Predictive Maintenance using Machine Learning Models

Import various Libararies

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, BaggingRegressor, StackingRegressor, Voti
from sklearn.svm import SVR
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
```

```
# Load dataset
df = pd.read_csv("predictive_maintenance.csv")
df
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target
0	1	M14860	M	298.1	308.6	1551	42.8	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0
...	...	...	...	...	...	...	...	...	...
9995	9996	M24855	M	298.8	308.4	1604	29.5	14	0
9996	9997	H39410	H	298.9	308.4	1632	31.8	17	0
9997	9998	M24857	M	299.0	308.6	1645	33.4	22	0
9998	9999	H39412	H	299.0	308.7	1408	48.5	25	0
9999	10000	M24859	M	299.0	308.7	1500	40.2	30	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

TO see the data properties and its distribution, we use the describe function

```
def describe_data(df):
  """
  Provides a comprehensive overview of the dataset, including:
    - Shape (rows, columns)
    - Data types of each column
    - Descriptive statistics (count, mean, std, min, max, etc.)
    - Missing value counts
    - Unique values for categorical features
  """
  print("Shape:", df.shape)
  print("\nData Types:\n", df.dtypes)
  print("\nDescriptive Statistics:\n", df.describe(include='all'))
  print("\nMissing Values:\n", df.isnull().sum())

  for col in df.columns:
    if df[col].dtype == 'object':
      print(f"\nUnique values for {col}:\n{df[col].unique()}")
```

```
describe_data(df)
```

```
25%      2500.75000        NaN    NaN           298.300000
50%      5000.50000        NaN    NaN           300.100000
75%      7500.25000        NaN    NaN           301.500000
max     10000.00000        NaN    NaN           304.500000

        Process temperature [K]  Rotational speed [rpm]   Torque [Nm]  \
count              10000.000000            10000.000000  10000.000000
unique                      NaN                     NaN           NaN
top                         NaN                     NaN           NaN
freq                        NaN                     NaN           NaN
mean                 310.005560             1538.776100     39.986910
std                    1.483734              179.284096      9.968934
min                  305.700000             1168.000000      3.800000
25%                  308.800000             1423.000000     33.200000
50%                  310.100000             1503.000000     40.100000
75%                  311.100000             1612.000000     46.800000
max                  313.800000             2886.000000     76.600000

        Tool wear [min]        Target
count      10000.000000  10000.000000
unique              NaN           NaN
top                 NaN           NaN
freq                NaN           NaN
mean         107.951000      0.033900
std           63.654147      0.180981
min            0.000000      0.000000
25%           53.000000      0.000000
50%          108.000000      0.000000
75%          162.000000      0.000000
max          253.000000      1.000000

Missing Values:
 UDI                       0
Product ID                0
Type                      0
Air temperature [K]       0
Process temperature [K]   0
Rotational speed [rpm]    0
Torque [Nm]               0
Tool wear [min]           0
Target                    0
dtype: int64

Unique values for Product ID:
['M14860' 'L47181' 'L47182' ... 'M24857' 'H39412' 'M24859']

Unique values for Type:
['M' 'L' 'H']
```

To identify the Null values, isnull is applied

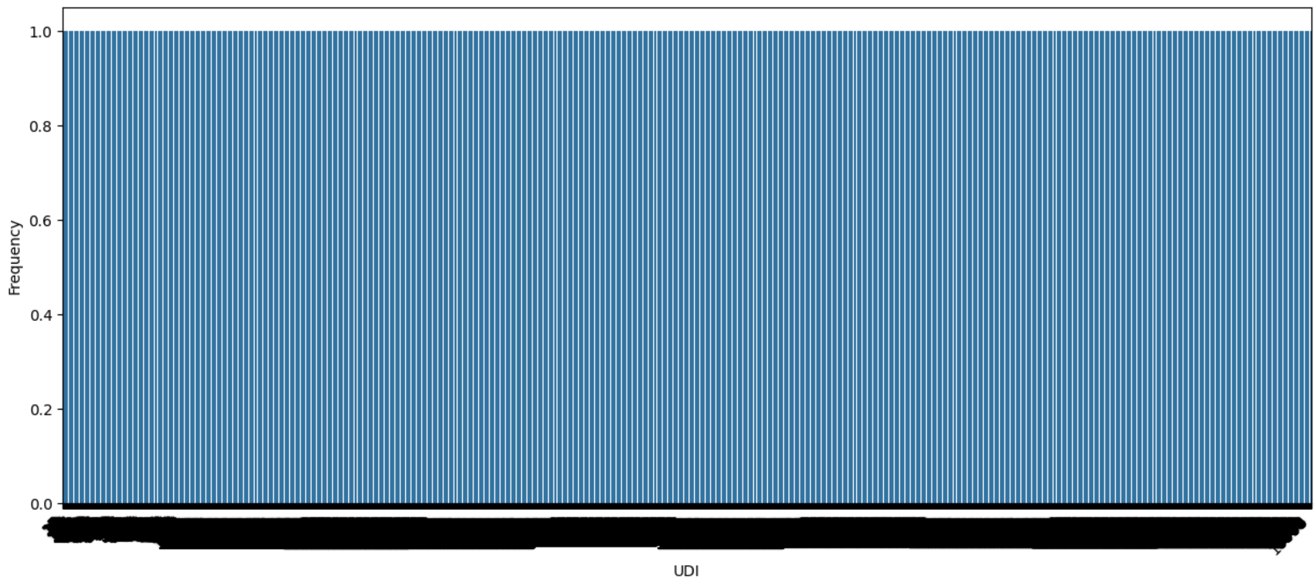
```
print(df.isnull().sum())
```

```
UDI                       0
Product ID                0
Type                      0
Air temperature [K]       0
Process temperature [K]   0
Rotational speed [rpm]    0
Torque [Nm]               0
Tool wear [min]           0
Target                    0
dtype: int64
```

