Summary

The size of the data set has 372 rows. The TSN value is seen as an object and not float Found that the Serial number is 187, this is the unique id of the item and there should be multiple values so this fits Next I saw that the TSN which should be unique value is 177 which is similar to the Serial number and this is wrong. There should be a lot more unique values. I check to see what the desperity of the values are for the short text for code, which is the reason it went for repairs. What I found was that majority of 292 are for repairs. For the TSN I needed to find out why the unique values is so low and the reason was that per Serial Number the TSN value will be the same. I show this on code bracket 11 and 12. The TSN value that was recorded is the total of all the repairs and not split between them. By nowing this I created a new table with columns Serial Number, TSN and Number Repair. The Number Repair is the amount of rows thats there for each Serial Number. Next I need to convert the TSN to a float and in doing that I found that there are UNK values. I decided to remove those values due to not being able to do anything with that value. The next step was creating the model and the Evaluation for it is below.

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from sklearn.metrics import max_error
from sklearn.metrics import mean_squared_error
import math
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

Data Understanding

I will first just load all the data and then look at the columns, the size of the data set, the type each column is.

In [ ]:
```
df = pd.read_csv('Project.csv')
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(372, 10)
```

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['Created On', 'Short text for code', 'Item Description',
       'Serial Number', 'A/C Serial No.', 'TAT Cust. Calculated (days)',
       'TAT Supplier calculated (days)', 'Completion by date',
       'TSN Component (Off)(hours)', 'Equipment'],
      dtype='object')
```

In [ ]:
```
df.dtypes
```

Out[ ]:
```
Created On                         object
Short text for code                object
Item Description                   object
Serial Number                      object
A/C Serial No.                    float64
TAT Cust. Calculated (days)        object
TAT Supplier calculated (days)     object
Completion by date                 object
TSN Component (Off)(hours)         object
Equipment                           int64
dtype: object
```

Next I want to see what columns have unique values.

In [ ]:
```
df.select_dtypes(include=["object"]).nunique()
```

Out[ ]:
```
Created On                         297
Short text for code                  7
Item Description                     1
Serial Number                      187
TAT Cust. Calculated (days)        172
TAT Supplier calculated (days)     173
Completion by date                 234
TSN Component (Off)(hours)         177
dtype: int64
```

The above data makes sense due to the main component that should not have a lot of unique values which is Short text for code and the Item Description. The Short text for code is the reason it went into repair and the Item Description is the type of item it is.

Take note that there are 372 lines in the dataset, however there are only 187 different Serial Numbers (it's the id), and there are only 177 different TSN Components.

Here I will check what the Unique values are for the columns with the least unique values.

In [ ]:
```
df['Item Description'].unique()
```

```
C:\Users\jan\AppData\Roaming\Python\Python311\site-packages\IPython\core\displayhook.py:281: UserWarning: Output cache limit (currently 1000 entries) hit.
Flushing oldest 200 entries.
  warn('Output cache limit (currently {sz} entries) hit.\n'
```

Out[ ]:
```
array(['ACTUATOR,SERVO'], dtype=object)
```

In [ ]:
```
df['Short text for code'].unique()
```

Out[ ]:
```
array(['MODIFICATION', 'REPAIR', 'NO FAULT FOUND', 'WARRANTY', nan,
       'RE-QUALIFICATION', 'INSPECTION / TEST', 'INVESTIGATION'],
      dtype=object)
```

In [ ]:
```
df.sample(10)
```

Out[ ]:

	Created On	Short text for code	Item Description	Serial Number	A/C Serial No.	TAT Cust. Calculated (days)	TAT Supplier calculated (days)	Con
215	04/11/2021	REPAIR	ACTUATOR,SERVO	1176	265.0	168	133	21/
326	06/09/2023	REPAIR	ACTUATOR,SERVO	1058	178.0	153	117	12/
229	28/02/2022	REPAIR	ACTUATOR,SERVO	1121	265.0	242	204	28/
42	27/03/2017	REPAIR	ACTUATOR,SERVO	21	110.0	105	78	18/
130	29/04/2020	REPAIR	ACTUATOR,SERVO	1127	275.0	189	159	04/
85	18/03/2019	REPAIR	ACTUATOR,SERVO	1093	230.0	210	188	14/
167	31/12/2020	REPAIR	ACTUATOR,SERVO	1047	199.0	180	134	05/
43	30/03/2017	REPAIR	ACTUATOR,SERVO	18	109.0	102	76	18/
302	04/06/2023	REPAIR	ACTUATOR,SERVO	1078	210.0	141	97	25/
298	22/05/2023	REPAIR	ACTUATOR,SERVO	72	151.0	283	142	29/

Above you can see the first 10 samples of the dataset and this is to gain an idea of the data. The main feature of the dataset is the TSN Component. The unique value for the part is the Serial Number. Short text for code is the reason it went into repair.

Next I want to see the amount of data for each short text for code. This is to see if there is a large distribution between the values.

In [ ]:
```
df.groupby(['Short text for code'])['Short text for code'].count()
```

Out[ ]:
```
Short text for code
INSPECTION / TEST     10
INVESTIGATION          1
MODIFICATION           8
NO FAULT FOUND        23
RE-QUALIFICATION       1
REPAIR               292
WARRANTY              18
Name: Short text for code, dtype: int64
```

In the group above we can see that majority of the parts problem was that it needed a repair which is 292, the second most is 23 for No Fault Found. From this I will mostly just focus on the Repair and for now I will not use this to determine the accuracy in the algorithm.

From what we saw before there are only 177 different TSN unique values which is strange with the amount of rows in the data set. I went and inspected the datasheet and from what I saw looking through the excel sheet is that the TSN has the same date for a single serial number. I will show it below.

In [ ]:
```
specific_id = '19'
```

```
lines_for_specific_id = df[df['Serial Number'] == specific_id]

lines_for_specific_id
```

Out[ ]:

	Created On	Short text for code	Item Description	Serial Number	A/C Serial No.	TAT Cust. Calculated (days)	TA Suppli calculat (day
0	09/07/2013	MODIFICATION	ACTUATOR,SERVO	19	NaN	106	1
15	11/09/2014	REPAIR	ACTUATOR,SERVO	19	NaN	0	1
27	12/07/2016	REPAIR	ACTUATOR,SERVO	19	107.0	1,087	1,0
350	24/11/2023	REPAIR	ACTUATOR,SERVO	19	103.0	104	1

In [ ]:
```
lines_for_specific_id = df[df['Serial Number'] == '19']
lines_for_specific_id[['Serial Number', 'TSN Component (Off)(hours)']]
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)
0	19	497.52
15	19	497.52
27	19	497.52
350	19	497.52

Data Preperation

As seen in the example above the TSN is the same for Serial Number 19. Now I did ask the client about this and the answer I got back is that the TSN value that is shown is the total amount of hours the part has flown.

For the next step I am going to make a new data table that will contain the id, TSN, and the number of repairs/changes.

In [ ]:
```
df_new = df[['Serial Number','TSN Component (Off)(hours)']].drop_duplicates(subs
df_new = df_new.merge((df['Serial Number'].value_counts()).rename('Number Repair
df_new
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair
0	19	497.52	4
1	26	3864.18	2
2	44	999.05	1
3	47	2113.90	2
4	41	3045.10	7
...	...	...	...
358	1014	1707.34	1
359	1084	2046.07	1
362	1106	2105.60	1
365	*00063	355.40	1
367	1018	1663.32	1

187 rows × 3 columns

The above table is what I will use to determine if I can get a significant accuracy from this table.

Below I will check the number of values that have more than 1 data point Check the type the columns are check for unique values

In [ ]:
```
(df_new['Number Repair'] > 1).sum()
```

Out[ ]:
```
95
```

In [ ]:
```
df_new.dtypes
```

Out[ ]:
```
Serial Number                object
TSN Component (Off)(hours)   object
Number Repair                 int64
dtype: object
```

In [ ]:
```
df_new.nunique()
```

Out[ ]:
```
Serial Number                187
TSN Component (Off)(hours)   177
Number Repair                  8
dtype: int64
```

I want to convert the TSN to a float because they are all numbers off coarse. but to do this I need to see if there are anything in TSN that will not allow me to change it to float.

In [ ]:
```
numeric_column = pd.to_numeric(df_new['TSN Component (Off)(hours)'], errors='coe

non_numeric_values = df_new[pd.isna(numeric_column)]['TSN Component (Off)(hours)

if not non_numeric_values.empty:
    print("Non-numeric values found in 'your_column':")
```

```
    print(non_numeric_values)
else:
    print("No non-numeric values found in 'your_column'")
```

```
Non-numeric values found in 'your_column':
14     UNK
17     UNK
204    UNK
239    UNK
247    UNK
276    UNK
298    UNK
299    UNK
Name: TSN Component (Off)(hours), dtype: object
```

In [ ]:
```
df_new['TSN Component (Off)(hours)'].unique()
```

Out[ ]:
```
array(['497.52', '3864.18', '999.05', '2113.90', '3045.10', '5242.10',
       '2841.40', '1216.22', '492.10', '2730.30', '517.06', '2970.90',
       '3471.90', '2587.40', 'UNK', '4062.50', '1656.40', '4946.80',
       '2460.40', '750.45', '964.00', '812.25', '3924.10', '1521.80',
       '824.60', '1335.95', '1349.80', '4746.60', '3386.40', '1739.10',
       '1087.80', '1033.88', '0.00', '1220.00', '1967.40', '684.77',
       '1853.60', '191.95', '1133.00', '697.68', '1131.70', '997.80',
       '1259.90', '1547.96', '533.06', '577.90', '674.92', '1640.40',
       '694.57', '365.80', '1996.90', '375.00', '1462.30', '1480.80',
       '1017.63', '1194.20', '817.05', '54.20', '1021.50', '1799.60',
       '1551.38', '4283.80', '159.10', '149.80', '958.60', '1874.80',
       '1043.90', '68.90', '1495.31', '1026.62', '1417.55', '1289.20',
       '405.00', '2017.21', '671.60', '868.67', '221.00', '726.10',
       '565.47', '1164.10', '928.20', '769.52', '927.60', '709.90',
       '1897.28', '893.30', '301.40', '836.20', '1796.40', '1068.41',
       '1243.40', '1246.40', '732.50', '1188.60', '1473.00', '1052.41',
       '1157.20', '300.70', '426.40', '1666.10', '366.00', '447.80',
       '854.30', '1337.40', '1571.63', '670.72', '783.80', '711.60',
       '615.35', '799.80', '1234.71', '1167.43', '1234.50', '1133.70',
       '1380.13', '2092.25', '2821.87', '1797.37', '1845.22', '1032.80',
       '660.60', '1277.42', '1189.00', '1496.74', '1355.27', '1298.66',
       '1456.60', '837.60', '1424.20', '539.00', '1848.32', '1430.60',
       '1596.51', '1370.14', '1387.24', '192.24', '1493.10', '1384.10',
       '1704.32', '1508.20', '1811.20', '1337.00', '158.30', '974.60',
       '1458.28', '7.77', '1550.45', '103.15', '1452.65', '1919.38',
       '12.22', '1241.03', '1229.70', '307.69', '1675.52', '1419.96',
       '219.93', '189.15', '1580.58', '1361.50', '205.14', '1125.60',
       '1942.13', '55.32', '1755.35', '1627.48', '607.33', '2102.12',
       '1666.80', '1889.60', '1496.50', '2226.80', '1707.34', '2046.07',
       '2105.60', '355.40', '1663.32'], dtype=object)
```

The above two code snipits show the unique value that is in the TSN Component that is not a number. Next I will check the serial number where the value is unkown.

In [ ]:
```
tsn_numeric = pd.to_numeric(df_new['TSN Component (Off)(hours)'], errors='coerce
invalid_values = df_new[tsn_numeric.isna()]
print("Rows with non-numeric values in 'TSN Component (Off)(hours)':")
print(invalid_values)
```

```
Rows with non-numeric values in 'TSN Component (Off)(hours)':
    Serial Number TSN Component (Off)(hours)  Number Repair
14             52                        UNK             4
17         *00069                        UNK             1
204          1087                        UNK             1
239            76                        UNK             1
247        *00060                        UNK             2
276          1092                        UNK             1
298            72                        UNK             1
299            79                        UNK             1
```

In [ ]:
```
df[df['TSN Component (Off)(hours)'] == 'UNK']
```

Out[ ]:

	Created On	Short text for code	Item Description	Serial Number	A/C Serial No.	TAT Cust. Calculated (days)	TAT Supplier calculated (days)
14	07/08/2014	REPAIR	ACTUATOR,SERVO	52	143.0	53	49
17	26/11/2014	REPAIR	ACTUATOR,SERVO	*00069	142.0	114	94
34	13/11/2016	REPAIR	ACTUATOR,SERVO	52	138.0	0	140
63	08/01/2018	WARRANTY	ACTUATOR,SERVO	52	129.0	0	119
204	04/10/2021	REPAIR	ACTUATOR,SERVO	1087	205.0	317	167
238	20/04/2022	REPAIR	ACTUATOR,SERVO	52	129.0	224	194
239	20/04/2022	REPAIR	ACTUATOR,SERVO	76	149.0	153	98
247	23/06/2022	REPAIR	ACTUATOR,SERVO	*00060	133.0	147	105
276	13/02/2023	REPAIR	ACTUATOR,SERVO	1092	202.0	122	76
298	22/05/2023	REPAIR	ACTUATOR,SERVO	72	151.0	283	142
299	22/05/2023	REPAIR	ACTUATOR,SERVO	79	150.0	283	142
300	22/05/2023	WARRANTY	ACTUATOR,SERVO	*00060	150.0	295	260

From the above I found that there are a few UNK values in the dataset and the only solution I have for these values is to remove them. The reason being that there is no possible way for me to give them a TSN value and I asked the client about this and his response was that it was not recorded.

In [ ]:
```
df_new = df_new[df_new['TSN Component (Off)(hours)'] != 'UNK']
```

In [ ]:
```
tsn_numeric = pd.to_numeric(df_new['TSN Component (Off)(hours)'], errors='coerce
invalid_values = df_new[tsn_numeric.isna()]
print("Rows with non-numeric values in 'TSN Component (Off)(hours)':")
print(invalid_values)
```

