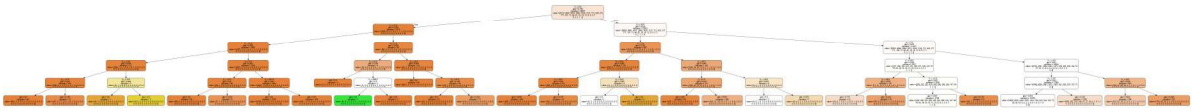
In [39]:
```
y_train_class.shape
```

Out[39]:
```
(25607,)
```

In [40]:
```
# Se define el modelo Decision Tree simple
model_tree = DecisionTreeClassifier(max_depth=5, min_samples_split=2)

# Entrena el modelo para el conjunto train reducido a 2 dimensiones
model_tree.fit(x_train, y_train_class)
```

Out[40]:
```
DecisionTreeClassifier(max_depth=5)
```

In [41]:
```
# Representación del árbol de decisión.
dot_data = StringIO()
export_graphviz(model_tree, out_file = dot_data,
                filled = True, rounded = True,
                special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[41]:


In [42]:
```
# Precisión sobre train y test
score_train = model_tree.score(x_train, y_train_class)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rou
score_test = model_tree.score(x_test, y_test_class)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Matriz de confusión sobre train
y_predict_train_6 = model_tree.predict(x_train)
y_predict_test_6 = model_tree.predict(x_test)
```

```
El grado de precisión del modelo para el conjunto train es 0.3711.
El grado de precisión del modelo para el conjunto test es 0.3326.
```

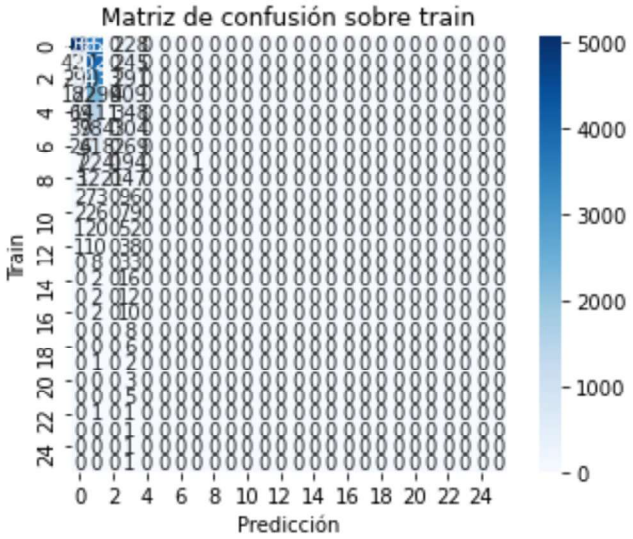
In [43]:
```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train_6, y_train_class))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train_6, y_train_cla
print('MAE in test:', mean_absolute_error(y_predict_test_6, y_test_class))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test_6, y_test_class)
```

```
MAE in train: 1.3684539383762253
RMSE in train: 2.259738085335643
MAE in test: 1.5518177704027756
RMSE in test: 2.432835917586868
```

In [44]:
```
matrix_train = confusion_matrix(y_train_class, y_predict_train_6)
sns.heatmap(matrix_train, annot=True, fmt="d", cmap='Blues', square=True)
plt.xlabel("Predicción")
plt.ylabel("Train")
plt.title('Matriz de confusión sobre train')
# plt.savefig('graph/matriz confusion arbol decision.jpg')
plt
```

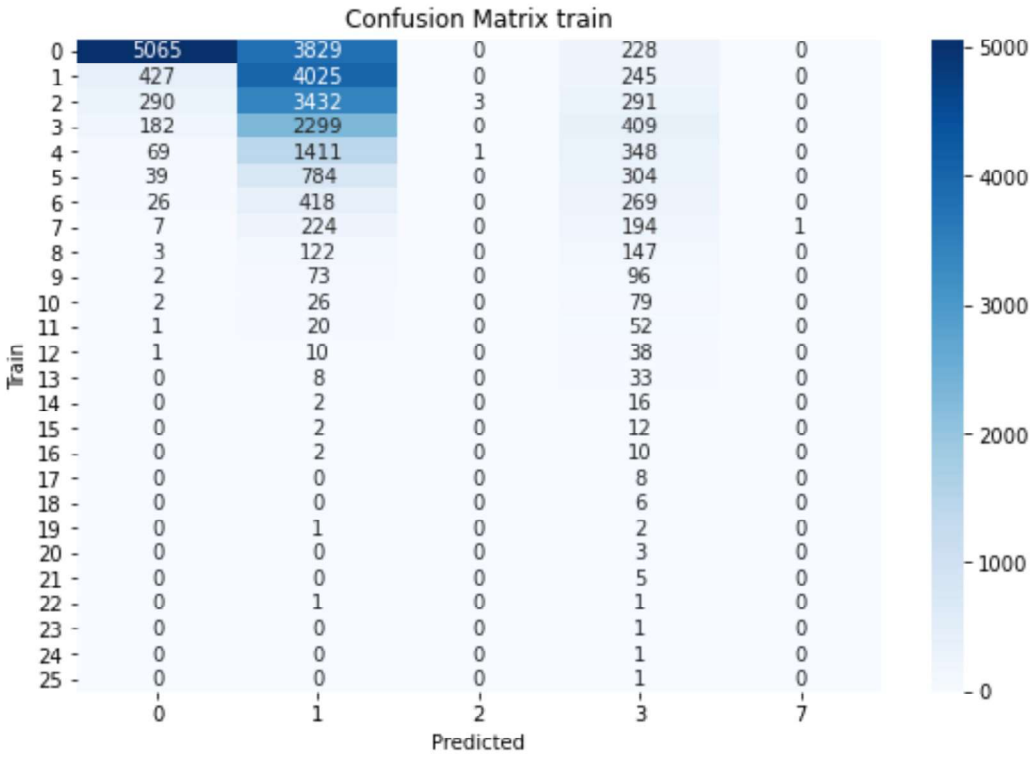
Out[44]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\jh100\\Anaconda3\\lib\\site-package
```

s\\matplotlib\\pyplot.py'>



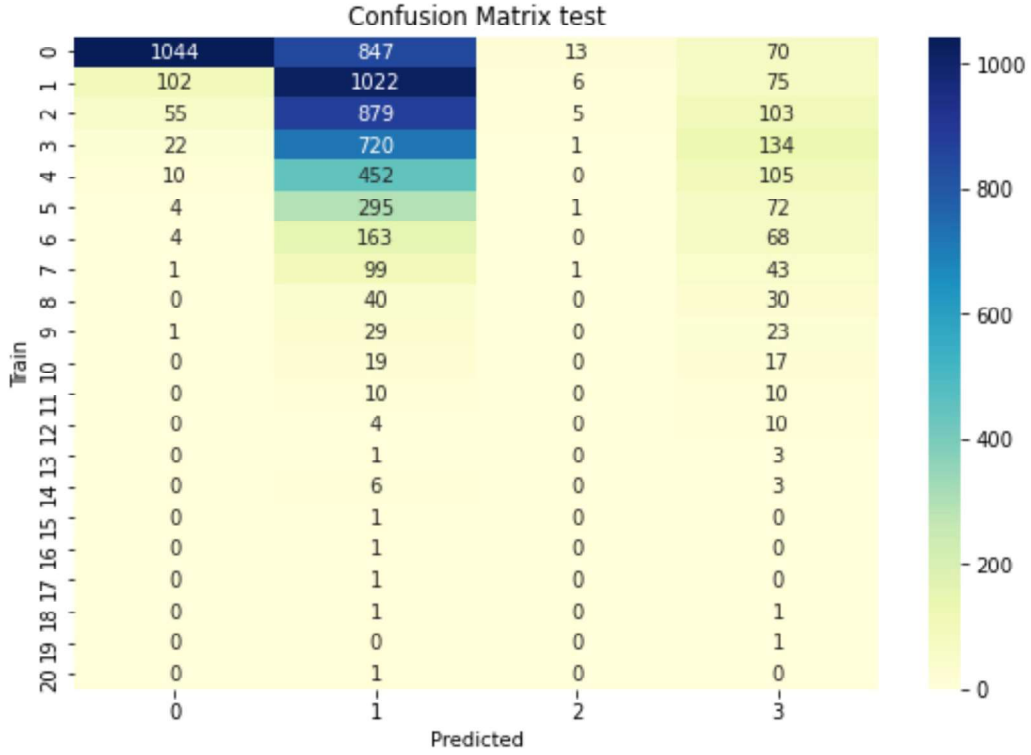
In [45]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_class, y_predict_train_6, rownames=[
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_DECISONTREEtrain_class.jpg')
plt.show()
```



In [46]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_class, y_predict_test_6, rownames=['Tr
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_DECISONTREEtest_class.jpg')
plt.show()
```



In [47]:

```
y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_Tree_class = y_predict_test_6.astype(int)
y_predict['test_preds_Tree_4'] = test_preds_Tree_class
y_predict.to_excel('data/y_predictsvm.xls')
```

3.4.3.2 Árbol decisión variable *udsVenta* categorizada en 7 intervalos de igual tamaño.

Previo al algoritmo, se obtienen mejores parámetros para configuración del modelo.

In [48]:

```
# Definición de los conjuntos parámetro
max_depth = [4, 5, 6, 7, 8, 9, 10]
min_samples_split = [2, 10, 20, 50, 100]

grid_tree = dict(max_depth=max_depth, min_samples_split=min_samples_split)
cv =  StratifiedKFold(n_splits=5, shuffle=True, random_state=15)

# Optimizar
tree_optimizer = GridSearchCV(estimator=DecisionTreeClassifier(),
                              param_grid=grid_tree,
                              n_jobs=-1,
                              scoring='accuracy',
                              refit = True,
                              cv=cv,
                              error_score=0)

grid_tree_optimizer = tree_optimizer.fit(x_train, y_train_2)

# Obtener la mejor combinación del clasificador SVM
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.fo
print ('Clases de clasificación del modelo {}.'.format(grid_tree_optimizer.classe

# Obtenemos la precisión del modelo
print('Precisión del modelo {}.'.format(round(grid_tree_optimizer.score(x_train, y
```

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 2 members, which is less than
n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
```

```
La mejor configuración de parámetros que maximiza la precisión es {'max_depth': 6,
'min_samples_split': 100}.
Clases de clasificación del modelo [0 1 2 3 4 5].
Precisión del modelo 0.9274.
```

In [49]:

```
# Se define el modelo Decision Tree simple para los mejores parámetros
model_tree2 = DecisionTreeClassifier(max_depth=8, min_samples_split=100)

# Entrena el modelo para el conjunto train.
model_tree2.fit(x_train, y_train_2)
```

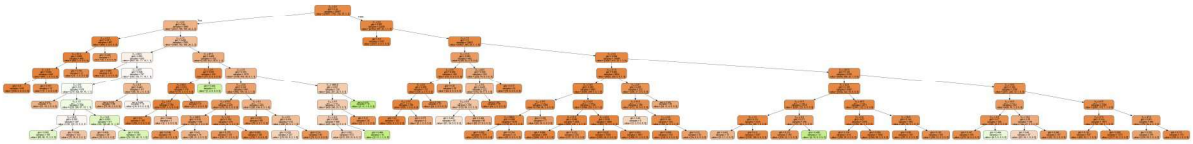
Out[49]:

```
DecisionTreeClassifier(max_depth=8, min_samples_split=100)
```

In [50]:

```
# Representación del árbol de decisión.
dot_data = StringIO()
export_graphviz(model_tree2, out_file = dot_data,
                filled = True, rounded = True,
                special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[50]:



In [51]:

```
# Precisión sobre train y test
score_train = model_tree2.score(x_train, y_train_2)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rou
score_test = model_tree2.score(x_test, y_test_2)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Predicciones sobre conjunto train y test
y_predict_train_2 = model_tree2.predict(x_train)
y_predict_test_2 = model_tree2.predict(x_test)
```

```
El grado de precisión del modelo para el conjunto train es 0.9299.
El grado de precisión del modelo para el conjunto test es 0.903.
```

In [52]:

```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train_2, y_train_2))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train_2, y_train_2)))
print('MAE in test:', mean_absolute_error(y_predict_test_2, y_test_2))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test_2, y_test_2)))
```

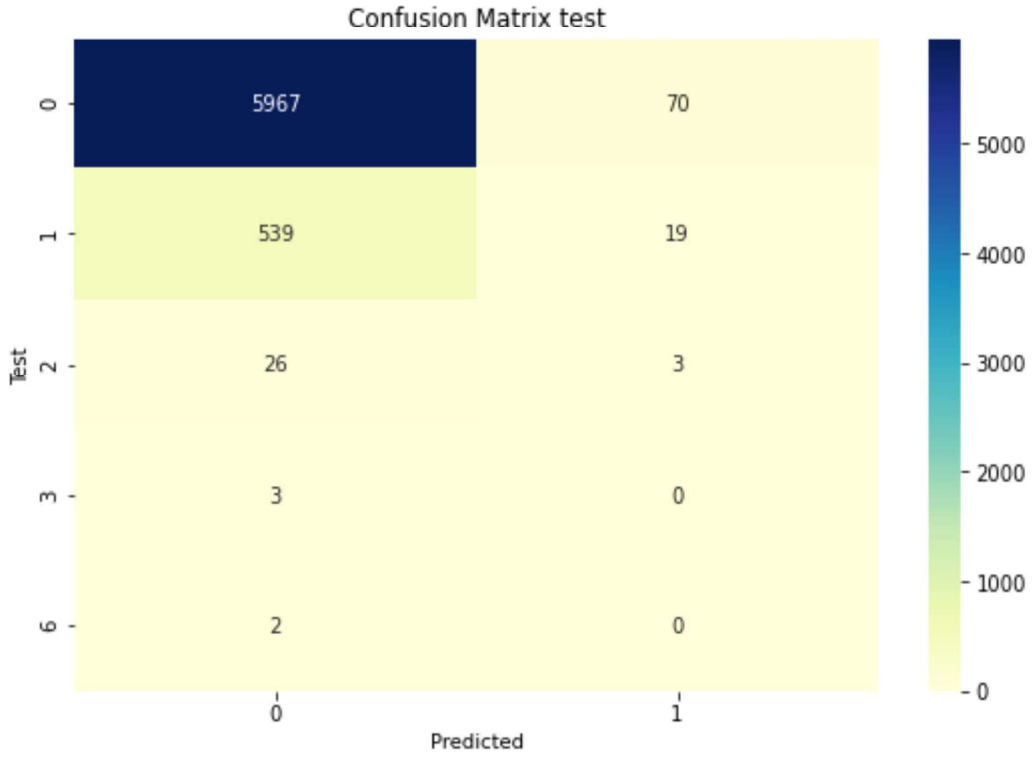
```
MAE in train: 0.07497949779357208
RMSE in train: 0.29497026782026287
MAE in test: 0.10333383617438528
RMSE in test: 0.35063462065258333
```

In [53]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_2, y_predict_train_2, rownames=['Tra
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_DECISONTREEtrain_2.jpg')
plt.show()
```

In [54]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_2, y_predict_test_2, rownames=['Test'
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_DECISONTREEtest_2.jpg')
plt.show()
```



In [55]:
```
y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_Tree_2 = y_predict_test_2.astype(int)
y_predict['test_preds_Tree_4'] = test_preds_Tree_2
y_predict.to_excel('data/y_predictsvm.xls')
```

3.4.3.3 Árbol decisión, variable *udsVenta* categorizada en 25 clases.

Se aplica algoritmo al modelo con variable udsVenta categorizada por clases. No se aplica filtro para sólo días de apertura.

In [56]:
```
# Optimizar
tree_optimizer = GridSearchCV(estimator=DecisionTreeClassifier(),
                              param_grid=grid_tree,
                              n_jobs=-1,
                              scoring='accuracy',
                              refit = True,
                              cv=cv,
                              error_score=0)

grid_tree_optimizer = tree_optimizer.fit(x_train, y_train_3)

# Obtener la mejor combinación del clasificador SVM
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.for
print ('Clases de clasificación del modelo {}.'.format(grid_tree_optimizer.classes

# Obtenemos la precisión del modelo
print('Precisión del modelo {}.'.format(round(grid_tree_optimizer.score(x_train, y
```

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 1 members, which is less than
n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
La mejor configuración de parámetros que maximiza la precisión es {'max_depth': 1
0, 'min_samples_split': 100}.
Clases de clasificación del modelo [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 1
5 16 17 18 19 20 21 22 23
```

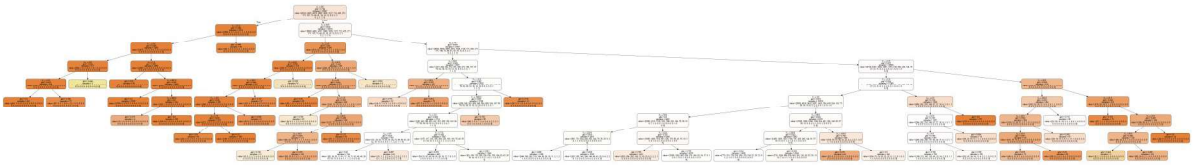
```
 24 25].
Precisión del modelo 0.3943.
```

In [57]:
```
# Se define el modelo Decision Tree simple para los mejores parámetros
model_tree3 = DecisionTreeClassifier(max_depth=8, min_samples_split=100)

# Entrena el modelo para el conjunto train reducido a días abierto
model_tree3.fit(x_train, y_train_3)
```

Out[57]:
```
DecisionTreeClassifier(max_depth=8, min_samples_split=100)
```

In [58]:
```
# Representación del árbol de decisión.
dot_data = StringIO()
export_graphviz(model_tree3, out_file = dot_data,
                filled = True, rounded = True,
                special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[58]:



In [59]:
```
# Precisión sobre train y test
score_train = model_tree3.score(x_train, y_train_3)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rour
score_test = model_tree3.score(x_test, y_test_3)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Predicciones sobre conjunto train y test
y_predict_train_3 = model_tree3.predict(x_train)
y_predict_test_3 = model_tree3.predict(x_test)
```

```
El grado de precisión del modelo para el conjunto train es 0.3844.
El grado de precisión del modelo para el conjunto test es 0.305.
```

In [60]:
```
c = cross_val_score(model_tree3, x_train, y_train_3, cv=6).mean()
print('La precisión media del modelo aplicado al conjunto train, es:',round(c*100,
```

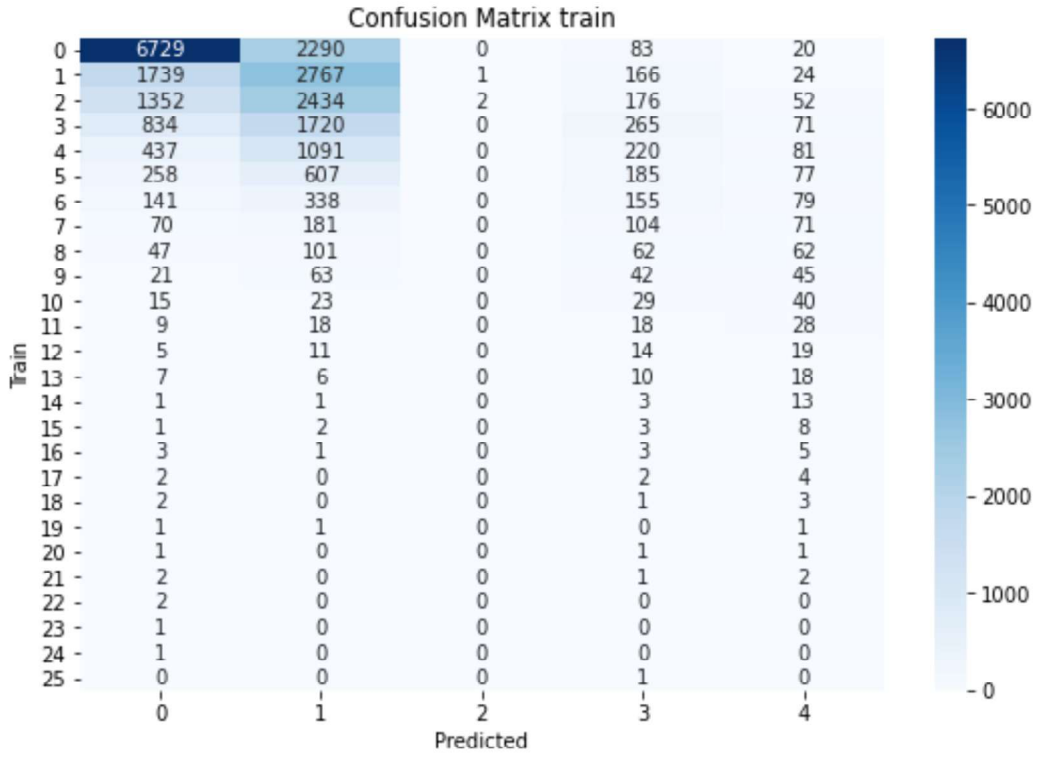
```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 1 members, which is less than
n_splits=6.
  % (min_groups, self.n_splits)), UserWarning)
La precisión media del modelo aplicado al conjunto train, es: 24.44 %
```

In [61]:
```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train_3, y_train_3))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train_3, y_train_3)))
print('MAE in test:', mean_absolute_error(y_predict_test_3, y_test_3))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test_3, y_test_3)))
```

```
MAE in train: 1.4494474167219902
RMSE in train: 2.3807194719340137
MAE in test: 1.768441695580027
RMSE in test: 2.683680700620191
```

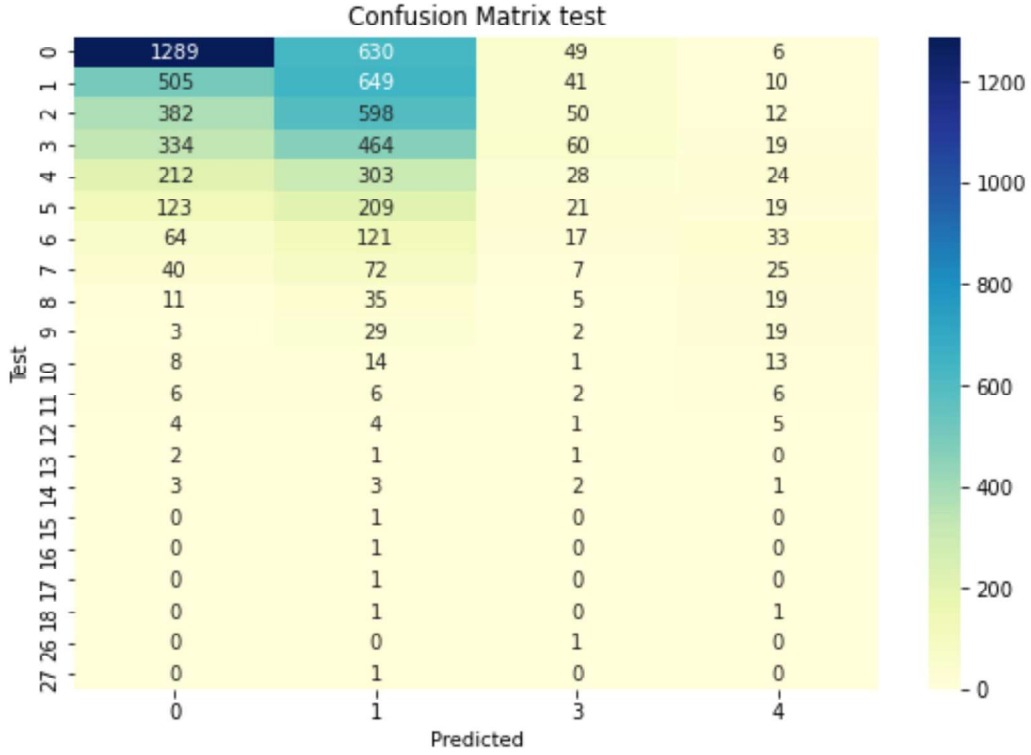
In [62]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_3, y_predict_train_3, rownames=['Tra
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_DECISONTREEtrain_3.jpg')
plt.show()
```



In [63]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_3, y_predict_test_3, rownames=['Test'
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_DECISONTREEtest_3.jpg')
plt.show()
```



