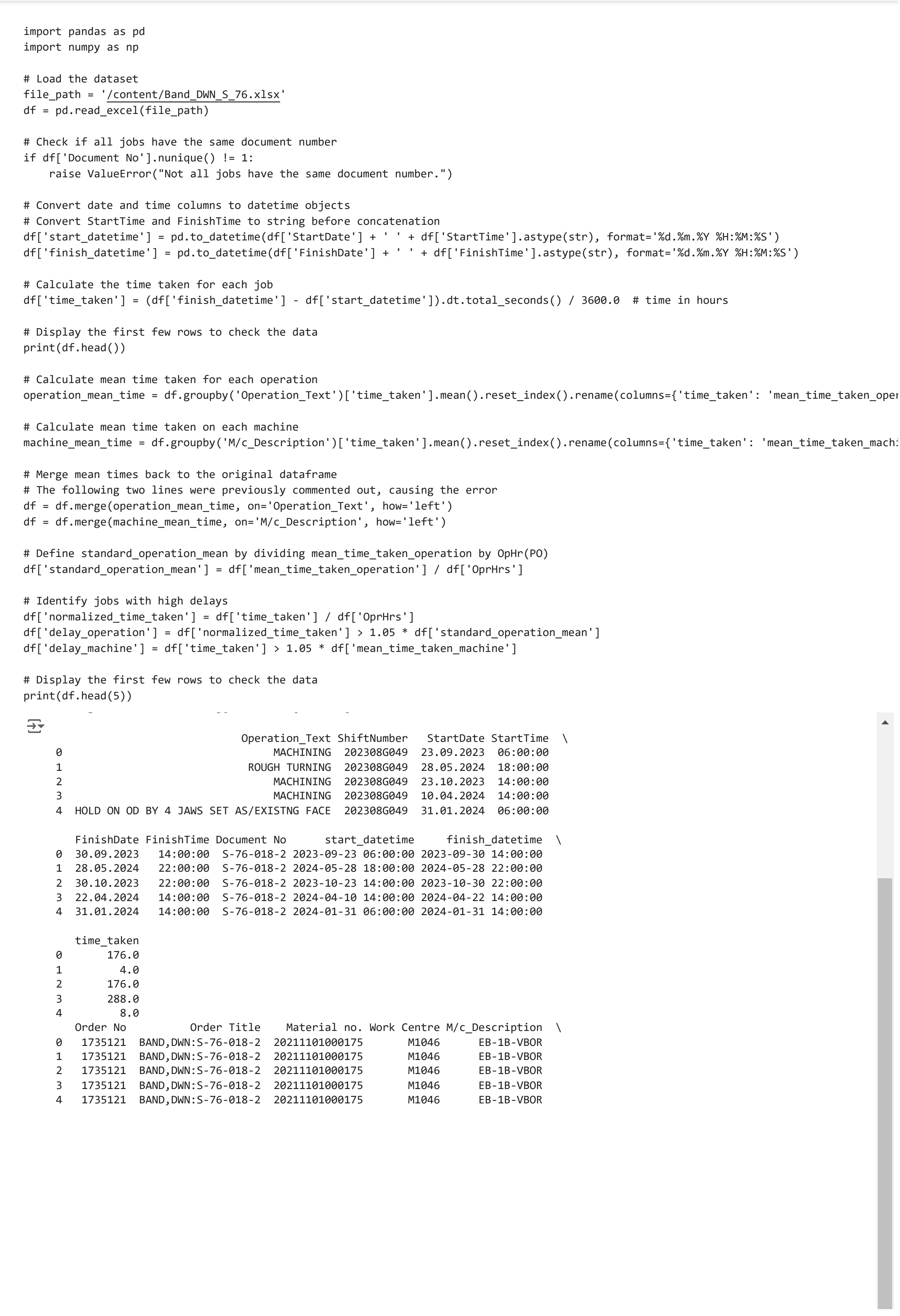

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/Band_DWN_S_76.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_oper

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_machi

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))
```

```
                           Operation_Text ShiftNumber   StartDate StartTime  \
0                               MACHINING  202308G049  23.09.2023  06:00:00
1                           ROUGH TURNING  202308G049  28.05.2024  18:00:00
2                               MACHINING  202308G049  23.10.2023  14:00:00
3                               MACHINING  202308G049  10.04.2024  14:00:00
4  HOLD ON OD BY 4 JAWS SET AS/EXISTNG FACE  202308G049  31.01.2024  06:00:00

   FinishDate FinishTime Document No      start_datetime     finish_datetime  \
0  30.09.2023   14:00:00  S-76-018-2 2023-09-23 06:00:00 2023-09-30 14:00:00
1  28.05.2024   22:00:00  S-76-018-2 2024-05-28 18:00:00 2024-05-28 22:00:00
2  30.10.2023   22:00:00  S-76-018-2 2023-10-23 14:00:00 2023-10-30 22:00:00
3  22.04.2024   14:00:00  S-76-018-2 2024-04-10 14:00:00 2024-04-22 14:00:00
4  31.01.2024   14:00:00  S-76-018-2 2024-01-31 06:00:00 2024-01-31 14:00:00

   time_taken
0       176.0
1         4.0
2       176.0
3       288.0
4         8.0
   Order No          Order Title    Material no. Work Centre M/c_Description  \
0   1735121  BAND,DWN:S-76-018-2  20211101000175       M1046      EB-1B-VBOR
1   1735121  BAND,DWN:S-76-018-2  20211101000175       M1046      EB-1B-VBOR
2   1735121  BAND,DWN:S-76-018-2  20211101000175       M1046      EB-1B-VBOR
3   1735121  BAND,DWN:S-76-018-2  20211101000175       M1046      EB-1B-VBOR
4   1735121  BAND,DWN:S-76-018-2  20211101000175       M1046      EB-1B-VBOR
```

```
4 2024-01-31 06:00:00 2024-01-31 14:00:00        8.0

  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                254.857143              274.470588                4.045351
1                250.500000              274.470588               10.020000
2                254.857143              274.470588                4.045351
3                254.857143              274.470588                4.045351
4                307.354839              274.470588               51.225806

  normalized_time_taken delay_operation  delay_machine
0              2.793651           False          False
1              0.160000           False          False
2              2.793651           False          False
3              4.571429            True          False
4              1.333333           False          False

[5 rows x 26 columns]
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

"""
# Prepare data for modeling
features = ['Operation_Text', 'M/c_Description', 'normalized_time_taken', 'standard_operation_mean']
df['Operation_Text'] = df['Operation_Text'].astype('category').cat.codes
df['M/c_Description'] = df['M/c_Description'].astype('category').cat.codes
"""
# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
4/4 - 0s - loss: 14281.2871 - val_loss: 25.9432 - 59ms/epoch - 15ms/step
Epoch 82/100
4/4 - 0s - loss: 14063.6553 - val_loss: 24.4938 - 45ms/epoch - 11ms/step
Epoch 83/100
4/4 - 0s - loss: 13870.7979 - val_loss: 24.8935 - 49ms/epoch - 12ms/step
Epoch 84/100
4/4 - 0s - loss: 13680.2402 - val_loss: 24.7638 - 56ms/epoch - 14ms/step
Epoch 85/100
4/4 - 0s - loss: 13366.6641 - val_loss: 23.6190 - 55ms/epoch - 14ms/step
Epoch 86/100
4/4 - 0s - loss: 13288.7041 - val_loss: 24.8967 - 48ms/epoch - 12ms/step
Epoch 87/100
4/4 - 0s - loss: 12935.9355 - val_loss: 23.8875 - 41ms/epoch - 10ms/step
Epoch 88/100
4/4 - 0s - loss: 12621.8008 - val_loss: 21.8152 - 39ms/epoch - 10ms/step
Epoch 89/100
4/4 - 0s - loss: 12413.8145 - val_loss: 21.5999 - 41ms/epoch - 10ms/step
Epoch 90/100
4/4 - 0s - loss: 12151.9814 - val_loss: 20.9334 - 40ms/epoch - 10ms/step
Epoch 91/100
4/4 - 0s - loss: 11928.0273 - val_loss: 19.7354 - 39ms/epoch - 10ms/step
Epoch 92/100
4/4 - 0s - loss: 11746.7900 - val_loss: 18.4458 - 44ms/epoch - 11ms/step
Epoch 93/100
4/4 - 0s - loss: 11509.3896 - val_loss: 16.6289 - 37ms/epoch - 9ms/step
Epoch 94/100
4/4 - 0s - loss: 11289.3936 - val_loss: 14.7747 - 41ms/epoch - 10ms/step
Epoch 95/100
4/4 - 0s - loss: 11088.8447 - val_loss: 13.3183 - 38ms/epoch - 9ms/step
Epoch 96/100
4/4 - 0s - loss: 10872.2480 - val_loss: 12.0861 - 37ms/epoch - 9ms/step
Epoch 97/100
4/4 - 0s - loss: 10624.0029 - val_loss: 10.9539 - 47ms/epoch - 12ms/step
Epoch 98/100
4/4 - 0s - loss: 10433.2627 - val_loss: 10.3683 - 37ms/epoch - 9ms/step
Epoch 99/100
4/4 - 0s - loss: 10200.7832 - val_loss: 9.8404 - 40ms/epoch - 10ms/step
Epoch 100/100
4/4 - 0s - loss: 10052.3457 - val_loss: 11.1265 - 38ms/epoch - 9ms/step
2/2 [==============================] - 0s 6ms/step
ANN - Mean Absolute Error: 2.7159257269684147
ANN - Mean Squared Error: 35.96655854355248
ANN - Root Mean Squared Error: 5.997212564479642
ANN - R-squared: 0.6832
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code'].t
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descriptio