```
Rows with non-numeric values in 'TSN Component (Off)(hours)':
Empty DataFrame
Columns: [Serial Number, TSN Component (Off)(hours), Number Repair]
Index: []
```

So above i have now removed all the UNK values and next I will make the TSN column a float so that it is easier to use.

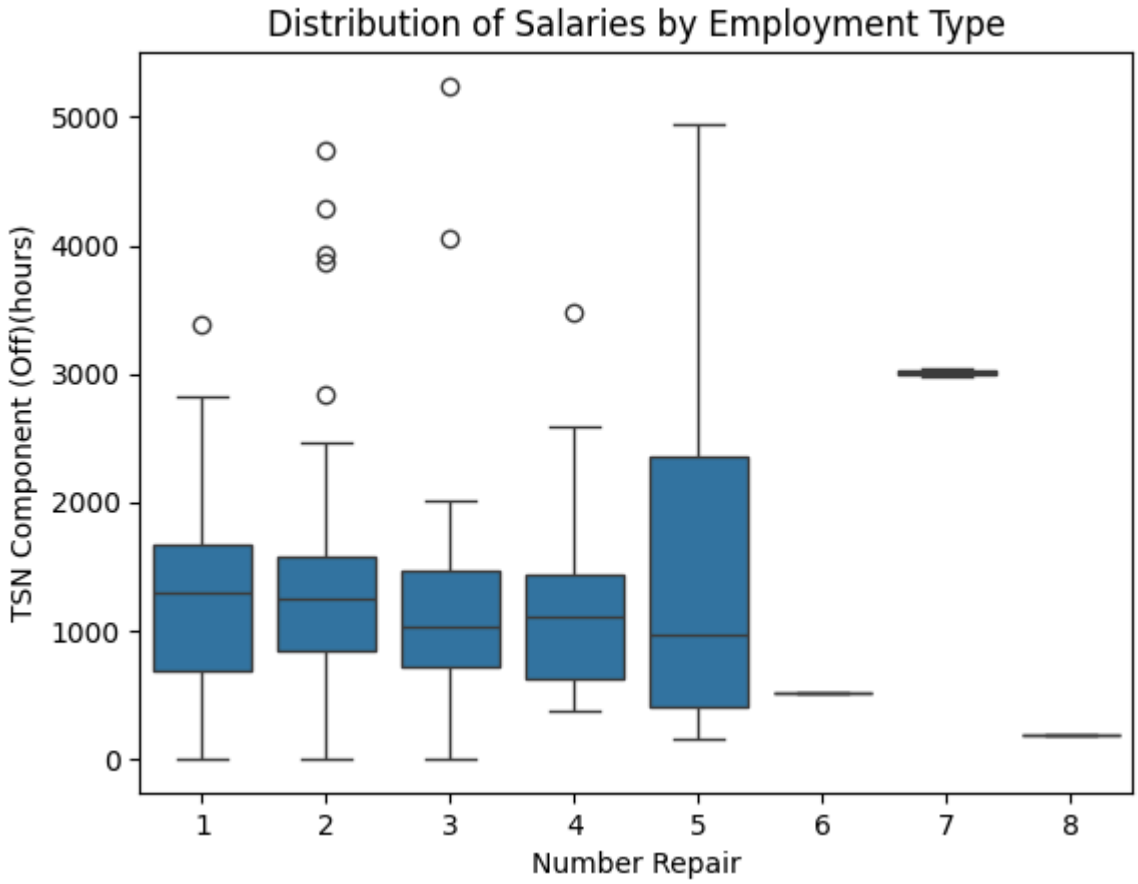
In [ ]:
```
df_new['TSN Component (Off)(hours)'] = df_new['TSN Component (Off)(hours)'].asty
df_new.dtypes
```

```
C:\Users\jan\AppData\Local\Temp\ipykernel_13788\836761620.py:1: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_new['TSN Component (Off)(hours)'] = df_new['TSN Component (Off)(hours)'].ast
ype(float)
```

Out[ ]:
```
Serial Number                 object
TSN Component (Off)(hours)   float64
Number Repair                  int64
dtype: object
```

Next I am creating a box plot to understand the distribution of the data for each number of repairs

In [ ]:
```
sns.boxplot(data=df_new, x='Number Repair', y='TSN Component (Off)(hours)', orde

plt.xlabel('Number Repair')
plt.ylabel('TSN Component (Off)(hours)')
plt.title('Distribution of Salaries by Employment Type')

plt.show()
```

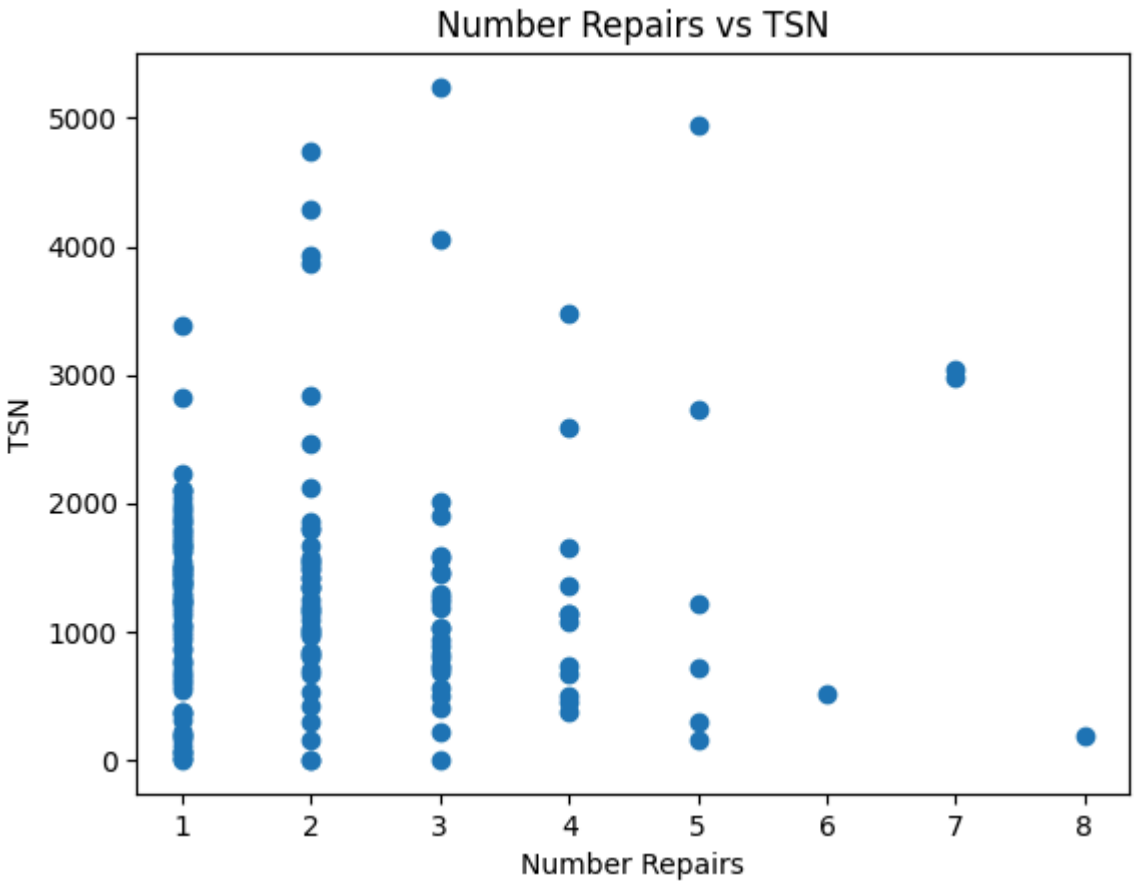


From the box plot above I can see where the median of the values are and it also shows what is consdered outliers for each Number Repair. From this I also learn that the median for is close too each other up until Number Repair 5. For 6, 7, 8 the median and box plot become to small. This means that there mught be a lot of concetrated data or now a lot of different values.

Now I want to visually see what the comparison is between the TSN and the Repair Number. To do this I am going to make a scatter plot to display it.

In [ ]:
```
plt.scatter(df_new['Number Repair'], df_new['TSN Component (Off)(hours)'], )


plt.xlabel('Number Repairs')
plt.ylabel('TSN')
plt.title('Number Repairs vs TSN')
```

Out[ ]:
```
Text(0.5, 1.0, 'Number Repairs vs TSN')
```



Above is to have a better visualization of the data compared to the box plot. This is to give me an understanding of where the data lies and an idea of the amount of points for each Number Repairs. In it I can see that the values for 6, 7, 8 are only 1 dot and this still does not give me a lot of information on those columns.

Next I will show the spread of the values between each Number Repair. This is to get an idea of where most of the data point are and which value will influence the algorithm the most.

In [ ]:
```
df_new['Number Repair'].value_counts()
```

Out[ ]:
```
Number Repair
1    86
2    45
3    26
4    12
5     6
7     2
6     1
8     1
Name: count, dtype: int64
```

From the information above I can now see that there are a small amount of values in 6, 7, 8. This also shows me that majority of the values are in 1 repair and that it decreases with repairs.

Modeling

In [ ]:
```
features = ["TSN Component (Off)(hours)"]
target = "Number Repair"
X = df_new[features]
y = df_new[target]
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("There are in total", len(X), "observations, of which", len(X_train), "are
```

```
There are in total 179 observations, of which 143 are now in the train set, and 3
6 in the test set.
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train, y_train)
pred = model.predict(X_test)

from sklearn.metrics import accuracy_score
score = accuracy_score(pred, y_test)
print("Accuracy:", score)
```

```
Accuracy: 0.5277777777777778
```

In [ ]:
```
features = ['Number Repair']
target = "TSN Component (Off)(hours)"
X = df_new[features]
y = df_new[target]
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("There are in total", len(X), "observations, of which", len(X_train), "are
```

```
There are in total 179 observations, of which 143 are now in the train set, and 3
6 in the test set.
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
model = KNeighborsRegressor()
model.fit(X_train, y_train)
pred = model.predict(X_test)

from sklearn.metrics import accuracy_score
score = mean_absolute_error(pred, y_test)
print("Mean Absolute Error::", score)

threshold = score

# Check if predictions are within threshold
accurate_predictions = sum(abs(pred - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Mean Absolute Error:: 839.5358888888888
Accuracy within threshold (839.5358888888888 away): 0.61
```

Evaluation

From the accuracy that I calculated we can see that the accuracy is really low and on average have a 50% chance of predicting the target variable. I have tested to see if it can predict the amount of repairs the and then the target variable the TSN which is the number of hours flown. On average from the running the predictions the number of repairs would have an accuracy between 0.4 to 0.5. Now this is not my target variable and is meant to be a test to see how accurate the prediction would be with the data that I have. Next the accuracy for the target variable TSN average around 0.5 to 0.6. The mosel that I used for it was the Nearest Neigbors Regression model. With this model I had to also found the absolute mean error for the prediction. With the absolute mean error you want to have it as low as possible and currently it is a large number, usually around 800 hours. What that means is that the prediction can be of by around 3 years, knowing that a plane flies around 1 hour a day.

Seen as the accuracies are so low I will begin a new process of creating a model and looking into the data again to see if there are other methods of improving the accuracy.

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

prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = predictions
prediction_overview["error"] = prediction_overview["truth"] - prediction_overvie
prediction_overview["error"] = abs(prediction_overview["error"].astype(int))
```

```
prediction_overview = prediction_overview.reset_index(drop=True)
prediction_overview
```

