Analisis de Clientes Banco ACME S.A

Customer Behavior de Tarjetas de Credito

Objetivos del analisis

Este Notebook tiene los siguientes objetivos:

- Explorar de base de datos de clientes, utilizando varios tipos de Visualización.
- Realizar el preprocesado de datos, preparandolos para el analisis y modelado.
- Agrupar clientes utilizando modelos de clusterización.
- Interpretar analisis sobre los grupos de clientes, perfilado de los grupos.
- Elevar propuesta de marketing en base a los perfiles y análisis realizados.

Librerias Utilzadas

In [269]:

```
1 !conda activate credit_cards
```

In [270]:

```
# --- Importing Libraries ---
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_profiling
#from ydata_profiling import ProfileReport

import scipy.stats as stats

#import ydata_profiling

import seaborn as sns
import warnings
import os
import yellowbrick
import scipy.cluster.hierarchy as shc
import matplotlib.patches as patches

from faker import Faker
import random

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
from IPython import display


from matplotlib.patches import Rectangle
from pandas_profiling import ProfileReport
from pywaffle import Waffle
from math import isnan
from random import sample
from numpy.random import uniform
from sklearn.neighbors import NearestNeighbors
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import davies_bouldin_score, silhouette_score, calinski_harabasz
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from yellowbrick.style import set_palette
from yellowbrick.contrib.wrapper import wrap

# --- Libraries Settings ---
warnings.filterwarnings('ignore')
sns.set_style('white')
plt.rcParams['figure.dpi'] = 600
#sns.set(rc = {'axes.facecolor': '#FBFBFB', 'figure.facecolor': '#FBFBFB'})
class clr:
    start = '\033[93m'+'\033[1m'
    color = '\033[93m'
    end = '\033[0m'
```

Creación del Dataset a utilizar

In [271]:

```
df = pd.read_csv('Customer_Data.csv')

# --- Reading Train Dataset ---
print(clr.start+'.: Dataset Importado :.'+clr.end)
print(clr.color+'*' * 23)
data.head().style.background_gradient(cmap='YlOrBr')
```

```
.: Dataset Importado :.
***********************
```

Out[271]:

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purc
0	C10001	40.900749	0.818182	95.400000	0.000000	95.4
1	C10002	3202.467416	0.909091	0.000000	0.000000	0.0
2	C10003	2495.148862	1.000000	773.170000	773.170000	0.0
3	C10004	1666.670542	0.636364	1499.000000	1499.000000	0.0
4	C10005	817.714335	1.000000	16.000000	16.000000	0.0

Descripcion de los datos

- Balance: Es el monto que el cliente posee disponible para seguir gastando
- Balance Frequency: Valor entre 0 y 1, determinar que tan frecuente es el movimiento de la tarjeta
- Purchases: Monto total invertido en compras
- oneoff_purchases: El monto total invertido en compras al contado
- Installments_purchases: Es el monto invertido en compras a cuotas
- Cash_advance: Monto que el clientes saca en cajeros / Adelantos de efectivo
- Purchases_frecuency: Frecuencia de compras
- Oneoff_purchases_rfecuency: frecuencia con la que el cliente realiza compras al contado
- Purchases_installments_frecuency: frecuencia con la que el cliente realiza compras a cuotas
- Cash_advance_frecuency: Frecuencia con la que el cliente realiza adelantos de efectivo
- Cash_advance_trx: Cantidad de transacciones de adelanto de efectivo
- purchases_trx: Cantidad total de transacciones
- credit_limit: Limite de credito del cliente
- payments: Montos ya pagados sobre la tarjeta
- minimum_payments:
- prc_full_payment:
- tenure:

Se adicionaran los siguientes datos:

- Antiguedad del cliente (En meses)
- Antiguedad del cliente con el productos (En Meses)
- Ciudad
- Edad
- Genero

- Rubro de mayor consumo (Categoria)

Seria importante agregar:

- Es digital? (Podria ser "Frecuencia de transacciones digitales")
- Es cliente final de "Pago de salario"
- Cantidad de productos del cliente
- Rubro de mayor consumo (Categoria)
- TMHI: Tiempo minimo hasta la inactiviad
- Promedio de ingresos del cliente

In [272]:

```
df.head()
```

Out[272]:

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purcha
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

Revision Pre-exploracion del dataset

In [273]:

```
# --- Dataset Report ---
ProfileReport(df, title='Credit Card Dataset Report', minimal=True, progress_bar=Fals
```

In [274]:

```
# eliminar las columnas 'cust_id',"ciudad","genero","Top_Rubro" por que no son numeri
df = df.drop(columns=['cust_id'])
```

In [275]:

```
df.head()
```

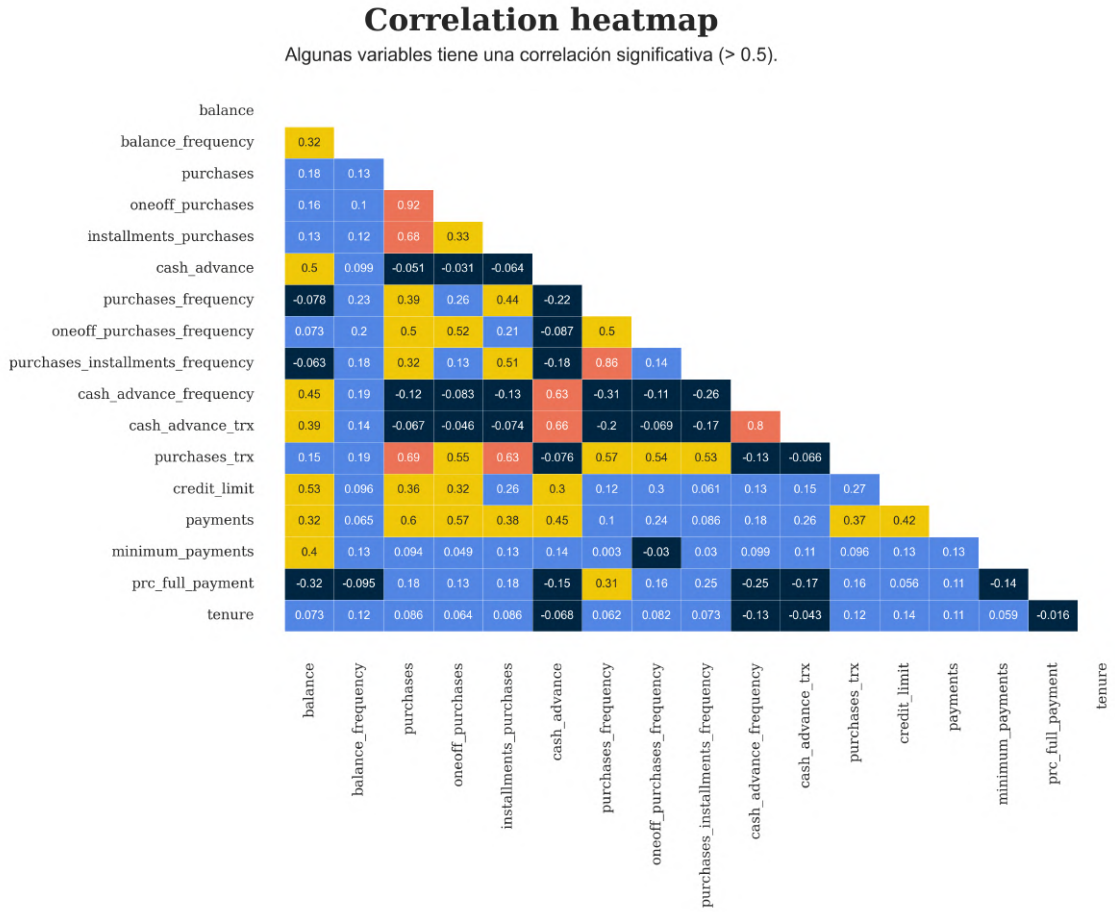
Out[275]:

	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases	cas
0	40.900749	0.818182	95.40	0.00	95.4	
1	3202.467416	0.909091	0.00	0.00	0.0	6
2	2495.148862	1.000000	773.17	773.17	0.0	
3	1666.670542	0.636364	1499.00	1499.00	0.0	
4	817.714335	1.000000	16.00	16.00	0.0	

Correlación entre variables

In [276]:

```
%matplotlib inline
# --- Correlation Map (Heatmap) ---
mask = np.triu(np.ones_like(df.corr(), dtype=bool))
fig, ax = plt.subplots(figsize=(7, 6))
sns.heatmap(df.corr(), mask=mask, annot=True, cmap= ['#002642', '#5386E4', '#F0C808',
yticks, ylabels = plt.yticks()
xticks, xlabels = plt.xticks()
ax.set_xticklabels(xlabels, size=6, fontfamily='serif')
ax.set_yticklabels(ylabels, size=6, fontfamily='serif')
plt.suptitle('Correlation heatmap', fontweight='heavy', x=0.327, y=0.96, ha='left', f
plt.title('Algunas variables tiene una correlación significativa (> 0.5).\n', fontsiz
plt.tight_layout(rect=[0, 0.04, 1, 1.01])
plt.show(block=True)
```



EDA

In [277]:

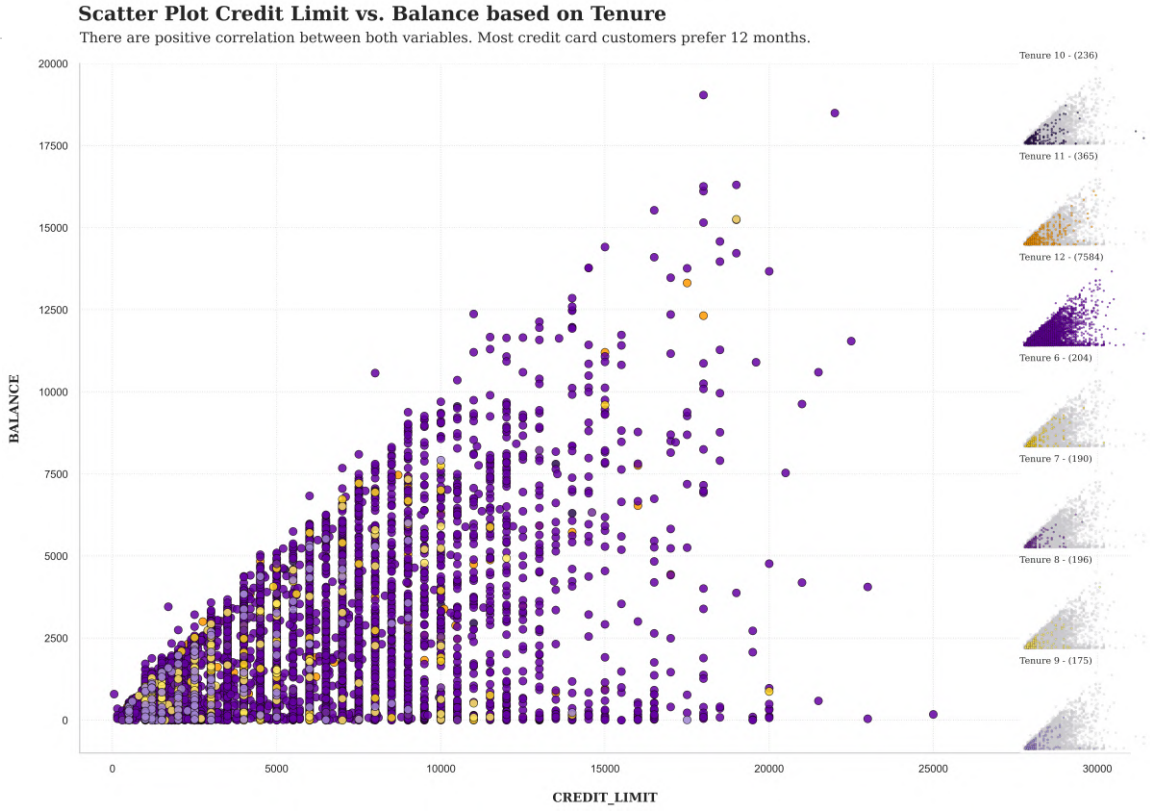
```
df.columns = df.columns.str.upper()
```