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('Operation_Text')['mean_time_taken_operation'].to_dict()

# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()



    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return


    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_mea

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred = operation_machine_mean_time_dict[operation_text]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 1.05 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time of Delay: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('ROUGH TURNING', 'EB-1B-VBOR')
predict_delay('MACHINING', 'EB-1B-VBOR')
print("\n\n\n\n")
```

```
1/1 [==============================] - 0s 24ms/step
Predicted Delay: No
Estimated Time: 250.50 hours
1/1 [==============================] - 0s 23ms/step
Predicted Delay: No
Estimated Time: 254.86 hours






/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: ROUGH TURNING
Enter machine description: EB-1B-VBOR
1/1 [==============================] - 0s 24ms/step
Predicted Delay: No
Estimated Time: 250.50 hours






/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
print("\nNew job code\n")
```

```

New job code

```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/Chain_carrier_type_2_132.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

# Display the first few rows to check the data
# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_oper

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_machi

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))
```

```
   Order No                     Order Title    Material no. Work Centre  \
0   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1090   
1   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1055   
2   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1090   
3   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1055   

  M/c_Description             Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  \
0      EB-27-MILL  Universal Rail Mill      1035     10         30      40   
1      EB-5C-HBOR  Universal Rail Mill       980     20         20      20   
2      EB-27-MILL  Universal Rail Mill       578     10         30      40   
3      EB-5C-HBOR  Universal Rail Mill       275     20         20      30   

        Operation_Text ShiftNumber   StartDate StartTime  FinishDate  \
0            MACHINING  202309G034  27.09.2023  06:00:00  27.09.2023   
1  DRILL BORE & M/CING  202309G034  27.10.2023  06:00:00  27.10.2023   
2            MACHINING  202309G034  19.10.2023  06:00:00  19.10.2023   
3  DRILL BORE & M/CING  202309G034  10.10.2023  06:00:00  10.10.2023   

  FinishTime Document No      start_datetime     finish_datetime  time_taken  
0   14:00:00  URM-22-132 2023-09-27 06:00:00 2023-09-27 14:00:00         8.0  
1   22:00:00  URM-22-132 2023-10-27 06:00:00 2023-10-27 22:00:00        16.0  
2   14:00:00  URM-22-132 2023-10-19 06:00:00 2023-10-19 14:00:00         8.0  
3   14:00:00  URM-22-132 2023-10-10 06:00:00 2023-10-10 14:00:00         8.0  
   Order No                     Order Title    Material no. Work Centre  \
0   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1090   
1   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1055   
2   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1090   
3   1738479  CHAIN CARRIER TYPE-2/URM-22-132  20810101000568       M1055   

  M/c_Description             Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  \
0      EB-27-MILL  Universal Rail Mill      1035     10         30      40   
1      EB-5C-HBOR  Universal Rail Mill       980     20         20      20   
2      EB-27-MILL  Universal Rail Mill       578     10         30      40   
3      EB-5C-HBOR  Universal Rail Mill       275     20         20      30   

   ... Document No      start_datetime     finish_datetime time_taken  \
0  ...  URM-22-132 2023-09-27 06:00:00 2023-09-27 14:00:00        8.0   
1  ...  URM-22-132 2023-10-27 06:00:00 2023-10-27 22:00:00       16.0   
2  ...  URM-22-132 2023-10-19 06:00:00 2023-10-19 14:00:00        8.0   
3  ...  URM-22-132 2023-10-10 06:00:00 2023-10-10 14:00:00        8.0   

  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                       8.0                     8.0                     0.2   
1                      12.0                    12.0                     0.6   
2                       8.0                     8.0                     0.2   
3                      12.0                    12.0                     0.4   

  normalized_time_taken delay_operation  delay_machine  
0              0.200000           False          False  
1              0.800000            True           True  
2              0.200000           False          False  
3              0.266667           False          False  

[4 rows x 26 columns]
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
#r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
#print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
1/1 - 0s - loss: 2.8871e-07 - val_loss: 2.0100e-07 - 39ms/epoch - 39ms/step
Epoch 94/100
1/1 - 0s - loss: 1.2971e-07 - val_loss: 2.6726e-08 - 41ms/epoch - 41ms/step
Epoch 95/100
1/1 - 0s - loss: 1.8394e-07 - val_loss: 1.0952e-08 - 39ms/epoch - 39ms/step
Epoch 96/100
1/1 - 0s - loss: 1.9580e-07 - val_loss: 1.0864e-07 - 32ms/epoch - 32ms/step
Epoch 97/100
1/1 - 0s - loss: 1.1537e-07 - val_loss: 2.4470e-07 - 32ms/epoch - 32ms/step
Epoch 98/100
1/1 - 0s - loss: 1.2444e-07 - val_loss: 3.5127e-07 - 40ms/epoch - 40ms/step
Epoch 99/100
1/1 - 0s - loss: 2.6131e-07 - val_loss: 3.8849e-07 - 31ms/epoch - 31ms/step
Epoch 100/100
1/1 - 0s - loss: 3.3718e-07 - val_loss: 3.4903e-07 - 32ms/epoch - 32ms/step
1/1 [==============================] - 0s 60ms/step
ANN - Mean Absolute Error: 2.8669201850891115
ANN - Mean Squared Error: 8.219231347671386
ANN - Root Mean Squared Error: 2.8669201850891115
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code'].t
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descriptior

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('M/c_Description')['mean_time_taken_operation'].to_dict()

# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()

    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return

    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_mear

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred = operation_machine_mean_time_dict[machine_description]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 1.05 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time of Delay: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('MACHINING', 'EB-27-MILL')
predict_delay('DRILL BORE & M/CING', 'EB-5C-HBOR')
print("\n\n\n\n")
```

```
1/1 [==============================] - 0s 24ms/step
Predicted Delay: Yes
Estimated Time of Delay: 7.80 hours
1/1 [==============================] - 0s 23ms/step
```

```
Predicted Delay: No
Estimated Time: 12.00 hours





/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: DRILL BORE & M/CING
Enter machine description: EB-27-MILL
1/1 [==============================] - 0s 32ms/step
Predicted Delay: No
Estimated Time: 8.00 hours





/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
print("\nNew job code\n")
```

```

New job code

```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/Chain_carrier_type_1_136.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_oper

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_machi

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))
```

```
   Order No                      Order Title    Material no. Work Centre  \
0   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1055   
1   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1055   
2   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1090   
3   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1055   

  M/c_Description             Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  \
0      EB-5C-HBOR  Universal Rail Mill       982     20         20      20   
1      EB-5C-HBOR  Universal Rail Mill       981     20         20      20   
2      EB-27-MILL  Universal Rail Mill       599     10         30      30   
3      EB-5C-HBOR  Universal Rail Mill        16     10         30      30   

        Operation_Text ShiftNumber   StartDate StartTime  FinishDate  \
0  DRILL BORE & M/CING  202310G016  30.10.2023  14:00:00  31.10.2023   
1  DRILL BORE & M/CING  202310G016  28.10.2023  06:00:00  29.10.2023   
2            MACHINING  202310G016  18.10.2023  06:00:00  18.10.2023   
3            MACHINING  202310G016  24.10.2023  14:00:00  25.10.2023   

  FinishTime Document No      start_datetime     finish_datetime  time_taken  
0   22:00:00  URM-21-136 2023-10-30 14:00:00 2023-10-31 22:00:00        32.0  
1   14:00:00  URM-21-136 2023-10-28 06:00:00 2023-10-29 14:00:00        32.0  
2   22:00:00  URM-21-136 2023-10-18 06:00:00 2023-10-18 22:00:00        16.0  
3   14:00:00  URM-21-136 2023-10-24 14:00:00 2023-10-25 14:00:00        24.0  
   Order No                      Order Title    Material no. Work Centre  \
0   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1055   
1   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1055   
2   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1090   
3   1738290  CHAIN CARRIER TYPE-1/URM-21-136  20810401000055       M1055   

  M/c_Description             Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  \
0      EB-5C-HBOR  Universal Rail Mill       982     20         20      20   
1      EB-5C-HBOR  Universal Rail Mill       981     20         20      20   
2      EB-27-MILL  Universal Rail Mill       599     10         30      30   
3      EB-5C-HBOR  Universal Rail Mill        16     10         30      30   

   ... Document No      start_datetime     finish_datetime time_taken  \
0  ...  URM-21-136 2023-10-30 14:00:00 2023-10-31 22:00:00       32.0   
1  ...  URM-21-136 2023-10-28 06:00:00 2023-10-29 14:00:00       32.0   
2  ...  URM-21-136 2023-10-18 06:00:00 2023-10-18 22:00:00       16.0   
3  ...  URM-21-136 2023-10-24 14:00:00 2023-10-25 14:00:00       24.0   