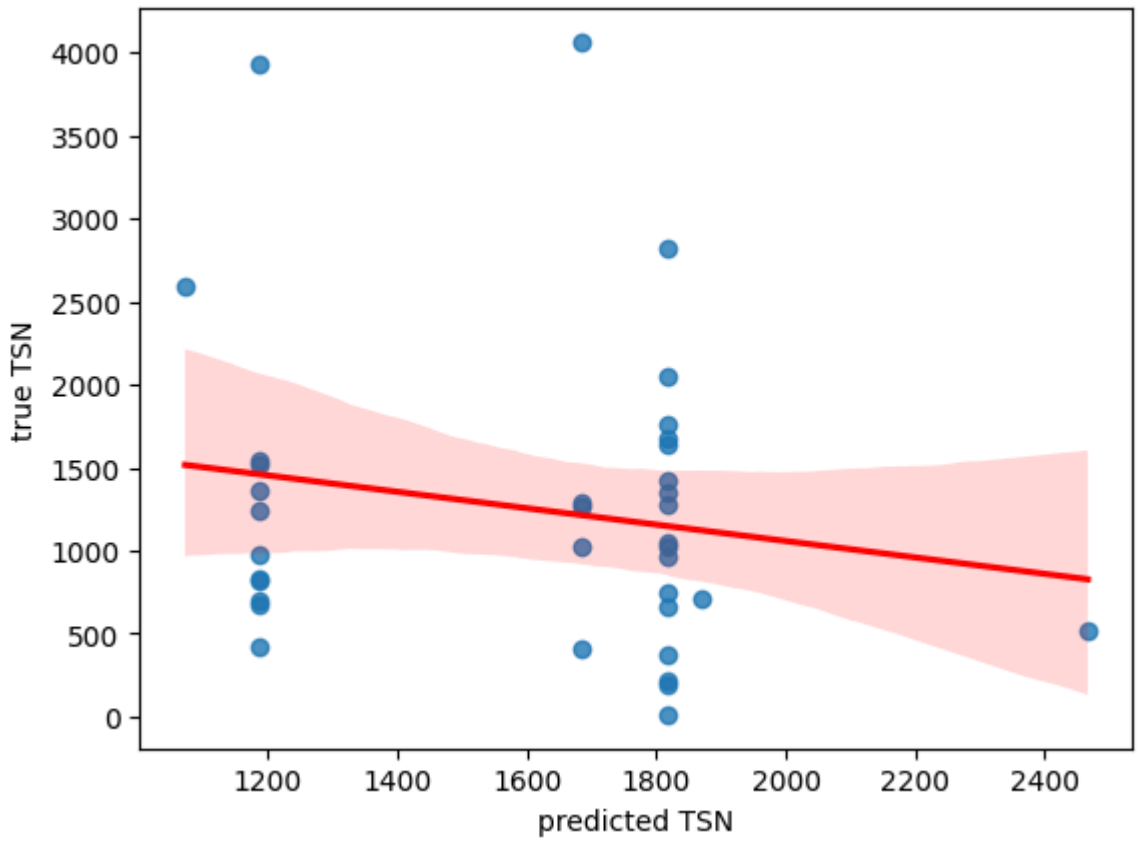
Out[ ]:

	truth	prediction	error
0	3924.10	1185.758	2738
1	219.93	1815.754	1595
2	709.90	1869.520	1159
3	974.60	1185.758	211
4	750.45	1815.754	1065
5	1361.50	1185.758	175
6	817.05	1185.758	368
7	189.15	1815.754	1626
8	1052.41	1815.754	763
9	375.00	1815.754	1440
10	660.60	1815.754	1155
11	2821.87	1815.754	1006
12	670.72	1185.758	515
13	1755.35	1815.754	60
14	964.00	1815.754	851
15	2587.40	1071.462	1515
16	1640.40	1815.754	175
17	1026.62	1815.754	789
18	836.20	1185.758	349
19	1259.90	1683.688	423
20	1289.20	1683.688	394
21	1243.40	1185.758	57
22	405.00	1683.688	1278
23	1277.42	1815.754	538
24	1021.50	1683.688	662
25	517.06	2466.520	1949
26	1355.27	1815.754	460
27	1521.80	1185.758	336
28	426.40	1185.758	759
29	1547.96	1185.758	362
30	1419.96	1815.754	395
31	12.22	1815.754	1803
32	4062.50	1683.688	2378

	truth	prediction	error
33	2046.07	1815.754	230
34	1675.52	1815.754	140
35	697.68	1185.758	488

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=predictions.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]: <Axes: xlabel='predicted TSN', ylabel='true TSN'>



In [ ]:
```
me = max_error(y_test, predictions)
me = math.ceil(me)
print("Max Error:", me)

mse = mean_squared_error(y_test, predictions)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)
```

```
Max Error: 2739
Root Mean Squared Error: 1072
```

With further inspection of the model we can see that a lot of the values lie outside of the predicted graph especially at the higher predicted values. In the graph you can see that a lot of point lie outside of the predicted area and the error margin.

With the error shown above I can see that on average the error is 1000 away from the actual. It also shows that is can be up to 3500 away from the actual. This is an indication

that the model is lacking and needs more information.

Using the KNN Regression might not fit the moddel with the limited information that it was given. For the next step I will add more data to the datasheet to increase the features that can be selected.

Data Provisioning

Showing the types again as a reminder

In [ ]:
```
df_new.dtypes
```

Out[ ]:
```
Serial Number                  object
TSN Component (Off)(hours)    float64
Number Repair                   int64
dtype: object
```

In [ ]:
```
df.dtypes
```

Out[ ]:
```
Created On                        object
Short text for code               object
Item Description                  object
Serial Number                     object
A/C Serial No.                   float64
TAT Cust. Calculated (days)       object
TAT Supplier calculated (days)    object
Completion by date                object
TSN Component (Off)(hours)        object
Equipment                          int64
dtype: object
```

Here I want to add a new column that will take the amount of days between the date the part went into repair and when it came out. I also saw that there are some that did not have a release date from the repairs yet and for those I used the current date. After finding the date I added them all up per serial number and with that I made a new column called Total Repair Days.

In [ ]:
```
from datetime import datetime
df_change = df.copy()

#convert the date to a date format
df_change["Created On"] = pd.to_datetime(df_change["Created On"], format="%d/%m/
df_change["Completion by date"] = pd.to_datetime(df_change["Completion by date"]

#calculating the days between the start and end date
current_date = datetime.now().date()
df["Completion by date"].fillna(current_date, inplace=True)

df_change["Repair Days"] = (df_change["Completion by date"] - df_change["Created

# Group by Serial Number and sum the repair days
total_repair_days = df_change.groupby("Serial Number")["Repair Days"].sum()

# Create a new column "Total Repair Days" in the original Dataset by mapping the
df_new["Total Repair Days"] = df_new["Serial Number"].map(total_repair_days)
```

```
df_new
```

```
C:\Users\jan\AppData\Local\Temp\ipykernel_13788\1611399464.py:10: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df["Completion by date"].fillna(current_date, inplace=True)
C:\Users\jan\AppData\Local\Temp\ipykernel_13788\1611399464.py:18: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_new["Total Repair Days"] = df_new["Serial Number"].map(total_repair_days)
```

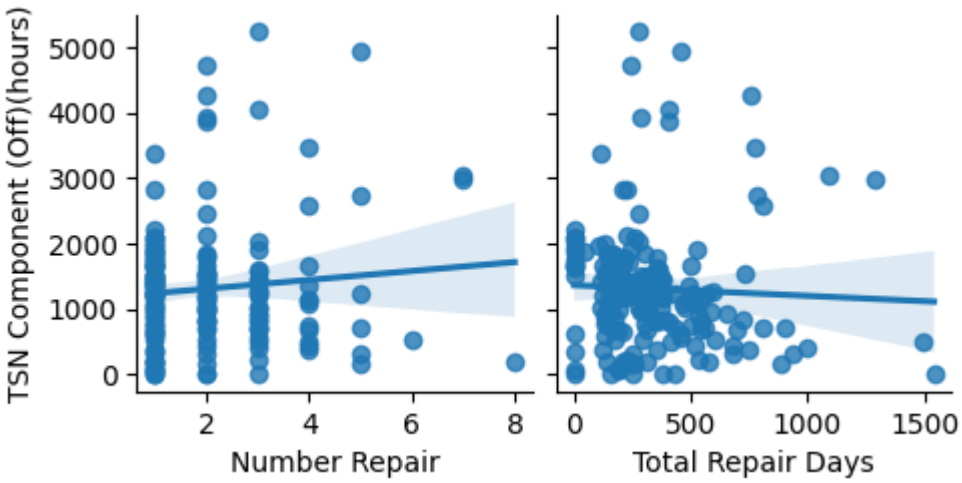
Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair	Total Repair Days
0	19	497.52	4	1492.0
1	26	3864.18	2	403.0
2	44	999.05	1	154.0
3	47	2113.90	2	228.0
4	41	3045.10	7	1088.0
...	...	...	...	...
358	1014	1707.34	1	0.0
359	1084	2046.07	1	0.0
362	1106	2105.60	1	0.0
365	*00063	355.40	1	0.0
367	1018	1663.32	1	0.0

179 rows × 4 columns

Visually show where the data lies

In [ ]:
```
features = ['Number Repair', 'Total Repair Days']
target = 'TSN Component (Off)(hours)'
plot = sns.pairplot(df_new, x_vars=features, y_vars=target, kind="reg")
```



Model

In [ ]:
```
X = df_new[features]
y = df_new[target]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("There are in total", len(X), "observations, of which", len(X_train), "are

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
There are in total 179 observations, of which 143 are now in the train set, and 3
6 in the test set.
```

In [ ]:
```
model = KNeighborsRegressor()
model.fit(X_train, y_train)
pred = model.predict(X_test)

score = mean_absolute_error(pred, y_test)
print("Mean Absolute Error::", score)

threshold = score

# Check if predictions are within threshold
accurate_predictions = sum(abs(pred - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Mean Absolute Error:: 575.3784444444443
Accuracy within threshold (575.3784444444443 away): 0.72
```

Evaluation

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

prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = predictions
prediction_overview["error"] = prediction_overview["truth"] - prediction_overvie
prediction_overview["error"] = abs(prediction_overview["error"].astype(int))
```

```
prediction_overview = prediction_overview.reset_index(drop=True)
prediction_overview
```

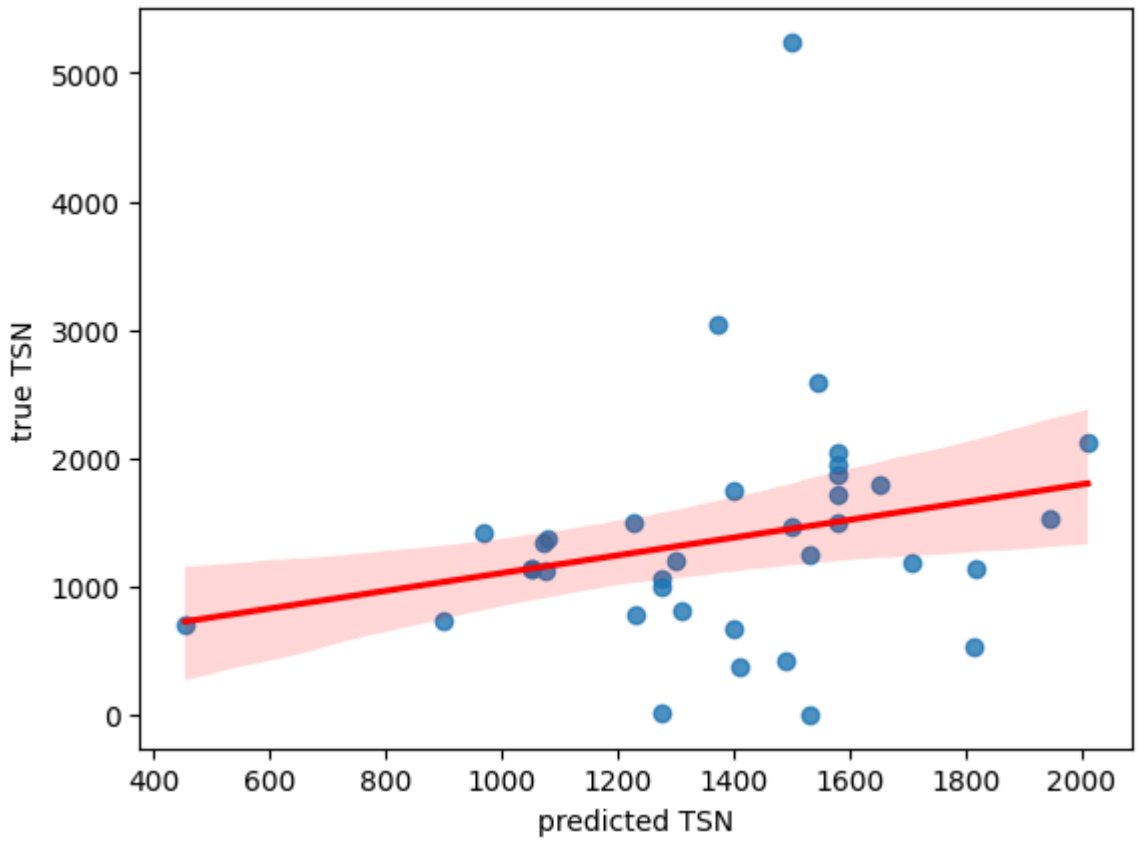
Out[ ]:

	truth	prediction	error
0	694.57	454.690	239
1	1337.40	1072.742	264
2	1133.00	1050.982	82
3	1131.70	1817.380	685
4	1125.60	1077.840	47
5	2113.90	2011.370	102
6	1739.10	1399.600	339
7	726.10	899.230	173
8	1370.14	1078.496	291
9	1133.70	1050.982	82
10	426.40	1489.172	1062
11	2587.40	1545.480	1041
12	3045.10	1372.320	1672
13	1496.74	1228.572	268
14	1496.50	1578.666	82
15	5242.10	1501.470	3740
16	12.22	1276.586	1264
17	1796.40	1653.540	142
18	1194.20	1302.394	108
19	1942.13	1578.666	363
20	1521.80	1947.176	425
21	1189.00	1709.302	520
22	671.60	1402.062	730
23	0.00	1530.272	1530
24	1874.80	1578.666	296
25	999.05	1276.586	277
26	812.25	1312.712	500
27	1424.20	970.632	453
28	2046.07	1578.666	467
29	366.00	1412.288	1046
30	783.80	1232.820	449
31	1243.40	1530.272	286
32	1052.41	1276.586	224

	truth	prediction	error
33	1707.34	1578.666	128
34	533.06	1814.130	1281
35	1462.30	1501.470	39

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=predictions.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]: <Axes: xlabel='predicted TSN', ylabel='true TSN'>



In [ ]:
```
me = max_error(y_test, predictions)
me = math.ceil(me)
print("Max Error:", me)

mse = mean_squared_error(y_test, predictions)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)
```