EDA 1: Balance del cliente vs Limite de Credito

In [278]:

```
# --- EDA 1 Variables ---
scatter_style=dict(linewidth=0.65, edgecolor='#100C07', alpha=0.85)
sub_scatter_style_color=dict(s=5, alpha=0.65, linewidth=0.15, zorder=10, edgecolor='#
sub_scatter_style_grey=dict(s=5, alpha=0.3, linewidth=0.7, zorder=5, color='#CAC9CD')
grid_style=dict(alpha=0.3, color='#9B9A9C', linestyle='dotted', zorder=1)
xy_label=dict(fontweight='bold', fontsize=14, fontfamily='serif')
suptitle=dict(fontsize=22, fontweight='heavy', fontfamily='serif')
title=dict(fontsize=16, fontfamily='serif')
color_pallete=['#2D0F51', '#FF9A00', '#6600A5', '#FFD61E', '#722E9A', '#FFE863', '#A4
sub_axes=[None] * 7

# --- EDA 1 Data Frame ---
eda1 = df[['CREDIT_LIMIT', 'BALANCE', 'TENURE']]
eda1['TENURE'] = eda1['TENURE'].astype(str)
tenure = sorted(eda1['TENURE'].unique())

# --- EDA 1 Settings ---
fig = plt.figure(figsize=(22, 14))
gs = fig.add_gridspec(7, 7)
ax = fig.add_subplot(gs[:, :7])
ax.set_aspect(1)

# --- EDA 1: Main Scatter Plot ---
for x in range(len(tenure)):
    eda1_x = eda1[eda1['TENURE']==tenure[x]]
    ax.scatter(eda1_x['CREDIT_LIMIT'], eda1_x['BALANCE'], s=80, color=color_pallete[x
    ax.set_title('There are positive correlation between both variables. Most credit
    ax.set_xlabel('\nCREDIT_LIMIT', **xy_label)
    ax.set_ylabel('BALANCE\n', **xy_label)
    ax.grid(axis='y', which='major', **grid_style)
    ax.grid(axis='x', which='major', **grid_style)
    for spine in ax.spines.values():
        spine.set_color('None')
    for spine in ['bottom', 'left']:
        ax.spines[spine].set_visible(True)
        ax.spines[spine].set_color('#CAC9CD')
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

# --- EDA 1: Sub Plots ---
for idx, tnr in enumerate(tenure):
    sub_axes[idx] = fig.add_subplot(gs[idx, 6], aspect=1)

    sub_axes[idx].scatter(eda1[eda1['TENURE']!=tnr]['CREDIT_LIMIT'], eda1[eda1['TENUR
    sub_axes[idx].scatter(eda1[eda1['TENURE']==tnr]['CREDIT_LIMIT'], eda1[eda1['TENUR

    cnt = (eda1['TENURE']==tnr).sum()
    sub_axes[idx].set_title(f'Tenure {tnr} - ({cnt})', loc='left', fontsize=10, fontf
    sub_axes[idx].set_xticks([])
    sub_axes[idx].set_yticks([])
    for spine in sub_axes[idx].spines.values():
        spine.set_color('None')

# --- EDA 1 XY Limit ---
for axes in [ax] + sub_axes:
    axes.set_xlim(-1000, 31000)
    axes.set_ylim(-1000, 20000)

# --- EDA 1 Title ---
```

```
60 plt.suptitle('Scatter Plot Credit Limit vs. Balance based on Tenure', x=0.138, y=0.94
61
62 plt.show();
```



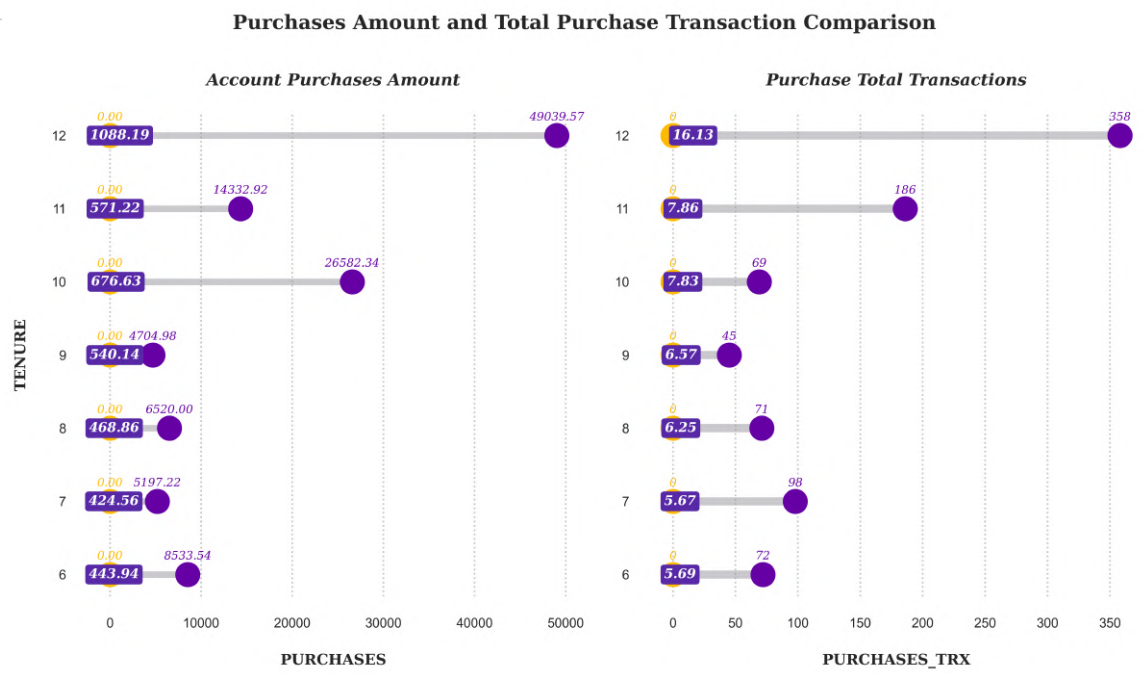
EDA 2: Promedio, Minimo y Maximo de "Monto de Compra" y "Cantidad de Transacciones"

Agrupado por Tenure del cliente

In [279]:

```
# --- EDA 2 Variables ---
title=dict(fontsize=10, fontfamily='serif', style='italic', weight='bold', ha='center
grid_style = dict(alpha=0.6, color='#9B9A9C', linestyle='dotted', zorder=1)
sct_style = dict(s=175, linewidth=2)
xy_label = dict(fontweight='bold', fontsize=9, fontfamily='serif')
ann_style = dict(xytext=(0, 0), textcoords='offset points', va='center', ha='center',
tenure = sorted(df['TENURE'].unique())
color_pallete = ['#2D0F51', '#FF9A00', '#6600A5', '#FFD61E', '#722E9A', '#FFE863', '#

# --- EDA 2.1 Data Frame ---
eda2_1 = df[['PURCHASES', 'TENURE']]
eda2_1 = eda2_1.groupby('TENURE').agg(MIN=('PURCHASES', 'min'), AVG=('PURCHASES', 'me

# --- EDA 2.2 Data Frame ---
eda2_2 = df[['PURCHASES_TRX', 'TENURE']]
eda2_2 = eda2_2.groupby('TENURE').agg(MIN=('PURCHASES_TRX', 'min'), AVG=('PURCHASES_T

# --- EDA 2.1 & 2.2 Settings ---
fig = plt.figure(figsize=(10, 6))
plt.suptitle('\nPurchases Amount and Total Purchase Transaction Comparison', fontweig

# --- EDA 2.1 (Left Dumbbell) ---
plt.subplot(1, 2, 1)
plt.tight_layout(rect=[0, 0, 1, 1.01])
axs_left=plt.gca()
min_sct = plt.scatter(x=eda2_1['MIN'], y=eda2_1['TENURE'], c='#FFBB00', **sct_style)
max_sct = plt.scatter(x=eda2_1['MAX'], y=eda2_1['TENURE'], c='#6600A5', **sct_style)
for i in range(len(tenure)):
    eda2_1_x = eda2_1[eda2_1['TENURE']==tenure[i]]
    plt.hlines(y=eda2_1_x['TENURE'], xmin=eda2_1_x['MIN'], xmax=eda2_1_x['MAX'], line
    plt.annotate('{0:.2f}'.format(eda2_1_x['MIN'].values[0]), xy=(eda2_1_x['MIN'].val
    plt.annotate('{0:.2f}'.format(eda2_1_x['AVG'].values[0]), xy=(eda2_1_x['AVG'].val
    plt.annotate('{0:.2f}'.format(eda2_1_x['MAX'].values[0]), xy=(eda2_1_x['MAX'].val
for spine in axs_left.spines.values():
    spine.set_color('None')
plt.xlabel('\nPURCHASES', **xy_label)
plt.ylabel('TENURE\n', **xy_label)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.grid(axis='y', alpha=0)
plt.grid(axis='x', which='major', **grid_style)
plt.title('\nAccount Purchases Amount\n', **title)

# --- EDA 2.2 (Right Dumbbell) ---
plt.subplot(1, 2, 2)
plt.tight_layout(rect=[0, 0, 1, 1.01])
axs_right=plt.gca()
min_sctt = plt.scatter(x=eda2_2['MIN'], y=eda2_2['TENURE'], c='#FFBB00', **sct_style)
max_sctt = plt.scatter(x=eda2_2['MAX'], y=eda2_2['TENURE'], c='#6600A5', **sct_style)
for i in range(len(tenure)):
    eda2_2_x = eda2_2[eda2_2['TENURE']==tenure[i]]
    plt.hlines(y=eda2_2_x['TENURE'], xmin=eda2_2_x['MIN'], xmax=eda2_2_x['MAX'], line
    plt.annotate('{:.0f}'.format(eda2_2_x['MIN'].values[0]), xy=(eda2_2_x['MIN'].valu
    plt.annotate('{0:.2f}'.format(eda2_2_x['AVG'].values[0]), xy=(eda2_2_x['AVG'].val
    plt.annotate('{:.0f}'.format(eda2_2_x['MAX'].values[0]), xy=(eda2_2_x['MAX'].valu
for spine in axs_right.spines.values():
    spine.set_color('None')
plt.xlabel('\nPURCHASES_TRX', **xy_label)
plt.ylabel('')
```

```
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.grid(axis='y', alpha=0)
plt.grid(axis='x', which='major', **grid_style)
plt.title('\nPurchase Total Transactions\n', **title)

plt.show();
```