  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                      32.0               29.333333                1.600000   
1                      32.0               29.333333                1.600000   
2                      20.0               16.000000                0.666667   
3                      20.0               29.333333                0.666667   

  normalized_time_taken delay_operation  delay_machine  
0              1.600000           False           True  
1              1.600000           False           True  
2              0.533333           False          False  
3              0.800000            True          False  

[4 rows x 26 columns]
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
#r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
#print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
Epoch 74/100
1/1 - 0s - loss: 1.2950e-04 - val_loss: 0.1124 - 46ms/epoch - 46ms/step
Epoch 75/100
1/1 - 0s - loss: 1.5409e-04 - val_loss: 0.1122 - 49ms/epoch - 49ms/step
Epoch 76/100
1/1 - 0s - loss: 1.7349e-04 - val_loss: 0.1121 - 129ms/epoch - 129ms/step
Epoch 77/100
1/1 - 0s - loss: 1.8665e-04 - val_loss: 0.1121 - 96ms/epoch - 96ms/step
Epoch 78/100
1/1 - 0s - loss: 1.9315e-04 - val_loss: 0.1121 - 116ms/epoch - 116ms/step
Epoch 79/100
1/1 - 0s - loss: 1.9314e-04 - val_loss: 0.1122 - 75ms/epoch - 75ms/step
Epoch 80/100
1/1 - 0s - loss: 1.8724e-04 - val_loss: 0.1124 - 151ms/epoch - 151ms/step
Epoch 81/100
1/1 - 0s - loss: 1.7641e-04 - val_loss: 0.1126 - 90ms/epoch - 90ms/step
Epoch 82/100
1/1 - 0s - loss: 1.6179e-04 - val_loss: 0.1129 - 33ms/epoch - 33ms/step
Epoch 83/100
1/1 - 0s - loss: 1.4462e-04 - val_loss: 0.1132 - 34ms/epoch - 34ms/step
Epoch 84/100
1/1 - 0s - loss: 1.2606e-04 - val_loss: 0.1135 - 38ms/epoch - 38ms/step
Epoch 85/100
1/1 - 0s - loss: 1.0720e-04 - val_loss: 0.1138 - 42ms/epoch - 42ms/step
Epoch 86/100
1/1 - 0s - loss: 8.8912e-05 - val_loss: 0.1140 - 44ms/epoch - 44ms/step
Epoch 87/100
1/1 - 0s - loss: 7.1885e-05 - val_loss: 0.1143 - 34ms/epoch - 34ms/step
Epoch 88/100
1/1 - 0s - loss: 5.6584e-05 - val_loss: 0.1145 - 35ms/epoch - 35ms/step
Epoch 89/100
1/1 - 0s - loss: 4.3285e-05 - val_loss: 0.1147 - 33ms/epoch - 33ms/step
Epoch 90/100
1/1 - 0s - loss: 3.2088e-05 - val_loss: 0.1149 - 32ms/epoch - 32ms/step
Epoch 91/100
1/1 - 0s - loss: 2.2969e-05 - val_loss: 0.1150 - 33ms/epoch - 33ms/step
Epoch 92/100
1/1 - 0s - loss: 1.5810e-05 - val_loss: 0.1150 - 35ms/epoch - 35ms/step
Epoch 93/100
1/1 - 0s - loss: 1.0428e-05 - val_loss: 0.1150 - 33ms/epoch - 33ms/step
Epoch 94/100
1/1 - 0s - loss: 6.6122e-06 - val_loss: 0.1150 - 34ms/epoch - 34ms/step
Epoch 95/100
1/1 - 0s - loss: 4.1429e-06 - val_loss: 0.1149 - 52ms/epoch - 52ms/step
Epoch 96/100
1/1 - 0s - loss: 2.7973e-06 - val_loss: 0.1148 - 37ms/epoch - 37ms/step
Epoch 97/100
1/1 - 0s - loss: 2.3656e-06 - val_loss: 0.1147 - 37ms/epoch - 37ms/step
Epoch 98/100
1/1 - 0s - loss: 2.6477e-06 - val_loss: 0.1146 - 32ms/epoch - 32ms/step
Epoch 99/100
1/1 - 0s - loss: 3.4559e-06 - val_loss: 0.1144 - 32ms/epoch - 32ms/step
Epoch 100/100
1/1 - 0s - loss: 4.6147e-06 - val_loss: 0.1143 - 36ms/epoch - 36ms/step
1/1 [==============================] - 0s 95ms/step
ANN - Mean Absolute Error: 0.003411197662353427
ANN - Mean Squared Error: 1.1636269491645484e-05
ANN - Root Mean Squared Error: 0.003411197662353427
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code'].t
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descriptio

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('Operation_Text')['mean_time_taken_operation'].to_dict()


# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()

    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return

    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_mear

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred =  operation_machine_mean_time_dict[operation_text]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 2 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('MACHINING', 'EB-27-MILL')
predict_delay('DRILL BORE & M/CING', 'EB-5C-HBOR')
print("\n\n\n\n")
```

```
1/1 [==============================] - 0s 39ms/step
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
Predicted Delay: No
Estimated Time: 20.00 hours
1/1 [==============================] - 0s 31ms/step
Predicted Delay: No
Estimated Time: 32.00 hours
```

```
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: DRILL BORE & M/CING
Enter machine description: EB-27-MILL
1/1 [==============================] - 0s 24ms/step
Predicted Delay: No
Estimated Time: 32.00 hours





/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
print("\nNew job code\n")
```

```

New job code

```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/Chain_carrier_type_2_137.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_op

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_mac

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))
```

```
   Order No                      Order Title    Material no. Work Centre  \
0   1738256  CHAIN CARRIER TYPE-2/URM-21-137  20810401000056       M1055
1   1738256  CHAIN CARRIER TYPE-2/URM-21-137  20810401000056       M1056
2   1738256  CHAIN CARRIER TYPE-2/URM-21-137  20810401000056       M1090

  M/c_Description             Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  \
0      EB-5C-HBOR  Universal Rail Mill       610     20         20      30
1      EB-5D-HBOR  Universal Rail Mill        12     20         20      20
2      EB-27-MILL  Universal Rail Mill        11     10         30      60

        Operation_Text ShiftNumber   StartDate StartTime  FinishDate  \
0  DRILL BORE & M/CING  202310G017  18.10.2023  06:00:00  18.10.2023
1  DRILL BORE & M/CING  202310G017  18.10.2023  06:00:00  20.10.2023
2            MACHINING  202310G017  03.09.2023  06:00:00  12.09.2023

  FinishTime Document No      start_datetime     finish_datetime  time_taken
0   22:00:00  URM-21-137 2023-10-18 06:00:00 2023-10-18 22:00:00        16.0
1   14:00:00  URM-21-137 2023-10-18 06:00:00 2023-10-20 14:00:00        56.0
2   18:00:00  URM-21-137 2023-09-03 06:00:00 2023-09-12 18:00:00       228.0
   Order No                      Order Title    Material no. Work Centre  \
0   1738256  CHAIN CARRIER TYPE-2/URM-21-137  20810401000056       M1055
1   1738256  CHAIN CARRIER TYPE-2/URM-21-137  20810401000056       M1056
2   1738256  CHAIN CARRIER TYPE-2/URM-21-137  20810401000056       M1090

  M/c_Description             Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  \
0      EB-5C-HBOR  Universal Rail Mill       610     20         20      30
1      EB-5D-HBOR  Universal Rail Mill        12     20         20      20
2      EB-27-MILL  Universal Rail Mill        11     10         30      60

   ... Document No      start_datetime     finish_datetime time_taken  \
0  ...  URM-21-137 2023-10-18 06:00:00 2023-10-18 22:00:00       16.0
1  ...  URM-21-137 2023-10-18 06:00:00 2023-10-20 14:00:00       56.0
2  ...  URM-21-137 2023-09-03 06:00:00 2023-09-12 18:00:00      228.0
```

```
  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                      36.0                    16.0                     1.2   
1                      36.0                    56.0                     1.8   
2                     228.0                   228.0                     3.8   

  normalized_time_taken delay_operation  delay_machine  
0              0.533333           False          False  
1              2.800000            True          False  
2              3.800000           False          False  

[3 rows x 26 columns]
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
#r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
#print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
Epoch 87/100
1/1 - 0s - loss: 7.2096e-06 - val_loss: 12.9011 - 33ms/epoch - 33ms/step
Epoch 88/100
1/1 - 0s - loss: 4.4089e-05 - val_loss: 12.8981 - 38ms/epoch - 38ms/step
Epoch 89/100
1/1 - 0s - loss: 2.2671e-04 - val_loss: 12.8953 - 39ms/epoch - 39ms/step
Epoch 90/100
1/1 - 0s - loss: 5.1041e-04 - val_loss: 12.8929 - 40ms/epoch - 40ms/step
Epoch 91/100
1/1 - 0s - loss: 8.5718e-04 - val_loss: 12.8908 - 42ms/epoch - 42ms/step
Epoch 92/100
1/1 - 0s - loss: 0.0012 - val_loss: 12.8889 - 39ms/epoch - 39ms/step
Epoch 93/100
1/1 - 0s - loss: 0.0016 - val_loss: 12.8873 - 40ms/epoch - 40ms/step
Epoch 94/100
1/1 - 0s - loss: 0.0020 - val_loss: 12.8860 - 40ms/epoch - 40ms/step
Epoch 95/100
1/1 - 0s - loss: 0.0023 - val_loss: 12.8849 - 40ms/epoch - 40ms/step
Epoch 96/100
1/1 - 0s - loss: 0.0026 - val_loss: 12.8841 - 48ms/epoch - 48ms/step
Epoch 97/100
1/1 - 0s - loss: 0.0029 - val_loss: 12.8834 - 36ms/epoch - 36ms/step
Epoch 98/100
1/1 - 0s - loss: 0.0031 - val_loss: 12.8830 - 32ms/epoch - 32ms/step
Epoch 99/100
1/1 - 0s - loss: 0.0032 - val_loss: 12.8827 - 34ms/epoch - 34ms/step
Epoch 100/100
1/1 - 0s - loss: 0.0033 - val_loss: 12.8826 - 32ms/epoch - 32ms/step
1/1 [==============================] - 0s 75ms/step
ANN - Mean Absolute Error: 5.387379137674968
ANN - Mean Squared Error: 29.023853973055477
ANN - Root Mean Squared Error: 5.387379137674968
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code'].t
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descriptior

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('Operation_Text')['mean_time_taken_operation'].to_dict()


# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()

    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return

    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_mear

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred =  operation_machine_mean_time_dict[operation_text]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 2 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('MACHINING', 'EB-27-MILL')
predict_delay('DRILL BORE & M/CING', 'EB-5C-HBOR')
print("\n\n\n\n")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
1/1 [==============================] - 0s 51ms/step
Predicted Delay: No
Estimated Time: 228.00 hours
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
1/1 [==============================] - 0s 77ms/step
Predicted Delay: Yes
Estimated Time: 34.20 hours
```

```
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: MACHINING
Enter machine description: EB-5C-HBOR
1/1 [==============================] - 0s 25ms/step
Predicted Delay: No
Estimated Time: 228.00 hours




/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
print("\nNew job code\n")
```

```

New job code
```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/WHEEL_DIA.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_op

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_mac

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))
```

```
4   1739868  WHEEL DIA 710'B', PMM-3880  60310101003133       M1048

  M/c_Description Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs Operation_Text  \
0     LB-26A-SLOT       PM       637     40          6       6       SLOTTING   
1     LB-26A-SLOT       PM       432     40          6       6       SLOTTING   
2   EB-60B-RDRILL       PM       337     20          3       3       DRILLING   
3      LB-4C-VBOR       PM       336     30          6       6         BORING   
4      LB-4C-VBOR       PM       323     10         24      24        TURNING   

   ShiftNumber   StartDate StartTime  FinishDate FinishTime Document No  \
0  202310WN004  18.03.2024  14:00:00  18.03.2024   22:00:00  PMM-3880R1   
1  202310WN004  10.04.2024  14:00:00  10.04.2024   22:00:00  PMM-3880R1   
2  202310WN004  08.04.2024  18:00:00  08.04.2024   22:00:00  PMM-3880R1   
3  202310WN004  06.04.2024  14:00:00  06.04.2024   22:00:00  PMM-3880R1   
4  202310WN004  09.11.2023  06:00:00  10.11.2023   14:00:00  PMM-3880R1   

       start_datetime     finish_datetime  time_taken  
0 2024-03-18 14:00:00 2024-03-18 22:00:00         8.0  
1 2024-04-10 14:00:00 2024-04-10 22:00:00         8.0  
2 2024-04-08 18:00:00 2024-04-08 22:00:00         4.0  
3 2024-04-06 14:00:00 2024-04-06 22:00:00         8.0  
4 2023-11-09 06:00:00 2023-11-10 14:00:00        32.0  
   Order No                 Order Title    Material no. Work Centre  \
0   1739868  WHEEL DIA 710'B', PMM-3880  60310101003133       M1063   
1   1739868  WHEEL DIA 710'B', PMM-3880  60310101003133       M1063   
2   1739868  WHEEL DIA 710'B', PMM-3880  60310101003133       M1115   
3   1739868  WHEEL DIA 710'B', PMM-3880  60310101003133       M1048   
4   1739868  WHEEL DIA 710'B', PMM-3880  60310101003133       M1048   

  M/c_Description Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  ...  \
0     LB-26A-SLOT       PM       637     40          6       6  ...   
1     LB-26A-SLOT       PM       432     40          6       6  ...   
2   EB-60B-RDRILL       PM       337     20          3       3  ...   
3      LB-4C-VBOR       PM       336     30          6       6  ...   
4      LB-4C-VBOR       PM       323     10         24      24  ...   

  Document No      start_datetime     finish_datetime time_taken  \
   mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                   8.000000                     8.0                1.333333
1                   8.000000                     8.0                1.333333
2                   4.000000                     4.0                1.333333
3                   8.000000                    19.2                1.333333
4                  26.666667                    19.2                1.111111

   normalized_time_taken delay_operation  delay_machine
0               1.333333           False          False
1               1.333333           False          False
2               1.333333           False          False
3               1.333333           False          False
4               1.333333            True           True

[5 rows x 26 columns]
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
#print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
Epoch 85/100
1/1 - 0s - loss: 0.0011 - val_loss: 0.0017 - 34ms/epoch - 34ms/step
Epoch 86/100
1/1 - 0s - loss: 0.0010 - val_loss: 0.0015 - 32ms/epoch - 32ms/step
Epoch 87/100
1/1 - 0s - loss: 9.0323e-04 - val_loss: 0.0012 - 34ms/epoch - 34ms/step
Epoch 88/100
1/1 - 0s - loss: 8.1043e-04 - val_loss: 0.0011 - 37ms/epoch - 37ms/step
Epoch 89/100
1/1 - 0s - loss: 7.2533e-04 - val_loss: 9.0095e-04 - 32ms/epoch - 32ms/step
Epoch 90/100
1/1 - 0s - loss: 6.4678e-04 - val_loss: 7.6548e-04 - 33ms/epoch - 33ms/step
Epoch 91/100
1/1 - 0s - loss: 5.7390e-04 - val_loss: 6.5110e-04 - 36ms/epoch - 36ms/step
Epoch 92/100
1/1 - 0s - loss: 5.0617e-04 - val_loss: 5.5484e-04 - 34ms/epoch - 34ms/step
Epoch 93/100
1/1 - 0s - loss: 4.4332e-04 - val_loss: 4.7364e-04 - 33ms/epoch - 33ms/step
Epoch 94/100
1/1 - 0s - loss: 3.8513e-04 - val_loss: 4.0516e-04 - 35ms/epoch - 35ms/step
Epoch 95/100
1/1 - 0s - loss: 3.3164e-04 - val_loss: 3.4739e-04 - 41ms/epoch - 41ms/step
Epoch 96/100
1/1 - 0s - loss: 2.8297e-04 - val_loss: 2.9859e-04 - 43ms/epoch - 43ms/step
Epoch 97/100
1/1 - 0s - loss: 2.3922e-04 - val_loss: 2.5734e-04 - 41ms/epoch - 41ms/step
Epoch 98/100
1/1 - 0s - loss: 2.0047e-04 - val_loss: 2.2238e-04 - 40ms/epoch - 40ms/step
Epoch 99/100
1/1 - 0s - loss: 1.6666e-04 - val_loss: 1.9266e-04 - 43ms/epoch - 43ms/step
Epoch 100/100
1/1 - 0s - loss: 1.3764e-04 - val_loss: 1.6731e-04 - 33ms/epoch - 33ms/step
1/1 [==============================] - 0s 67ms/step
ANN - Mean Absolute Error: 0.010730266571044922
ANN - Mean Squared Error: 0.00017165735252117572
ANN - Root Mean Squared Error: 0.013101807223477825
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code'].t
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descriptio

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('Operation_Text')['mean_time_taken_operation'].to_dict()


# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()

    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return

    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_mea

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred =  operation_machine_mean_time_dict[operation_text]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 2 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('TURNING', 'LB-4C-VBOR')
predict_delay('BORING', 'LB-4C-VBOR')
print("\n\n\n\n")
```

```
1/1 [==============================] - 0s 56ms/step
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
Predicted Delay: Yes
Estimated Time: 25.56 hours
1/1 [==============================] - 0s 87ms/step
Predicted Delay: Yes
Estimated Time: 6.67 hours
```

```
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: DRILLING
Enter machine description: EB-60B-RDRILL
1/1 [==============================] - 0s 25ms/step
Predicted Delay: No
Estimated Time: 4.00 hours




/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
print("\nNew job code\n")
```

