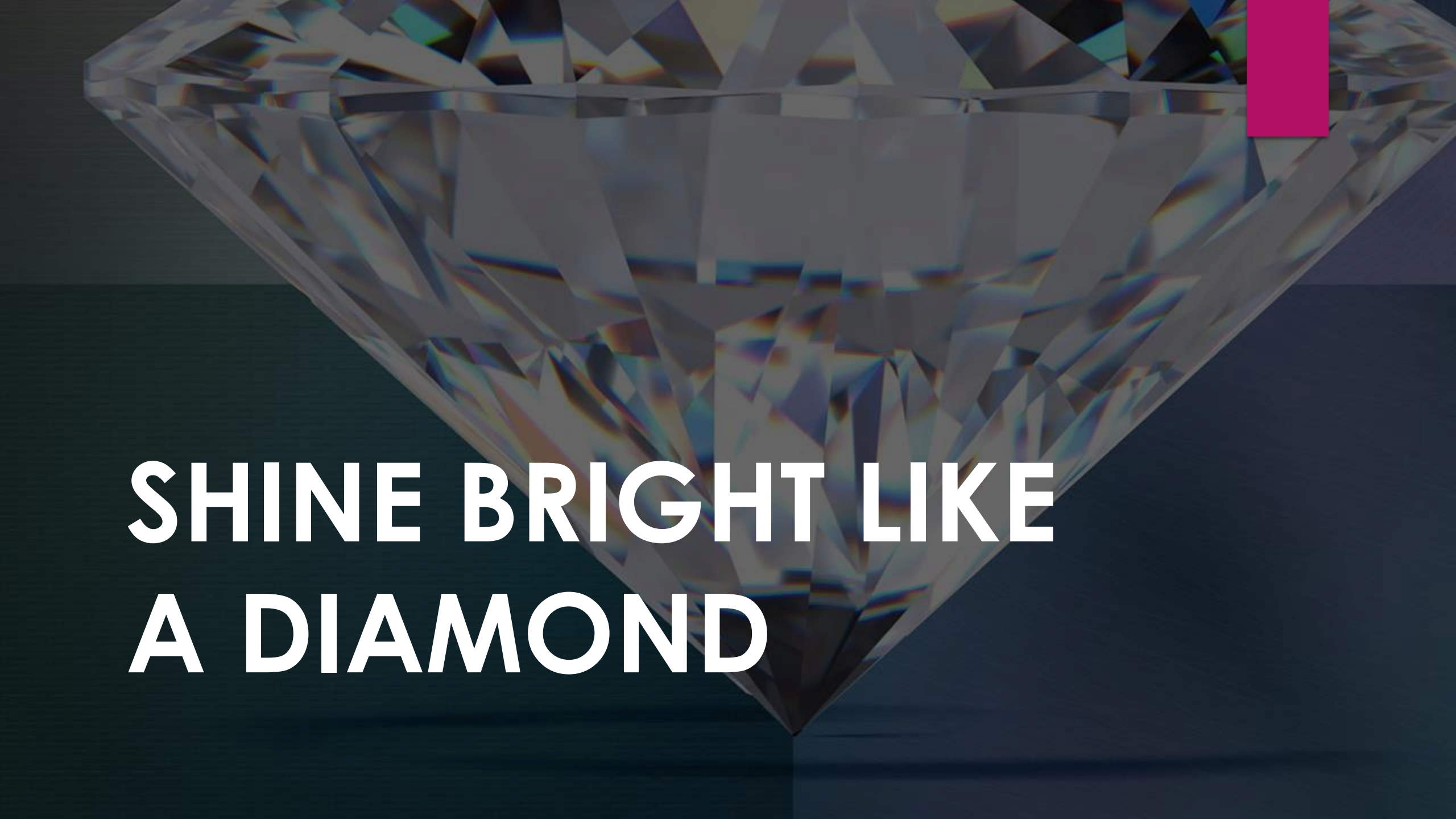

SHINE BRIGHT LIKE
A DIAMOND

TABLE OF CONTENT:

PURPOSE OF PROJECT

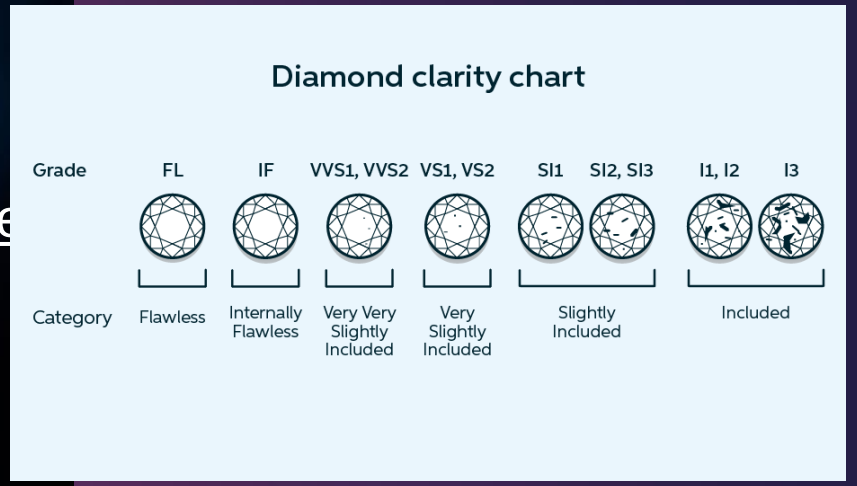
- CAN WE PREDICT A PRICE OF A DIAMOND BASED ON IT'S PROPERTIES SUCH AS THE CARAT?
- WHAT KIND OF MODEL CAN PREDICT THE PRICE OF THE DIAMOND THE BEST

DATA SOURCES

The dataset utilised for the analysis contains the prices and other attributes of over 54,000 different diamonds.

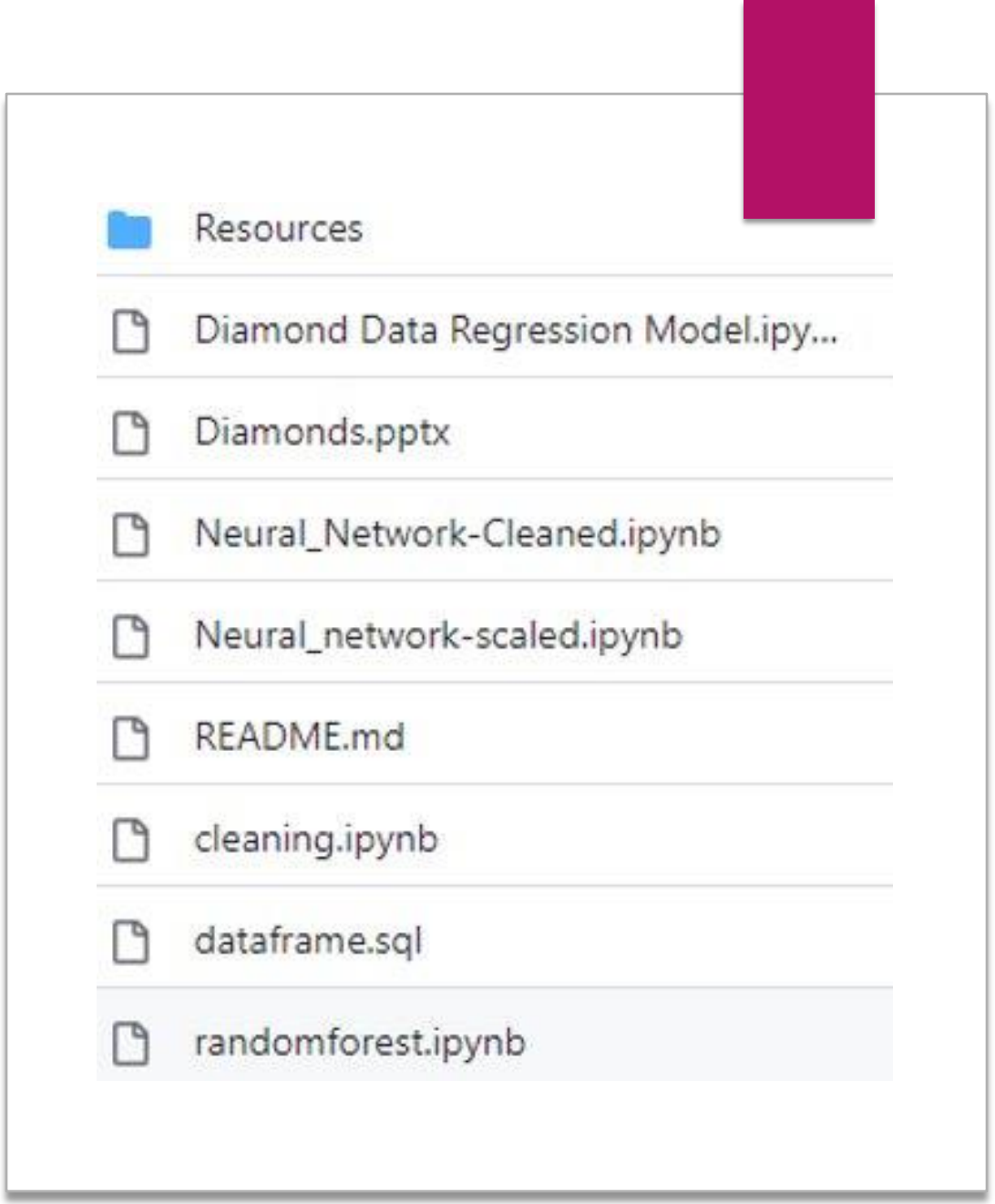
The dataset is derived via a CSV download from Kaggle: (https://www.kaggle.com/datasets/ulrikthygepedersen/diamonds?resource=download