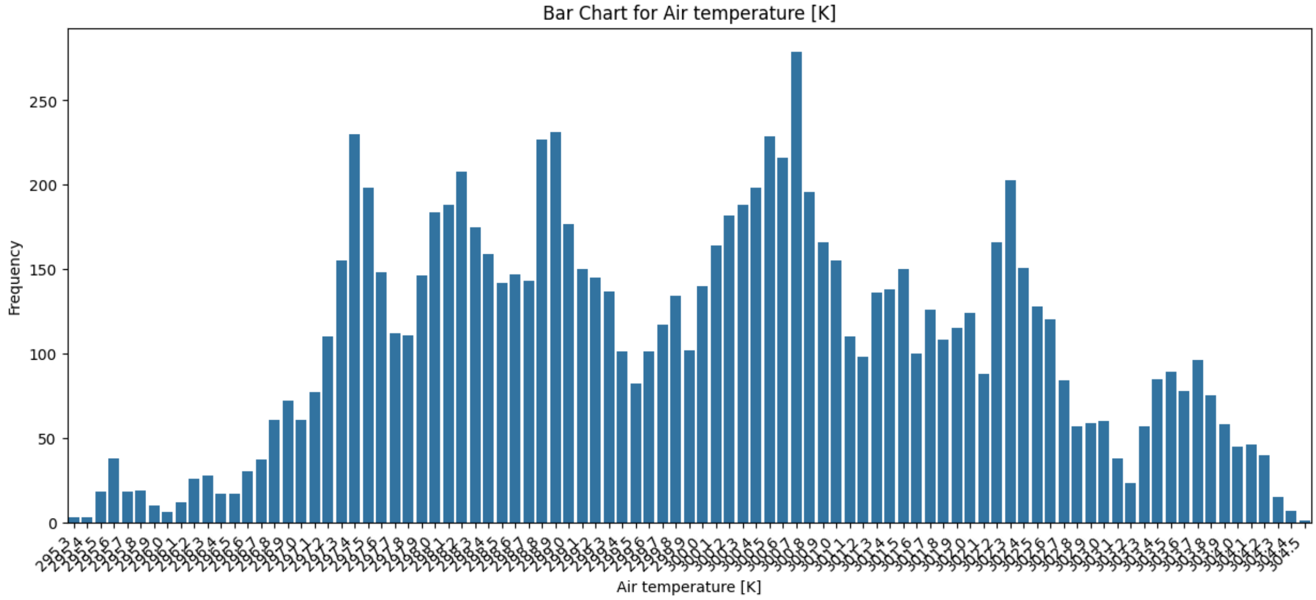
Double-click (or enter) to edit

```
#To check the data distribution we plotted bar chart against each variable

import matplotlib.pyplot as plt
import seaborn as sns

for column in df.columns:
  if df[column].dtype in ['int64', 'float64']:  # Check if the column is numeric
    plt.figure(figsize=(15, 6))
    sns.barplot(x=df[column].value_counts().index, y=df[column].value_counts().values)
    plt.title(f'Bar Chart for {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.xticks(rotation=45, ha='right')
    plt.show()
```