```

New job code

```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/FLAT_HAMMER.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_oper

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_machi

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))
```

```
   Order No                        Order Title    Material no. Work Centre  \
0   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   
1   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   
2   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1111   
3   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   
4   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   

  M/c_Description Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs Operation_Text  \
0   LB-39A-VDRILL    SP-II      1107     10        0.5    32.5   DRILL DIA 52   
1   LB-39A-VDRILL    SP-II      1055     10        0.5    25.0   DRILL DIA 52   
2    LB-39-VDRILL    SP-II      1054     10        0.5    10.0   DRILL DIA 52   
3   LB-39A-VDRILL    SP-II      1032     10        0.5    33.0   DRILL DIA 52   
4   LB-39A-VDRILL    SP-II      1029     10        0.5    19.5   DRILL DIA 52   

  ShiftNumber   StartDate StartTime  FinishDate FinishTime Document No  \
0  202310G041  31.10.2023  06:00:00  31.10.2023   14:00:00  SP2MM-1269   
1  202310G041  21.10.2023  06:00:00  21.10.2023   14:00:00  SP2MM-1269   
2  202310G041  21.10.2023  14:00:00  21.10.2023   22:00:00  SP2MM-1269   
3  202310G041  30.10.2023  06:00:00  30.10.2023   14:00:00  SP2MM-1269   
4  202310G041  28.10.2023  18:00:00  28.10.2023   22:00:00  SP2MM-1269   

       start_datetime     finish_datetime  time_taken  
0 2023-10-31 06:00:00 2023-10-31 14:00:00         8.0  
1 2023-10-21 06:00:00 2023-10-21 14:00:00         8.0  
2 2023-10-21 14:00:00 2023-10-21 22:00:00         8.0  
3 2023-10-30 06:00:00 2023-10-30 14:00:00         8.0  
4 2023-10-28 18:00:00 2023-10-28 22:00:00         4.0  
   Order No                        Order Title    Material no. Work Centre  \
0   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   
1   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   
2   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1111   
3   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   
4   1739798  FLAT HAMMER,50MM,DWN:SP2-MM-1269  20211101000186       M1239   

  M/c_Description Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  ...  \
0   LB-39A-VDRILL    SP-II      1107     10        0.5    32.5  ...   
1   LB-39A-VDRILL    SP-II      1055     10        0.5    25.0  ...   
2    LB-39-VDRILL    SP-II      1054     10        0.5    10.0  ...   
3   LB-39A-VDRILL    SP-II      1032     10        0.5    33.0  ...   
4   LB-39A-VDRILL    SP-II      1029     10        0.5    19.5  ...   

  Document No      start_datetime     finish_datetime time_taken  \
0  SP2MM-1269 2023-10-31 06:00:00 2023-10-31 14:00:00        8.0   
1  SP2MM-1269 2023-10-21 06:00:00 2023-10-21 14:00:00        8.0   
2  SP2MM-1269 2023-10-21 14:00:00 2023-10-21 22:00:00        8.0   
3  SP2MM-1269 2023-10-30 06:00:00 2023-10-30 14:00:00        8.0   
4  SP2MM-1269 2023-10-28 18:00:00 2023-10-28 22:00:00        4.0   

  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                  6.604651                     7.3                0.203220   
1                  6.604651                     7.3                0.264186   
2                  6.604651                     6.0                0.660465   
3                  6.604651                     7.3                0.200141   
4                  6.604651                     7.3                0.338700   

  normalized_time_taken delay_operation  delay_machine  
0              0.246154            True           True  
1              0.320000            True           True  
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
Epoch 1/100
1/1 - 1s - loss: 0.5498 - val_loss: 0.1552 - 968ms/epoch - 968ms/step
Epoch 2/100
1/1 - 0s - loss: 0.4880 - val_loss: 0.1309 - 49ms/epoch - 49ms/step
Epoch 3/100
1/1 - 0s - loss: 0.4305 - val_loss: 0.1092 - 33ms/epoch - 33ms/step
Epoch 4/100
1/1 - 0s - loss: 0.3770 - val_loss: 0.0899 - 32ms/epoch - 32ms/step
Epoch 5/100
1/1 - 0s - loss: 0.3281 - val_loss: 0.0728 - 38ms/epoch - 38ms/step
Epoch 6/100
1/1 - 0s - loss: 0.2840 - val_loss: 0.0582 - 42ms/epoch - 42ms/step
Epoch 7/100
1/1 - 0s - loss: 0.2474 - val_loss: 0.0458 - 44ms/epoch - 44ms/step
Epoch 8/100
1/1 - 0s - loss: 0.2145 - val_loss: 0.0355 - 34ms/epoch - 34ms/step
Epoch 9/100
1/1 - 0s - loss: 0.1856 - val_loss: 0.0269 - 33ms/epoch - 33ms/step
Epoch 10/100
1/1 - 0s - loss: 0.1601 - val_loss: 0.0199 - 58ms/epoch - 58ms/step
Epoch 11/100
1/1 - 0s - loss: 0.1380 - val_loss: 0.0144 - 102ms/epoch - 102ms/step
Epoch 12/100
1/1 - 0s - loss: 0.1188 - val_loss: 0.0101 - 99ms/epoch - 99ms/step
Epoch 13/100
1/1 - 0s - loss: 0.1024 - val_loss: 0.0071 - 53ms/epoch - 53ms/step
Epoch 14/100
1/1 - 0s - loss: 0.0887 - val_loss: 0.0053 - 151ms/epoch - 151ms/step
Epoch 15/100
1/1 - 0s - loss: 0.0771 - val_loss: 0.0043 - 50ms/epoch - 50ms/step
Epoch 16/100
1/1 - 0s - loss: 0.0676 - val_loss: 0.0040 - 91ms/epoch - 91ms/step
Epoch 17/100
1/1 - 0s - loss: 0.0598 - val_loss: 0.0042 - 142ms/epoch - 142ms/step
Epoch 18/100
1/1 - 0s - loss: 0.0536 - val_loss: 0.0050 - 192ms/epoch - 192ms/step
Epoch 19/100
1/1 - 0s - loss: 0.0486 - val_loss: 0.0059 - 183ms/epoch - 183ms/step
Epoch 20/100
1/1 - 0s - loss: 0.0446 - val_loss: 0.0070 - 100ms/epoch - 100ms/step
Epoch 21/100
1/1 - 0s - loss: 0.0412 - val_loss: 0.0079 - 188ms/epoch - 188ms/step
Epoch 22/100
1/1 - 0s - loss: 0.0382 - val_loss: 0.0086 - 71ms/epoch - 71ms/step
Epoch 23/100
1/1 - 0s - loss: 0.0354 - val_loss: 0.0090 - 154ms/epoch - 154ms/step
Epoch 24/100
1/1 - 0s - loss: 0.0328 - val_loss: 0.0092 - 67ms/epoch - 67ms/step
Epoch 25/100
1/1 - 0s - loss: 0.0303 - val_loss: 0.0091 - 59ms/epoch - 59ms/step
Epoch 26/100
1/1 - 0s - loss: 0.0278 - val_loss: 0.0088 - 66ms/epoch - 66ms/step
Epoch 27/100
1/1 - 0s - loss: 0.0253 - val_loss: 0.0082 - 89ms/epoch - 89ms/step
```

```
Epoch 28/100
1/1 - 0s - loss: 0.0227 - val_loss: 0.0075 - 63ms/epoch - 63ms/step
Epoch 29/100
1/1 - 0s - loss: 0.0203 - val_loss: 0.0067 - 41ms/epoch - 41ms/step
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code']
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descripti

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('Operation_Text')['mean_time_taken_operation'].to_dict()


# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()

    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return

    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_me

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred =  operation_machine_mean_time_dict[operation_text]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 2 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('DRILL DIA 52', 'LB-39A-VDRILL')
predict_delay('DRILL DIA 52', 'LB-39-VDRILL')
print("\n\n\n\n")
```

```
1/1 [==============================] - 0s 32ms/step
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
Predicted Delay: No
Estimated Time: 6.60 hours
1/1 [==============================] - 0s 34ms/step
Predicted Delay: Yes
Estimated Time: 6.47 hours
```

```
op = input("Enter operation text: ")
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: DRILL DIA 52
Enter machine description: LB-39A-VDRILL
1/1 [==============================] - 0s 24ms/step
Predicted Delay: No
Estimated Time: 6.60 hours





/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
print("\nNew job code\n")
```

```

New job code
```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/Leading_nut_cover.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

print(df.shape)

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_op

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_mac

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))

# Select only numerical columns before checking for infinite values
numerical_df = df.select_dtypes(include=np.number)

# Apply any() along the correct axis (axis=1 for rows) to get a boolean index for each row
inf_rows = numerical_df.index[np.isinf(numerical_df).any(axis=1)]

# Print the row indices with infinity values
print("Row indices with infinity values:")
print(inf_rows)

# Remove rows with infinity values
df = df.drop(inf_rows)

print("Shape of DataFrame after removing infinity rows:", df.shape)
```

```
0   HB-11A-HLATHE       BF      1131     40          9       9      THREADING
1   HB-11A-HLATHE       BF      1123     40          9       9      THREADING
2    HB-9A-HLATHE       BF      1122     30          6       6  ROUGH TURNING
3    EB-28B-WMILL       BF      1097     10          7       7      MACHINING
4   HB-11A-HLATHE       BF      1044     40          9       9      THREADING

  ShiftNumber   StartDate StartTime  FinishDate FinishTime        Document No  \
0  202309G049  29.09.2023  14:00:00  29.09.2023   18:00:00  B-603196-B-603179
1  202309G049  31.10.2023  10:00:00  31.10.2023   14:00:00  B-603196-B-603179
2  202309G049  31.10.2023  10:00:00  31.10.2023   14:00:00  B-603196-B-603179
3  202309G049  28.10.2023  10:00:00  28.10.2023   14:00:00  B-603196-B-603179
4  202309G049  30.10.2023  10:00:00  30.10.2023   14:00:00  B-603196-B-603179

       start_datetime     finish_datetime  time_taken
```

```
3   1734906  LEADINGNUTCOVER(LH&RH)/BF  20310701000027       M1084
4   1734906  LEADINGNUTCOVER(LH&RH)/BF  20310701000027       M1016

  M/c_Description Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  ...  \
0   HB-11A-HLATHE       BF      1131     40          9       9  ...
1   HB-11A-HLATHE       BF      1123     40          9       9  ...
2    HB-9A-HLATHE       BF      1122     30          6       6  ...
3    EB-28B-WMILL       BF      1097     10          7       7  ...
4   HB-11A-HLATHE       BF      1044     40          9       9  ...

