Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 22/05/2022
Practica No.15 Pronostico.

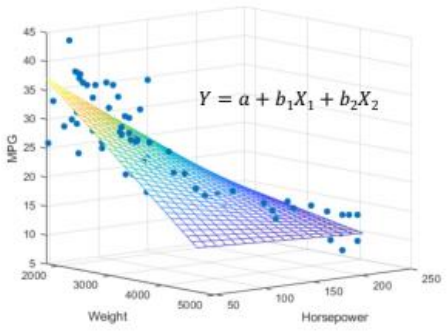
Objetivo.

Obtener un pronóstico acertado de las ventas totales mensuales en un cine basándonos en sus características.

Características.

Datos relacionados a las ventas, ocupación, capacidad y películas proyectadas en cines antes de la pandemia de diferentes áreas.

La regresión lineal múltiple es un algoritmo que busca calcular una salida Y (Una variable dependiente) en función de múltiples entradas (Variables dependientes) calculando la ecuación que genere la recta o el hiperplano el cual minimiza la distancia entre cada uno de los puntos y el hiperplano ajustado.



$$\hat{Y} = a + b_1x_1 + b_2x_2 + \cdots + b_nx_n + \mu$$

Donde μ, es el residuo del modelo que representa la diferencia entre el punto pronosticado y la observación real. Es importante considerar que al ser un algoritmo de aprendizaje supervisado los datos tienen que estar etiquetados, es decir, que todos deberán tener un valor real de la variable dependiente el cual se va a utilizar para entrenar al modelo.

Los árboles de decisión son algoritmos los cuales toman una serie de variables independientes como entrada y las analizan una por una en cada uno de los diferentes niveles del árbol partiendo de una raíz para obtener el determinado valor de una variable dependiente.

En este aspecto tendremos 2 diferentes tipos de árboles de decisión, primero los árboles de clasificación los cuales trabajan con etiquetas como resultados (A, B, C), (0/1), etc. y después están los árboles de regresión, dichos arboles están diseñados para entregar como resultado un valor continuo.

Los bosques aleatorios son algoritmos que buscan generalizar más las soluciones que nos pueden entregar los árboles aleatorios mediante la combinación de varios de estos árboles en la misma estructura de tal forma que cada uno de ellos aporte una solución similar para después llegar a un consenso evitando además un sobreajuste en los datos.

Al igual que en los árboles de decisión principalmente se trabajará con 2 tipos de estructuras para los bosques aleatorios, primero los bosques compuestos por árboles de clasificación los cuales trabajan con etiquetas como resultados (A, B, C), (0/1), etc. tomando aquella etiqueta que se repita más entre la salida de los árboles como la salida del modelo y después están los bosques compuestos por árboles de regresión, dichos arboles están diseñados para entregar como resultado un valor continuo y para los bosques se busca obtener un promedio con los resultados de cada árbol.

Desarrollo.

Como primer paso tenemos la importación de las librerías necesarias para trabajar con el conjunto de datos de los cines y sus ventas. `pandas` para la manipulación de los datos, `numpy` para el manejo de matrices, `matplotlib` y `seaborn` para la visualización de estos datos mediante gráficos de diferente tipo dependiendo del análisis.

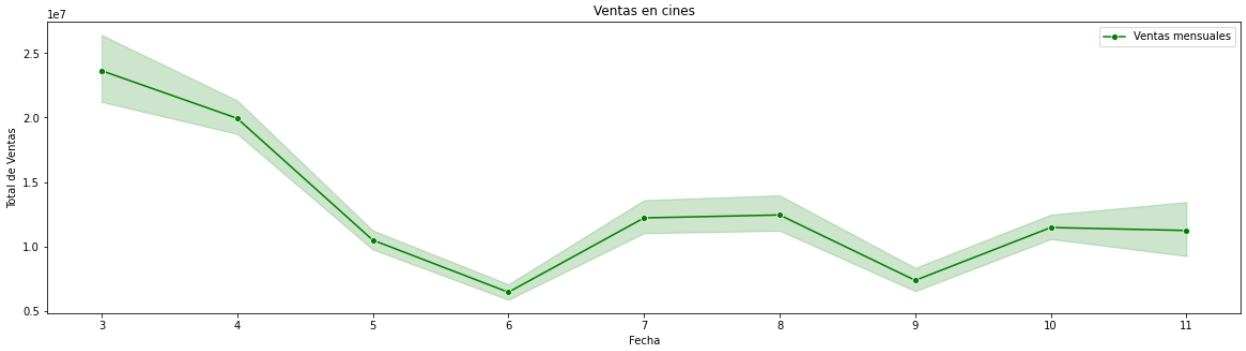
Después tenemos que hacer la lectura de nuestros datos los cuales se conforman de diferentes datos relacionados a diferentes cines como; el código de las películas proyectadas, los boletos vendidos, la ocupación de esa función, etc. todos estos datos se usaran en el entrenamiento para pronosticar el valor de ventas totales durante un determinado mes futuro.

Tenemos que leer los datos con ayuda de pandas el cual nos genera un `dataframe` con 142254 registros y 14 columnas asociadas a las características de los cines.