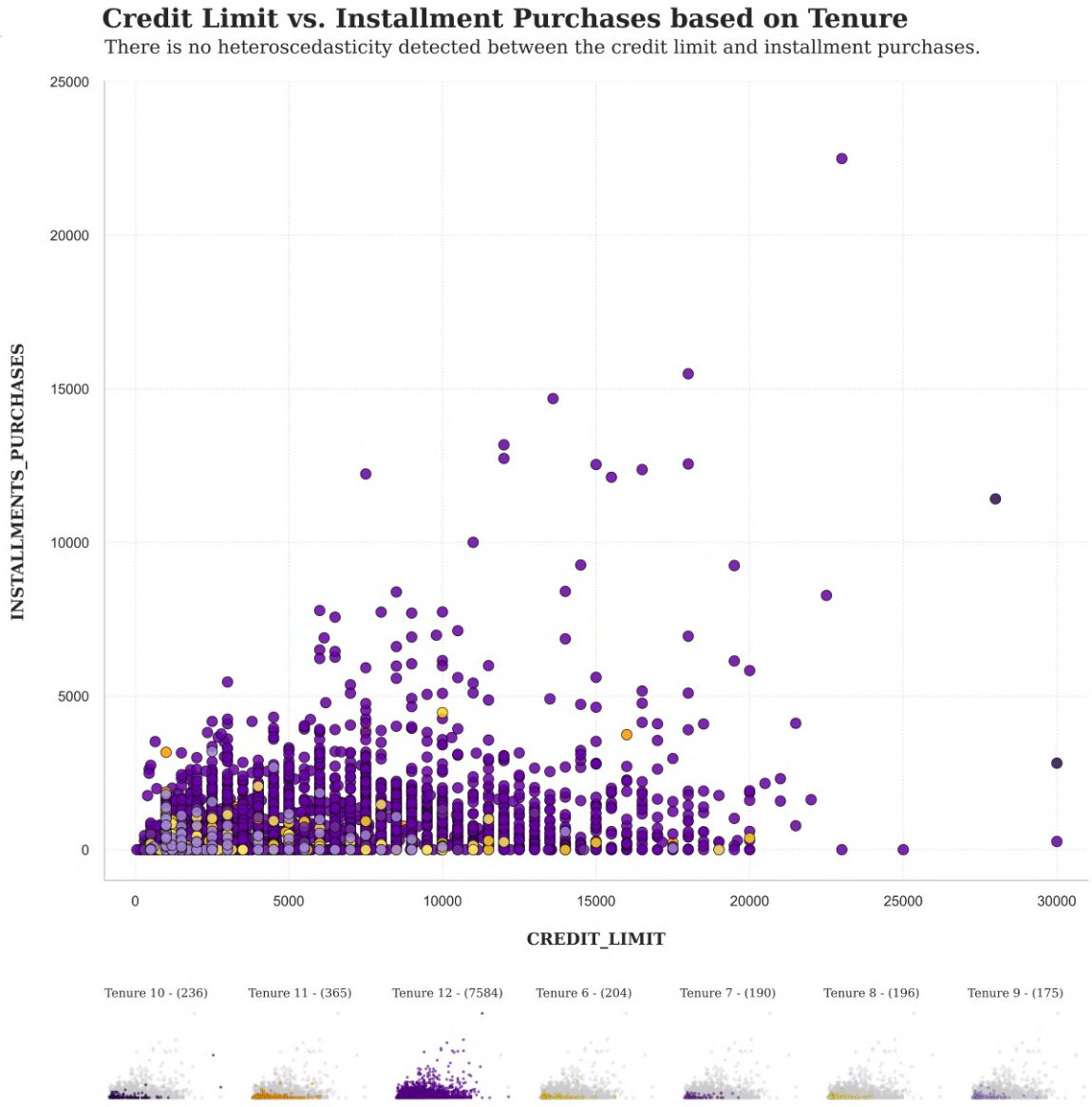


EDA 3: Monto invertido en Compras a Cuotas vs Limite de credito del cliente

In [280]:

```
# --- EDA 3 Variables ---
scatter_style=dict(linewidth=0.65, edgecolor='#100C07', alpha=0.85)
sub_scatter_style_color=dict(s=5, alpha=0.65, linewidth=0.15, zorder=10, edgecolor='#
sub_scatter_style_grey=dict(s=5, alpha=0.3, linewidth=0.7, zorder=5, color='#CAC9CD')
grid_style=dict(alpha=0.3, color='#9B9A9C', linestyle='dotted', zorder=1)
xy_label=dict(fontweight='bold', fontsize=14, fontfamily='serif')
suptitle=dict(fontsize=22, fontweight='heavy', fontfamily='serif')
title=dict(fontsize=16, fontfamily='serif')
color_pallete=['#2D0F51', '#FF9A00', '#6600A5', '#FFD61E', '#722E9A', '#FFE863', '#A4
sub_axes=[None] * 7

# --- EDA 3 Data Frame ---
eda3 = df[['CREDIT_LIMIT', 'INSTALLMENTS_PURCHASES', 'TENURE']]
eda3['TENURE'] = eda1['TENURE'].astype(str)
tenure = sorted(eda1['TENURE'].unique())

# --- EDA 3 Settings ---
fig = plt.figure(figsize=(15, 20))
gs = fig.add_gridspec(7, 7)
ax = fig.add_subplot(gs[:7, :])
ax.set_aspect(1)

# --- EDA 3: Main Scatter Plot ---
for x in range(len(tenure)):
    eda3_x = eda3[eda3['TENURE']==tenure[x]]
    ax.scatter(eda3_x['CREDIT_LIMIT'], eda3_x['INSTALLMENTS_PURCHASES'], s=80, color=
    ax.set_title('There is no heteroscedasticity detected between the credit limit an
    ax.set_xlabel('\nCREDIT_LIMIT', **xy_label)
    ax.set_ylabel('INSTALLMENTS_PURCHASES\n', **xy_label)
    ax.grid(axis='y', which='major', **grid_style)
    ax.grid(axis='x', which='major', **grid_style)
    for spine in ax.spines.values():
        spine.set_color('None')
    for spine in ['bottom', 'left']:
        ax.spines[spine].set_visible(True)
        ax.spines[spine].set_color('#CAC9CD')
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

# --- EDA 3: Sub Plots ---
for idx, tnr in enumerate(tenure):
    sub_axes[idx] = fig.add_subplot(gs[6, idx], aspect=1)

    sub_axes[idx].scatter(eda3[eda3['TENURE']!=tnr]['CREDIT_LIMIT'], eda3[eda3['TENUR
    sub_axes[idx].scatter(eda3[eda3['TENURE']==tnr]['CREDIT_LIMIT'], eda3[eda3['TENUR

    cnt = (eda3['TENURE']==tnr).sum()
    sub_axes[idx].set_title(f'Tenure {tnr} - ({cnt})', loc='left', fontsize=10, fontf
    sub_axes[idx].set_xticks([])
    sub_axes[idx].set_yticks([])
    for spine in sub_axes[idx].spines.values():
        spine.set_color('None')

# --- EDA 3 XY Limit ---
for axes in [ax] + sub_axes:
    axes.set_xlim(-1000, 31000)
    axes.set_ylim(-1000, 25000)

# --- EDA 3 Title ---
```

```
60 plt.suptitle('Credit Limit vs. Installment Purchases based on Tenure', x=0.123, y=0.7
61
62 plt.show();
```



Preprocesado de los datos

Solo debemos tener variables numericas hasta este punto

In [281]:

```
# --- Showing Dataframe ---
df = df #xd
df.head().style.background_gradient(cmap='afmhot_r')
```

Out[281]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS
0	40.900749	0.818182	95.400000	0.000000	
1	3202.467416	0.909091	0.000000	0.000000	
2	2495.148862	1.000000	773.170000	773.170000	
3	1666.670542	0.636364	1499.000000	1499.000000	
4	817.714335	1.000000	16.000000	16.000000	

Imputation

Como el conjunto de datos es para clustering, se utilizará KNNImputer() para evitar resultados de clustering sesgados. El valor medio de los n_neighbors más cercanos encontrados en el conjunto de datos se utiliza para imputar los valores faltantes para cada muestra.

Las columnas 'credit_limit' y 'minimum_payments' contienen valores nulos que deben ser cargados con valores antes de continuar.

In [282]:

```
# --- List Null Columns ---
null_columns = df.columns[df.isnull().any()].tolist()

# --- Perform Imputation ---
imputer = KNNImputer()
df_imp = pd.DataFrame(imputer.fit_transform(df[null_columns]), columns=null_columns)
df = df.fillna(df_imp)

# --- Showing Dataframe ---
print(clr.start+'.: Dataframe after Imputation :.'+clr.end)
print(clr.color+'*' * 33)
df.head().style.background_gradient(cmap='afmhot_r')
```

```
.: Dataframe after Imputation :.
*********************************
```

Out[282]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS
0	40.900749	0.818182	95.400000	0.000000	
1	3202.467416	0.909091	0.000000	0.000000	
2	2495.148862	1.000000	773.170000	773.170000	
3	1666.670542	0.636364	1499.000000	1499.000000	
4	817.714335	1.000000	16.000000	16.000000	

Scaling 📐

El siguiente paso es escalar el conjunto de datos. La escalación es esencial ya que maneja la variabilidad del conjunto de datos, transforma los datos en un rango definido utilizando una transformación lineal para producir clusters de alta calidad y mejora la precisión de los algoritmos de clustering. En este caso, se utilizará un escalador estándar para estandarizar las características eliminando la media y escalando a la varianza unitaria.

In [283]:

```
# --- Scaling Dataset w/ Standard Scaler ---
X = pd.DataFrame(StandardScaler().fit_transform(df))
```

Hopkins Test 🧪

El siguiente paso es realizar una prueba estadística utilizando la prueba estadística de Hopkins para el conjunto de datos preprocesado para medir la tendencia de clustering de los datos (medida de en qué grado existen clusters en los datos a ser agrupados).

A continuación se muestra la hipótesis de la prueba estadística de Hopkins: H0: El conjunto de datos no está distribuido uniformemente (contiene clusters significativos). H1: El conjunto de datos está distribuido uniformemente (no hay clusters significativos). Criterios: Si el valor está entre {0.7, ..., 0.99}, se acepta H0 (tiene una alta tendencia a agruparse).

In [284]:

```
# --- Hopkins Test (codes by Matevž Kunaver) ---
def hopkins(X):
    d = X.shape[1]
    n = len(X)
    m = int(0.1 * n)
    nbrs = NearestNeighbors(n_neighbors=1).fit(X)

    rand_X = sample(range(0, n, 1), m)

    ujd = []
    wjd = []
    for j in range(0, m):
        u_dist, _ = nbrs.kneighbors(uniform(np.amin(X,axis=0),np.amax(X,axis=0),d).re
        ujd.append(u_dist[0][1])
        w_dist, _ = nbrs.kneighbors(X.iloc[rand_X[j]].values.reshape(1, -1), 2, retur
        wjd.append(w_dist[0][1])

    H = sum(ujd) / (sum(ujd) + sum(wjd))
    if isnan(H):
        print (ujd, wjd)
        H = 0

    return H

# --- Perform Hopkins Test ---
hopkins_value = hopkins(X)
hopkins_result = 'Result: '+clr.start+'{:.4f}'.format(hopkins_value)+clr.end
print(clr.start+'.: Hopkins Test :.'+clr.end)
print(clr.color+'*' * 19+clr.end)
print(hopkins_result)
if  0.7 < hopkins_value < 0.99:
    print('>> From the result above,'+clr.color+' it has a high tendency to cluster (
    print('\n'+clr.color+'*' * 31+clr.end)
    print(clr.start+'.:. Conclusions: Accept H0 .:.'+clr.end)
    print(clr.color+'*' * 31+clr.end)
else:
    print('>> From the result above,'+clr.color+' it has no meaningful clusters'+clr.
    print('\n'+clr.color+'*' * 31+clr.end)
    print(clr.start+'.:. Conclusions: Reject H0 .:.'+clr.end)
    print(clr.color+'*' * 31+clr.end)
```

```
.: Hopkins Test :.
*******************
Result: 0.9666
>> From the result above, it has a high tendency to cluster (contains mean
ingful clusters)

*******************************
.:. Conclusions: Accept H0 .:.
*******************************
```

PCA 🔧

Debido a que el conjunto de datos tiene 18 columnas, es fundamental aplicar algún método de reducción de dimensionalidad. Sin duda, uno de los métodos más utilizados es el PCA. El análisis de componentes principales (PCA) es un método utilizado en el aprendizaje automático no supervisado (como el

agrupamiento) que reduce los datos de alta dimensión a dimensiones más pequeñas mientras se preserva tanta información como sea posible.

- En este notebook, el número de características se reducirá a 2 dimensiones para que los resultados del clustering puedan ser visualizados.

In [285]:

```
# --- Transform into Array ---
X = np.asarray(X)

# --- Applying PCA ---
pca = PCA(n_components=2, random_state=24)
X = pca.fit_transform(X)
```

Clusterización de clientes

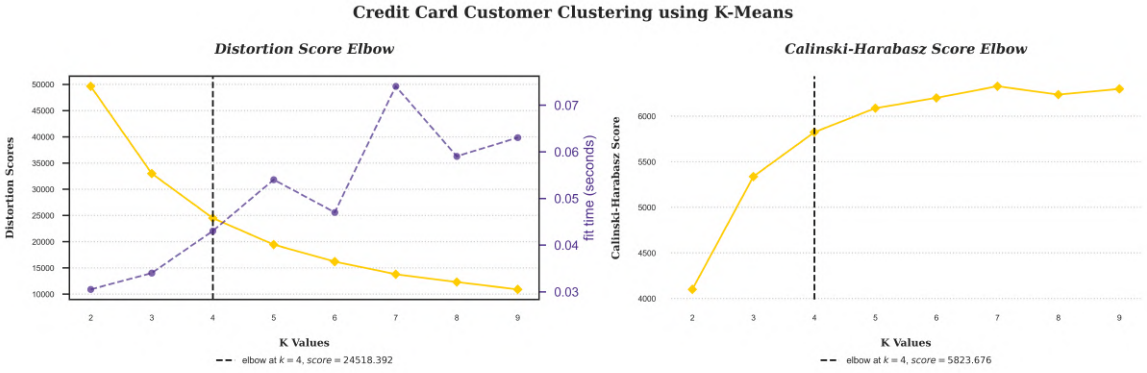
En esta sección, implementaremos las técnicas de clustering mencionadas en la primera sección. Además, se proporcionará una explicación para cada modelo.

7.1 | K-Means

Este algoritmo de aprendizaje no supervisado agrupa datos en clusters, con el objetivo de minimizar la variación dentro de cada cluster. Primero se elige el número de clusters (k) que se desea crear, y luego se seleccionan k puntos iniciales aleatorios como centroides. Luego, cada punto de datos se asigna al cluster cuyo centroide está más cerca. Los centroides se recalculan en función de los puntos que se han asignado a su cluster, y el proceso se repite hasta que no hay cambios en la asignación de los puntos de datos.

