patrones iniciales y tener una comprensión preliminar de las correlaciones entre las variables.

In [8]:
```
# Ejemplo de visualización: histograma de la presión arterial en reposo (tre
hist_trestbps = px.histogram(data, x='trestbps', nbins=20, title='Distribuci

# Gráfico de dispersión: Relación entre presión arterial y colesterol
scatter_trestbps_chol = px.scatter(data, x='trestbps', y='chol', color='num'
                                   title='Relación entre Presión Arterial y
```

3. Crear el layout del dashboard:

El layout define la estructura de la interfaz gráfica del dashboard. Utilizamos html.Div y dcc.Graph para organizar los componentes como gráficos y filtros en el dashboard. El filtro interactivo (dropdown) permite a los usuarios seleccionar un tipo de dolor en el pecho para ver cómo afecta las relaciones entre las variables. La disposición lógica de estos elementos garantiza una interfaz intuitiva y fácil de navegar para los usuarios.

In [9]:
```
# Inicializar la aplicación Dash
app = dash.Dash(__name__)

# Layout de la aplicación
app.layout = html.Div([
    html.H1("Dashboard de Análisis de Enfermedades Cardíacas"),
    dcc.Graph(id='hist_trestbps', figure=hist_trestbps),
    dcc.Graph(id='scatter_trestbps_chol', figure=scatter_trestbps_chol),
    html.Label("Seleccionar Tipo de Dolor en el Pecho (cp):"),
    dcc.Dropdown(
        id='cp_filter',
        options=[{'label': 'Tipo {}'.format(i), 'value': i} for i in data['c
        value=data['cp'].unique()[0]
    )
])
```

4. Agregar interactividad con callbacks:

Los callbacks de Dash son fundamentales para actualizar dinámicamente las visualizaciones en función de las interacciones del usuario. En este caso, se configura un callback que toma el valor seleccionado del dropdown y actualiza el gráfico de dispersión según el tipo de dolor en el pecho. Esto permite a los usuarios filtrar los datos y explorar las relaciones entre las variables en tiempo real, mejorando la experiencia interactiva del dashboard.

In [10]:
```
@app.callback(
    Output('scatter_trestbps_chol', 'figure'),
    Input('cp_filter', 'value')
)
def update_graph(selected_cp):
    # Filtrar los datos por el tipo de dolor en el pecho
    filtered_data = data[data['cp'] == selected_cp]

    # Actualizar el gráfico de dispersión
    updated_scatter = px.scatter(filtered_data, x='trestbps', y='chol', colo
                                 title=f'Relación entre Presión Arterial y C
    return updated_scatter
```

5. Contexto y narrativa:

La narrativa es clave para guiar al usuario a través de los datos. Aquí, se añade una descripción que contextualiza los gráficos, explicando qué variables se están visualizando y por qué son importantes. El objetivo es hacer que los datos cuenten una historia coherente, permitiendo al usuario comprender fácilmente los insights clave del análisis sin tener que interpretar todo por sí mismo. Esto también hace el dashboard más accesible para audiencias no técnicas.

In [11]:
```
# En el layout, incluir texto explicativo
app.layout = html.Div([
    html.H1("Dashboard de Análisis de Enfermedades Cardíacas"),
    html.P("Este dashboard explora la relación entre las variables clave rel
           "como la presión arterial y los niveles de colesterol, junto con
    dcc.Graph(id='hist_trestbps', figure=hist_trestbps),
    dcc.Graph(id='scatter_trestbps_chol', figure=scatter_trestbps_chol),
    html.Label("Seleccionar Tipo de Dolor en el Pecho (cp):"),
    dcc.Dropdown(id='cp_filter', options=[{'label': 'Tipo {}'.format(i), 'va
])
```

6. Personalización del dashboard:

En este paso, se ajustan los colores, fuentes y otros aspectos visuales del dashboard para que sea estéticamente agradable y fácil de entender. Una buena personalización no solo mejora la apariencia del dashboard, sino que también puede ayudar a destacar puntos importantes y hacer que la información sea más clara. Utilizar hojas de estilo CSS externas es una buena práctica para mantener un diseño limpio y coherente en toda la aplicación.