```
Max Error: 3741
Root Mean Squared Error: 897
```

From the new table that was created and the extra data that I used you can see that the accuracy either improves or is similar to the previouse accuracy. The biggest change is the absolute error that decrease by a lot compared to the previouse version which made the prediction much better. From running it multiple times on averegate there are a lot

of inconsistancies with the accuracy where it can range from 0.55 to 0.7 and this is better than the previouse version.

With the error I can also see that on average the mean error is still around 1000. However the Max error has decreased with the lowest being 2100. From the added data I can tell that with both the increase in accuracy and lowering of the error it was a good decision.

Data Provisioning

For this itteration I am going to add data for another plane part and see if that will improve the algorithm or work better than the previouse one. I have another data set that has not been used yet. I will now clean the new data and also combine it with the df_new to see if it will change the accuracy.

In [ ]:
```
df_comp = pd.read_csv('Project2.csv')
```

In [ ]:
```
df_comp.shape
```

Out[ ]:
```
(444, 8)
```

In [ ]:
```
df_comp.columns
```

Out[ ]:
```
Index(['Created On', 'Short text for code', 'Item Description',
       'Serial Number', 'A/C Serial No.', 'Completion by date',
       'TSN Component (Off)(hours)', 'Equipment'],
      dtype='object')
```

In [ ]:
```
df_comp.dtypes
```

Out[ ]:
```
Created On                   object
Short text for code          object
Item Description             object
Serial Number                object
A/C Serial No.               object
Completion by date           object
TSN Component (Off)(hours)   object
Equipment                     int64
dtype: object
```

In [ ]:
```
df_comp.select_dtypes(include=["object"]).nunique()
```

Out[ ]:
```
Created On                   323
Short text for code            8
Item Description               1
Serial Number                258
A/C Serial No.               157
Completion by date           226
TSN Component (Off)(hours)   237
dtype: int64
```

In [ ]:
```
df_comp['Short text for code'].unique()
```

Out[ ]:
```
array([nan, 'REPAIR', 'WARRANTY', 'INSPECTION / TEST', 'BENCHTEST',
       'NO FAULT FOUND', 'MODIFICATION', 'INVESTIGATION', 'RETURN AS IS'],
      dtype=object)
```

In [ ]:
```
df_comp.groupby(['Short text for code'])['Short text for code'].count()
```

Out[ ]:
```
Short text for code
BENCHTEST              1
INSPECTION / TEST     23
INVESTIGATION          9
MODIFICATION           2
NO FAULT FOUND        98
REPAIR               228
RETURN AS IS           8
WARRANTY              30
Name: Short text for code, dtype: int64
```

In [ ]:
```
df_comp2 = df_comp[['Serial Number','TSN Component (Off)(hours)']].drop_duplicat
df_comp2 = df_comp2.merge((df_comp['Serial Number'].value_counts()).rename('Numb
df_comp2
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair
0	S10740375	NaN	1
1	S10709288	NaN	5
2	S10709287	NaN	10
3	S10709297	NaN	2
4	S10770751	NaN	1
...	...	...	...
437	S17341948	1771.40	1
438	S18139470	1565.70	1
439	S17317846	2495.04	1
440	S17341950	1716.60	1
441	S10740387	3000.00	1

258 rows × 3 columns

Next to check if there are any problems with the TSN component and then remove the problem

In [ ]:
```
numeric_column = pd.to_numeric(df_comp2['TSN Component (Off)(hours)'], errors='c

non_numeric_values = df_comp2[pd.isna(numeric_column)]['TSN Component (Off)(hour

if not non_numeric_values.empty:
    print("Non-numeric values found in 'your_column':")
    print(non_numeric_values)
else:
    print("No non-numeric values found in 'your_column'")
```

```
Non-numeric values found in 'your_column':
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
5      NaN
6      NaN
7      NaN
8      NaN
9      NaN
10     NaN
11     NaN
12     NaN
13     NaN
19     UNK
25     UNK
37     UNK
61     UNK
84     UNK
95     UNK
104    UNK
232    UNK
385    UNK
402    UNK
403    UNK
405    UNK
Name: TSN Component (Off)(hours), dtype: object
```

In [ ]:
```
df_comp2 = df_comp2[df_comp2['TSN Component (Off)(hours)'] != 'UNK']
```

In [ ]:
```
df_comp2 = df_comp2.dropna(subset=['TSN Component (Off)(hours)'])
```

In [ ]:
```
tsn_numeric = pd.to_numeric(df_comp2['TSN Component (Off)(hours)'], errors='coer
invalid_values = df_comp2[tsn_numeric.isna()]
print("Rows with non-numeric values in 'TSN Component (Off)(hours)':")
print(invalid_values)
```

```
Rows with non-numeric values in 'TSN Component (Off)(hours)':
Empty DataFrame
Columns: [Serial Number, TSN Component (Off)(hours), Number Repair]
Index: []
```

Here I change the TSN to a float

In [ ]:
```
df_comp2['TSN Component (Off)(hours)'] = df_comp2['TSN Component (Off)(hours)'].
df_comp2.dtypes
```

Out[ ]:
```
Serial Number                 object
TSN Component (Off)(hours)    float64
Number Repair                   int64
dtype: object
```

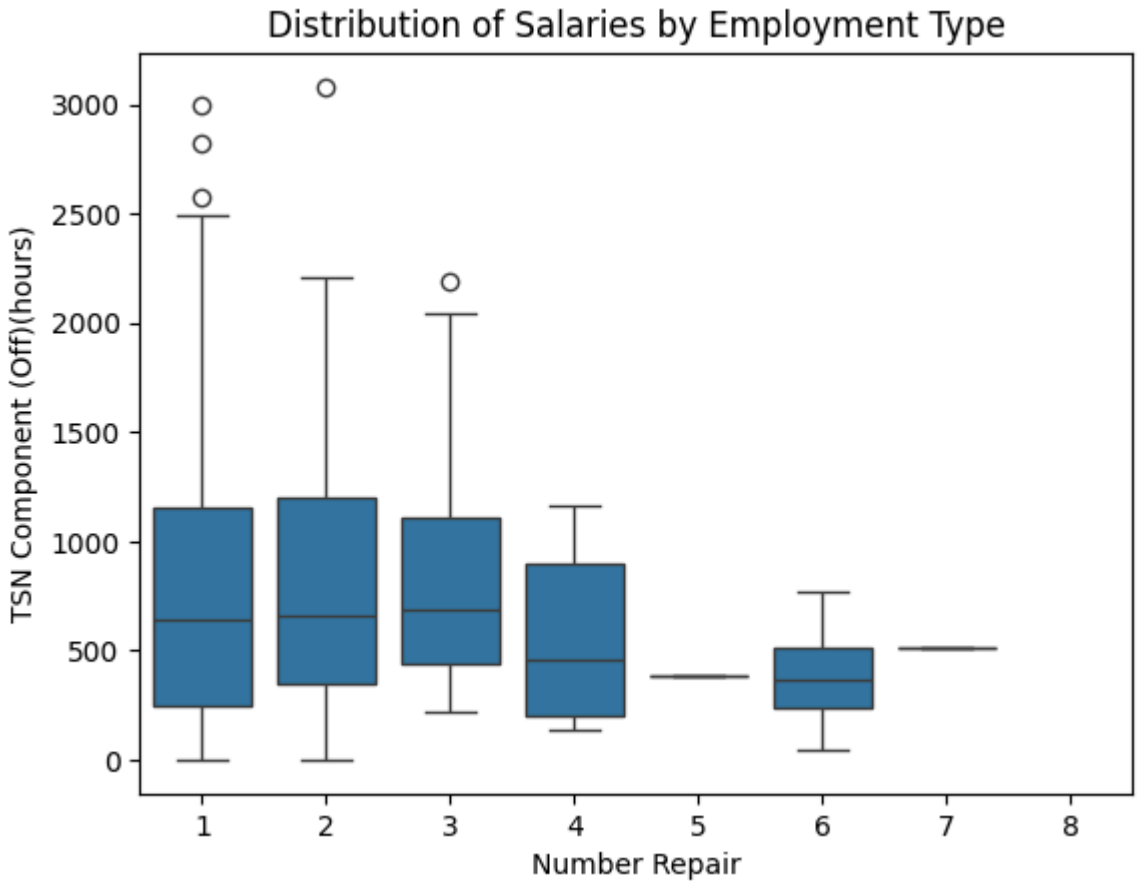
In [ ]:
```
df_comp2
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair
15	S10740390	511.30	7
17	S10740403	176.30	1
18	S10770721	114.20	1
20	S10709278	1505.00	2
21	S10709279	237.10	1
...	...	...	...
437	S17341948	1771.40	1
438	S18139470	1565.70	1
439	S17317846	2495.04	1
440	S17341950	1716.60	1
441	S10740387	3000.00	1

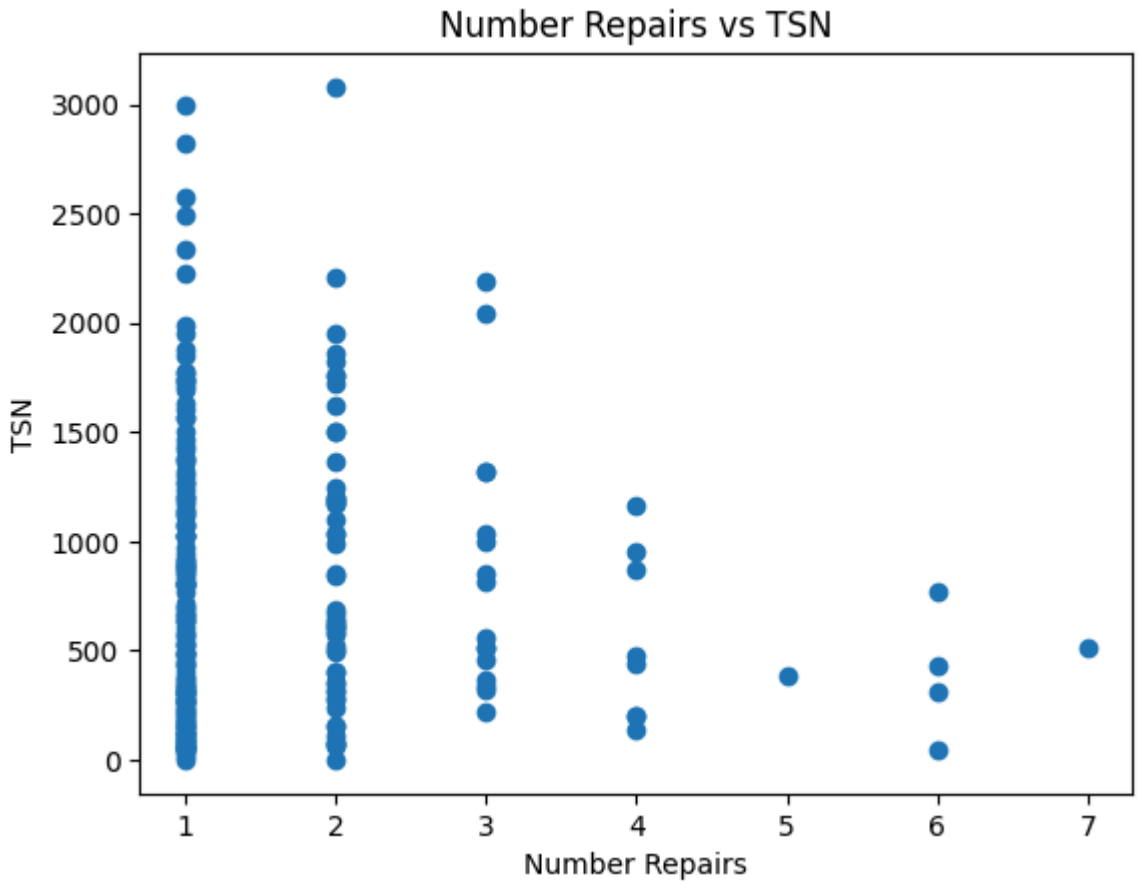
232 rows × 3 columns

In [ ]:
```
sns.boxplot(data=df_comp2, x='Number Repair', y='TSN Component (Off)(hours)', or

plt.xlabel('Number Repair')
plt.ylabel('TSN Component (Off)(hours)')
plt.title('Distribution of Salaries by Employment Type')

plt.show()
```



In [ ]:
```
plt.scatter(df_comp2['Number Repair'], df_comp2['TSN Component (Off)(hours)'], )


plt.xlabel('Number Repairs')
plt.ylabel('TSN')
plt.title('Number Repairs vs TSN')
```

Out[ ]:
```
Text(0.5, 1.0, 'Number Repairs vs TSN')
```



In [ ]:
```
df_comp2['Number Repair'].value_counts()
```

Out[ ]:
```
Number Repair
1    148
2     54
3     16
4      8
6      4
7      1
5      1
Name: count, dtype: int64
```

Creating a total repair days again

In [ ]:
```
df_change = df_comp.copy()

df_change["Created On"] = pd.to_datetime(df_change["Created On"], format="%d/%m/
df_change["Completion by date"] = pd.to_datetime(df_change["Completion by date"]

df_change["Created On"] = pd.to_datetime(df_change["Created On"], format="%Y-%m-
df_change["Completion by date"] = pd.to_datetime(df_change["Completion by date"]

current_date = datetime.now().date()
```

```
df_comp["Completion by date"].fillna(current_date, inplace=True)

df_change["Repair Days"] = (df_change["Completion by date"] - df_change["Created

# Group by Serial Number and sum the repair days
total_repair_days = df_change.groupby("Serial Number")["Repair Days"].sum()

