This project is about ahmedabad plain **crash bold text**

Import the required libraries

```
# Import Numerical Python for performing numerical operations on different data structures
import numpy as np

# Import Pandas for converting different file formate into pandas dataframe
import pandas as pd

#Import seaborn and matplotlib for creating the interactive visualization

import seaborn as sns
import matplotlib.pyplot as plt
```

Load the data

```
# Load the data from local
df_airplain_crash= pd.read_csv('airplain_crash.csv')
```

```
# check the first five observation
df_airplain_crash.head()
```

	Timestamp	Flight_ID	Aircraft_Model	Electrical_System_Status	Emergency_Power_Activation	Voltage_Level	Current_Load	Engine_P
0	12-06-2025 08:43	AI114	Boeing 787-8 Dreamliner	Normal	0	244.90	579.26	
1	12-06-2025 08:19	AI111	Boeing 787-8 Dreamliner	Normal	0	246.28	514.29	
2	12-06-2025 06:50	AI191	Boeing 787-8 Dreamliner	Normal	0	229.27	653.33	
3	12-06-2025 06:40	AI189	Boeing 787-8 Dreamliner	Normal	0	206.65	769.53	
4	12-06-2025 06:24	AI145	Boeing 787-8 Dreamliner	Normal	0	164.65	443.42	

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

```
# check the last five observations
df_airplain_crash.tail()
```

	Timestamp	Flight_ID	Aircraft_Model	Electrical_System_Status	Emergency_Power_Activation	Voltage_Level	Current_Load	Engin
9267	12-06-2025 06:11	AI133	Boeing 787-8 Dreamliner	Normal	0	284.37	837.61	
9268	12-06-2025 08:14	AI150	Boeing 787-8 Dreamliner	Warning	0	195.80	564.29	
9269	12-06-2025 06:31	AI180	Boeing 787-8 Dreamliner	Normal	0	213.78	685.73	
9270	12-06-2025 07:50	AI146	Boeing 787-8 Dreamliner	Normal	0	224.33	730.17	
9271	12-06-2025 08:32	AI144	Boeing 787-8 Dreamliner	Normal	0	177.12	735.58	

Data Overview

```
# check the columns we have into the data
df_airplain_crash.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Emergency_Power_Activation', 'Voltage_Level', 'Current_Load',
       'Engine_Performance', 'Altitude', 'Airspeed', 'Flight_Phase',
       'Maintenance_History', 'Last_Maintenance_Date', 'Weather_Condition',
       'Temperature', 'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

interpratation

- 'Timestamp' : The exact date and time when the record or measurement was captured,
- 'Flight_ID' : A unique identifier for each flight, such as the airline designator and flight number,
- 'Aircraft_Model' :The make and model of the aircraft ,
- 'Electrical_System_Status' : The health and operational state of the aircraft's electrical power system,
- 'Emergency_Power_Activation' : A flag indicating if backup electrical systems ,
- 'Voltage_Level' : Measured voltage (in volts) available in the electrical system,
- 'Current_Load' : The electrical current (amperes) being drawn by aircraft systems,
- 'Engine_Performance' : Metrics such as thrust, fuel flow, RPM, EGT,,
- 'Altitude' : The aircraft's vertical position above a reference datum,
- 'Airspeed' : Aircraft speed relative to the surrounding air,
- 'Flight_Phase' : Which segment of the flight the aircraft,
- 'Maintenance_History' : Recorded service and maintenance events,
- 'Last_Maintenance_Date' : The calendar date of the most recent maintenance action performed on this specific aircraft or system.,
- 'Weather_Condition' : Environmental description at the aircraft location/time,
- 'Temperature' : Ambient air temperature around the aircraft,
- 'Air_Traffic_Control_Delay' : Whether delays have been imposed by ATC ,
- 'Ground_Staff_Activity' : Activities taking place on the ground,
- 'CCTV_Anomaly_Flag' : A binary indicator from closed-circuit video monitoring,
- 'Pilot_Communication_Score' : A metric (possibly derived from assessments like NOTECHS) evaluating the pilot's communication quality,
- 'System_Failure' : A flag or description noting if a critical system malfunction occurred.

```
# Check the shape of the data
df_airplain_crash.shape
```

```
(9272, 20)
```

Interpretation

- We have 10000 observations and 20 attributes

```
# check the basic info
df_airplain_crash.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9272 entries, 0 to 9271
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Timestamp                   9272 non-null   object
 1   Flight_ID                   9272 non-null   object
 2   Aircraft_Model              9272 non-null   object
 3   Electrical_System_Status    9272 non-null   object
 4   Emergency_Power_Activation  9272 non-null   int64
 5   Voltage_Level               9272 non-null   float64
 6   Current_Load                9272 non-null   float64
 7   Engine_Performance          9272 non-null   float64
 8   Altitude                    9272 non-null   float64
 9   Airspeed                    9272 non-null   float64
 10  Flight_Phase                9272 non-null   object
 11  Maintenance_History         9272 non-null   int64
 12  Last_Maintenance_Date       9272 non-null   object
 13  Weather_Condition           9272 non-null   object
 14  Temperature                 9272 non-null   float64
 15  Air_Traffic_Control_Delay   9272 non-null   float64
 16  Ground_Staff_Activity       9272 non-null   int64
 17  CCTV_Anomaly_Flag           9272 non-null   int64
 18  Pilot_Communication_Score   9272 non-null   float64
 19  System_Failure              9272 non-null   int64
dtypes: float64(8), int64(5), object(7)
memory usage: 1.4+ MB
```

interpretation

- We have 13 numerical columns and 7 categorical columns
- This data acquires 1.5+mb space.

```
# Check basic statistics
df_airplain_crash.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Emergency_Power_Activation	9272.0	0.000000	0.000000	0.00	0.0000	0.000	0.0000	0.00
Voltage_Level	9272.0	229.349516	29.537585	139.77	209.2750	229.570	249.4350	318.14
Current_Load	9272.0	599.058526	147.941955	159.98	496.7275	597.330	701.5925	1000.00
Engine_Performance	9272.0	94.699889	4.218522	81.71	91.7900	95.080	98.4000	100.00
Altitude	9272.0	4973.770274	2711.004930	0.00	2978.5050	4986.945	6973.0600	10000.00
Airspeed	9272.0	249.872783	28.397533	164.40	230.1400	249.910	270.7800	300.00
Maintenance_History	9272.0	4.990725	3.155879	0.00	2.0000	5.000	8.0000	10.00
Temperature	9272.0	14.796247	14.350509	-10.00	2.3000	14.800	27.0000	40.00
Air_Traffic_Control_Delay	9272.0	4.619702	4.091780	0.00	1.4400	3.440	6.6425	19.42
Ground_Staff_Activity	9272.0	9.885785	6.049521	0.00	5.0000	10.000	15.0000	20.00
CCTV_Anomaly_Flag	9272.0	0.000000	0.000000	0.00	0.0000	0.000	0.0000	0.00
Pilot_Communication_Score	9272.0	0.290439	0.162216	0.01	0.1600	0.270	0.3900	0.82
System_Failure	9272.0	0.096527	0.295329	0.00	0.0000	0.000	0.0000	1.00

Interpretation

- After observing this table, i conclude that there are 3 column that content the outliers, remaining not
- the columns are :
  - Voltage_Level
  - Current_Load
  - Air_Traffic_Control_Delay

```
import pandas as pd

# Assuming your data is in a pandas DataFrame called df

def cap_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # For Current_Load, only cap the lower bound (min)
    if column == 'Current_Load':
        df.loc[df[column] < lower_bound, column] = lower_bound
    else:
        # For others, cap both sides if needed
        df.loc[df[column] < lower_bound, column] = lower_bound
        df.loc[df[column] > upper_bound, column] = upper_bound

    return df

# Cap Voltage_Level max
df_airplain_crash = cap_outliers(df_airplain_crash, 'Voltage_Level')

# Cap Current_Load min
df_airplain_crash = cap_outliers(df_airplain_crash, 'Current_Load')

# Altitude and Temperature - no treatment needed, skip

# Cap Air_Traffic_Control_Delay max
df_airplain_crash = cap_outliers(df_airplain_crash, 'Air_Traffic_Control_Delay')

# Verify results
print(df_airplain_crash[['Voltage_Level', 'Current_Load', 'Air_Traffic_Control_Delay']].describe())
```

```
       Voltage_Level  Current_Load  Air_Traffic_Control_Delay
count    9272.000000   9272.000000                9272.000000
mean      229.353407    599.072892                   4.549174
std        29.484386    147.901064                   3.890903
min       149.035000    189.430000                   0.000000
25%       209.275000    496.727500                   1.440000
50%       229.570000    597.330000                   3.440000
75%       249.435000    701.592500                   6.642500
max       309.675000   1000.000000                  14.446250
```

```
df_airplain_crash = df_airplain_crash.drop('Emergency_Power_Activation', axis=1)
```

Data Preprocessing

Data Cleaning

```
# Check the name of the column
df_airplain_crash.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Flight_Phase', 'Maintenance_History',
       'Last_Maintenance_Date', 'Weather_Condition', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Interpretation

- After looking at the columns, I conclude that there is no need to rename the columns

```
# check the first five observation
df_airplain_crash.head()
```

	Timestamp	Flight_ID	Aircraft_Model	Electrical_System_Status	Voltage_Level	Current_Load	Engine_Performance	Altitude	Airspe
0	12-06-2025 08:43	AI114	Boeing 787-8 Dreamliner	Normal	244.90	579.26	98.24	9569.09	242.
1	12-06-2025 08:19	AI111	Boeing 787-8 Dreamliner	Normal	246.28	514.29	90.38	0.00	278.
2	12-06-2025 06:50	AI191	Boeing 787-8 Dreamliner	Normal	229.27	653.33	97.09	7497.39	241.
3	12-06-2025 06:40	AI189	Boeing 787-8 Dreamliner	Normal	206.65	769.53	96.87	3840.58	215.
4	12-06-2025 06:24	AI145	Boeing 787-8 Dreamliner	Normal	164.65	443.42	95.86	5972.60	272.

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

Interpretaation

- There are a few impurities present, which we solved in basic Excel

Null value handling

