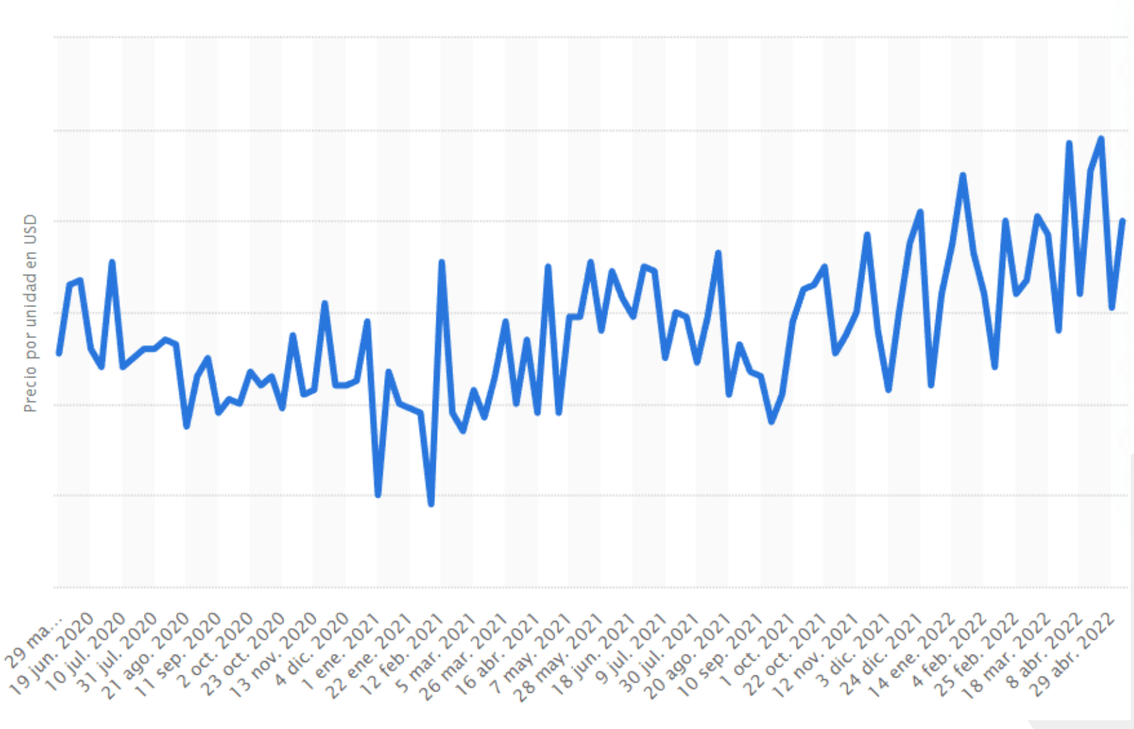
Predictor de precios de aguacate en Estados unidos

Lizeth Fernanda Ropero Cárdenas -2204001
Juan Camilo Guerrero Ortega - 2183266
Faiber Stiven Angarita Mendoza - 2191963

Contenidos

1 Planteamiento del problema

Nuestro problema se basa en las fluctuaciones en los precios del aguacate en USA, estas fluctuaciones generan problemas de sobreproducción y escases afectando a empresas y comunidades que no pueden acceder al alimento.



2 Objetivo general

- Desarrollar un sistema de IA basado en modelos de aprendizaje que pueda predecir el precio del aguacate

Objetivo especificos

- Entrenar varios modelos basados en regresión lineal, para comparar y obtener el mejor prediciendo el precio del aguacate

- Predecir el precio de los aguacates a partir de los datos proporcionados por el dataset en Estados Unidos

