KUTXABANK - DATA SCIENCE - MEMORIA

ORGANIZACIÓN Y PLANIFICACIÓN

INGRESOS Y GASTOS

DESAFÍO TRIPULACIONES
DATA SCIENCE
(Grupo 3)

DESAFÍO TRIPULACIONES KUTXABANK Grupo 3 - The Bridge Digital Talent Accelerator | Data Science
Rodrigo Meza, Josu Goiria, Mikel Cerio, Américo Carrillo, Juan Azpiazu
25/10/2024

Introducción

INTRODUCCIÓN

¿Cuál es el desafío?
Diseñar y desarrollar una solución basada en el concepto de salud financiera para la Banca Móvil (app), que dé respuesta a alguno de los 4 ejes que se exponían en la presentación del desafío.

A saber:

- Se valorará positivamente aspectos sobre educación y asesoramiento financiero.
- Se valorará la adaptación de la solución a las diferentes necesidades de los usuarios según los segmentos de edad.
- No hay que crear nuevos productos financieros.

RETO GRUPO 3: ORGANIZACIÓN Y PLANIFICACIÓN
Capacidad de establecer presupuestos de ingresos y gastos, y de crear retos para la generación de ahorro y la toma de decisión de inversiones.

Datos sintéticos

DATOS SINTÉTICOS - Posiblemente el gran reto del proyecto.

El hecho de no tener acceso a datos de clientes reales de KUTXABANK ha representado probablemente el mayor de los retos del desafío y una parte importante de nuestro trabajo.

Crear datos que sean realistas, que repliquen tendencias y comportamientos de gasto e ingreso diferenciados por usuarios, grupos de consumo, en definitiva, generar históricos de vidas de familias ficticias, es un trabajo bastante complejo y laborioso.

1.- METODOLOGíA

Creación de un script que genera un conjunto de **usuarios sintéticos** con características avanzadas, simula sus **transacciones bancarias** y establece **objetivos de ahorro**.

Utiliza librerías como **NumPy** y **Pandas** para la creación de datos falsos de forma realista.

2.- COMPONENTES CLAVE:

- **Configuración inicial**:
    - Se establece una semilla para garantizar que los resultados sean reproducibles.

Funciones de generación:

- `generar_sueldo(grupo, grupos_caracteristicas)`: Asigna un sueldo a un usuario basado en el grupo al que pertenece.
- `calcular_dinero_banco(grupo, grupos_caracteristicas)`: Calcula un saldo inicial en el banco según las características del grupo.
- `asignar_grupo(probabilidades_grupos)`: Asigna un grupo a un usuario según las probabilidades definidas.
- `generar_iban_kutxabank()`: Genera un IBAN (número de cuenta bancaria) para un banco específico.
- `generar_edad(rango_edad)`: Genera una edad aleatoria para el usuario dentro de un rango.
- `generar_sexo()` y `generar_nombre(sexo)`: Genera el sexo y el nombre del usuario en función de su sexo.
- `generar_apellidos()`: Genera apellidos aleatorios.
- `definir_objetivos_ahorro(grupo)`: Define diferentes objetivos de ahorro para los usuarios según el grupo al que pertenecen (como comprar una casa, pagar estudios o vacaciones).
- `generar_transacciones_corriente(usuario_id, sueldo, gastos_fijos)`: Genera transacciones bancarias en la cuenta corriente del usuario (ingresos y gastos) para un periodo de 36 meses.
- `generar_transacciones_ahorro(usuario_id, objetivos_ahorro, saldo_disponible)`: Simula la generación de transacciones de ahorro, en las que los usuarios apartan un porcentaje de su saldo disponible para cumplir con sus objetivos de ahorro.
    -

Proceso de generación de usuarios:

- `generar_usuarios_avanzados(n, grupos_caracteristicas, probabilidades_grupos, rango_edad, seed=None)`: Esta es la función principal del script. Genera un conjunto de usuarios con características avanzadas, sus respectivas transacciones de cuenta corriente y ahorro, y los objetivos de ahorro. Para cada usuario:
    - Se asignan características personales y financieras (edad, sexo, sueldo, hijos, vivienda, coche).
    - Se generan IBANs únicos para evitar duplicidades.
    - Se definen objetivos de ahorro según el grupo asignado.
    - Se simulan las transacciones bancarias (gastos e ingresos) y, si el usuario tiene saldo disponible, se generan transacciones de ahorro.

Variables configurables:

- `grupos_caracteristicas`: Define las características económicas de cada grupo, como el rango de sueldo, el dinero disponible en el banco, el porcentaje de vivienda propia y la probabilidad de tener un coche o hijos.
- `probabilidades_grupos`: Asigna las probabilidades de pertenencia a cada grupo de usuarios.
- `rango_edad`: Define el rango de edad de los usuarios (25 a 40 años en este caso).
- **Exportación a CSV**:
- Los datos generados (usuarios, objetivos de ahorro, transacciones de cuenta corriente y transacciones de ahorro) se guardan en archivos CSV para su posterior análisis o visualización.

Selección de un usuario aleatorio:

- Finalmente, el script selecciona un usuario aleatorio para imprimir sus transacciones bancarias, tanto en la cuenta corriente como en las cuentas de ahorro.

3.- Resumen de los DataFrames generados:

1. `usuarios_df`: Contiene la información personal y financiera de los usuarios generados (nombre, apellidos, edad, sexo, sueldo, dinero en el banco, IBAN, etc.).
2. `objetivos_ahorro_df`: Lista los objetivos de ahorro de cada usuario (meta de ahorro, monto objetivo, y fecha límite).
3. `transacciones_corriente_df`: Registra todas las transacciones en la cuenta corriente de los usuarios (ingresos y gastos mensuales).
4. `transacciones_ahorro_df`: Detalla las transacciones de ahorro, donde se guarda el dinero disponible de los usuarios para cumplir con sus metas.