```
 # Check the count of null records
df_airplain_crash.isnull().sum()
```

	0
Timestamp	0
Flight_ID	0
Aircraft_Model	0
Electrical_System_Status	0
Voltage_Level	0
Current_Load	0
Engine_Performance	0
Altitude	0
Airspeed	0
Flight_Phase	0
Maintenance_History	0
Last_Maintenance_Date	0
Weather_Condition	0
Temperature	0
Air_Traffic_Control_Delay	0
Ground_Staff_Activity	0
CCTV_Anomaly_Flag	0
Pilot_Communication_Score	0
System_Failure	0

dtype: int64

```
# Find the percentage for null records
# Note: We have to find the percentage of null records because the loss we have for handling the null values depends on the percentage
df_airplain_crash.isnull().sum()/len(df_airplain_crash) * 100
```

	0
Timestamp	0.0
Flight_ID	0.0
Aircraft_Model	0.0
Electrical_System_Status	0.0
Voltage_Level	0.0
Current_Load	0.0
Engine_Performance	0.0
Altitude	0.0
Airspeed	0.0
Flight_Phase	0.0
Maintenance_History	0.0
Last_Maintenance_Date	0.0
Weather_Condition	0.0
Temperature	0.0
Air_Traffic_Control_Delay	0.0
Ground_Staff_Activity	0.0
CCTV_Anomaly_Flag	0.0
Pilot_Communication_Score	0.0
System_Failure	0.0

dtype: float64

```
len(df_airplain_crash)
```

```
9272
```

interpretation

- After observation, we conclude that there is no null records were present

EDA

Univariate Analysis

```
# To perform the univariate analysis, let's segregate numerical and categorical data
df_num = df_airplain_crash.select_dtypes(include = 'number')

# Check the numerical data

df_num.head()
```

	Voltage_Level	Current_Load	Engine_Performance	Altitude	Airspeed	Maintenance_History	Temperature	Air_Traffic_Control_Delay
0	244.90	579.26	98.24	9569.09	242.98	10	13.0	2.03
1	246.28	514.29	90.38	0.00	278.51	0	16.2	2.55
2	229.27	653.33	97.09	7497.39	241.20	8	38.3	8.26
3	206.65	769.53	96.87	3840.58	215.24	1	23.1	1.87
4	164.65	443.42	95.86	5972.60	272.38	8	38.1	9.31

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

```
# Separate the categorical variables from the main dataframe

df_cat = df_airplain_crash.select_dtypes(include = 'object')

# Check the categorical data

df_cat.head()
```

	Timestamp	Flight_ID	Aircraft_Model	Electrical_System_Status	Flight_Phase	Last_Maintenance_Date	Weather_Condition
0	12-06-2025 08:43	AI114	Boeing 787-8 Dreamliner	Normal	Takeoff	16-04-2025	Clear
1	12-06-2025 08:19	AI111	Boeing 787-8 Dreamliner	Normal	Takeoff	20-05-2025	Clear
2	12-06-2025 06:50	AI191	Boeing 787-8 Dreamliner	Normal	Climb	17-04-2025	Clear
	12-06-2025		Boeing 787-8				

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

Univeriate Analysis on Numerical Data

```
# Check the columns
df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Voltage_Level

```
# Find the minimum

df_num.Voltage_Level.min()
```

```
np.float64(149.03500000000003)
```

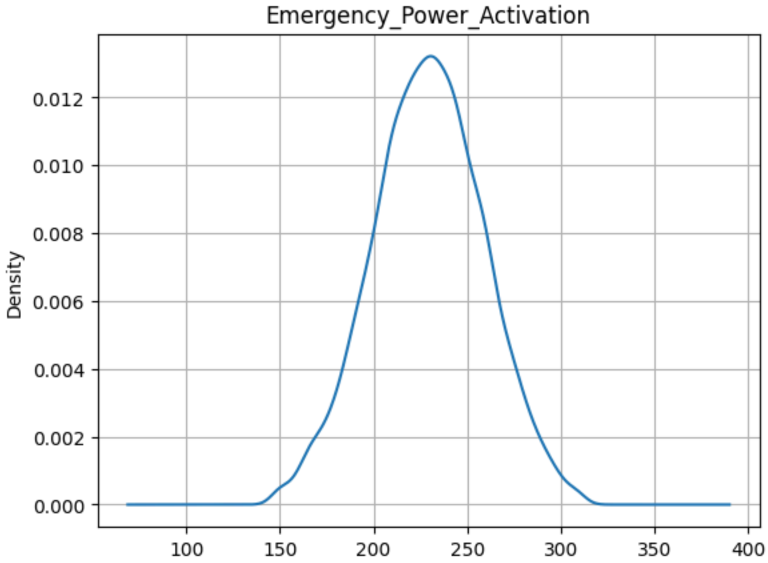
```
# Find the maximum

df_num.Voltage_Level.max()
```

```
np.float64(309.675)
```

```
# Find the average

df_num.Voltage_Level.mean()
```

```
np.float64(229.35340703192406)
```

```
# Check the visualization

df_num.Voltage_Level.plot(kind = 'kde',) # slightly normaly distributed kde = 'kernal density Estimation'
plt.title('Emergency_Power_Activation')
plt.grid()
```



Interpritation

- Minimum value is 149.03500000000003
- Maximum value is 309.675
- Average value is 229.35340703192406
- After looking at the distribution, I conclude that the graph shows Central tendency

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Current_Load

```
# Find the minimum
df_num.Current_Load.min()
```

```
np.float64(189.43000000000012)
```

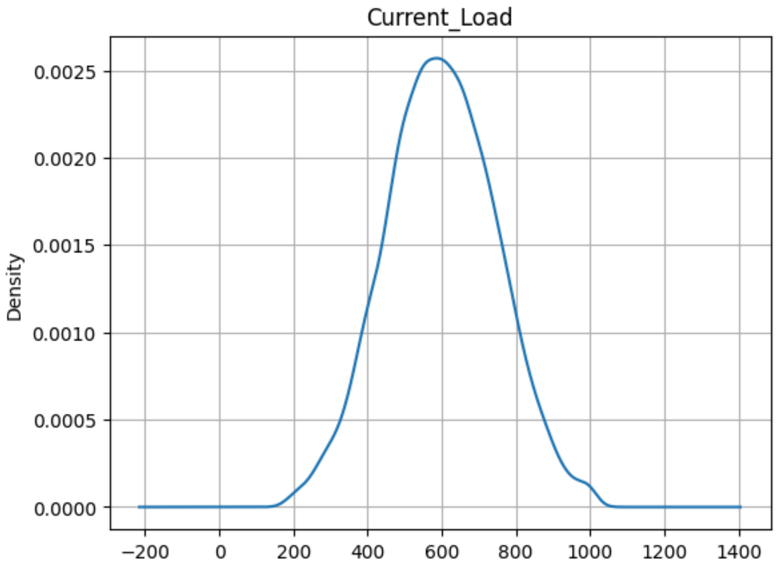
```
# Find the maximum
df_num.Current_Load.max()
```

```
np.float64(1000.0)
```

```
# Find the average
df_num.Current_Load.mean()
```

```
np.float64(599.0728915012942)
```

```
# Check the visualization
df_num.Current_Load.plot(kind='kde')  # Kernel Density Estimate
plt.title('Current_Load')
plt.grid()
plt.show()
```



Interpritation

- Minimum value is 189.43000000000012
- Maximum value is 1000.0
- Average value is 599.0728915012942.
- After looking at the distribution, I conclude that the graph shows Central tendency

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Engine_Performance

```
# minimum value
df_num.Engine_Performance.min()
```

```
np.float64(81.71)
```

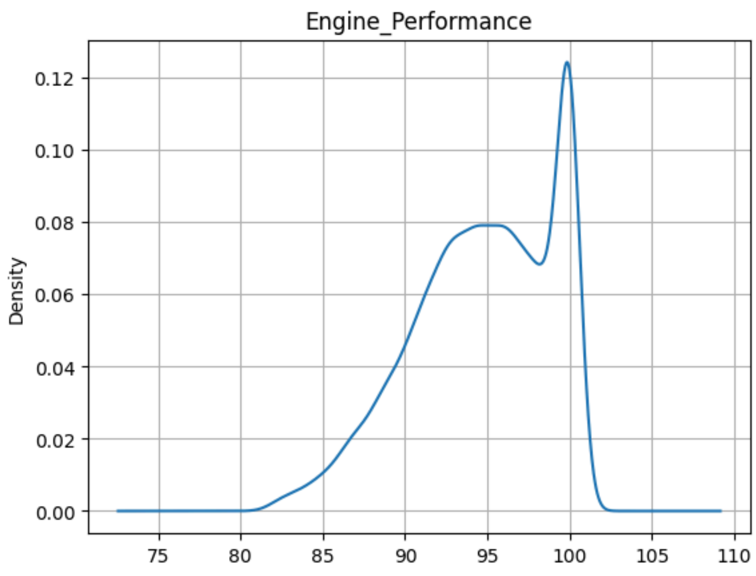
```
# maximum value is
df_num.Engine_Performance.max()  # Maximum value
```

```
np.float64(100.0)
```

```
# average value
df_num.Engine_Performance.mean()
```

```
np.float64(94.69988891285593)
```

```
# data visualitation
df_num.Engine_Performance.plot(kind='kde')  # Density plot
plt.title('Engine_Performance')
plt.grid()
plt.show()
```