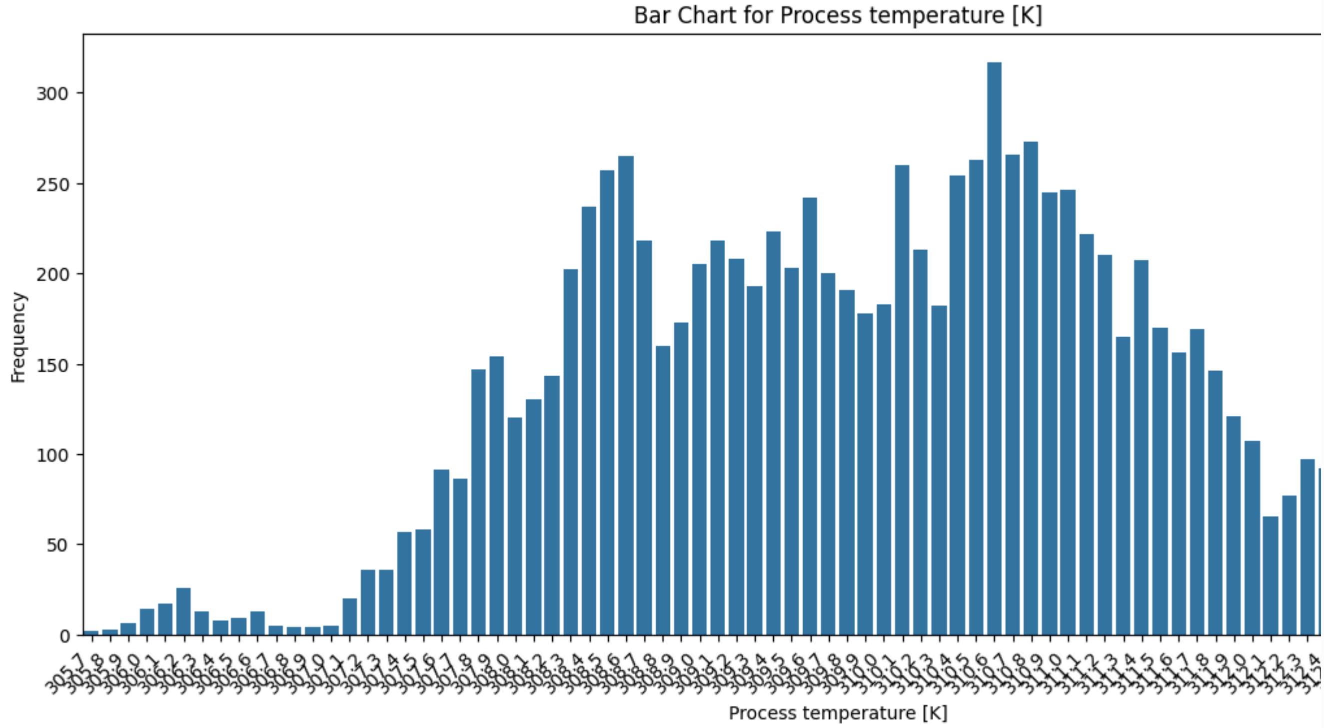
Bar Chart for UDI



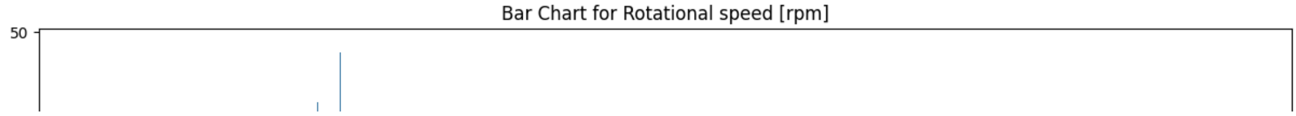
Bar Chart for Air temperature [K]

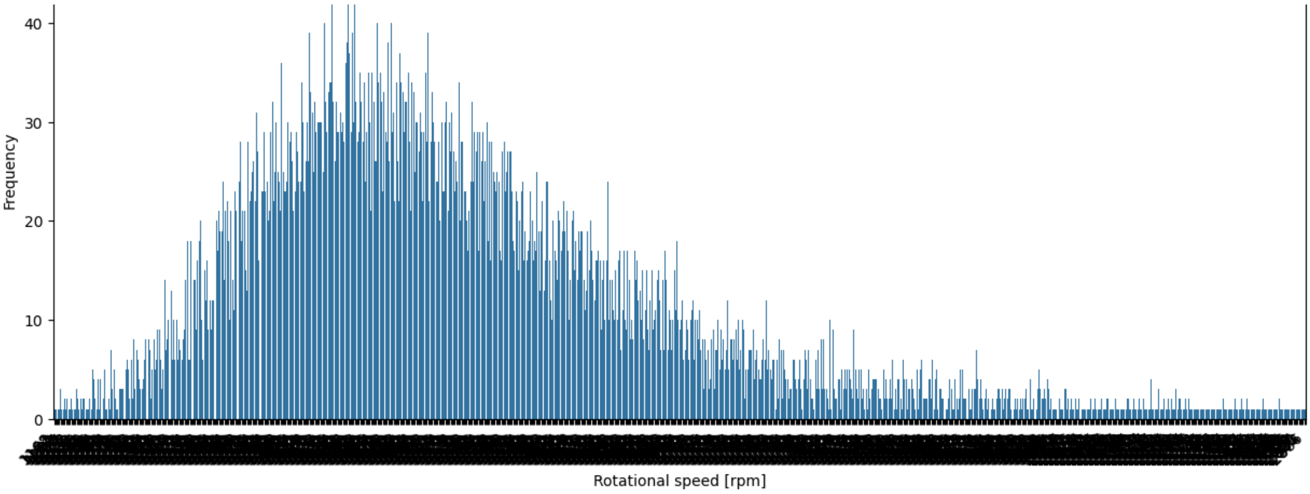


Bar Chart for Process temperature [K]

