https://drive.google.com/drive/folders/1oNT2ydRH3PkxGuucXbDFZGukx11M3rpv?usp=sharing

GANESH G

25MML0043

LAB – 1

AIM:

The goal of this lab is to get hands-on experience with preparing real-world data for machine learning. Using the Titanic dataset, we'll clean the data, handle missing values, create and transform features, convert categories into numbers, scale values, and finally organize everything into a pipeline — making the dataset ready for building predictive models.

INTRODUCTION:

Before building any machine learning model, it's important to clean and prepare the data — a process called data preprocessing. In this lab, we'll use the Titanic dataset to practice these skills. This dataset is great for learning because it includes real-world challenges like missing values, different data types, and the need to create new features. Step by step, we'll get the data ready for modeling.

REAL WORLD EXAMPLE:

Predicting Hospital Patient Survival:

1. Load patient records from hospital database
2. Fill missing blood pressure or age from previous visits
3. Create "Risk Score" using age + comorbidities
4. Convert 'Admission Type' or 'Gender' into numeric codes
5. Normalize "Hospital Stay" and "Blood Pressure"
6. Drop irrelevant features (e.g., Patient ID)
7. Split into training (80%) and testing (20%) sets
8. Automate the above for any future patient prediction

ALGORITHM:

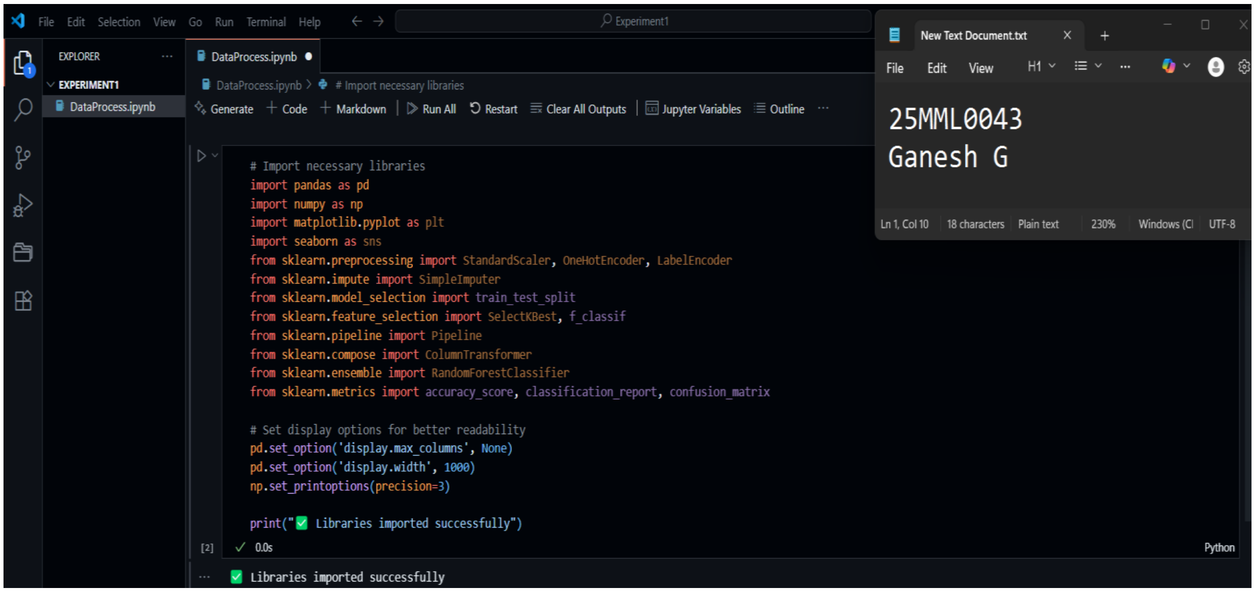
1. Import necessary libraries
   Import pandas, NumPy, seaborn, matplotlib, and scikit-learn tools.
2. Load the dataset
   Read the Titanic dataset into a Data Frame.
3. Explore the dataset
   View the first few rows, Check column data types, Identify missing values.
4. Handle missing values

Fill or drop missing values depending on the column.

5. Perform feature engineering
   Create new features (e.g., family size from siblings/spouses and parents/children).
   Extract titles from passenger names if needed.
6. Encode categorical variables
   Convert categorical columns like sex, embarked into numeric form using Label Encoding or One-Hot Encoding.
7. Split the dataset
   Divide the dataset into training and testing sets (e.g., 80% train, 20% test).