Interpritation

- Minimum value is 81.71
- Maximum value is 100.0
- Average value is 94.69988891285593
- After looking at the distribution, I conclude that the graph shows Central tendency

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Altitude

```
# minimum value
df_num.Altitude.min()
```

```
np.float64(4973.770273943055)
```

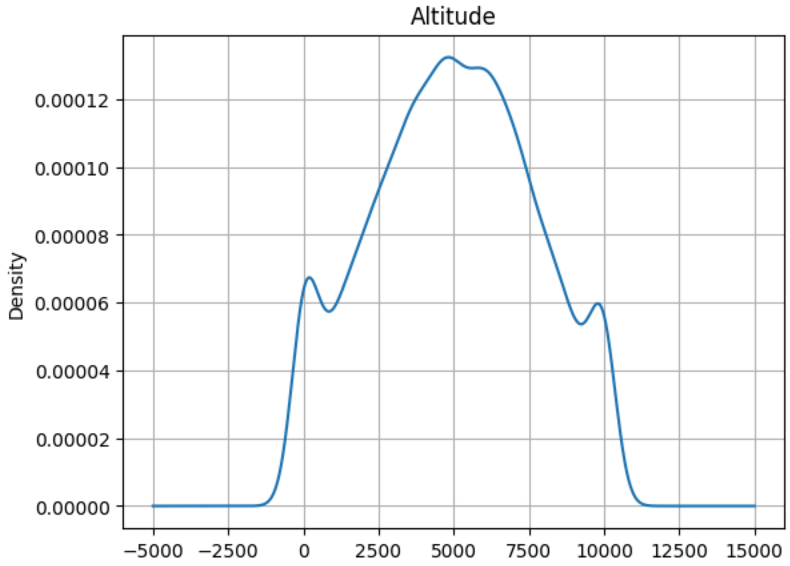
```
# maimum value
df_num.Altitude.max()
```

```
np.float64(10000.0)
```

```
# average value
df_num.Altitude.mean()
```

```
np.float64(4973.770273943055)
```

```
# data visualitation
df_num.Altitude.plot(kind='kde')
plt.title('Altitude')
plt.grid()
plt.show()
```



Interpritation

- Minimum value is 4973.77027394305
- Maximum value is 10000.0
- Average value is 4973.770273943055
- After looking at the distribution, I conclude that the graph shows Central tendency

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Airspeed

```
# minimum value
df_num.Airspeed.min()
```

```
np.float64(164.4)
```

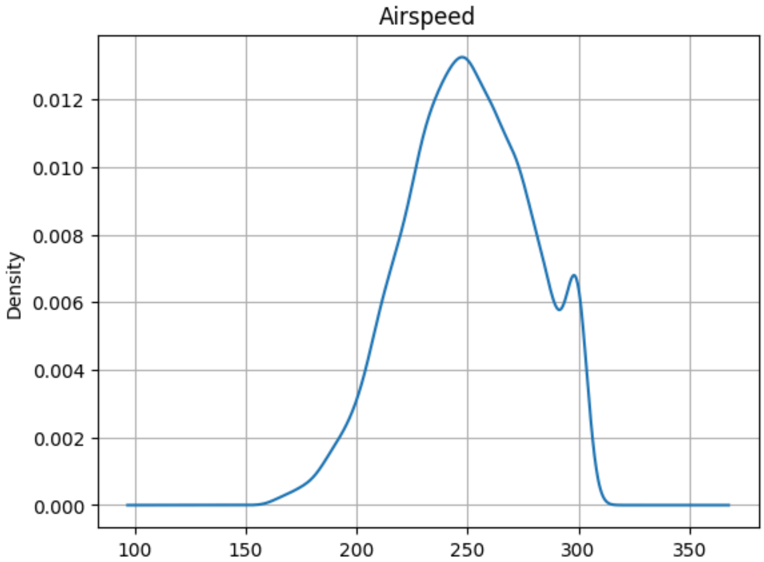
```
# maximum value
df_num.Airspeed.max()
```

```
np.float64(300.0)
```

```
# average value
df_num.Airspeed.mean()
```

```
np.float64(249.87278257118206)
```

```
# data visualitation
df_num.Airspeed.plot(kind='kde')
plt.title('Airspeed')
plt.grid()
plt.show()
```



Interpritation

- Minimum value is 4973.77027394305
- Maximum value is 300.0
- Average value is 249.87278257118206
- After looking at the distribution, I conclude that the graph shows Central tendency

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Maintenance_History

```
# minimum value
df_num.Maintenance_History.min()
```

```
np.int64(0)
```

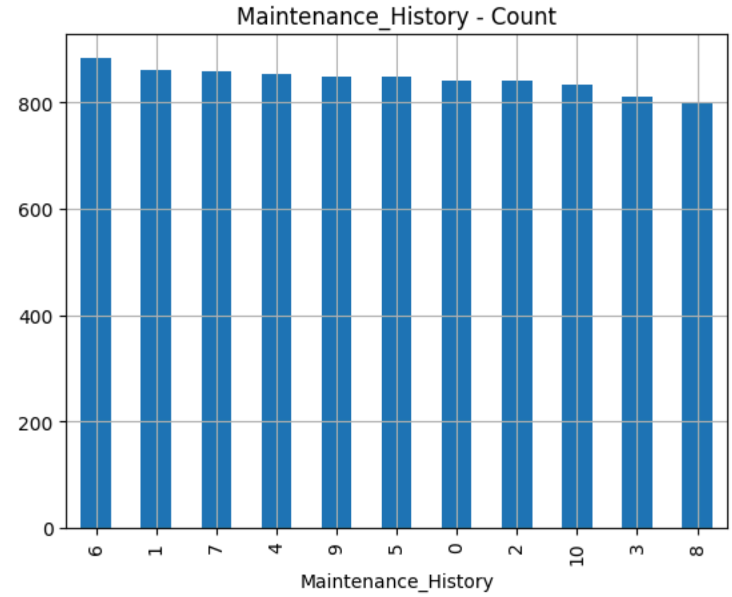
```
# maxmum value
df_num.Maintenance_History.max()
```

```
np.int64(10)
```

```
# average value
df_num.Maintenance_History.mean()
```

```
np.float64(4.990724762726488)
```

```
# Plot frequency of categorical values
df_num.Maintenance_History.value_counts().plot(kind='bar')
plt.title('Maintenance_History - Count')
plt.grid()
plt.show()
```



Interpritation

- Minimum value is 0
- Maximum value is 10.0
- Average value is 4.990724762726488

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Temperature

Generate 10 random numbers using numpy Close

```
df_num.Temperature.min()
```

```
np.float64(-10.0)
```

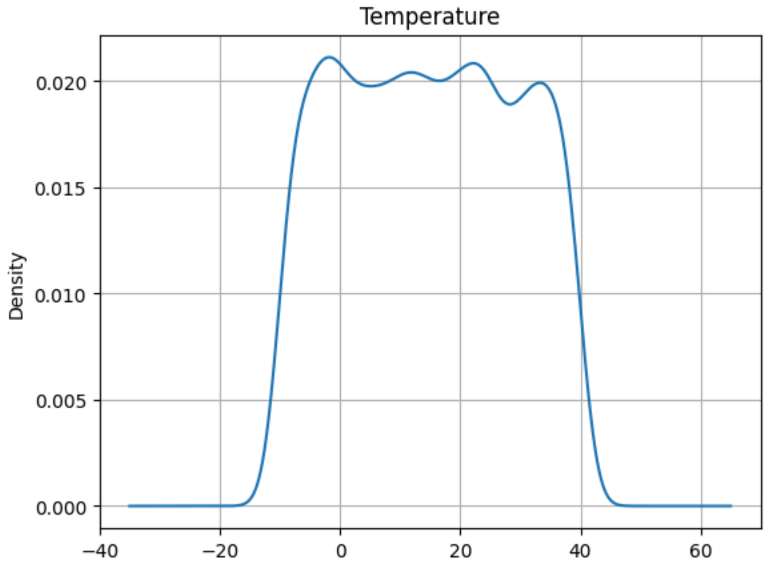
```
df_num.Temperature.max()
```

```
np.float64(40.0)
```

```
df_num.Temperature.mean()
```

```
np.float64(14.796246764452112)
```

```
df_num.Temperature.plot(kind='kde')
plt.title('Temperature')
plt.grid()
plt.show()
```



Interpritation

- Minimum Insider Sentiment_Score_Social is -10
- Maximum Insider Sentiment_Score_Social is 40
- Average Insider Sentiment_Score_Social is 14.79624676445211
- After looking at the distribution, I conclude that the distribution is normal

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Air_Traffic_Control_Delay

```
df_num.Air_Traffic_Control_Delay.min()
```

```
np.float64(0.0)
```