          Document No      start_datetime     finish_datetime time_taken  \
0  B-603196-B-603179 2023-09-29 14:00:00 2023-09-29 18:00:00        4.0
1  B-603196-B-603179 2023-10-31 10:00:00 2023-10-31 14:00:00        4.0
2  B-603196-B-603179 2023-10-31 10:00:00 2023-10-31 14:00:00        4.0
3  B-603196-B-603179 2023-10-28 10:00:00 2023-10-28 14:00:00        4.0
4  B-603196-B-603179 2023-10-30 10:00:00 2023-10-30 14:00:00        4.0

  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                610.517241              581.114754               67.835249
1                610.517241              581.114754               67.835249
2                633.607143              668.754717              105.601190
3                 17.133333               16.918033                2.447619
4                610.517241              581.114754               67.835249

  normalized_time_taken delay_operation  delay_machine
0              0.444444           False          False
1              0.444444           False          False
2              0.666667           False          False
3              0.571429           False          False
4              0.444444           False          False

[5 rows x 26 columns]
Row indices with infinity values:
Index([105, 106], dtype='int64')
Shape of DataFrame after removing infinity rows: (224, 26)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'time_taken']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
Epoch 1/100
5/5 - 1s - loss: 106114.0469 - val_loss: 5.8981 - 955ms/epoch - 191ms/step
Epoch 2/100
```

```
5/5 - 0s - loss: 106075.0469 - val_loss: 5.5925 - 58ms/epoch - 12ms/step
Epoch 3/100
5/5 - 0s - loss: 106026.6172 - val_loss: 5.3518 - 42ms/epoch - 8ms/step
Epoch 4/100
5/5 - 0s - loss: 105973.5000 - val_loss: 5.1683 - 39ms/epoch - 8ms/step
Epoch 5/100
5/5 - 0s - loss: 105946.5078 - val_loss: 5.0141 - 45ms/epoch - 9ms/step
Epoch 6/100
5/5 - 0s - loss: 105906.3672 - val_loss: 4.8956 - 70ms/epoch - 14ms/step
Epoch 7/100
5/5 - 0s - loss: 105849.4531 - val_loss: 4.8049 - 42ms/epoch - 8ms/step
Epoch 8/100
5/5 - 0s - loss: 105830.3203 - val_loss: 4.7340 - 58ms/epoch - 12ms/step
Epoch 9/100
5/5 - 0s - loss: 105754.4297 - val_loss: 4.6874 - 69ms/epoch - 14ms/step
Epoch 10/100
5/5 - 0s - loss: 105713.9531 - val_loss: 4.6534 - 62ms/epoch - 12ms/step
Epoch 11/100
5/5 - 0s - loss: 105667.2891 - val_loss: 4.6303 - 42ms/epoch - 8ms/step
Epoch 12/100
5/5 - 0s - loss: 105615.7656 - val_loss: 4.6152 - 60ms/epoch - 12ms/step
Epoch 13/100
5/5 - 0s - loss: 105548.8672 - val_loss: 4.6045 - 57ms/epoch - 11ms/step
Epoch 14/100
5/5 - 0s - loss: 105460.6328 - val_loss: 4.5924 - 41ms/epoch - 8ms/step
Epoch 15/100
5/5 - 0s - loss: 105406.0938 - val_loss: 4.5824 - 60ms/epoch - 12ms/step
Epoch 16/100
5/5 - 0s - loss: 105330.8359 - val_loss: 4.5703 - 50ms/epoch - 10ms/step
Epoch 17/100
5/5 - 0s - loss: 105228.0312 - val_loss: 4.5537 - 57ms/epoch - 11ms/step
Epoch 18/100
5/5 - 0s - loss: 105192.6484 - val_loss: 4.5395 - 49ms/epoch - 10ms/step
Epoch 19/100
5/5 - 0s - loss: 105052.2344 - val_loss: 4.5221 - 46ms/epoch - 9ms/step
Epoch 20/100
5/5 - 0s - loss: 105012.4844 - val_loss: 4.5083 - 62ms/epoch - 12ms/step
Epoch 21/100
5/5 - 0s - loss: 104841.6250 - val_loss: 4.4851 - 42ms/epoch - 8ms/step
Epoch 22/100
5/5 - 0s - loss: 104783.8359 - val_loss: 4.4640 - 59ms/epoch - 12ms/step
Epoch 23/100
5/5 - 0s - loss: 104617.6719 - val_loss: 4.4456 - 84ms/epoch - 17ms/step
Epoch 24/100
5/5 - 0s - loss: 104490.6250 - val_loss: 4.4228 - 54ms/epoch - 11ms/step
Epoch 25/100
5/5 - 0s - loss: 104343.7344 - val_loss: 4.3926 - 51ms/epoch - 10ms/step
Epoch 26/100
5/5 - 0s - loss: 104218.5469 - val_loss: 4.3695 - 76ms/epoch - 15ms/step
Epoch 27/100
5/5 - 0s - loss: 104157.5234 - val_loss: 4.3463 - 61ms/epoch - 12ms/step
Epoch 28/100
5/5 - 0s - loss: 103923.1094 - val_loss: 4.3150 - 57ms/epoch - 11ms/step
Epoch 29/100
5/5 - 0s - loss: 103804.6953 - val_loss: 4.2917 - 42ms/epoch - 8ms/step
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code']
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descripti

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('Operation_Text')['mean_time_taken_operation'].to_dict()


# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()

    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return

    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_me

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred =  operation_machine_mean_time_dict[operation_text]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 2 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('MACHINING', 'EB-28B-WMILL')
predict_delay('ROUGH TURNING', 'HB-9A-HLATHE')
print("\n\n\n\n")
```

```
1/1 [==============================] - 0s 23ms/step
Predicted Delay: No
Estimated Time: 17.13 hours
1/1 [==============================] - 0s 25ms/step
Predicted Delay: No
Estimated Time: 633.61 hours





/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: THREADING
Enter machine description: HB-11A-HLATHE
1/1 [==============================] - 0s 36ms/step
Predicted Delay: No
Estimated Time: 610.52 hours
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
print('\nNew job code\n')
```