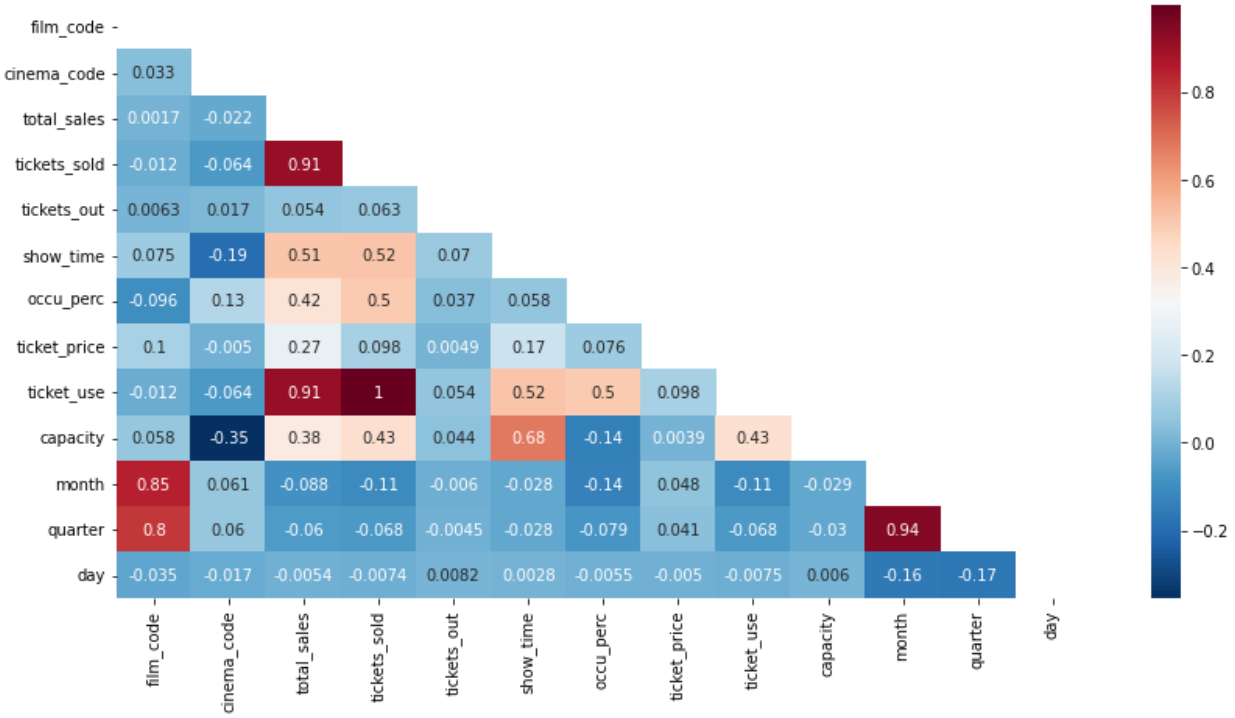
El conjunto de datos leídos se muestra a continuación.

	film_code	cinema_code	total_sales	tickets_sold	tickets_out	show_time	occu_perc	ticket_price	ticket_use	capacity	date	month	quarter	day
0	1492	304	3900000	26	0	4	4.26	150000.0	26	610.328638	2018-05-05	5	2	5
1	1492	352	3360000	42	0	5	8.08	80000.0	42	519.801980	2018-05-05	5	2	5
2	1492	489	2560000	32	0	4	20.00	80000.0	32	160.000000	2018-05-05	5	2	5
3	1492	429	1200000	12	0	1	11.01	100000.0	12	108.991826	2018-05-05	5	2	5
4	1492	524	1200000	15	0	3	16.67	80000.0	15	89.982004	2018-05-05	5	2	5

A continuación, se muestra la gráfica en la cual podemos ver las ventas mensuales de los cines durante las proyecciones de los meses de marzo a noviembre del año 2018.



Para las 14 variables en el modelo se decidió hacer un análisis para la selección de características trazando para ello un mapa de calor el cual nos muestra con un índice de -1 a 1 el nivel de correlación que tiene cada variable con las demás. A continuación, se muestra el mapa de calor obtenido resaltando las relaciones fuertes entre Month y Quarter porque ambos son un indicativo de la fecha, se eliminará Quarter, además de Tickets_sold y Ticket_use los cuales presentan exactamente el mismo valor por lo que se eliminara ticket_use, de igual manera Ticket_use con Total_sales presenta una correlación fuerte, sin embargo ya se ha eliminado Ticket_use.



Al final del análisis de correlación las variables que se seleccionaron fueron las siguientes.

1.- film_code [SE ELIMINA]

2.- cinema_code [SE CONSERVA] *

3- tota_sales [VARIABLE A PRONOSTICAR] *

4- tickets_sold [SE CONSERVA] *

5.- tickets_out [SE ELIMINA]

6.- show_time [SE CONSERVA] *

7.- occu_perc [SE CONSERVA] *

8.- ticket_price [SE CONSERVA] *

9.- ticket_use [SE ELIMINA]

10.- capacity [SE CONSERVA] *

11.- date [SE ELIMINA]

12.- month [SE CONSERVA] *

13.- quarter [SE ELIMINA]

14.- day [SE ELIMINA]

Por lo cual nos quedaremos con 7 variables que nos ayudaran a encontrar el valor de la variable pronostico total_sales.

Regresión Lineal Múltiple.

Para la creación del modelo de regresión lineal se utilizaron de las librerías de `linear_model` para generar el modelo, `mean_squared_error`, `max_error` y `r2_score` los cuales nos van a ayudar a generar las métricas para evaluar la efectividad del modelo una vez lo tengamos.

Habiéndose seleccionado la variable dependiente del modelo (X) la cual será el *total_sales* y cuáles serán las variables independientes (Y); cinema_code, tickets_sold, show_time, occu_perc, ticket_price y ticket_use. Después se hicieron 2 arreglos de `numpy` a los cuales llamaremos `X` y `Y`, sin embargo, tomar los más de 140, 000 registros pueden resultar en un desbordamiento de la memoria de Google colab por lo cual se decicio particionar aleatoriamente este conjunto de datos para tomar una muestra de solamente 20, 000 registros con la siguiente línea.

```
tickets_corto = tickets_info.dropna().sample(n=20000, random_state=1)
```

Con los conjuntos de datos de entrenamiento X y Y de 20, 000 registros cada uno se entreno el modelo de regresión lineal creando un objeto a partir del módulo importado y pasando los conjuntos de datos como parámetro al método `fit` para después generar nuevos pronósticos con `predict`.

```
RLMultiple = linear_model.LinearRegression()
RLMultiple.fit(X_train, Y_train)                #Se entrena el modelo
-----------------------
#Se genera el pronóstico
Y_pronostico = RLMultiple.predict(X_train)
pd.DataFrame(Y_pronostico)
```

Esto nos generara los valores que se pronostican para cada uno de los registros con los que se entrenó el modelo, sin embargo, debemos recordar que estos valores presentan cierto grado de error al no ajustarse totalmente al hiperplano.

