Ejercicios UT4

Ejercicio 1: Añadir un campo en Ventas

NO:

Modificar en addons/sale/models/sale_order.py:

```
class SaleOrder(models.Model):
    _inherit = "sale.order"
    ...
    x_custom_note = fields.Char(
        string="Nota personalizada"
    )
    ...
```

Con _inherit indicamos el módulo del que heredamos (coincide con el _name original). Además, no incluimos campo _name (si lo dejamos tal cual habrá problemas con el módulo original). El campo name solo lo usaremos cuando creemos módulos nuevos.

Cuando creemos un campo nuevo, por convención, empezará por x, para indicar a los desarrolladores que no es un campo del módulo original (**x**_custom_note)

Con x_custom_note = fields.Char(string="Nota personalizada") creamos un atributo de instancia como string corto de una sola línea (similar a un VARCHAR). Con "Nota personalizada" indicamos el texto breve que se mostrará en el formulario.
x_custom_note será el nombre interno del campo (en código, XML, búsquedas…)

Y crear vista en addons/sale/views/sale_order_view.xml:

```
<odoo>
  <record id="view_order_form_custom" model="ir.ui.view">
    <field name="name">sale.order.form.custom.note</field>
    <field name="model">sale.order</field>
    <field name="inherit_id" ref="sale.view_order_form"/>
    <field name="arch" type="xml">
        <xpath expr="//page[@name='other_information']"
position="inside">
            <group>
                <field name="x_custom_note"/>
            </group>
        </xpath>
    </field>
  </record>
</odoo>
```

Como indicamos un inherit_id, no se crea una nueva pantalla, sino que sustituye a aquella a la que referencia.
Además, inyecta nuevas partes visuales con xpath. En este caso, se incluirá el <group><field /></group> tras la cabecera **sheet**.

Con el ref de sale.view_order_form se referencia al módulo sale, al record con id=view_order_form", que es el que vamos a extender.

model="**ir.ui.view**" es el modelo interno de Odoo que representa vistas de interfaz (pantallas/formularios/listas/kanban…).

Campo del modelo ir.ui.view	Qué guarda
name	nombre "humano" de la vista, para identificarla
model	el modelo de negocio al que pertenece (ej: sale.order)
arch	el XML que define la vista
inherit_id	si es herencia, a qué vista extiende
priority	cuál se carga si hay varias herencias
type	tree, form, kanban, etc.

Tras realizar el cambio, abrimos el fichero **__manifest__.py** de sale y añadimos la línea: **'views/sale_order_view.xml'** (tras 'views/sale_order_views.xml', por ejemplo).

Tras esto:

```
docker exec -u root odoo-app bash -c "odoo -d odoo_demo --db_host=db --db_port=5432 --db_user=odoo --db_password=odoo -u sale --stop-after-init --dev=all"
```

```
docker compose restart odoo
```

De todas formas, **modificar un módulo del core es una mala práctica**, así que vamos a ver otra forma:

Nuestro objetivo va a ser crear esta estructura:

```
<mis_modulos>
├── custom_sales/
│   ├── __init__.py
│   ├── __manifest__.py
│   ├── models/
│   │   ├── __init__.py
│   │   └── sale_order.py
│   ├── views/
│   │   └── sale_order_views.xml
│   └── security/
│       └── ir.model.access.csv
```

Creamos una nueva carpeta para un módulo de odoo en nuestro proyecto y dentro creamos un **__init__.py** (se crea solo si seleccionamos ”python package”):

```
from . import models
```

y **__manifest__.py**:

```
{
    'name': 'Custom Sales - Nota Personalizada',
    'version': '1.0.0',
    'summary': 'Añade campo nota personalizada a pedidos de
venta',
    'description': """
        Añade un campo 'custom_note' al modelo sale.order.
    """,
    'category': 'Sales/Sales',
    'author': 'JAB',
    'license': 'LGPL-3',

    'depends': ['sale'],

    'data': [
        'views/sale_order_views.xml',
        'security/ir.model.access.csv'
    ],
    "assets": {},
    'installable': True,
    'application': False,
    'auto_install': False,
}
```

luego otro python package: **models**, con:

__init__.py:

```
from . import sale_order
```

y **sales_order.py** (la clase que vamos a extender):

```
from odoo import models, fields


class SaleOrder(models.Model):
    _inherit = 'sale.order'

    custom_note = fields.Char(
        string='Nota Personalizada',
        tracking=True,
        help='Nota personalizada para este pedido de venta'
    )
```