# Create a new column "Total Repair Days" in the original Dataset by mapping the
df_comp2["Total Repair Days"] = df_comp2["Serial Number"].map(total_repair_days)

df_comp2
```

```
C:\Users\jan\AppData\Local\Temp\ipykernel_13788\1242775822.py:10: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  df_comp["Completion by date"].fillna(current_date, inplace=True)
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair	Total Repair Days
15	S10740390	511.30	7	1131.0
17	S10740403	176.30	1	109.0
18	S10770721	114.20	1	109.0
20	S10709278	1505.00	2	475.0
21	S10709279	237.10	1	153.0
...	...	...	...	...
437	S17341948	1771.40	1	0.0
438	S18139470	1565.70	1	0.0
439	S17317846	2495.04	1	0.0
440	S17341950	1716.60	1	0.0
441	S10740387	3000.00	1	0.0

232 rows × 4 columns

The next steps are to combine the two table and to do that I need to add in the name of the part.

In [ ]:
```
df_new = pd.merge(df_new, df[['Serial Number', 'Item Description']], on='Serial

df_new.rename(columns={'Item Description': 'Type'}, inplace=True)
```

In [ ]:
```
df_new.sample(10)
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair	Total Repair Days	Type
138	1093	1194.20	2	363.0	ACTUATOR,SERVO
350	1162	2102.12	1	0.0	ACTUATOR,SERVO
252	1192	670.72	2	312.0	ACTUATOR,SERVO
100	1020	684.77	3	544.0	ACTUATOR,SERVO
219	1200	1068.41	4	517.0	ACTUATOR,SERVO
68	*00058	812.25	2	404.0	ACTUATOR,SERVO
97	1012	1967.40	1	105.0	ACTUATOR,SERVO
235	1183	300.70	5	940.0	ACTUATOR,SERVO
315	1111	1384.10	1	253.0	ACTUATOR,SERVO
142	1146	54.20	1	187.0	ACTUATOR,SERVO

In [ ]:
```
df_comp2 = pd.merge(df_comp2, df_comp[['Serial Number', 'Item Description']], on

df_comp2.rename(columns={'Item Description': 'Type'}, inplace=True)
```

In [ ]:
```
df_comp2.sample(10)
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair	Total Repair Days	Type
328	S18107427	898.90	1	212.0	REMOTE IO CONCENTRATOR
220	S17341942	160.40	1	559.0	REMOTE IO CONCENTRATOR
31	S13031954	199.30	4	2338.0	REMOTE IO CONCENTRATOR
230	S16360014	341.70	3	539.0	REMOTE IO CONCENTRATOR
132	S13166133	360.37	1	226.0	REMOTE IO CONCENTRATOR
215	S17341954	459.90	3	594.0	REMOTE IO CONCENTRATOR
354	S10709302	2575.00	1	0.0	REMOTE IO CONCENTRATOR
52	S10709289	769.00	6	103.0	REMOTE IO CONCENTRATOR
344	S19200883	1024.10	1	260.0	REMOTE IO CONCENTRATOR
22	S10770736	556.10	3	488.0	REMOTE IO CONCENTRATOR

In [ ]:
```
merged_tables = pd.concat([df_new, df_comp2], ignore_index=True)
```

Now I am adding a new column that will either be a 1 or 2 and depending on the Type name

In [ ]:
```
def map_type_to_numeric(type_str):
    if 'ACTUATOR,SERVO' in type_str:
        return 1
    elif 'REMOTE IO CONCENTRATOR' in type_str:
        return 2
    else:
        return None

merged_tables['Type_numeric'] = merged_tables['Type'].apply(map_type_to_numeric)
```

In [ ]:
```
merged_tables.sample(10)
```

Out[ ]:

	Serial Number	TSN Component (Off)(hours)	Number Repair	Total Repair Days	Type	Type_numeric
258	1187	711.60	3	811.0	ACTUATOR,SERVO	1
466	S14196482	1320.90	3	551.0	REMOTE IO CONCENTRATOR	2
393	S13214140	46.10	6	1052.0	REMOTE IO CONCENTRATOR	2
266	1131	1167.43	2	376.0	ACTUATOR,SERVO	1
440	S14027061	873.90	4	877.0	REMOTE IO CONCENTRATOR	2
702	S17057458	1948.54	2	307.0	REMOTE IO CONCENTRATOR	2
682	S17001761	614.20	2	182.0	REMOTE IO CONCENTRATOR	2
113	1054	997.80	2	289.0	ACTUATOR,SERVO	1
317	1157	1508.20	1	174.0	ACTUATOR,SERVO	1
685	S18334567	694.60	1	176.0	REMOTE IO CONCENTRATOR	2

In [ ]:
```
merged_tables.dtypes
```

Out[ ]:
```
Serial Number                 object
TSN Component (Off)(hours)   float64
Number Repair                  int64
Total Repair Days            float64
Type                          object
Type_numeric                   int64
dtype: object
```

In [ ]:
```
merged_tables.shape
```

Out[ ]:
```
(732, 6)
```

In [ ]:
```
merged_tables.nunique()
```

Out[ ]:
```
Serial Number                411
TSN Component (Off)(hours)   403
Number Repair                  8
Total Repair Days            271
Type                           2
Type_numeric                   2
dtype: int64
```

Modeling

In [ ]:
```
predictions = model.predict(df_comp2[['Number Repair', 'Total Repair Days']])

prediction_overview = pd.DataFrame()
prediction_overview["truth"] = df_comp2['TSN Component (Off)(hours)']
```

```
prediction_overview["prediction"] = predictions
prediction_overview["error"] = prediction_overview["truth"] - prediction_overvie
prediction_overview["error"] = abs(prediction_overview["error"].astype(int))
prediction_overview = prediction_overview.reset_index(drop=True)
prediction_overview
```

```
c:\Python311\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature na
mes, but KNeighborsRegressor was fitted without feature names
  warnings.warn(
```

Out[ ]:

	truth	prediction	error
0	511.30	834.824	323
1	511.30	834.824	323
2	511.30	834.824	323
3	511.30	834.824	323
4	511.30	834.824	323
...	...	...	...
367	1771.40	1389.348	382
368	1565.70	1389.348	176
369	2495.04	1389.348	1105
370	1716.60	1389.348	327
371	3000.00	1389.348	1610

372 rows × 3 columns

In [ ]:
```
prediction_overview.mean(axis=0)
```

Out[ ]:
```
truth         762.840376
prediction    878.127323
error         485.868280
dtype: float64
```

In [ ]:
```
threshold = mean_absolute_error(predictions, df_comp2['TSN Component (Off)(hours
accurate_predictions = sum(abs(predictions - df_comp2['TSN Component (Off)(hours
total_predictions = len(df_comp2['TSN Component (Off)(hours)'])
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Accuracy within threshold (486.38148387096777 away): 0.57
```

The top bit was to test how well the previouse prediction would work with the new data and as you can see it has a low percentage of working.

In [ ]:
```
X = df_comp2[features]
y = df_comp2[target]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("There are in total", len(X), "observations, of which", len(X_train), "are

scaler = StandardScaler()
```

```
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
There are in total 372 observations, of which 297 are now in the train set, and 7
5 in the test set.
```

In [ ]:
```
model = KNeighborsRegressor()
model.fit(X_train, y_train)
pred = model.predict(X_test)

score = mean_absolute_error(pred, y_test)
print("Mean Absolute Error::", score)

threshold = score

# Check if predictions are within threshold
accurate_predictions = sum(abs(pred - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Mean Absolute Error:: 408.4824
Accuracy within threshold (408.4824 away): 0.59
```

Above is the accuracy for the Computer and here you can see the the absolute mean error is lower than that of the acuator servo.

Evaluate

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

prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = predictions
prediction_overview["error"] = prediction_overview["truth"] - prediction_overvie
prediction_overview["error"] = abs(prediction_overview["error"].astype(int))
prediction_overview = prediction_overview.reset_index(drop=True)
prediction_overview
```

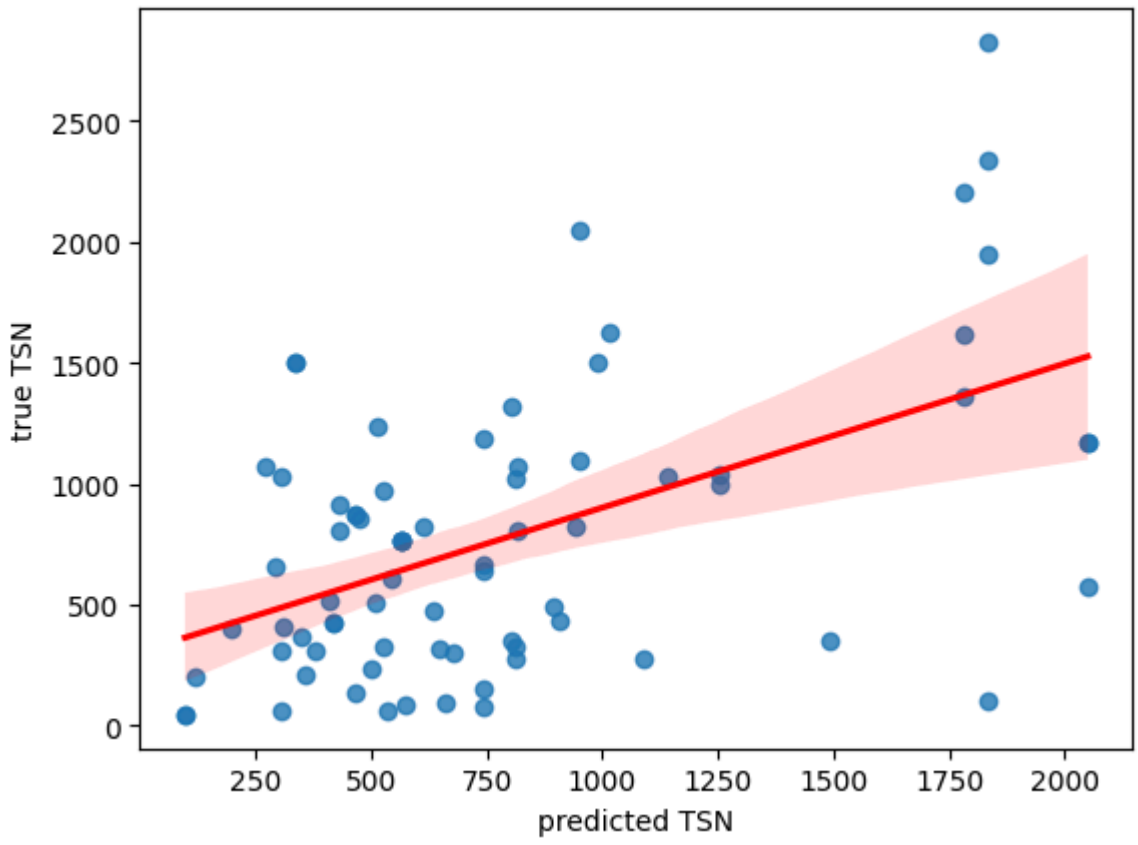
Out[ ]:

	truth	prediction	error
0	1169.30	2049.928	880
1	2338.20	1834.668	503
2	327.62	525.404	197
3	435.40	906.372	470
4	406.58	312.168	94
...	...	...	...
70	1365.10	1783.892	418
71	351.70	805.540	453
72	62.30	306.660	244
73	608.00	543.266	64
74	819.80	942.300	122

75 rows × 3 columns

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=predictions.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]:
```
<Axes: xlabel='predicted TSN', ylabel='true TSN'>
```



In [ ]:
```
me = max_error(y_test, predictions)
me = math.ceil(me)
```

```
print("Max Error:", me)

mse = mean_squared_error(y_test, predictions)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)
```

```
Max Error: 1734
Root Mean Squared Error: 543
```

With the Computer dataset I can tell that it is alot more linear than the Actuators data. The main indication of it is that the accuracy is higher on averegate compared to the Actuator and the absolute error is alos lower on average. The accuracy also ranges from .59 to .69 depending on the training data.

Next the Max error is lower than the Actuator max error by around 400 and the average mean error is also lower by around 500.

Modeling

Next I will be testing for the merged tables

In [ ]:
```
features = ["Number Repair", "Total Repair Days", "Type_numeric"]
target = "TSN Component (Off)(hours)"
X = merged_tables[features]
y = merged_tables[target]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("There are in total", len(X), "observations, of which", len(X_train), "are

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
There are in total 732 observations, of which 585 are now in the train set, and 1
47 in the test set.
```

In [ ]:
```
model = KNeighborsRegressor()
model.fit(X_train, y_train)
pred = model.predict(X_test)

score = mean_absolute_error(pred, y_test)
print("Mean Absolute Error::", score)

threshold = score

# Check if predictions are within threshold
accurate_predictions = sum(abs(pred - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Mean Absolute Error:: 491.49838095238084
Accuracy within threshold (491.49838095238084 away): 0.62
```

Evaluation

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

prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = predictions
prediction_overview["error"] = prediction_overview["truth"] - prediction_overvie
prediction_overview["error"] = abs(prediction_overview["error"].astype(int))
prediction_overview = prediction_overview.reset_index(drop=True)
prediction_overview
```

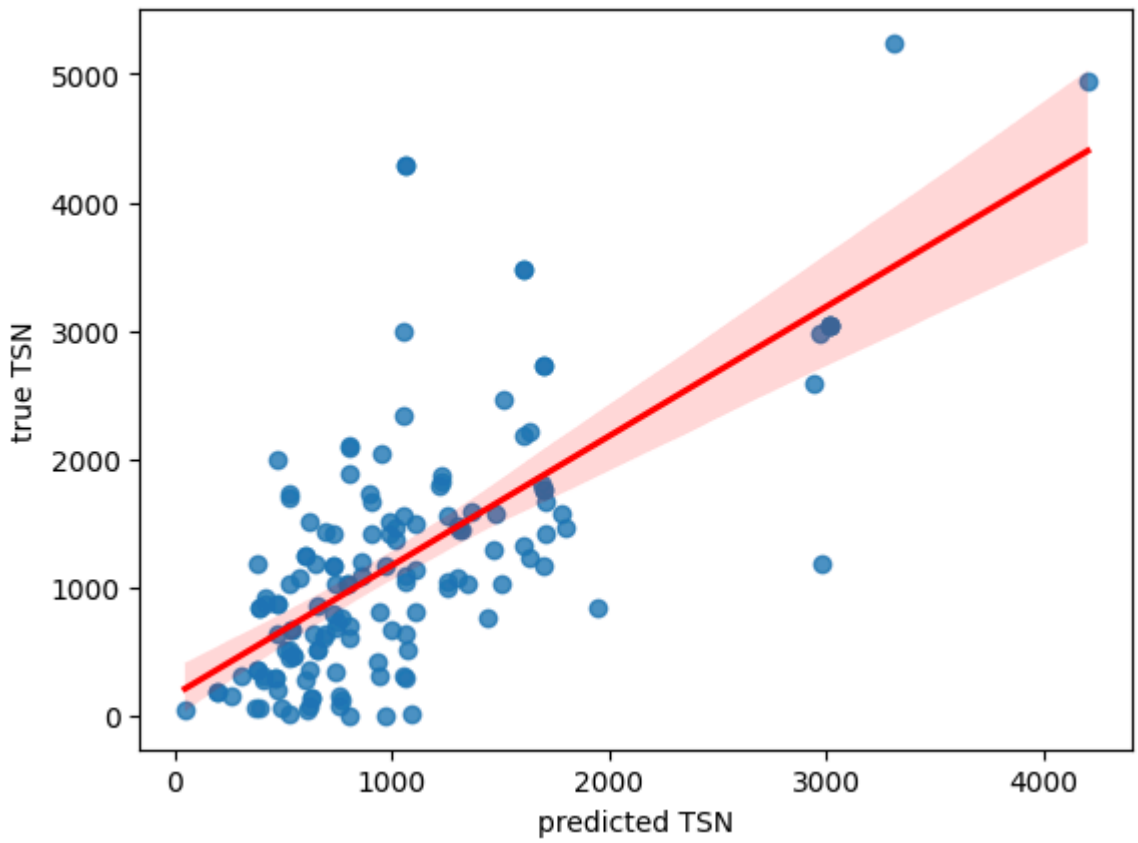
Out[ ]:

	truth	prediction	error
0	697.68	799.346	101
1	1571.63	1473.816	97
2	16.07	526.924	510
3	642.93	468.680	174
4	879.44	414.632	464
...	...	...	...
142	138.30	629.700	491
143	426.40	934.960	508
144	1043.90	1060.268	16
145	65.00	487.320	422
146	1424.00	727.480	696

147 rows × 3 columns

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=predictions.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]:
```
<Axes: xlabel='predicted TSN', ylabel='true TSN'>
```



In [ ]:
```
me = max_error(y_test, predictions)
me = math.ceil(me)
print("Max Error:", me)

mse = mean_squared_error(y_test, predictions)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)
```

```
Max Error: 3221
Root Mean Squared Error: 726
```

Combining the two tables gives by far the best predictions compared to the individual results. The accuracy for the merged table averages around .6 to .7 with an error margin of around 500. This is a lot better than the previouse results.

The mean error is also low with an average of 600 and the max error is unfortunately high upt to 2800

Modeling

For this step I will use LR to predict.

In [ ]:
```
features = ["Number Repair", "Total Repair Days", "Type_numeric"]
target = "TSN Component (Off)(hours)"
X = merged_tables[features]
y = merged_tables[target]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print("There are in total", len(X), "observations, of which", len(X_train), "are
```

There are in total 732 observations, of which 585 are now in the train set, and 147 in the test set.

In [ ]:
```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict
lr_pred = lr_model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
lr_score = mean_absolute_error(lr_pred, y_test)
print("Linear Regression Mean Absolute Error:", lr_score)

# Check if predictions are within threshold
lr_accurate_predictions = sum(abs(lr_pred - y_test) <= threshold)
lr_accuracy_within_threshold = lr_accurate_predictions / total_predictions
print("Linear Regression Accuracy within threshold ({} away): {:.2f}".format(thr
```

```
Linear Regression Mean Absolute Error: 607.8481405088866
Linear Regression Accuracy within threshold (491.49838095238084 away): 0.54
```

It is similar to Kneighbors regression

Evaluation

In [ ]:
```
predictions = lr_pred

prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = predictions
prediction_overview["error"] = prediction_overview["truth"] - prediction_overvie
prediction_overview["error"] = abs(prediction_overview["error"].astype(int))
prediction_overview = prediction_overview.reset_index(drop=True)
prediction_overview
```

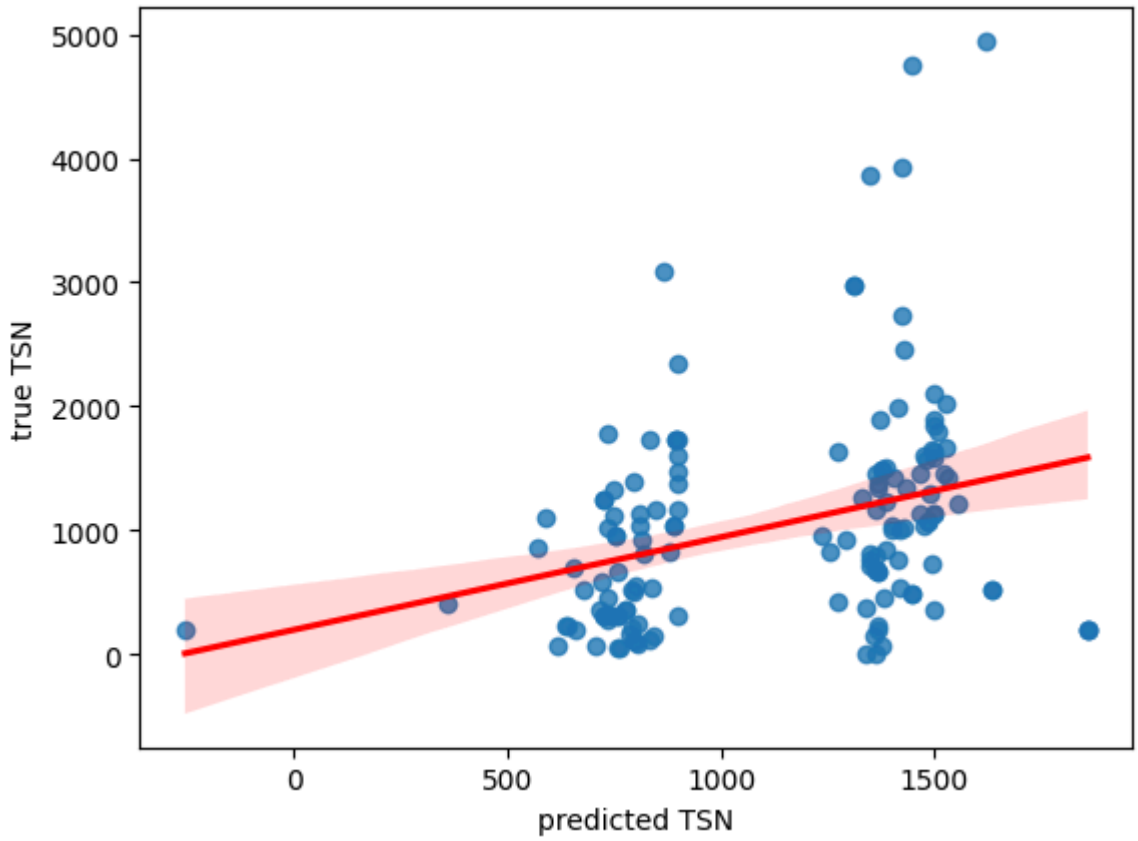
Out[ ]:

	truth	prediction	error
0	1220.00	1553.911651	333
1	3080.44	864.998764	2215
2	1889.60	1497.844308	391
3	1248.70	723.473073	525
4	556.10	800.268451	244
...	...	...	...
142	1452.65	1362.580816	90
143	221.70	638.703724	417
144	106.10	801.750557	695
145	915.20	814.504621	100
146	225.30	641.667937	416

147 rows × 3 columns

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=predictions.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]: <Axes: xlabel='predicted TSN', ylabel='true TSN'>



In [ ]:
```
me = max_error(y_test, predictions)
me = math.ceil(me)
print("Max Error:", me)

mse = mean_squared_error(y_test, predictions)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)

score = mean_absolute_error(predictions, y_test)
print("Mean Absolute Error::", score)

threshold = score

accurate_predictions = sum(abs(predictions - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Max Error: 3325
Root Mean Squared Error: 832
Mean Absolute Error:: 607.8481405088866
Accuracy within threshold (607.8481405088866 away): 0.65
```

As expected the average error is around 900 and then the max error is around 3200. This is hisher than the previouse result that I have gotten for the models that I used.

Model

In [ ]:
```
from sklearn.svm import SVR

# Load the data
data = merged_tables.copy()

# Separate features and target variable
X = data.drop(['Serial Number', 'TSN Component (Off)(hours)', "Type"], axis=1)
y = data['TSN Component (Off)(hours)']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

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

# Initialize and train SVR model
svr = SVR(kernel='linear')
svr.fit(X_train_scaled, y_train)

# Predictions
y_pred_train = svr.predict(X_train_scaled)
y_pred_test = svr.predict(X_test_scaled)

# Evaluate the model
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

```
Train RMSE: 858.1295355058705
Test RMSE: 1112.8260001152842
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

Evaluation

In [ ]:
```
prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = y_pred_test
prediction_overview["error"] = abs(prediction_overview["truth"] - prediction_ove
prediction_overview
```