Para la obtención de las métricas del modelo se comienza con los atributos `coef_` e `intercept_` del modelo generado (RLMultiple) pudiendo consultar cuales son los coeficientes de nuestro hiperplano el cual se va a ajustar a la mayoría de los registros siguiendo la siguiente ecuación.

$$\hat{Y} = -14928105.81811552 + 7432.40245(cinema\_code) + 101576.128(tickets\_sold) + 343244.148(show\_time) - 102047.097(occu\_perc) + 162.682795\,(ticket\_price) - 1384.65794(capacity) - 111591.241\,(month)$$

Sin embargo, aun faltaría agregar el residuo del modelo el cual lo podemos obtener con ayuda de la clase `max_error` que se encuentra en el paquete `sklearn.metrics` de Sklearn.

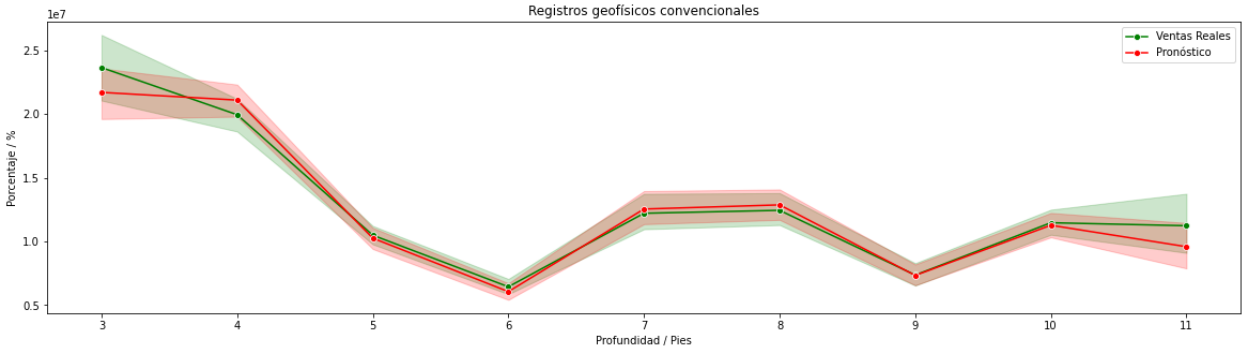
A continuación, se muestra el modelo de regresión lineal múltiple utilizado para hacer las predicciones.

$$\hat{Y} = -14928105.81811552 + 7432.40245(cinema\_code) + 101576.128(tickets\_sold) + 343244.148(show\_time) - 102047.097(occu\_perc) + 162.682795\,(ticket\_price) - 1384.65794(capacity) - 111591.241\,(month) + 331158958.1762$$

Donde 331158958.1762 representa el residuo, es decir una medida general la cual nos dice que tan diferentes son los valores pronosticados de los valores reales.

Además, podemos calcular el MSE y su raíz, el RMSE las cuales son dos métricas que nos indican en promedio que tantas unidades se alejan los valores pronosticados de los valores reales los cuales son 114047874971880.9375 y 10679319.9677 respectivamente. Estos valores se calcularon con la clase `mean_squared_error` de Sklearn, pareciera ser un error muy alto, pero recordemos que las ventas que se manejan mensuales de un cine son de millones.

Como ultima métrica se calculó el score que indica el porcentaje de precisión del modelo siendo un indicativo de su efectividad con nuevos valores el cual se logró de 87.02% que para el contexto en el que se está trabajando resulta ser un modelo muy efectivo. A continuación, se muestra una grafica donde se comparan los valores predichos (Linea roja) con los valores reales (línea azul) donde se observa que realmente no existe mucha diferencia entre ellos.



Arboles de Decisión.

Con las variables independientes del modelo seleccionadas se procedió con la aplicación del algoritmo, Para ello fue necesaria la clase `DecisionTreeRegressor` de `sklearn`, también se ocuparon `mean_squared_error`, `mean_absolute_error`, `r2_score` para el cálculo de los errores del modelo y por último `model_selection` para la partición del conjunto de entrenamiento y conjunto de pruebas.

Además de obtener el conjunto de 20, 000 registros, es necesario separarlo en uno de entrenamiento y otro para las pruebas, para ello se usó la librería de `sklearn model_selection` con su método `train_test_split` el cual recibe como datos el arreglo de `numpy` con la variable independiente del modelo (X), el arreglo de `numpy` con las variables dependientes (Y) , cuál será el tamaño en porcentaje del conjunto de pruebas, una semilla y el atributo `shuffle` el cual mezcla los datos de tal forma que no se tomen según la secuencia dada en un inicio. Como resultado ahora se tienen los 4 conjuntos 2 para entrenamiento y 2 para las pruebas con una proporción de 80% (16, 000 registros) para el entrenamiento y 20% (4, 000 registros) para las pruebas.

```
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
                                                                    test_size = 0.2,
                                                                    random_state = 1234,
                                                                    shuffle = True)
```

Con los conjuntos de datos establecidos, para generar el árbol es necesario utilizar la clase `DecisionTreeRegressor` para crear un objeto a partir de ella sobre la que es posible definir ciertos parámetros del árbol. En esta práctica se trabajó con `max_deep` podemos establecer la profundidad del árbol, con `min_samples_split` se define un mínimo de hojas que se tienen que generar y con `min_samples_leaf` podemos controlar el número de registros que definan una hoja de tal forma que los nodos hoja que no cumplan con este mínimo no se consideran para las reglas que se generen al final. Además, es necesario plantar una semilla de aleatoriedad y `criterion` la cual define la función que se usara para medir la calidad en las divisiones que presenten los nodos intermedios, para los árboles de esta práctica se usó el error absoluto (`absolute_error`).