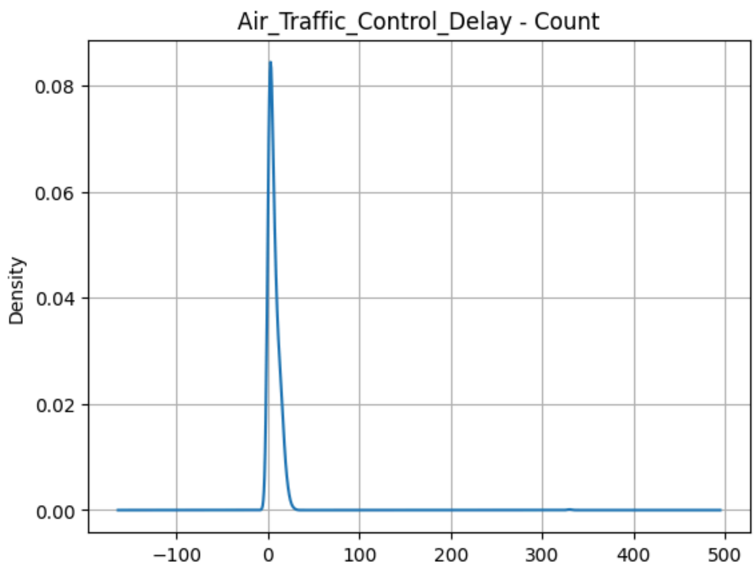
```
df_num.Air_Traffic_Control_Delay.max()
```

```
np.float64(14.446250000000001)
```

```
df_num.Air_Traffic_Control_Delay.mean()
```

```
np.float64(4.549174126402071)
```

```
df_num.Air_Traffic_Control_Delay.value_counts().plot(kind='kde')
plt.title('Air_Traffic_Control_Delay - Count')
plt.grid()
plt.show()
```



Interpritation

- Minimum Insider Sentiment_Score_Social is 0
- Maximum Insider Sentiment_Score_Social is 14.4466
- Average Insider Sentiment_Score_Social is 4.549174126402071
- After looking at the distribution, I conclude that the graph shows Central tendency

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Ground_Staff_Activity

```
df_num.Ground_Staff_Activity.min()
```

```
np.int64(0)
```

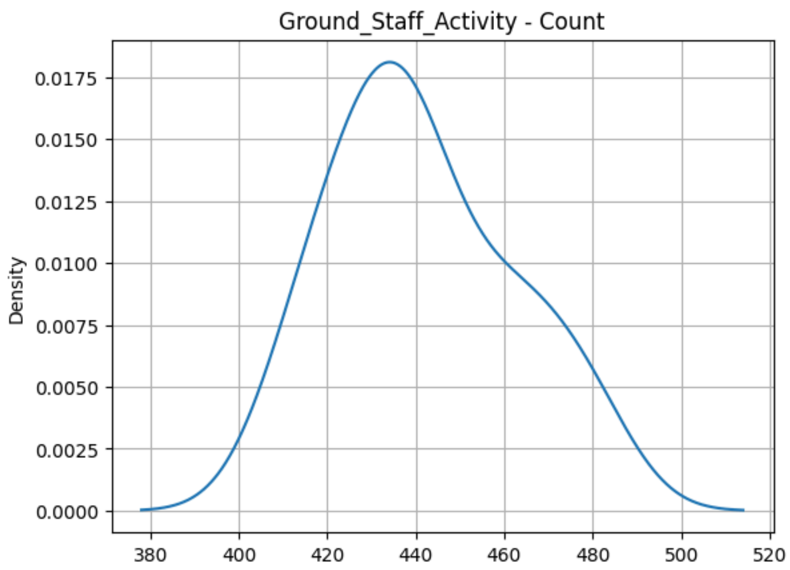
```
df_num.Ground_Staff_Activity.max()
```

```
np.int64(20)
```

```
df_num.Ground_Staff_Activity.mean()
```

```
np.float64(9.885785159620362)
```

```
df_num.Ground_Staff_Activity.value_counts().plot(kind='kde')
plt.title('Ground_Staff_Activity - Count')
plt.grid()
plt.show()
```



Interpritation

- Minimum Insider Sentiment_Score_Social is 0
- Maximum Insider Sentiment_Score_Social is 20
- Average Insider Sentiment_Score_Social is 9.885785159620362
- After looking at the distribution, I conclude that the graph shows Central tendency

```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

'CCTV_Anomaly_Flag

```
df_num.CCTV_Anomaly_Flag.min()
```

```
np.int64(0)
```

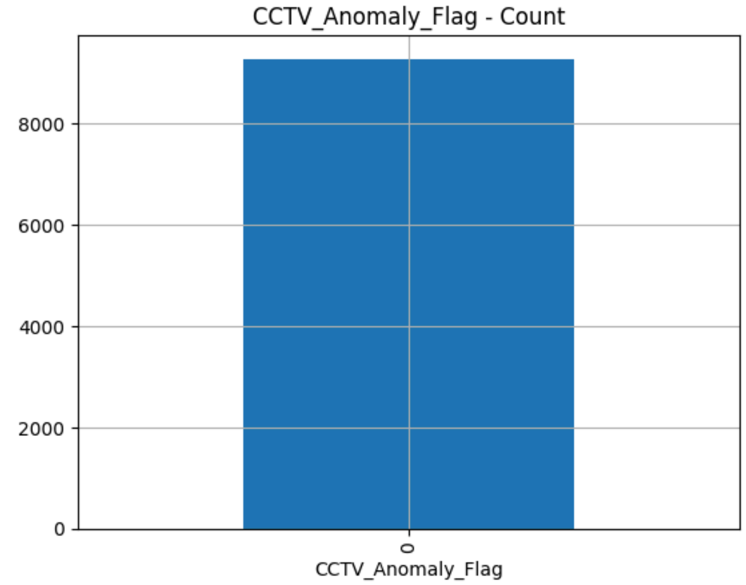
```
df_num.CCTV_Anomaly_Flag.max()
```

```
np.int64(0)
```

```
df_num.CCTV_Anomaly_Flag.mean()
```

```
np.float64(0.0)
```

```
df_num.CCTV_Anomaly_Flag.mean()

df_num.CCTV_Anomaly_Flag.value_counts().plot(kind='bar')
plt.title('CCTV_Anomaly_Flag - Count')
plt.grid()
plt.show()
```




```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

Pilot_Communication_Score

```
df_num.Pilot_Communication_Score.min()
```

```
np.float64(0.01)
```

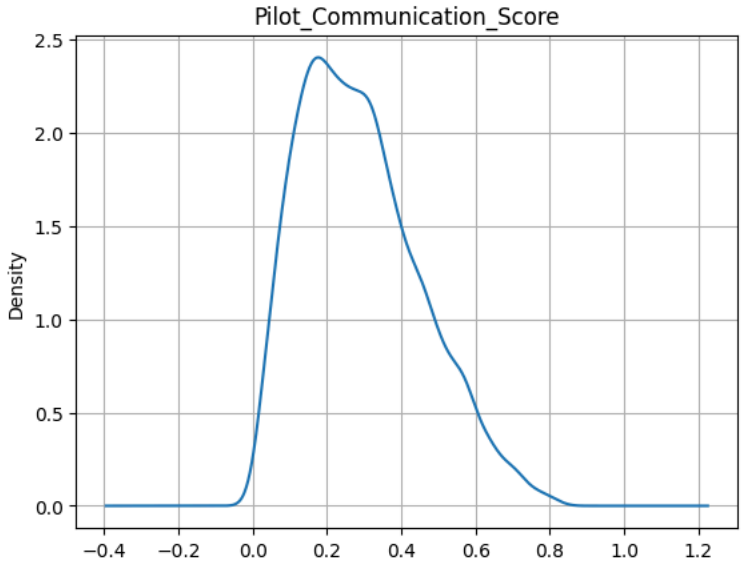
```
df_num.Pilot_Communication_Score.max()
```

```
np.float64(0.82)
```

```
df_num.Pilot_Communication_Score.mean()
```

```
np.float64(0.29043895599654873)
```

```
df_num.Pilot_Communication_Score.plot(kind='kde')
plt.title('Pilot_Communication_Score')
plt.grid()
plt.show()
```




```
# Check the columns

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

System_Failure

```
df_num.System_Failure.min()
```

```
np.int64(0)
```

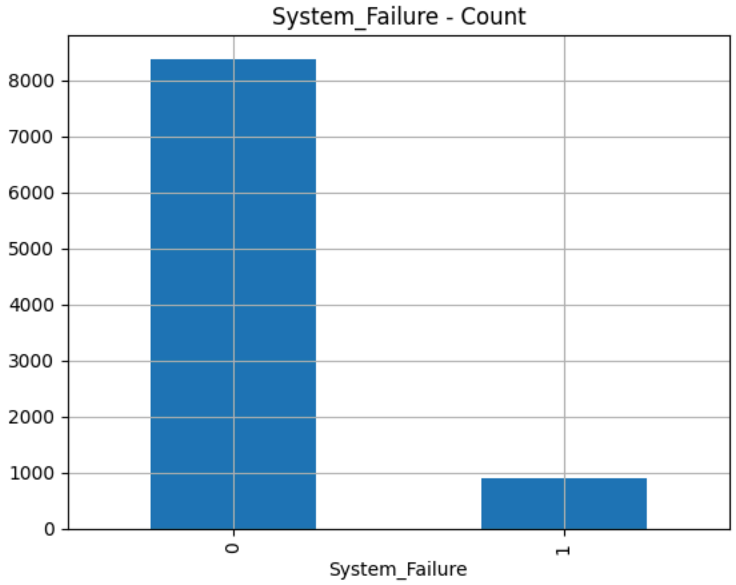
```
df_num.System_Failure.max()
```

```
np.int64(1)
```

```
df_num.System_Failure.mean()
```

```
np.float64(0.09652717860224332)
```

```
df_num.System_Failure.value_counts().plot(kind='bar')
plt.title('System_Failure - Count')
plt.grid()
plt.show()
```



Univeriated analysis an categorical variaables