Además, en views crearemos el fichero para la vista:
sale_order_views.xml

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <!-- Extender formulario de pedido de venta -->
    <record id="view_order_form_custom" model="ir.ui.view">
        <field name="name">sale.order.form.custom</field>
        <field name="model">sale.order</field>
        <field name="inherit_id" ref="sale.view_order_form"/>
        <field name="arch" type="xml">
            <xpath expr="//page[@name='other_information']"
position="inside">
                <group string="Campos Personalizados">
                    <field name="custom_note"
                           placeholder="Escribe una nota
personalizada aquí..."
                           class="oe_inline"/>
                </group>
            </xpath>
        </field>
    </record>

    <!-- Extender vista lista -->
    <record id="view_order_tree_custom" model="ir.ui.view">
        <field name="name">sale.order.tree.custom</field>
        <field name="model">sale.order</field>
        <field name="inherit_id" ref="sale.view_order_tree"/>
        <field name="arch" type="xml">
            <xpath expr="//field[@name='partner_id']"
position="after">
                <field name="custom_note" string="Nota"
                       optional="show"
                       class="o_optional_columns"/>
            </xpath>
```

```
        </field>
    </record>
</odoo>
```

Aquí indicamos tanto que aparezca en el formulario de pedido como en el listado de pedidos.

Por último, opcionalmente, podemos crear un fichero de control de acceso: **ir.model.access.csv**:

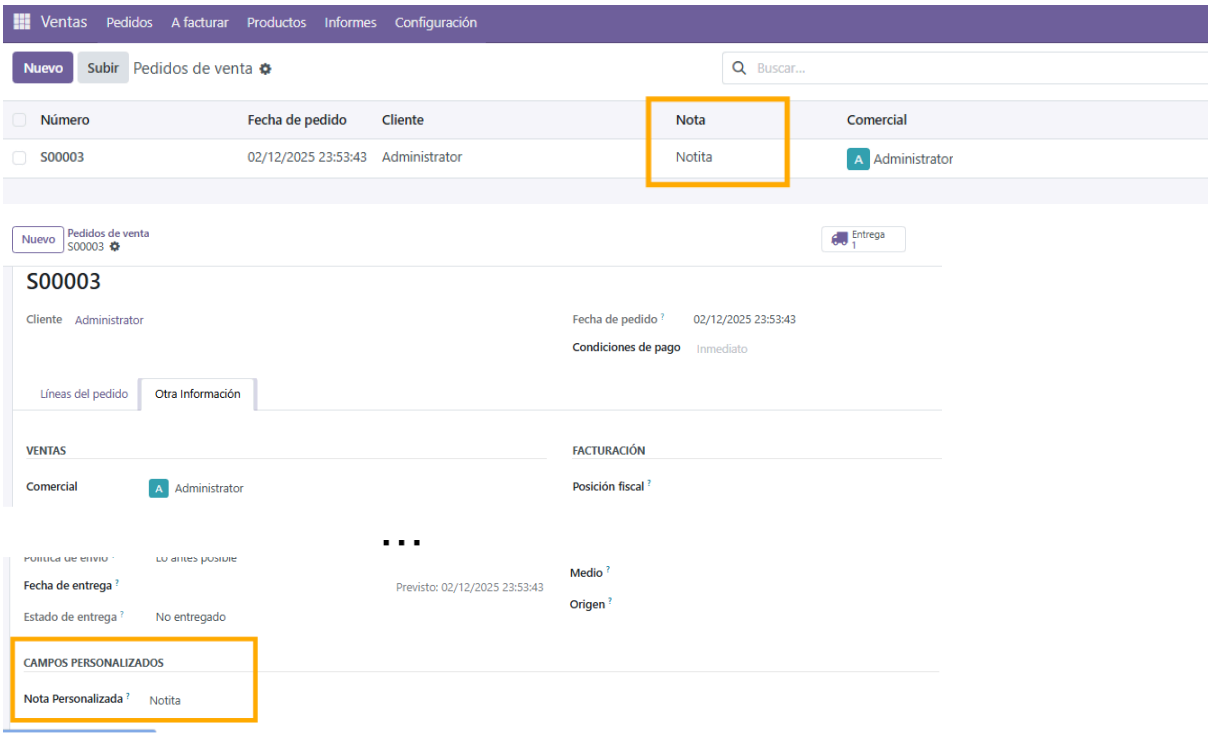
```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,
perm_unlink
access_custom_sales,custom_sales.access_sale_order_custom,model_s
ale_order,base.group_user,1,1,1,0
```

Una vez hecho todo esto, le decimos a odoo que instale el nuevo módulo:

```
docker compose exec -u odoo odoo odoo -d odoo_demo --db_host=db
--db_port=5432 --db_user=odoo --db_password=odoo
--addons-path=/usr/lib/python3/dist-packages/odoo/addons,/mnt/extra-a
ddons -i custom_sale --stop-after-init
```

y reiniciamos:

```
docker compose restart odoo
```



Ejercicio 2: Cambiar etiquetas de campos

Para cambiar una etiqueta en la vista de Odoo usando hay varias formas dependiendo de qué queramos.

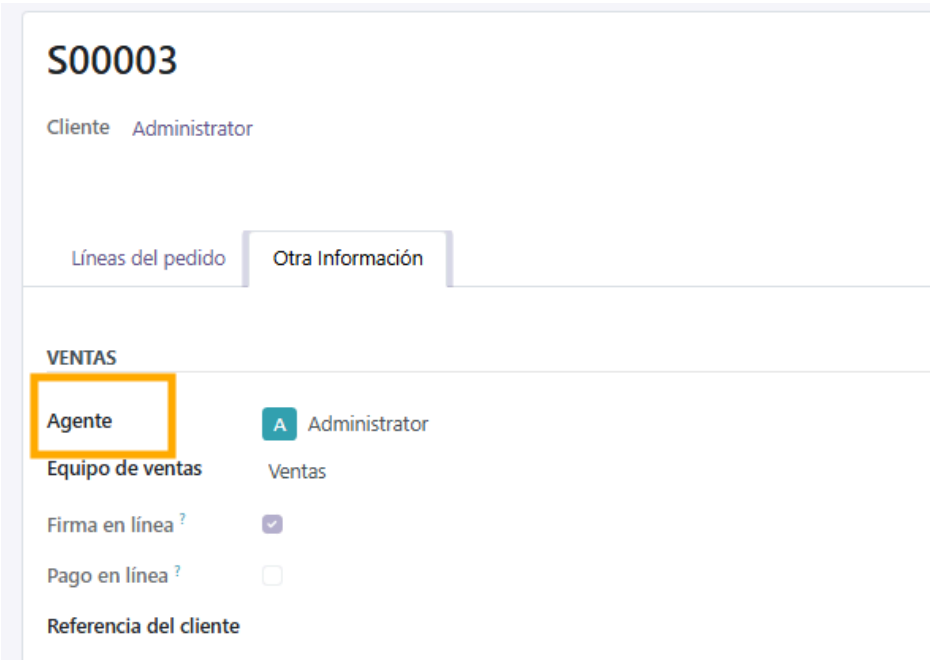
Método 1: Cambiamos la etiqueta del campo existente:

Vamos a suponer que queremos cambiar Comercial por Agente en la pestaña de “Otra Información” de un pedido. En tu módulo custom_sales:

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
   <!-- Extender formulario de pedido de venta -->
   <record id="view_order_form_custom" model="ir.ui.view">
       <field name="name">sale.order.form.custom</field>
       <field name="model">sale.order</field>
       <field name="inherit_id" ref="sale.view_order_form"/>
       <field name="arch" type="xml">
           <xpath expr="//page[@name='other_information']" position="inside">
               <group string="Campos Personalizados">
                   <field name="custom_note"
                          placeholder="Escribe una nota personalizada aquí..."
                          class="oe_inline"/>
               </group>
           </xpath>
           <!-- Usando xpath con atributo-->
           <xpath expr="//field[@name='user_id']" position="attributes">
               <attribute name="string">Agente</attribute>
           </xpath>
       </field>
   </record>

   <!-- Extender vista lista -->
   <record id="view_order_tree_custom" model="ir.ui.view">
       <field name="name">sale.order.tree.custom</field>
       <field name="model">sale.order</field>
       <field name="inherit_id" ref="sale.view_order_tree"/>
       <field name="arch" type="xml">
           <xpath expr="//field[@name='partner_id']" position="after">
               <field name="custom_note" string="Nota"
                      optional="show"
                      class="o_optional_columns"/>
           </xpath>
       </field>
   </record>
</odoo>
```

