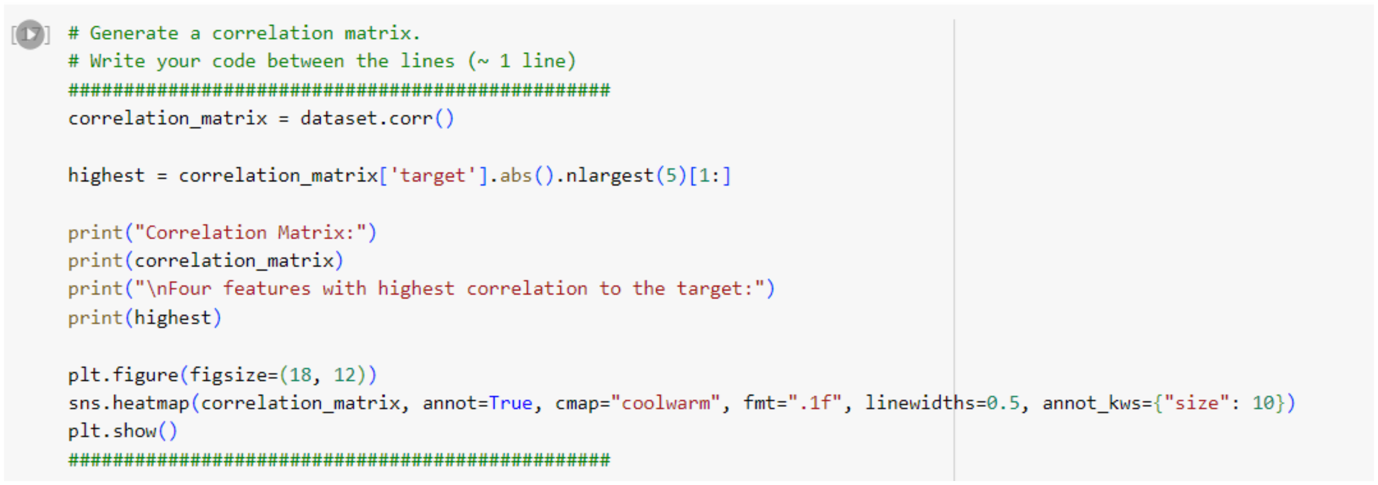
Project 2

Sareh Jalalizad

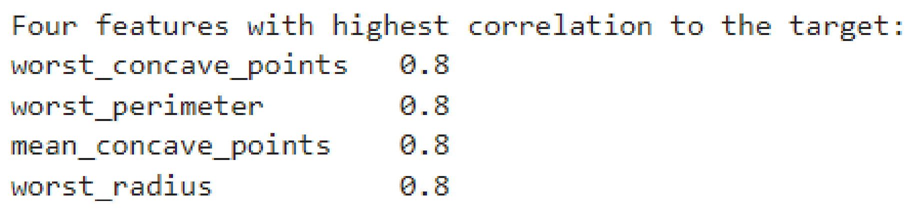
Question 1

Code

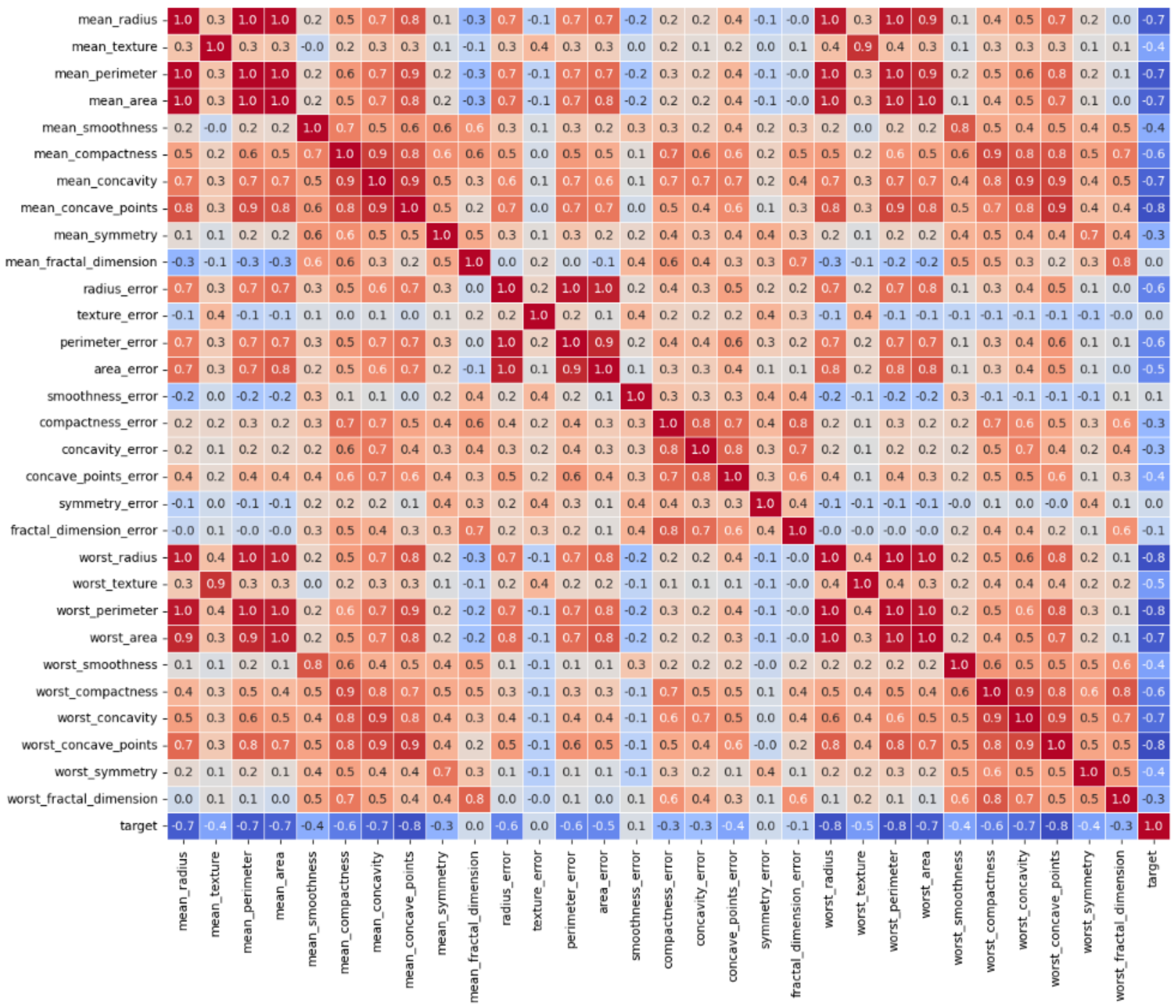


```
# Generate a correlation matrix.
# Write your code between the lines (~ 1 line)
##################################################
correlation_matrix = dataset.corr()

highest = correlation_matrix['target'].abs().nlargest(5)[1:]

print("Correlation Matrix:")
print(correlation_matrix)
print("\nFour features with highest correlation to the target:")
print(highest)

plt.figure(figsize=(18, 12))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".1f", linewidths=0.5, annot_kws={"size": 10})
plt.show()
##################################################
```