In [64]:
```
y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_Tree_3 = y_predict_test_3.astype(int)
y_predict['test_preds_Tree_3'] = test_preds_Tree_3
y_predict.to_excel('data/y_predictsvm.xls')
```

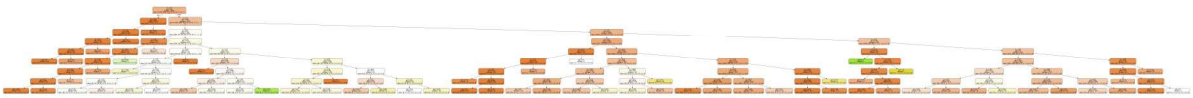
3.4.3.4 Árbol decisión para la modelo con variable *udsVenta* categorizada en 14 intervalos de igual tamaño.

In [65]:
```
# Se define el modelo Decision Tree simple para los mejores parámetros
model_tree4 = DecisionTreeClassifier(max_depth=8, min_samples_split=100)

# Entrena el modelo para el conjunto train reducido a días abierto
model_tree4.fit(x_train, y_train_4)
```

Out[65]:
```
DecisionTreeClassifier(max_depth=8, min_samples_split=100)
```

In [66]:
```
# Representación del árbol de decisión.
dot_data = StringIO()
export_graphviz(model_tree4, out_file = dot_data,
                filled = True, rounded = True,
                special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[66]:



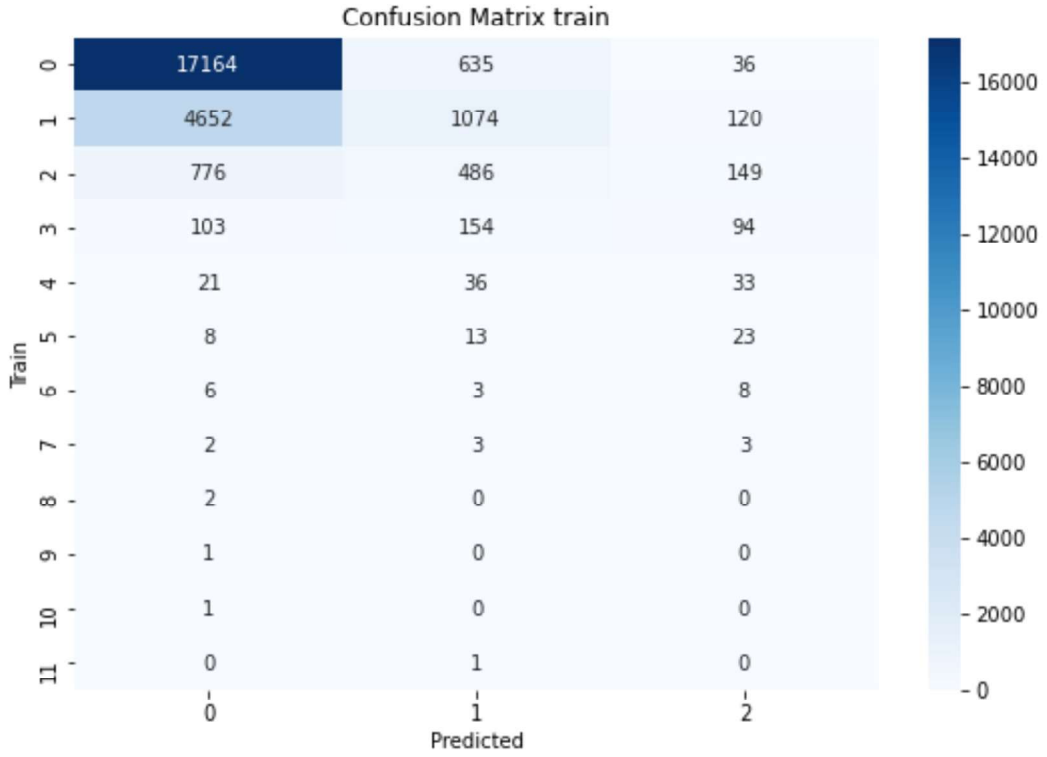
In [67]:
```
# Precisión sobre train y test
score_train = model_tree4.score(x_train, y_train_4)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rou
score_test = model_tree4.score(x_test, y_test_4)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Predicciones sobre conjunto train y test
y_predict_train_4 = model_tree4.predict(x_train)
y_predict_test_4 = model_tree4.predict(x_test)
```

```
El grado de precisión del modelo para el conjunto train es 0.718.
El grado de precisión del modelo para el conjunto test es 0.6245.
```

In [68]:
```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train_4, y_train_4))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train_4, y_train_4)))
print('MAE in test:', mean_absolute_error(y_predict_test_4, y_test_4))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test_4, y_test_4)))
```

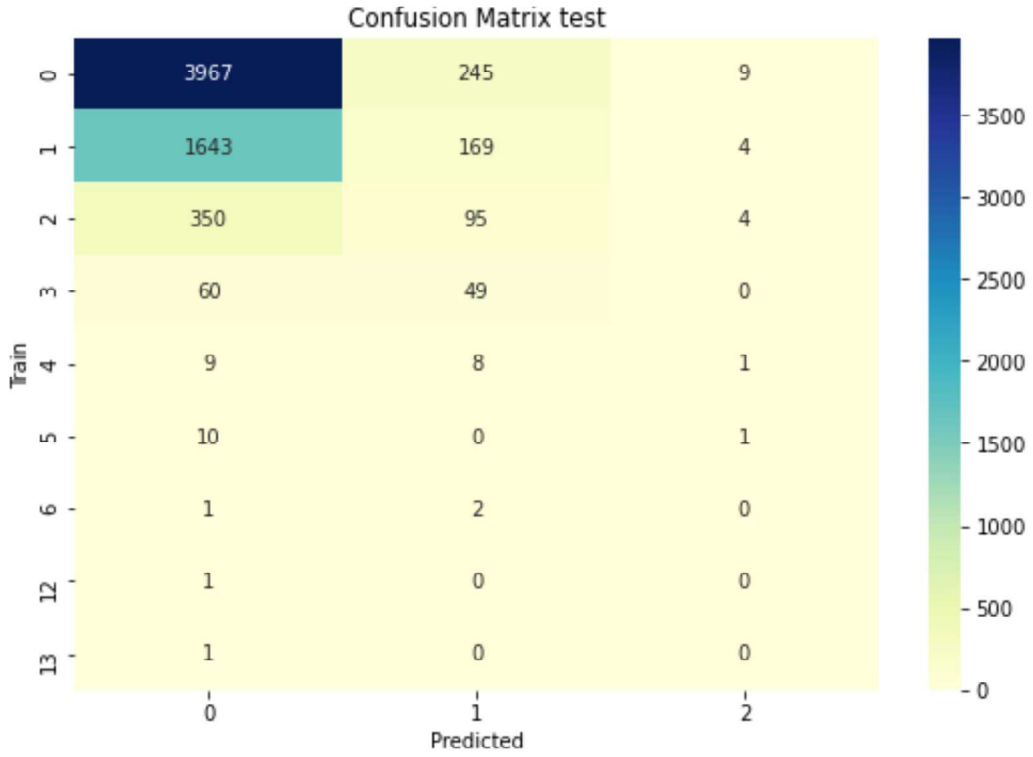
```
MAE in train: 0.3445151716327567
RMSE in train: 0.7205240582915685
MAE in test: 0.4735254186151757
RMSE in test: 0.8716898035378146
```

In [69]:
```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_4, y_predict_train_4, rownames=['Tra
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_DECISONTREEtrain_4.jpg')
plt.show()
```



In [70]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_4, y_predict_test_4, rownames=['Train
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_DECISONTREEtest_4.jpg')
plt.show()
```



In [71]:

```
y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_Tree_4 = y_predict_test_4.astype(int)
y_predict['test_preds_Tree_4'] = test_preds_Tree_4
y_predict.to_excel('data/y_predictsvm.xls')
```

3.4.3.5 Árbol de decisión para la variable *udsVenta* categorizada en 7 intervalos y bolOpen=1 (abierto).

Por último, se prueba nuevamente para la variable categórica y_train2_2;

- udsVenta discretizada en intervalos de igual amplitud.
- sólo días abiertos (bolOpen = 1).

In [72]:
```
# Optimizar
tree_optimizer = GridSearchCV(estimator=DecisionTreeClassifier(),
                              param_grid=grid_tree,
                              n_jobs=-1,
                              scoring='accuracy',
                              refit = True,
                              cv=cv,
                              error_score=0)

grid_tree_optimizer = tree_optimizer.fit(x_train2, y_train2_2)

# Obtener la mejor combinación del clasificador.
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.fo
print ('Clases de clasificación del modelo {}.'.format(grid_tree_optimizer.classe

# Obtenemos la precisión del modelo
print('Precisión del modelo {}.'.format(round(grid_tree_optimizer.score(x_train2,
```

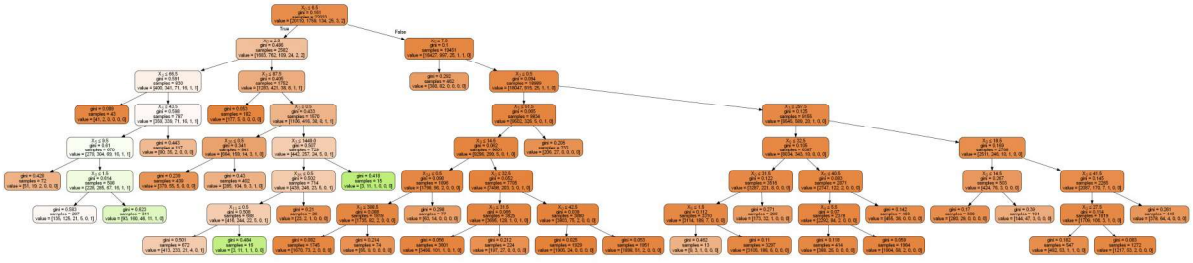
```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 2 members, which is less than
n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
La mejor configuración de parámetros que maximiza la precisión es {'max_depth': 9,
'min_samples_split': 100}.
Clases de clasificación del modelo [0 1 2 3 4 5].
Precisión del modelo 0.9201.
```

In [73]:
```
# Se define el modelo Decision Tree simple para los mejores parámetros
model_tree6 = DecisionTreeClassifier(max_depth=7, min_samples_split=500)

# Entrena el modelo para el conjunto train reducido a días abierto
model_tree6.fit(x_train2, y_train2_2)
```

Out[73]:
```
DecisionTreeClassifier(max_depth=7, min_samples_split=500)
```

In [74]:
```
# Representación del árbol de decisión.
dot_data = StringIO()
export_graphviz(model_tree6, out_file = dot_data,
                filled = True, rounded = True,
                special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[74]:



In [75]:
```
# Precisión sobre train y test
score_train = model_tree6.score(x_train2, y_train2_2)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rou
score_test = model_tree6.score(x_test2, y_test2_2)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Predicciones sobre conjunto train y test
y_predict_train6 = model_tree6.predict(x_train2)
y_predict_test6 = model_tree6.predict(x_test2)
```

```
El grado de precisión del modelo para el conjunto train es 0.9165.
El grado de precisión del modelo para el conjunto test es 0.894.
```

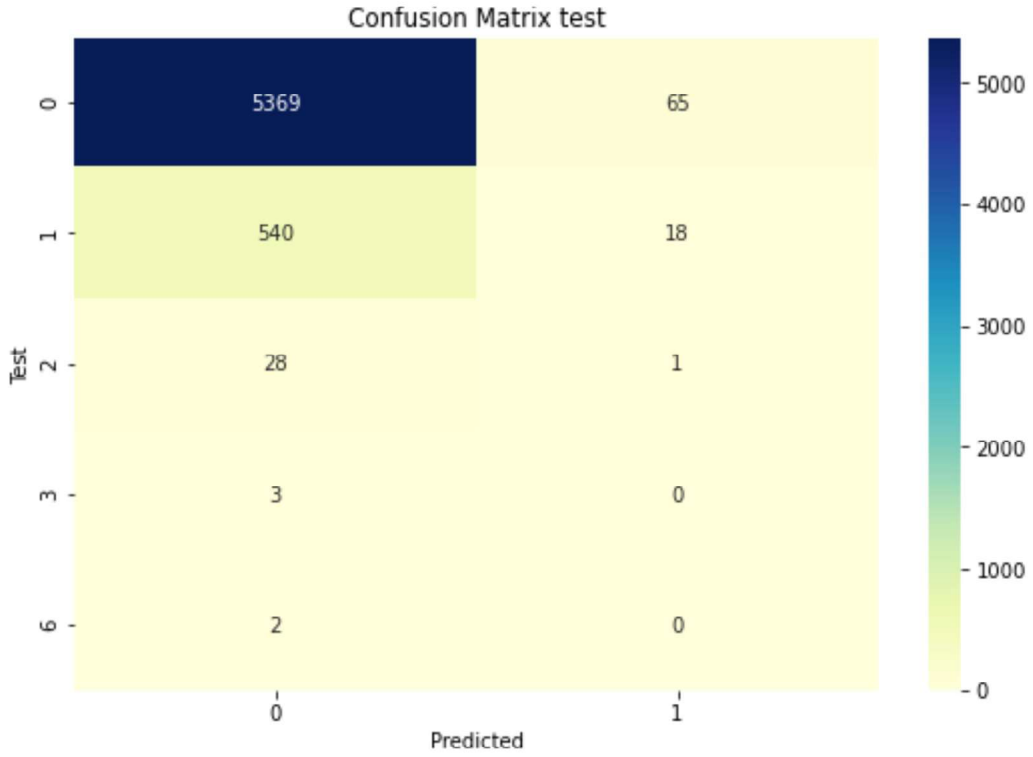
In [76]:
```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train6, y_train2_2))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train6, y_train2_2)))
print('MAE in test:', mean_absolute_error(y_predict_test6, y_test2_2))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test6, y_test2_2)))
```

```
MAE in train: 0.08986520219670495
RMSE in train: 0.3247739928269173
MAE in test: 0.11334218386989711
RMSE in test: 0.36821075077514004
```

In [77]:
```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train2_2, y_predict_train6, rownames=['Tra
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_DECISONTREEtrain2_2.jpg')
plt.show()
```

In [78]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test2_2, y_predict_test6, rownames=['Test'
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_DECISONTREEtest2_2.jpg')
plt.show()
```



3.4.3.6 Árbol decisión, para la variable *udsVenta* categorizada en 24 clases y para días bolOpen = 1.

Veamos a continuación el mejor modelo de árbol decisión, ahora sólo para días en que el establecimiento se encuentra abierto.

In [79]:
```
# Definición de los conjuntos parámetro
max_depth = [4, 5, 6, 7, 8, 9, 10]
min_samples_split = [20, 50, 100, 500]

grid_tree = dict(max_depth=max_depth, min_samples_split=min_samples_split)
cv =  StratifiedKFold(n_splits=5, shuffle=True, random_state=15)

# Optimizar
tree_optimizer = GridSearchCV(estimator=DecisionTreeClassifier(),
                              param_grid=grid_tree,
                              n_jobs=-1,
                              scoring='accuracy',
                              refit = True,
                              cv=cv,
                              error_score=0)

grid_tree_optimizer = tree_optimizer.fit(x_train2, y_train2_3)

# Obtener la mejor combinación del clasificador SVM
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.fo
print ('Clases de clasificación del modelo {}.'.format(grid_tree_optimizer.classe

# Obtenemos la precisión del modelo
print('Precisión del modelo {}.'.format(round(grid_tree_optimizer.score(x_train2,
```

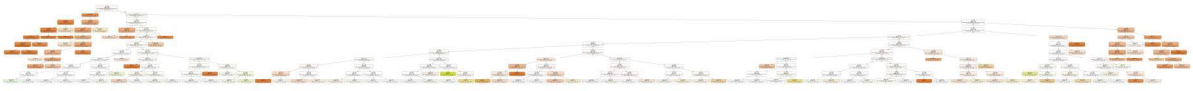
```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 1 members, which is less than
n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
La mejor configuración de parámetros que maximiza la precisión es {'max_depth': 9,
'min_samples_split': 50}.
Clases de clasificación del modelo [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 1
5 16 17 18 19 20 21 22 23
 24 25].
Precisión del modelo 0.2987.
```

In [80]:
```
# Se define el modelo Decision Tree simple para los mejores parámetros
model_tree5 = DecisionTreeClassifier(max_depth=10, min_samples_split=100)

# Entrena el modelo para el conjunto train reducido a días abierto
model_tree5.fit(x_train2, y_train2_3)
```

Out[80]:
```
DecisionTreeClassifier(max_depth=10, min_samples_split=100)
```

In [81]:
```
# Representación del árbol de decisión.
dot_data = StringIO()
export_graphviz(model_tree5, out_file = dot_data,
                filled = True, rounded = True,
                special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[81]:



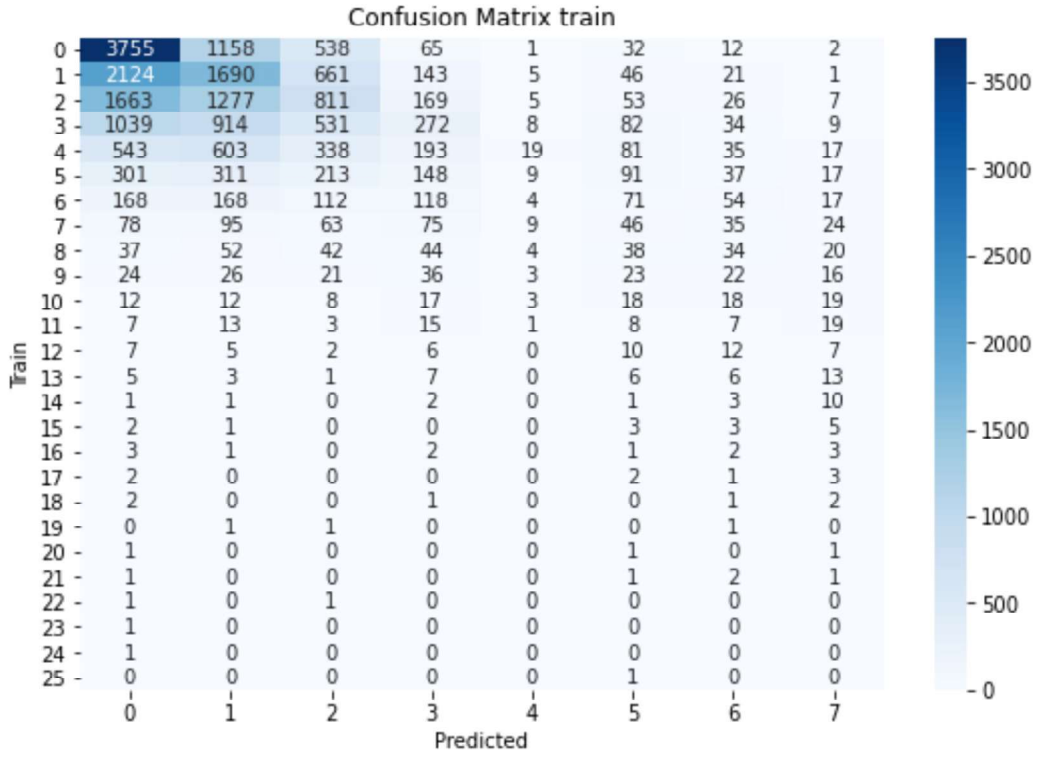
In [82]:
```
# Precisión sobre train y test
score_train = model_tree5.score(x_train2, y_train2_3)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rou
score_test = model_tree5.score(x_test2, y_test2_3)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Predicciones sobre conjunto train y test
y_predict_train5 = model_tree5.predict(x_train2)
y_predict_test5 = model_tree5.predict(x_test2)
```

```
El grado de precisión del modelo para el conjunto train es 0.3048.
El grado de precisión del modelo para el conjunto test es 0.2355.
```

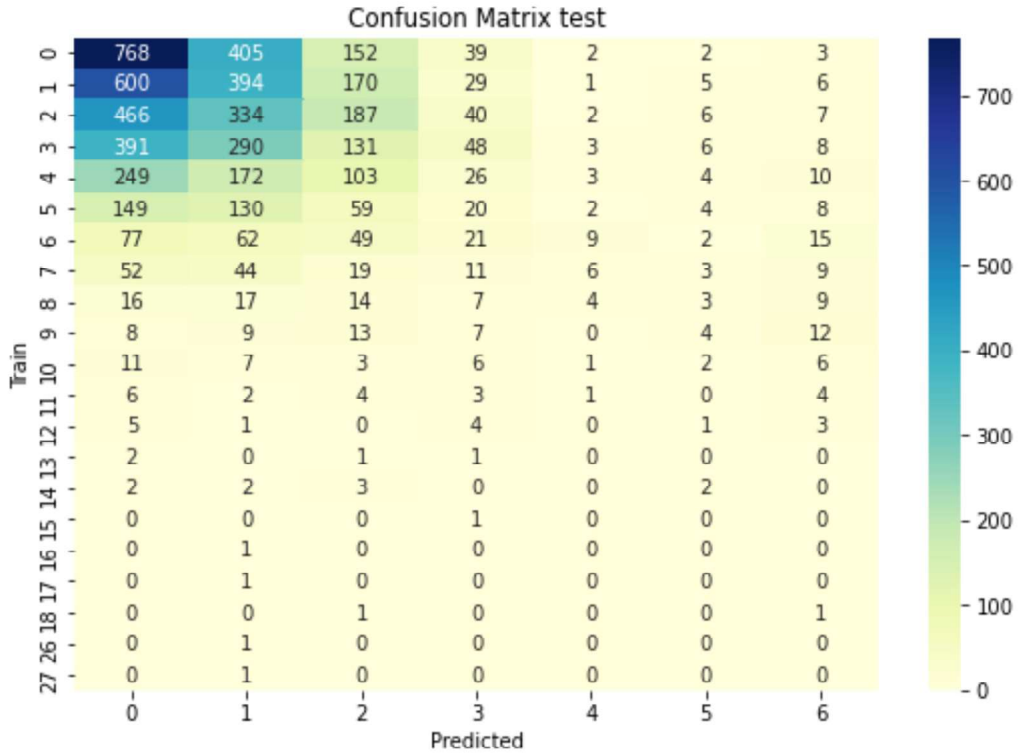
In [83]:
```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train5, y_train2_3))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train5, y_train2_3)))
print('MAE in test:', mean_absolute_error(y_predict_test5, y_test2_3))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test5, y_test2_3)))
```