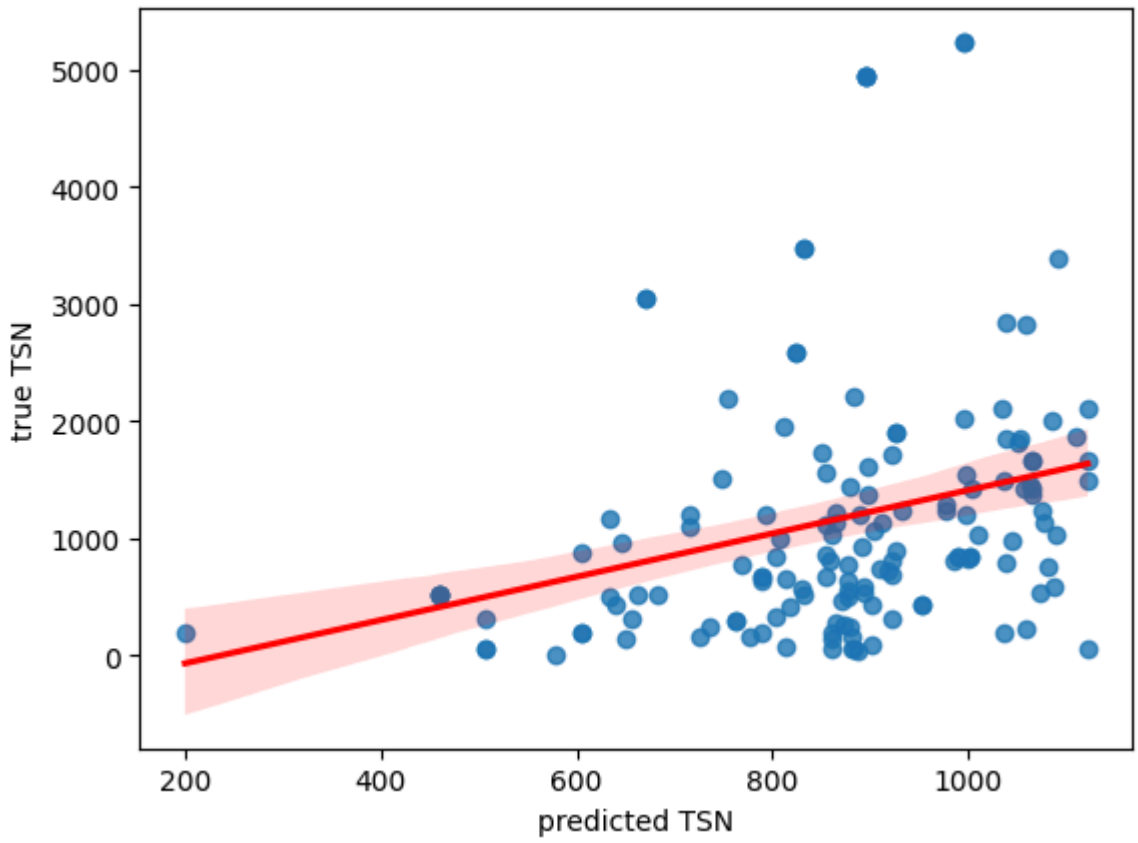
Out[ ]:

	truth	prediction	error
555	34.60	887.857133	853
353	1496.50	1122.681833	373
587	430.00	640.181617	210
401	406.58	818.526715	411
318	1811.20	1050.301717	760
...	...	...	...
179	671.60	854.251105	182
490	152.80	880.926000	728
268	1234.50	931.898881	302
426	138.30	649.646142	511
553	2189.40	753.909468	1435

147 rows × 3 columns

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=y_pred_test.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]:
```
<Axes: xlabel='predicted TSN', ylabel='true TSN'>
```



In [ ]:
```
me = max_error(y_test, y_pred_test)
me = math.ceil(me)
```

```
print("Max Error:", me)

mse = mean_squared_error(y_test, y_pred_test)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)

score = mean_absolute_error(y_pred_test, y_test)
print("Mean Absolute Error::", score)

threshold = score

accurate_predictions = sum(abs(y_pred_test - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Max Error: 4247
Root Mean Squared Error: 1113
Mean Absolute Error:: 691.9464660052487
Accuracy within threshold (691.9464660052487 away): 0.70
```

Model

In [ ]:

```
# Load the data
data = merged_tables.copy()

# Separate features and target variable
X = data.drop(['Serial Number', 'TSN Component (Off)(hours)', 'Type'], axis=1)
y = data['TSN Component (Off)(hours)']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Initialize and train Random Forest Regressor model
rf_regressor = RandomForestRegressor(random_state=42)
rf_regressor.fit(X_train, y_train)

# Predictions
y_pred_train = rf_regressor.predict(X_train)
y_pred_test = rf_regressor.predict(X_test)

# Evaluate the model
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

```
Train RMSE: 279.04451356131733
Test RMSE: 491.64613794291824
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

Evaluation

In [ ]:
```
prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = y_pred_test
prediction_overview["error"] = abs(prediction_overview["truth"] - prediction_ove
prediction_overview
```

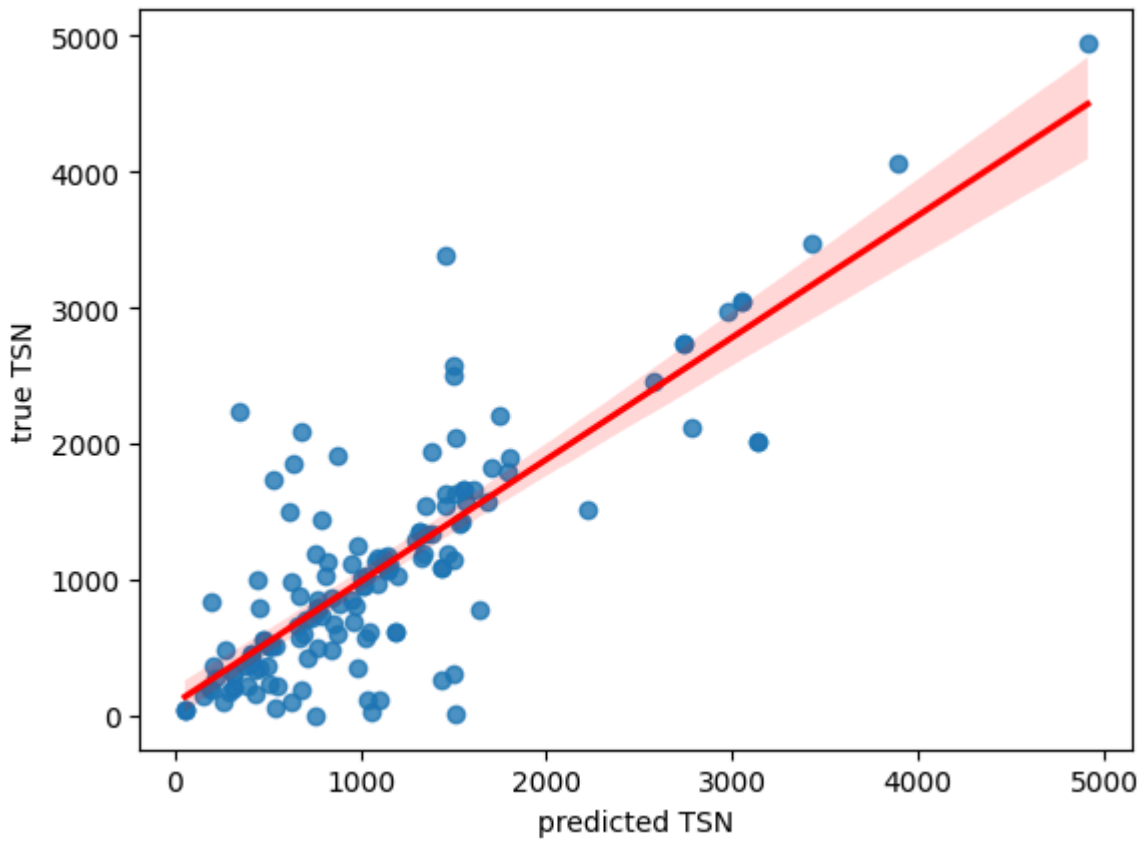
Out[ ]:

	truth	prediction	error
256	783.80	1637.426200	853
330	1919.38	875.683733	1043
14	3045.10	3045.100000	0
448	197.90	313.608000	115
171	405.00	410.715600	5
...	...	...	...
436	1439.10	793.231800	645
365	511.30	511.300000	0
412	769.00	769.000000	0
247	1337.40	1381.468967	44
260	615.35	1049.175600	433

147 rows × 3 columns

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=y_pred_test.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]:
```
<Axes: xlabel='predicted TSN', ylabel='true TSN'>
```



In [ ]:
```
me = max_error(y_test, y_pred_test)
me = math.ceil(me)
print("Max Error:", me)

mse = mean_squared_error(y_test, y_pred_test)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)

score = mean_absolute_error(y_pred_test, y_test)
print("Mean Absolute Error::", score)

threshold = score

accurate_predictions = sum(abs(y_pred_test - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Max Error: 1932
Root Mean Squared Error: 492
Mean Absolute Error:: 293.38769298351
Accuracy within threshold (293.38769298351 away): 0.70
```

Model

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Load the data
data = merged_tables.copy()

# Separate features and target variable
X = data.drop(['Serial Number', 'TSN Component (Off)(hours)', 'Type'], axis=1)
```

```
y = data['TSN Component (Off)(hours)']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Initialize and train Gradient Boosting Regressor model
gb_regressor = GradientBoostingRegressor(random_state=42)
gb_regressor.fit(X_train, y_train)

# Predictions
y_pred_train = gb_regressor.predict(X_train)
y_pred_test = gb_regressor.predict(X_test)

# Evaluate the model
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

```
Train RMSE: 440.38615220708857
Test RMSE: 722.0218609342793
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

Evaluation

In [ ]:

```
prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = y_pred_test
prediction_overview["error"] = abs(prediction_overview["truth"] - prediction_ove
prediction_overview
```

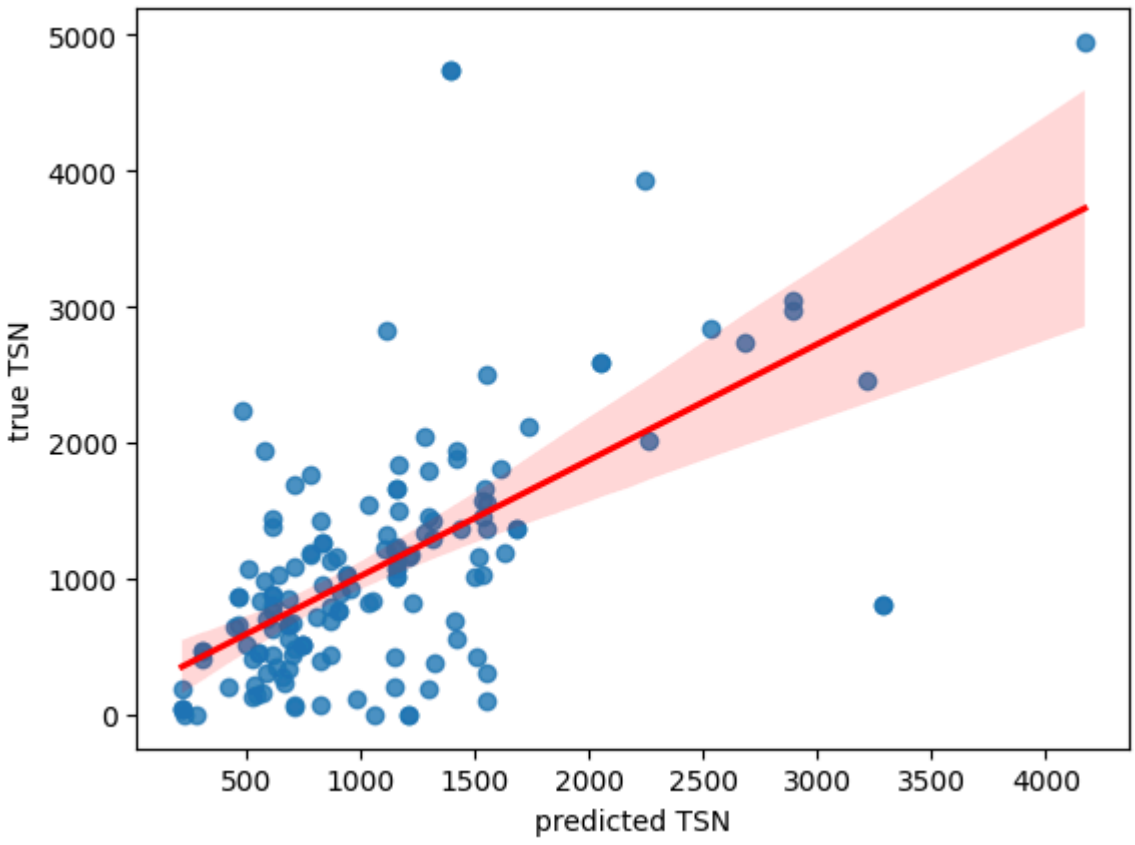
Out[ ]:

	truth	prediction	error
692	101.10	1549.537380	1448
666	1318.19	1114.912280	203
352	1889.60	1418.565342	471
396	46.10	225.062475	178
247	1337.40	1277.058258	60
...	...	...	...
83	4746.60	1395.202961	3351
276	2821.87	1116.293386	1705
478	991.60	578.958429	412
240	1666.10	1541.777233	124
575	459.90	557.008992	97

147 rows × 3 columns

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=y_pred_test.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]:
```
<Axes: xlabel='predicted TSN', ylabel='true TSN'>
```



In [ ]:
```
me = max_error(y_test, y_pred_test)
me = math.ceil(me)
```

```
print("Max Error:", me)

mse = mean_squared_error(y_test, y_pred_test)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)

score = mean_absolute_error(y_pred_test, y_test)
print("Mean Absolute Error::", score)

threshold = score

accurate_predictions = sum(abs(y_pred_test - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Max Error: 3352
Root Mean Squared Error: 723
Mean Absolute Error:: 462.24751911532263
Accuracy within threshold (462.24751911532263 away): 0.69
```

Model

In [ ]:

```
from sklearn.neural_network import MLPRegressor

# Load the data
data = merged_tables.copy()

# Separate features and target variable
X = data.drop(['Serial Number', 'TSN Component (Off)(hours)', 'Type'], axis=1)
y = data['TSN Component (Off)(hours)']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Initialize and train MLP Regressor model
mlp_regressor = MLPRegressor(random_state=42)
mlp_regressor.fit(X_train, y_train)