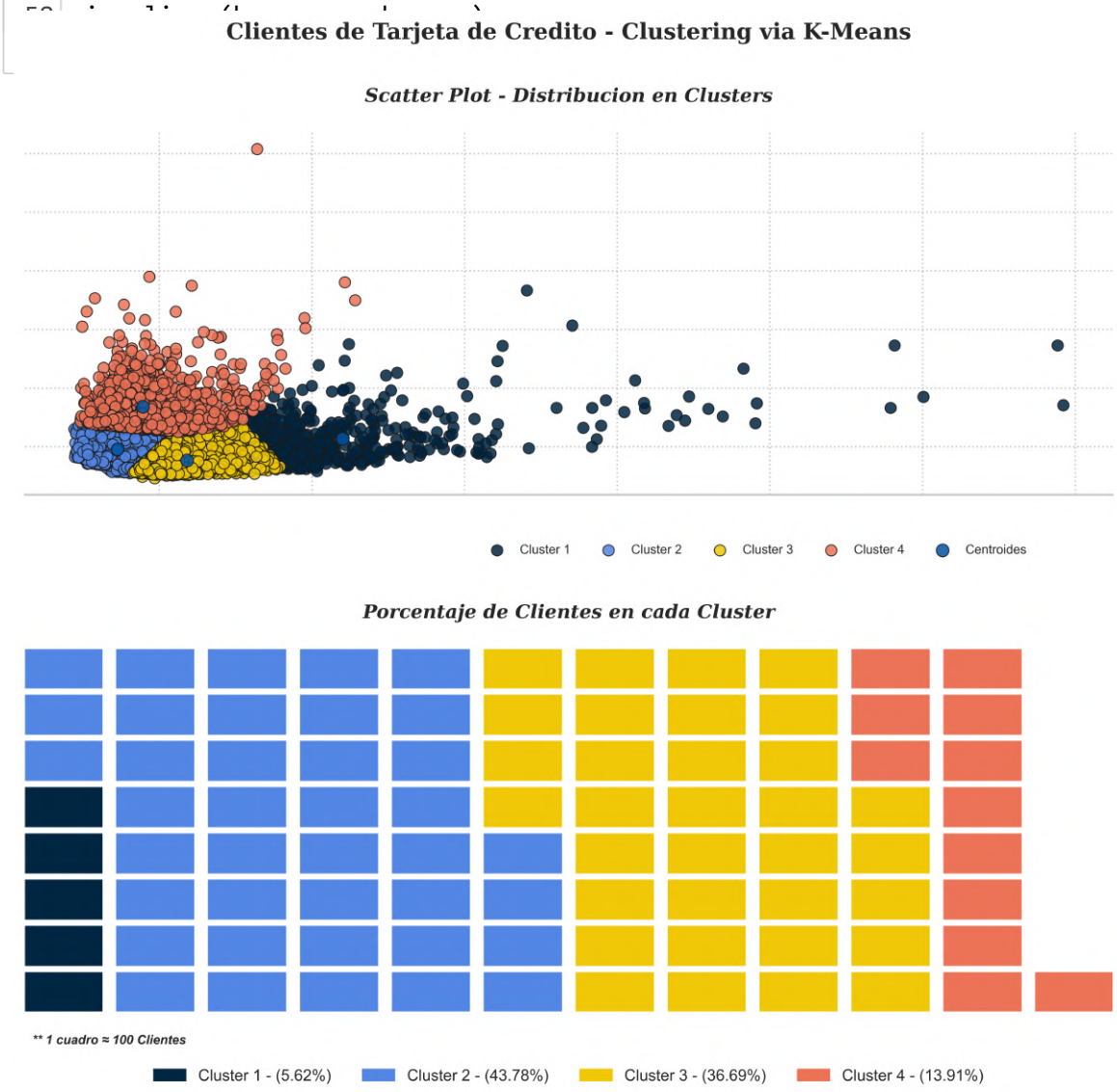
In [286]:

```
# --- Define K-Means Functions ---
def kmeans():

    # --- Figures Settings ---
    color_palette=['#FFCC00', '#54318C']
    set_palette(color_palette)
    title=dict(fontsize=12, fontweight='bold', style='italic', fontfamily='serif')
    text_style=dict(fontweight='bold', fontfamily='serif')
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

    # --- Elbow Score ---
    elbow_score = KElbowVisualizer(KMeans(random_state=32, max_iter=500), k=(2, 10),
    elbow_score.fit(X)
    elbow_score.finalize()
    elbow_score.ax.set_title('Distortion Score Elbow\n', **title)
    elbow_score.ax.tick_params(labelsize=7)
    for text in elbow_score.ax.legend_.texts:
        text.set_fontsize(9)
    for spine in elbow_score.ax.spines.values():
        spine.set_color('None')
    elbow_score.ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), borderpad=
    elbow_score.ax.grid(axis='y', alpha=0.5, color='#9B9A9C', linestyle='dotted')
    elbow_score.ax.grid(axis='x', alpha=0)
    elbow_score.ax.set_xlabel('\nK Values', fontsize=9, **text_style)
    elbow_score.ax.set_ylabel('Distortion Scores\n', fontsize=9, **text_style)

    # --- Elbow Score (Calinski-Harabasz Index) ---
    elbow_score_ch = KElbowVisualizer(KMeans(random_state=32, max_iter=500), k=(2, 10
    elbow_score_ch.fit(X)
    elbow_score_ch.finalize()
    elbow_score_ch.ax.set_title('Calinski-Harabasz Score Elbow\n', **title)
    elbow_score_ch.ax.tick_params(labelsize=7)
    for text in elbow_score_ch.ax.legend_.texts:
        text.set_fontsize(9)
    for spine in elbow_score_ch.ax.spines.values():
        spine.set_color('None')
    elbow_score_ch.ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), borderp
    elbow_score_ch.ax.grid(axis='y', alpha=0.5, color='#9B9A9C', linestyle='dotted')
    elbow_score_ch.ax.grid(axis='x', alpha=0)
    elbow_score_ch.ax.set_xlabel('\nK Values', fontsize=9, **text_style)
    elbow_score_ch.ax.set_ylabel('Calinski-Harabasz Score\n', fontsize=9, **text_styl

    plt.suptitle('Credit Card Customer Clustering using K-Means', fontsize=14, **text

    plt.tight_layout()
    plt.show();

# --- Calling K-Means Functions ---
kmeans();
```


Credit Card Customer Clustering using K-Means
Distortion Score Elbow
Distortion Scores
fit time (seconds)
K Values
elbow at k = 4, score = 24518.392
Calinski-Harabasz Score Elbow
Calinski-Harabasz Score
K Values
elbow at k = 4, score = 5823.676

In [259]:

```
# --- Implementing K-Means ---
kmeans = KMeans(n_clusters=4, random_state=32, max_iter=500)
y_kmeans = kmeans.fit_predict(X)

# --- Define K-Means Visualizer & Plots ---
def visualizer(kmeans, y_kmeans):

    # --- Figures Settings ---
    #cluster_colors=['#FFBB00', '#3C096C', '#9D4EDD', '#FFE270']
    cluster_colors = ['#002642', '#5386E4', '#F0C808', '#EC7357']



    labels = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Centroides']
    title=dict(fontsize=12, fontweight='bold', style='italic', fontfamily='serif')
    text_style=dict(fontweight='bold', fontfamily='serif')
    scatter_style=dict(linewidth=0.65, edgecolor='#1B1B1B', alpha=0.85)
    legend_style=dict(borderpad=2, frameon=False, fontsize=8)
    fig, (ax2, ax3) = plt.subplots(2, 1, figsize=(10, 10))


    y_kmeans_labels = list(set(y_kmeans.tolist()))
    for i in y_kmeans_labels:
        ax2.scatter(X[y_kmeans==i, 0], X[y_kmeans == i, 1], s=50, c=cluster_colors[i]
    ax2.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=65, c
    for spine in ax2.spines.values():
        spine.set_color('None')
    ax2.set_title('Scatter Plot - Distribucion en Clusters\n', **title)
    ax2.legend(labels, bbox_to_anchor=(0.95, -0.05), ncol=5, **legend_style)
    ax2.grid(axis='both', alpha=0.5, color='#9B9A9C', linestyle='dotted')
    ax2.tick_params(left=False, right=False , labelleft=False , labelbottom=False, bo
    ax2.spines['bottom'].set_visible(True)
    ax2.spines['bottom'].set_color('#CAC9CD')

    # --- Waffle Chart ---
    unique, counts = np.unique(y_kmeans, return_counts=True)
    df_waffle = dict(zip(unique, counts))
    total = sum(df_waffle.values())
    wfl_square = {key: value/100 for key, value in df_waffle.items()}
    wfl_label = {key: round(value/total*100, 2) for key, value in df_waffle.items()}

    ax3=plt.subplot(2, 2, (3,4))
    ax3.set_title('Porcentaje de Clientes en cada Cluster\n', **title)
    ax3.set_aspect(aspect='auto')
    Waffle.make_waffle(ax=ax3, rows=8, values=wfl_square, colors=cluster_colors,
                       labels=[f"Cluster {i+1} - ({k}%)" for i, k in wfl_label.items(
                       legend={'loc': 'upper center', 'bbox_to_anchor': (0.5, -0.05),
                               'frameon': False, 'fontsize':10})
    ax3.text(0.01, -0.09, '** 1 cuadro ≈ 100 Clientes', weight = 'bold', style='itali
    # --- Suptitle & WM ---
    plt.suptitle('Clientes de Tarjeta de Credito - Clustering via K-Means\n', fontsiz

    plt.tight_layout()
    plt.show();

# --- Calling K-Means Functions ---
```



In [287]:

```
# --- Evaluate Clustering Quality Function ---
def evaluate_clustering(X, y):
    db_index = round(davies_bouldin_score(X, y), 3)
    s_score = round(silhouette_score(X, y), 3)
    ch_index = round(calinski_harabasz_score(X, y), 3)
    print(clr.start+'.: Evaluate Clustering Quality :.'+clr.end)
    print(clr.color+'*' * 34+clr.end)
    print('.: Davies-Bouldin Index: '+clr.start, db_index)
    print(clr.end+'.: Silhouette Score: '+clr.start, s_score)
    print(clr.end+'.: Calinski Harabasz Index: '+clr.start, ch_index)
    return db_index, s_score, ch_index

# --- Evaluate K-Means Cluster Quality ---
db_kmeans, ss_kmeans, ch_kmeans = evaluate_clustering(X, y_kmeans)
```

```
.: Evaluate Clustering Quality :.
**********************************
.: Davies-Bouldin Index:  0.801
.: Silhouette Score:  0.408
.: Calinski Harabasz Index:  5823.676
```

- 📌 El índice Davies-Bouldin es una métrica para evaluar algoritmos de agrupamiento. Se define como una relación entre la dispersión del cluster y la separación del cluster. Los puntajes oscilan entre 0 y

más. 0 indica un mejor agrupamiento.

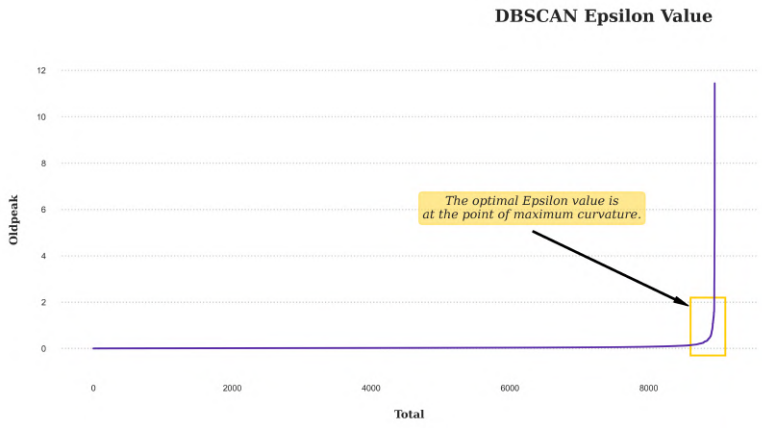
- 📌 El Coeficiente/Puntuación de Silueta es una métrica utilizada para calcular la bondad de una técnica de agrupamiento. Su valor varía de -1 a 1. Cuanto mayor sea la puntuación, mejor. 1 significa que los clusters están bien separados entre sí y claramente distinguibles. Cero significa que los clusters son indiferentes/la distancia entre clusters no es significativa. -1 significa que los clusters están asignados de manera incorrecta.
- 📌 El índice Calinski-Harabasz (también conocido como el criterio de relación de varianza) es la relación entre la suma de la dispersión entre clusters y la dispersión dentro de los clusters para todos los clusters. Cuanto mayor sea la puntuación, mejores serán los resultados.

7.2 | DBSCAN

Este algoritmo también es de aprendizaje no supervisado y se utiliza para agrupar puntos de datos en función de su densidad. El algoritmo comienza con un punto aleatorio y luego busca todos los puntos cercanos que estén dentro de una distancia específica (epsilon). Luego, se identifican los grupos de puntos que están dentro de un radio específico (MinPoints). Los puntos que no están dentro de ningún grupo se consideran valores atípicos.

In [288]:

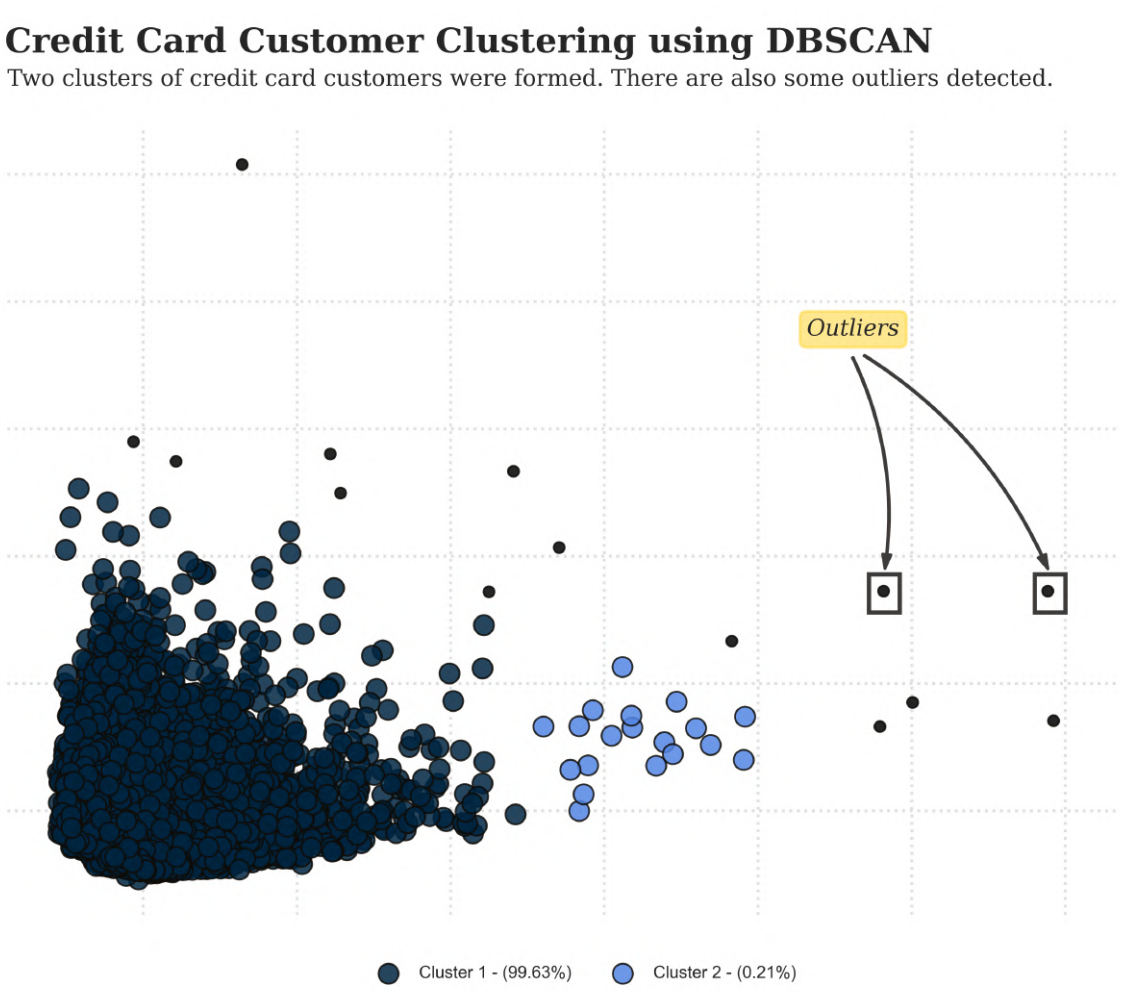
```
# --- Define Epsilon Values ---
def epsilon():

    # --- Calculate Nearest Neighbors ---
    neighbors=NearestNeighbors(n_neighbors=2)
    nbrs=neighbors.fit(X)
    distances, indices=nbrs.kneighbors(X)
    distances=np.sort(distances, axis = 0)

    # --- Figure Settings ---
    bbox=dict(boxstyle='round', pad=0.3, color='#FFDA47', alpha=0.6)
    txt1=dict(textcoords='offset points', va='center', ha='center', fontfamily='serif
    txt2=dict(textcoords='offset points', va='center', fontfamily='serif', style='ita
    kw=dict(arrowstyle='Simple, tail_width=0.1, head_width=0.4, head_length=1', color
    text_style=dict(fontweight='bold', fontfamily='serif')
    fig=plt.figure(figsize=(14, 5))

    # --- Epsilon Plot ---
    distances_1=distances[:, 1]
    ax1=fig.add_subplot(1, 3, (1, 2))
    plt.plot(distances_1, color='#5829A7')
    plt.xlabel('\nTotal', fontsize=9, **text_style)
    plt.ylabel('Oldpeak\n', fontsize=9, **text_style)
    ax1.add_patch(Rectangle((8600, -0.3), 500, 2.5, edgecolor='#FFCC00', fill=False,
    plt.annotate('The optimal Epsilon value is\nat the point of maximum curvature.',
    plt.annotate('', xy=(8600, 1.8), xytext=(6300, 5.1), arrowprops=kw)
    for spine in ax1.spines.values():
        spine.set_color('None')
    plt.grid(axis='y', alpha=0.5, color='#9B9A9C', linestyle='dotted')
    plt.grid(axis='x', alpha=0)
    plt.tick_params(labelsize=7)

    # --- Explanations ---
    ax2=fig.add_subplot(1, 3, 3)
    plt.annotate('From the plot, the maximum curvature\nof the curve is about 2, and
    for spine in ax2.spines.values():
        spine.set_color('None')
    plt.grid(axis='both', alpha=0)
    plt.axis('off')

    plt.suptitle('DBSCAN Epsilon Value\n', fontsize=14, **text_style)

    plt.tight_layout()
    plt.show();

# --- Calling Epsilon Functions ---
epsilon();
```



From the plot, the maximum curvature of the curve is about 2, and thus we picked our Eps as 2.

In [299]:

```
# --- Implementing DBSCAN ---
dbscan = DBSCAN(eps=2, min_samples=4)
y_dbscan = dbscan.fit_predict(X)

# --- Define DBSCAN Result Distribution ---
def dbscan_visualizer(dbscan, y_dbscan):

    # --- Figures Settings ---
    #cluster_colors=['#FFBB00', '#9D4EDD', 'black']
    cluster_colors = ['#002642', '#5386E4', 'black']
    labels = ['Cluster 1', 'Cluster 2', 'Outliers']
    suptitle=dict(fontsize=12, fontweight='heavy', fontfamily='serif')
    title=dict(fontsize=8, fontfamily='serif')
    scatter_style=dict(linewidth=0.65, edgecolor='#100C07', alpha=0.85)
    bbox=dict(boxstyle='round', pad=0.3, color='#FFDA47', alpha=0.6)
    txt=dict(textcoords='offset points', va='center', ha='center', fontfamily='serif'
    legend_style=dict(borderpad=2, frameon=False, fontsize=6)

    # --- Arrow Settings ---
    style = 'Simple, tail_width=0.3, head_width=3, head_length=5'
    kw = dict(arrowstyle=style, color='#3E3B39')
    arrow1 = patches.FancyArrowPatch((23, 18), (24.1, 9.3), connectionstyle='arc3, ra
    arrow2 = patches.FancyArrowPatch((23.3, 18), (29.5, 9.3), connectionstyle='arc3,

    # --- Percentage labels ---
    unique, counts = np.unique(y_dbscan, return_counts=True)
    dbscan_count = dict(zip(unique, counts))
    total = sum(dbscan_count.values())
    dbscan_label = {key: round(value/total*100, 2) for key, value in dbscan_count.ite

    # --- Clusters Distribution ---
    y_dbscan_labels = list(set(y_dbscan.tolist()))
    fig, ax = plt.subplots(1, 1, figsize=(7, 5))
    for i in np.arange(0, 2, 1):
        plt.scatter(X[y_dbscan==i, 0], X[y_dbscan == i, 1], s=50, c=cluster_colors[i]
    plt.scatter(X[y_dbscan==-1, 0], X[y_dbscan == -1, 1], s=15, c=cluster_colors[2],
    for spine in ax.spines.values():
        spine.set_color('None')
    plt.legend([f"Cluster {i+1} - ({k}%)" for i, k in dbscan_label.items()], bbox_to_
    plt.grid(axis='both', alpha=0.3, color='#9B9A9C', linestyle='dotted')
    ax.add_patch(Rectangle((29, 7.8), 1, 1.5, edgecolor='#3E3B39', fill=False, lw=1.5
    ax.add_patch(Rectangle((23.6, 7.8), 1, 1.5, edgecolor='#3E3B39', fill=False, lw=1
    ax.add_patch(arrow1)
    ax.add_patch(arrow2)
    plt.annotate('Outliers', xy=(23, 18.8), xytext=(1, 1), fontsize=8, bbox=bbox, **t
    plt.tick_params(left=False, right=False , labelleft=False , labelbottom=False, bc
    plt.title('Two clusters of credit card customers were formed. There are also some
    plt.suptitle('Credit Card Customer Clustering using DBSCAN', x=0.123, y=0.98, ha=

    plt.show();

# --- Calling DBSCAN Functions ---
dbscan_visualizer(dbscan, y_dbscan);
```



In [193]:

```
# --- Evaluate DBSCAN Cluster Quality ---
db_dbscan, ss_dbscan, ch_dbscan = evaluate_clustering(X, y_dbscan)
```

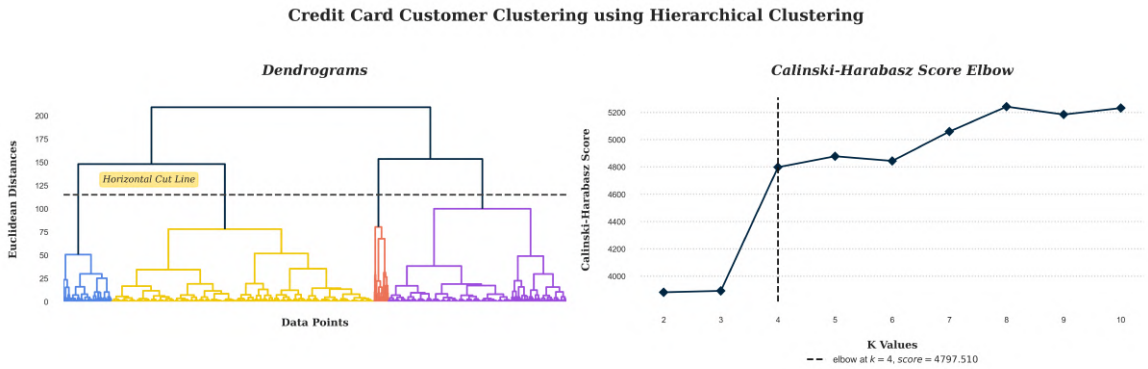
```
.: Evaluate Clustering Quality :.
**********************************
.: Davies-Bouldin Index:  1.287
.: Silhouette Score:  0.803
.: Calinski Harabasz Index:  685.303
```

7.3 | Hierarchical Clustering (Agglomerative)

Hierarchical Clustering (Agglomerative): Este algoritmo también agrupa datos en clusters, pero utiliza una técnica jerárquica. En primer lugar, cada punto de datos se considera un cluster individual. Luego, se unen los dos clusters más cercanos, y este proceso se repite hasta que todos los puntos de datos se agrupan en un solo cluster. Se puede visualizar el proceso en un dendrograma, que muestra la estructura jerárquica de los clusters.

In [300]:

```
# --- Define Dendrogram ---
def agg_dendrogram():

    # --- Figure Settings ---
    #color_palette=['#472165', '#FFBB00', '#3C096C', '#9D4EDD', '#FFE270']
    color_palette = ['#002642', '#5386E4', '#F0C808', '#EC7357','#9D4EDD']
    set_palette(color_palette)
    text_style=dict(fontweight='bold', fontfamily='serif')
    ann=dict(textcoords='offset points', va='center', ha='center', fontfamily='serif'
    title=dict(fontsize=12, fontweight='bold', style='italic', fontfamily='serif')
    bbox=dict(boxstyle='round', pad=0.3, color='#FFDA47', alpha=0.6)
    fig=plt.figure(figsize=(14, 5))

    # --- Dendrogram Plot ---
    ax1=fig.add_subplot(1, 2, 1)
    dend=shc.dendrogram(shc.linkage(X, method='ward', metric='euclidean'))
    plt.axhline(y=115, color='#3E3B39', linestyle='--')
    plt.xlabel('\nData Points', fontsize=9, **text_style)
    plt.ylabel('Euclidean Distances\n', fontsize=9, **text_style)
    plt.annotate('Horizontal Cut Line', xy=(15000, 130), xytext=(1, 1), fontsize=8, b
    plt.tick_params(labelbottom=False)
    for spine in ax1.spines.values():
        spine.set_color('None')
    plt.grid(axis='both', alpha=0)
    plt.tick_params(labelsize=7)
    plt.title('Dendrograms\n', **title)

    # --- Elbow Score (Calinski-Harabasz Index) ---
    ax2=fig.add_subplot(1, 2, 2)
    elbow_score_ch = KElbowVisualizer(AgglomerativeClustering(), metric='calinski_har
    elbow_score_ch.fit(X)
    elbow_score_ch.finalize()
    elbow_score_ch.ax.set_title('Calinski-Harabasz Score Elbow\n', **title)
    elbow_score_ch.ax.tick_params(labelsize=7)
    for text in elbow_score_ch.ax.legend_.texts:
        text.set_fontsize(9)
    for spine in elbow_score_ch.ax.spines.values():
        spine.set_color('None')
    elbow_score_ch.ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), borderp
    elbow_score_ch.ax.grid(axis='y', alpha=0.5, color='#9B9A9C', linestyle='dotted')
    elbow_score_ch.ax.grid(axis='x', alpha=0)
    elbow_score_ch.ax.set_xlabel('\nK Values', fontsize=9, **text_style)
    elbow_score_ch.ax.set_ylabel('Calinski-Harabasz Score\n', fontsize=9, **text_styl

    plt.suptitle('Credit Card Customer Clustering using Hierarchical Clustering\n', f

    plt.tight_layout()
    plt.show();

# --- Calling Dendrogram Functions ---
agg_dendrogram();
```

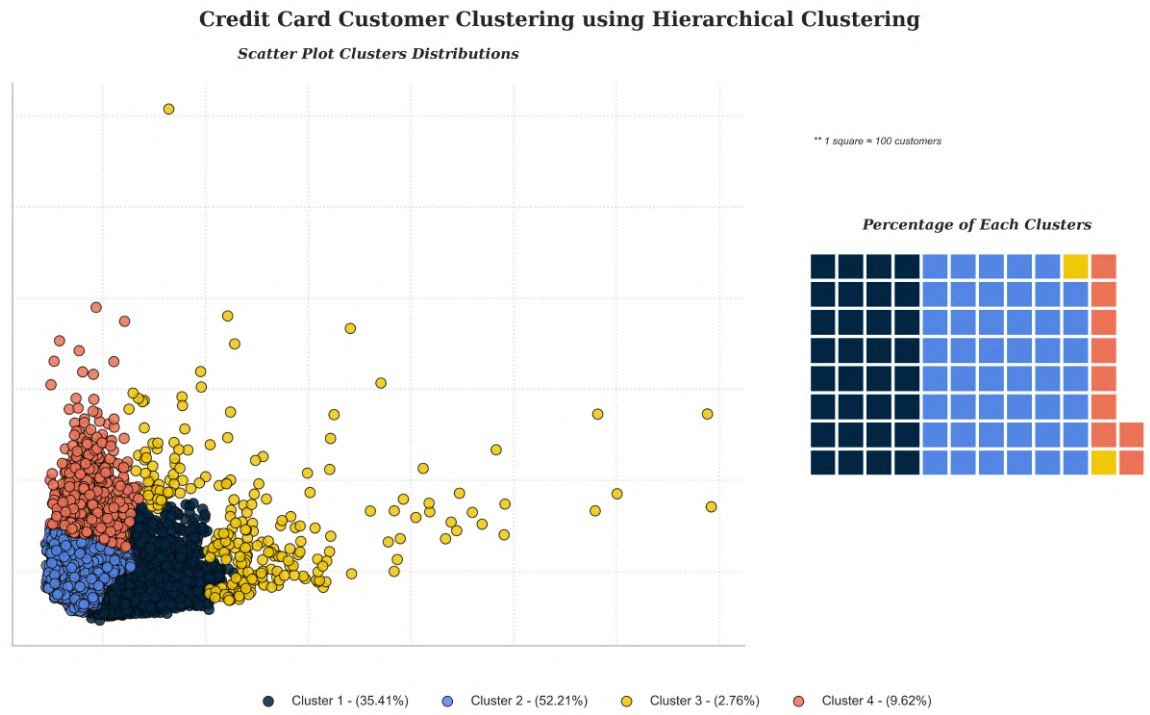


Basándonos en la distancia euclidiana en el dendrograma de arriba, se puede concluir que el número de clusters será cuatro ya que la línea vertical más alta/mayor distancia se encuentra en la primera línea/rama (a la izquierda de la imagen) y el umbral corta el dendrograma en cuatro partes. Además, según la puntuación de Calinski-Harabasz, el número óptimo de clusters obtenido es 4.

A continuación, implementaremos este número en el algoritmo de clustering aglomerativo y visualizaremos y evaluaremos los clusters creados.

In [301]:

```
# --- Implementing Hierarchical Clustering ---
agg_cluster = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='wa
y_agg_cluster = agg_cluster.fit_predict(X)

# --- Define Hierarchical Clustering Distributions ---
def agg_visualizer(agg_cluster, y_agg_cluster):

    # --- Figures Settings ---
    cluster_colors=['#002642', '#5386E4', '#F0C808', '#EC7357']


    labels = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4']
    suptitle=dict(fontsize=14, fontweight='heavy', fontfamily='serif')
    title=dict(fontsize=10, fontweight='bold', style='italic', fontfamily='serif')
    scatter_style=dict(linewidth=0.65, edgecolor='#100C07', alpha=0.85)
    legend_style=dict(borderpad=2, frameon=False, fontsize=9)
    fig=plt.figure(figsize=(14, 7))

    # --- Percentage Labels ---
    unique, counts = np.unique(y_agg_cluster, return_counts=True)
    df_waffle = dict(zip(unique, counts))
    total = sum(df_waffle.values())
    wfl_square = {key: value/100 for key, value in df_waffle.items()}
    wfl_label = {key: round(value/total*100, 2) for key, value in df_waffle.items()}

    # --- Clusters Distribution ---
    y_agg_labels = list(set(y_agg_cluster.tolist()))
    ax1=fig.add_subplot(1, 3, (1, 2))
    for i in y_agg_labels:
        ax1.scatter(X[y_agg_cluster==i, 0], X[y_agg_cluster == i, 1], s=50, c=cluster
    for spine in ax1.spines.values():
        spine.set_color('None')
    for spine in ['bottom', 'left']:
        ax1.spines[spine].set_visible(True)
        ax1.spines[spine].set_color('#CAC9CD')
    ax1.legend([f"Cluster {i+1} - ({k}%)" for i, k in wfl_label.items()], bbox_to_anc
    ax1.grid(axis='both', alpha=0.3, color='#9B9A9C', linestyle='dotted')
    ax1.tick_params(left=False, right=False , labelleft=False , labelbottom=False, bc
    plt.title('Scatter Plot Clusters Distributions\n', **title)

    # --- Waffle Chart ---
    ax2=fig.add_subplot(1, 3, 3)
    ax2.set_title('Percentage of Each Clusters\n', **title)
    ax2.set_aspect(aspect='equal')
    Waffle.make_waffle(ax=ax2, rows=8, values=wfl_square, colors=cluster_colors)
    ax2.get_legend().remove()


    ax2.text(0.01, 1.5, '** 1 square ≈ 100 customers', style='italic', fontsize=7)

    plt.suptitle('Credit Card Customer Clustering using Hierarchical Clustering\n', *
    plt.show();

# --- Calling Hierarchical Clustering Functions ---
agg_visualizer(agg_cluster, y_agg_cluster);
```



A partir de la implementación de clustering jerárquico, se puede ver que se formaron 4 grupos. De los 4 grupos, el grupo 2 tiene la mayoría de los puntos de datos, seguido por el grupo 1. Sin embargo, cuando se comparan con los resultados del clustering por K-Means, los resultados del grupo 2 utilizando jerárquico tienen un porcentaje más significativo. Además, el algoritmo de clustering jerárquico considera que los valores atípicos son parte del grupo 3.

El último paso es evaluar la calidad del clustering que ofrece el clustering jerárquico. Se utilizarán el coeficiente de silueta y el índice Davies-Bouldin para evaluar la calidad.

In [292]:

```
# --- Evaluate DBSCAN Cluster Quality ---
db_agg, ss_agg, ch_agg = evaluate_clustering(X, y_agg_cluster)
```

```
.: Evaluate Clustering Quality :.
**********************************
.: Davies-Bouldin Index:  0.863
.: Silhouette Score:  0.388
.: Calinski Harabasz Index:  4797.51
```

Evaluando los modelos utilizados

El siguiente paso es evaluar la calidad del agrupamiento proporcionado por K-Means. La evaluación de calidad utilizará el índice Davies-Bouldin, la puntuación de silueta y el índice Calinski-Harabasz.

In [293]:

```
# --- Comparison Table ---
compare = pd.DataFrame({'Model': ['K-Means', 'DBSCAN', 'Hierarchical Clustering'],
                        'Davies-Bouldin Index': [db_kmeans, db_dbscan, db_agg],
                        'Silhouette Score': [ss_kmeans, ss_dbscan, ss_agg],
                       'Calinski-Harabasz Index': [ch_kmeans, ch_dbscan, ch_agg]})

# --- Create Accuracy Comparison Table ---
print(clr.start+'.: Model Accuracy Comparison :.'+clr.end)
print(clr.color+'*' * 32+clr.end)
compare.sort_values(by='Model', ascending=False).style.background_gradient(cmap='infe
```

```
.: Model Accuracy Comparison :.
********************************
```

Out[293]:

	Model	Davies-Bouldin Index	Silhouette Score	Calinski-Harabasz Index
0	K-Means	0.801000	0.408000	5823.676000
2	Hierarchical Clustering	0.863000	0.388000	4797.510000
1	DBSCAN	1.287000	0.803000	685.303000

- La tabla anterior muestra que el algoritmo K-Means tiene el índice Davies-Bouldin más bajo en comparación con los otros dos algoritmos, por lo que se puede concluir que **K-Means tiene una calidad de agrupamiento decente en comparación con los otros dos algoritmos**.
- Sin embargo, según la puntuación de silueta, **K-Means tiene** la segunda puntuación de silueta más alta (hay algunos **clusters superpuestos formados con este algoritmo**).
- A partir de los resultados del índice Calinski-Harabasz, K-Means tiene el índice más alto en comparación con otros algoritmos. Esto indica que **K-Means funciona mejor y es más denso que otros algoritmos.**

Perfilado de clientes

8. | K-means results

Debido al performance de k-means, de aquí en adelante solo sera utilizada su clusterización para el análisis y perfilado de los clientes.

En la siguiente gráfica, podemos ver como se agrupan los mismos teniendo en cuenta el PCA, también podemos ver cuál es el % de clientes en cada grupo.

In [294]:

```
# --- Add K-Means Prediction to Data Frame ----
df['cluster_result'] = y_kmeans+1
df['cluster_result'] = 'Cluster '+df['cluster_result'].astype(str)

# --- Calculationg Overall Mean from Current Data Frame ---
df_profile_overall = pd.DataFrame()

df_profile_overall['Overall'] = df.describe().loc[['mean']].T

# --- Summarize Mean of Each Clusters ---
df_cluster_summary = df.groupby('cluster_result').describe().T.reset_index().rename(c
df_cluster_summary = df_cluster_summary[df_cluster_summary['Metrics'] == 'mean'].set_


# --- Combining Both Data Frame ---
print(clr.start+'.: Summarize of Each Clusters :.'+clr.end)
print(clr.color+'*' * 33)
df_profile = df_cluster_summary.join(df_profile_overall).reset_index()


df_profile.style.background_gradient(axis=1,cmap='YlOrBr')
```

```
.: Summarize of Each Clusters :.
*********************************
```

Out[294]:

	Column Name	Metrics	Cluster 1	Cluster 2	Cluster 3
0	BALANCE	mean	3401.840056	1012.915503	824.18035
1	BALANCE_FREQUENCY	mean	0.988427	0.799792	0.91999
2	PURCHASES	mean	6894.613917	223.081279	1236.49940
3	ONEOFF_PURCHASES	mean	4511.889901	157.576608	621.73876
4	INSTALLMENTS_PURCHASES	mean	2383.916859	65.833785	614.96572
5	CASH_ADVANCE	mean	773.154467	614.588758	147.44219
6	PURCHASES_FREQUENCY	mean	0.954443	0.190099	0.86244
7	ONEOFF_PURCHASES_FREQUENCY	mean	0.726667	0.074901	0.30188
8	PURCHASES_INSTALLMENTS_FREQUENCY	mean	0.808946	0.109962	0.67511
9	CASH_ADVANCE_FREQUENCY	mean	0.084806	0.122459	0.03069
10	CASH_ADVANCE_TRX	mean	2.363817	2.264421	0.56973
11	PURCHASES_TRX	mean	82.902584	2.960949	21.31729
12	CREDIT_LIMIT	mean	9541.650099	3109.010550	4250.05184
13	PAYMENTS	mean	6723.271522	856.400147	1328.94902
14	MINIMUM_PAYMENTS	mean	1830.297811	589.839851	600.41054
15	PRC_FULL_PAYMENT	mean	0.288014	0.065965	0.28286
16	TENURE	mean	11.960239	11.364216	11.66169

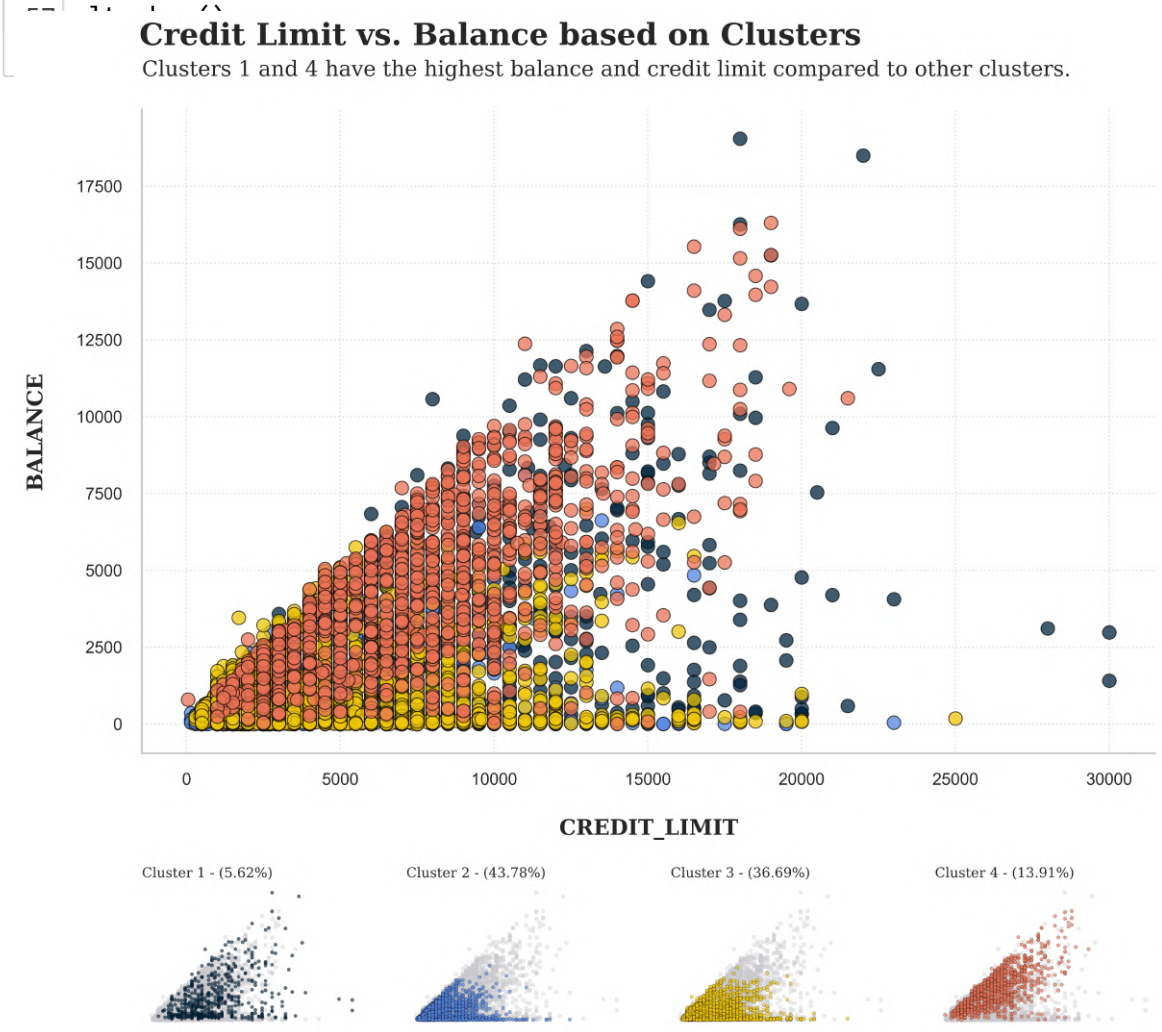
Basándonos en la tabla anterior, se puede concluir que cada cluster tiene las siguientes características:

- **Cluster 1 (Full Player Customer):** Los clientes en este cluster son usuarios activos de la tarjeta de crédito del banco. Esto se puede observar por la frecuencia con la que el saldo cambia y el monto de los saldos es lo suficientemente alto en comparación con los otros clusters. Además, en comparación con los otros clusters, este cluster tiene valores promedio más altos en varios aspectos. Los clientes de tarjetas de crédito en este cluster también usan activamente las tarjetas de crédito para facilitar transacciones y pagos a plazos. Los adelantos en efectivo, las transacciones y los pagos a plazos en este cluster también ocurren con más frecuencia. La antigüedad relativamente alta también muestra que el puntaje crediticio en este cluster es muy bueno.
- **Cluster 2 (Usuarios Principiantes/Estudiantes):** En contraste con el cluster 1, los clientes rara vez o casi nunca usan las tarjetas de crédito para transacciones y pagos a plazos en este cluster. Esto se debe a que el cliente tiene un saldo relativamente pequeño, la frecuencia del saldo cambia rara vez y los pagos a plazos son muy bajos. Además, un límite de crédito bajo también muestra que los clientes rara vez o casi nunca usan las tarjetas de crédito para procesar transacciones de crédito, y los clientes en este cluster también rara vez hacen adelantos en efectivo. Por lo tanto, se puede suponer que los clientes utilizan las tarjetas de crédito para procesos de adelanto de efectivo solo con la suficiente frecuencia. Además, el bajo saldo permite que los clientes de este cluster sean estudiantes o nuevos usuarios que utilizan las tarjetas de crédito en este banco.
- **Cluster 3 (Usuarios de Pagos a Plazos):** En este cluster, los clientes utilizan las tarjetas de crédito específicamente para pagos a plazos. Esto se debe al nivel relativamente alto de transacciones que utilizan pagos a plazos en este cluster. Además, los clientes en este cluster a menudo realizan transacciones con montos muy grandes por transacción y la frecuencia y transacciones de adelantos en efectivo son muy pequeñas. Los clientes en este cluster rara vez hacen pagos y adelantos en efectivo y tienen una frecuencia de adelantos en efectivo y un monto de pagos relativamente pequeños. Se puede concluir que los clientes en este cluster son muy adecuados para tarjetas de crédito específicamente para necesidades de pagos a plazos.
- **Cluster 4 (Usuarios de Adelantos en Efectivo/Retiros):** Los clientes en este cluster tienen saldos altos, la frecuencia de los saldos siempre cambia y la frecuencia de adelantos en efectivo y pagos anticipados es alta. Además, los clientes en este cluster tienen las tasas de interés más bajas en comparación con los otros clusters y tienen el segundo límite de crédito más alto y los pagos de los cuatro clusters. Sin embargo, los usuarios de tarjetas de crédito en este cluster rara vez hacen pagos a plazos o compras únicas y tienen la tercera antigüedad más alta de los cuatro clusters. Por lo tanto, se puede concluir que los clientes en este cluster solo usan tarjetas de crédito para la necesidad de retirar dinero o hacer adelantos en efectivo.

Las siguientes son visualizaciones sobre 2 variables para cada cluster:

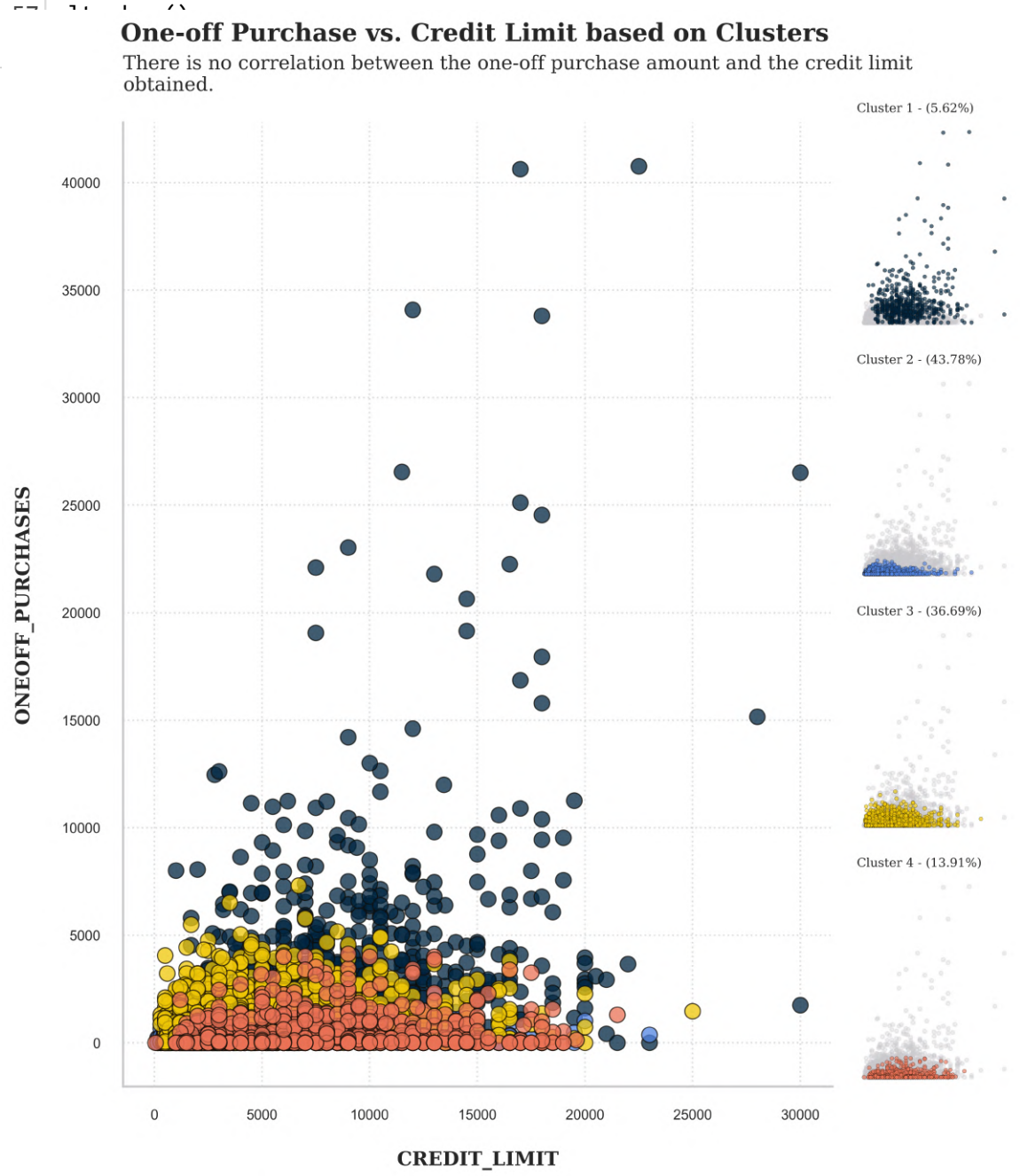
In [302]:

```
# --- Cluster Visualization 1: Variables ---
scatter_style=dict(linewidth=0.65, edgecolor='#100C07', alpha=0.75)
sub_scatter_style_color=dict(s=5, alpha=0.65, linewidth=0.15, zorder=10, edgecolor='#
sub_scatter_style_grey=dict(s=5, alpha=0.3, linewidth=0.7, zorder=5, color='#CAC9CD')
grid_style=dict(alpha=0.3, color='#9B9A9C', linestyle='dotted', zorder=1)
xy_label=dict(fontweight='bold', fontsize=14, fontfamily='serif')
suptitle=dict(fontsize=20, fontweight='heavy', fontfamily='serif')
title=dict(fontsize=14, fontfamily='serif')
color_pallete=['#002642', '#5386E4', '#F0C808', '#EC7357']
sub_axes=[None] * 4

# --- Cluster Visualization 1: Data Frame ---
df_cv1 = df[['CREDIT_LIMIT', 'BALANCE', 'cluster_result']]
cluster_result = sorted(df_cv1['cluster_result'].unique())

# --- Cluster Visualization 1: Settings ---
fig = plt.figure(figsize=(12, 16))
gs = fig.add_gridspec(4, 4)
ax = fig.add_subplot(gs[:4, :])
ax.set_aspect(1)

# --- Cluster Visualization 1: Main Scatter Plot ---
for x in range(len(cluster_result)):
    df_cv1_x = df_cv1[df_cv1['cluster_result']==cluster_result[x]]

    ax.scatter(df_cv1_x['CREDIT_LIMIT'], df_cv1_x['BALANCE'], s=80, color=color_palle
    ax.set_title('Clusters 1 and 4 have the highest balance and credit limit compared
    ax.set_xlabel('\nCREDIT_LIMIT', **xy_label)
    ax.set_ylabel('BALANCE\n', **xy_label)
    ax.grid(axis='y', which='major', **grid_style)
    ax.grid(axis='x', which='major', **grid_style)
    for spine in ax.spines.values():
        spine.set_color('None')
    for spine in ['bottom', 'left']:
        ax.spines[spine].set_visible(True)
        ax.spines[spine].set_color('#CAC9CD')
    plt.xticks(fontsize=11)
    plt.yticks(fontsize=11)

# --- Cluster Visualization 1: Sub Plots ---
for idx, clstr in enumerate(cluster_result):
    sub_axes[idx] = fig.add_subplot(gs[3, idx], aspect=1)

    sub_axes[idx].scatter(df_cv1[df_cv1['cluster_result']!=clstr]['CREDIT_LIMIT'], df
    sub_axes[idx].scatter(df_cv1[df_cv1['cluster_result']==clstr]['CREDIT_LIMIT'], df

    cnt = round((df_cv1['cluster_result']==clstr).sum()/8950*100, 2)
    sub_axes[idx].set_title(f'{clstr} - ({cnt}%)', loc='left', fontsize=9, fontfamily
    sub_axes[idx].set_xticks([])
    sub_axes[idx].set_yticks([])
    for spine in sub_axes[idx].spines.values():
        spine.set_color('None')

# --- Cluster Visualization 1: Title ---
plt.suptitle('Credit Limit vs. Balance based on Clusters', x=0.123, y=0.73, ha='left'

```


Credit Limit vs. Balance based on Clusters
Clusters 1 and 4 have the highest balance and credit limit compared to other clusters.
BALANCE
17500
15000
12500
10000
7500
5000
2500
0
0
5000
10000
15000
20000
25000
30000
CREDIT_LIMIT
Cluster 1 - (5.62%)
Cluster 2 - (43.78%)
Cluster 3 - (36.69%)
Cluster 4 - (13.91%)

In [303]:

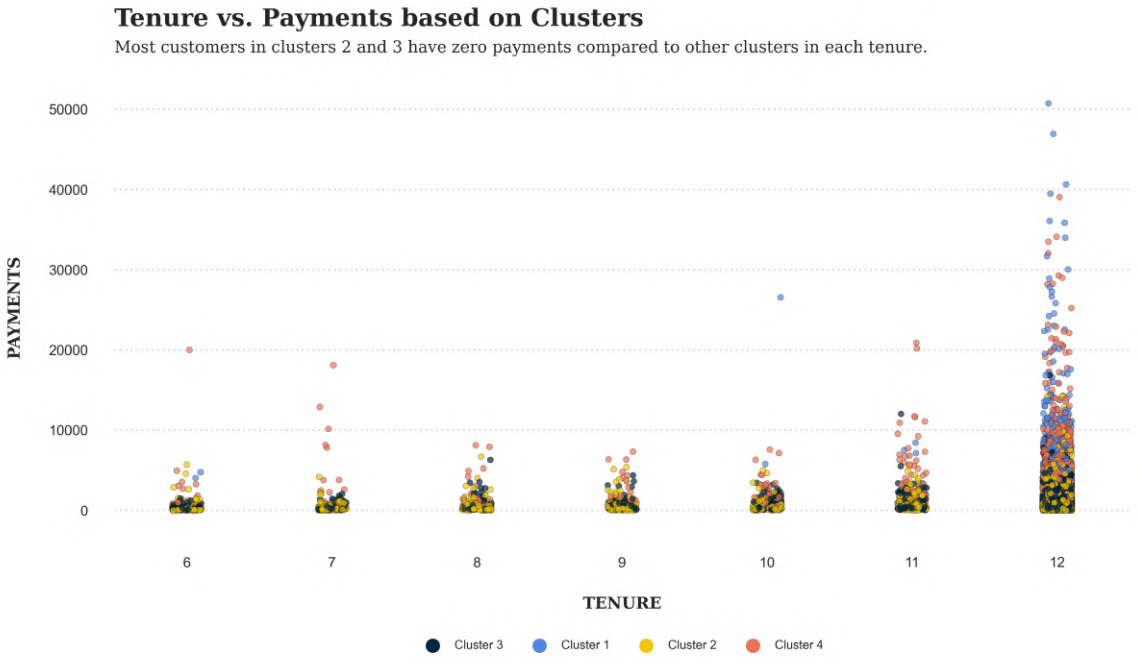
```
# --- Cluster Visualization 2: Variables ---
scatter_style=dict(linewidth=0.65, edgecolor='#100C07', alpha=0.75)
sub_scatter_style_color=dict(s=5, alpha=0.65, linewidth=0.15, zorder=10, edgecolor='#
sub_scatter_style_grey=dict(s=5, alpha=0.3, linewidth=0.7, zorder=5, color='#CAC9CD')
grid_style=dict(alpha=0.3, color='#9B9A9C', linestyle='dotted', zorder=1)
xy_label=dict(fontsize=11, fontweight='bold', fontfamily='serif')
suptitle=dict(fontsize=14, fontweight='heavy', fontfamily='serif')
title=dict(fontsize=11, fontfamily='serif')
color_pallete=['#002642', '#5386E4', '#F0C808', '#EC7357']
sub_axes=[None] * 4

# --- Cluster Visualization 2: Data Frame ---
df_cv2 = df[['CREDIT_LIMIT', 'ONEOFF_PURCHASES', 'cluster_result']]
cluster_result = sorted(df_cv1['cluster_result'].unique())

# --- Cluster Visualization 2: Settings ---
fig = plt.figure(figsize=(12, 10))
gs = fig.add_gridspec(4, 4)
ax = fig.add_subplot(gs[:4, :4])
ax.set_aspect(1)

# --- Cluster Visualization 2: Main Scatter Plot ---
for x in range(len(cluster_result)):
    df_cv2_x = df_cv2[df_cv2['cluster_result']==cluster_result[x]]

    ax.scatter(df_cv2_x['CREDIT_LIMIT'], df_cv2_x['ONEOFF_PURCHASES'], s=80, color=cc
    ax.set_title('There is no correlation between the one-off purchase amount and the
    ax.set_xlabel('\nCREDIT_LIMIT', **xy_label)
    ax.set_ylabel('ONEOFF_PURCHASES\n', **xy_label)
    ax.grid(axis='y', which='major', **grid_style)
    ax.grid(axis='x', which='major', **grid_style)
    for spine in ax.spines.values():
        spine.set_color('None')
    for spine in ['bottom', 'left']:
        ax.spines[spine].set_visible(True)
        ax.spines[spine].set_color('#CAC9CD')
    plt.xticks(fontsize=8)
    plt.yticks(fontsize=8)

# --- Cluster Visualization 2: Sub Plots ---
for idx, clstr in enumerate(cluster_result):
    sub_axes[idx] = fig.add_subplot(gs[idx, 3], aspect=1)

    sub_axes[idx].scatter(df_cv2[df_cv2['cluster_result']!=clstr]['CREDIT_LIMIT'], df
    sub_axes[idx].scatter(df_cv2[df_cv2['cluster_result']==clstr]['CREDIT_LIMIT'], df

    cnt = round((df_cv2['cluster_result']==clstr).sum()/8950*100, 2)
    sub_axes[idx].set_title(f'{clstr} - ({cnt}%)', loc='left', fontsize=7, fontfamily
    sub_axes[idx].set_xticks([])
    sub_axes[idx].set_yticks([])
    for spine in sub_axes[idx].spines.values():
        spine.set_color('None')

# --- Cluster Visualization 2: Title ---
plt.suptitle('One-off Purchase vs. Credit Limit based on Clusters', x=0.275, y=0.96,