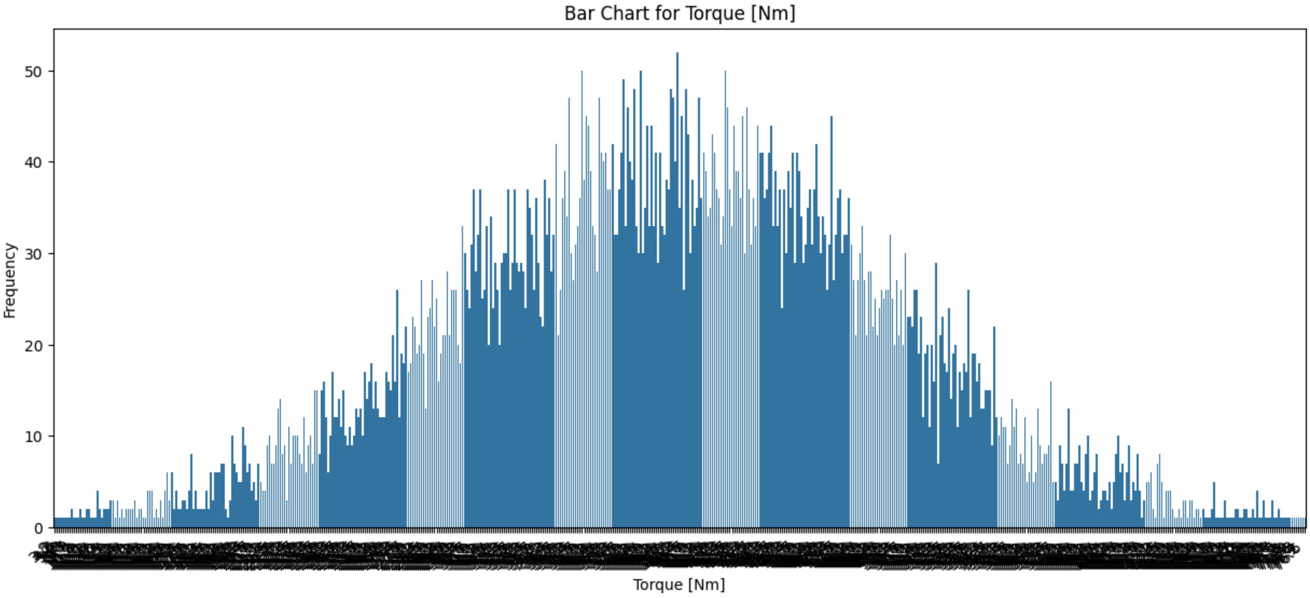


Bar Chart for Rotational speed [rpm]