Conclusión:

Este script proporciona una manera completa de generar datos de usuarios sintéticos con características realistas, simulando su comportamiento financiero a lo largo de un periodo de tiempo. Esto es útil para crear datasets de prueba para proyectos relacionados con banca, ahorro o simulaciones financieras.

API

API para la Generación de Transacciones Simuladas

Esta API fue desarrollada utilizando **FastAPI** y tiene como objetivo la **generación automatizada de transacciones financieras simuladas**, tanto de ingresos como de gastos, para perfiles de usuarios ficticios. El sistema permite a los usuarios definir el rango de fechas y la cantidad de transacciones a generar, y devuelve los datos en un formato **CSV** descargable, lo que lo hace ideal para su uso en aplicaciones que requieren simulaciones de datos financieros para pruebas o análisis.

1. Estructura General de la API

La API está diseñada en torno a tres elementos principales:

- **Perfiles de usuario**: Generación de perfiles de usuarios ficticios con información básica como nombre, edad, balance en la cuenta, tipo de cuenta y la fecha de creación de la misma.
- **Transacciones**: Creación de transacciones financieras, tanto ingresos como gastos, con categorías predefinidas y montos generados aleatoriamente.
- **Exportación de Datos**: Devolución de los datos generados en formato **CSV**, permitiendo al usuario descargarlos directamente desde el navegador.

El desarrollo de la API se realiza con **FastAPI**, lo que garantiza alta eficiencia, un rendimiento superior y una gestión de rutas y datos sencilla.

2. Componentes Principales

a. Perfiles de Usuario

La función `generar_perfil_usuario(id_usuario)` permite la creación de perfiles básicos para los usuarios ficticios. Cada perfil de usuario contiene:

- **ID de Usuario**: Un número entero que identifica al usuario.
- **Nombre**: Un nombre generado dinámicamente utilizando una plantilla "Usuario_X".
- **Edad**: Un valor aleatorio entre 18 y 70 años.
- **Balance de cuenta**: Un monto inicial aleatorio entre 1,000 y 10,000 euros.
- **Tipo de cuenta**: Se asigna aleatoriamente entre "Ahorros" o "Corriente".
- **Fecha de creación de la cuenta**: Se genera la fecha actual.

Este enfoque permite crear un entorno financiero simulado donde los usuarios se asemejan a perfiles reales, con detalles precisos sobre sus cuentas.

b. Transacciones Simuladas

El núcleo de la API está en la generación de **transacciones financieras simuladas**, que pueden ser tanto **ingresos** como **gastos**. Esto se realiza mediante la función `generar_transacciones(cantidad, fecha_inicio, fecha_fin)`, la cual recibe los siguientes parámetros:

- **Cantidad de transacciones**: Número de transacciones a generar.
- **Fecha de inicio y fin**: Rango de fechas en el cual se distribuyen aleatoriamente las transacciones.

Para cada transacción, el sistema:

1. **Genera montos aleatorios**: Los montos de las transacciones se generan utilizando la función `uniform`, con un rango que va desde -500 (gastos) hasta 500 (ingresos).
2. **Clasifica la transacción**: La función `categorizar_transaccion(monto)` clasifica automáticamente las transacciones como "Ingreso" o "Gasto", basándose en si el monto es positivo o negativo. Luego se asigna una categoría específica para el ingreso o gasto, seleccionada aleatoriamente entre varias categorías predefinidas.
3. **Asigna una fecha aleatoria**: La fecha de la transacción se selecciona aleatoriamente entre la fecha de inicio y la fecha de fin, de manera que las transacciones queden distribuidas a lo largo del tiempo.

Las **categorías de transacciones** para los ingresos incluyen "Salario", "Inversiones" y "Premios", mientras que para los gastos incluyen categorías como "Alquiler", "Comida", "Transporte", "Entretenimiento" y "Compras". Esto proporciona una amplia gama de posibles movimientos financieros, imitando el comportamiento financiero cotidiano.

c. Exportación de Datos a CSV

Una vez generadas las transacciones, la API permite la exportación de los datos a un archivo **CSV**, que el usuario puede descargar directamente desde su navegador. Para lograr esto:

- Se utiliza la función `StreamingResponse` de **FastAPI**, que permite enviar flujos de datos de manera eficiente.
- Las transacciones se escriben en memoria utilizando **StringIO** y **csv.writer**, para posteriormente devolverlas en formato CSV.

El archivo resultante incluye las siguientes columnas:

- **ID Transacción**: Un identificador único para cada transacción.
- **Fecha**: La fecha en que ocurrió la transacción.
- **Tipo**: Si fue un ingreso o un gasto.
- **Categoría**: La categoría asociada a la transacción.
- **Monto**: El monto de dinero involucrado en la transacción.

3. Validación de Parámetros

La API incluye validación automática de los parámetros de entrada, gracias a **FastAPI**:

- **Fechas**: Se valida que las fechas de inicio y fin estén en el formato correcto (`YYYY-MM-DD`), y que la fecha de fin no sea anterior a la de inicio. Si se detecta un error en la entrada de fechas, la API devuelve un error HTTP 400 con un mensaje descriptivo.
- **Cantidad de transacciones**: Se recibe como parámetro y es validada para asegurar que es un valor entero positivo.

4. Rutas Principales

El único **endpoint** disponible es `/generar-transacciones`, el cual permite generar las transacciones y devolverlas en formato CSV. Los parámetros que acepta son:

- **`cantidad`**: La cantidad de transacciones a generar.
- **`fecha_inicio`**: La fecha de inicio del rango de transacciones.
- **`fecha_fin`**: La fecha de fin del rango de transacciones.

El endpoint realiza todas las validaciones necesarias antes de proceder a la generación de los datos y su posterior devolución como archivo descargable.

5. Aplicaciones Prácticas

Este sistema tiene varias aplicaciones prácticas, especialmente en entornos que requieren simulación de datos financieros:

- **Pruebas de sistemas bancarios**: El script permite generar rápidamente datos de prueba que pueden ser utilizados en el desarrollo y pruebas de plataformas financieras.
- **Entrenamiento de modelos de machine learning**: Los datos generados son ideales para ser utilizados en modelos que requieran detectar patrones de comportamiento financiero.
- **Simulaciones de comportamiento del cliente**: Las transacciones simuladas permiten realizar análisis de comportamiento del cliente, observando cómo los ingresos y gastos afectan su balance financiero a lo largo del tiempo.

Conclusión

Esta API desarrollada con **FastAPI** y respaldada por funciones de generación de datos aleatorios es una herramienta poderosa para la **simulación de transacciones financieras**.

Su capacidad para generar perfiles de usuarios ficticios, simular ingresos y gastos en categorías comunes, y exportar los datos a CSV, la hace una solución robusta y eficiente para pruebas, simulaciones y análisis en entornos financieros.

La facilidad de despliegue y el manejo de las solicitudes, junto con la validación automática de los parámetros, aseguran un flujo de trabajo ágil y altamente personalizable.

Base de Datos

MySQL - Base de Datos bd_reto

La base de datos **bd_reto** fue diseñada en **MySQL** para proporcionar soporte al equipo de desarrollo **FullStack**, utilizando **DBeaverCE** como herramienta de gestión de la base de datos.

A continuación, se detallan las tablas que componen la base de datos, junto con sus respectivas columnas y relaciones.

1. Tabla `tb_usu`

Descripción: Esta tabla almacena la información general de los usuarios.
Columnas:

- **id_usu** *(int, NOT NULL)*: Identificador único del usuario.
- **nombre** *(varchar(45))*: Nombre del usuario.
- **apellidos** *(varchar(45))*: Apellidos del usuario.
- **edad** *(int)*: Edad del usuario.
- **sexo** *(varchar(45))*: Sexo del usuario.
- **grupo** *(varchar(45))*: Grupo al que pertenece el usuario.
- **sueldo** *(int)*: Sueldo del usuario.
- **vivienda** *(varchar(45))*: Tipo de vivienda del usuario.
- **coche** *(tinyint(1))*: Indica si el usuario tiene coche (1) o no (0).
- **hijos** *(int)*: Número de hijos del usuario.

Llave primaria: `id_usu`.

2. Tabla `tb_auth`

Descripción: Almacena la información de autenticación de los usuarios.
Columnas:

- **dni** *(varchar(20), NOT NULL)*: Documento Nacional de Identidad del usuario.
- **id_usu** *(int)*: Identificador del usuario, relacionado con la tabla `tb_usu`.
- **contraseña_encriptada** *(varchar(255), NOT NULL)*: Contraseña encriptada del usuario.
- **token** *(varchar(255))*: Token de autenticación.
- **rol** *(enum('cliente', 'admin'), NOT NULL)*: Rol del usuario dentro del sistema (cliente o administrador).

Llave primaria: `dni`.
Llave foránea: `id_usu` (referencia a `tb_usu(id_usu)`, con eliminación en cascada *SET NULL*).

3. Tabla `tb_cta`

Descripción: Almacena la información de las cuentas bancarias de los usuarios.
Columnas:

- **id_usu** *(int, NOT NULL)*: Identificador del usuario, relacionado con la tabla `tb_usu`.
- **iban** *(varchar(45), NOT NULL)*: IBAN de la cuenta bancaria.
- **tipo_cta** *(varchar(45), NOT NULL)*: Tipo de cuenta (corriente o de ahorro).
- **saldo_cta** *(int, NOT NULL)*: Saldo actual de la cuenta.
- **estatus_cta** *(int, NOT NULL)*: Estado de la cuenta (activa, inactiva, bloqueada, etc.).

Llave primaria: Combinación de `id_usu` y `iban`.
Llave foránea: `id_usu` (referencia a `tb_usu(id_usu)`).

4. Tabla `tb_cta_hucha`

Descripción: Almacena la información de las cuentas de ahorro de los usuarios.
Columnas:

- **id_usu** *(int, NOT NULL)*: Identificador del usuario, relacionado con la tabla `tb_usu`.
- **iban** *(varchar(45), NOT NULL)*: IBAN de la cuenta de ahorro.
- **saldo_cta_hucha** *(int, NOT NULL)*: Saldo de la cuenta de ahorro.
- **recurrencia** *(varchar(45), NOT NULL)*: Frecuencia de los depósitos en la cuenta (diaria, mensual, etc.).
- **cuota_hucha** *(int, NOT NULL)*: Cuota de ahorro fijada.
- **razon_hucha** *(varchar(45), NOT NULL)*: Razón del ahorro (compra de vivienda, emergencias, etc.).
- **fecha_ini** *(date, NOT NULL)*: Fecha de inicio del plan de ahorro.
- **fecha_fin** *(date, NOT NULL)*: Fecha estimada de finalización del plan de ahorro.

Llave primaria: Combinación de `id_usu` y `iban`.
Llave foránea: `id_usu` (referencia a `tb_usu(id_usu)`).

5. Tabla `tb_trans_banc_ahor`

Descripción: Registra las transacciones bancarias relacionadas con cuentas de ahorro.
Columnas:

- **id_tran_banc** *(int, NOT NULL)*: Identificador único de la transacción bancaria.
- **id_usu** *(int, NOT NULL)*: Identificador del usuario, relacionado con la tabla `tb_usu`.
- **iban** *(varchar(45), NOT NULL)*: IBAN de la cuenta relacionada con la transacción.
- **importe** *(int)*: Importe de la transacción.
- **fecha** *(date)*: Fecha en que se realizó la transacción.

Llave primaria: `id_tran_banc`.
Llave foránea: `id_usu` (referencia a `tb_usu(id_usu)`).

6. Tabla `tb_trans_banc_cte`

Descripción: Almacena las transacciones bancarias de las cuentas corrientes de los usuarios.
Columnas:

- **id_tran_banc** *(int, NOT NULL)*: Identificador único de la transacción bancaria.
- **id_usu** *(int, NOT NULL)*: Identificador del usuario, relacionado con la tabla `tb_usu`.
- **iban** *(varchar(45), NOT NULL)*: IBAN de la cuenta relacionada con la transacción.
- **tipo** *(varchar(45))*: Tipo de transacción (ingreso, gasto, transferencia, etc.).
- **detalle** *(varchar(45))*: Descripción o detalle de la transacción.
- **importe** *(int)*: Importe de la transacción.
- **balance** *(int)*: Balance restante después de la transacción.
- **fecha** *(date)*: Fecha en que se realizó la transacción.

Llave primaria: `id_tran_banc`.
Llave foránea: `id_usu` (referencia a `tb_usu(id_usu)`).

Relaciones entre Tablas

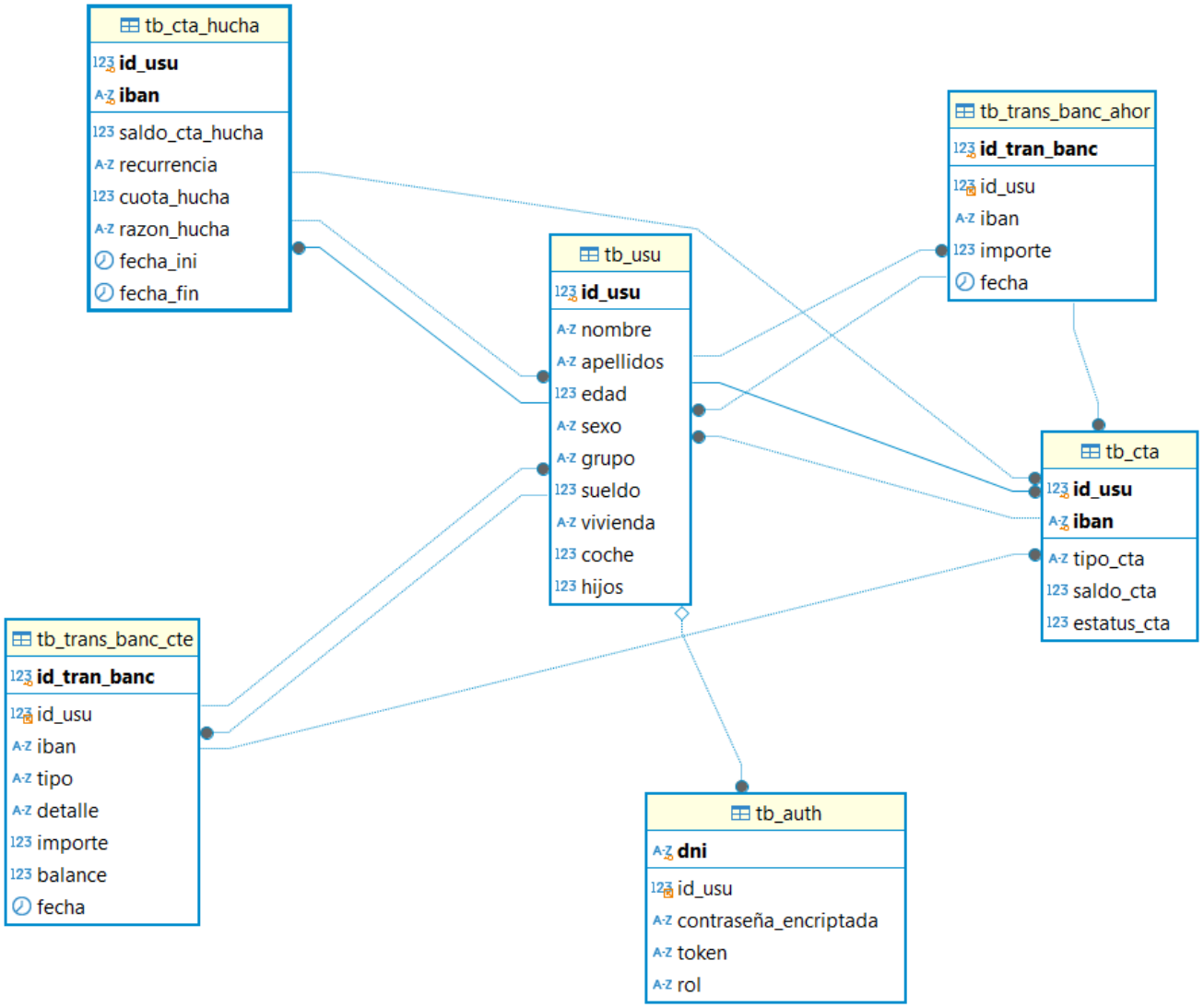
La base de datos establece varias relaciones entre las tablas, principalmente a través de la columna `id_usu` en la tabla `tb_usu`. Estas relaciones permiten la gestión y el seguimiento de las transacciones bancarias, la información de autenticación y las cuentas bancarias de los usuarios:

- **`tb_auth`** referencia a **`tb_usu`** a través de `id_usu`.
- **`tb_cta`** referencia a **`tb_usu`** a través de `id_usu`.
- **`tb_cta_hucha`** referencia a **`tb_usu`** a través de `id_usu`.
- **`tb_trans_banc_ahor`** referencia a **`tb_usu`** a través de `id_usu`.
- **`tb_trans_banc_cte`** referencia a **`tb_usu`** a través de `id_usu`.

Estas relaciones aseguran la integridad referencial de la base de datos, permitiendo mantener una estructura sólida para la información de los usuarios, sus cuentas y transacciones.

Diagrama de Relaciones

El diagrama de relaciones de la base de datos **bd_reto** muestra visualmente cómo están conectadas las tablas a través de las llaves foráneas. Esto refleja la interdependencia de las tablas, con la tabla `tb_usu` como el núcleo central de la base de datos, referenciada por la mayoría de las otras tablas.



Conclusión

Este diseño relacional proporciona una estructura sólida y eficiente para gestionar la información de usuarios y sus transacciones, permitiendo una adecuada escalabilidad y soporte para las aplicaciones del equipo **FullStack**.

Además, facilita el manejo de los datos bancarios de los usuarios, asegurando la integridad y consistencia en las transacciones de cuentas corrientes y de ahorro.

Modelos

MODELOS MACHINE LEARNING

Dentro de los modelos predictivos y de machine learning consideramos que los más idóneos para este proyecto bancario serían los siguientes:

1. **Clustering de usuarios**: Aplica k-medoids o k-means para identificar diferentes grupos basados en su perfil financiero.
2. **Sistema de recomendación**:
    - Si es filtrado colaborativo, usa Surprise o LightFM para recomendaciones basadas en la interacción histórica de usuarios.
3. **Clasificación basada en Score**: Usa algoritmos de clasificación como Random Forest o Logistic Regression para etiquetar a los usuarios con un score financiero o similar.
4. **Predicción de gastos e ingresos**. Power BI comparativa con mismo mes año anterior

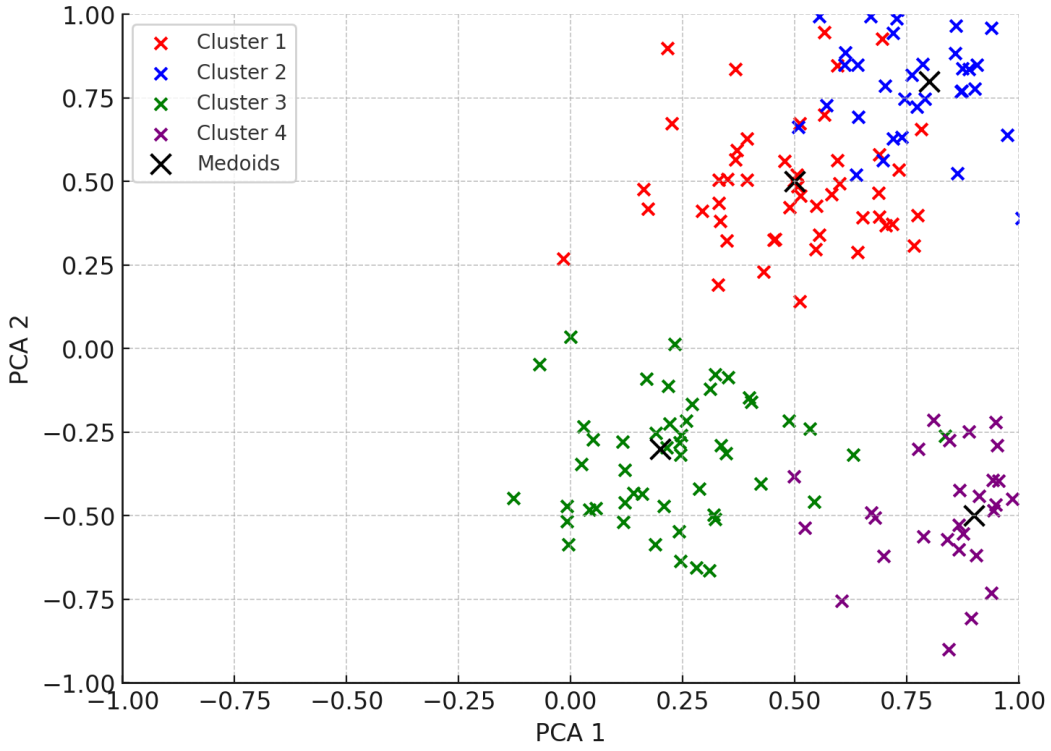
Clusterización de Usuarios

Una de las propuestas que estamos diseñando para Kutxabank es mediante un modelo no supervisado de Machine Learning con el que se pueda **identificar grupos o segmentos de usuarios que tengan ciertas características en común,** como, por ejemplo, score financiero, distribución de gastos, características personales como(estado civil, sueldo, edad, hijos, hábitos de gastos).

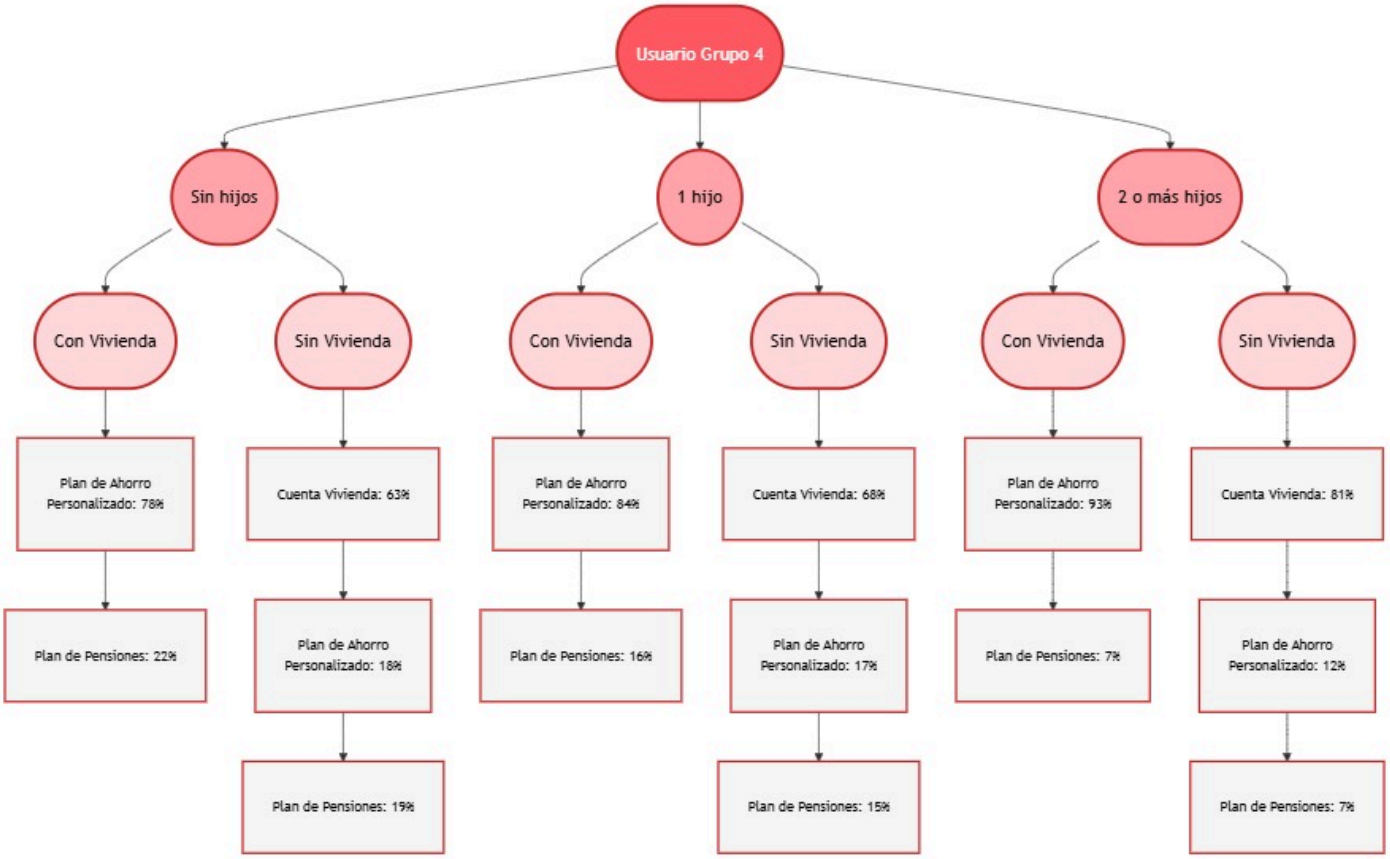
Entre los modelos que podrían considerarse, pensamos que el que mejor se adaptaría es:

- **K-MEDOIDS**, modelo de Machine Learning No Supervisado se podría identificar los grupos de clientes que tiene el banco.

  k-medoids es un algoritmo de **clustering** que busca agrupar un conjunto de datos en **k** grupos, donde cada grupo está representado por un punto que es el centroide (llamado medoid) de ese grupo.



Sistema de Recomendación



Para aplicar un modelo de machine learning al banco de forma práctica, el proceso sería el siguiente:

1. **Segmentación de Clientes:** El modelo clasifica automáticamente a los clientes en diferentes grupos según sus características, como número de hijos, tipo de vivienda, ingresos, etc. Esto permite identificar rápidamente sus necesidades financieras.
2. **Recomendación Personalizada de Productos**: Basado en la segmentación, el sistema recomendaría productos financieros (planes de ahorro, cuentas de vivienda, planes de pensiones) adaptados a cada perfil de cliente. Por ejemplo, clientes sin hijos y con vivienda podrían recibir recomendaciones de cuentas de ahorro o planes de pensiones, como se muestra en la imagen.
3. **Optimización Continua:** A medida que más clientes interactúan con el banco y se recopilan más datos sobre sus elecciones de productos, el sistema mejorará sus recomendaciones. Los datos históricos se utilizarán para ajustar las sugerencias, ofreciendo productos que otros clientes con perfiles similares ya hayan adoptado con éxito.

4. **Automatización para Gestores:** El modelo permite a los gestores del banco enfocarse en productos relevantes para cada cliente, reduciendo el tiempo dedicado a personalizar manualmente las recomendaciones. Con la información segmentada, los gestores podrán ver rápidamente qué productos ofrecer a cada cliente basado en su perfil.
5. **Aumento de la Satisfacción y Retención de Clientes:** Al recibir productos alineados con sus necesidades, los clientes estarán más satisfechos y es más probable que mantengan una relación a largo plazo con el banco, mejorando la retención y aumentando la venta cruzada de productos financieros.

Este sistema automatizado ofrecería recomendaciones eficientes y escalables, ayudando al banco a maximizar el valor de cada cliente

AURRE SCORE: Score de Salud Financiera

```
Score Total = 0.4 * (Relación Ingresos/Gastos)
            + 0.3 * (Ahorros Disponibles)
            + 0.2 * (Propiedad de Vivienda)
            + 0.1 * (Número de Hijos)
```

Detalle de las Ponderaciones:

1. **Relación Ingresos/Gastos** (40%)
    - Mide si estás gastando menos de lo que ganas de manera consistente.
    - Fórmula: `(ingresos_totales - gastos_totales) / ingresos_totales`
2. **Ahorros Disponibles** (30%)
    - Mide cuánto has ahorrado en comparación con un fondo de emergencia recomendado de 6 meses de sueldo.
    - Fórmula: `ahorros_actuales / (sueldo_mensual * 6)`
3. **Propiedad de Vivienda** (20%)
    - Evalúa si tienes una vivienda propia, lo que mejora tu estabilidad financiera.
    - Valor: `1 si tiene vivienda` o `0 si no tiene`
4. **Número de Hijos** (10%)
    - Ajusta el score en función de la carga financiera adicional que representan los hijos.
    - Fórmula: `max(1 - (gastos_hijos / sueldo_mensual), 0)`

Este esquema asegura que los **ingresos/gastos** y los **ahorros** tienen el mayor peso, ya que son clave para una buena salud financiera, mientras que la **vivienda** y el **número de hijos** influyen en menor medida pero son importantes para la estabilidad a largo plazo.

Ejemplo

Ander e Irati (GRUPO 4), 1 hijo
Ingresos Totales: 2,881.39 EUR (14 Pagas)
Gastos Totales: 2,373.79 EUR
Vivienda No
Ahorro inicial 9327,93 EUR (1 de Noviembre 2021)
Ahorro final: 34735.18 EUR (25 de Octubre 2024)

El **AURRE SCORE** para Ander e Irati en los dos momentos evaluados es:

1. **1 de Noviembre de 2021**: El score es **0.32**, lo que indica una situación financiera ajustada en ese momento, con ahorros limitados y gastos cercanos a los ingresos.
2. **25 de Octubre de 2024**): El score es **0.72**, lo que refleja una notable mejora en su salud financiera, impulsada por un incremento considerable en sus ahorros.

Este progreso muestra una buena gestión de los ingresos y los ahorros a lo largo del tiempo, acercándose a una situación financiera sólida.

Predicción de gastos e ingresos

Modelos de Series Temporales

Hemos valorado varios modelos para **predecir ingresos y gastos** en el futuro. Entre ellos, estaban:

- **ARIMA (AutoRegressive Integrated Moving Average)**

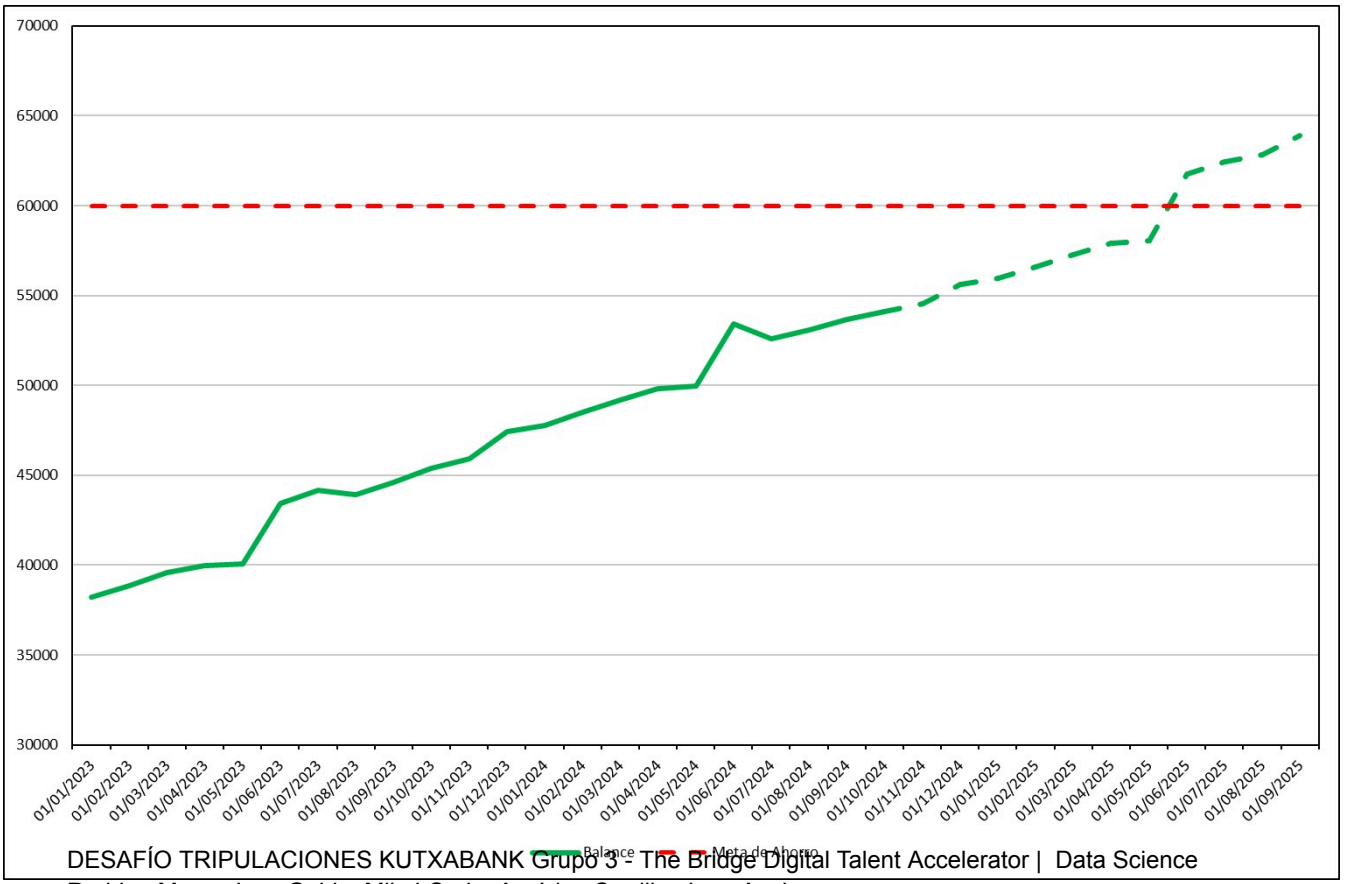
  ARIMA es uno de los modelos más usados para series temporales y puede ser útil si tienes una tendencia lineal o estacional en tus datos.