In [12]:
```
# Personalización adicional con CSS (si es necesario)
app.css.append_css({
    'external_url': 'https://codepen.io/chriddyp/pen/bWLwgP.css'
})
```

7. Pruebas y despliegue:

Finalmente, se ejecuta la aplicación Dash.

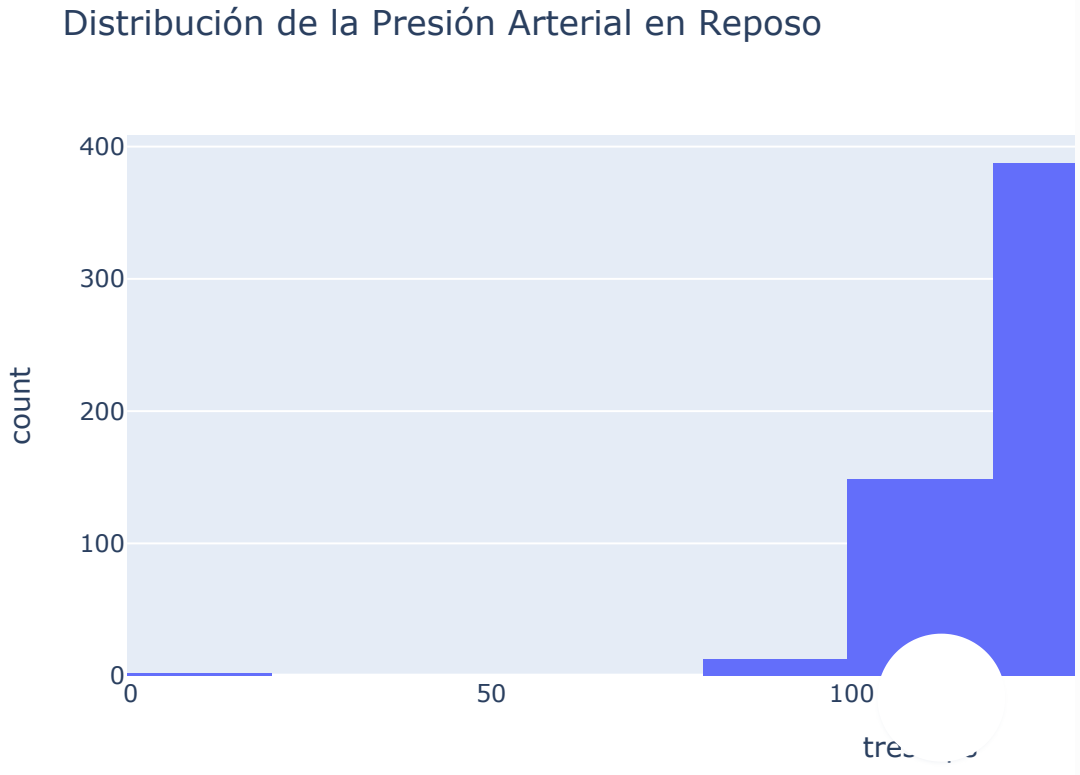
In [13]:
```
if __name__ == '__main__':
    app.run(debug=True)
```

```
C:\Users\jefen\anaconda3\lib\site-packages\dash\resources.py:61: UserWarni
ng:

You have set your config to `serve_locally=True` but A local version of ht
tps://codepen.io/chriddyp/pen/bWLwgP.css (https://codepen.io/chriddyp/pen/
bWLwgP.css) is not available.
If you added this file with `app.scripts.append_script` or `app.css.append
_css`, use `external_scripts` or `external_stylesheets` instead.
See https://dash.plotly.com/external-resources (https://dash.plotly.com/ex
ternal-resources)
```

Dashboard de Análisis de Enfermedades Cardíacas

Este dashboard explora la relación entre las variables clave relacionadas con enfermedades cardíacas, como la presión arterial y los niveles de colesterol, junto con el tipo de dolor en el pecho.



In [ ]: