MINI PROYECTO

Equipo 12 – Deep Learning

Descripción breve

- El presente informe aborda la exploración de los datos para su entendimiento dentro del contexto organizacional, la preparación de los datos para poder utilizarlos como entrada para modelos predictivos, el análisis preliminar de selección de modelos relevantes para responder a la pregunta, el desarrollo y calibración de modelos y la visualización de resultados .

INTEGRANTES

-Christian Beraún Chamorro
- D'sharlie Sánchez Rozo
- William Alexander Morales Valera

1. INTRODUCCIÓN

El presente miniproyecto se basa en las características musicales de una canción y si estas influyen o son determinantes respecto del año de publicación o lanzamiento. En resumen, se busca responder a la pregunta ¿Existe una relación entre las características musicales de una canción y el año en que fue publicada/lanzada? Una respuesta positiva a esta pregunta revelaría una profunda comprensión de la naturaleza de una composición musical y, lo que es más importante, esta comprensión se demostraría matemáticamente.

En este proyecto, se utilizan datos reales para predecir el año en que fue lanzada una canción a partir de sus características del timbre en la grabación. En total, son 90 atributos predictores: los 12 primeros corresponden al timbre promedio y los 78 siguientes a la covarianza. Originalmente, este tipo de datos fue recolectado en un proyecto llamado Million Song Dataset de la Universidad de Columbia (http://millionsongdataset.com/). Los datos para trabajar tienen los mismos predictores pero con canciones no consideradas en la base de datos original.

En general, se deben reportar los resultados del modelamiento predictivo siguiendo los pasos que se muestran a continuación:

- Exploración de los datos para su entendimiento dentro del contexto organizacional.
- Preparación de los datos para poder utilizarlos como entrada para modelos predictivos.
- Análisis preliminar de selección de modelos relevantes para responder a la pregunta.
- Desarrollo y calibración de modelos.
- Visualización de resultados.

DATASETS PROPORCIONADOS

- sampleSubmission.csv
- testReg.csv
- trainReg.csv
  Disponibles en: https://www.kaggle.com/competitions/music-year-prediction/data

2. PARTE I: EXPLORACIÓN DE LOS DATOS Y CONTEXTO ORGANIZACIONAL

Para empezar, necesitamos mapear la descripción, cantidad de variables y calidad de los datos de ambos datasets, los cuales se realizó a continuación:

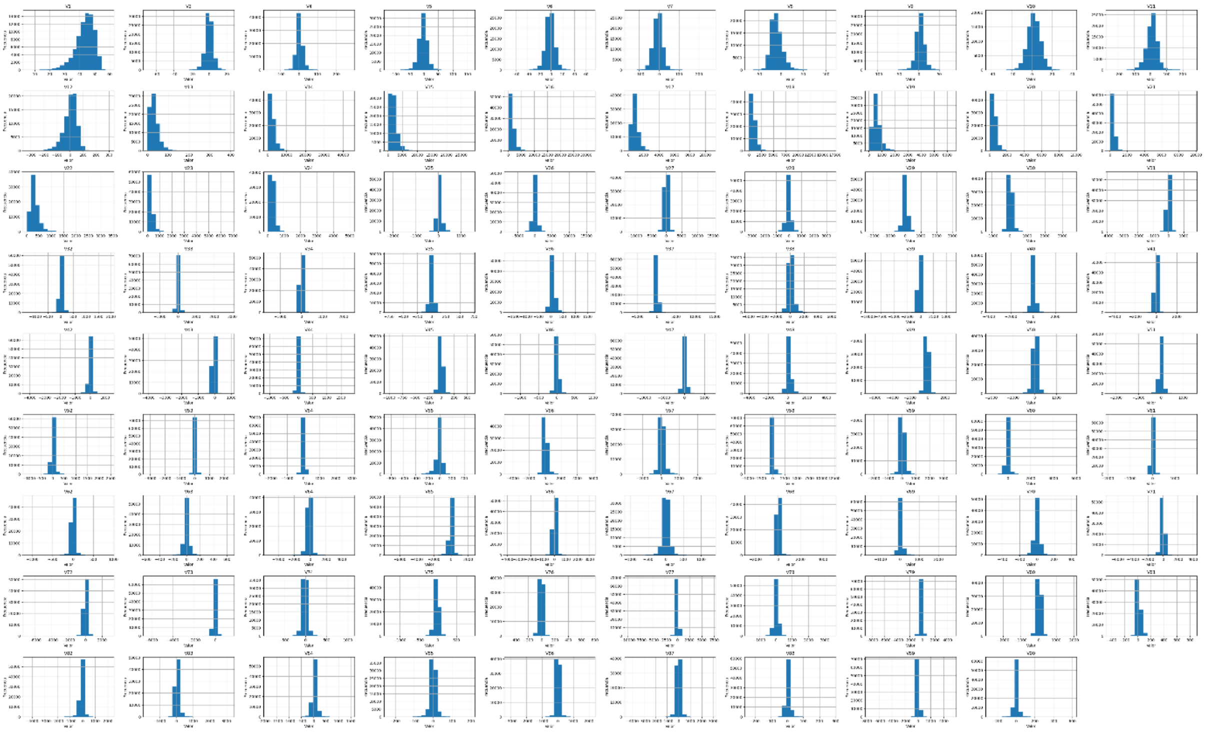
Train Data: 77779 rows × 92 columns

Tabla 1. Análisis Descriptivo de los datos

Type	ID	Y	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
count	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779
mean	48244.2383	2002.30813	43.425185	-0.13671979	0	3.75531308	-2.33976807	-1.64309543	-6.81499697	-9.58726436	3.74957225	1.94851205	8.67485686	1.97787842	33.6375361	2419.77863	1960.66285
std	27846.2837	10.811038	6.12886878	4.37046639	0	17.6091827	14.4839754	7.89857439	22.980515	12.9119143	10.6385807	6.43009341	35.0960495	52.1597874	22.6040562	1734.20162	1266.74071
min	1	1926	4.83688	-69.68087	0	-165.22161	-121.47534	-72.50385	-152.40755	-70.69342	-119.48753	-41.63166	-249.10745	-329.57138	0.5874	59.01231	71.92873
25%	24079.5	1998	40.060315	-2.612435	0	-7.01624	-10.685075	-6.30069	-21.265305	-18.583065	-2.2894	-2.270755	-11.115595	-25.00927	18.04011	1298.63774	1098.55515
50%	48357	2006	44.32385	-0.063	0	2.0221	-2.05456	-1.58624	-6.30761	-11.23833	3.92844	1.89768	10.89415	9.3701	28.84442	1995.79785	1679.37829
75%	72340.5	2010	47.90008	2.46595	0	12.77645	6.4239	3.077055	7.617995	-2.32933	10.039325	6.163805	29.76277	37.11787	43.343385	3030.43891	2484.72071
max	96435	2014	60.03401	23.81526	0	274.65858	160.81522	68.44796	262.06887	112.97141	80.78712	39.97683	262.28304	304.26639	396.91851	43921.6472	28231.0998
Type	**V16**	**V17**	**V18**	**V19**	**V20**	**V21**	**V22**	**V23**	**V24**	**V25**	**V26**	**V27**	**V28**	**V29**	**V30**	**V31**	**V32**
count	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779
mean	1518.44301	908.683312	882.374897	602.121692	515.486995	391.864407	324.662498	289.652977	291.749852	42.6531646	45.1744792	-51.7059202	-27.2903774	16.3361058	45.3634331	5.10268845	24.1629371
std	1146.29573	476.064042	590.64115	319.122294	314.465741	212.942125	169.499811	193.684396	155.259611	122.34173	705.925653	565.245898	219.284056	164.130994	137.895841	101.773912	72.0424704
min	68.11035	60.89138	23.90388	43.6845	9.15615	22.12135	18.35053	10.92598	10.83169	-2214.38045	-7645.54417	-12017.0889	-3163.0211	-2243.27056	-1121.16473	-3805.66617	-1328.84733
25%	810.625395	602.9576	484.929745	389.49676	315.046905	255.597385	211.5009	167.39054	197.37992	-11.42276	-282.901785	-255.459435	-117.066795	-58.5	-24.470705	-38.45101	-6.26207
50%	1236.65707	812.65969	739.569	539.12575	441.19302	348.22422	290.62639	241.30872	263.54078	27.66455	-38.70731	-41.89223	-22.16499	8.90315	30.62104	4.20457	23.2163
75%	1881.72451	1099.89625	1118.72944	737.833145	623.834955	474.79821	395.4166	356.105855	349.54909	88.85793	286.10921	165.780825	66.428495	85.818855	97.90438	48.68869	55.569455
max	31792.7148	10787.6512	16831.949	6360.24268	9569.77809	9616.6156	3365.4697	6737.1215	5036.61707	1432.56131	16337.8676	14505.3422	3410.61556	2963.43419	3553.18488	1584.70058	1954.35548
Type	**V33**	**V34**	**V35**	**V36**	**V37**	**V38**	**V39**	**V40**	**V41**	**V42**	**V43**	**V44**	**V45**	**V46**	**V47**	**V48**	**V49**
count	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779
mean	9.45009103	-3.79119002	0.56498076	72.4339041	-52.1220811	115.195914	-191.301896	19.8639784	0.17065014	17.4710593	-52.3960796	3.10810553	-2.10795885	5.8277606	75.6339744	145.50257	-88.6308129
std	75.0767429	56.2198568	43.4797702	108.840815	420.088149	450.650852	275.823815	205.132499	122.914826	120.701322	76.5278512	39.5468929	42.1739681	55.8935541	468.856902	265.295702	220.503087
min	-1679.11832	-1590.63713	-765.99379	-1711.484	-7298.32896	-5924.44473	-9752.74109	-3919.39572	-4434.06853	-4175.41268	-4071.55347	-1072.95552	-1021.28921	-1329.95974	-13731.1786	-3992.68866	-6642.39958
25%	-24.06061	-28.466095	-16.70903	14.397875	-223.686765	-96.732745	-289.399435	-71.23451	-51.736725	-27.09117	-78.900325	-14.606395	-21.06588	-17.78936	-113.413005	13.063705	-158.43634
50%	8.12595	-9.53697	3.47933	55.35636	-73.57698	80.60075	-159.15431	19.10876	-2.66917	26.01287	-45.36519	3.09825	-1.28065	3.62073	71.60452	107.81089	-63.27651
75%	40.85454	12.379805	20.87695	116.15152	80.962925	304.055655	-60.162905	111.54419	48.462315	75.45868	-18.1711	21.5464	18.549035	27.342455	278.486235	243.07607	11.28586
max	2868.35173	2330.33366	691.12904	1636.96095	14148.9979	6011.87812	6065.05481	3542.43713	3537.50359	1301.4294	934.82714	1830.54468	568.99828	943.93652	6664.22793	4550.38543	2792.32447
Type	**V50**	**V51**	**V52**	**V53**	**V54**	**V55**	**V56**	**V57**	**V58**	**V59**	**V60**	**V61**	**V62**	**V63**	**V64**	**V65**	**V66**
count	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779
mean	26.2429905	6.30456856	28.7691004	12.1550628	1.05757721	-10.7076745	62.2000638	104.668247	2.92360621	36.7551425	-25.9475077	3.74109316	0.33110399	-0.6415598	-138.292931	-2.66932844	0.82408989
std	122.029293	93.4899766	75.9489943	72.5352572	83.9874145	58.1211171	273.554783	310.125825	279.538596	165.606073	147.081674	61.3342389	49.6714256	38.4150885	305.029857	225.204798	130.350883
min	-2075.49185	-2270.81107	-1089.50545	-3188.17738	-2199.78221	-847.6934	-4536.69953	-3494.01692	-4730.5991	-2327.42228	-1601.64462	-1900.1048	-1129.51344	-583.20098	-5365.13856	-7375.97744	-3896.27522
25%	-23.57536	-38.51442	-8.26689	-18.690025	-34.089275	-29.20602	-70.45673	-38.60207	-115.189255	-45.623365	-93.068185	-24.852835	-21.619025	-19.032475	-261.40949	-86.45431	-51.019845
50%	32.13812	5.13511	26.22027	8.00347	0.23035	-2.10108	27.89036	84.5646	-14.28362	26.60129	-30.27674	0.09307	1.64608	-3.45574	-114.62037	6.52295	5.6545
75%	86.83728	50.610915	62.20244	38.64616	36.61764	18.41268	161.617165	223.208035	88.179795	108.99046	31.846855	27.21505	23.779375	14.7436	1.72641	104.23111	54.864545
max	1739.29042	1426.84804	2460.43343	2394.66234	2900.52018	506.60846	5734.17435	5489.79489	11971.1799	3607.75017	5690.29165	1811.22866	906.46898	647.15692	5100.98473	2313.09535	3127.04473
Type	**V67**	**V68**	**V69**	**V70**	**V71**	**V72**	**V73**	**V74**	**V75**	**V76**	**V77**	**V78**	**V79**	**V80**	**V81**	**V82**	**V83**
count	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779	77779
mean	3.53222347	28.1411986	32.0184738	-0.93296169	-4.96434133	4.55240036	-30.4072039	-11.8669472	-21.2431762	-5.47216709	-23.2969748	31.9069743	-105.995059	25.663725	15.7172149	-71.7330078	41.6792868
std	99.1226656	118.435311	105.55083	36.9420541	249.503583	229.706924	172.780891	64.0142015	64.5118109	26.8709447	269.101216	144.142312	209.971584	124.016038	32.2119947	176.893351	123.523393
min	-1048.04155	-2564.78812	-1578.06641	-549.15769	-7057.71245	-6953.35736	-6395.81177	-889.47244	-1312.45848	-495.54551	-9812.57578	-1628.81769	-8390.03545	-2760.59631	-424.51757	-4402.37644	-1733.72211
25%	-45.54368	-19.128765	-19.16078	-16.079025	-92.167525	-89.70404	-87.59399	-42.55214	-50.531905	-14.907965	-150.129885	-30.22832	-162.72255	-29.04156	-1.968265	-136.511095	-21.467975
50%	2.83274	18.20059	22.21309	-0.72087	17.17829	12.88197	-21.7561	-11.91485	-16.55181	-2.22658	-57.01251	38.54099	-71.00022	25.72258	9.07998	-51.32257	28.21756
75%	51.79019	61.58248	71.772825	14.743075	112.53175	103.427945	39.601015	17.31348	11.458575	7.68559	61.996475	104.181415	-7.34313	84.283625	26.2117	13.99987	89.190475
max	1290.30838	4779.80027	2696.86075	411.36502	3958.07011	2907.04705	1395.24174	1061.60085	879.54753	741.02902	6913.59405	3423.59535	4723.48192	1911.27553	840.97338	2147.94214	3210.7017
Type	**V84**	**V85**	**V86**	**V87**	**V88**	**V89**	**V90**										
count	77779	77779	77779	77779	77779	77779	77779										
mean	37.8163527	0.32821288	17.5750966	-25.6288567	4.46348419	18.6648855	1.240194										
std	95.3353567	16.2726893	115.243644	173.310304	13.5264139	186.690183	22.3796535										
min	-1848.70226	-238.38673	-3168.92457	-4319.99232	-236.03926	-7458.37815	-281.1506										
25%	-4.604295	-6.70237	-31.56939	-100.66848	-2.56936	-60.12198	-8.90012										
50%	33.41737	0.78898	15.23736	-21.58164	3.1369	5.94149	-0.09534										
75%	77.216175	8.35427	67.361475	51.33315	10.002145	84.389125	9.5207										
max	1482.64214	199.1215	2144.10391	2833.60895	275.35366	5289.11138	600.76624										

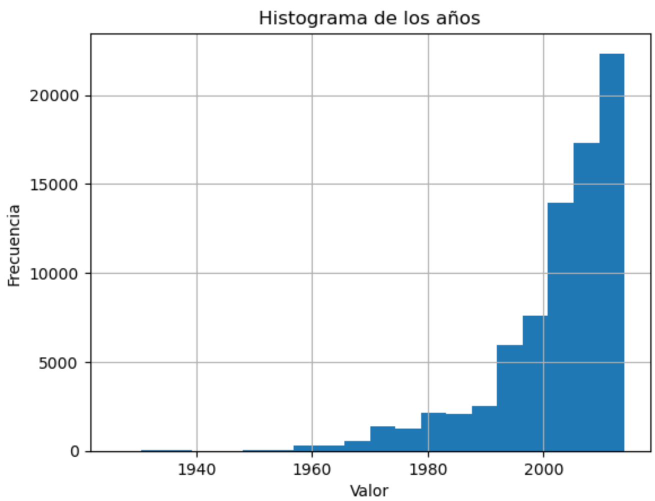
Se realiza un primer análisis descriptivos de las variables revisando la medida, mínimos, máximos y desviación estándar, a simple vista se puede evidenciar que la columna V3 esta presente en todos los registros pero su valor es cero, por otra lado en las primeras 12 variables se pueden evidenciar valores mas acordes según su desviación estándar, siendo que son 91 variables, se revisó en el notebook de una forma más amplia en histogramas, para efectos de observar el comportamiento y distribución de las variables.

Gráfico n.° 1 – Histogramas



Luego se realizó un análisis de la distribución de la variable a predecir:

Gráfico n.° 2 – Distribución de la variable a predecir



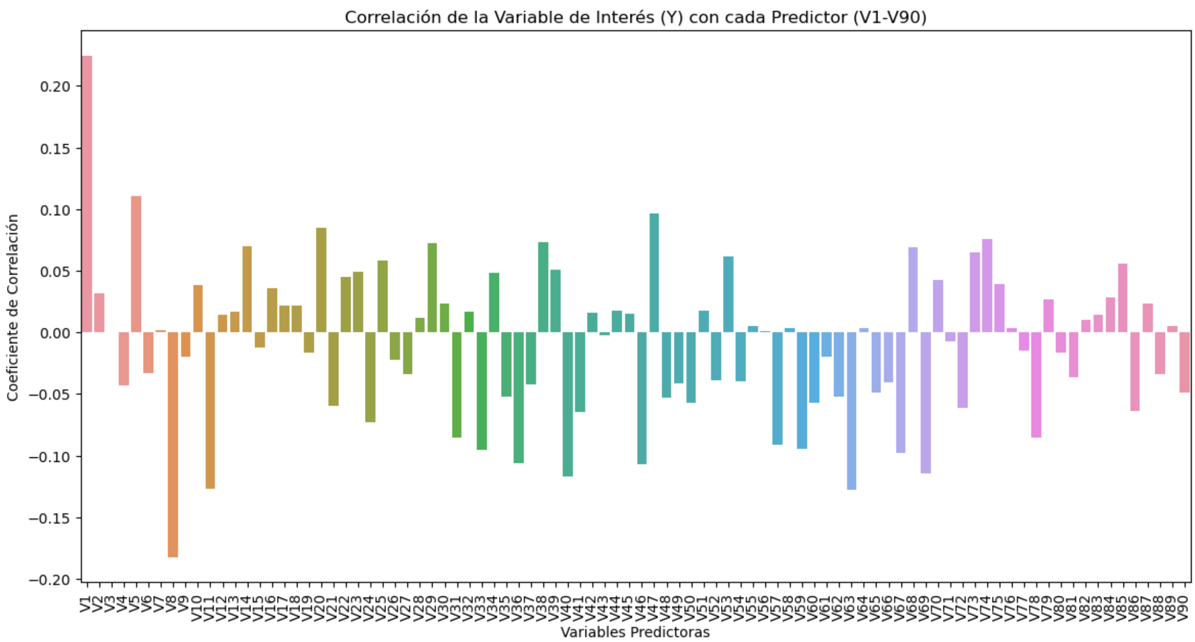
Como podemos observar, existe una distribución mayor para los años más cercanos a la fecha; ello podría deberse a que los datos se encuentran disponibles en diversas plataformas streaming de música siendo que estas surgieron en los últimos 20 años, tal como lo precisa en la página web de los datasets:

- SecondHandSongs dataset -> cover songs
- musiXmatch dataset -> lyrics
- Last.fm dataset -> song-level tags and similarity
- Taste Profile subset -> user data
- thisismyjam-to-MSD mapping -> more user data

- tagtraum genre annotations -> genre labels
- Top MAGD dataset -> more genre labels

Ahora bien, tengamos en cuenta este dato importante "los 12 primeros (V1 A V12) corresponden al timbre promedio y los 78 siguientes a la covarianza (V13 a V90)". Analizaremos la correlación que existe entre las variables predictoras con la variable de interés:

Gráfico n.° 3 – Matriz de Correlación

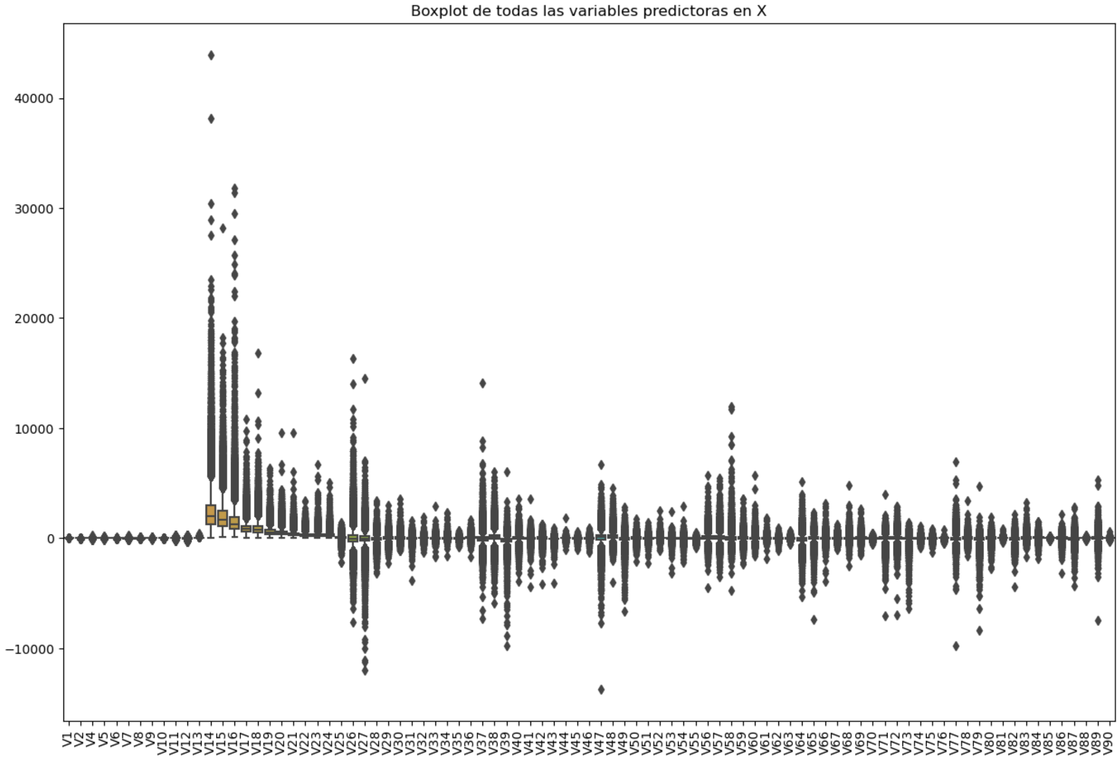


Realizando el análisis de correlación de las variables predictoras vs la variable objetivo, podemos evidenciar que hay una correlación de -20 a 20, lo que indica que hay muy poca correlación entre la variable objetivo y las variables predictoras.

3. PARTE II: PREPARACIÓN DE LOS DATOS PARA MODELOS PREDICTIVOS

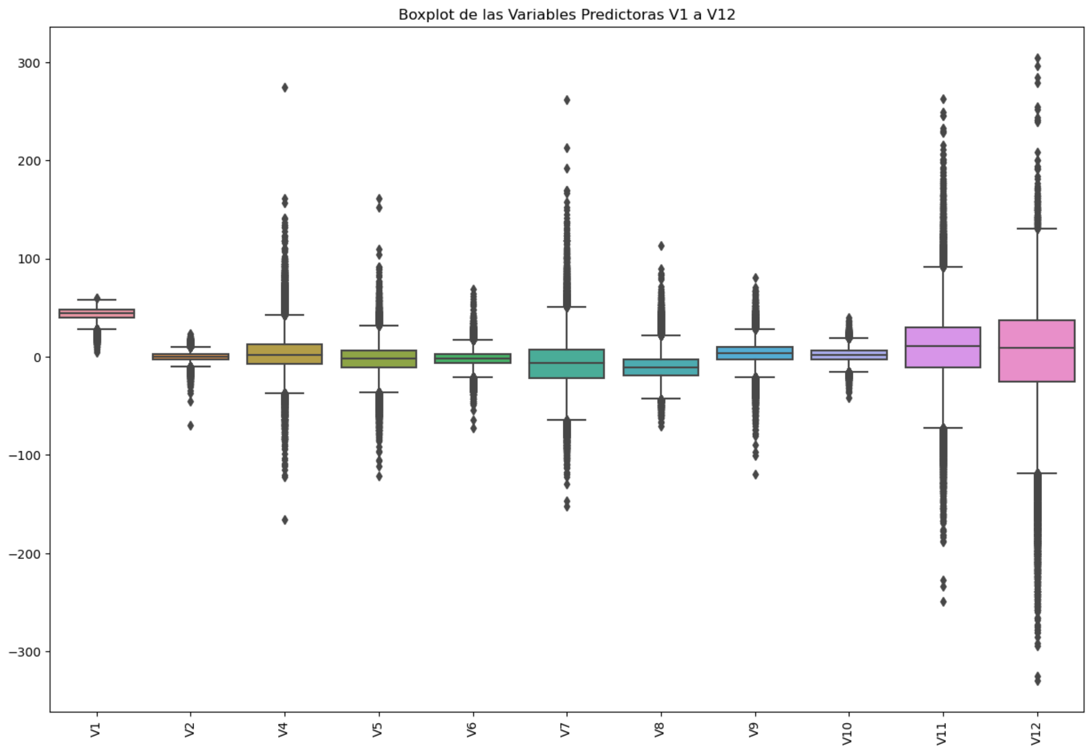
Como pudimos observar, las variables predictoras van de V1 a V90, pero se debe eliminar V3 y ID, siendo que no agregan valor al modelo, por ser valores vacíos o el índice de los registros. Ahora procedemos a analizar la dispersión de los datos

Gráfico n.° 4 – Determinación de outliers



Ahora bien, si observamos las variables V14 en adelante, podemos ver que hay mucha variabilidad. Mientras que de V1 a V12, no hay mucha.

Gráfico n.° 5 – Boxplot de las variables V1 a V12

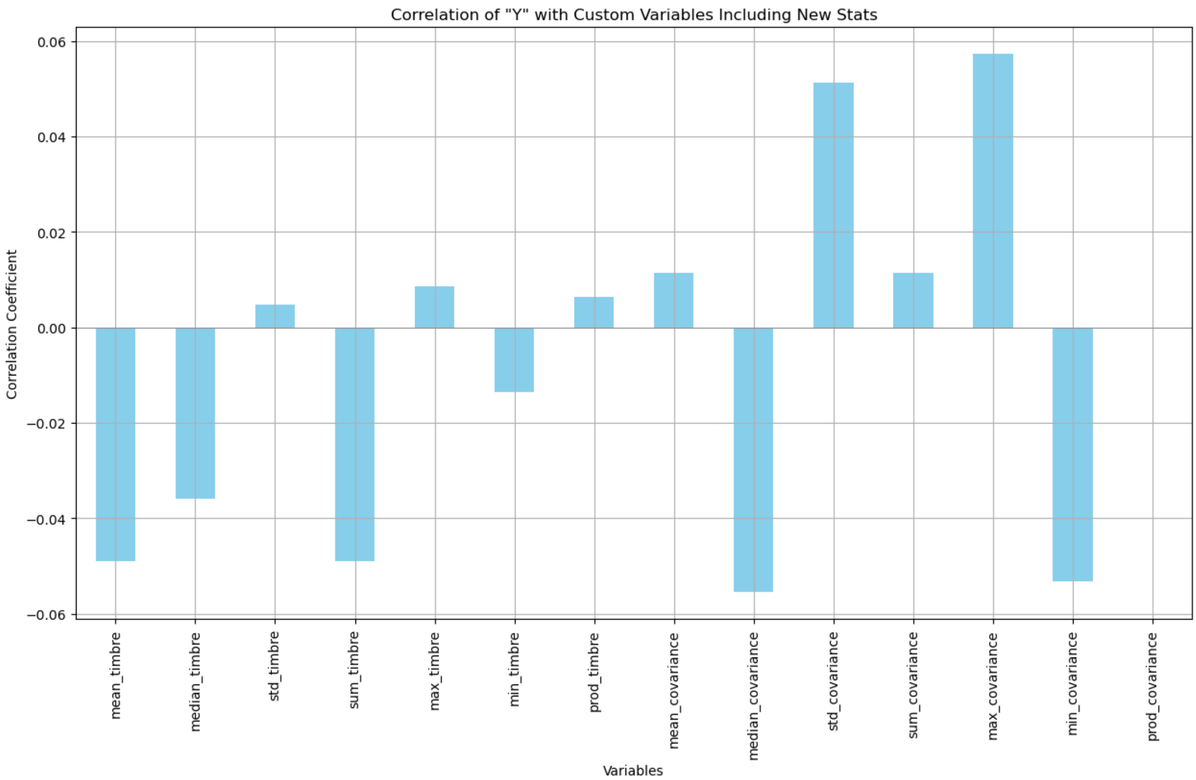


Considerando todo lo anterior, se vio por conveniente crear variables que capturen los datos de V1 a V12 y de V13 a V90, usando métricas como suma, promedio, mediana, producto, entre otras. Conforme se detalla a continuación:

```python
# Función para crear características
train_data = train_data.drop(columns=['ID','V3'])
test_data = test_data.drop(columns=['ID'])

def create_features(data):
    # Adjusted range to exclude 'V3'
    timbre_cols = [f"V{i}" for i in range(1, 13) if i != 3]

    # Check if all required timbre columns except V3 exist
    if all(col in data.columns for col in timbre_cols):
        data['mean_timbre'] = data[timbre_cols].mean(axis=1)
        data['median_timbre'] = data[timbre_cols].median(axis=1)
        data['std_timbre'] = data[timbre_cols].std(axis=1)
        data['sum_timbre'] = data[timbre_cols].sum(axis=1)
        data['max_timbre'] = data[timbre_cols].max(axis=1)
        data['min_timbre'] = data[timbre_cols].min(axis=1)
        data['prod_timbre'] = data[timbre_cols].prod(axis=1)
    else:
        print("Algunas columnas necesarias (V1, V2, V4 a V12) no existen en el DataFrame")
    # Check if all covariance columns (V13 to V90) exist
    if all(f"V{i}" in data.columns for i in range(13, 91)):
        covariance_cols = [f"V{i}" for i in range(13, 91)]
        data['mean_covariance'] = data[covariance_cols].mean(axis=1)
        data['median_covariance'] = data[covariance_cols].median(axis=1)
        data['std_covariance'] = data[covariance_cols].std(axis=1)
        data['sum_covariance'] = data[covariance_cols].sum(axis=1)
        data['max_covariance'] = data[covariance_cols].max(axis=1)
        data['min_covariance'] = data[covariance_cols].min(axis=1)
        # Using logarithmic addition to prevent overflow in product
        data['prod_covariance'] = np.exp(data[covariance_cols].apply(np.log).sum(axis=1))
    else:
        print("Algunas columnas de V13 a V90 no existen en el DataFrame")

    return data
# Aplicar la creación de características
train_data = create_features(train_data)
test_data = create_features(test_data)
```

Como se puede evidenciar en el fragmento de código anterior, se valida la existencia de columnas en el set de datos con el objetivo de manejar posibles errores al momento de realizar la ejecución de proceso.

Gráfico n.° 6 – Correlación con "Y" de las nuevas variables



En la gráfico anterior podemos observar que fue buena decisión agregar estas características, especialmente aquellas que muestran una correlación más alta de 0.02. Estas no solo capturan los datos de los dos grupos mencionados, sino que también están más estrechamente relacionadas con la variable objetivo

4. PARTE III: ANÁLISIS PRELIMINAR DE SELECCIÓN DE MODELOS RELEVANTES

*Modelo 1: Una sola neurona *

In [21]:
```python
# Modelo 1: Simplificado con una sola neurona
model1 = Sequential([
    Dense(1, input_dim=X_train_scaled.shape[1], activation='linear')  # Una sola salida para regresión
])

# Compilación del modelo con el optimizador Adam
optimizer = Adam(learning_rate=0.2)
model1.compile(optimizer=optimizer, loss='mean_squared_error')

# Callback de EarlyStopping para evitar el sobreajuste
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)

# Entrenamiento del modelo con EarlyStopping
model1.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_data=(X_val_scaled, y_val), callbacks=[early_stopping])

# Evaluación del modelo
y_pred = model1.predict(X_val_scaled)
if np.isnan(y_pred).any():
    raise ValueError("Las predicciones contienen NaN")

# Cálculo del RMSE para evaluar el rendimiento del modelo
rmse = sqrt(mean_squared_error(y_val, y_pred))
print(f'RMSE: {rmse}')
```

```
Epoch 1/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 2s 640us/step - loss: 3646274.7500 - val_loss: 2653277.7500
Epoch 2/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 602us/step - loss: 2378656.0000 - val_loss: 1631743.1250
Epoch 3/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 597us/step - loss: 1429066.5000 - val_loss: 888608.5000
Epoch 4/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 621us/step - loss: 748726.3125 - val_loss: 392420.5625
Epoch 5/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 616us/step - loss: 309997.9375 - val_loss: 116953.0547
Epoch 6/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 602us/step - loss: 81577.9453 - val_loss: 14191.3877
Epoch 7/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 611us/step - loss: 7712.5869 - val_loss: 218.8185
Epoch 8/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 613us/step - loss: 135.9781 - val_loss: 102.4660
Epoch 9/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 599us/step - loss: 95.4779 - val_loss: 97.0490
Epoch 10/10
1945/1945 ━━━━━━━━━━━━━━━━━━━━ 1s 605us/step - loss: 102.8793 - val_loss: 108.8134
Restoring model weights from the end of the best epoch: 9.
487/487 ━━━━━━━━━━━━━━━━━━━━ 0s 491us/step
RMSE: 9.851345353532812
```

*Modelo 2: Múltiples Capas *

*Modelo 2: Múltiples Capas *

```
model2 = Sequential([
    Dense(200, input_dim=X_train_scaled.shape[1], activation='relu'),  # Primera capa oculta con 100 neuronas
    Dense(50, activation='relu'),  # Segunda capa oculta con 20 neuronas
    Dense(1, activation='linear')  # Capa de salida para regresión
])

# Compilación del modelo con el optimizador Adam
optimizer = Adam(learning_rate=0.001)
model2.compile(optimizer=optimizer, loss='mean_squared_error')

# Callback de EarlyStopping para evitar el sobreajuste
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)

# Entrenamiento del modelo con EarlyStopping
model2.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_data=(X_val_scaled, y_val), callbacks=[early_stopping])

# Evaluación del modelo
y_pred = model2.predict(X_val_scaled)
if np.isnan(y_pred).any():
    raise ValueError("Las predicciones contienen NaN")

# Cálculo del RMSE para evaluar el rendimiento del modelo
rmse = sqrt(mean_squared_error(y_val, y_pred))
print(f'RMSE: {rmse}')
```

```
Epoch 1/10
C:\Users\CHRISTIAN\anaconda3\lib\site-packages\keras\src\layers\core\dense.py:86: UserWarning: Do not pass an `input_shape`/`in
put_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the mod
el instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 754us/step - loss: 1203407.5000 - val_loss: 51299.4648
Epoch 2/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 1s 716us/step - loss: 37058.0742 - val_loss: 11845.1768
Epoch 3/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 719us/step - loss: 7206.3481 - val_loss: 1883.6832
Epoch 4/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 722us/step - loss: 1215.3699 - val_loss: 666.9042
Epoch 5/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 737us/step - loss: 395.8275 - val_loss: 492.5071
Epoch 6/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 725us/step - loss: 283.0558 - val_loss: 659.5401
Epoch 7/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 1s 713us/step - loss: 341.4096 - val_loss: 618.0717
Epoch 8/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 727us/step - loss: 418.1860 - val_loss: 289.3042
Epoch 9/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 722us/step - loss: 258.7532 - val_loss: 192.1150
Epoch 10/10
2066/2066 ━━━━━━━━━━━━━━━━━━━━ 2s 721us/step - loss: 308.6641 - val_loss: 233.8360
Restoring model weights from the end of the best epoch: 9.
365/365 ━━━━━━━━━━━━━━━━━━━━ 0s 606us/step
RMSE: 13.86055419151385
```

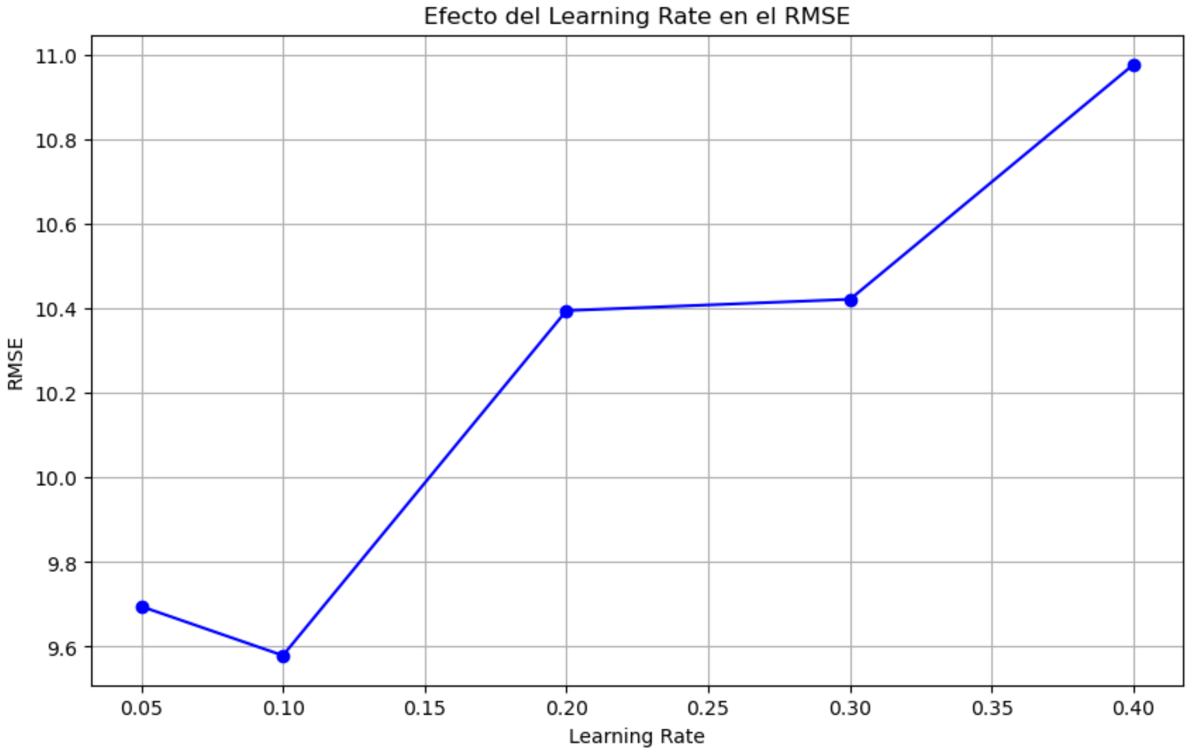
Cabe precisar que existirá un modelo 3, pero será resultante de la calibración del modelo 2 lo cual se observará en las siguientes páginas.

5. PARTE IV: DESARROLLO Y CALIBRACIÓN DE MODELOS

Se realizó la calibración de ambos modelos, por ejemplo a nivel de learning rate:

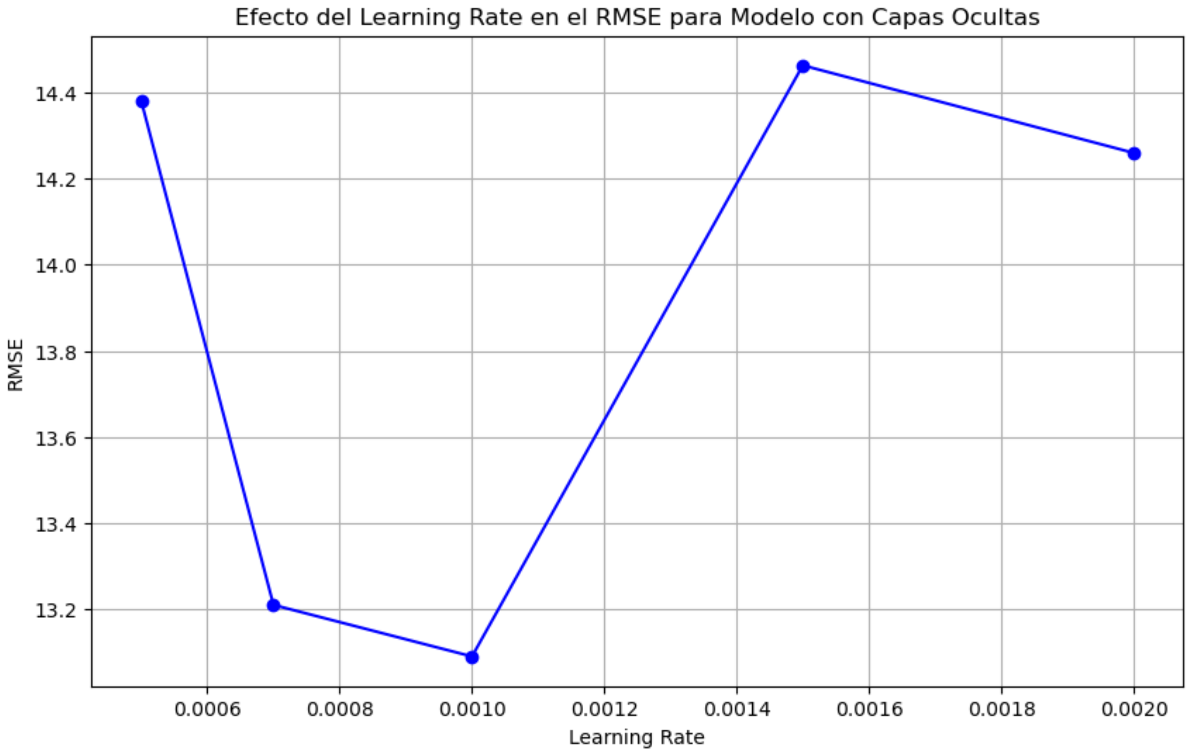
```
In [80]: learning_rates = [0.05, 0.1, 0.2, 0.3, 0.4]
rmse_results = []
for lr in learning_rates:
    print(f"Entrenando modelo con learning rate: {lr}")
    model1 = Sequential([
        Dense(1, input_dim=X_train_scaled.shape[1], activation='linear')
    ])
    model1.compile(optimizer=Adam(learning_rate=lr), loss='mean_squared_error')
    model1.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_data=(X_val_scaled, y_val), verbose=0)
    y_pred = model1.predict(X_val_scaled)
    rmse = sqrt(mean_squared_error(y_val, y_pred))
    rmse_results.append(rmse)
plt.figure(figsize=(10, 6))
plt.plot(learning_rates, rmse_results, marker='o', linestyle='-', color='b')
plt.title('Efecto del Learning Rate en el RMSE')
plt.xlabel('Learning Rate')
plt.ylabel('RMSE')
plt.grid(True)
plt.show()
```

Gráfico n.° 7 – Calibración Learning Rate Modelo 1



Y para el modelo 2 con capas ocultas (múltiples capas):

Gráfico n.° 78– Calibración Learning Rate Modelo 2



Como podemos observar, en ambos casos el óptimo del Leargnin Rate es 0.001; ahora procedemos calibrar aún más el modelo 2:

```python
# Configuraciones de neuronas para cada modelo
configurations = {
    'Config 1': [160, 80, 45, 25, 20],  # Configuración original
    'Config 2': [165, 80, 45, 25, 20],  # Ligeramente más grande
    'Config 3': [170, 80, 45, 25, 20],   # Ligeramente más pequeño
    'Config 4': [175, 80, 45, 25, 20],  # Configuración original
    'Config 5': [180, 80, 45, 25, 20],  # Ligeramente más grande
    'Config 6': [160, 85, 45, 25, 20],   # Ligeramente más pequeño
    'Config 7': [160, 75, 45, 25, 20],  # Configuración original
    'Config 8': [160, 70, 45, 25, 20],  # Ligeramente más grande
    'Config 9': [160, 80, 45, 20, 20],   # Ligeramente más pequeño
    'Config 10': [160, 80, 45, 20, 15],  # Configuración original
    'Config 11': [160, 80, 45, 25, 10],  # Ligeramente más grande
    'Config 12': [160, 90, 45, 25, 20]   # Ligeramente más pequeño
}

learning_rate = 0.001
results = []

# Pruebas de cada configuración
for config_name, layers in configurations.items():
    print(f"Probando modelo {config_name} con configuración de capas: {layers}")
    model = Sequential()
    model.add(Dense(layers[0], input_dim=X_train_scaled.shape[1], activation='elu'))
    model.add(BatchNormalization())

    for neurons in layers[1:-1]:
        model.add(Dense(neurons, activation='elu'))
        model.add(BatchNormalization())

    # Última capa antes de la salida
    model.add(Dense(layers[-1], activation='relu'))
    model.add(BatchNormalization())

    model.add(Dense(1, activation='linear'))  # Salida para regresión

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)
    model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_data=(X_val_scaled, y_val), callbacks=[early_stopping

    y_pred = model.predict(X_val_scaled)
    if np.isnan(y_pred).any():
        raise ValueError("Las predicciones contienen NaN")

    rmse = sqrt(mean_squared_error(y_val, y_pred))
    results.append((config_name, layers, rmse))
    print(f"RMSE para {config_name}: {rmse}")

# Encontrar la mejor configuración
best_result = min(results, key=lambda x: x[2])
print(f"Mejor configuración: {best_result[0]} con capas {best_result[1]}, RMSE: {best_result[2]}")
```

El código anterior fue diseñado para explorar la configuración óptima de una red neuronal multicapa. Una vez establecida la estructura básica, se procedió a calibrar ciertos parámetros y a incorporar diversas técnicas para mejorar el rendimiento y la generalización del modelo. Entre estas técnicas se incluyeron el uso de dropout y normalización por lotes (batch normalization) para regularizar el modelo, la implementación de paradas tempranas (early stopping) para evitar el sobreajuste, la optimización mediante el algoritmo Adam, y la selección cuidadosa de las funciones de activación. Estas estrategias fueron aplicadas con el objetivo de afinar el modelo y alcanzar los mejores resultados posibles en términos de precisión y eficiencia durante el entrenamiento.

En el modelo final calibrado (Modelo 3), tenemos lo siguiente:

In [30]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error
from math import sqrt

# Modelo modificado
model = Sequential([
    Dense(360, input_dim=X_train_scaled.shape[1], activation='relu'),
    Dropout(0.3),  # Agregar dropout para regularización
    BatchNormalization(),
    Dense(180, activation='elu'),
    Dropout(0.3),  # Agregar dropout para regularización
    BatchNormalization(),
    Dense(60, activation='relu'),
    BatchNormalization(),
    Dense(25, activation='relu'),
    BatchNormalization(),
    Dense(20, activation='relu'),
    BatchNormalization(),
    Dense(1, activation='linear')  # Salida para regresión
])

# Compilación con Adam
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mean_squared_error')

# Callback de EarlyStopping modificado
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)

# Entrenamiento con EarlyStopping
model.fit(X_train_scaled, y_train, epochs=50, batch_size=50, validation_data=(X_val_scaled, y_val), callbacks=[early_stopping])

# Evaluación
y_pred = model.predict(X_val_scaled)
if np.isnan(y_pred).any():
    raise ValueError("Las predicciones contienen NaN")

rmse = sqrt(mean_squared_error(y_val, y_pred))
print(f'RMSE: {rmse}')
```

Sobre el modelo:

- **Dense:** Representa una capa densamente conectada (o completamente conectada). Cada neurona en una capa densa recibe entrada de todas las neuronas de la capa anterior, lo cual es una característica de las redes neuronales tradicionales.
- **La primera capa densa con 360 neuronas**, donde input_dim especifica el número de características de entrada del modelo. La función de activación 'relu' (Rectified Linear Unit) es comúnmente utilizada por su eficiencia y efectividad en redes neuronales profundas.
- **Dropout(0.3):** Una técnica de regularización donde aleatoriamente se "apagan" un porcentaje de las neuronas durante el entrenamiento para prevenir el sobreajuste. 0.3 significa que el 30% de las neuronas en la capa anterior se desactivarán aleatoriamente en cada paso durante el entrenamiento.
- **BatchNormalization():** Normaliza las activaciones de la capa anterior al restar la media del batch y dividir por la desviación estándar, lo que ayuda a mejorar la velocidad, rendimiento y estabilidad del entrenamiento.

Considerando que tiene normalización, aplicación técnicas y ajustes, se considero a este modelo 2 calibrado, como el modelo 3 "Múltiples capas Adam y calibración de hiperparámetros"

Siendo el mejor modelo obtenido, se generaron los predicts

In [31]:
```python
test_data2 = pd.read_csv('testReg.csv')
test_ids = test_data2['ID'].copy()
test_data = test_data[selected_features2]

# Asumiendo que 'scaler' y 'model' ya están definidos y entrenados
X_test_scaled = scaler.transform(test_data)
predictions = model.predict(X_test_scaled).flatten()

# Creación de un DataFrame para guardar las predicciones junto con los IDs 8.67698
output = pd.DataFrame({'ID': test_ids, 'Y': np.round(predictions).astype(int)})
output.to_csv('predictionsplease292.csv', index=False)
print("Predicciones guardadas en 'predictionsplease292.csv'.")
```

```
584/584 ━━━━━━━━━━━━━━━━━━━━ 0s 741us/step
Predicciones guardadas en 'predictionsplease292.csv'.
```

Hito:

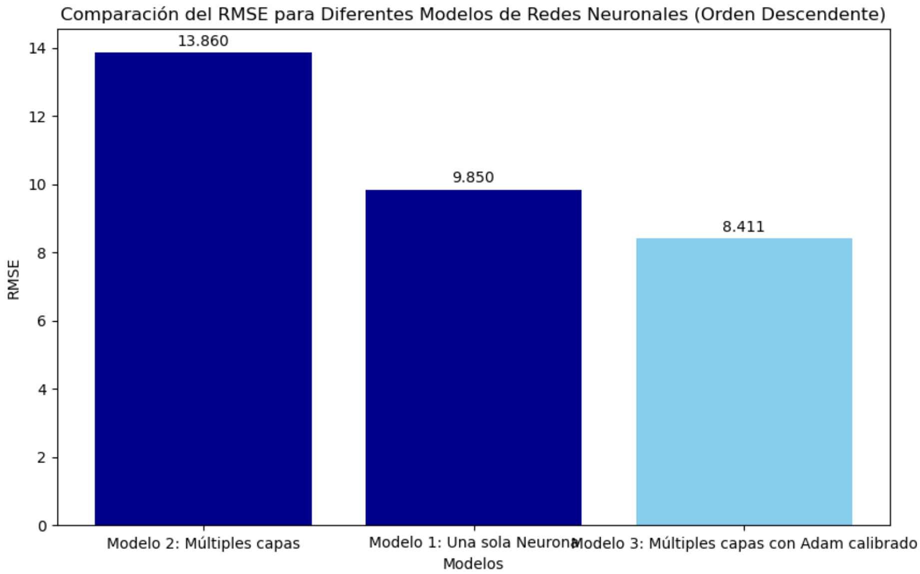
- RMSE EN LAB LOCAL - 8.41185
- RMSE EN KAGGLE - 8.64070
- RANKING EN EL PUBLIC LEADERBOARD 5 DE 21 EQUIPOS.
- DIFERENCIA CON EL RANKING 1: 8.64070 - 8.4901 = 0.1505

Fuente: Public Leaderboard de Kaggle: https://www.kaggle.com/competitions/music-year-prediction/leaderboard, con información al 29.04.2024 a las 4:15 am.

6. PARTE V: VISUALIZACIÓN DE RESULTADOS

A continuación se presentan los resultados obtenidos por cada tipo de modelo empleado:

Gráfico n.° X– Comparación de modelos



Como podemos observar, inicialmente daba la impresión de que el modelo de una sola neurona sería mejor que de múltiples capas, pero a lo largo de las calibraciones realizadas, empleando batchnormalization, técnica de dropout y búsqueda intensiva de parámetros óptimos que incluye el learning rate, se obtuvieron los siguientes resultados:

1. Modelo 1: Una sola Neurona / RMSE OBTENIDO: 9.85
2. Modelo 2: Múltiples capas / RMSE OBTENIDO: 13.86
3. Modelo 3: Múltiples capas Adam y calibración de hiperparámetros / RMSE OBTENIDO: 8.411

En ese sentido, el modelo 3 al tener el mejor RMSE, fue utilizado para generar las predicciones y así obtener el puntaje de 8.64070, con lo cual nuestro equipo se colocó en el ranking 5 de la competencia.

Tabla 2. Modelos y los resultados obtenidos

Detalle del Modelo	RMSE JupyterNotebook	Puntuación Kaggle
Modelo 01: Red neuronal – Una sola neurona	9.85	10.11
Modelo 02: Red neuronal – Multicapa	13.86	15.86
Modelo 03: Red neuronal – Multicapa Adam calibrado	8.411	8.64

Recomendaríamos utilizar el modelo de redes neuronales Adam, calibrando los parámetros de número de capas, tasa de aprendizaje, neuronas por capa, tipo de función de activación, implementando estrategias de batchnormalization, dropout y earlystopping, a efectos de mejorar aún más el modelo.

Ahora bien, respondiendo a la pregunta de interés... sí, es posible siendo que las variables indican que existiría una relación entre las características musicales de una canción y el año en que fue publicada/lanzada.

Anexos.

Anexo 1. Resultado en Competencia (pdf)

Anexo 2. HTML Notebook

Anexo 3. Notebook en Jupyter