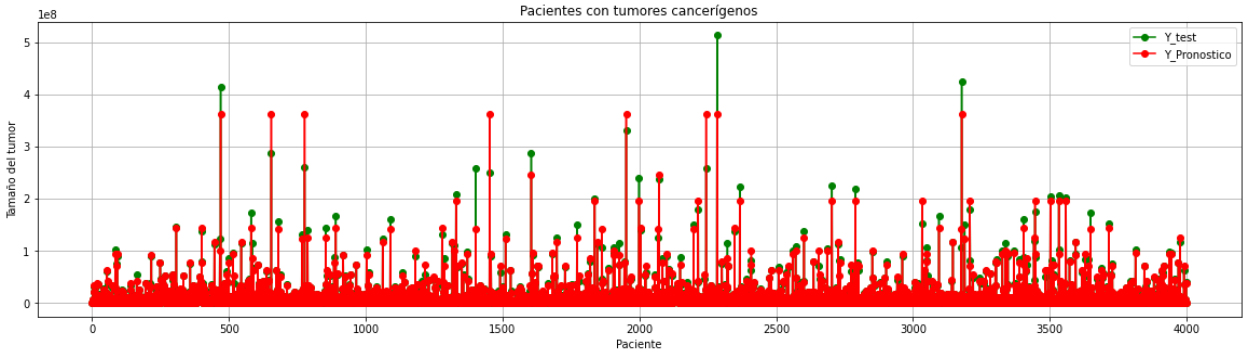
Una vez que se crea el objeto con los parámetros establecidos, se utiliza el método `fit` pasando como parámetros los conjuntos de datos de entrenamiento que corresponden al 80% de los registros. Para generar las métricas de rendimiento del modelo primero es necesario utilizar el método `predict` pasando como parámetros el conjunto de datos de prueba que corresponde a las variables independientes del modelo `X_test` para generar sus respectivos valores de área predichos los cuales se evaluaran con el conjunto de datos que tiene los valores reales para estos registros, es decir `Y_test`.

```
#ARBOL SIN MODIFICAR LOS HIPERPARAMETROS 99.38%
PronosticoAD = DecisionTreeRegressor(random_state=0)
```

```
#MODIFICANDO LOS HIPERPARAMETROS 96.36%
#La mitad de profundidad, por lo menos 2 hojas y mínimo 4 elementos por hoja
PronosticoAD = DecisionTreeRegressor(max_depth=10, min_samples_split=50,
min_samples_leaf=2, random_state=0, criterion = "absolute_error")
PronosticoAD.fit(X_train, Y_train)
```

El siguiente paso sería evaluar los parámetros del algoritmo primero generando un pronostico con los 4000 registros que se consideraron para el conjunto de pruebas y después calculando el score con la librería que se ha venido trabajando r2_score. Se obtuvo un porcentaje de precisión con este algoritmo del 99.38% lo cual es un fuerte indicativo de que se dio un sobreajuste, para solucionarlo se modificaron los hiperparametros para una altura máxima de 10 niveles, por lo menos 50 registros en la misma hoja para que se hiciera una separación y mínimo 2 hojas obteniendo un score de 96.36% el cual se tomó como modelo para hacer los nuevos pronósticos.

A modo de comparación se generó una gráfica donde se pueden ver los valores predichos para total de ventas en color rojo y los valores reales de las ventas en verde.



Se puede apreciar que el modelo se comporta bien pronosticando sobre todo los valores de ventas que son pequeños donde la diferencia no es mucha y comportándose de una peor manera pronosticando valores atípicos, pero esto en parte también se debe a que gana la capacidad de generalizar mejor los datos.

Para las dos configuraciones presentadas se generaron diferentes arboles partiendo de diferentes variables las cuales se detallan, pare generar las gráficas de los árboles y el conjunto de reglas fue necesario trabajar con los módulos `export_graphviz`, `plot_tree` y `export_text` de `sklearn.tree`.

Cervantes Rubí Brandon

Árbol con todas las variables del modelo.

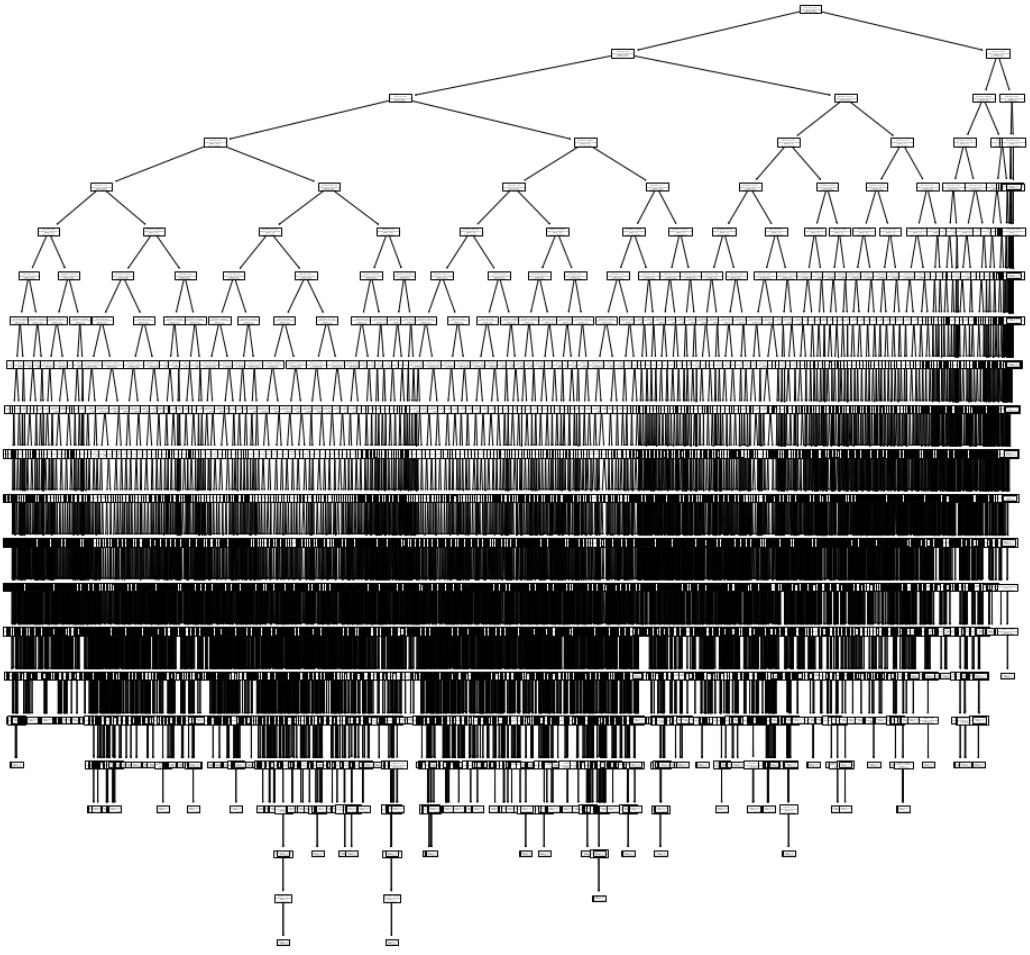
Nodo del que se parte.

```
tickets_sold <= 734.5
squared_error = 899489545034252.1
samples = 16000
value = 12392676.827
```