Output

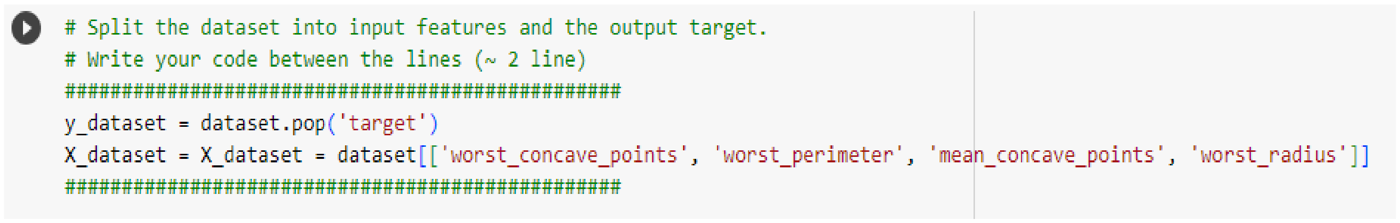


```
Four features with highest correlation to the target:
worst_concave_points    0.8
worst_perimeter         0.8
mean_concave_points     0.8
worst_radius            0.8
```


mean_radius
mean_texture
mean_perimeter
mean_area
mean_smoothness
mean_compactness
mean_concavity
mean_concave_points
mean_symmetry
mean_fractal_dimension
radius_error
texture_error
perimeter_error
area_error
smoothness_error
compactness_error
concavity_error
concave_points_error
symmetry_error
fractal_dimension_error
worst_radius
worst_texture
worst_perimeter
worst_area
worst_smoothness
worst_compactness
worst_concavity
worst_concave_points
worst_symmetry
worst_fractal_dimension
target

Question 2

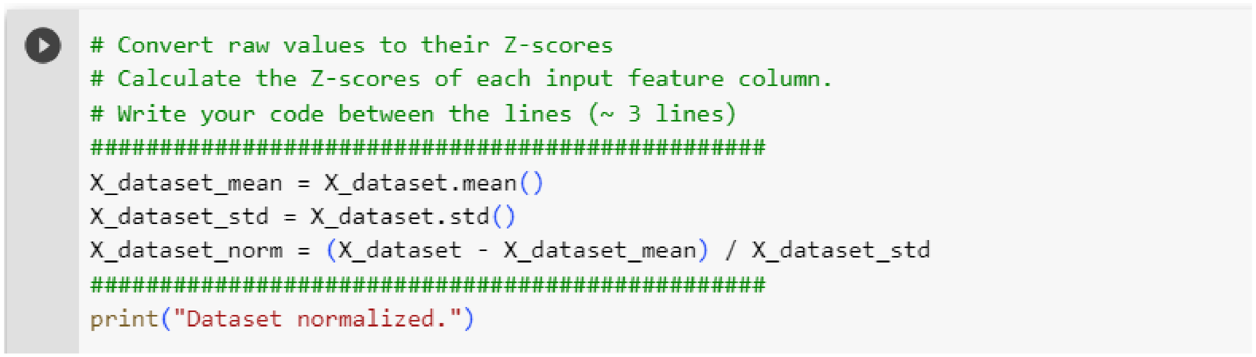
Code



```
# Split the dataset into input features and the output target.
# Write your code between the lines (~ 2 line)
##################################################
y_dataset = dataset.pop('target')
X_dataset = X_dataset = dataset[['worst_concave_points', 'worst_perimeter', 'mean_concave_points', 'worst_radius']]
##################################################
```

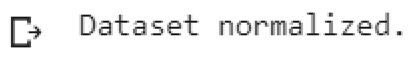
Question 3

Code



```
# Convert raw values to their Z-scores
# Calculate the Z-scores of each input feature column.
# Write your code between the lines (~ 3 lines)
##################################################
X_dataset_mean = X_dataset.mean()
X_dataset_std = X_dataset.std()
X_dataset_norm = (X_dataset - X_dataset_mean) / X_dataset_std
##################################################
print("Dataset normalized.")
```