```
# Check the columns

df_cat.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

Timestamp

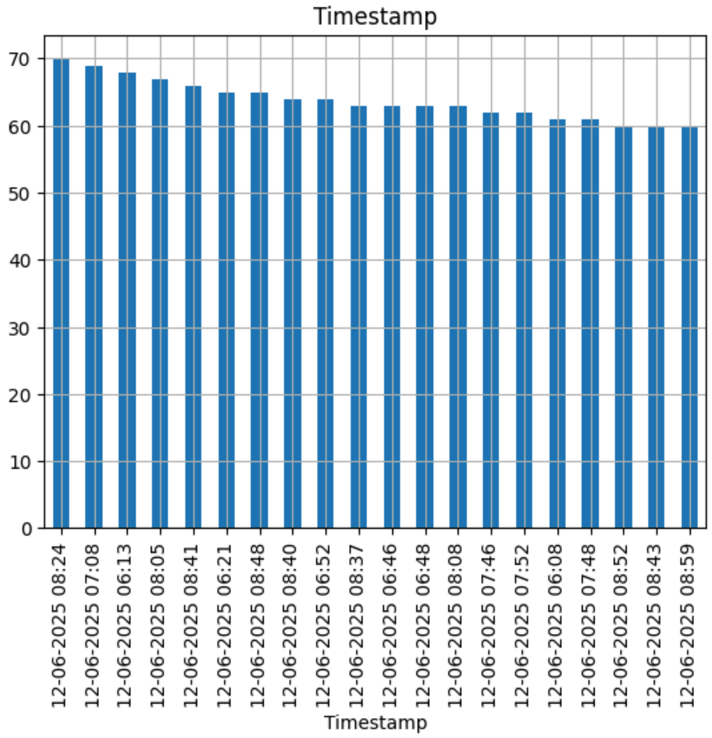
```
# Check the count of each category present in the column

df_cat.Timestamp.value_counts()
```

	count
Timestamp	
12-06-2025 08:24	70
12-06-2025 07:08	69
12-06-2025 06:13	68
12-06-2025 08:05	67
12-06-2025 08:41	66
...	...
12-06-2025 06:11	38
12-06-2025 08:34	38
12-06-2025 07:14	38
12-06-2025 06:44	36
12-06-2025 07:18	33

181 rows × 1 columns

dtype: int64

```
# Create a Visualization

df_cat.Timestamp.value_counts().head(20).plot(kind = 'bar')
plt.title('Timestamp')
plt.grid()
```




```
# Check the columns

df_cat.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

Flight_ID

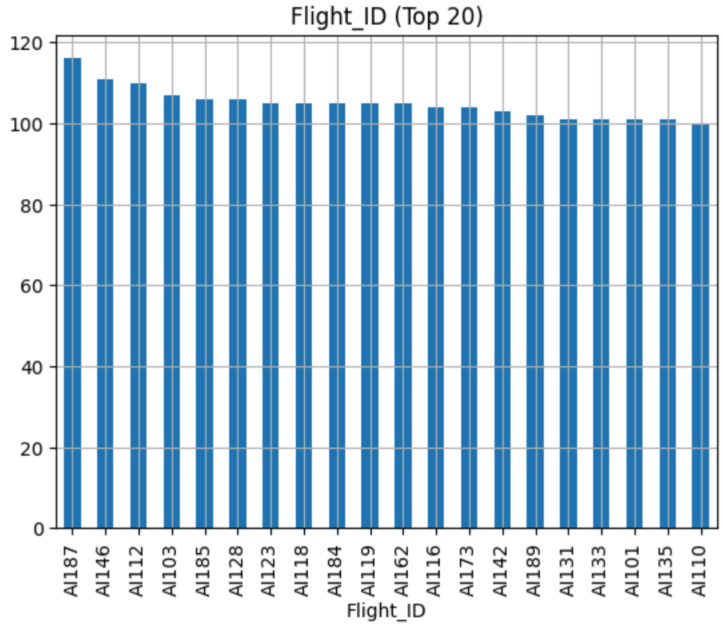
```
# Check the count of each category present in the column
df_cat.Flight_ID.value_counts()
```

Flight_ID	count
AI187	116
AI146	111
AI112	110
AI103	107
AI185	106
...	...
AI190	81
AI161	80
AI121	78
AI165	77
AI183	75

100 rows × 1 columns

dtype: int64

```
# Create a visualization (top 20 IDs for clarity)
df_cat.Flight_ID.value_counts().head(20).plot(kind='bar')
plt.title('Flight_ID (Top 20)')
plt.grid()
plt.show()
```




```
# Check the columns

df_cat.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

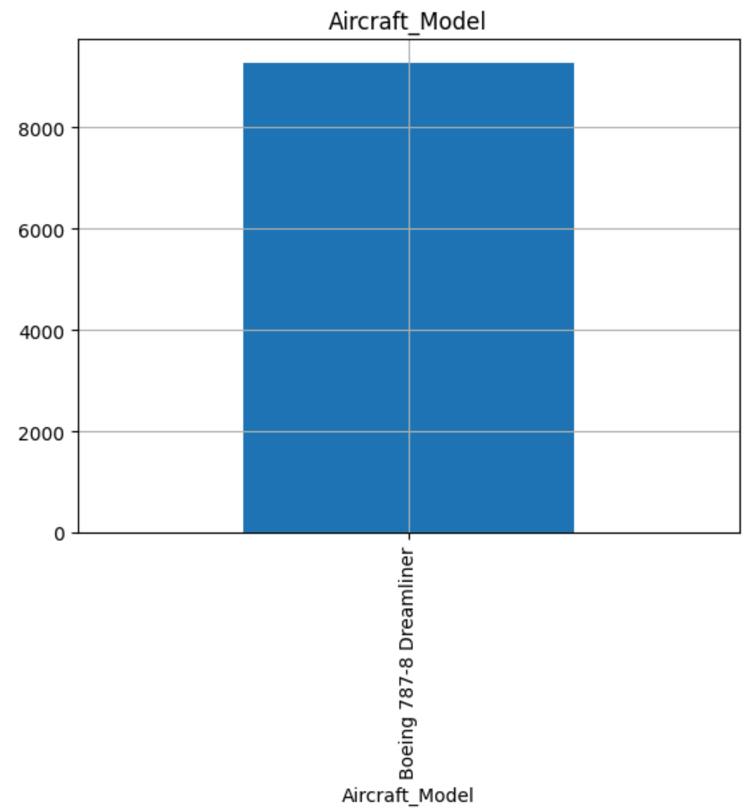
'Aircraft_Model

```
# Check the count of each category present in the column
df_cat.Aircraft_Model.value_counts()
```

Aircraft_Model	count
Boeing 787-8 Dreamliner	9272

dtype: int64

```
# Create a visualization
df_cat.Aircraft_Model.value_counts().plot(kind='bar')
plt.title('Aircraft_Model')
plt.grid()
plt.show()
```




```
# Check the columns

df_cat.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

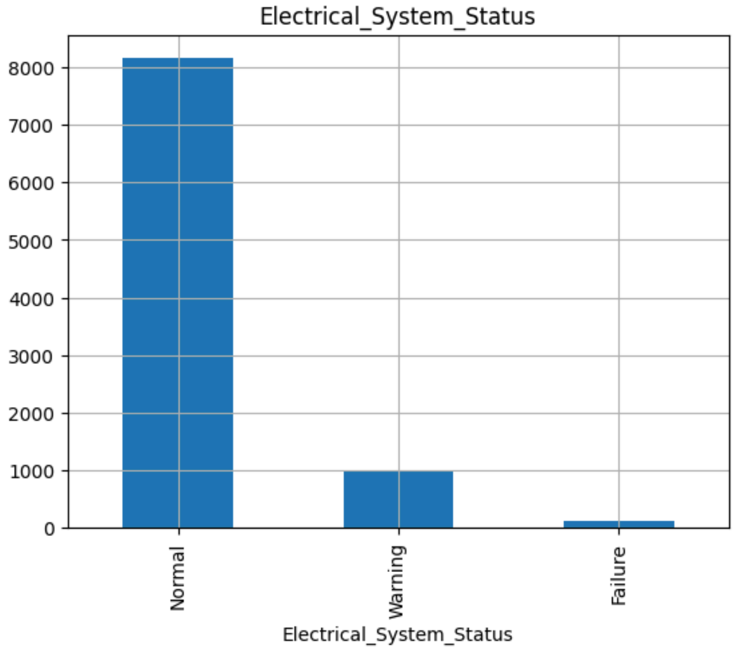
Electrical_System_Status

```
# Check the count of each category present in the column
df_cat.Electrical_System_Status.value_counts()
```

Electrical_System_Status	count
Normal	8153
Warning	991
Failure	128

dtype: int64

```
# Create a visualization
df_cat.Electrical_System_Status.value_counts().plot(kind='bar')
plt.title('Electrical_System_Status')
plt.grid()
plt.show()
```




```
# Check the columns

df_cat.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

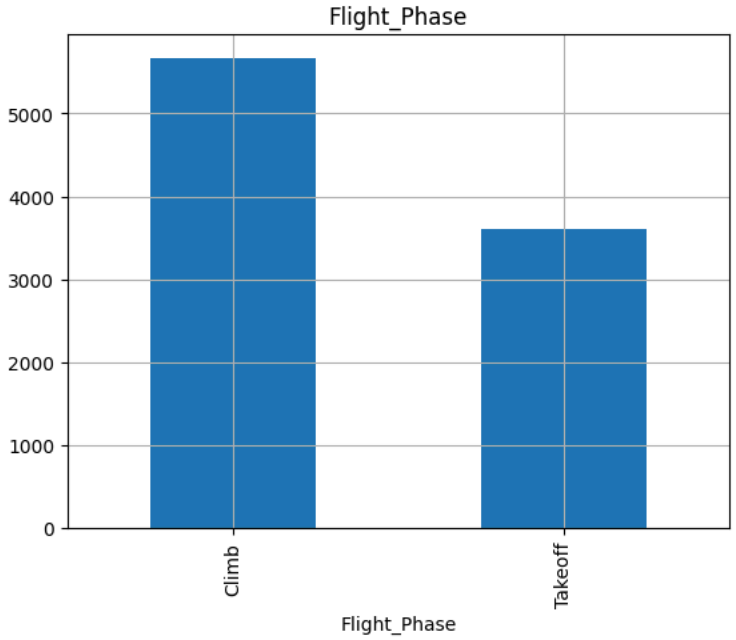
Flight_Phase

```
# Check the count of each category present in the column
df_cat.Flight_Phase.value_counts()
```