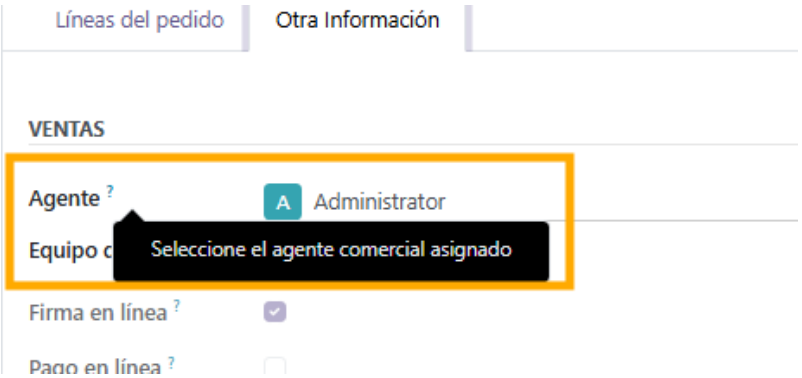
Aquí lo que hacemos es indicar la ruta al atributo que queremos modificar (string) y lo cambiamos al que queremos.



También se puede redefinir el campo, indicando qué atributos queremos que tenga

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <!-- Extender formulario de pedido de venta -->
    <record id="view_order_form_custom" model="ir.ui.view">
        <field name="name">sale.order.form.custom</field>
        <field name="model">sale.order</field>
        <field name="inherit_id" ref="sale.view_order_form"/>
        <field name="arch" type="xml">
            <field name="user_id" position="attributes">
                <attribute name="string">Agente</attribute>
                <attribute name="help">Seleccione el agente comercial
asignado</attribute>
                <attribute name="placeholder">Buscar agente...</attribute>
            </field>
            <!-- Usando xpath con atributo-->
            <xpath expr="//field[@name='user_id']" position="attributes">
                <attribute name="string">Agente</attribute>
            </xpath>
        </field>
    </record>

    <!-- Extender vista lista -->
    <record id="view_order_tree_custom" model="ir.ui.view">
        <field name="name">sale.order.tree.custom</field>
        <field name="model">sale.order</field>
        <field name="inherit_id" ref="sale.view_order_tree"/>
        <field name="arch" type="xml">
            <xpath expr="//field[@name='partner_id']" position="after">
                <field name="custom_note" string="Nota"
                       optional="show"
                       class="o_optional_columns"/>
            </xpath>
        </field>
    </record>
</odoo>
```

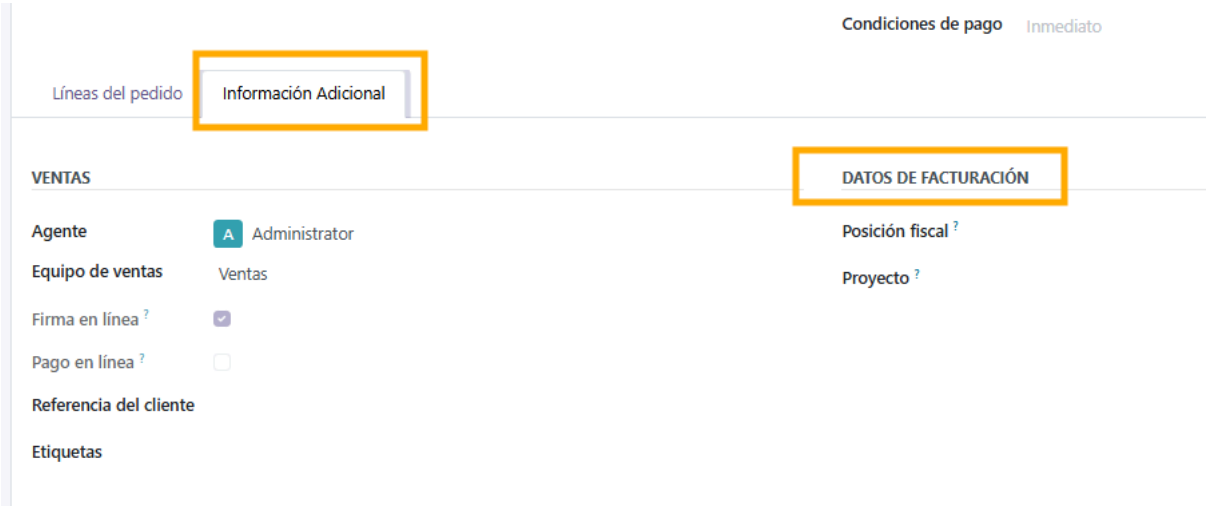


Método 2: Cambiamos la etiqueta del grupo/sección:

Ahora vamos a cambiar etiquetas de pestañas y de agrupaciones:

```
<xpath expr="//page[@name='other_information']"
position="attributes">
   <attribute name="string">Información Adicional</attribute>
</xpath>

<xpath expr="//group[@name='sale_info']" position="attributes">
   <attribute name="string">Datos de Facturación</attribute>
</xpath>
```



Método 3: Cambiamos etiquetas con diccionario de traducción

Creamos un objeto i18n/es.po. (viene de Internacionalización, ESpaña y Portable Object)

```
# Archivo de traducción español
msgid ""
msgstr ""
"Content-Type: text/plain; charset=UTF-8\n"
"Language: es_ES\n"

# ================================
# TRADUCCIÓN del campo custom_notes
# ================================

# Etiqueta del campo
#. module: custom_sale
#:
model:ir.model.fields,field_description:custom_sale.field_sale_order__custo
m_note
msgid "Custom Note"
msgstr "Nota Personalizada"

# Texto de ayuda (tooltip)
#. module: custom_sale
#: model:ir.model.fields,help:custom_sale.field_sale_order__custom_note
msgid "Additional notes for this order"
msgstr "Notas adicionales para este pedido"


# ================================================
# TRADUCCIÓN de elementos de VISTA (opcional)
# ================================================

# Placeholder (texto dentro del campo)
#. module: custom_sale
#: model:ir.ui.view,arch_db:custom_sale.view_order_form_custom
msgid "Write notes here..."
msgstr "Escribe notas aquí..."
```

Y actualizamos el __manifest__.py para recogerlo:

```
'i18n': [
    'i18n/es.po',
    'i18n/en.po',
],
```

```
docker compose exec -u odoo odoo odoo -d odoo_demo --db_host=db
--db_port=5432 --db_user=odoo --db_password=odoo
--addons-path=/mnt/extra-addons,/usr/lib/python3/dist-packages/odoo/a
ddons -u custom_sale --i18n-overwrite --stop-after-init
```

Ejercicio 3: Campos calculados

Nos vamos a nuestro modelo y añadimos un nuevo campo que va a ser calculado (no necesita guardarse en BBDD, por lo que indicamos store=False

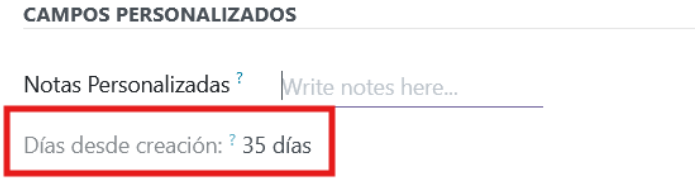
```
order_duration = fields.Integer(
    string='Days since creation',
    compute='_compute_order_duration',
    store=False,
    help='Days elapsed since order creation'
)

@api.depends('date_order')
def _compute_order_duration(self):
    """Calcula días desde creación del pedido"""
    today = date.today()

    for order in self:
        if order.date_order:  # type: ignore
            order_date = order.date_order.date()  # type: ignore
            delta = (today - order_date).days
            order.order_duration = max(0, delta)
        else:
            order.order_duration = 0
```

Y, tras esto lo incluimos en nuestra vista, en el mismo grupo que ya creamos:

```
<xpath expr="//page[@name='other_information']"
position="inside">
    <group string="Campos Personalizados">
        <field name="custom_notes"
               placeholder="Write notes here..."
               class="oe_inline"/>
        <div>
            <label for="order_duration" string="Días desde
creación:"/>
            <field name="order_duration"
                   nolabel="1"
                   readonly="1"
                   class="oe_inline"/>
            <span> días</span>
        </div>
    </group>
</xpath>
```



Ejercicios para hacer

EJERCICIO 4.1: Sistema de Entrega y Logística

Objetivo: Gestionar información de envío y logística

Requisitos:

1. Campo `numero_guia` (Char) - "Número de guía de transporte"
2. Campo `transportista` (Many2one a `res.partner` con dominio solo partners transportistas)
3. Campo `costo_envio` (Monetario) - "Costo de envío adicional"
4. Campo `total_con_envio` (Monetario, calculado) - `amount_total + costo_envio`
5. Visualización: Mostrar `total_con_envio` en negrita si `costo_envio > 0`

Validaciones:

- Si se ingresa `numero_guia`, `transportista` es obligatorio
- `costo_envio` no puede ser negativo

Pistas:

- Usar `domain=[('category_id', 'ilike', 'transportista')]`
- `widget="monetary"` para campos monetarios
- `@api.depends('amount_total', 'costo_envio')`

EJERCICIO 4.2: Gestión de Garantías y Post-Venta

Objetivo: Control de garantías y servicio post-venta

Requisitos:

1. Campo `tiene_garantia` (Booleano) - "Incluye garantía extendida"
2. Campo `meses_garantia` (Integer) - "Meses de garantía" (visible solo si `tiene_garantia=True`)
3. Campo `fecha_fin_garantia` (Date, calculado) - `date_order + meses_garantia meses`
4. Campo `dias_restantes_garantia` (Integer, calculado) - días hasta `fecha_fin_garantia`
5. Visualización: Si `dias_restantes_garantia < 30`, mostrar alerta amarilla

Validaciones:

- Si `tiene_garantia=True`, `meses_garantia` entre 1 y 24
- `fecha_fin_garantia` no puede ser en el pasado al crear

Pistas:

- `attrs` para visibilidad condicional
- `relativedelta` de `dateutil` para cálculos de meses
- `style="background-color: #fffacd;"` para alertas

EJERCICIO 4.3: Sistema de Comisiones por Venta

Objetivo: Cálculo y asignación de comisiones

Requisitos:

1. Campo `porcentaje_comision` (Float) - "% Comisión vendedor" (0-20)
2. Campo `monto_comision` (Monetario, calculado) - `amount_total * porcentaje_comision / 100`
3. Campo `pagada_comision` (Booleano) - "Comisión pagada"
4. Campo `fecha_pago_comision` (Date) - "Fecha pago comisión" (solo si `pagada_comision=True`)
5. Reporte: Añadir columna `monto_comision` a vista lista con totalización

Validaciones:

- Solo usuarios del grupo "Manager" pueden modificar `pagada_comision`
- Si `pagada_comision=True`, `fecha_pago_comision` obligatorio y no futura

Pistas:

- `groups="sales_team.group_sale_manager"` para permisos
- `sum` en vista tree: `<field name="monto_comision" sum="Total Comisiones"/>`
- `@api.onchange('pagada_comision')` para establecer fecha automática

EJERCICIO 4.4: Gestión de Proyectos Asociados

Objetivo: Relacionar pedidos con proyectos

Requisitos:

1. Campo `proyecto_asociado_id` (Many2one a `project.project`) - "Proyecto asociado"
2. Campo `codigo_proyecto` (Char, relacionado) - Código del proyecto (se llena automático)
3. Campo `presupuesto_proyecto` (Monetario, relacionado) - Presupuesto del proyecto
4. Campo `porcentaje_presupuesto` (Float, calculado) - `(amount_total / presupuesto_proyecto) * 100`
5. Validación: Si `porcentaje_presupuesto > 100`, impedir confirmar pedido

Visualización:

- Barra de progreso para `porcentaje_presupuesto`
- Si `porcentaje_presupuesto > 80`, mostrar advertencia

Pistas:

- `related="proyecto_asociado_id.code"` para campo relacionado
- `widget="progressbar"` para porcentaje
- `states="{'confirm': [('readonly', True)]}"` para validación en confirmación

EJERCICIO 4.5: Sistema de Clasificación y Análisis

Objetivo: Clasificar pedidos para análisis comercial

Requisitos:

1. Campo `segmento_cliente` (Selection: Nuevo/Recurrente/Corporate/VIP)
2. Campo `valoracion_dificultad` (Selection: Baja/Media/Alta) - "Dificultad de venta"
3. Campo `puntuacion_venta` (Integer, calculado) - Puntos según:
    - Nuevo: 10 puntos, Recurrente: 5, Corporate: 3, VIP: 15
    - Baja dificultad: +5, Media: +0, Alta: -5
4. Campo `categoria_venta` (Char, calculado) - "Bronce/Plata/Oro" según `puntuacion_venta`
5. Dashboard: Vista kanban agrupada por `categoria_venta` con colores

Cálculos:

- Bronce: < 10 puntos
- Plata: 10-15 puntos
- Oro: > 15 puntos

Visualización:

- Badge con color según categoría en vista lista
- Filtros por segmento y categoría

Pistas:

- `@api.depends('segmento_cliente', 'valoracion_dificultad')`
- `widget="badge"` con `decoration-*` para colores
- `context="{'group_by': 'categoria_venta'}"` para vista kanban