```
New job code
```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/Leading_nut_body.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

print(df.shape)

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_oper

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_machi

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))

# Select only numerical columns before checking for infinite values
numerical_df = df.select_dtypes(include=np.number)

# Apply any() along the correct axis (axis=1 for rows) to get a boolean index for each row
inf_rows = numerical_df.index[np.isinf(numerical_df).any(axis=1)]

# Print the row indices with infinity values
print("Row indices with infinity values:")
print(inf_rows)

# Remove rows with infinity values
df = df.drop(inf_rows)

print("Shape of DataFrame after removing infinity rows:", df.shape)
```

```
0    HB-9A-HLATHE       BF      1144     30          6       6  ROUGH TURNING
1   HB-11A-HLATHE       BF      1121     40          9       9      THREADING
2    HB-9A-HLATHE       BF      1120     30          6       6  ROUGH TURNING
3   EB-60B-RDRILL       BF      1077     20          2       4       DRILLING
4    LB-43-RDRILL       BF      1068     20          2       4       DRILLING

  ShiftNumber   StartDate StartTime  FinishDate FinishTime        Document No  \
0  202310G062  28.10.2023  14:00:00  28.10.2023   22:00:00  B-603197-B-603180
1  202310G062  31.10.2023  06:00:00  31.10.2023   10:00:00  B-603197-B-603180
2  202310G062  31.10.2023  06:00:00  31.10.2023   10:00:00  B-603197-B-603180
3  202310G062  27.04.2024  08:00:00  27.04.2024   10:00:00  B-603197-B-603180
4  202310G062  27.04.2024  06:00:00  27.04.2024   08:00:00  B-603197-B-603180

       start_datetime     finish_datetime  time_taken
0 2023-10-28 14:00:00 2023-10-28 22:00:00         8.0
1 2023-10-31 06:00:00 2023-10-31 10:00:00         4.0
2 2023-10-31 06:00:00 2023-10-31 10:00:00         4.0
3 2024-04-27 08:00:00 2024-04-27 10:00:00         2.0
4 2024-04-27 06:00:00 2024-04-27 08:00:00         2.0
   Order No               Order Title    Material no. Work Centre  \
0   1738469  LEADINGNUTBODY(LH&RH)/BF  20310701000026       M1012
1   1738469  LEADINGNUTBODY(LH&RH)/BF  20310701000026       M1016
2   1738469  LEADINGNUTBODY(LH&RH)/BF  20310701000026       M1012
3   1738469  LEADINGNUTBODY(LH&RH)/BF  20310701000026       M1115
4   1738469  LEADINGNUTBODY(LH&RH)/BF  20310701000026       M1113

  M/c_Description Customer  JobCardN  OPrNo  OprHr(PO)  OprHrs  ...  \
0    HB-9A-HLATHE       BF      1144     30          6       6  ...
1   HB-11A-HLATHE       BF      1121     40          9       9  ...
2    HB-9A-HLATHE       BF      1120     30          6       6  ...
3   EB-60B-RDRILL       BF      1077     20          2       4  ...
4    LB-43-RDRILL       BF      1068     20          2       4  ...

         Document No      start_datetime     finish_datetime time_taken  \
0  B-603197-B-603180 2023-10-28 14:00:00 2023-10-28 22:00:00        8.0
1  B-603197-B-603180 2023-10-31 06:00:00 2023-10-31 10:00:00        4.0
2  B-603197-B-603180 2023-10-31 06:00:00 2023-10-31 10:00:00        4.0
3  B-603197-B-603180 2024-04-27 08:00:00 2024-04-27 10:00:00        2.0
4  B-603197-B-603180 2024-04-27 06:00:00 2024-04-27 08:00:00        2.0

  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                 38.097222               38.700000                6.349537
1                  6.583333                6.864865                0.731481
2                 38.097222               38.700000                6.349537
3                  3.901639                3.333333                0.975410
4                  3.901639                3.957447                0.975410

  normalized_time_taken delay_operation  delay_machine
0             1.333333           False          False
1             0.444444           False          False
2             0.666667           False          False
3             0.500000           False          False
4             0.500000           False          False

[5 rows x 26 columns]
Row indices with infinity values:
Index([124, 125, 222], dtype='int64')
Shape of DataFrame after removing infinity rows: (268, 26)
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))
```

```
# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
Epoch 1/100
6/6 - 1s - loss: 4.2211 - val_loss: 1.0167 - 907ms/epoch - 151ms/step
Epoch 2/100
6/6 - 0s - loss: 3.4740 - val_loss: 0.7558 - 43ms/epoch - 7ms/step
Epoch 3/100
6/6 - 0s - loss: 2.9046 - val_loss: 0.5949 - 44ms/epoch - 7ms/step
Epoch 4/100
6/6 - 0s - loss: 2.4190 - val_loss: 0.4518 - 43ms/epoch - 7ms/step
Epoch 5/100
6/6 - 0s - loss: 1.9543 - val_loss: 0.3457 - 43ms/epoch - 7ms/step
Epoch 6/100
6/6 - 0s - loss: 1.5520 - val_loss: 0.2715 - 42ms/epoch - 7ms/step
Epoch 7/100
6/6 - 0s - loss: 1.2466 - val_loss: 0.2258 - 41ms/epoch - 7ms/step
Epoch 8/100
6/6 - 0s - loss: 0.9397 - val_loss: 0.2099 - 41ms/epoch - 7ms/step
Epoch 9/100
6/6 - 0s - loss: 0.6851 - val_loss: 0.1941 - 58ms/epoch - 10ms/step
Epoch 10/100
6/6 - 0s - loss: 0.5404 - val_loss: 0.1695 - 40ms/epoch - 7ms/step
Epoch 11/100
6/6 - 0s - loss: 0.3640 - val_loss: 0.1266 - 46ms/epoch - 8ms/step
Epoch 12/100
6/6 - 0s - loss: 0.2405 - val_loss: 0.0891 - 59ms/epoch - 10ms/step
Epoch 13/100
6/6 - 0s - loss: 0.1664 - val_loss: 0.0604 - 63ms/epoch - 11ms/step
Epoch 14/100
6/6 - 0s - loss: 0.1096 - val_loss: 0.0337 - 58ms/epoch - 10ms/step
Epoch 15/100
6/6 - 0s - loss: 0.0601 - val_loss: 0.0180 - 61ms/epoch - 10ms/step
Epoch 16/100
6/6 - 0s - loss: 0.0353 - val_loss: 0.0125 - 44ms/epoch - 7ms/step
Epoch 17/100
6/6 - 0s - loss: 0.0285 - val_loss: 0.0109 - 43ms/epoch - 7ms/step
Epoch 18/100
6/6 - 0s - loss: 0.0260 - val_loss: 0.0097 - 43ms/epoch - 7ms/step
Epoch 19/100
6/6 - 0s - loss: 0.0241 - val_loss: 0.0086 - 43ms/epoch - 7ms/step
Epoch 20/100
6/6 - 0s - loss: 0.0222 - val_loss: 0.0079 - 52ms/epoch - 9ms/step
Epoch 21/100
6/6 - 0s - loss: 0.0207 - val_loss: 0.0075 - 75ms/epoch - 12ms/step
Epoch 22/100
6/6 - 0s - loss: 0.0192 - val_loss: 0.0069 - 66ms/epoch - 11ms/step
Epoch 23/100
6/6 - 0s - loss: 0.0174 - val_loss: 0.0067 - 64ms/epoch - 11ms/step
Epoch 24/100
6/6 - 0s - loss: 0.0162 - val_loss: 0.0065 - 43ms/epoch - 7ms/step
Epoch 25/100
6/6 - 0s - loss: 0.0153 - val_loss: 0.0064 - 53ms/epoch - 9ms/step
Epoch 26/100
6/6 - 0s - loss: 0.0141 - val_loss: 0.0064 - 57ms/epoch - 10ms/step
Epoch 27/100
6/6 - 0s - loss: 0.0136 - val_loss: 0.0059 - 45ms/epoch - 8ms/step
Epoch 28/100
6/6 - 0s - loss: 0.0118 - val_loss: 0.0058 - 47ms/epoch - 8ms/step
Epoch 29/100
6/6 - 0s - loss: 0.0108 - val_loss: 0.0053 - 43ms/epoch - 7ms/step
```

```
# Create the necessary mappings again
operation_text_map = df[['Operation_Text', 'Operation Text Code']].drop_duplicates().set_index('Operation_Text')['Operation Text Code']
machine_description_map = df[['M/c_Description', 'M/c Description Code']].drop_duplicates().set_index('M/c_Description')['M/c Descripti

# Create a dictionary for standard operation mean times
operation_mean_time_dict = df.set_index('Operation_Text')['standard_operation_mean'].to_dict()
operation_machine_mean_time_dict = df.set_index('Operation_Text')['mean_time_taken_operation'].to_dict()


# Define the function to predict delay and estimate time of delay
def predict_delay(operation_text, machine_description):
    # Convert inputs to string and strip leading/trailing whitespaces
    operation_text = str(operation_text).strip()
    machine_description = str(machine_description).strip()

    # Handle cases where operation_text or machine_description are not in the mapping
    if operation_text not in operation_text_map:
        print(f"Warning: Operation '{operation_text}' not found in training data. Prediction may be inaccurate.")
        return

    if machine_description not in machine_description_map:
        print(f"Warning: Machine '{machine_description}' not found in training data. Prediction may be inaccurate.")
        return

    # Convert categorical features to numerical codes
    operation_code = operation_text_map[operation_text]
    machine_code = machine_description_map[machine_description]

    # Prepare the feature array
    feature_array = np.array([[operation_code, machine_code, 0, 0]])  # Placeholder for normalized_time_taken and standard_operation_me

    # Standardize the feature array
    feature_array_standardized = scaler.transform(feature_array)

    # Predict the normalized time taken using the ANN model
    normalized_time_taken_pred = model_ann.predict(feature_array_standardized)[0][0]

    # Calculate the actual time taken based on the operation's standard operation mean
    standard_operation_mean = operation_mean_time_dict.get(operation_text, None)
    if standard_operation_mean is None:
        print(f"Warning: Standard operation mean for '{operation_text}' not found. Cannot estimate time.")
        return

    time_taken_pred =  operation_machine_mean_time_dict[operation_text]

    # Calculate the delay based on 1.05 threshold
    delay_operation = normalized_time_taken_pred > 2 * standard_operation_mean

    # Print the results
    if delay_operation:
        print(f"Predicted Delay: Yes")
        print(f"Estimated Time: {time_taken_pred - standard_operation_mean:.2f} hours")
    else:
        print(f"Predicted Delay: No")
        print(f"Estimated Time: {time_taken_pred:.2f} hours")

# Example usage
predict_delay('MACHINING', 'EB-28B-WMILL')
predict_delay('ROUGH TURNING', 'HB-9A-HLATHE')
print("\n\n\n\n")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
1/1 [==============================] - 0s 37ms/step
Predicted Delay: No
Estimated Time: 4.58 hours
1/1 [==============================] - 0s 32ms/step
Predicted Delay: No
Estimated Time: 38.10 hours





/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
op = input("Enter operation text: ")
mc = input("Enter machine description: ")
predict_delay(op, mc)
print("\n\n\n\n")
```

```
Enter operation text: THREADING
Enter machine description: HB-11A-HLATHE
1/1 [==============================] - 0s 23ms/step
Predicted Delay: No
Estimated Time: 6.58 hours
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler w
  warnings.warn(
```

```
print('\nNew Job Code\n')
```

```
New Job Code
```

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = '/content/INSERT_RSM.xlsx'
df = pd.read_excel(file_path)

# Check if all jobs have the same document number
if df['Document No'].nunique() != 1:
    raise ValueError("Not all jobs have the same document number.")