A	B	C	D	E	F	G	H	I	J
carat	cut	color	clarity	depth	table	price	'x'	'y'	'z'
0.23	b'Ideal'	b'E'	b'SI2'	61.5	55	326	3.95	3.98	2.43
0.21	b'Premiur	b'E'	b'SI1'	59.8	61	326	3.89	3.84	2.31



STEPS UTILISED

- Data Extraction and Cleaning – Python Pandas
- Data Model Testing – Linear Regression Model
- Data Model Testing – Neural Network
- Data Model Testing – Random Forest Regression
- Database Construction – SQL (pgAdmin)



FINDINGS: Random Forest Regression

RANDOM FOREST MODEL (WITH THE SPECIFICATIONS SHOWN) PREDICTED PRICE WITH AN EXTREMELY GOOD ACCURACY. IT ALSO SHOWED THAT THE CARAT WAS BY FAR THE MOST IMPORTANT FACTOR WHEN DETERMINING PRICE

```
sorted(zip(regr.feature_importances_, X.columns), reverse=True)
```

```
[(0.8736046978322891, 'carat'),
 (0.018016124668787534, "clarity_b'SI2'"),
 (0.012913736236062l5, "color_b'J'"),
 (0.011893034417644298, 'depth'),
```

```
regr.score(X_test, y_test)
```

```
0.9648427197951752
```

```
regr = RandomForestRegressor(n_estimators=50, max_depth=15, random_state=0)
regr = regr.fit(X_train, y_train)
```

FINDINGS: Linear Regression Model

- With the Linear Regression Model, we used the price to train the model and see how well it would predict price based on various features. In this context we used price vs carat.
- As expected, the more the carats the higher the price.
- The MSE of the models perfomance is quite low which is good.

