Diamonds Showcase

Group 1: Cindy Detraz, James Dreussi, McKala Krauss, Samuel Neal

Goal

Engagement with diamonds

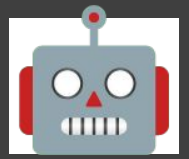
Approach

Interactive diamond experience

Result

Diamonds Showcase

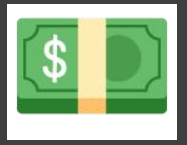
CHATBOT



DATA PREPROCESSING

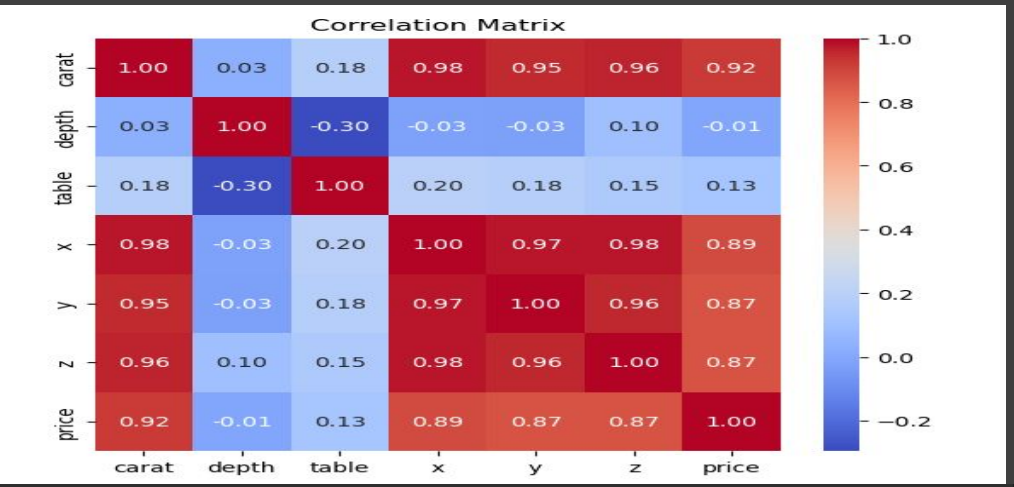
- Created API keys for Pinecone to vectorize text and OpenAI for embedding / response.
- Gathered text on diamonds and combine the results to vectorize.
- Used a function to answer the list of questions and return results with matching score.
- Matching score will also answer questions with a text response that users prompt

PREDICTIVE MODELING



DATA PREPROCESSING

- Dataset from Kaggle with 53,939 entries
  - price, carat, cut, color, clarity, depth, table, price, length, width, height
- No nulls; 35 - ‘0’s in dimensions > entries removed
- Extra index row removed
- Categorical: Cut, color, clarity > LabelEncoder
- All values scaled with MinMaxScaler



	carat	depth	table	price	x	y	z
t	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000	53920.000000
n	0.797698	61.749514	57.456834	3930.993231	5.731627	5.734887	3.540046
d	0.473795	1.432331	2.234064	3987.280446	1.119423	1.140126	0.702530
n	0.200000	43.000000	43.000000	326.000000	3.730000	3.680000	1.070000
%	0.400000	61.000000	56.000000	949.000000	4.710000	4.720000	2.910000
%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
%	1.040000	62.500000	59.000000	5323.250000	6.540000	6.540000	4.040000
x	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

DATA EXPLORATION

Categorical

Anova: Cut, Color, Clarity $p<0.05$
All 3 significantly influence or associated with price

Continuous

Highly correlated w/price except table and depth

APPROACH - TENSORFLOW

- Removed price from data frame (X, y)
- Split data into training and testing sets
- Multiple classical ML algorithms with thorough optimization of two

  High performance; overfit
- Optimized DNN outperforms
- Gradio Interface: User input -> $ prediction

```
callbacks=[keras.callbacks.EarlyStopping(monitor='loss', patience=10)])
```