- **PROPHET (de Facebook)**

  Prophet es especialmente útil si hay estacionalidad en tus ingresos o gastos, como ingresos más altos en ciertos meses o gastos picos en vacaciones.

Evaluación del Modelo

Una vez tengas las predicciones, compara los valores predichos con los datos reales de ingresos y gastos para evaluar la precisión del modelo utilizando métricas como MAE (Mean Absolute Error) o RMSE (Root Mean Square Error).



Rodrigo Meza, Josu Goiria, Mikel Cerio, Américo Carrillo, Juan Azpiazu
25/10/2024

Proyectos Futuros

PROYECTOS FUTUROS

Para ver la efectividad de esta propuesta del proyecto sería ideal que se trabajara con datos reales para ver la robustez de estas.

1. Predicción de capacidad de ahorro

La aplicación podría predecir cuánto puede ahorrar el usuario cada mes, sugiriendo ajustes en el presupuesto mensual de forma realista para cumplir con su objetivo de ahorro. Utilizando datos históricos de ingresos, gastos y ahorro, el sistema puede:

• Estimar cuánto podría ahorrar el usuario en un tiempo determinado, basado en patrones previos de ingreso y gastos.

• Proporcionar alertas automáticas si se detecta que el usuario podría tener dificultades para cumplir con su objetivo de ahorro.

-Técnicas:

• Series temporales para predecir los ingresos y gastos futuros en función de patrones históricos.

• Modelos de regresión lineal o no lineal para prever la evolución del ahorro basado en cambios en el comportamiento financiero.

2. Sugerencias personalizadas para metas de ahorro

A partir del objetivo del usuario (como un viaje en crucero), el sistema puede sugerir un plan de ahorro personalizado, ajustado a las características financieras del usuario. El algoritmo puede ajustar recomendaciones de acuerdo con:

• El plazo para lograr el objetivo (por ejemplo, "si deseas irte de crucero en 12 meses, deberías ahorrar X por mes").

• Cambios en las fuentes de ingresos o variaciones en los gastos.

-Técnicas:

• Sistemas de recomendación basados en los datos históricos del usuario y el comportamiento de otros usuarios con objetivos similares.

• Optimización matemática para determinar el mejor plan de ahorro en función de las restricciones financieras del usuario.

3. Análisis de riesgo y seguridad financiera

El uso de ML podría ayudar al usuario a gestionar mejor sus riesgos financieros. Los modelos de evaluación de riesgo financiero pueden analizar el comportamiento financiero del usuario y su capacidad de afrontar imprevistos, como emergencias o fluctuaciones en ingresos.

Técnicas:

• Modelos de clasificación (árboles de decisión, SVM) para predecir la probabilidad de que el usuario no pueda cumplir sus metas financieras debido a factores como la pérdida de ingresos o gasto excesivo.

4. Asistente virtual inteligente

Una de las áreas donde ML puede destacar es en la creación de un asistente virtual inteligente que ayude al usuario a gestionar sus finanzas. Este asistente puede:

• Resolver preguntas sobre la planificación del ahorro.

• Hacer recomendaciones sobre inversiones o nuevas estrategias de ahorro.

• Ayudar a los usuarios a identificar áreas donde pueden recortar gastos, basándose en los hábitos financieros.

Técnicas:

• Procesamiento del lenguaje natural (NLP) para que el asistente pueda interpretar y responder preguntas del usuario de manera natural y fluida.

• Aprendizaje por refuerzo para mejorar la interacción con el usuario y ajustar las recomendaciones según sus preferencias.

5. Monitorización automática del progreso

La aplicación puede utilizar ML para monitorizar continuamente el progreso del usuario hacia su objetivo de ahorro, detectando desviaciones e informando automáticamente. Si el sistema detecta que el usuario está gastando más de lo esperado, puede ajustar el plan de ahorro o enviar recomendaciones para corregir el rumbo.

Técnicas:

• Anomalía de detección para identificar gastos atípicos que pueden desbalancear el plan de ahorro.

• Redes neuronales recurrentes (RNNs) o modelos de series temporales avanzados para hacer un seguimiento continuo del progreso del ahorro y ajustar las predicciones a medida que cambian las condiciones.

6. Modelos de optimización para asignación presupuestaria

Machine Learning puede ayudar a crear un sistema de optimización del presupuesto mensual que ajuste automáticamente cuánto dinero destinar a diferentes categorías (vivienda, comida, ocio) según las prioridades y objetivos de ahorro del usuario.

Técnicas:

- Optimización de presupuestos usando técnicas de programación lineal o algoritmos genéticos para encontrar la asignación óptima de ingresos a los distintos gastos.