The Model

In [9]:
```
#dependencies
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,r2_score
```

creating and defining the Lin_regres model

In [10]:
```
model = linear_model.LinearRegression()
```

Building training model

In [11]:
```
model.fit(X_train, y_train)
```

Out[11]:
```
LinearRegression()
```

applying trained model on our test dataset

In [12]:
```
y_pred = model.predict(X_test)
```

Linear Regression Model

Create Y and x matrices

In [4]:
```
X = scaled_df.drop(columns=['price'])
y= scaled_df['price']
```

In [5]:
```
X.shape,y.shape
```

Out[5]:
```
((39756, 20), (39756,))
```

Splitting the data

In [6]:
```
X_train, X_test,y_train,y_test= train_test_split(X,y,test_size= 0.20)
```

Perfomance test to see how our model is perfoming so far

[13]:
```
print('Coefficients:',model.coef_)
print('Intercept:', model.intercept_)
print('Mean Squared Error(MSE): %.2f'
      % mean_squared_error(y_test,y_pred))
print('Coefficient of Determination (R^2): %.2f'
      % r2_score(y_test,y_pred))
```

```
Coefficients: [ 1.04800162 -0.01331726 -0.01686196  0.03803016  0.09483733  0.06422302
  0.07020698 -0.01587754 -0.0231132  -0.05171548 -0.09796752 -0.12086617
 -0.14490019  0.23325495  0.37037604  0.25200708  0.39665191  0.41392549
  0.30485224  0.34958165]
Intercept: 0.00041028354155236215
Mean Squared Error(MSE): 0.09
Coefficient of Determination (R^2): 0.91
```

FINDINGS: Neural Network

NEURAL NETWORK (WITH THE SPECIFICATIONS SHOWN) PERFORMED POORLY.

R2 SCORE WAS USED TO EVALUATE THE PERFORMANCE OF THE REGRESSION BASED MODEL.

THE MEAN SQUARED ERROR WAS USED TO CALCULATE THE DIFFERENCE BETWEEN A PREDICTED VALUE AND THE ACTUAL ONE.

THERE WASN'T A SPECIFICATION THAT STOOD OUT, HOWEVER THE MODEL THAT USED THE SCALED DATA AND DEPTH AS A SPECIFICATION.

```
# SCALED DATASET
y = diamond_df['depth']
X = diamond_df.drop('depth', axis=1)
```

```
r2_score(y_test, prediction)
```

```
0.33117671715778785
```

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test,y_test,verbose=2)
print(f"Loss: {model_loss}, Mean Squared Error: {model_accuracy}")
```

```
311/311 - 0s - loss: 0.6396 - mean_squared_error: 0.6396 - 435ms/epoch - 1ms/step
Loss: 0.6396331787109375, Mean Squared Error: 0.6396331787109375
```

```
y = diamond_df['depth']
X = diamond_df.drop('depth', axis=1)
```

```
r2_score(y_test, prediction)
```

```
-2.37236328827171
```

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test,y_test,verbose=2)
print(f"Loss: {model_loss}, Mean Squared Error: {model_accuracy}")
```

```
311/311 - 0s - loss: 7.5805 - mean_squared_error: 7.5805 - 462ms/epoch - 1ms/step
Loss: 7.580453872680664, Mean Squared Error: 7.580453872680664
```

WHAT ELSE COULD WE HAVE DONE

- Ideally, we could have looked into the dataset's quality. Through tableau we could have analysed the data differently.
  - Especially since one of the most common issues faced within machine learning is the quality of the data.
  - Ideally we could have found an API that connected directly to the source of the research to avoid intermediary issues.
- Another possible activity we could have done to improve our presentation could have been building an interactive website where we could have showcased different specifications of diamonds.
  - An example could have been: "what are the most common diamonds and what is their price?"

CHALLENGES AND CONCLUSIONS

- The dataset was not suitable for neural networks despite using linear regression as the outcomes are not as straightforward as binary information. It is often quite hard to predict the carat or price of a diamond since every attribute contribute to its value.
- However, Random Forest Regression suits continuous variables and is comfortably able to handle large datasets. The Random Forest Regression, in addition does not need normalisation, which is ideal for a dataset such as ours.
- Due to the complicated and complex, trying to import the CSV into SQL(pgAdmin) was a challenge.

THE TEAM

- JAMES
- DALITSO
- THARUSHA
- JADE

WILL YOU… ASK A (nice) QUESTION?