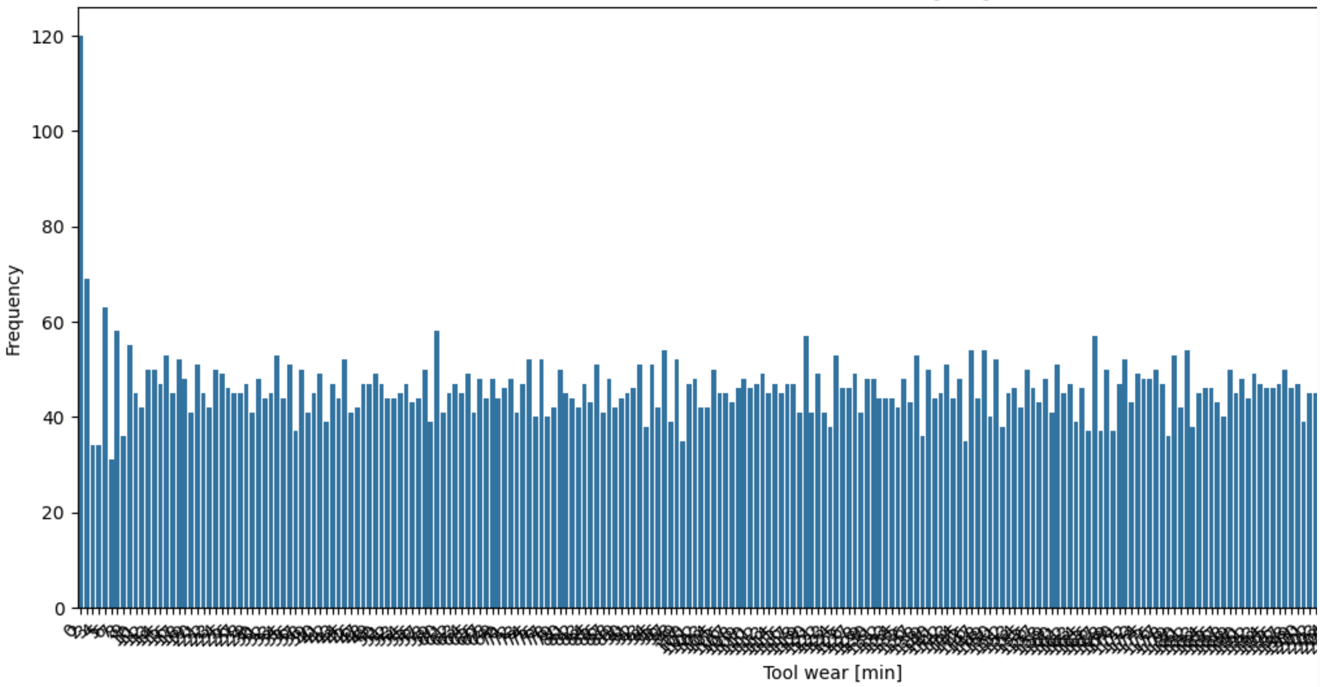




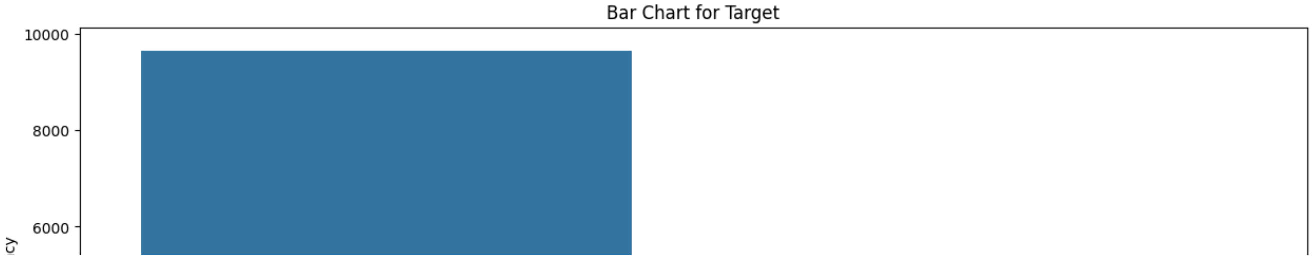
Bar Chart for Torque [Nm]

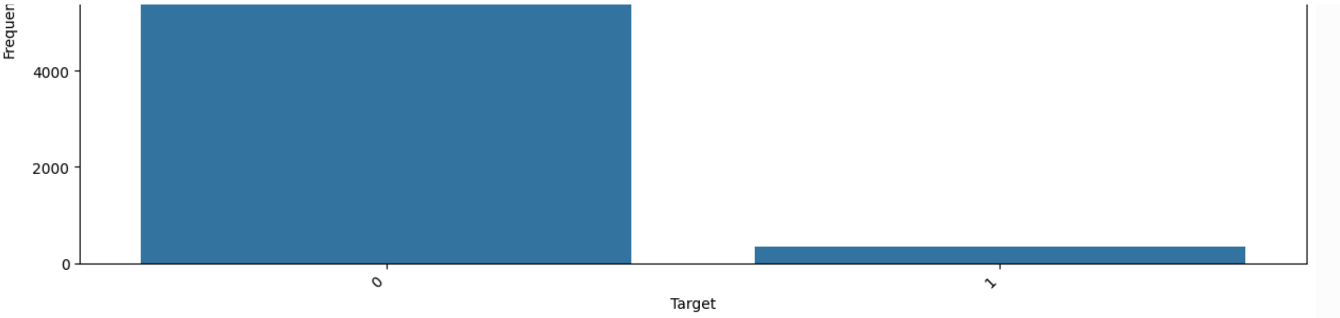


Bar Chart for Tool wear [min]



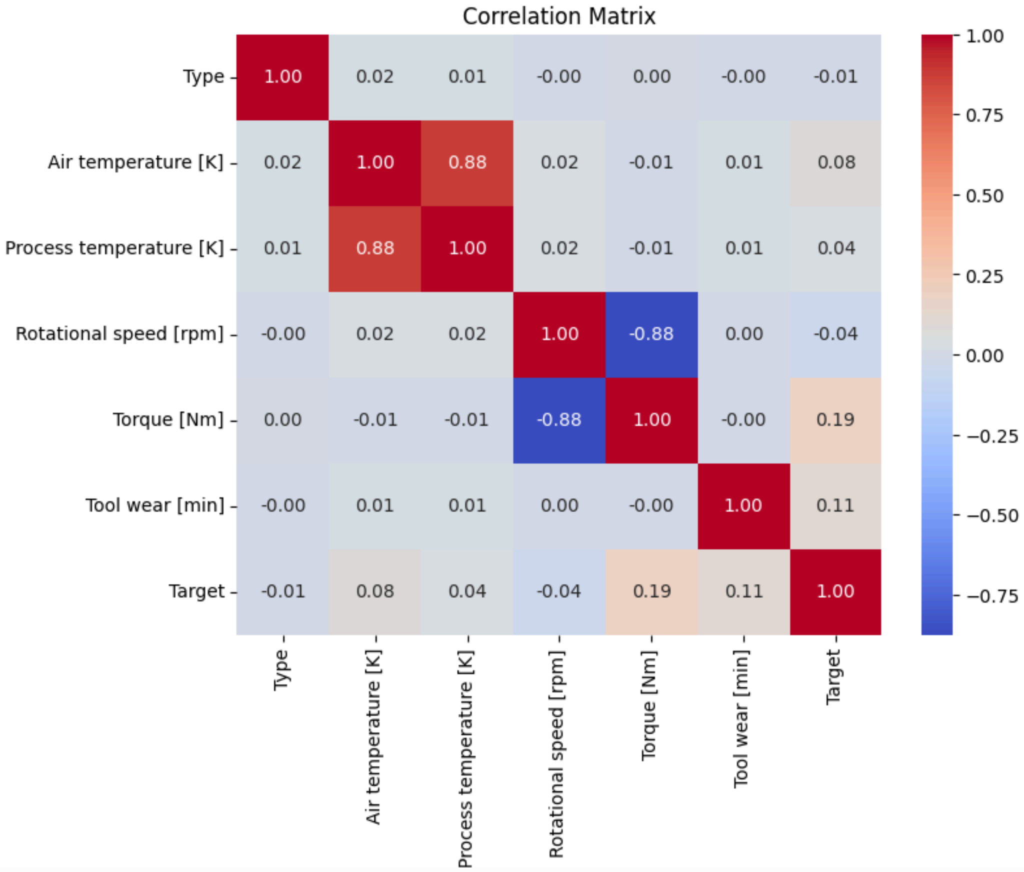
Bar Chart for Target





Frequen
4000
2000
0
0
1
Target

```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Plot the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```




```
# Since UDI is of no use in the analysis so Drop irrelevant columns
df = df.drop(columns=["UDI", "Product ID"])
df
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target
0	M	298.1	308.6	1551	42.8	0	0
1	L	298.2	308.7	1408	46.3	3	0
2	L	298.1	308.5	1498	49.4	5	0
3	L	298.2	308.6	1433	39.5	7	0
4	L	298.2	308.7	1408	40.0	9	0
...	...	...	...	...	...	...	...
9995	M	298.8	308.4	1604	29.5	14	0
9996	H	298.9	308.4	1632	31.8	17	0
9997	M	299.0	308.6	1645	33.4	22	0
9998	H	299.0	308.7	1408	48.5	25	0
9999	M	299.0	308.7	1500	40.2	30	0

10000 rows × 7 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Encode categorical variable 'Type'
label_encoder = LabelEncoder()
df["Type"] = label_encoder.fit_transform(df["Type"])
```

```
# Defining features and target
X = df.drop(columns=["Target"])
y = df["Target"]
```

```
# Split dataset into Training and Testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
4058	2	302.0	310.9	1456	47.2	54
1221	2	297.0	308.3	1399	46.4	132
6895	2	301.0	311.6	1357	45.6	137
9863	1	298.9	309.8	1411	56.3	84
8711	1	297.1	308.5	1733	28.7	50
...	...	...	...	...	...	...
980	1	296.1	306.7	1409	42.8	134
4266	1	302.7	311.1	1440	39.5	146
7772	0	300.3	311.5	1464	41.0	29
5780	1	301.7	311.2	1517	42.4	113
1424	1	298.7	309.7	1462	46.8	4

8000 rows × 6 columns

Next steps: Generate code with `X_train` View recommended plots New interactive sheet

```
X_test
```

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
2997	1	300.5	309.8	1345	62.7	153
4871	1	303.7	312.4	1513	40.1	135
3858	1	302.5	311.4	1559	37.6	209
951	0	295.6	306.3	1509	35.8	60
6463	0	300.5	310.0	1358	60.4	102
...	...	...	...	...	...	...
1686	0	297.9	307.3	1663	28.7	7
6952	1	300.8	311.3	1498	40.2	73
9954	2	298.1	307.9	1446	42.8	121
5728	2	302.4	311.9	1422	46.4	194
9191	2	297.9	309.0	1970	21.6	37