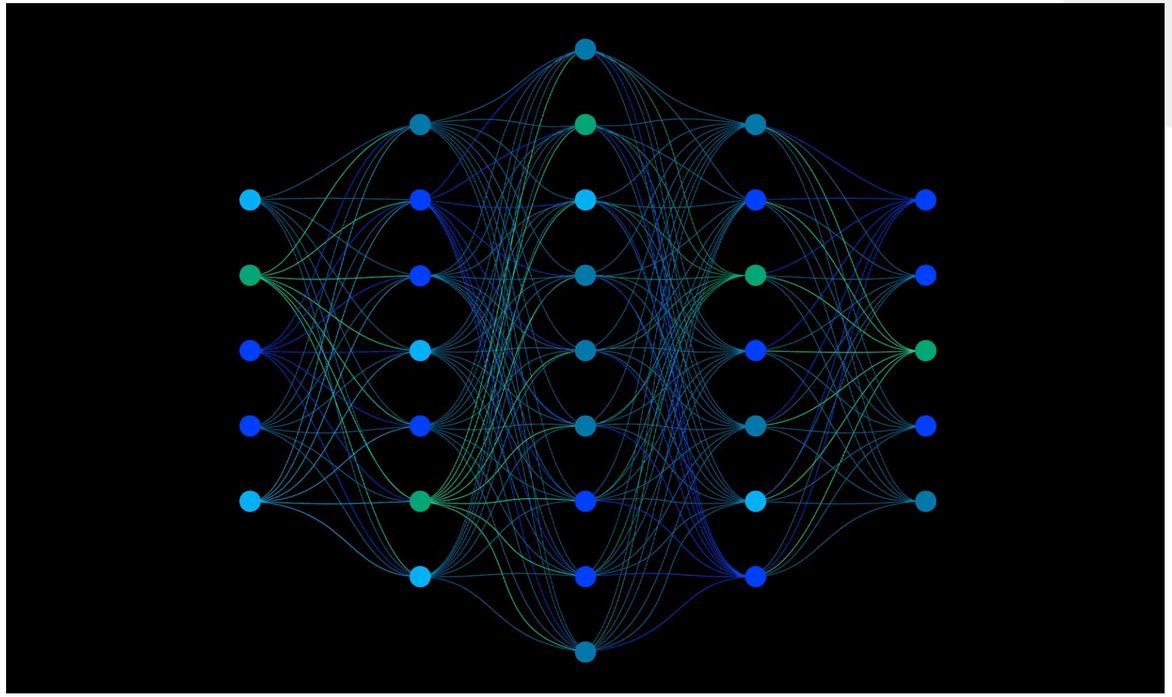
Dataset utilizado

https://www.kaggle.com/datasets/neuromusic/avocado-prices

4046 : Aguacate Hass estándar
4225:Aguacate Hass grande
4770: Fuerte. Bacon, Zutano

Modelos

- Decision Tree Regressor
- Random Forest Regressor
- Redes neuronales
  - -Standard Scaler
  - -Dropout



Decision Tree Regressor

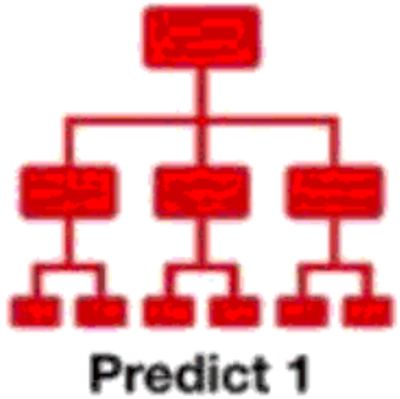
```
s = cross_val_score(regressor, np.array([df_avocado['Total Volume']]).T, df_avocado['AveragePrice'], cv=KFold(10, shuffle=True), scoring=make_scorer(mean_squared_error
print ("MSE depth: 2 %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))

s = cross_val_score(regressorSA, np.array([df_avocado['Total Volume']]).T, df_avocado['AveragePrice'], cv=KFold(10, shuffle=True), scoring=make_scorer(mean_squared_erro
print ("MSE depth: 30 %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
```