Flight_Phase	count
Climb	5666
Takeoff	3606

dtype: int64

```
# Create a visualization
df_cat.Flight_Phase.value_counts().plot(kind='bar')
plt.title('Flight_Phase')
plt.grid()
plt.show()
```




```
# Check the columns

df_cat.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

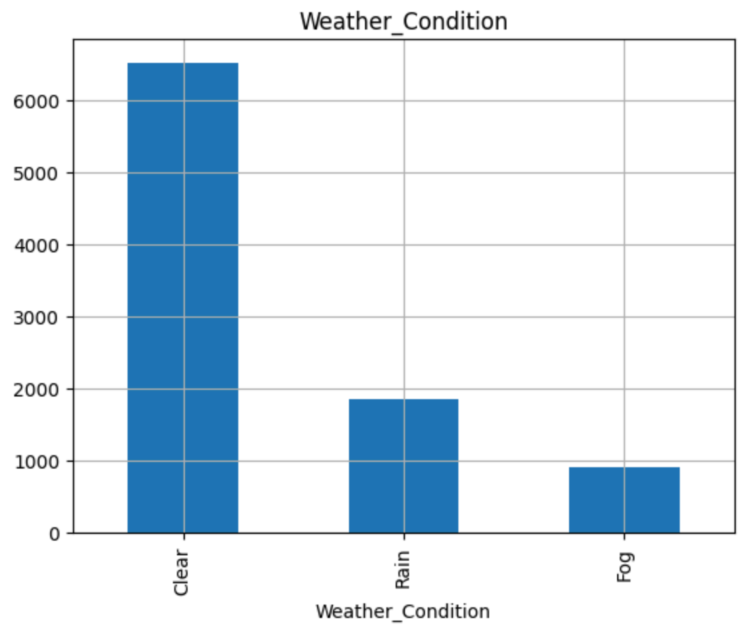
Weather_Condition

```
# Check the count of each category present in the column
df_cat.Weather_Condition.value_counts()
```

	count
Weather_Condition	
Clear	6517
Rain	1847
Fog	908

dtype: int64

```
# Create a visualization
df_cat.Weather_Condition.value_counts().plot(kind='bar')
plt.title('Weather_Condition')
plt.grid()
plt.show()
```



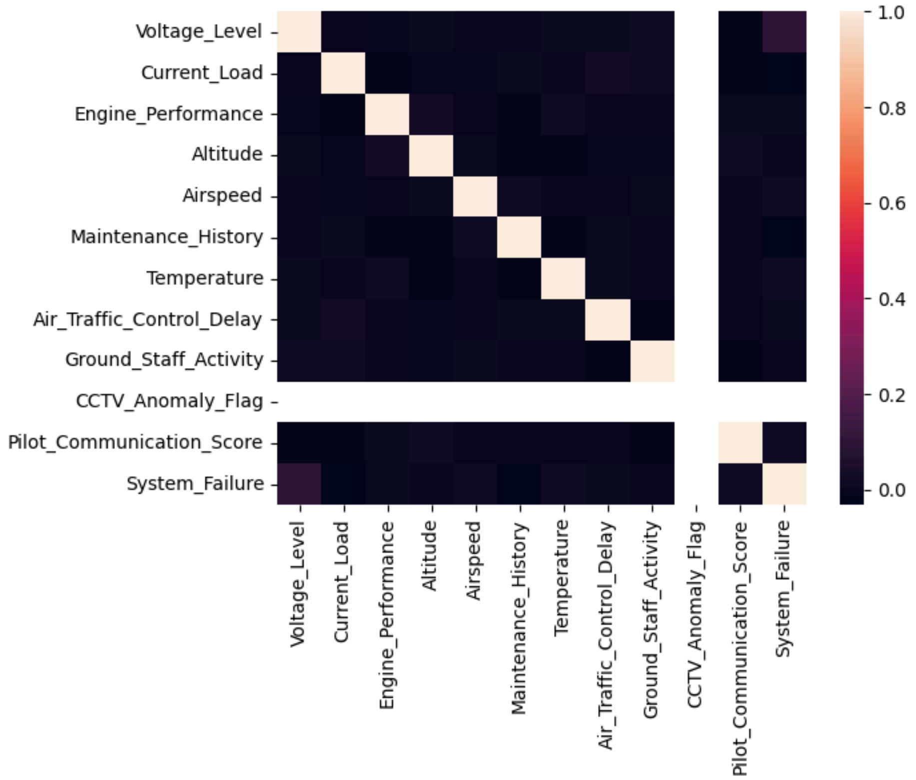
Bivariate Analysis

```
# To find the best pairs, we have to conclude the correlation matrix
df_num.corr()
```

	Voltage_Level	Current_Load	Engine_Performance	Altitude	Airspeed	Maintenance_History	Temperature	Ai
Voltage_Level	1.000000	-0.007081	-0.010190	0.003049	0.000674	0.000264	0.002769	
Current_Load	-0.007081	1.000000	-0.012897	-0.009052	-0.008944	0.005412	-0.005843	
Engine_Performance	-0.010190	-0.012897	1.000000	0.025060	-0.004624	-0.012317	0.014078	
Altitude	0.003049	-0.009052	0.025060	1.000000	0.006055	-0.011756	-0.012935	
Airspeed	0.000674	-0.008944	-0.004624	0.006055	1.000000	0.014505	-0.001767	
Maintenance_History	0.000264	0.005412	-0.012317	-0.011756	0.014505	1.000000	-0.022750	
Temperature	0.002769	-0.005843	0.014078	-0.012935	-0.001767	-0.022750	1.000000	
Air_Traffic_Control_Delay	0.006470	0.024029	-0.007142	-0.009638	-0.002596	0.008049	0.002976	
Ground_Staff_Activity	0.012410	0.012806	-0.000006	-0.010356	0.005658	-0.005073	-0.005135	
CCTV_Anomaly_Flag	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Pilot_Communication_Score	-0.011411	-0.012745	0.003049	0.011485	-0.004982	-0.004294	-0.000256	
System_Failure	0.091142	-0.028598	0.007008	0.000628	0.015875	-0.031328	0.013157	

```
# Visualize the correlation matrix
sns.heatmap(df_num.corr())
```

```
<Axes: >
```



Interpretation

- After looking at the data and correlation matrix, I conclude that there is no correlation
- This is good for machine learning, because it avoids the problem of multicollinearity
- NOTE: Multicolinearity means when independent variables are correlated with each other and we don;t want multicolinearity while performing machine learning on any data

Bivariate analysis on one categorical and one numerical variable

```
# Check the categorical variables we have

df_cat.columns
```

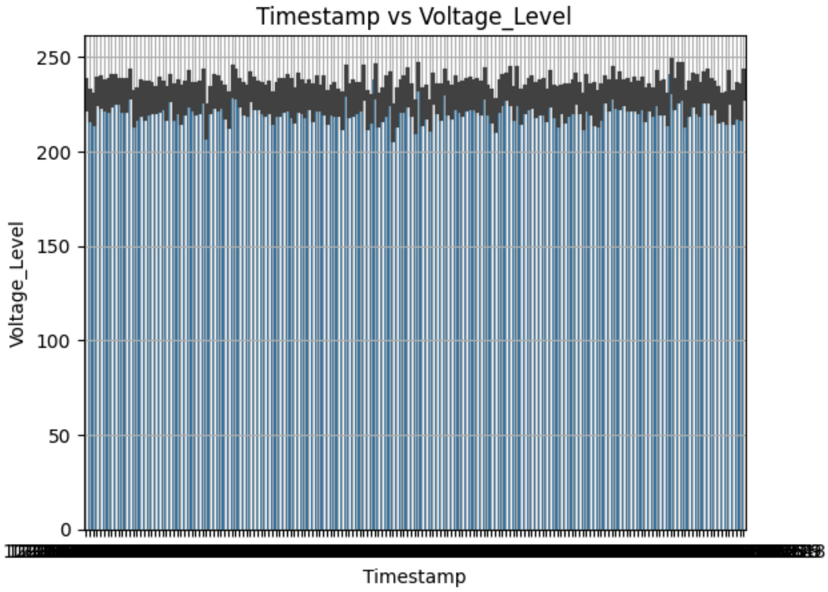
```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

```
# Check the numerical variable present in the data

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

'Timestamp','voltage_level'