	Variable	Importancia
1	tickets_sold	0.884745
4	ticket_price	0.111245
5	capacity	0.002569
3	occu_perc	0.000666
6	month	0.000389
0	cinema_code	0.000358
2	show_time	0.000027

El árbol generado presenta 21 niveles y el nodo raíz parte del atributo boletos vendidos con un score del 99.38% tardando alrededor de 10 minutos en proyectarse.

Los boletos vendidos es la variable que presenta un mayor nivel de entropía o que mayor importancia toma para el modelo, por el contrario, la menos importante es el tiempo de pantalla.



Reporte de parámetros.

```
MAE: 343164.3660
MSE: 4915927179474.3457
RMSE: 2217189.0266
Score: 0.9938
```

Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 22/05/2022
Practica No.15 Pronostico.

Árbol con las variables seleccionadas del análisis de correlación.

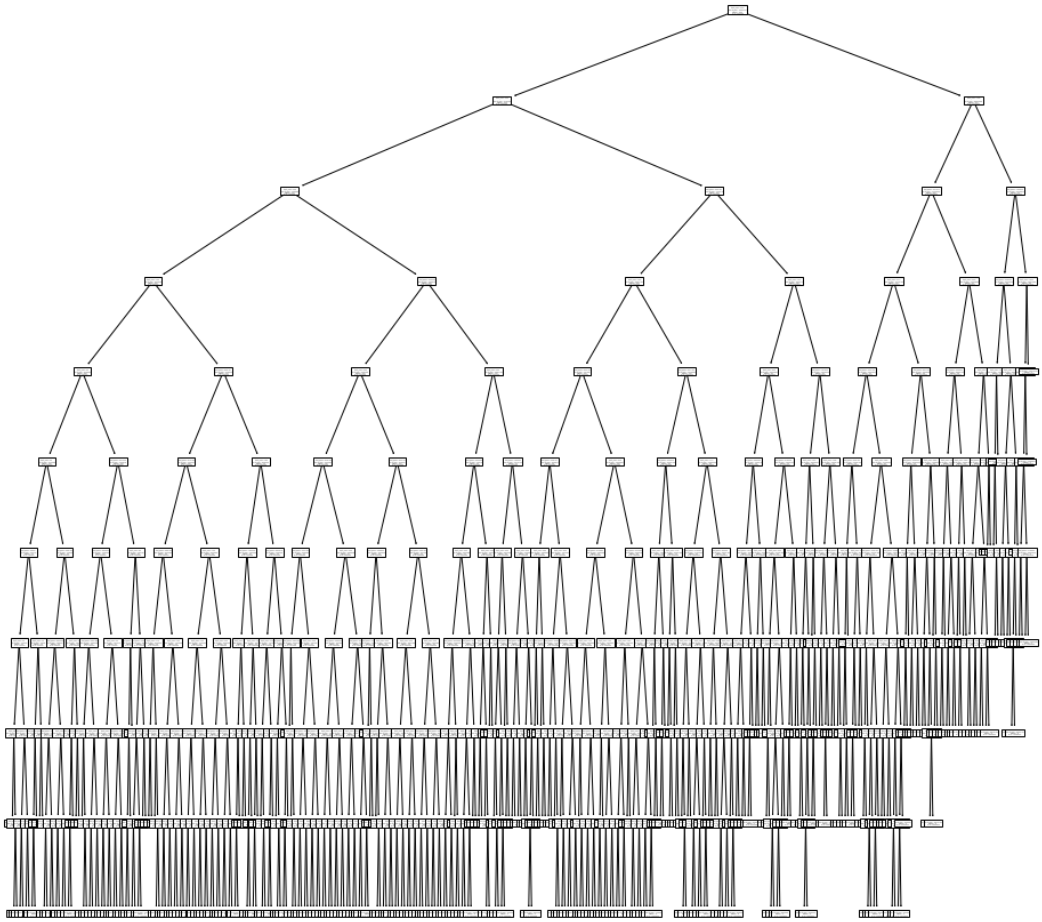
Nodo del que se parte.

```
tickets_sold <= 235.5
absolute_error = 10911571.577
samples = 16000
value = 3760000.0
```

	Variable	Importancia
1	tickets_sold	0.776509
4	ticket_price	0.223491
0	cinema_code	0.000000
2	show_time	0.000000
3	occu_perc	0.000000
5	capacity	0.000000
6	month	0.000000

El árbol generado presenta 10 niveles y el nodo raíz parte los boleros vendidos con un score del 96.36%.

Los boletos vendidos es la variable que presenta un mayor nivel de entropía o mayor importancia toma para el modelo, por el contrario, la menos importante es el mes.