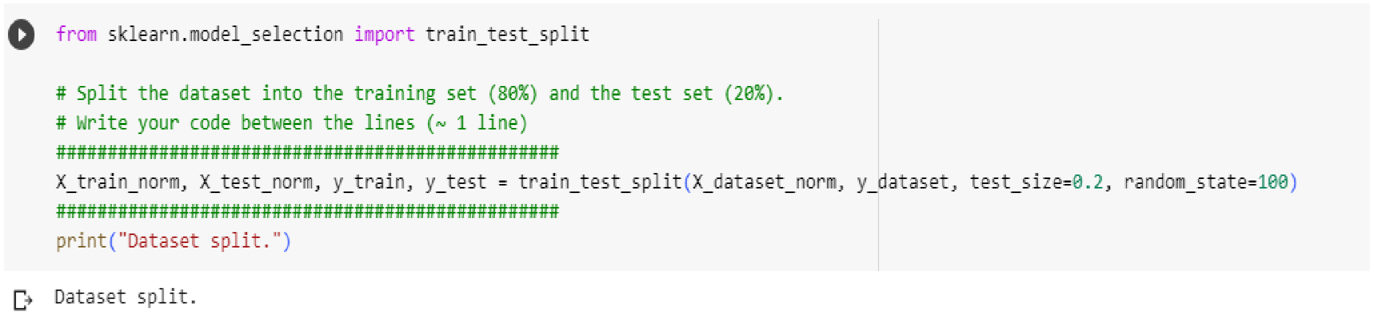
Output



```
Dataset normalized.
```

Question 4

Code

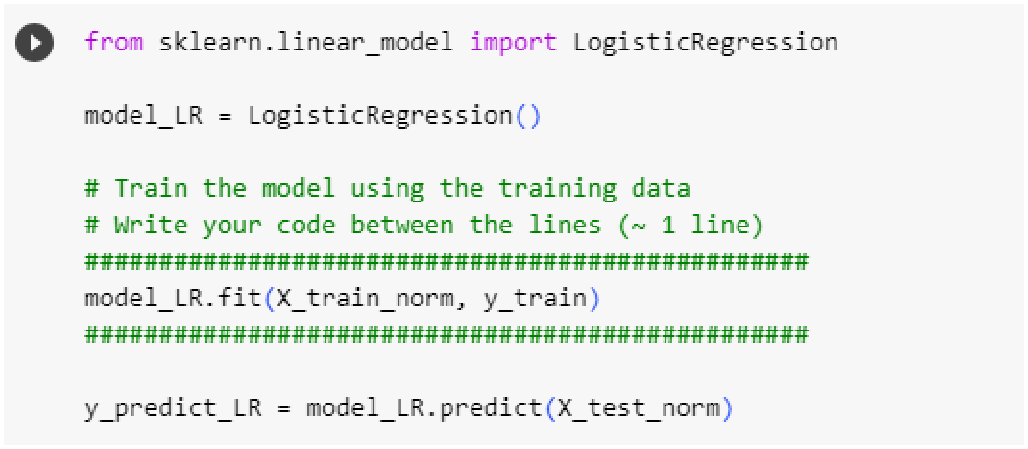


```
from sklearn.model_selection import train_test_split

# Split the dataset into the training set (80%) and the test set (20%).
# Write your code between the lines (~ 1 line)
##################################################
X_train_norm, X_test_norm, y_train, y_test = train_test_split(X_dataset_norm, y_dataset, test_size=0.2, random_state=100)
##################################################
print("Dataset split.")
```

```
Dataset split.
```