2000 rows × 6 columns

Next steps: Generate code with `X_test` View recommended plots New interactive sheet

```
# Normalize numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier, BaggingClassifier, S
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Define models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Ada Boost": AdaBoostClassifier(random_state=42),
    "Gradient Boost": GradientBoostingClassifier(random_state=42),
    "LGBMR": LGBMClassifier(random_state=42),
    "XGBR": XGBClassifier(random_state=42),
    "SVM": SVC(probability=True),
    "Voting Classifier": VotingClassifier(
        estimators=[("rf", RandomForestClassifier()), ("gb", GradientBoostingClassifier()), ("xgb", XGBClassifier())], voting='soft'
    ),
```

```
    "Bagging Classifier": BaggingClassifier(random_state=42),
    "Stacking Classifier": StackingClassifier(
        estimators=[("rf", RandomForestClassifier()), ("gb", GradientBoostingClassifier()), ("xgb", XGBClassifier())]
    )
}


# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier, BaggingClassifier, S
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
#Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier # Import the KNeighborsClassifier class

# Define models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "KNN": KNeighborsClassifier(),
    "AdaBoost": AdaBoostClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "XGBoost": XGBClassifier(),
    "LGBM": LGBMClassifier()
  }


# Train and evaluate models
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

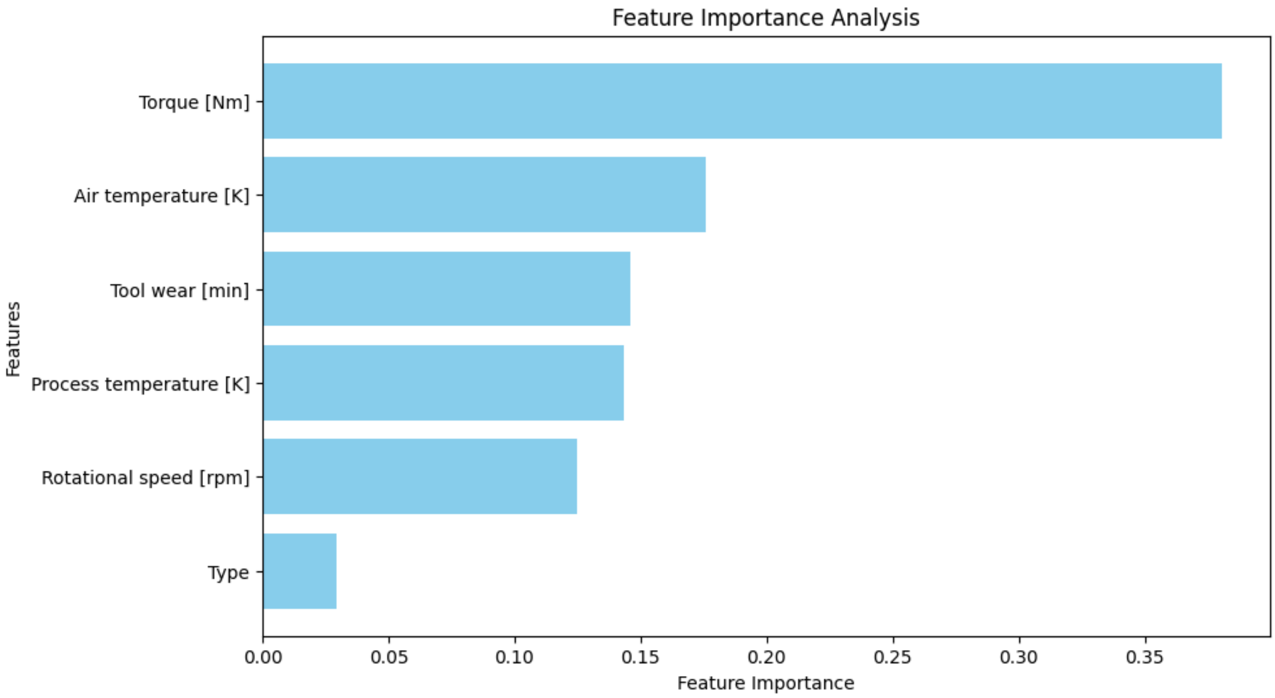
```
[LightGBM] [Info] Number of positive: 271, number of negative: 7729
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001441 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 928
[LightGBM] [Info] Number of data points in the train set: 8000, number of used features: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.033875 -> initscore=-3.350616
[LightGBM] [Info] Start training from score -3.350616
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_
  warnings.warn(
```

```
# Train and evaluate models
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score # Import accuracy_score

results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-score": f1_score(y_test, y_pred)
    })
```

```
[LightGBM] [Info] Number of positive: 271, number of negative: 7729
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000689 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 928
[LightGBM] [Info] Number of data points in the train set: 8000, number of used features: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.033875 -> initscore=-3.350616
[LightGBM] [Info] Start training from score -3.350616
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_
  warnings.warn(
```

```
# Train Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Feature importance analysis
feature_importances = model.feature_importances_
```

```
feature_names = X.columns

# Create DataFrame for better visualization
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10,6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance Analysis')
plt.gca().invert_yaxis()
plt.show()

# Print feature importance values
print(importance_df)
```




```
                  Feature  Importance
4             Torque [Nm]    0.380621
1     Air temperature [K]    0.175616
5         Tool wear [min]    0.146162
2 Process temperature [K]    0.143359
3  Rotational speed [rpm]    0.124845
0                    Type    0.029396
```

```
# Eevaluate the performance matrix of each of the models specified above

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix

# Train and evaluate models
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    try:
        y_pred_proba = model.predict_proba(X_test)[:, 1]
        roc_auc = roc_auc_score(y_test, y_pred_proba)
    except AttributeError:
        roc_auc = None  # Some models don't support predict_proba

    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred, average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted'),
        "F1-score": f1_score(y_test, y_pred, average='weighted'),
        "Confusion Matrix": confusion_matrix(y_test, y_pred)
    })