# Predictions
y_pred_train = mlp_regressor.predict(X_train)
y_pred_test = mlp_regressor.predict(X_test)

# Evaluate the model
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

```
Train RMSE: 1117.4692732443489
Test RMSE: 1153.8943140793303
```

```
c:\Python311\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:6
91: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached an
d the optimization hasn't converged yet.
  warnings.warn(
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

Evaluation

In [ ]:
```
prediction_overview = pd.DataFrame()
prediction_overview["truth"] = y_test
prediction_overview["prediction"] = y_pred_test
prediction_overview["error"] = abs(prediction_overview["truth"] - prediction_ove
prediction_overview
```

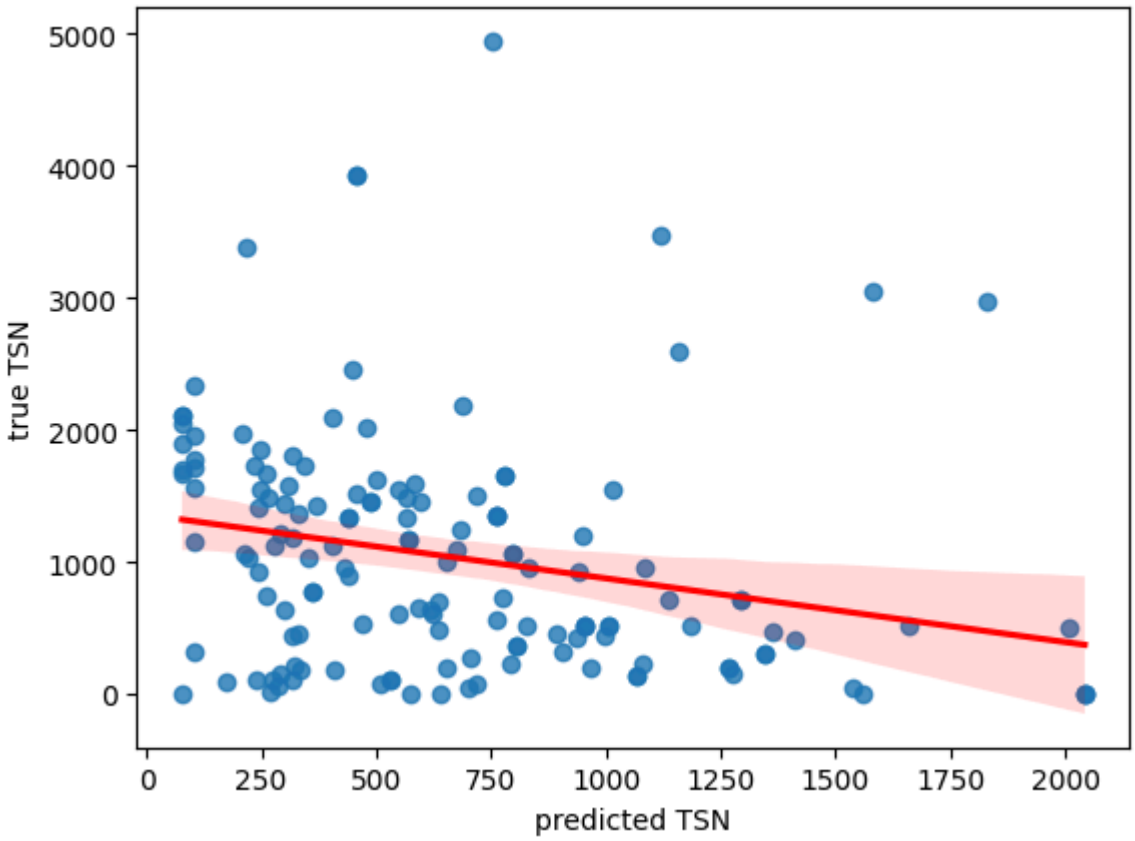
Out[ ]:

	truth	prediction	error
604	1627.55	500.033603	1127
33	517.06	951.560098	434
300	1430.60	369.479812	1061
457	365.21	803.855783	438
634	1723.40	342.914509	1380
...	...	...	...
70	3924.10	456.854479	3467
192	928.20	244.130219	684
328	103.15	316.113154	212
165	1417.55	244.628931	1172
135	1480.80	564.828880	915

147 rows × 3 columns

In [ ]:
```
plot = sns.regplot(y=y_test.values.flatten(), x=y_pred_test.flatten(), line_kws=
plot.set_xlabel("predicted TSN")
plot.set_ylabel("true TSN")
plot
```

Out[ ]:
```
<Axes: xlabel='predicted TSN', ylabel='true TSN'>
```



In [ ]:
```
me = max_error(y_test, y_pred_test)
me = math.ceil(me)
print("Max Error:", me)

mse = mean_squared_error(y_test, y_pred_test)
rmse = math.sqrt(mse)
rmse = math.ceil(rmse)
print("Root Mean Squared Error:", rmse)

score = mean_absolute_error(y_pred_test, y_test)
print("Mean Absolute Error::", score)

threshold = score

accurate_predictions = sum(abs(y_pred_test - y_test) <= threshold)
total_predictions = len(y_test)
accuracy_within_threshold = accurate_predictions / total_predictions
print("Accuracy within threshold ({} away): {:.2f}".format(threshold, accuracy_w
```

```
Max Error: 4196
Root Mean Squared Error: 1154
Mean Absolute Error:: 899.3103041363576
Accuracy within threshold (899.3103041363576 away): 0.59
```

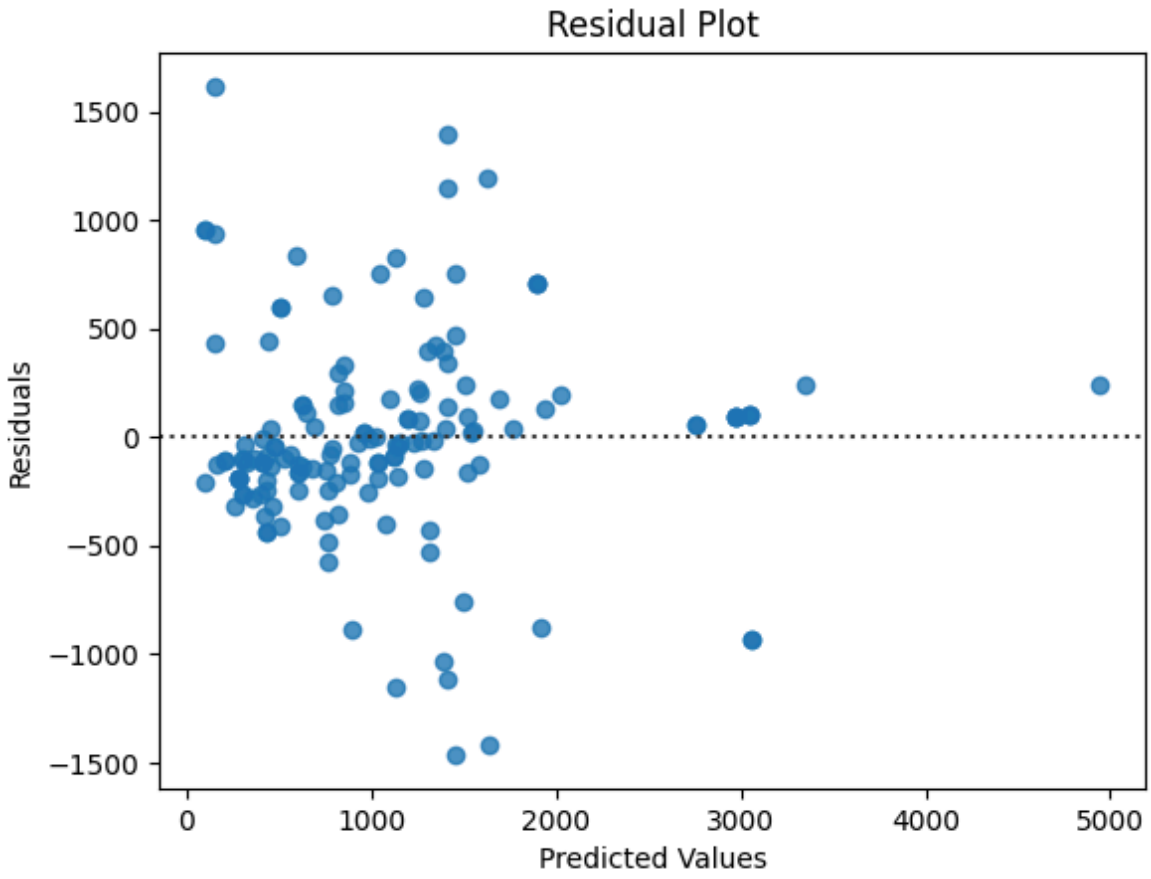
Conclusion

out of all the models the random forest regressor does the best with the highest accuracy of around 0.7. This this us the best model out of all the models that I have used. Furthermore the RMSE is the lowest compared to all the other models which also helps with making the prediction a lot better. It also has the lowest max error out of all the tests.

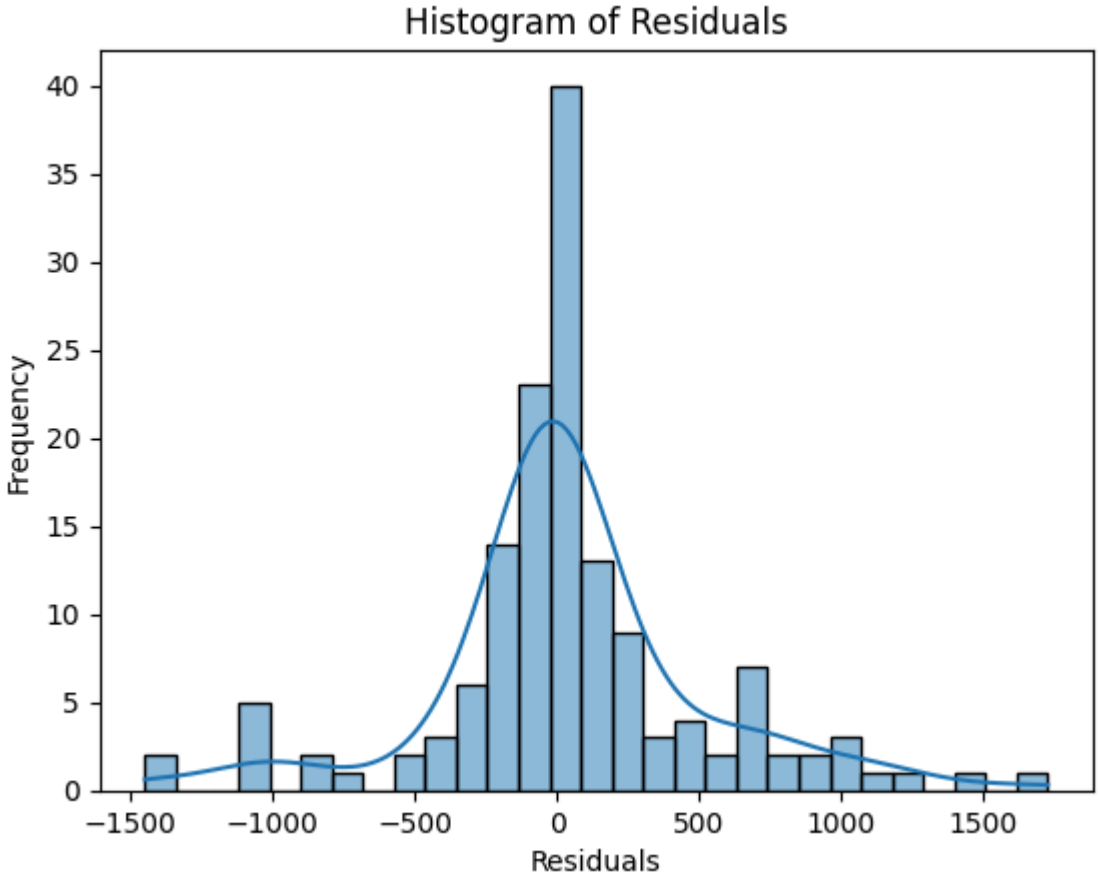
In [ ]:
```
data = merged_tables.copy()

# Separate features and target variable
X = data.drop(['Serial Number', 'TSN Component (Off)(hours)', 'Type'], axis=1)
y = data['TSN Component (Off)(hours)']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Initialize and train Random Forest Regressor model
rf_regressor = RandomForestRegressor(random_state=42)
rf_regressor.fit(X_train, y_train)

# Predictions
y_pred_train = rf_regressor.predict(X_train)
y_pred_test = rf_regressor.predict(X_test)

# Evaluate the model
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)

print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

```
Train RMSE: 267.1875254142382
Test RMSE: 479.4732424187631
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
c:\Python311\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning:
'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate t
he root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

In [ ]:
```
residuals = y_test - y_pred_test

sns.residplot(x = y_pred_test, y = residuals, line_kws=dict(color="r"))
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



In [ ]:
```
sns.histplot(residuals, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()
```


Histogram of Residuals
Frequency
Residuals
0
5
10
15
20
25
30
35
40
−1500
−1000
−500
0
500
1000
1500