Reporte de parámetros.

```
MAE: 919539.4982
MSE: 28900889860711.0000
RMSE: 5375954.7860
Score: 0.9636
```

Con la función `export_test` podemos generar una lista de reglas las cuales definen la estructura de divisiones que va a tomar el árbol de decisión para hacer los nuevos pronósticos, si la imprimimos podremos visualizar cada uno de los limites que dictan la división en los nodos de la siguiente manera.

```
|--- tickets_sold <= 734.50
|   |--- tickets_sold <= 235.50
|   |   |--- tickets_sold <= 89.50
|   |   |   |--- tickets_sold <= 38.50
|   |   |   |   |--- tickets_sold <= 17.50
|   |   |   |   |   |--- tickets_sold <= 10.50
|   |   |   |   |   |   |--- tickets_sold <= 6.50
|   |   |   |   |   |   |   |--- ticket_price <= 81666.67
|   |   |   |   |   |   |   |   |--- tickets_sold <= 3.50
|   |   |   |   |   |   |   |   |   |--- tickets_sold <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- tickets_sold <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 5
|   |   |   |   |   |   |   |   |   |   |--- tickets_sold >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 6
```

Bosques aleatorios.

Con los conjuntos de datos establecidos, para generar el bosque aleatorio de árboles de regresión es necesario utilizar la clase `RandomForestRegressor` para crear un objeto a partir de ella sobre la que es posible definir ciertos parámetros de los árboles que conformaran al bosque. Los hiperparametros con los que se trabajó fueron los siguientes; Con `n_estimators` podemos definir el número de árboles que conformaran el bosque, con `max_deep` podemos establecer la profundidad del árbol, con `min_samples_split` se define un mínimo de hojas que se tienen que generar, con `min_samples_leaf` podemos controlar el número de registros que definan una hoja de tal forma que los nodos hoja que no cumplan con este mínimo no se consideran para las reglas que se generen al final y por ultimo tenemos max_features para que cada árbol tome el número de variables especificadas correspondientes a las más importantes.

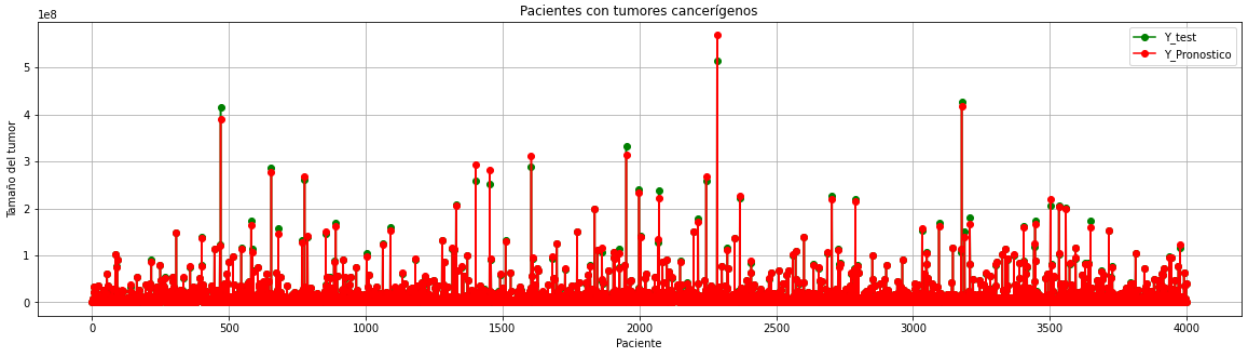
```
#CONF 1 99.68%
PronosticoBA = RandomForestRegressor(random_state=0)

#CONF 2 97.94%
PronosticoBA = RandomForestRegressor(n_estimators=100, max_depth=10,
min_samples_split=50, min_samples_leaf=2, random_state=0, max_features=5)

PronosticoBA.fit(X_train, Y_train)
```

Cuando se crea el objeto con los parámetros establecidos, se usa el método `fit` pasando como parámetros los conjuntos de datos de entrenamiento que corresponden al 80% de los registros generado en el algoritmo de árboles aleatorios. Para obtener las métricas de rendimiento del modelo primero es necesario utilizar el método `predict` pasando como parámetros el conjunto de datos de prueba que corresponde a las variables independientes del modelo `X_test` para generar los valores del área predichos por el mismo (`Y_pronostico`) y comparar los resultados con el conjunto de datos que tiene los valores reales para estos registros, es decir `Y_test`.

Para obtener la precisión del modelo se utilizó la función r2_score pasando como parámetros los conjuntos de los valores predichos para el conjunto de pruebas y sus valores reales obteniendo diferentes valores de precisión.

```
r2_score(Y_test, Y_Pronostico)
```

para la configuración sin modificar los hiperparametros se obtuvo un valor del 99.69% y para la configuración con 100 estimadores, una profundidad de 10, mínimo 50 registros en las hojas para que se dividan, mínimo 2 nodos hoja y considerando solo las 5 variables más importantes se obtuvo un valor de precisión del 97.94%Considerando una altura de 8, mínimo 4 nodos hoja con 2 registros en cada uno y variando el número de árboles que compone el bosque aleatorio se llegaron a las siguientes métricas de score, este último modelo fue el que se seleccionó para generar nuevos pronósticos. La siguiente grafica muestra una comparación de como es que se comporta el modelo prediciendo los valores de prueba.



Con el atributo `feature_importances_` podemos generar una lista ordenada con la importancia de cada una de las variables que se consideraron, a partir de esta lista se creó un `dataframe` ordenado por valor de importancia en el que podemos observar variaciones dependiendo de la configuración y pudiendo tomar las más altas configurando el atributo `max_features`.

Para las dos configuraciones presentadas se generaron diferentes bosques aleatorios partiendo de diferentes valores para los hiperparametros, se generaron las gráficas del último árbol y fue necesario trabajar con los módulos `export_graphviz`, `plot_tree` y `export_text` de `sklearn.tree`.

Cervantes Rubí Brandon

Bosque con todas las variables del modelo.