```
# create a visualization

sns.barplot(x = df_airplain_crash.Timestamp,
           y = df_airplain_crash.Voltage_Level,
           data = df_airplain_crash)
plt.title('Timestamp vs Voltage_Level')
plt.grid()
plt.show()
```

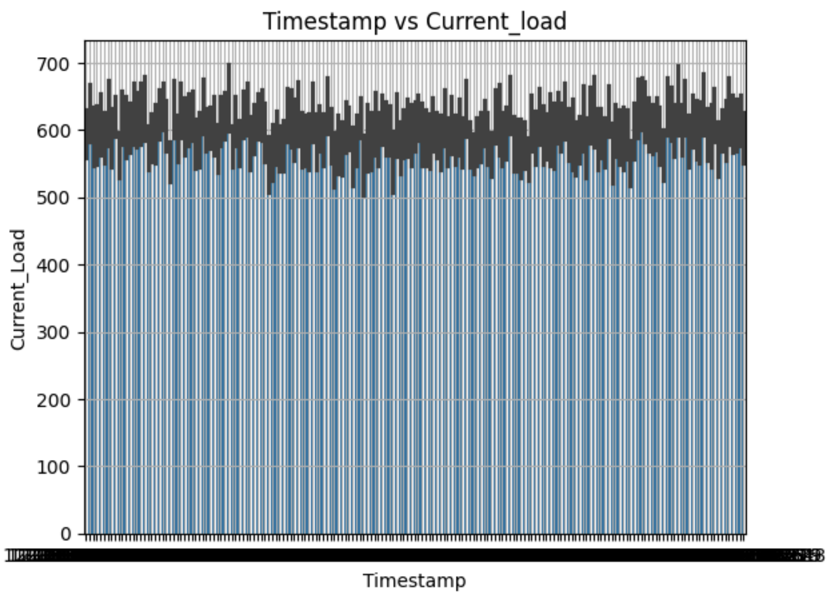


Interpretation

- The data is balanced, which means all Timestamp are contributing equal in Voltage_Level amaount

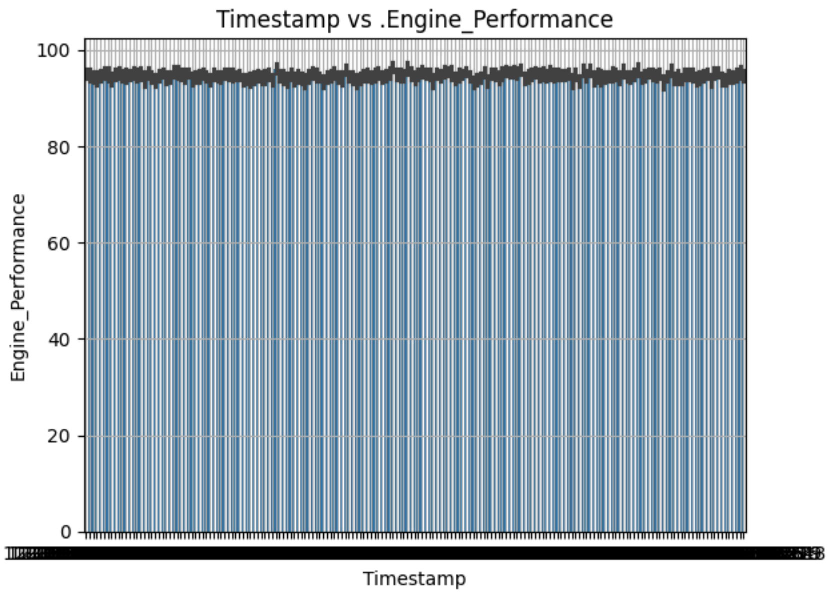
'Timestamp','Current_load'

```
# create a visualization

sns.barplot(x = df_airplain_crash.Timestamp,
           y = df_airplain_crash.Current_Load,
           data = df_airplain_crash)
plt.title('Timestamp vs Current_load')
plt.grid()
plt.show()
```



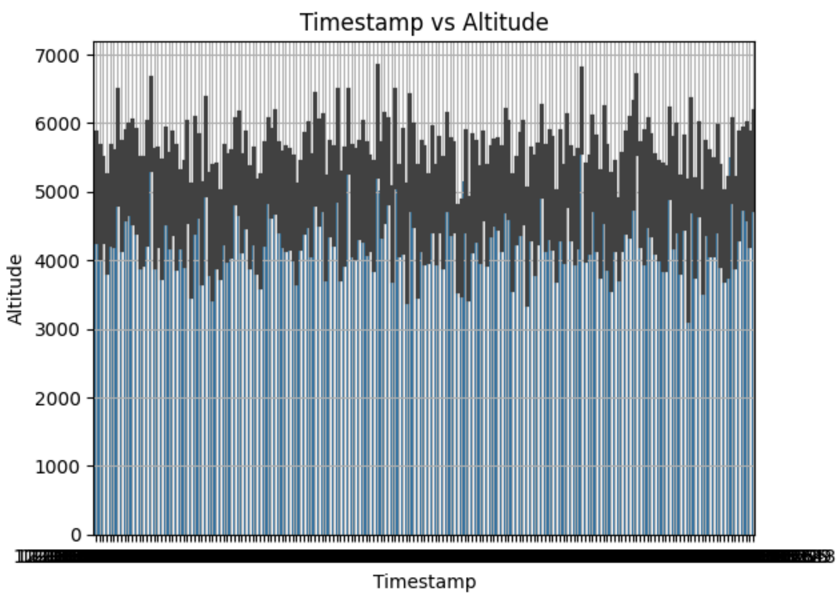
'Timestamp','Engine_Performance'

```
# create a visualization

sns.barplot(x = df_airplain_crash.Timestamp,
           y = df_airplain_crash.Engine_Performance,
           data = df_airplain_crash)
plt.title('Timestamp vs .Engine_Performance')
plt.grid()
plt.show()
```



'Timestamp','Altitude'

```
# create a visualization

sns.barplot(x = df_airplain_crash.Timestamp,
           y = df_airplain_crash.Altitude,
           data = df_airplain_crash)
plt.title('Timestamp vs Altitude')
plt.grid()
plt.show()
```



Multivariate Analysis

```
# Here also we have to find the pairs with two categorical and one numericle pair
# Check for cat. var

df_cat.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Flight_Phase', 'Last_Maintenance_Date', 'Weather_Condition'],
      dtype='object')
```

```
# Check for num. var

df_num.columns
```

```
Index(['Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Maintenance_History', 'Temperature',
```

```
            'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
            'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
           dtype='object')
```

'aircraft_model','Flight_ID','Voltage_level'

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
               values = df_airplain_crash.Voltage_Level.head(),
               aggfunc = 'min')
```

Aircraft_Model Flight_ID	Boeing 787-8 Dreamliner
AI111	246.28
AI114	244.90
AI145	164.65
AI189	206.65
AI191	229.27

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
               values = df_airplain_crash.Voltage_Level.head(),
               aggfunc = 'max')
```

Aircraft_Model Flight_ID	Boeing 787-8 Dreamliner
AI111	246.28
AI114	244.90
AI145	164.65
AI189	206.65
AI191	229.27

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
               values = df_airplain_crash.Voltage_Level.head(),
               aggfunc = 'mean')
```

Aircraft_Model Flight_ID	Boeing 787-8 Dreamliner
AI111	246.28
AI114	244.90
AI145	164.65
AI189	206.65
AI191	229.27

'aircraft_model','Flight_ID','current_load'

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
               values = df_airplain_crash.Current_Load.head(),
               aggfunc = 'min')
```

Aircraft_Model Flight_ID	Boeing 787-8 Dreamliner
AI111	514.29
AI114	579.26
AI145	443.42
AI189	769.53
AI191	653.33

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Current_Load.head(),
            aggfunc = 'mean')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	514.29
AI114	579.26
AI145	443.42
AI189	769.53
AI191	653.33

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Current_Load.head(),
            aggfunc = 'max')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	514.29
AI114	579.26
AI145	443.42
AI189	769.53
AI191	653.33

'aircraft_model','Flight_ID','altitude'

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Altitude.head(),
            aggfunc = 'min')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	0.00
AI114	9569.09
AI145	5972.60
AI189	3840.58
AI191	7497.39

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Altitude.head(),
            aggfunc = 'max')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	0.00
AI114	9569.09
AI145	5972.60
AI189	3840.58
AI191	7497.39

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Altitude.head(),
            aggfunc = 'mean')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	0.00
AI114	9569.09
AI145	5972.60
AI189	3840.58
AI191	7497.39

'aircraft_model','Flight_ID','airspeed'

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Airspeed.head(),
            aggfunc = 'min')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	278.51
AI114	242.98
AI145	272.38
AI189	215.24
AI191	241.20

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Airspeed.head(),
            aggfunc = 'max')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	278.51
AI114	242.98
AI145	272.38
AI189	215.24
AI191	241.20

```
pd.crosstab(index = df_airplain_crash.Flight_ID,columns = df_airplain_crash.Aircraft_Model,
            values = df_airplain_crash.Airspeed.head(),
            aggfunc = 'mean')
```

Aircraft_Model	Boeing 787-8 Dreamliner
Flight_ID	
AI111	278.51
AI114	242.98
AI145	272.38
AI189	215.24
AI191	241.20

Make data ready for machine learning

```
# Saperaate the dependent and independent variaables
```

```
df_airplain_crash.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Flight_Phase', 'Maintenance_History',
```

```
           'Last_Maintenance_Date', 'Weather_Condition', 'Temperature',
           'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
           'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
          dtype='object')
```

```
# Here in the data we are going to predict Stock_Price_Impact

target = df_airplain_crash.System_Failure
```

```
# Check for columns

df_airplain_crash.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Flight_Phase', 'Maintenance_History',
       'Last_Maintenance_Date', 'Weather_Condition', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score', 'System_Failure'],
      dtype='object')
```

```
# Drop the dependent variable from main dataframe

ind = df_airplain_crash.drop('System_Failure', axis = 1)
```

```
# Check whether that column is deleted or not
ind. columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Flight_Phase', 'Maintenance_History',
       'Last_Maintenance_Date', 'Weather_Condition', 'Temperature',
       'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score'],
      dtype='object')
```

```
# sales date is a date time column, and we never keep a date time column in the machine learning process

ind = ind.drop('Last_Maintenance_Date', axis = 1)
```

```
# Check whether that column is deleted or not

ind.columns
```

```
Index(['Timestamp', 'Flight_ID', 'Aircraft_Model', 'Electrical_System_Status',
       'Voltage_Level', 'Current_Load', 'Engine_Performance', 'Altitude',
       'Airspeed', 'Flight_Phase', 'Maintenance_History', 'Weather_Condition',
       'Temperature', 'Air_Traffic_Control_Delay', 'Ground_Staff_Activity',
       'CCTV_Anomaly_Flag', 'Pilot_Communication_Score'],
      dtype='object')
```

Perform encoding on categorical variable