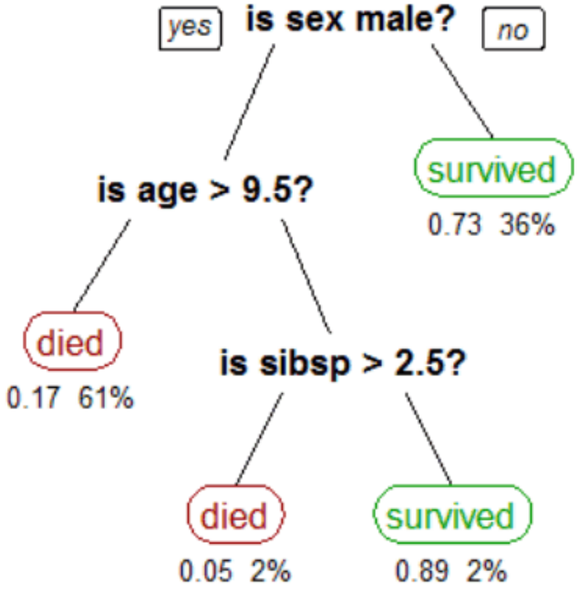
```
depth: 2 0.105 (+/- 0.00367)
depth: 30 0.109 (+/- 0.00461)
```

Tiempo de ejecución 5 segundos

y

Max Depth de 2 y 10





Random Forest Regressor

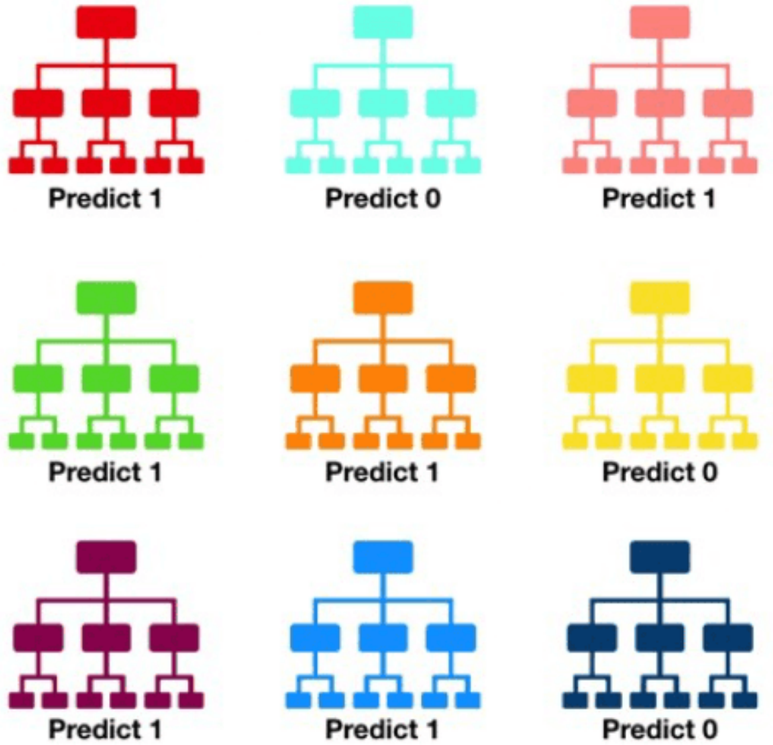
```
s = cross_val_score(regressor_1, np.array([df_avocado['Total Volume']]).T, df_avocado['AveragePrice'], cv=KFold(10, shuffle=True), scoring=make_scorer(mean_squared_error))
print ("MSE depth: 1 %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))

s = cross_val_score(regressor_6, np.array([df_avocado['Total Volume']]).T, df_avocado['AveragePrice'], cv=KFold(10, shuffle=True), scoring=make_scorer(mean_squared_error))
print ("MSE depth: 6 %.3f (+/- %.5f)"%(np.mean(s), np.std(s)))
```

```
MSE depth: 1 0.110 (+/- 0.00197)
MSE depth: 6 0.102 (+/- 0.00349)
```