```
MAE in train: 1.6127172877048064
RMSE in train: 2.4367550462548846
MAE in test: 1.946896780617325
RMSE in test: 2.793834186026607
```

In [84]:
```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train2_3, y_predict_train5, rownames=['Tra
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_DECISONTREEtrain2_3.jpg')
plt.show()
```



In [85]:
```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test2_3, y_predict_test5, rownames=['Train
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_DECISONTREEtest2_3.jpg')
plt.show()
```



Se han probado diferentes combinaciones, y con el que mejor porcentaje de precisión se obtiene es con la variable udsVenta categorizada en intervalos de igual amplitud y sólo para días bolOpen = 1.

3.4.4 Algoritmo Gradiente Boosting Regressor.

Este algoritmo combina algoritmo de árbol de decisión con regresión. Entre las diversas categorizaciones posibles, y activar el filtro de sólo dias abierto, la que mejor precisión ha dado en los anteriores algoritmos es;

- categorizar *udsVenta* en 7 intervalos de igual tamaño.
- considerar todos los días.

In [86]:
```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.model_selection import cross_val_score

model_GBR = DecisionTreeRegressor(max_depth = 10,
                                  criterion = 'mse')
c = cross_val_score(model_GBR, x_train, y_train_2, cv=6).mean()
print('La precisión media del modelo aplicado al conjunto train, es:',round(c*100,
```

```
La precisión media del modelo aplicado al conjunto train, es: -56.33 %
```

Se entrena el modelo para el conjunto completo y variable *udsVenta* discretizada en 7 intervalos de igual tamaño.

In [87]:
```
# Definición de los conjuntos parámetro
max_depth = [4, 5, 6, 7, 8, 9, 10]
min_samples_split = [20, 50, 100, 500]

grid_tree = dict(max_depth=max_depth, min_samples_split=min_samples_split)
cv =  StratifiedKFold(n_splits=5, shuffle=True, random_state=15)

# Optimizar
GBR_optimizer = GridSearchCV(estimator=DecisionTreeRegressor(),
                             param_grid=grid_tree,
                             n_jobs=-1,
                             scoring='accuracy',
                             refit = True,
                             cv = 7,
                             error_score=0)

grid_GBR_optimizer = GBR_optimizer.fit(x_train, y_train_2)

# Obtener la mejor combinación del clasificador GBR
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.for
```

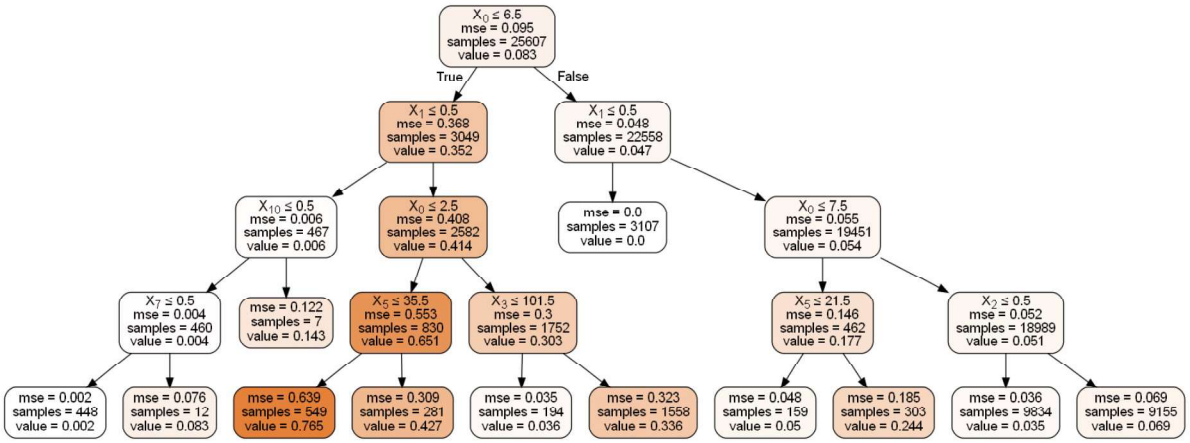
```
La mejor configuración de parámetros que maximiza la precisión es {'max_depth': 4,
'min_samples_split': 20}.
```

In [88]:
```
model_GBR = DecisionTreeRegressor(max_depth = 4,
                                  min_samples_split = 20,
                                  criterion = 'mse')

# Entrena el modelo para el conjunto train reducido a días abierto
model_GBR.fit(x_train, y_train_2)
```

Out[88]:
```
DecisionTreeRegressor(max_depth=4, min_samples_split=20)
```

In [89]:
```
# Representación del árbol de decisión.
dot_data = StringIO()
export_graphviz(model_GBR, out_file = dot_data,
                filled = True, rounded = True,
                special_characters = True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[89]:



In [90]:
```
# Precisión sobre train y test
score_train = model_GBR.score(x_train, y_train_2)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rou
score_test = model_GBR.score(x_test, y_test_2)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Predicciones sobre conjunto train y test
y_predict_train_GBR = model_GBR.predict(x_train)
y_predict_test_GBR = model_GBR.predict(x_test)
```

```
El grado de precisión del modelo para el conjunto train es 0.1824.
El grado de precisión del modelo para el conjunto test es 0.058.
```

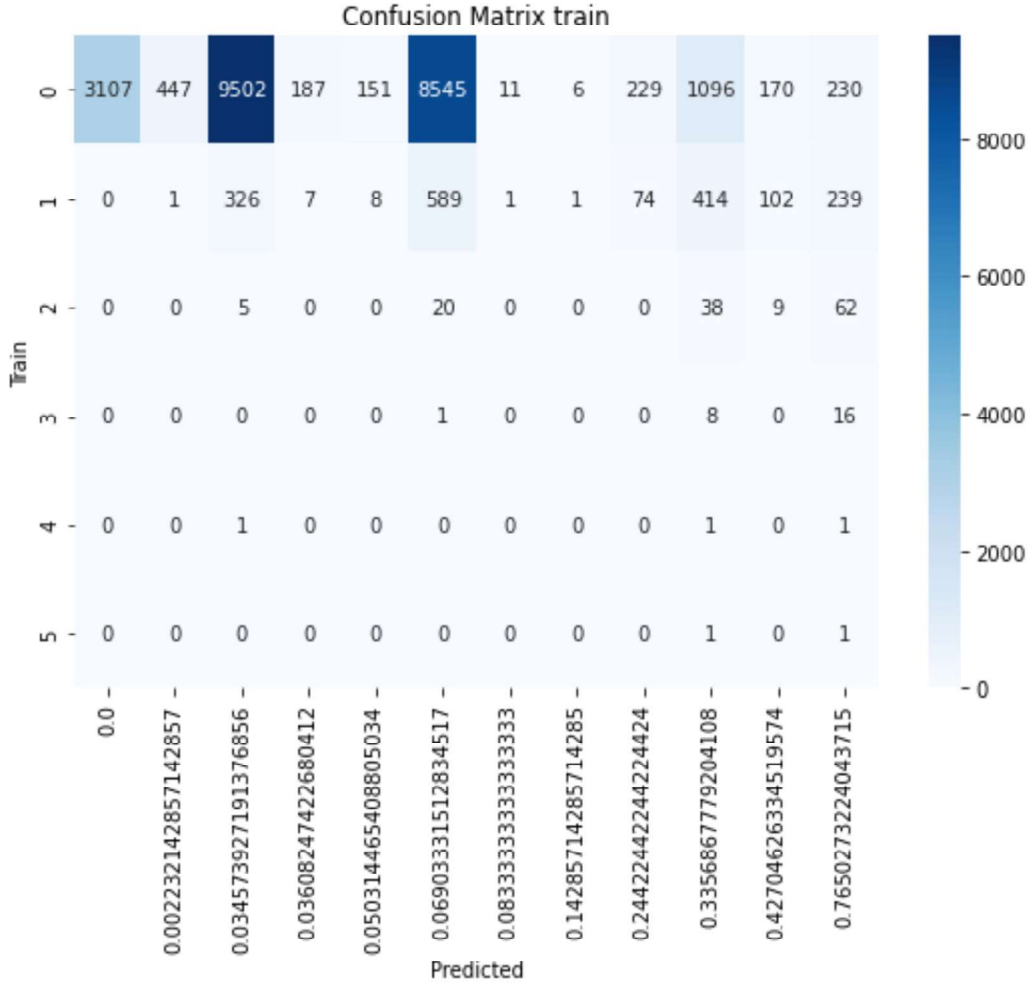
In [91]:
```
c = cross_val_score(model_GBR, x_train, y_train_2, cv=4).mean()
print('La precisión media del modelo aplicado al conjunto train, es:',round(c*100,
```

```
La precisión media del modelo aplicado al conjunto train, es: -2.63 %
```

In [92]:
```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train_GBR, y_train_2))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train_GBR, y_train_2
print('MAE in test:', mean_absolute_error(y_predict_test_GBR, y_test_2))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test_GBR, y_test_2)))
```

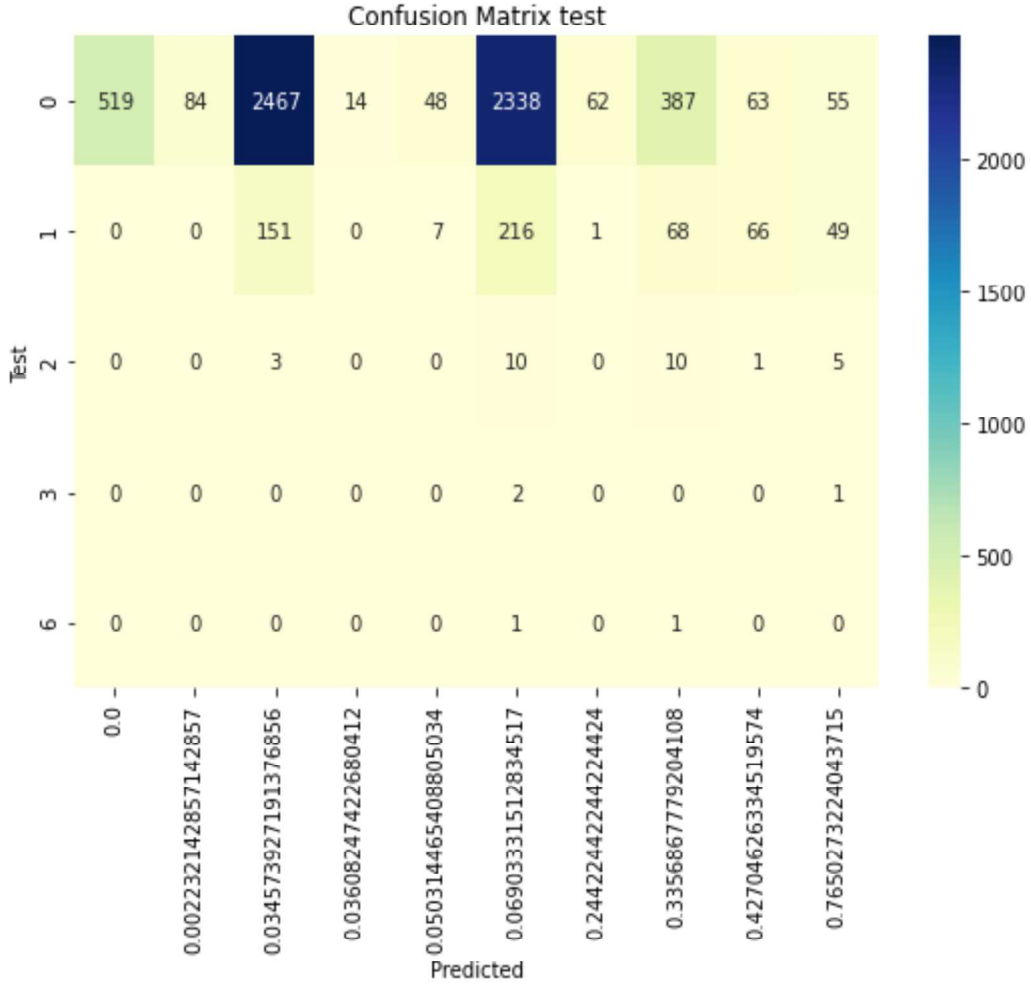
```
MAE in train: 0.1255845564528449
RMSE in train: 0.2793681590905569
MAE in test: 0.1481214866850737
RMSE in test: 0.31804488346915055
```

In [93]:
```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_2, y_predict_train_GBR, rownames=['Tr
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_GBRtrain_2.jpg')
plt.show()
```



In [94]:

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_2, y_predict_test_GBR, rownames=['Test
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_GBRtest_2.jpg')
plt.show()
```



In [95]:
```
y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_GradientBoosting = y_predict_test_GBR.astype(int)
y_predict['test_preds_GradientBoosting'] = test_preds_GradientBoosting
y_predict.to_excel('data/y_predictsvm.xls')
```

3.4.5 Algoritmo Arbol de Decisión de Gradiente Creciente (XGBoost).

Se aborda en este apartado el diseño de un modelo de predicción de la demanda a partir de algoritmo Gradiente Boosting Classifier, basado en árbol de decisión mejorado mediante sistema boosting.

Al igual que en el algoritmo GBR, se entrena con el fichero de *udsVenta* categorizado en 7 intervalos de igual tamaño, y sin filtro en bolOpen.

In [96]:
```
# Definición de los conjuntos parámetro
max_depth = [4, 5, 6, 7, 8, 9, 10]
min_samples_split = [20, 50, 100, 500]

grid_tree = dict(max_depth=max_depth, min_samples_split=min_samples_split)
cv =  StratifiedKFold(n_splits=5, shuffle=True, random_state=15)

# Optimizar
tree_optimizer = GridSearchCV(estimator=GradientBoostingClassifier(),
                              param_grid=grid_tree,
                              n_jobs=-1,
                              scoring='accuracy',
                              refit = True,
                              cv=cv,
                              error_score=0)

grid_tree_optimizer = tree_optimizer.fit(x_train, y_train_2)

# Obtener la mejor combinación del clasificador XGBoost
print ('La mejor configuración de parámetros que maximiza la precisión es {}.'.fo
print ('Clases de clasificación del modelo {}.'.format(grid_tree_optimizer.classe

# Obtenemos la precisión del modelo
print('Precisión del modelo {}.'.format(round(grid_tree_optimizer.score(x_train, y
```

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 2 members, which is less than
n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
La mejor configuración de parámetros que maximiza la precisión es {'max_depth': 4,
'min_samples_split': 500}.
Clases de clasificación del modelo [0 1 2 3 4 5].
Precisión del modelo 0.9333.
```

In [97]:
```
model_boosting = GradientBoostingClassifier(n_estimators=7,
                                            max_depth=5,
                                            min_samples_split = 100,
                                            learning_rate=0.01,
                                            criterion='mse')
c = cross_val_score(model_boosting, x_train, y_train_2, cv=7).mean()
print('La precisión media del modelo aplicado al conjunto train, es:',round(c*100,
```

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 2 members, which is less than
n_splits=7.
  % (min_groups, self.n_splits)), UserWarning)
La precisión media del modelo aplicado al conjunto train, es: 91.23 %
```

In [98]:
```
b = cross_val_score(model_boosting, x_test, y_test_2, cv=7).mean()
print('La precisión media del modelo aplicado al conjunto test, es:',round(b*100,2
```

```
C:\Users\jh100\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668:
UserWarning: The least populated class in y has only 2 members, which is less than
n_splits=7.
  % (min_groups, self.n_splits)), UserWarning)
La precisión media del modelo aplicado al conjunto test, es: 80.34 %
```

In [99]:
```
gradient_boosting = model_boosting.fit(x_train, y_train_2)
```

In [100…

```
# Precisión sobre train y test
score_train = gradient_boosting.score(x_train, y_train_2)
print('El grado de precisión del modelo para el conjunto train es {}.'.format(rou
score_test = gradient_boosting.score(x_test, y_test_2)
print('El grado de precisión del modelo para el conjunto test es {}.'.format(roun

# Predicciones sobre conjunto train y test
y_predict_train_boosting = gradient_boosting.predict(x_train)
y_predict_test_boosting = gradient_boosting.predict(x_test)
```

```
El grado de precisión del modelo para el conjunto train es 0.9248.
El grado de precisión del modelo para el conjunto test es 0.9107.
```

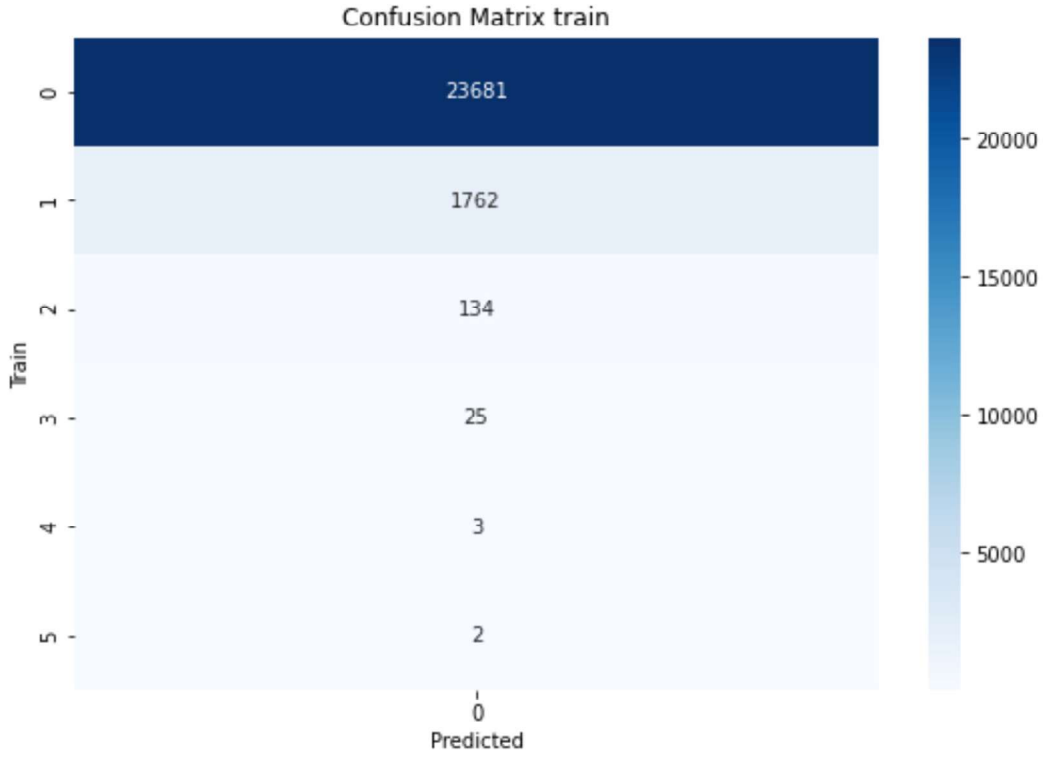
In [101…

```
#Vemos el error del algoritmo a la hora de predecir, para comprobar si a ajustado
print('MAE in train:', mean_absolute_error(y_predict_train_boosting, y_train_2))
print('RMSE in train:', np.sqrt(mean_squared_error(y_predict_train_boosting, y_tra
print('MAE in test:', mean_absolute_error(y_predict_test_boosting, y_test_2))
print('RMSE in test:', np.sqrt(mean_squared_error(y_predict_test_boosting, y_test_
```

```
MAE in train: 0.08306322489944155
RMSE in train: 0.31992940604542647
MAE in test: 0.09609292502639916
RMSE in test: 0.34148036538825066
```

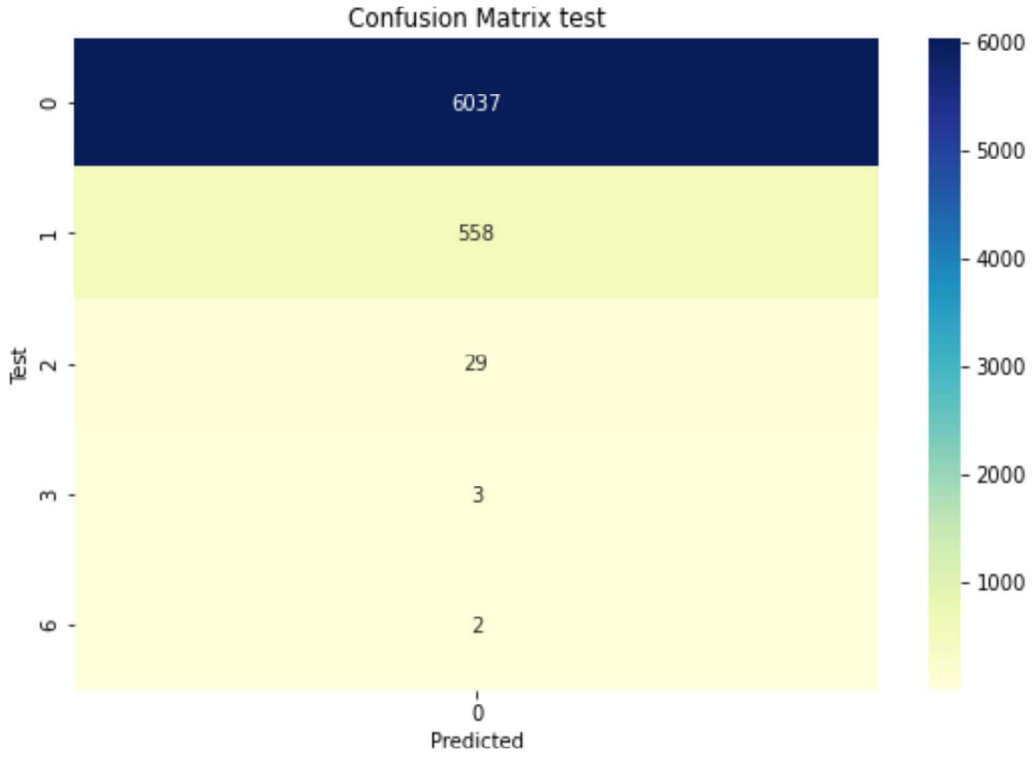
In [102…

```
# Representación gráfica de la matriz de confusión sobre el conjunto train.
confusion_matrix_table = pd.crosstab(y_train_2, y_predict_train_boosting, rownames
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="Blues", fmt='g')
plt.title("Confusion Matrix train")
plt.savefig('graph/matriz_confusion_XGBoosttrain_2.jpg')
plt.show()
```



In [103…

```
# Representación gráfica de la matriz de confusión sobre el conjunto test.
confusion_matrix_table = pd.crosstab(y_test_2, y_predict_test_boosting, rownames=[
plt.figure(1, figsize=(9, 6))
sn.heatmap(confusion_matrix_table, annot=True, cmap="YlGnBu", fmt='g')
plt.title("Confusion Matrix test")
plt.savefig('graph/matriz_confusion_XGBoosttest_2.jpg')
plt.show()
```



In [104…

```
y_predict = pd.read_excel('data/y_predictsvm.xls')
test_preds_XGBoost = y_predict_test_boosting.astype(int)
y_predict['test_preds_XGBoosting'] = test_preds_XGBoost
y_predict.to_excel('data/y_predictsvm.xls')
```

3.4.6 Algoritmo redes neuronales.

Vease ahora otros algoritmos de Machine Learning aplicando redes neuronales.

3.4.6.1 Red neuronal para la variable *udsVenta* discretizada en 7 intervalos de igual tamaño.

Se entrena modelo basado en redes neuronales para 7 clases.

- 4 capas ocultas con 64, 32, 10, 8 neuronas
- Función de activación elu y ReLU

In [146...

```
# Número de clases
num_classes = 7

model_neuronal2 = Sequential()

# Añadir las capas indicadas
model_neuronal2.add(Dense(64, input_shape=(44,), activation="relu", kernel_initia
model_neuronal2.add(Dense(32, activation="elu"))
model_neuronal2.add(Dense(10, activation="relu"))
model_neuronal2.add(Dense(8, activation="relu"))
model_neuronal2.add(Dense(num_classes, activation="softmax"))
```

In [145...

```
ohe = OneHotEncoder ()
y_train_transf2 = pd.get_dummies(
    y_train_2,columns=['udsVenta'])
y_test_transf2 = pd.get_dummies(
    y_test_2, columns=['udsVenta'])
```

In [147...

```
# Se compila el modelo con la función de perdida más adecuada.
model_neuronal2.compile(
    optimizer=Adam(lr=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy', 'mse'])
```

In [148...

```
batch = x_train.shape[0]
history = model_neuronal2.fit(
    x_train, y_train_transf2,
    validation_data=(x_test, y_test_transf2),
    epochs=300,
    batch_size=batch,
    verbose=2)
```

```
Train on 25607 samples, validate on 6629 samples
Epoch 1/300
 - 0s - loss: 15.2290 - accuracy: 0.0683 - mse: 0.2436 - val_loss: 9.4915 - val_ac
curacy: 0.0054 - val_mse: 0.2643
Epoch 2/300
 - 0s - loss: 9.0929 - accuracy: 0.0056 - mse: 0.2656 - val_loss: 4.9722 - val_acc
uracy: 0.0318 - val_mse: 0.2227
Epoch 3/300
 - 0s - loss: 4.7880 - accuracy: 0.0139 - mse: 0.2139 - val_loss: 2.7453 - val_acc
uracy: 0.0733 - val_mse: 0.1599
Epoch 4/300
 - 0s - loss: 2.6958 - accuracy: 0.0506 - mse: 0.1540 - val_loss: 2.3192 - val_acc
uracy: 0.1551 - val_mse: 0.1670
Epoch 5/300
 - 0s - loss: 2.2496 - accuracy: 0.1766 - mse: 0.1553 - val_loss: 0.9572 - val_acc
uracy: 0.7823 - val_mse: 0.0693
Epoch 6/300
 - 0s - loss: 0.9489 - accuracy: 0.8828 - mse: 0.0669 - val_loss: 0.6058 - val_acc
uracy: 0.9107 - val_mse: 0.0409
Epoch 7/300
 - 0s - loss: 0.6408 - accuracy: 0.9248 - mse: 0.0421 - val_loss: 0.4981 - val_acc
uracy: 0.9107 - val_mse: 0.0319
Epoch 8/300
 - 0s - loss: 0.5196 - accuracy: 0.9248 - mse: 0.0311 - val_loss: 0.4696 - val_acc
uracy: 0.9107 - val_mse: 0.0278
Epoch 9/300
 - 0s - loss: 0.4731 - accuracy: 0.9248 - mse: 0.0251 - val_loss: 0.4872 - val_acc
```

```
uracy: 0.9107 - val_mse: 0.0264
Epoch 10/300
 - 0s - loss: 0.4763 - accuracy: 0.9248 - mse: 0.0227 - val_loss: 0.5170 - val_acc
uracy: 0.9107 - val_mse: 0.0258
Epoch 11/300
 - 0s - loss: 0.4973 - accuracy: 0.9248 - mse: 0.0219 - val_loss: 0.5320 - val_acc
uracy: 0.9107 - val_mse: 0.0256
Epoch 12/300
 - 0s - loss: 0.5084 - accuracy: 0.9248 - mse: 0.0216 - val_loss: 0.5334 - val_acc
uracy: 0.9107 - val_mse: 0.0254
Epoch 13/300
 - 0s - loss: 0.5081 - accuracy: 0.9248 - mse: 0.0214 - val_loss: 0.5237 - val_acc
uracy: 0.9107 - val_mse: 0.0252
Epoch 14/300
 - 0s - loss: 0.4999 - accuracy: 0.9248 - mse: 0.0213 - val_loss: 0.5195 - val_acc
uracy: 0.9107 - val_mse: 0.0251
Epoch 15/300
 - 0s - loss: 0.4992 - accuracy: 0.9248 - mse: 0.0213 - val_loss: 0.5132 - val_acc
uracy: 0.9107 - val_mse: 0.0250
Epoch 16/300
 - 0s - loss: 0.4959 - accuracy: 0.9248 - mse: 0.0212 - val_loss: 0.4989 - val_acc
uracy: 0.9107 - val_mse: 0.0250
Epoch 17/300
 - 0s - loss: 0.4851 - accuracy: 0.9248 - mse: 0.0212 - val_loss: 0.4782 - val_acc
uracy: 0.9107 - val_mse: 0.0249
Epoch 18/300
 - 0s - loss: 0.4693 - accuracy: 0.9248 - mse: 0.0212 - val_loss: 0.4550 - val_acc
uracy: 0.9107 - val_mse: 0.0248
Epoch 19/300
 - 0s - loss: 0.4518 - accuracy: 0.9248 - mse: 0.0213 - val_loss: 0.4333 - val_acc
uracy: 0.9107 - val_mse: 0.0247
Epoch 20/300
 - 0s - loss: 0.4353 - accuracy: 0.9248 - mse: 0.0214 - val_loss: 0.4157 - val_acc
uracy: 0.9107 - val_mse: 0.0247
Epoch 21/300
 - 0s - loss: 0.4211 - accuracy: 0.9248 - mse: 0.0216 - val_loss: 0.4031 - val_acc
uracy: 0.9107 - val_mse: 0.0248
Epoch 22/300
 - 0s - loss: 0.4102 - accuracy: 0.9248 - mse: 0.0218 - val_loss: 0.3948 - val_acc
uracy: 0.9107 - val_mse: 0.0249
Epoch 23/300
 - 0s - loss: 0.4021 - accuracy: 0.9248 - mse: 0.0220 - val_loss: 0.3897 - val_acc
uracy: 0.9107 - val_mse: 0.0250
Epoch 24/300
 - 0s - loss: 0.3964 - accuracy: 0.9248 - mse: 0.0220 - val_loss: 0.3845 - val_acc
uracy: 0.9107 - val_mse: 0.0250
Epoch 25/300
 - 0s - loss: 0.3881 - accuracy: 0.9248 - mse: 0.0219 - val_loss: 0.3780 - val_acc
uracy: 0.9107 - val_mse: 0.0248
Epoch 26/300
 - 0s - loss: 0.3762 - accuracy: 0.9248 - mse: 0.0215 - val_loss: 0.3711 - val_acc
uracy: 0.9107 - val_mse: 0.0245
Epoch 27/300
 - 0s - loss: 0.3623 - accuracy: 0.9248 - mse: 0.0210 - val_loss: 0.3656 - val_acc
uracy: 0.9107 - val_mse: 0.0243
Epoch 28/300
 - 0s - loss: 0.3491 - accuracy: 0.9248 - mse: 0.0205 - val_loss: 0.3631 - val_acc
uracy: 0.9107 - val_mse: 0.0241
Epoch 29/300
 - 0s - loss: 0.3398 - accuracy: 0.9248 - mse: 0.0202 - val_loss: 0.3623 - val_acc
uracy: 0.9107 - val_mse: 0.0240
Epoch 30/300
 - 0s - loss: 0.3341 - accuracy: 0.9248 - mse: 0.0201 - val_loss: 0.3603 - val_acc
uracy: 0.9107 - val_mse: 0.0240
Epoch 31/300
 - 0s - loss: 0.3278 - accuracy: 0.9248 - mse: 0.0200 - val_loss: 0.3568 - val_acc
```

```
uracy: 0.9107 - val_mse: 0.0240
Epoch 32/300
 - 0s - loss: 0.3210 - accuracy: 0.9248 - mse: 0.0200 - val_loss: 0.3536 - val_acc
uracy: 0.9107 - val_mse: 0.0241
Epoch 33/300
 - 0s - loss: 0.3151 - accuracy: 0.9248 - mse: 0.0200 - val_loss: 0.3514 - val_acc
uracy: 0.9107 - val_mse: 0.0242
Epoch 34/300
 - 0s - loss: 0.3106 - accuracy: 0.9248 - mse: 0.0201 - val_loss: 0.3501 - val_acc
uracy: 0.9107 - val_mse: 0.0243
Epoch 35/300
 - 0s - loss: 0.3068 - accuracy: 0.9248 - mse: 0.0201 - val_loss: 0.3493 - val_acc
uracy: 0.9107 - val_mse: 0.0242
Epoch 36/300
 - 0s - loss: 0.3035 - accuracy: 0.9248 - mse: 0.0200 - val_loss: 0.3491 - val_acc
uracy: 0.9107 - val_mse: 0.0241
Epoch 37/300
 - 0s - loss: 0.3009 - accuracy: 0.9248 - mse: 0.0199 - val_loss: 0.3494 - val_acc
uracy: 0.9107 - val_mse: 0.0240
Epoch 38/300
 - 0s - loss: 0.2991 - accuracy: 0.9248 - mse: 0.0198 - val_loss: 0.3498 - val_acc
uracy: 0.9107 - val_mse: 0.0239
Epoch 39/300
 - 0s - loss: 0.2980 - accuracy: 0.9248 - mse: 0.0197 - val_loss: 0.3489 - val_acc
uracy: 0.9107 - val_mse: 0.0238
Epoch 40/300
 - 0s - loss: 0.2964 - accuracy: 0.9248 - mse: 0.0197 - val_loss: 0.3458 - val_acc
uracy: 0.9107 - val_mse: 0.0238
Epoch 41/300
 - 0s - loss: 0.2933 - accuracy: 0.9248 - mse: 0.0196 - val_loss: 0.3420 - val_acc
uracy: 0.9107 - val_mse: 0.0238
Epoch 42/300
 - 0s - loss: 0.2901 - accuracy: 0.9248 - mse: 0.0195 - val_loss: 0.3395 - val_acc
uracy: 0.9107 - val_mse: 0.0239
Epoch 43/300
 - 0s - loss: 0.2883 - accuracy: 0.9248 - mse: 0.0195 - val_loss: 0.3376 - val_acc
uracy: 0.9107 - val_mse: 0.0239
Epoch 44/300
 - 0s - loss: 0.2874 - accuracy: 0.9248 - mse: 0.0195 - val_loss: 0.3356 - val_acc
uracy: 0.9107 - val_mse: 0.0237
Epoch 45/300
 - 0s - loss: 0.2862 - accuracy: 0.9248 - mse: 0.0194 - val_loss: 0.3338 - val_acc
uracy: 0.9107 - val_mse: 0.0235
Epoch 46/300
 - 0s - loss: 0.2851 - accuracy: 0.9248 - mse: 0.0193 - val_loss: 0.3325 - val_acc
uracy: 0.9107 - val_mse: 0.0233
Epoch 47/300
 - 0s - loss: 0.2845 - accuracy: 0.9248 - mse: 0.0192 - val_loss: 0.3307 - val_acc
uracy: 0.9107 - val_mse: 0.0233
Epoch 48/300
 - 0s - loss: 0.2833 - accuracy: 0.9248 - mse: 0.0192 - val_loss: 0.3288 - val_acc
uracy: 0.9107 - val_mse: 0.0233
Epoch 49/300
 - 0s - loss: 0.2814 - accuracy: 0.9248 - mse: 0.0192 - val_loss: 0.3285 - val_acc
uracy: 0.9107 - val_mse: 0.0233
Epoch 50/300
 - 0s - loss: 0.2803 - accuracy: 0.9248 - mse: 0.0192 - val_loss: 0.3298 - val_acc
uracy: 0.9107 - val_mse: 0.0234
Epoch 51/300
 - 0s - loss: 0.2799 - accuracy: 0.9248 - mse: 0.0192 - val_loss: 0.3313 - val_acc
uracy: 0.9107 - val_mse: 0.0234
Epoch 52/300
 - 0s - loss: 0.2795 - accuracy: 0.9248 - mse: 0.0191 - val_loss: 0.3316 - val_acc
uracy: 0.9107 - val_mse: 0.0233
Epoch 53/300
 - 0s - loss: 0.2788 - accuracy: 0.9248 - mse: 0.0191 - val_loss: 0.3309 - val_acc
```