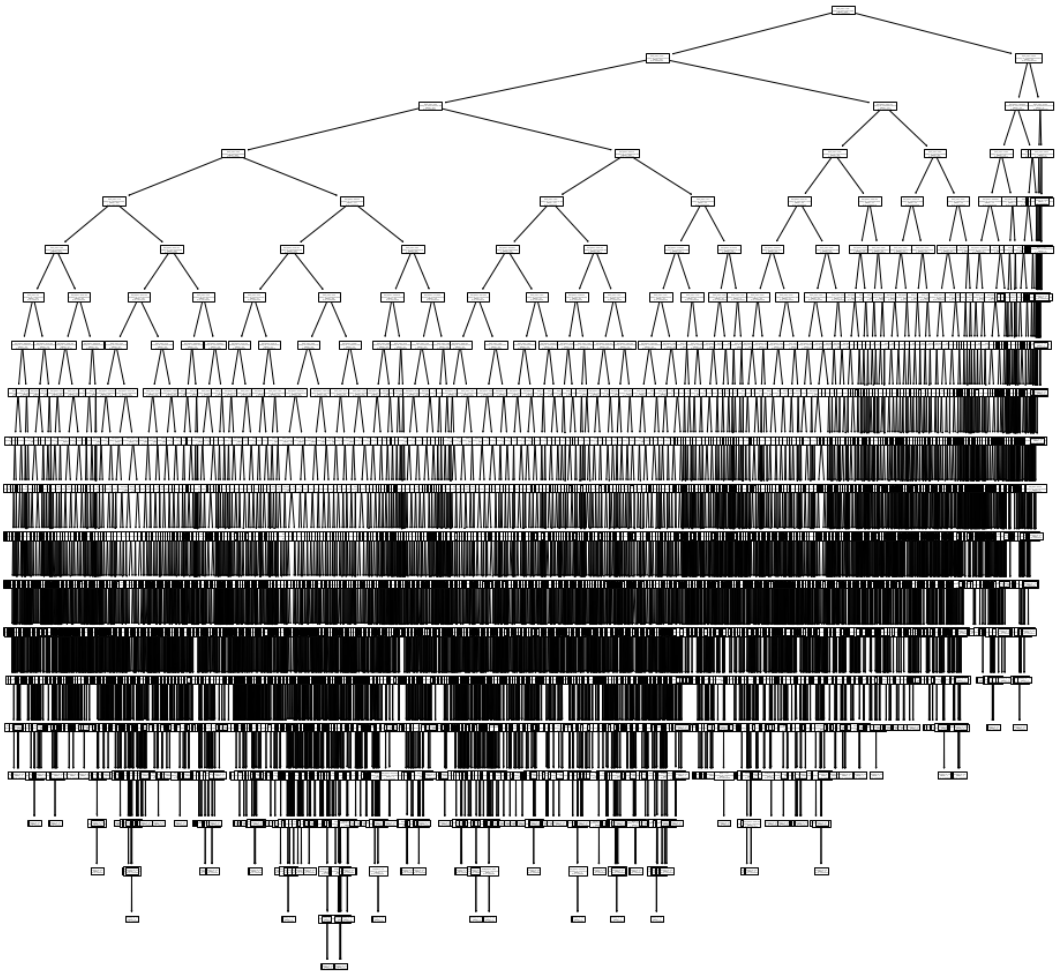
	Variable	Importancia
1	tickets_sold	0.869574
4	ticket_price	0.114552
2	show_time	0.006825
5	capacity	0.003750
3	occu_perc	0.003443
0	cinema_code	0.001128
6	month	0.000727

Nodo del que se parte.

tickets_sold <= 734.5
squared_error = 888975220661435.0
samples = 10096
value = 12306205.863

El árbol generado presenta 19 niveles y el nodo raíz parte de los boletos vendidos con un score del 99.69%.

Boletos vendidos es la variable que presenta un mayor nivel de entropía o que mayor importancia toma para el modelo, por el contrario, la menos importante es el mes de la proyección.



Reporte de parámetros.

```
MAE: 214970.5238
MSE: 2481802379156.2832
RMSE: 1575373.7268
Score: 0.9969
```

Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 22/05/2022
Practica No.15 Pronostico.

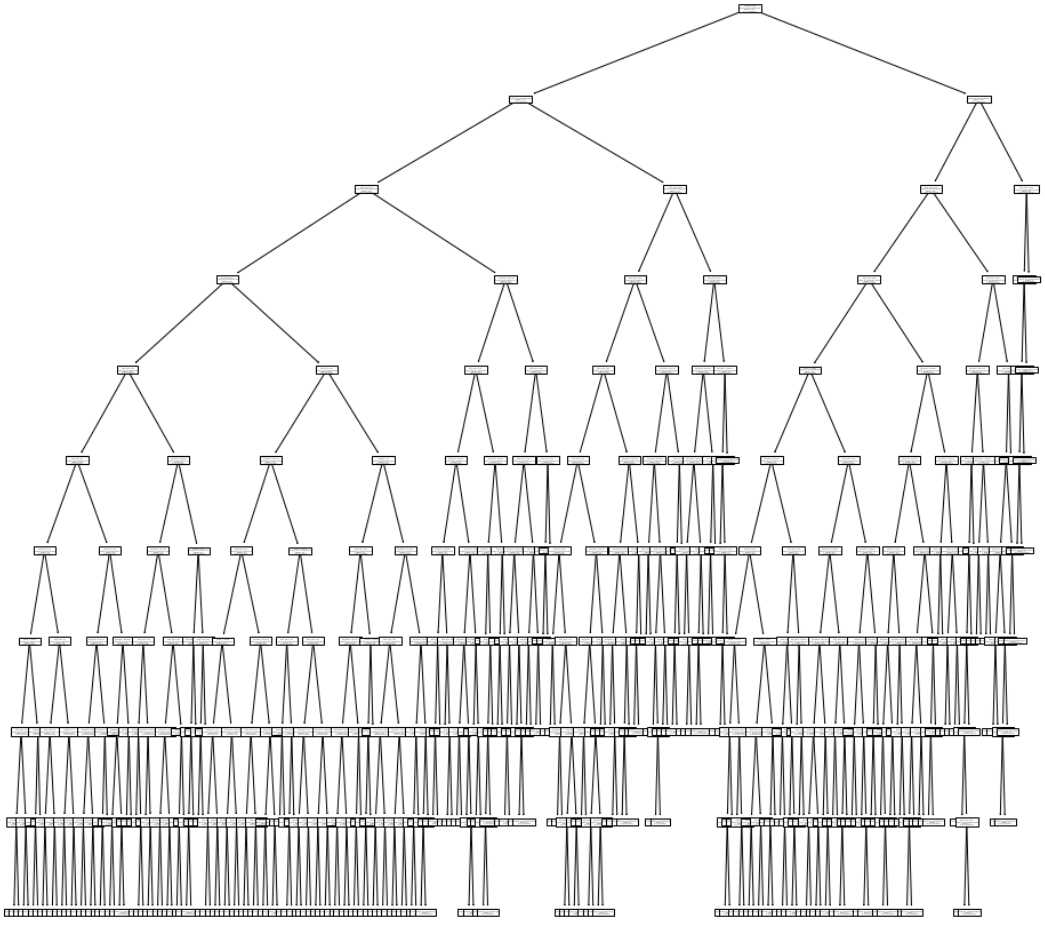
Bosque aleatorio con modificaciones en los hiperparametros.