Tiempo de ejecución 22 segundos

Max depth de 1 y 6

Random State de 100



Red neuronal Sencilla

Tiempo de ejecución
53 segundos

1.layers.Dense(128, activation='relu',)
2.layers.Dense(64, activation='relu'),
3.layers.Dense(1)

```
365/365 [==============================] - 1s 3ms/step - loss: 445391.0312 - val_loss: 146152.3906
Epoch 37/50
365/365 [==============================] - 1s 4ms/step - loss: 157116.0938 - val_loss: 102459.8906
Epoch 38/50
365/365 [==============================] - 1s 4ms/step - loss: 208917.0625 - val_loss: 116063.3125
Epoch 39/50
365/365 [==============================] - 1s 3ms/step - loss: 354010.9688 - val_loss: 141707.4688
Epoch 40/50
365/365 [==============================] - 1s 2ms/step - loss: 191178.3906 - val_loss: 58376.6133
Epoch 41/50
365/365 [==============================] - 1s 2ms/step - loss: 5252085.0000 - val_loss: 1823353.8750
Epoch 42/50
365/365 [==============================] - 1s 2ms/step - loss: 233900.5312 - val_loss: 99243.5078
Epoch 43/50
365/365 [==============================] - 1s 2ms/step - loss: 79842.5391 - val_loss: 88691.3281
Epoch 44/50
365/365 [==============================] - 1s 2ms/step - loss: 52165.7773 - val_loss: 45149.6641
Epoch 45/50
365/365 [==============================] - 1s 2ms/step - loss: 38974.0586 - val_loss: 37617.4805
Epoch 46/50
365/365 [==============================] - 1s 2ms/step - loss: 67264.7031 - val_loss: 142724.6406
Epoch 47/50
365/365 [==============================] - 1s 3ms/step - loss: 47901.0508 - val_loss: 44808.6914
Epoch 48/50
365/365 [==============================] - 1s 4ms/step - loss: 30148.8301 - val_loss: 25063.0762
Epoch 49/50
365/365 [==============================] - 1s 4ms/step - loss: 135103.2344 - val_loss: 46559.5234
Epoch 50/50
365/365 [==============================] - 1s 4ms/step - loss: 21563.4160 - val_loss: 16492.5781
115/115 [==============================] - 0s 2ms/step - loss: 16214.7051
Test MSE: 16214.705078125
```

Red neuronal usando StandardScaler

Tiempo de ejecución
1 minuto 25 segundos

1.layers.Dense(128,
2.activation='relu')
3.layers.Dense(64,
activation='relu')
4.layers.Dense(1)

```
Epoch 34/50
365/365 [==============================] - 1s 2ms/step - loss: 1.4774e-04 - val_loss: 5.5803e-05
Epoch 35/50
365/365 [==============================] - 1s 2ms/step - loss: 1.7762e-04 - val_loss: 1.1039e-04
Epoch 36/50
365/365 [==============================] - 1s 2ms/step - loss: 4.5201e-05 - val_loss: 3.9359e-05
Epoch 37/50
365/365 [==============================] - 1s 2ms/step - loss: 1.4641e-04 - val_loss: 9.0649e-05
Epoch 38/50
365/365 [==============================] - 1s 2ms/step - loss: 1.3697e-04 - val_loss: 9.9553e-05
Epoch 39/50
365/365 [==============================] - 1s 2ms/step - loss: 1.0130e-04 - val_loss: 1.0910e-04
Epoch 40/50
365/365 [==============================] - 1s 4ms/step - loss: 6.0097e-05 - val_loss: 5.8214e-05
Epoch 41/50
365/365 [==============================] - 1s 3ms/step - loss: 6.4730e-05 - val_loss: 1.6587e-04
Epoch 42/50
365/365 [==============================] - 1s 4ms/step - loss: 1.4656e-04 - val_loss: 1.2343e-04
Epoch 43/50
365/365 [==============================] - 1s 3ms/step - loss: 8.2505e-05 - val_loss: 4.3594e-05
Epoch 44/50
365/365 [==============================] - 1s 2ms/step - loss: 4.0572e-05 - val_loss: 6.1744e-05
Epoch 45/50
365/365 [==============================] - 1s 2ms/step - loss: 3.0473e-05 - val_loss: 2.4828e-04
Epoch 46/50
365/365 [==============================] - 1s 2ms/step - loss: 3.7640e-05 - val_loss: 1.5468e-05
Epoch 47/50
365/365 [==============================] - 1s 2ms/step - loss: 5.9914e-05 - val_loss: 1.8635e-05
Epoch 48/50
365/365 [==============================] - 1s 2ms/step - loss: 6.0512e-05 - val_loss: 2.2543e-05
Epoch 49/50
365/365 [==============================] - 1s 2ms/step - loss: 5.4630e-05 - val_loss: 1.8859e-05
Epoch 50/50
365/365 [==============================] - 1s 2ms/step - loss: 2.8191e-05 - val_loss: 8.1404e-06
115/115 [==============================] - 0s 1ms/step - loss: 1.1835e-05
Test MSE: 1.1835090845124796e-05
```