```
uracy: 0.9107 - val_mse: 0.0232
Epoch 54/300
 - 0s - loss: 0.2780 - accuracy: 0.9248 - mse: 0.0190 - val_loss: 0.3293 - val_acc
uracy: 0.9107 - val_mse: 0.0231
Epoch 55/300
 - 0s - loss: 0.2770 - accuracy: 0.9248 - mse: 0.0190 - val_loss: 0.3274 - val_acc
uracy: 0.9107 - val_mse: 0.0231
Epoch 56/300
 - 0s - loss: 0.2762 - accuracy: 0.9248 - mse: 0.0190 - val_loss: 0.3258 - val_acc
uracy: 0.9107 - val_mse: 0.0232
Epoch 57/300
 - 0s - loss: 0.2758 - accuracy: 0.9248 - mse: 0.0190 - val_loss: 0.3246 - val_acc
uracy: 0.9107 - val_mse: 0.0231
Epoch 58/300
 - 0s - loss: 0.2753 - accuracy: 0.9248 - mse: 0.0190 - val_loss: 0.3237 - val_acc
uracy: 0.9107 - val_mse: 0.0231
Epoch 59/300
 - 0s - loss: 0.2744 - accuracy: 0.9248 - mse: 0.0190 - val_loss: 0.3235 - val_acc
uracy: 0.9107 - val_mse: 0.0230
Epoch 60/300
 - 0s - loss: 0.2736 - accuracy: 0.9248 - mse: 0.0189 - val_loss: 0.3237 - val_acc
uracy: 0.9107 - val_mse: 0.0229
Epoch 61/300
 - 0s - loss: 0.2730 - accuracy: 0.9248 - mse: 0.0189 - val_loss: 0.3241 - val_acc
uracy: 0.9107 - val_mse: 0.0229
Epoch 62/300
 - 0s - loss: 0.2725 - accuracy: 0.9248 - mse: 0.0189 - val_loss: 0.3242 - val_acc
uracy: 0.9107 - val_mse: 0.0230
Epoch 63/300
 - 0s - loss: 0.2722 - accuracy: 0.9248 - mse: 0.0189 - val_loss: 0.3234 - val_acc
uracy: 0.9107 - val_mse: 0.0229
Epoch 64/300
 - 0s - loss: 0.2717 - accuracy: 0.9248 - mse: 0.0189 - val_loss: 0.3218 - val_acc
uracy: 0.9107 - val_mse: 0.0229
Epoch 65/300
 - 0s - loss: 0.2710 - accuracy: 0.9248 - mse: 0.0188 - val_loss: 0.3204 - val_acc
uracy: 0.9107 - val_mse: 0.0228
Epoch 66/300
 - 0s - loss: 0.2705 - accuracy: 0.9248 - mse: 0.0188 - val_loss: 0.3194 - val_acc
uracy: 0.9107 - val_mse: 0.0228
Epoch 67/300
 - 0s - loss: 0.2701 - accuracy: 0.9248 - mse: 0.0188 - val_loss: 0.3188 - val_acc
uracy: 0.9107 - val_mse: 0.0228
Epoch 68/300
 - 0s - loss: 0.2698 - accuracy: 0.9248 - mse: 0.0188 - val_loss: 0.3187 - val_acc
uracy: 0.9107 - val_mse: 0.0228
Epoch 69/300
 - 0s - loss: 0.2695 - accuracy: 0.9248 - mse: 0.0188 - val_loss: 0.3188 - val_acc
uracy: 0.9107 - val_mse: 0.0228
Epoch 70/300
 - 0s - loss: 0.2692 - accuracy: 0.9248 - mse: 0.0187 - val_loss: 0.3191 - val_acc
uracy: 0.9107 - val_mse: 0.0227
Epoch 71/300
 - 0s - loss: 0.2689 - accuracy: 0.9248 - mse: 0.0187 - val_loss: 0.3195 - val_acc
uracy: 0.9107 - val_mse: 0.0227
Epoch 72/300
 - 0s - loss: 0.2687 - accuracy: 0.9248 - mse: 0.0187 - val_loss: 0.3196 - val_acc
uracy: 0.9107 - val_mse: 0.0227
Epoch 73/300
 - 0s - loss: 0.2685 - accuracy: 0.9248 - mse: 0.0187 - val_loss: 0.3192 - val_acc
uracy: 0.9107 - val_mse: 0.0227
Epoch 74/300
 - 0s - loss: 0.2683 - accuracy: 0.9248 - mse: 0.0187 - val_loss: 0.3184 - val_acc
uracy: 0.9107 - val_mse: 0.0226
Epoch 75/300
 - 0s - loss: 0.2680 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3173 - val_acc
```

```
uracy: 0.9107 - val_mse: 0.0226
Epoch 76/300
 - 0s - loss: 0.2676 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3163 - val_acc
uracy: 0.9107 - val_mse: 0.0226
Epoch 77/300
 - 0s - loss: 0.2673 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3153 - val_acc
uracy: 0.9107 - val_mse: 0.0225
Epoch 78/300
 - 0s - loss: 0.2671 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3147 - val_acc
uracy: 0.9107 - val_mse: 0.0225
Epoch 79/300
 - 0s - loss: 0.2669 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3143 - val_acc
uracy: 0.9107 - val_mse: 0.0225
Epoch 80/300
 - 0s - loss: 0.2666 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3142 - val_acc
uracy: 0.9107 - val_mse: 0.0224
Epoch 81/300
 - 0s - loss: 0.2663 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3143 - val_acc
uracy: 0.9107 - val_mse: 0.0224
Epoch 82/300
 - 0s - loss: 0.2660 - accuracy: 0.9248 - mse: 0.0186 - val_loss: 0.3144 - val_acc
uracy: 0.9107 - val_mse: 0.0224
Epoch 83/300
 - 0s - loss: 0.2658 - accuracy: 0.9248 - mse: 0.0185 - val_loss: 0.3145 - val_acc
uracy: 0.9107 - val_mse: 0.0224
Epoch 84/300
 - 0s - loss: 0.2656 - accuracy: 0.9248 - mse: 0.0185 - val_loss: 0.3143 - val_acc
uracy: 0.9107 - val_mse: 0.0224
Epoch 85/300
 - 0s - loss: 0.2653 - accuracy: 0.9248 - mse: 0.0185 - val_loss: 0.3140 - val_acc
uracy: 0.9107 - val_mse: 0.0224
Epoch 86/300
 - 0s - loss: 0.2651 - accuracy: 0.9248 - mse: 0.0185 - val_loss: 0.3134 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 87/300
 - 0s - loss: 0.2649 - accuracy: 0.9248 - mse: 0.0185 - val_loss: 0.3129 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 88/300
 - 0s - loss: 0.2646 - accuracy: 0.9248 - mse: 0.0185 - val_loss: 0.3126 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 89/300
 - 0s - loss: 0.2644 - accuracy: 0.9248 - mse: 0.0185 - val_loss: 0.3125 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 90/300
 - 0s - loss: 0.2642 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3125 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 91/300
 - 0s - loss: 0.2639 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3127 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 92/300
 - 0s - loss: 0.2637 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3129 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 93/300
 - 0s - loss: 0.2634 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3130 - val_acc
uracy: 0.9107 - val_mse: 0.0223
Epoch 94/300
 - 0s - loss: 0.2632 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3129 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 95/300
 - 0s - loss: 0.2630 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3128 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 96/300
 - 0s - loss: 0.2628 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3127 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 97/300
 - 0s - loss: 0.2626 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3127 - val_acc
```

```
uracy: 0.9107 - val_mse: 0.0222
Epoch 98/300
 - 0s - loss: 0.2624 - accuracy: 0.9248 - mse: 0.0184 - val_loss: 0.3127 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 99/300
 - 0s - loss: 0.2622 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3128 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 100/300
 - 0s - loss: 0.2620 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3126 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 101/300
 - 0s - loss: 0.2618 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3124 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 102/300
 - 0s - loss: 0.2616 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3122 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 103/300
 - 0s - loss: 0.2615 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3120 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 104/300
 - 0s - loss: 0.2613 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3118 - val_acc
uracy: 0.9107 - val_mse: 0.0222
Epoch 105/300
 - 0s - loss: 0.2611 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3117 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 106/300
 - 0s - loss: 0.2609 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3115 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 107/300
 - 0s - loss: 0.2607 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3114 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 108/300
 - 0s - loss: 0.2605 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3113 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 109/300
 - 0s - loss: 0.2604 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3111 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 110/300
 - 0s - loss: 0.2602 - accuracy: 0.9248 - mse: 0.0183 - val_loss: 0.3109 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 111/300
 - 0s - loss: 0.2600 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3107 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 112/300
 - 0s - loss: 0.2598 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3107 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 113/300
 - 0s - loss: 0.2596 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3106 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 114/300
 - 0s - loss: 0.2594 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3106 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 115/300
 - 0s - loss: 0.2592 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3104 - val_acc
uracy: 0.9107 - val_mse: 0.0221
Epoch 116/300
 - 0s - loss: 0.2591 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3104 - val_acc
uracy: 0.9107 - val_mse: 0.0220
Epoch 117/300
 - 0s - loss: 0.2589 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3101 - val_acc
uracy: 0.9107 - val_mse: 0.0220
Epoch 118/300
 - 0s - loss: 0.2587 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3097 - val_acc
uracy: 0.9107 - val_mse: 0.0220
Epoch 119/300
 - 0s - loss: 0.2585 - accuracy: 0.9248 - mse: 0.0182 - val_loss: 0.3097 - val_acc
```