```
import numpy as np # algebra lineal\n
import pandas as pd # procesado de datos,ficheros CSV\n
import math

import matplotlib.pyplot as plt
plt.style.use("seaborn-whitegrid")

import seaborn as sns

from collections import Counter

import warnings
warnings.filterwarnings("ignore")


USA_Housing= pd.read_csv("USA_Housing.csv")

USA_Housing.columns

USA_Housing.head()

USA_Housing.describe()

USA_Housing.info()

def bar_plot(variable):

  var = USA_Housing[variable]
  varValue = var.value_counts()

  plt.figure(figsize = (9, 3))
  plt.bar(varValue.index, varValue)
  plt.xticks(varValue.index, varValue.index.values)
  plt.ylabel("Frecuencia")
  plt.title(variable)
  plt.show()
  print("{}: \n {}".format(variable, varValue))

category1 = ["Address"]
for c in category1:
  bar_plot(c)


category2 = ["Address"]
for c in category2:
  print ("{} \n".format(USA_Housing[c].value_counts()))

cats = ['Address']

def plotFrequency(cats):
  '''A plot for visualize categorical data, showing both absolute and relative frequencies'''

fig, axes = plt.subplotes(math.ceil(len(cats) / 3), 3, figsize=(20, 12))
axes = axes.flatten()

for ax, cat in zip(axes, cats):
  total = float(len(covid19_tweets[cat]))
  sns.countplot(covid19_tweets[cat], palette='plasma', ax=ax)

  for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2.,
           height + 10,
           '{:1.2f}%'.format((height / total) * 100),
           ha="center")
  plt.ylabel('Count', fontsize=15, weight='bold')
plotFrequency(cats)

def plotsurvival(cats, data):
  '''A plot for bivariate analysis.'''

  fig, axes = plt.subplots(math.ceil(len(cats) / 3), 3, figsize=(20, 12))
  axes = axes.flatten()

  for ax, cat in zip(axes, cats):
      if cat == 'Address':
          sns.countplot(USA_Housing[cat], palette='plasma', ax=ax)
      else:

          sns.countplot(x=cat,
                        data=data,
                        hue='Address',
                        palette='plasma',
                        ax=ax)
          ax.legend(title='Address?',
                    loc='upper right',
```

```
                    labels=['No', 'Yes'])

      plt.ylabel('Count', frontsize=15, weight='bold')

  plotsurvival(cats, USA_Housing)

def plot_hist(variable):
    plt.figure(figsize = (9,3))
    plt.hist(USA_Housing[variable], bins = 50)
    plt.xlabel(variable)
    plt.ylabel("Frecuencia")
    plt.title("Distribución variable {} con histograma".format(variable))
    plt.show()

numericVar = ["Avg. Area Income", "Avg. Area House Age", "Avg. Area Number of Rooms", "Avg. Area Number of Bedrooms", "Area Population",
"Price"]
for n in numericVar:
    plot_hist(n)

def plot_3chart(df, feature):
    import matplotlib.gridspec as gridspec
    from matplotlib.ticker import MaxNLocator
    from scipy.stats import norm
    from scipy import stats

    # Creating a customized chart. and giving in figsize and everything.

    fig = plt.figure(constrained_layout=True, figsize=(12, 8))

    # Creating a grid of 3 cols and 3 rows

    grid = gridspec.GridSpec(ncols=3, nrows=3, figure=fig)

    # Customizing the histogram grid.

    ax1 = fig.add_subplot(grid[0, :2])

    # Set the title.

    ax1.set_title('Histogram')

    # Plot the histogram.

    sns.distplot(df.loc[:, feature],
                 hist=True,
                 kde=True,
                 fit=norm,
                 ax=ax1,
                 color='#e74c3c')
    ax1.legend(labels=['Normal', 'Actual'])

    # Customizing the QQ_plot.

    ax2 = fig.add_subplot(grid[1, :2])

    # Set the title.

    ax2.set_title('Probability Plot')

    # Plotting the QQ_Plot.

    stats.probplot(df.loc[:, feature].fillna(np.mean(df.loc[:, feature])),
                   plot=ax2)
    ax2.get_lines()[0].set_markerfacecolor('#e74c3c')
    ax2.get_lines()[0].set_markersize(12.0)

    # Customizing the Box Plot.

    ax3 = fig.add_subplot(grid[:, 2])

    # Set title.

    ax3.set_title('Box Plot')

    #Plotting the box plot.

    sns.boxplot(df.loc[:, feature], orient='v', ax=ax3, color='#e74c3c')
    ax3.yaxis.set_major_locator(MaxNLocator(nbins=24))

    plt.suptitle(f'{feature}', fontsize=24)

plot_3chart(USA_Housing, 'Avg. Area House Age')
plot_3chart(USA_Housing, 'Avg. Area Income')

USA_Housing.info()

# Address vs Address
```

```
USA_Housing[["Address","Address"]].groupby(["Address"], as_index = False).mean().sort_values(by="Address",ascending = False)

# Address vs Address
USA_Housing[["Address","Address"]].groupby(["Address"], as_index = False).mean().sort_values(by="Address",ascending = False))

# Address vs Address
USA_Housing[["Address","Address"]].groupby(["Address"], as_index = False).mean().sort_values(by="Address",ascending = False))

# Address vs Address
USA_Housing[["Address","Address"]].groupby(["Address"], as_index = False).mean().sort_values(by="Address",ascending = False))

def detect_outliners(df,features):
    outliner_indices = []

    for c in features:
        # 1st quartile
        Q1 = np.percentile(df[c],25)
        # 3rd quartile
        Q3 = np.percentile(df[c],75)
        # IQR
        IQR = Q3 - Q1
        # Outlier step
        outlier_step = IQR * 1.5
        # detect outlier and their indeces
        outlier_list_col = df[(df[c] < Q1 - outlier_step) | (df[c] > Q3 + outlier_step)].index
        # store indeces
        outlier_indices.extend(outlier_list_col)
    outliner_indices = Counter(outlier_indices)
    multiple_outliers = list(i for i, v in outlier_indices.items() if v > 2)

    return multiple_outliers

USA_Housing.loc[detect_outliners(USA_Housing,["Avg. Area House Age","Sibsip","Address","Avg. Area Income"])]

USA_Housing_len = len(USA_Housing)

USA_Housing.head()

sns.heatmap(USA_Housing.isnull(),
            yticklabels=False,
            cbar=False,
            cmap='magma')
plt.title('Valores perdidos en conjunto de train')

plt.xticks(rotation=90)
plt.show()

USA_Housing.columns[USA_Housing.isnull().any()]
Index(['Avg. Area House Age', 'Address', 'Address'], dtype='object')
USA_Housing.isnull().sum()

USA_Housing[USA_Housing["Address"].isnull()]

USA_Housing.boxplot(column="Avg. Area Income",by = "Address")
plt.show()

USA_Housing["Address"] = USA_Housing["Address"].fillna("c")
USA_Housing[USA_Housing["Address"].isnull()]

USA_Housing[USA_Housing["Avg. Area Income"].isnull()]

USA_Housing["Avg. Area Income"] = USA_Housing["Avg. Area Income"].fillna(np.meanUSA_Housing[USA_Housing["Address"] == 3]["Avg. Area Income"]))
USA_Housing[USA_Housing["Avg. Area Income"].isnull()]

corr = USA_Housing.corr()

f,ax = plt.subplots(figsize=(9,6))
sns.heatmap(corr, annot = True, linewidths=1.5 , fmt = '.2f',ax=ax)
plt.show()

g = sns.factorplot(x = "Address", y = "Address", data = USA_Housing, kind = "bar", size = 6)
g.set_ylabels("Probabilidad de supervivencia")
plt.show()

g = sns.factorplot(x = "Address", y = "Address",kind = "bar", data = USA_Housing, size = 6)
g.set_ylabels("Probabilidad de supervivencia")
plt.show()

g = sns.FacetGrid(USA_Housing, col = "Address")
g.map(sns.displot, "Avg. Area House Age", bins = 25)
plt.show()

g = sns.FacetGrid(USA_Housing, row = "Address", col = "Address", size = 2.3)
g.map(sns.barplot, "Address", "Avg. Area Income")
g.add_legend()
plt.show()
```

```
USA_Housing[USA_Housing["Avg. Area House Age"].isnull()]
sns.factorplot(x = "Address", y = "Avg. Area House Age", data = USA_Housing, kind = "box")
plt.show()

sns.factorplot(x = "Address", y = "Avg. Area House Age", hue = "Address", data = USA_Housing, kind = "box")
plt.show()

sns.factorplot(x = "Address", y = "Avg. Area House Age", data = USA_Housing, kind = "box")
sns.factorplot(x = "Address", y = "Avg. Area House Age", data = USA_Housing, kind = "box")
plt.show()

sns.heatmap(USA_Housing[["Avg. Area House Age","Address","Address","Address","Address"]].corr(),annot = True)
plt.show()

index_nan_Avg. Area House Age = list(USA_Housing["Avg. Area House Age"][USA_Housing["Avg. Area House Age"].isnull()].index)
for i in index_nan_Avg. Area House Age:
    Avg. Area House Age_pred = USA_Housing["Avg. Area House Age"][((USA_Housing["Address"] == USA_Housing.iloc[i]["Address"]) &(USA_Housing[
"Address"]== USA_Housing.iloc[i]["Address"]))]
    Avg. Area House Age_med = USA_Housing["Avg. Area House Age"].median()
    if not np.isnan(Avg. Area House Age_pred):
        USA_Housing["Avg. Area House Age"].iloc[i] = Avg. Area House Age_pred
    else:
        USA_Housing["Avg. Area House Age"].iloc[i] = Avg. Area House Age_pred

USA_Housing[USA_Housing["Avg. Area House Age"].isnull()]

USA_Housing["Title"]=USA_Housing['Address'].str.split(',',expand=True)[1].str.split('.',expand=True)[0].str.strip(' ')

plt.figure(figsize=(6, 5))
ax=sns.countplot(x= 'Title', data = USA_Housing, palette = "hls", order = USA_Housing['Title'].value_counts().index)
_= plt.xticks(
rotation=45,
horizontalalignment='right',
fontweight='light'
)
plt.title('Passengers distribution by titles', fontsize= 14)
plt.ylabel ('Number of passengers')
# calculate passengers for each category
labels = (train df['Title'].value counts())
# add result numbers on barchart
for i, v in enumerate (labels):
ax. text(i, v+10, str(v), horizontalalignment = 'center', size = 10, color = 'black')

plt.tight_layout()
plt.show()

category_Address = sns.catplot(x="Title_category", col="Address",
                data = USA_Housing, kind="count",
                height=4, aspect=7)
category_Address.set_xticklabels(rotation=45,
    horizontalalignment='right',
    fontweight='light')
plt.tight_layout()

USA_Housing['deck'] = USA_Housing['Address'].str.split('', expand = True)[1]
USA_Housing.loc[USA_Housing['deck'].isna(),'deck']='U'
print('Unique deck letters from the Address numbers:', USA_Housing['deck'].unique())

fig = plt.figure (figsize=(20, 5))

ax1= fig.add_subplot(131)
sns.countplot(x = 'deck', data = USA_Housing, palette ="hls", order = USA_Housing['deck'].value_counts().index, ax = ax1)
plt.title('Passengers distribution by deck', fontsize= 16)
plt.ylabel ('Number of passengers')

ax2= fig.add_subplot(132)
deck_by_class = USA_Housing.groupby('deck') ['Address'].value_counts (normalize = True).unstack()
deck_by_class.plot(kind='bar', stacked='True',color
= ['#eed4do','#cdalaa','#a2708e'], ax = ax2)
plt.legend(('1st class','2nd class','3rd class'), loc=(1.04,0))
plt.title('Proportion of classes on each deck', fontsize= 16)
plt.xticks(rotation = False)

ax3 = fig.add_subplot(133)
deck_by_Address = USA_Housing.groupby ('deck')['Address'].value_counts (normalize = True).unstack()
deck_by_Address = deck_by_Address.sort_values(by=1, ascending = False)
deck_by_Address.plot(kind='bar', stacked='True',color=["#3f3e6fd1", "#85c6a9"], ax = ax3)
plt.title('Proportion of Address/drowned passengers by deck', fontsize= 16)
plt.legend(( 'Drowned', 'Address' ), loc=(1.04,0) )
plt. xticks(rotation = False)
plt.tight_layout()
plt.show()

USA_Housing['Family_size'] = USA_Housing['Address'] + USA_Housing['Address'] + 1 family_size = USA_Housing['Family_size'].value_counts()
print('Family size and number of passengers:')
print(family_size)
```

```
fig = plt.figure (figsize = (12,4))
ax1= fig.add_subplot(121)
ax = sns.countplot (USA_Housing['Family_size'], ax = ax1)
# calculate passengers for each category
labels = (USA_Housing['Family_size' ].value_counts())
# add result numbers on barchart
for i, v in enumerate(labels):
ax.text(i, v+6, str(v), horizontalalignment = 'center', size = 10, color = 'black')
plt.title('Passengers distribution by family size')
plt.ylabel('Number of passengers')


ax2= fig.add_subplot(122)
d = USA_Housing.groupby('Family_size')['Address'].value_counts(normalize=True).unstack()
d.plot (kind=' bar', color=["#3f3e6fd1","#85c6a9"], stacked='True', ax = ax2)
plt.title('Proportion of Address/drowned passengers by family size (train data)')
plt.legend(('Drowned'
'Address' ), loc=(1.04,0) )
plt.xticks (rotation = False)
plt.tight_layout()

ax = sns.countplot(USA_Housing['Address'], palette = ['#eed4d0','#cda0aa','#a2708e'])
# cañculate passengers for each category
labels = (USA_Housing['Address'].value_counts(sort = False))
# add result numbers on barchart
for i, v in enumerate(labels):
    ax.text(i, v+2, str(v), horizontalalignment = 'center', size = 12, color = 'black', fontweight= 'bold')

plt.title('Passengers distribution by family size')
plt.ylabel('Number of passengers')
plt.tight_layout()

fig = plt.figure (figsize=(14, 5))
ax1 = fig.add_subplot(121)
sns.countplot (x = 'Address', hue = 'Address', data = USA_Housing, palette=["#3f3e6fd1", "#85c6a9"], ax=ax1)
plt.title('Number of Address/drowned passengers by class (train data)')
plt.ylabel ('Number of passengers')
plt.legend(("Drowned',

'Address' ), loc=(1.04,0) )
_= plt.xt cks(rotati n=False)
"#85c6a9"], ax = ax1)
ax2= fig.add_subplot(122)
d= USA_Housing.groupby('Address')['Address'].value_counts(normalize=True).unstack()
d.plot (kind='bar', stacked='True', ax = ax2, color =["#3f3e6fd1","#85¢6a9"])
plt.title('Proportion of Address/drowned passengers by class (train data) ')
plt.legend(("Drowned',

'Address'),loc=(1.04,0))
= plt.xticks(rotation=False)
plt.tight_layout()

fig = plt. figure(figsize = (15,4))
ax1= fig.add_subplot(131)
palette = sns.cubehelix_palette(5, start = 2)
ax = sns.countplot(USA_Housing['Address'], palette = palette, order = ['C','Q','S'], ax = ax1)
plt.title('Number of passengers by Address")

plt.ylabel('Number of passengers')

# calculate passengers for each category
labels=(USA_Housing['Address'].value_counts())
labels = labels.sort_index()

" dd result numbers on barchart
for i, v in enumerate (labels):
ax.text(i, v+10, str(v), horizontalalignment = 'center', size = 10, color = 'black')

ax2= fig.add_subplot(132)
surv_by_emb = USA_Housing.groupby("Address")["Address").value_counts(normalize=True)
surv_by_emb = surv_by_emb.unstack().sort_index()
surv_by_emb.plot(kind='bar',stacked='True',color=["#3f3e6fd1","#85c6a9"],ax=ax2)
plt.title( 'Proportion of Address/drowned passengers by Address (train data)')
plt.legend(('Drowned',
'Address' ), loc=(1.04,0) )
_ = plt.xticks(rotation=False)

ax3= fig.add_subplot(133)
class_by_emb=USA_Housing.groupby('Address')['Address'].value_counts(normalize=True)
class_by_emb=class_by_emb.unstack().sort_index()
class_by_emb.plot(kind='bar',stacked='True',color=["#eed4d0","#cd0aa","#a2708e"], ax = ax3)
plt.legend (('1st class','2nd class','3rd class'),
loc=(1.04,0)
_ = plt.xticks(rotation=False)

plt.tight_layout()
```

```
sns.catplot(x="Address",y="Avg. Area Income", kind="swarm", data=USA_Housing, palette=sns.cubehelix_palette(5, start = 3), height = 6)

plt.tight_layout()

sns.catplot(x="Address", y="Avg. Area Income", hue ="Address", kind="swarm", data=USA_Housing,
                                palette=["#3f3e6fd1","#85c6a9"], height = 6)
plt.tight_layout()

from sklearn.decomposition import PCA
from sklearn import preprocessing

USA_Housing= pd.read_csv('USA_Housing.csv', dtype={'Avg. Area House Age': np.float16})
USA_Housing.head()

nans = {}
for colAddress in USA_Housing.columns:
    nans[colAddress] = USA_Housing[USA_Housing[colAddress].isnull()].size
nans

old_train_data = USA_Housing.copy()
USA_Housing.drop('Address',1, inplace=True)
USA_Housing= USA_Housing[USA_Housing['Avg. Area House Age'].notnull()]
USA_Housing.size

USA_Housing['Child'] = USA_Housing.apply(lambda row['Address']== 'male' else 1, axis=1)
train_X = USA_Housing[USA_Housing].values
train_Y = USA_Housing.Address.Values
USA_Housing[train_features].head()

my_pca = PCA(n_components=2)
preprocessed_train = preprocessing.normalize(preprocessing.scale(train_X))
print(preprocessed_train.shape)
#missing feature scaling and normalization
my_pca.fit(preprocessed_train)
trans = my_pca.transform(preprocessed_train)

fig, axs = plt.subplots(3, 2, squeeze=False, sharex=True, sharey=True, figsize=(12, 18))
axs[0, 0].plot(trans[:,0], trans[:,1], '.')
axs[0, 0].set_title('PCA: Basic')
female_trans = np.array([tran for is_female, tran in zip(USA_Housing['Address_number'], trans) if is_female==1])
axs[0,1].plot(trans[:,e],trans[:,1], '.', label='Male')
axs[0, 1].plot (female_trans[:,0], female_trans[:,1], 'r.', label='Female')
axs[0,1].set_title('Address')
axs[0,1].legend()

child_trans = np.array([tran for is_child, tran in zip(USA_Housing['Child'], trans) if is_child==1])
axs[1, 0].plot(trans[:,0], trans[:,1], '.', label='Adult')
axs[1, 0].plot(child_trans[:,0], child_trans[:,1], 'r.', label='Child')
axs[1, ].set title("child/Adulte)

axs[1, 0].legend()
I
third_trans = p.array([tran for my_class, tran in zip(USA_Housing['Address'], trans) if my_class==3])
second_trans = p.array([tran for my_class, tran in zip(USA_Housing['Address'], trans) if my_class==2])
axs[1, 1].plot(trans[:,e], trans[:,1],
, label='First')
axs[1,1].plot(third_trans[:,0],
third_trans[:,1], '.r',label='Second')
axs[1,1].plot(second_trans[:,e],second_trans[:,1],
'.g', label='Third')
axs[1,1].set_title('Class')
axs[1,1].legend()
axs[2, 0].scatter(trans[:,e], trans[:,1], edgecolors='face', c=USA_Housing['Avg. Area House Age'].values)

axs[2, 0]. set title('Avg. Area House Age')
axs[2, 1].scatter(trans[:,0], trans[:,1], edgecolors='face', c=USA_Housing['Avg. Area Income' ].values)
axs[2, 1].set_title('Avg. Area Income')

from numpy import unique
from numpy import whe
from sklearn.cluster import KMeans
from sklearn.mixture umport GaussianMixture
from matplotlib import pyplot
# define the model
model = KMeans(n_clusters=2)
#model = GaussianMixture(n_clusters=2)
# fit the model
model.fit(trans)
# assign a cluster to each example
yhat = model.predict(trans)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
    #get row indexes for samples with this cluster
    row_ix = where(yhat==cluster)
    #create scatter of these samples
```

```
    pyplot.scatter(trans[row_ix, 0], trans[row_ix, 1])
pyplot.show()
```