PERFORMANCE - TENSORFLOW

	Algorithm	R2 Train	R2 Test	MAE Train	MAE Test
0	Optimized Neural Network	0.98	0.98	302.85	308.55
1	Optimized ExtraTreesRegressor	1.00	0.98	0.41	285.89
2	Optimized KNeighborsRegressor	1.00	0.97	0.40	320.24
3	LinearRegression	0.88	0.89	863.41	849.41
4	KNeighborsRegressor	0.97	0.97	326.66	367.24
5	RandomForestRegressor	1.00	0.98	101.91	266.74
6	ExtraTreesRegressor	1.00	0.98	0.40	263.33
7	AdaBoostRegressor	0.84	0.84	1342.46	1332.52

APPROACH - PYTORCH

. Transform into a Tensor object

```
tensor([[0.2300, 2.0000, 1.0000,  ..., 3.9500, 3.9800, 2.4300],
        [0.2100, 3.0000, 1.0000,  ..., 3.8900, 3.8400, 2.3100],
        [0.2300, 1.0000, 1.0000,  ..., 4.0500, 4.0700, 2.3100],
```

2. Define Neural Network

```
class DiamondsModel(nn.Module): # Inherits from nn.Module,
    def __init__(self):         # Constructor
        super(DiamondsModel, self).__init__()   # Superclass constructor
        self.model = nn.Sequential( # Sequential model, a linear stack of layers
            nn.Linear(9, 128),  # Input layer, 9 features, 128 neurons
            nn.ReLU(),  # Activation function, ReLU, Rectified Linear Unit
            nn.Linear(128, 128),    # Hidden layer, 128 neurons, 128 neurons, fully connected
            nn.ReLU(),  # Activation function, ReLU, Rectified Linear Unit
            nn.Linear(128, 1)   # Output layer, 128 neurons, 1 neuron, fully connected
        )

    def forward(self, x):   # Forward pass, x is the input data
        return self.model(x)    # Returns the model
```

3. Create Model

```
model = DiamondsModel()    # Initialize the model
critereon = nn.MSELoss()    # Define the loss function, Mean Squared Error,
optimizer = torch.optim.Adam(model.parameters(), lr=0.001) # Define the optimizer, Adam, with a learning rate of 0.001
```

PERFORMANCE - PYTORCH

Train Model

```
for epoch in range(100): # Loop through 100 epochs
    for x, y in diamonds_dataloader: # Iterate over the dataloader to get the features and target variable
        optimizer.zero_grad() # Zero the gradients, to prevent accumulation of gradients
        output = model(x) # Get the model's prediction, given the features
        loss = critereon(output, y) # Calculate the loss, comparing the model's prediction to the actual target variable
        loss.backward() # Automatic differentiation to compute the gradients
        optimizer.step() # Update the model's weights, based on the gradients, using the optimizer
    print(f'Epoch: {epoch}, Loss: {loss.item()}')
```

```
Epoch: 0, Loss: 24.369115829467773
Epoch: 1, Loss: 57.395469665527344
Epoch: 2, Loss: 0.0515870563685894
Epoch: 3, Loss: 0.07215720415115356
Epoch: 4, Loss: 0.2337525188922882
Epoch: 5, Loss: 0.3866133689880371
```

```
Epoch: 22, Loss: 0.010987097397446632
Epoch: 23, Loss: 0.010998344048857689
Epoch: 24, Loss: 0.00396603113040328
...
Epoch: 96, Loss: 0.0037782727740705013
Epoch: 97, Loss: 0.0015064210165292025
Epoch: 98, Loss: 0.007015039213001728
Epoch: 99, Loss: 0.0083926348015666
```

DIAMOND IMAGE GENERATOR

APPROACH – IMAGE GENERATOR

- Trial (& Error) with Gradio, Flask, Panel, Streamlit, Ipywidgets for text to image with DALL-E
- Succeeded with Gradio

WEBPAGE

APPROACH - WEBPAGE

- Created using HTML
- Used Co-pilot to create some of images
- Used Gradio to host the text chatbot, price predictor with iFrames
- Adjusted facecolor features

FUTURE WORK

- Expand diamond data available to chatbot
- Assess predictive modelling performance on external data
- Incorporate Diamond Image Generator
- Find a location to host webpage

THANK YOU!