```
# Separate the categorical columns from the independent variables
df_cat_ind = ind.select_dtypes(include = 'object')
df_cat_ind.head()
```

	Timestamp	Flight_ID	Aircraft_Model	Electrical_System_Status	Flight_Phase	Weather_Condition
0	12-06-2025 08:43	AI114	Boeing 787-8 Dreamliner	Normal	Takeoff	Clear
1	12-06-2025 08:19	AI111	Boeing 787-8 Dreamliner	Normal	Takeoff	Clear
2	12-06-2025 06:50	AI191	Boeing 787-8 Dreamliner	Normal	Climb	Clear
3	12-06-2025 06:40	AI189	Boeing 787-8 Dreamliner	Normal	Takeoff	Fog
4	12-06-2025 06:24	AI145	Boeing 787-8 Dreamliner	Normal	Climb	Rain

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

```
# Perform encoding

encoded = pd.get_dummies(df_cat_ind)
encoded.columns
```

```
Index(['Timestamp_12-06-2025 06:00', 'Timestamp_12-06-2025 06:01',
       'Timestamp_12-06-2025 06:02', 'Timestamp_12-06-2025 06:03',
       'Timestamp_12-06-2025 06:04', 'Timestamp_12-06-2025 06:05',
```

```
           'Timestamp_12-06-2025 06:06', 'Timestamp_12-06-2025 06:07',
           'Timestamp_12-06-2025 06:08', 'Timestamp_12-06-2025 06:09',
           ...
           'Flight_ID_AI199', 'Aircraft_Model_Boeing 787-8 Dreamliner',
           'Electrical_System_Status_Failure', 'Electrical_System_Status_Normal',
           'Electrical_System_Status_Warning', 'Flight_Phase_Climb',
           'Flight_Phase_Takeoff', 'Weather_Condition_Clear',
           'Weather_Condition_Fog', 'Weather_Condition_Rain'],
          dtype='object', length=290)
```

```
# Check the data encoding data frame

encoded.head().T
```

	0	1	2	3	4
Timestamp_12-06-2025 06:00	False	False	False	False	False
Timestamp_12-06-2025 06:01	False	False	False	False	False
Timestamp_12-06-2025 06:02	False	False	False	False	False
Timestamp_12-06-2025 06:03	False	False	False	False	False
Timestamp_12-06-2025 06:04	False	False	False	False	False
...	...	...	...	...	...
Flight_Phase_Climb	False	False	True	False	True
Flight_Phase_Takeoff	True	True	False	True	False
Weather_Condition_Clear	True	True	True	False	False
Weather_Condition_Fog	False	False	False	True	False
Weather_Condition_Rain	False	False	False	False	True

290 rows × 5 columns

Perform scaling on numerical data

```
# Make a new dataframe for the independent numerical variable

df_num_ind = ind.select_dtypes(include = 'number')
```

```
# Import the standard scaler for scaling

from sklearn.preprocessing import StandardScaler
```

```
# Initiate the StandardScaler

ss = StandardScaler()
```

```
# Transform the data (Numerical Data)

scalled = ss.fit_transform(df_num_ind)
```

```
# Check the normalized data

scalled
```

```
array([[ 0.52731067, -0.13396766,  0.83922802, ..., -0.47705278,
         0.        , -0.12600547],
       [ 0.57411762, -0.57327149, -1.0240842 , ...,  0.84543729,
         0.        , -0.31095445],
       [-0.00282901,  0.3668671 ,  0.566606  , ..., -0.31174152,
         0.        ,  0.30554215],
       ...,
       [-0.52822015,  0.58594428,  0.16122892, ...,  0.84543729,
         0.        , -0.86580138],
       [-0.17038435,  0.88643161, -0.12324623, ..., -1.13829781,
         0.        ,  2.03173261],
       [-1.77165715,  0.92301209, -1.20425179, ..., -1.30360907,
         0.        , -0.9891007 ]])
```

```
# Create a dataframe for scaled numerical data

df_scalled = pd.DataFrame(scalled, columns = df_num_ind.columns)
```

```
# Check the scaled DataFrame

df_scalled.head()
```

	Voltage_Level	Current_Load	Engine_Performance	Altitude	Airspeed	Maintenance_History	Temperature	Air_Traffic_Control_Delay
0	0.527311	-0.133968	0.839228	1.695153	-0.242738	1.587369	-0.125176	-0.647487
1	0.574118	-0.573271	-1.024084	-1.834758	1.008495	-1.581491	0.097824	-0.513835
2	-0.002829	0.366867	0.566606	0.930930	-0.305423	0.953597	1.637922	0.953770
3	-0.770056	1.152570	0.514452	-0.418019	-1.219636	-1.264605	0.578669	-0.688611
4	-2.194616	-1.052469	0.275019	0.368455	0.792619	0.953597	1.623985	1.223645

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

```
# Concatenate the encoded and scaled dataframe

df_main = pd.concat([encoded, df_scalled], axis = 1)
```

```
# Check whether the concatenation is successful or not

df_main.isnull().sum()
```

	0
Timestamp_12-06-2025 06:00	0
Timestamp_12-06-2025 06:01	0
Timestamp_12-06-2025 06:02	0
Timestamp_12-06-2025 06:03	0
Timestamp_12-06-2025 06:04	0
...	...
Temperature	0
Air_Traffic_Control_Delay	0
Ground_Staff_Activity	0
CCTV_Anomaly_Flag	0
Pilot_Communication_Score	0

301 rows × 1 columns

dtype: int64

Split the data in train and testing

```
# import the library for training and testing

from sklearn.model_selection import train_test_split
```

```
# Separate the data

X_train, X_test, y_train, y_test = train_test_split(df_main, target, test_size = 0.3)
```

```
# Check the data we have in training set

X_train.head()
```

	Timestamp_12-06-2025 06:00	Timestamp_12-06-2025 06:01	Timestamp_12-06-2025 06:02	Timestamp_12-06-2025 06:03	Timestamp_12-06-2025 06:04	Timestamp_12-06-2025 06:05	Timestamp_12-06-2025 06:06	Timestamp_12-06-2025 06:07	Timest 06-202
2804	False	False	False	False	False	False	False	False	
6868	False	False	False	False	False	False	False	False	
801	False	False	False	False	False	False	False	False	
1375	False	False	False	False	False	False	True	False	
3396	False	False	False	False	False	False	False	False	

5 rows × 301 columns

```
y_train.head()
```

	System_Failure
2804	1
6868	0
801	1
1375	0
3396	0

dtype: int64

```
# check the testing set
```

```
X_test.head()
```

	Timestamp_12-06-2025 06:00	Timestamp_12-06-2025 06:01	Timestamp_12-06-2025 06:02	Timestamp_12-06-2025 06:03	Timestamp_12-06-2025 06:04	Timestamp_12-06-2025 06:05	Timestamp_12-06-2025 06:06	Timestamp_12-06-2025 06:07	Timest 06-202
6753	False	False	False	False	False	False	False	False	
1602	False	False	False	False	False	False	False	False	
605	False	False	False	False	False	False	False	False	
8883	False	False	False	False	False	False	False	False	
1774	False	False	False	False	False	False	False	False	

5 rows × 301 columns

```
y_test.head()
```

	System_Failure
6753	0
1602	0
605	0
8883	0
1774	0

dtype: int64

Select the base model

Linear Regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Assuming X_train, X_test, y_train, y_test are already defined from train_test_split

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict on the testing data
predictions = model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.09166569728921695
```

Ridge Model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Assuming X_train, X_test, y_train, y_test are already defined from train_test_split

# Create a Ridge regression model
ridge_model = Ridge(alpha=1.0)  # alpha is the regularization strength

# Fit the model to the training data
ridge_model.fit(X_train, y_train)

# Predict on the testing data
predictions = ridge_model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.09143002804622594
```

Lasso Model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

# Assuming X_train, X_test, y_train, y_test are already defined from train_test_split

# Create a Lasso regression model
lasso_model = Lasso(alpha=1.0)  # alpha is the regularization strength

# Fit the model to the training data
lasso_model.fit(X_train, y_train)

# Predict on the testing data
predictions = lasso_model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.08734354134729566
```

Decision Tree Regressor

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Assuming X_train, X_test, y_train, y_test are already defined from train_test_split

# Create a Decision Tree regression model
tree_model = DecisionTreeRegressor()

# Fit the model to the training data
tree_model.fit(X_train, y_train)

# Predict on the testing data
predictions = tree_model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.16606757728253055
```

Random Forest Regressor

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Assuming X_train, X_test, y_train, y_test are already defined from train_test_spli

# Create a Random Forest regression model
```

```
forest_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
forest_model.fit(X_train, y_train)

# Predict on the testing data
predictions = forest_model.predict(X_test)

# Calculate mean squared error
mse = mean_squared_error(y_test, predictions)
```

```
Mean Squared Error: 0.09091362329259525
```

Gaussian Process Regressor

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Assuming X_train, X_test, y_train, y_test are already defined from train_test_split

# Create a Gaussian Process regression model with an RBF kernel
kernel = 1.0 * RBF(length_scale=1.0)
gp_model = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, random_state=42)

# Fit the model to the training data
gp_model.fit(X_train, y_train)

# Predict on the testing data
predictions, std_dev = gp_model.predict(X_test, return_std=True)

# Calculate mean squared error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

Polynomial Regression

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Assuming X_train, X_test, y_train, y_test are already defined from train_test_split

# Generate polynomial features
degree = 3  # degree of the polynomial
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
```