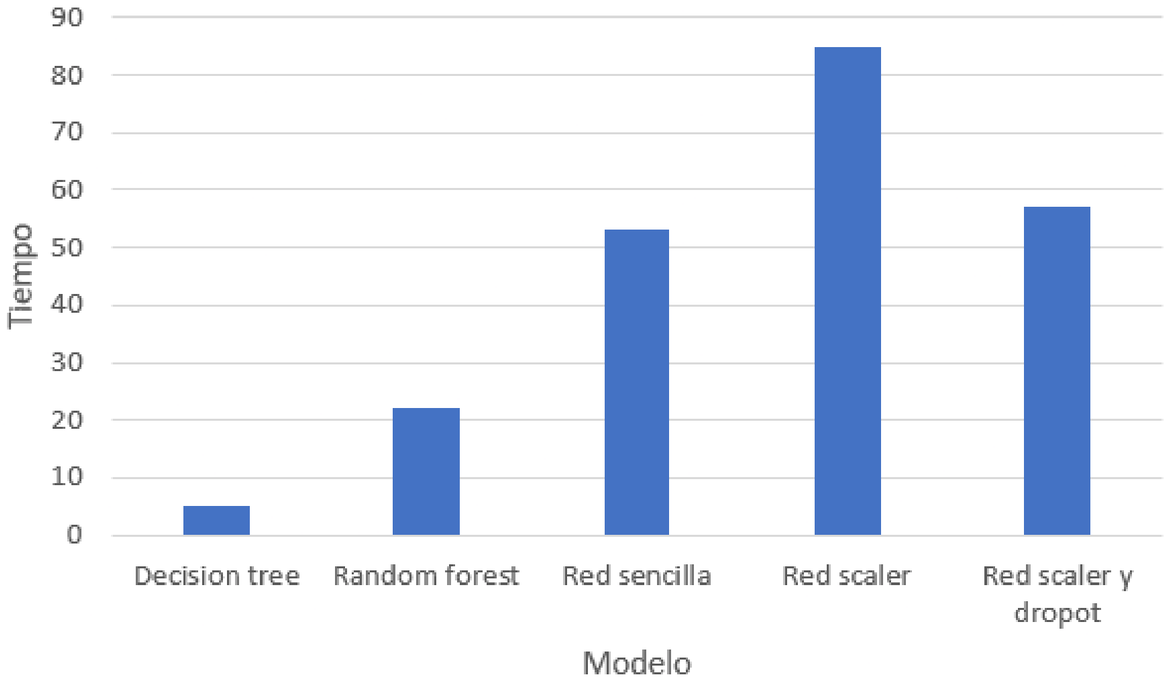
Red neuronal usando StandardScaler y Dropout