Nodo del que se parte.

```
occu_perc <= 25.93
squared_error = 888975220661435.0
samples = 10096
value = 12306205.863
```

	Variable	Importancia
1	tickets_sold	0.772622
4	ticket_price	0.098398
2	show_time	0.067596
3	occu_perc	0.039871
5	capacity	0.019764
0	cinema_code	0.001650
6	month	0.000098

La altura del árbol presentado es de 10, el nodo raíz del que se parte es la ocupación con un score final del 97.94%.



Reporte de parámetros.

```
MAE: 737482.4295
MSE: 16386324475289.6895
RMSE: 4048002.5291
Score: 0.9794
```

Utilizando la librería `export_test` es posible generar la lista de reglas en forma textual que componen al algoritmo para darnos una idea de cómo se están haciendo las divisiones en cada uno de los nodos presentados en el árbol del bosque. A continuación, se muestra una sección de la lista de reglas las cuales definen al árbol con una altura de 8 niveles, 150 estimadores, mínimo 4 nodos hoja y 2 registros en cada uno de los nodos hoja.

```
|--- occu_perc <= 25.93
|   |--- capacity <= 1309.83
|   |   |--- tickets_sold <= 67.50
|   |   |   |--- occu_perc <= 4.14
|   |   |   |   |--- tickets_sold <= 14.50
|   |   |   |   |   |--- tickets_sold <= 8.50
|   |   |   |   |   |   |--- occu_perc <= 0.75
|   |   |   |   |   |   |   |--- capacity <= 845.24
|   |   |   |   |   |   |   |   |--- tickets_sold <= 2.50
|   |   |   |   |   |   |   |   |   |--- ticket_price <= 85000.00
|   |   |   |   |   |   |   |   |   |   |--- value: [105576.92]
|   |   |   |   |   |   |   |   |   |--- ticket_price >  85000.00
|   |   |   |   |   |   |   |   |   |   |--- value: [213170.73]
```

Por último, es posible hacer nuevas predicciones con cada uno de los bosques aleatorios que se generaron teniendo la certeza de que el modelo se comporta de buena manera con nuevos datos por tener un score superior al 97%.

Nuevos pronósticos.

Por último, es importante tener en cuenta que con este modelo podemos llegar a hacer nuevas extrapolaciones ingresando nuevos valores para cada una de las variables independientes que seleccionamos para el modelo en un inicio, los cuales en este caso serían las características con las que presenta el cine.

Para probar los tres modelos ingresamos datos para un código de cine de 489, boletos vendidos de 32, tiempo de pantalla de 4, ocupación del cine de 20, precios de los boletos de 80000, una capacidad de 168 y el mes 5 correspondiente a Mayo el cual tiene un valor real de ventas totales de 2,560,000. Se obtuvieron los siguientes resultados:

	Regresión Lineal Múltiple (87.02%)	Aboles aleatorios (96.36%)	Bosques Aleatorios (97.94%)
Valor pronosticado	3,523,931.83796487	2,720,000	2,527,316.01144735
Error con respecto al valor real	963,931.83796487	160,000	-32,683.98855265

Conclusiones.

En esta práctica se generaron tres modelos que atendían un problema de pronostico para el cual se tenían que proyectar las ventas totales de meses futuros considerando las ventas de diferentes cines en las fechas de marzo a noviembre del 2018. Para los modelos se planteó un análisis de correlación de variables el cual redujo la dimensionalidad del modelo en la mitad de 14 a 7.

El primer modelo fue la regresión lineal múltiple con el que se obtuvo un porcentaje de precisión del 87.02% siendo el mas bajo de todos, seguido de esto se generó un árbol aleatorio el cual sin modificar los hiperparametros presentaba sobreajuste (Score arriba del 99%) por lo cual se modificaron los hiperparametros como la altura máxima, el criterio de división, etc. obteniendo una precisión final de 96.36% la cual se considera aceptable para el contexto que se trabaja. Por ultimo se genero un modelo de bosques aleatorios y sucedió un fenómeno similar que con los arboles aleatorios se tuvieron que modificar los hiperparametros para eliminar el sobreajuste a los datos.

Fuentes.

pandas documentation — pandas 1.4.1 documentation. (2022). Pandas. 2022, de https://pandas.pydata.org/docs/index.html#

sklearn.tree.DecisionTreeRegressor. (2022). Scikit-Learn. 2022, de https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

sklearn.ensemble.RandomForestRegressor. (2022). Scikit-Learn. 2022, de https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

User guide: contents. (2022). Scikit-Learn. 2022, de https://scikit-learn.org/stable/user_guide.html

Dataset: https://www.kaggle.com/datasets/arashnic/cinema-ticket