# Convert date and time columns to datetime objects
# Convert StartTime and FinishTime to string before concatenation
df['start_datetime'] = pd.to_datetime(df['StartDate'] + ' ' + df['StartTime'].astype(str), format='%d.%m.%Y %H:%M:%S')
df['finish_datetime'] = pd.to_datetime(df['FinishDate'] + ' ' + df['FinishTime'].astype(str), format='%d.%m.%Y %H:%M:%S')

# Calculate the time taken for each job
df['time_taken'] = (df['finish_datetime'] - df['start_datetime']).dt.total_seconds() / 3600.0  # time in hours

print(df.shape)

# Display the first few rows to check the data
print(df.head())

# Calculate mean time taken for each operation
operation_mean_time = df.groupby('Operation_Text')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_op

# Calculate mean time taken on each machine
machine_mean_time = df.groupby('M/c_Description')['time_taken'].mean().reset_index().rename(columns={'time_taken': 'mean_time_taken_mac

# Merge mean times back to the original dataframe
# The following two lines were previously commented out, causing the error
df = df.merge(operation_mean_time, on='Operation_Text', how='left')
df = df.merge(machine_mean_time, on='M/c_Description', how='left')

# Define standard_operation_mean by dividing mean_time_taken_operation by OpHr(PO)
df['standard_operation_mean'] = df['mean_time_taken_operation'] / df['OprHrs']

# Identify jobs with high delays
df['normalized_time_taken'] = df['time_taken'] / df['OprHrs']
df['delay_operation'] = df['normalized_time_taken'] > 1.05 * df['standard_operation_mean']
df['delay_machine'] = df['time_taken'] > 1.05 * df['mean_time_taken_machine']

# Display the first few rows to check the data
print(df.head(5))

# Select only numerical columns before checking for infinite values
numerical_df = df.select_dtypes(include=np.number)

# Apply any() along the correct axis (axis=1 for rows) to get a boolean index for each row
inf_rows = numerical_df.index[np.isinf(numerical_df).any(axis=1)]

# Print the row indices with infinity values
print("Row indices with infinity values:")
print(inf_rows)

# Remove rows with infinity values
df = df.drop(inf_rows)

print("Shape of DataFrame after removing infinity rows:", df.shape)
```

```
0      1187    220          3       4                           EXTRA M/C
1      1136     20          8       8                 MACHINING &  BORING
2      1100    210          6       6                           EXTRA M/C
3      1043     40          2       2  DRILL 2 HOLES D26 & COUNTER 50*50
4       968    210          6       6                           EXTRA M/C

  ShiftNumber   StartDate StartTime  FinishDate FinishTime      Document No  \
0  202309G036  31.10.2023  06:00:00  31.10.2023   14:00:00  RSM-20/12AREV-3
1  202309G036  18.10.2023  06:00:00  28.10.2023   22:00:00  RSM-20/12AREV-3
2  202309G036  28.10.2023  14:00:00  28.10.2023   18:00:00  RSM-20/12AREV-3
3  202309G036  29.03.2024  14:00:00  29.03.2024   22:00:00  RSM-20/12AREV-3
4  202309G036  25.10.2023  14:00:00  25.10.2023   22:00:00  RSM-20/12AREV-3

       start_datetime     finish_datetime  time_taken
0 2023-10-31 06:00:00 2023-10-31 14:00:00         8.0
1 2023-10-18 06:00:00 2023-10-28 22:00:00       256.0
2 2023-10-28 14:00:00 2023-10-28 18:00:00         4.0
3 2024-03-29 14:00:00 2024-03-29 22:00:00         8.0
4 2023-10-25 14:00:00 2023-10-25 22:00:00         8.0
   Order No Order Title    Material no. Work Centre M/c_Description Customer  \
0   1737881  INSERT/RSM  20811901000285       M1044      HB-3A-VBOR      RSM
1   1737881  INSERT/RSM  20811901000285       M1044      HB-3A-VBOR      RSM
2   1737881  INSERT/RSM  20811901000285       M1056      EB-5D-HBOR      RSM
3   1737881  INSERT/RSM  20811901000285       M1240   EB-60C-RDRILL      RSM
4   1737881  INSERT/RSM  20811901000285       M1088    EB-28C-PMILL      RSM

   JobCardN  OPrNo  OprHr(PO)  OprHrs  ...      Document No  \
0      1187    220          3       4  ...  RSM-20/12AREV-3
1      1136     20          8       8  ...  RSM-20/12AREV-3
2      1100    210          6       6  ...  RSM-20/12AREV-3
3      1043     40          2       2  ...  RSM-20/12AREV-3
4       968    210          6       6  ...  RSM-20/12AREV-3

       start_datetime     finish_datetime time_taken  \
0 2023-10-31 06:00:00 2023-10-31 14:00:00        8.0
1 2023-10-18 06:00:00 2023-10-28 22:00:00      256.0
2 2023-10-28 14:00:00 2023-10-28 18:00:00        4.0
3 2024-03-29 14:00:00 2024-03-29 22:00:00        8.0
4 2023-10-25 14:00:00 2023-10-25 22:00:00        8.0

  mean_time_taken_operation mean_time_taken_machine standard_operation_mean  \
0                  9.411765               33.066667                2.352941
1                 41.904762               33.066667                5.238095
2                  9.411765               20.000000                1.568627
3                 15.272727               10.571429                7.636364
4                  9.411765                7.916667                1.568627

  normalized_time_taken delay_operation  delay_machine
0              2.000000           False          False
1             32.000000            True           True
2              0.666667           False          False
3              4.000000           False          False
4              1.333333           False          False

[5 rows x 26 columns]
Row indices with infinity values:
Index([16, 61], dtype='int64')
Shape of DataFrame after removing infinity rows: (142, 26)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Prepare data for modeling
df['Operation Text Code'] = df['Operation_Text'].astype('category').cat.codes
df['M/c Description Code'] = df['M/c_Description'].astype('category').cat.codes
features = ['Operation Text Code', 'M/c Description Code', 'normalized_time_taken', 'standard_operation_mean']
X = df[features]
y = df['time_taken']/ df['OprHrs']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Build the ANN model
model_ann = Sequential()
model_ann.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dense(units=1))

# Compile the model
model_ann.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)

# Predict and evaluate the model
y_pred_ann = model_ann.predict(X_test)
mae_ann = mean_absolute_error(y_test, y_pred_ann)
mse_ann = mean_squared_error(y_test, y_pred_ann)
rmse_ann = np.sqrt(mse_ann)
r2_ann = r2_score(y_test, y_pred_ann)

print(f'ANN - Mean Absolute Error: {mae_ann}')
print(f'ANN - Mean Squared Error: {mse_ann}')
print(f'ANN - Root Mean Squared Error: {rmse_ann}')
print(f'ANN - R-squared: {r2_ann:.4f}')
```

```
3/3 - 0s - loss: 0.1352 - val_loss: 0.4073 - 68ms/epoch - 23ms/step
Epoch 75/100
3/3 - 0s - loss: 0.1301 - val_loss: 0.3943 - 53ms/epoch - 18ms/step
Epoch 76/100
3/3 - 0s - loss: 0.1256 - val_loss: 0.3838 - 69ms/epoch - 23ms/step
Epoch 77/100
3/3 - 0s - loss: 0.1207 - val_loss: 0.3762 - 71ms/epoch - 24ms/step
Epoch 78/100
3/3 - 0s - loss: 0.1167 - val_loss: 0.3638 - 55ms/epoch - 18ms/step
Epoch 79/100
3/3 - 0s - loss: 0.1129 - val_loss: 0.3573 - 74ms/epoch - 25ms/step
Epoch 80/100
3/3 - 0s - loss: 0.1084 - val_loss: 0.3474 - 95ms/epoch - 32ms/step
Epoch 81/100
3/3 - 0s - loss: 0.1047 - val_loss: 0.3339 - 61ms/epoch - 20ms/step
Epoch 82/100
3/3 - 0s - loss: 0.1010 - val_loss: 0.3211 - 78ms/epoch - 26ms/step
Epoch 83/100
3/3 - 0s - loss: 0.0975 - val_loss: 0.3134 - 54ms/epoch - 18ms/step
Epoch 84/100
3/3 - 0s - loss: 0.0944 - val_loss: 0.3009 - 62ms/epoch - 21ms/step
```