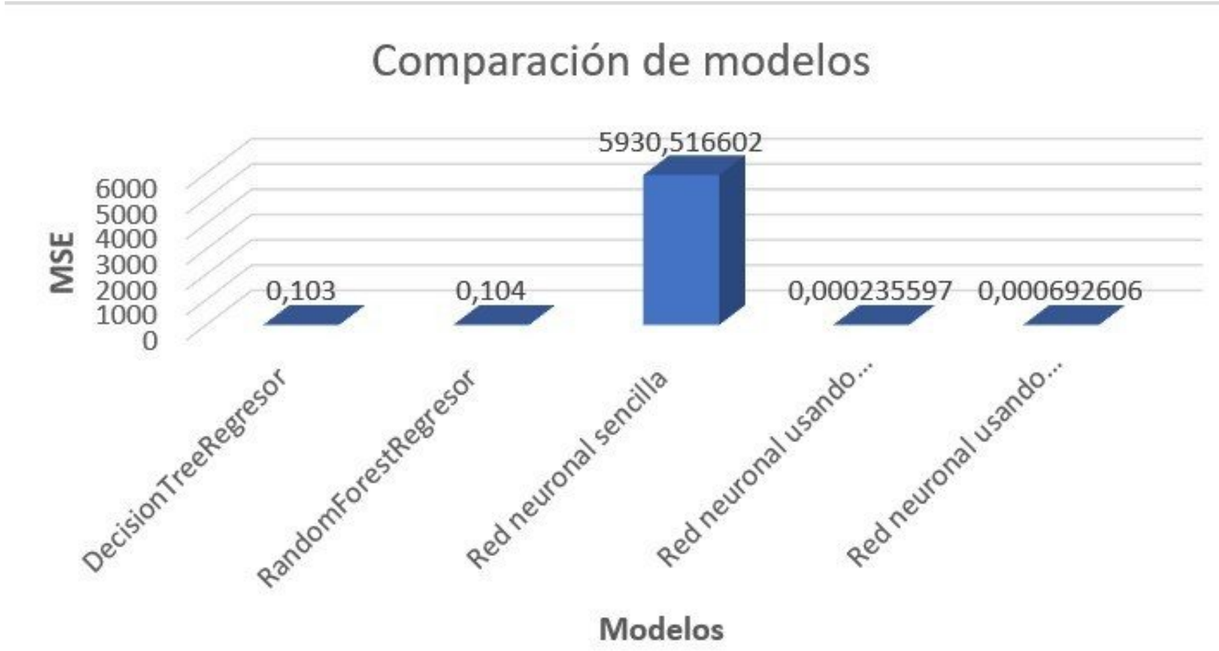
Tiempo de ejecución
57 segundos

1.layers.Dense(128, activation='relu')
2.layers.Dropout(0.2)
3.layers.Dense(64,
4.activation='relu')
5.layers.Dropout(0.2)
6.layers.Dense(1)

```
Epoch 38/50
365/365 [==============================] - 1s 3ms/step - loss: 0.0022 - val_loss: 7.1219e-04
Epoch 39/50
365/365 [==============================] - 1s 2ms/step - loss: 0.0024 - val_loss: 2.5070e-04
Epoch 40/50
365/365 [==============================] - 1s 2ms/step - loss: 0.0024 - val_loss: 3.0964e-04
Epoch 41/50
365/365 [==============================] - 1s 3ms/step - loss: 0.0023 - val_loss: 3.5134e-04
Epoch 42/50
365/365 [==============================] - 1s 3ms/step - loss: 0.0023 - val_loss: 8.6720e-04
Epoch 43/50
365/365 [==============================] - 1s 3ms/step - loss: 0.0025 - val_loss: 5.5612e-04
Epoch 44/50
365/365 [==============================] - 1s 3ms/step - loss: 0.0023 - val_loss: 2.0782e-04
Epoch 45/50
365/365 [==============================] - 1s 2ms/step - loss: 0.0024 - val_loss: 2.0203e-04
Epoch 46/50
365/365 [==============================] - 1s 2ms/step - loss: 0.0023 - val_loss: 2.0956e-04
Epoch 47/50
365/365 [==============================] - 1s 3ms/step - loss: 0.0022 - val_loss: 6.0315e-05
Epoch 48/50
365/365 [==============================] - 1s 3ms/step - loss: 0.0022 - val_loss: 7.9760e-04
Epoch 49/50
365/365 [==============================] - 2s 4ms/step - loss: 0.0022 - val_loss: 6.3293e-04
Epoch 50/50
365/365 [==============================] - 2s 4ms/step - loss: 0.0024 - val_loss: 1.6765e-04
115/115 [==============================] - 0s 3ms/step - loss: 1.9887e-04
Test MSE: 0.0001988701696973294
```

Comparativa entre los modelos





Conclusión

Mediante el entrenamiento de diferentes modelos se puede concluir que el mejor son las redes neuonales usando StandardScaler y Dropout.

Gracias!