# Create a DataFrame for the results
results_df = pd.DataFrame(results)

# Display the results
```

```
results_df
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_all_
  warnings.warn(
[LightGBM] [Info] Number of positive: 271, number of negative: 7729
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000533 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 928
[LightGBM] [Info] Number of data points in the train set: 8000, number of used features: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.033875 -> initscore=-3.350616
[LightGBM] [Info] Start training from score -3.350616
```

	Model	Accuracy	Precision	Recall	F1-score	Confusion Matrix
0	Logistic Regression	0.9680	0.959512	0.9680	0.957029	[[1928, 4], [60, 8]]
1	Decision Tree	0.9805	0.980930	0.9805	0.980703	[[1911, 21], [18, 50]]
2	Random Forest	0.9835	0.982269	0.9835	0.982201	[[1925, 7], [26, 42]]
3	SVM	0.9720	0.969458	0.9720	0.963519	[[1930, 2], [54, 14]]
4	KNN	0.9740	0.970868	0.9740	0.967929	[[1928, 4], [48, 20]]
5	AdaBoost	0.9710	0.965130	0.9710	0.965519	[[1922, 10], [48, 20]]
6	Gradient Boosting	0.9845	0.983457	0.9845	0.983569	[[1924, 8], [23, 45]]
7	XGBoost	0.9875	0.986854	0.9875	0.986860	[[1926, 6], [19, 49]]
8	LGBM	0.9885	0.988007	0.9885	0.987809	[[1928, 4], [19, 49]]

Next steps: Generate code with results_df   View recommended plots   New interactive sheet

```
# Convert results to DataFrame
results_df = pd.DataFrame(results).sort_values(by="Accuracy", ascending=False)

# Display results
print(results_df)
```

```
                 Model  Accuracy  Precision  Recall  F1-score  \
8                 LGBM    0.9885   0.988007  0.9885  0.987809
7              XGBoost    0.9875   0.986854  0.9875  0.986860
6    Gradient Boosting    0.9845   0.983457  0.9845  0.983569
2        Random Forest    0.9835   0.982269  0.9835  0.982201
1        Decision Tree    0.9805   0.980930  0.9805  0.980703
4                  KNN    0.9740   0.970868  0.9740  0.967929
3                  SVM    0.9720   0.969458  0.9720  0.963519
5             AdaBoost    0.9710   0.965130  0.9710  0.965519
0  Logistic Regression    0.9680   0.959512  0.9680  0.957029

         Confusion Matrix
8   [[1928, 4], [19, 49]]
7   [[1926, 6], [19, 49]]
6   [[1924, 8], [23, 45]]
2   [[1925, 7], [26, 42]]
1  [[1911, 21], [18, 50]]
4   [[1928, 4], [48, 20]]
3   [[1930, 2], [54, 14]]
5  [[1922, 10], [48, 20]]
0    [[1928, 4], [60, 8]]
```

```
# Selection of the best model among the models specified above

# Find the best model based on accuracy
best_model_name = results_df.loc[results_df['Accuracy'].idxmax()]['Model']
best_model_accuracy = results_df.loc[results_df['Accuracy'].idxmax()]['Accuracy']

print(f"The best model is '{best_model_name}' with an accuracy of {best_model_accuracy:.4f}")

# You can also select the best model based on other metrics like F1-score, precision, recall, etc.
# For example, to select the best model based on F1-score:
# best_model_name = results_df.loc[results_df['F1-score'].idxmax()]['Model']
# best_model_f1_score = results_df.loc[results_df['F1-score'].idxmax()]['F1-score']
# print(f"The best model based on F1-score is '{best_model_name}' with an F1-score of {best_model_f1_score:.4f}")
```

```
The best model is 'LGBM' with an accuracy of 0.9885
```

Use Hyperparameter tuning.

Hyperparameter tuning on all the models

```
from sklearn.model_selection import GridSearchCV

# Define parameter grids for hyperparameter tuning for different models
param_grids = {
    "Logistic Regression": {
        'C': [0.1, 1, 10],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    },
    "Decision Tree": {
        'max_depth': [None, 5, 10],
        'min_samples_split': [2, 5, 10],
        'criterion': ['gini', 'entropy']
    },
    "Random Forest": {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 5, 10],
        'min_samples_split': [2, 5, 10]
    },
    "SVM": {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    },
    "KNN": {
        'n_neighbors': [3, 5, 7],
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan']
    },
    "AdaBoost": {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.1, 1, 10]
    },
    "Gradient Boosting": {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.1, 1, 10],
        'max_depth': [3, 5, 7]
    },
    "XGBoost": {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.1, 1, 10],
        'max_depth': [3, 5, 7]
    },
    "LGBM": {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.1, 1, 10],
        'max_depth': [3, 5, 7]
    }
}

tuned_results = []
for name, model in models.items():
  if name in param_grids:
      grid_search = GridSearchCV(estimator=model, param_grid=param_grids[name], scoring='accuracy', cv=5)
      grid_search.fit(X_train, y_train)
      best_model = grid_search.best_estimator_
      y_pred = best_model.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)

      tuned_results.append({
          "Model": name,
          "Best Parameters": grid_search.best_params_,
          "Accuracy": accuracy
      })
  else:
      tuned_results.append({
          "Model": name,
          "Best Parameters": "No tuning parameters defined",
          "Accuracy": "Not Tuned"
      })

# Convert results to DataFrame
tuned_results_df = pd.DataFrame(tuned_results).sort_values(by="Accuracy", ascending=False)

# Display results
tuned_results_df
```

Show hidden output

Next steps: Generate code with tuned_results_df | View recommended plots | New interactive sheet