```



Las compras al contado no tiene relacion (aparente) con el límite de crédito adicional obtenido por el usuario. En la figura de arriba, como se mencionó anteriormente, se puede ver que el grupo 1 tiene un cliente con la mayor cantidad de compra para una transacción.

In [304]:

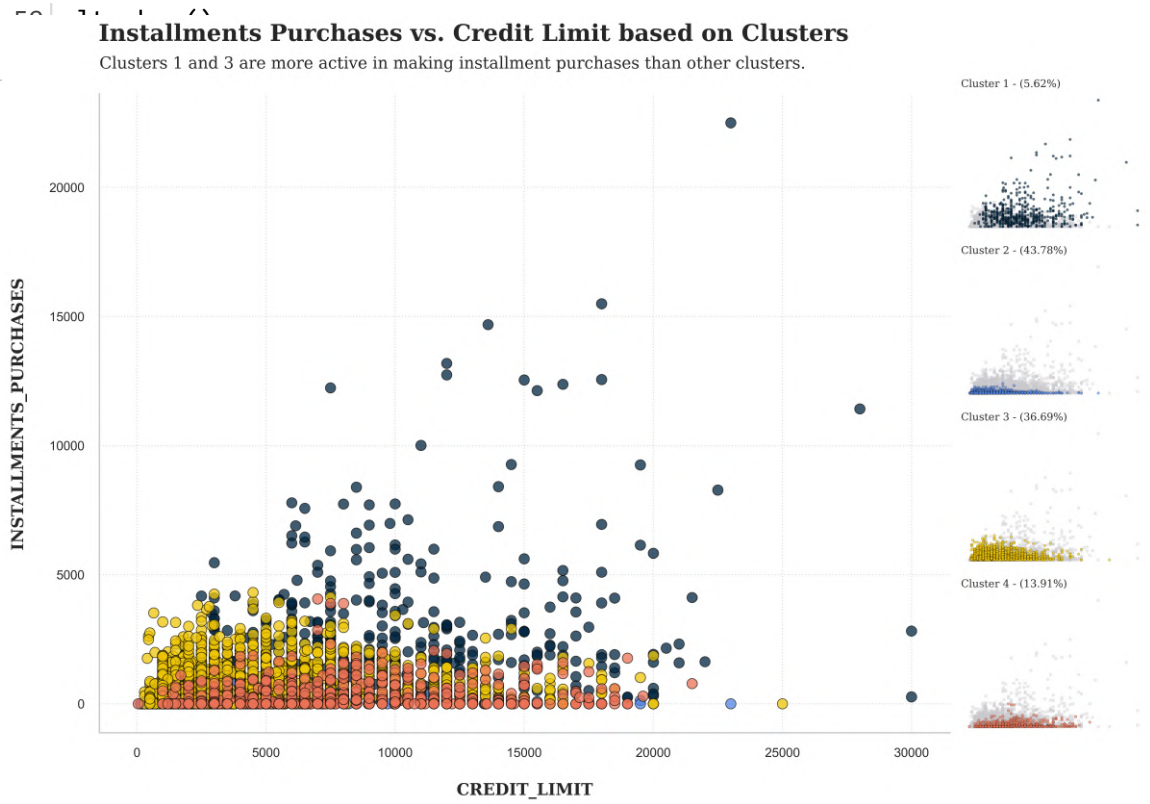
```
# --- Cluster Visualization 3: Data Frame ---
df_cv3 = df[['TENURE', 'PAYMENTS', 'cluster_result']]

# --- Cluster Visualization 3: Variables ---
color_pallete = ['#002642', '#5386E4', '#F0C808', '#EC7357']
suptitle = dict(fontsize=12, ha='left', fontweight='heavy', fontfamily='serif')
title = dict(fontsize=8, loc='left', fontfamily='serif')
cluster_result = sorted(df_cv3['cluster_result'].unique())
stripplot_style = dict(edgecolor='#100C07', s=3, linewidth=0.15, alpha=0.7, palette=c
legend_style = dict(ncol=5, borderpad=3, frameon=False, fontsize=6, title=None)
xy_label = dict(fontweight='bold', fontsize=8, fontfamily='serif')
grid_style = dict(alpha=0.3, color='#9B9A9C', linestyle='dotted', zorder=1)

# --- Cluster Visualization 3: Visuals ---
stplot=sns.stripplot(data=df_cv3, x='TENURE', y='PAYMENTS', hue='cluster_result', **s
sns.move_legend(stplot, 'upper center', bbox_to_anchor=(0.5, -0.15), **legend_style)
sns.despine(top=True, right=True, left=True, bottom=True)
plt.suptitle('Tenure vs. Payments based on Clusters', x=0.125, y=1.01, **suptitle)
plt.title('Most customers in clusters 2 and 3 have zero payments compared to other cl
plt.xlabel('\nTENURE', **xy_label)
plt.ylabel('PAYMENTS\n', **xy_label)
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(axis='x', alpha=0)
plt.grid(axis='y', **grid_style)

plt.gcf().set_size_inches(9, 4)
plt.show();
```



La mayoría de los clientes en los grupos 2 y 3 no tienen pagos en comparación con otros grupos en cada periodo de tiempo. Como se mencionó anteriormente, se puede observar que la mayoría de los clientes tienden a elegir un plazo de 12 meses.

In [305]:

```
# --- Cluster Visualization 4: Data Frame ---
df_cv4 = df[['INSTALLMENTS_PURCHASES', 'CREDIT_LIMIT', 'cluster_result']]

# --- Cluster Visualization 4: Variables ---
cluster_result = sorted(df_cv4['cluster_result'].unique())
scatter_style=dict(linewidth=0.65, edgecolor='#100C07', alpha=0.75)
sub_scatter_style_color=dict(s=5, alpha=0.65, linewidth=0.15, zorder=10, edgecolor='#
sub_scatter_style_grey=dict(s=5, alpha=0.3, linewidth=0.7, zorder=5, color='#CAC9CD')
grid_style=dict(alpha=0.3, color='#9B9A9C', linestyle='dotted', zorder=1)
xy_label=dict(fontsize=14, fontweight='bold', fontfamily='serif')
suptitle=dict(fontsize=20, fontweight='heavy', fontfamily='serif')
title=dict(fontsize=14, fontfamily='serif')
xy_label=dict(fontweight='bold', fontsize=14, fontfamily='serif')
grid_style=dict(alpha=0.3, color='#9B9A9C', linestyle='dotted', zorder=1)
color_pallete=['#002642', '#5386E4', '#F0C808', '#EC7357']
sub_axes=[None] * 4

# --- Cluster Visualization 4: Settings ---
fig = plt.figure(figsize=(21, 10))
gs = fig.add_gridspec(4, 4)
ax = fig.add_subplot(gs[:4, :4])
ax.set_aspect(1)

# --- Cluster Visualization 4: Main Scatter Plot ---
for x in range(len(cluster_result)):
    df_cv4_x = df_cv4[df_cv4['cluster_result']==cluster_result[x]]

    ax.scatter(df_cv4_x['CREDIT_LIMIT'], df_cv4_x['INSTALLMENTS_PURCHASES'], s=80, co
    ax.set_title('Clusters 1 and 3 are more active in making installment purchases th
    ax.set_xlabel('\nCREDIT_LIMIT', **xy_label)
    ax.set_ylabel('INSTALLMENTS_PURCHASES\n', **xy_label)
    ax.grid(axis='y', which='major', **grid_style)
    ax.grid(axis='x', which='major', **grid_style)
    for spine in ax.spines.values():
        spine.set_color('None')
    for spine in ['bottom', 'left']:
        ax.spines[spine].set_visible(True)
        ax.spines[spine].set_color('#CAC9CD')
    plt.xticks(fontsize=11)
    plt.yticks(fontsize=11)

# --- Cluster Visualization 4: Sub Plots ---
for idx, clstr in enumerate(cluster_result):
    sub_axes[idx] = fig.add_subplot(gs[idx, 3], aspect=1)

    sub_axes[idx].scatter(df_cv4[df_cv4['cluster_result']!=clstr]['CREDIT_LIMIT'], df
    sub_axes[idx].scatter(df_cv4[df_cv4['cluster_result']==clstr]['CREDIT_LIMIT'], df

    cnt = round((df_cv4['cluster_result']==clstr).sum()/8950*100, 2)
    sub_axes[idx].set_title(f'{clstr} - ({cnt}%)', loc='left', fontsize=9, fontfamily
    sub_axes[idx].set_xticks([])
    sub_axes[idx].set_yticks([])
    for spine in sub_axes[idx].spines.values():
        spine.set_color('None')

# --- Cluster Visualization 4: Title ---
plt.suptitle('Installments Purchases vs. Credit Limit based on Clusters', x=0.268, y=

```



It can be seen that clusters 1 and 3 have more installment purchases than clusters 2 and 4. However, it can also be seen that a large number of installment purchases are not correlated with the credit limit increase.

8.2 | Strategy Suggestions 💡

Basándonos en los resultados del perfil anterior y en el análisis realizado para cada uno de los clusters, aquí hay algunas sugerencias para estrategias:

- **Los clientes del cluster 1** pueden convertirse en el objetivo principal para el marketing de tarjetas de crédito. **Esto se debe a que los clientes en este cluster son muy activos en el uso de tarjetas de crédito y tienen la antigüedad y los límites de crédito más altos en comparación con los otros clusters.** Al enfocar el marketing en este cluster, los bancos pueden aumentar sus ganancias al usar más tarjetas de crédito y reducir los costos de marketing incurridos. Los bancos pueden ofrecer beneficios o recompensas por el uso de tarjetas de crédito para atraer a los clientes a usar las tarjetas de crédito con más frecuencia.
- Para las tarjetas de crédito para pagos a plazos, los bancos pueden centrar su marketing en los clientes del cluster 3. Esto se debe a que **los clientes en el cluster 3 son más propensos a realizar transacciones con tarjeta de crédito para fines de pagos a plazos**. Los bancos pueden ofrecer programas de pagos a plazos con intereses bajos o del 0% que se puedan utilizar para diversas necesidades de pagos a plazos a los clientes en este cluster para atraer a los clientes a utilizar las tarjetas de crédito. **Los requisitos de pagos a plazos que se pueden ofrecer pueden ser en forma de pagos de viajes, electrodomésticos, dispositivos electrónicos, smartphones o ciertas marcas que son más demandadas por el público.**
- **Para el cluster 2**, los bancos pueden ofrecer tarjetas de crédito especiales para principiantes o estudiantes (tarjetas de nivel de entrada) que puedan carecer de un perfil crediticio extenso. **Esta tarjeta de crédito se puede utilizar para construir su crédito y aprender a utilizar la tarjeta de crédito de manera responsable.** Esta tarjeta puede incluir varias características, como **exención de tarifas**, **recompensas** por establecer una rutina de pagos puntuales, **barreras bajas** para convertirse en titular de la tarjeta y tasas de interés indulgentes. Además, los bancos pueden **ofrecer**

oportunidades para actualizar a nuevos productos y mejores términos y condiciones si el cliente paga consistentemente las facturas a tiempo. Además, los bancos pueden ofrecer recompensas de registro para que los clientes que no son usuarios de la tarjeta de crédito del banco puedan interesarse en registrarse.

- Dado que los clientes en **el cluster 4 tienden a realizar adelantos en efectivo**, los bancos pueden ofrecer tarjetas de crédito especiales con diversos beneficios. Estos beneficios pueden ser en forma de bajos o ningún cargo por adelantos en efectivo o administrativos, bajos intereses, antigüedad relativamente alta, etc. Además, los bancos también **pueden ofrecer programas bancarios distintos de las tarjetas de crédito, como programas de pay-later con colaboraciones de terceros o préstamos personales proporcionados por los bancos.**

In [ ]:

```
1
```