Question 5

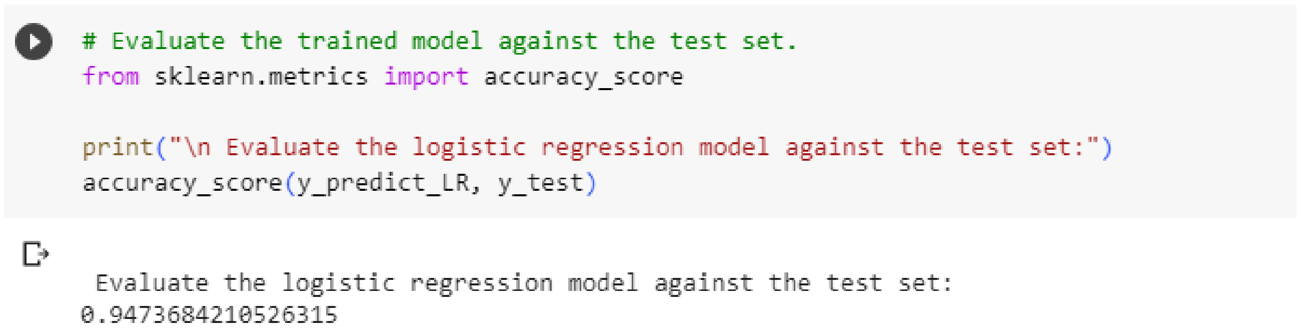
Code



```
from sklearn.linear_model import LogisticRegression

model_LR = LogisticRegression()

# Train the model using the training data
# Write your code between the lines (~ 1 line)
##################################################
model_LR.fit(X_train_norm, y_train)
##################################################

y_predict_LR = model_LR.predict(X_test_norm)
```

Question 6

Code and output

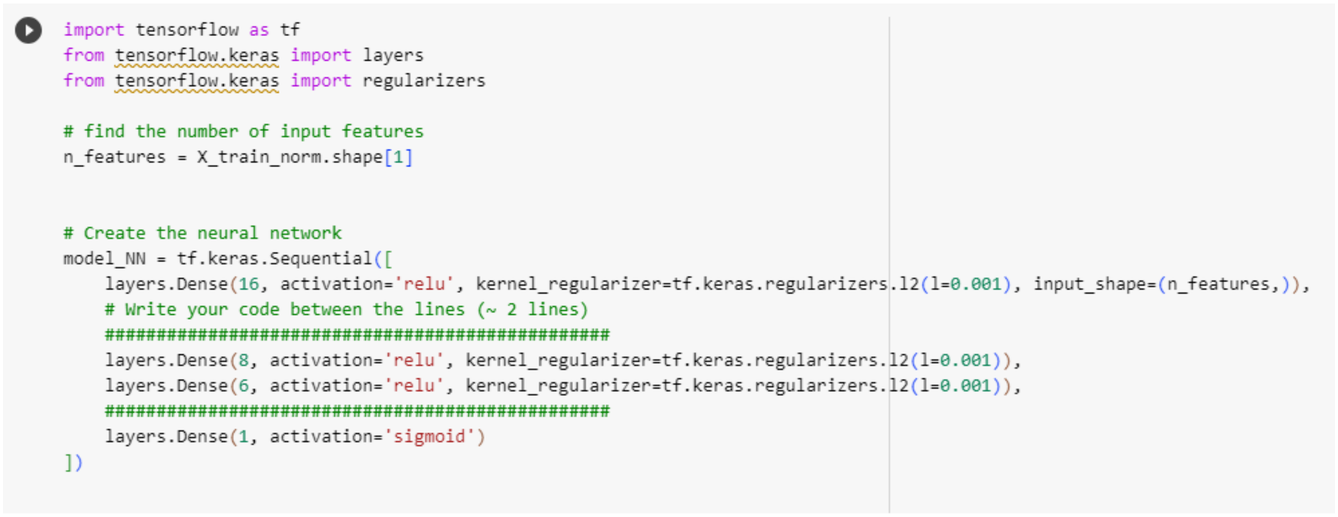


```
# Evaluate the trained model against the test set.
from sklearn.metrics import accuracy_score

print("\n Evaluate the logistic regression model against the test set:")
accuracy_score(y_predict_LR, y_test)
```

```

 Evaluate the logistic regression model against the test set:
0.9473684210526315
```

Question 7

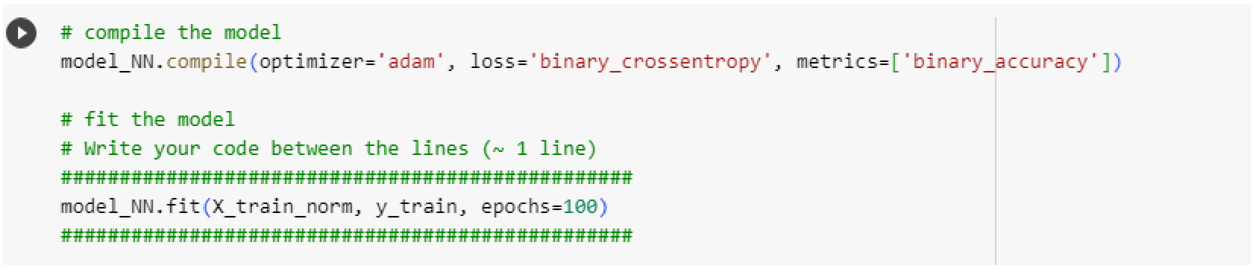
Code



```python
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import regularizers

# find the number of input features
n_features = X_train_norm.shape[1]


# Create the neural network
model_NN = tf.keras.Sequential([
    layers.Dense(16, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(l=0.001), input_shape=(n_features,)),
    # Write your code between the lines (~ 2 lines)
    ##################################################
    layers.Dense(8, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(l=0.001)),
    layers.Dense(6, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(l=0.001)),
    ##################################################
    layers.Dense(1, activation='sigmoid')
])
```

Question 8

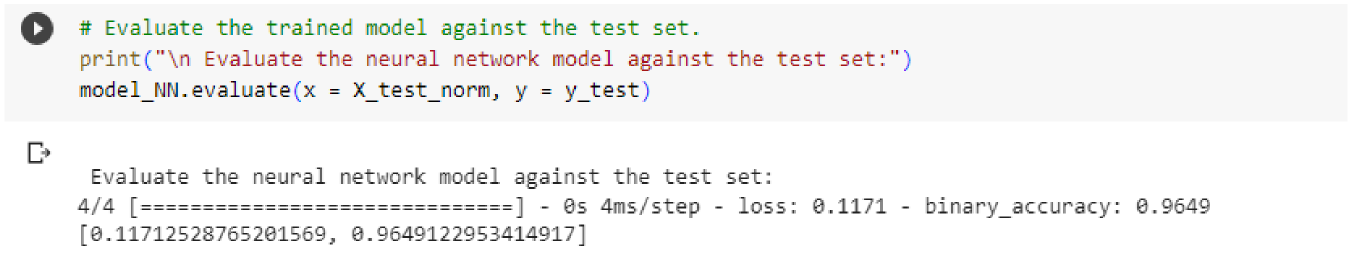
Code



```python
# compile the model
model_NN.compile(optimizer='adam', loss='binary_crossentropy', metrics=['binary_accuracy'])

# fit the model
# Write your code between the lines (~ 1 line)
##################################################
model_NN.fit(X_train_norm, y_train, epochs=100)
##################################################
```

Question 9

Code and output



```python
# Evaluate the trained model against the test set.
print("\n Evaluate the neural network model against the test set:")
model_NN.evaluate(x = X_test_norm, y = y_test)
```

```
 Evaluate the neural network model against the test set:
4/4 [==============================] - 0s 4ms/step - loss: 0.1171 - binary_accuracy: 0.9649
[0.11712528765201569, 0.9649122953414917]
```

Question 10

The neural network model performed better against the test set in predicting the output, with slightly higher accuracy. The logistic regression model achieved an accuracy of approximately 0.947 on the test set, and the neural network model achieved an accuracy of approximately 0.965.

Regarding overfitting, the training accuracy for both models is approximately the same (0.9429 for the neural network and 0.9428 for the logistic regression). In this case, both models have similar training and test accuracies, which suggests that there is no significant overfitting observed in either model. This indicates that both models are generalizing well to unseen data.

Improvement with a more complex model:
The neural network model showed a slight improvement in accuracy compared to the logistic regression model. Although the accuracy improvement is not substantial, the neural network's ability to capture more complex relationships between features may have contributed to the improvement. The additional hidden layers with non-linear activations allowed the neural network to learn more intricate patterns in the data.