IMPLEMENTATION AND RESULTS:

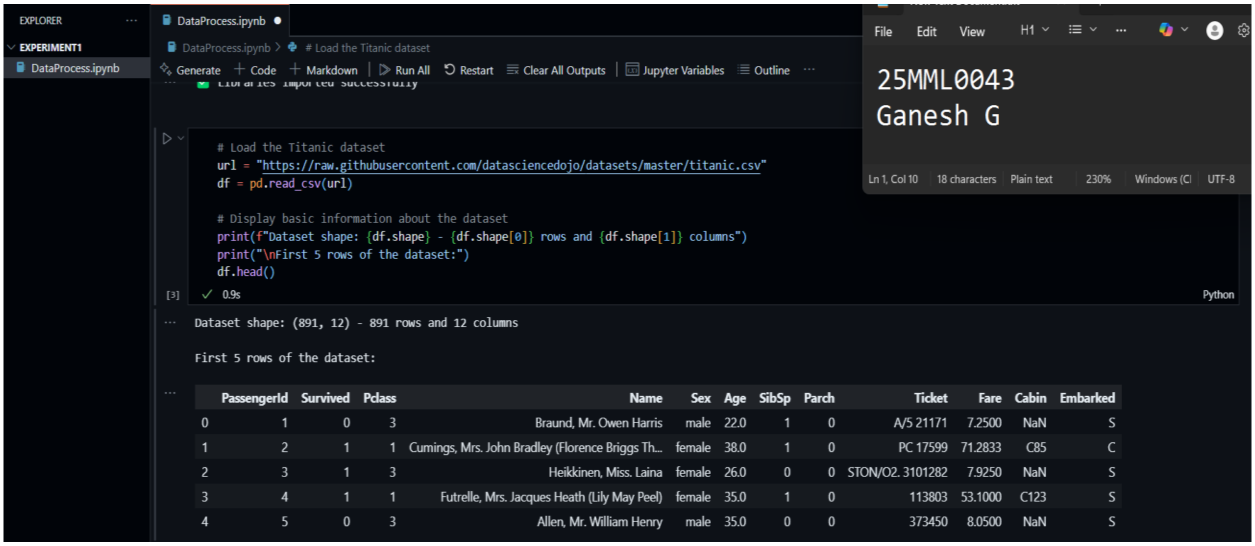
Import necessary libraries:




```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Set display options for better readability
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
np.set_printoptions(precision=3)

print("✅ Libraries imported successfully")
```

```
✅ Libraries imported successfully
```

Data Loading:

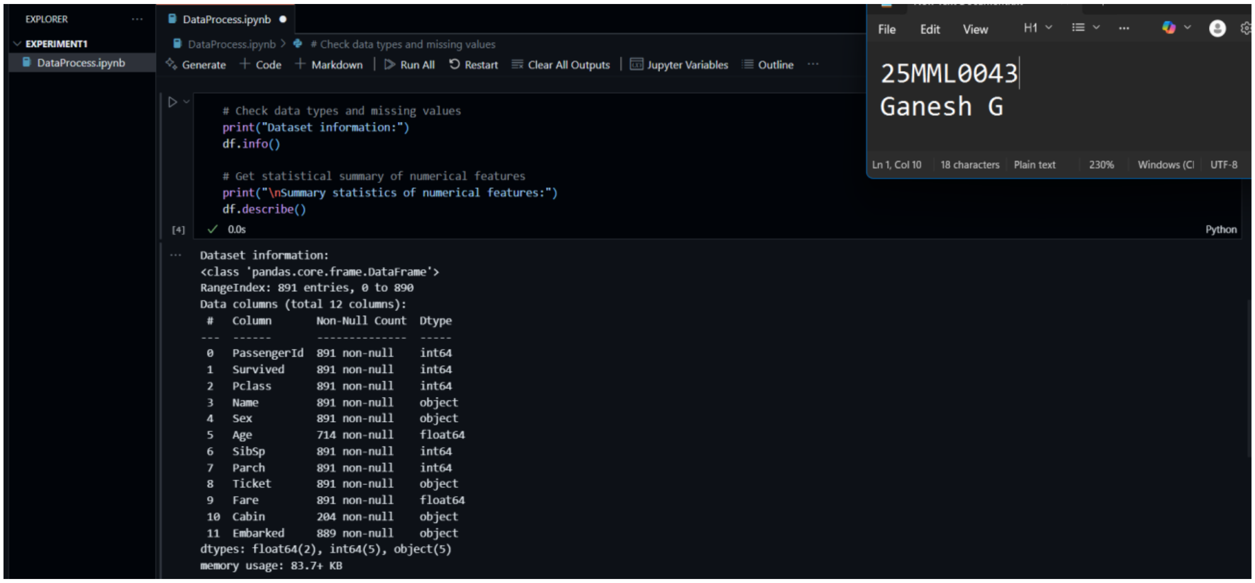


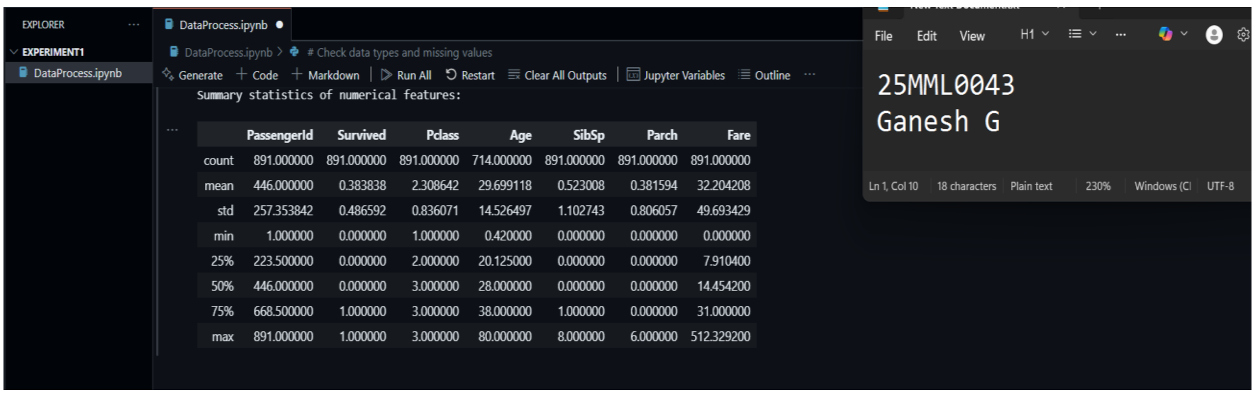

```python
# Load the Titanic dataset
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
df = pd.read_csv(url)

# Display basic information about the dataset
print(f"Dataset shape: {df.shape} - {df.shape[0]} rows and {df.shape[1]} columns")
print("\nFirst 5 rows of the dataset:")
df.head()
```

```
Dataset shape: (891, 12) - 891 rows and 12 columns

First 5 rows of the dataset:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

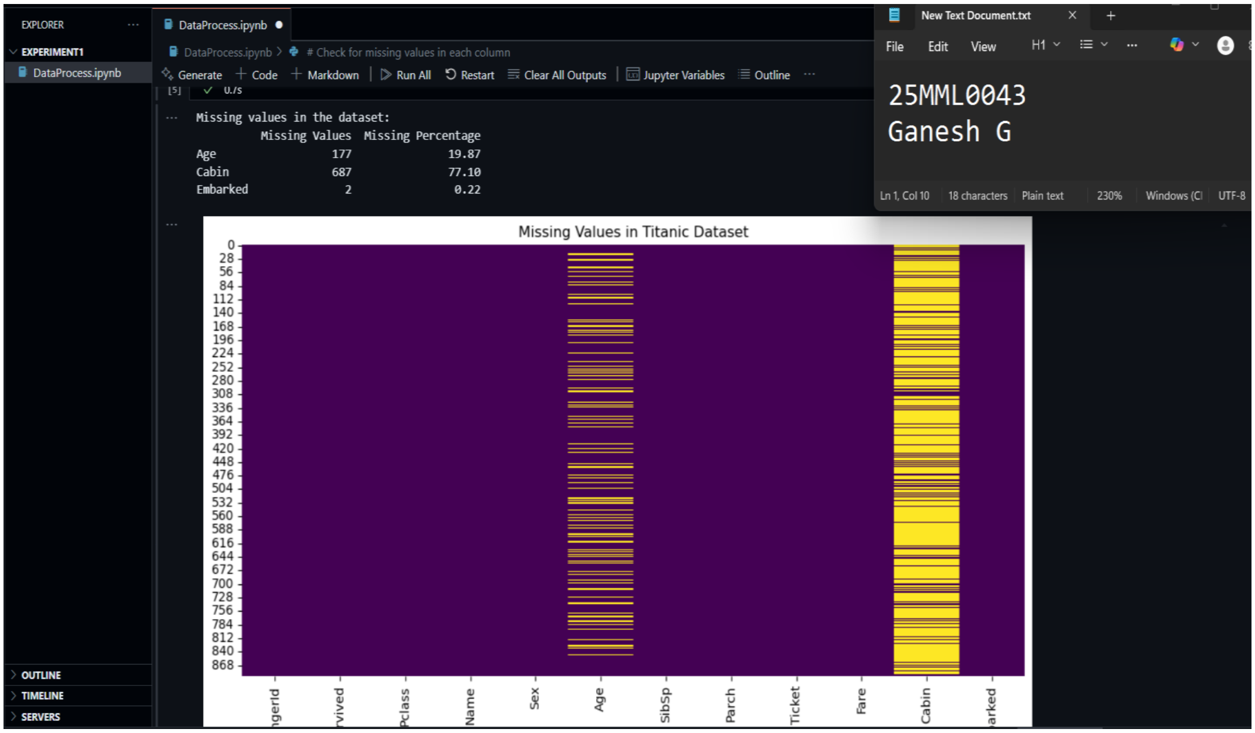
Understanding the Dataset Features:



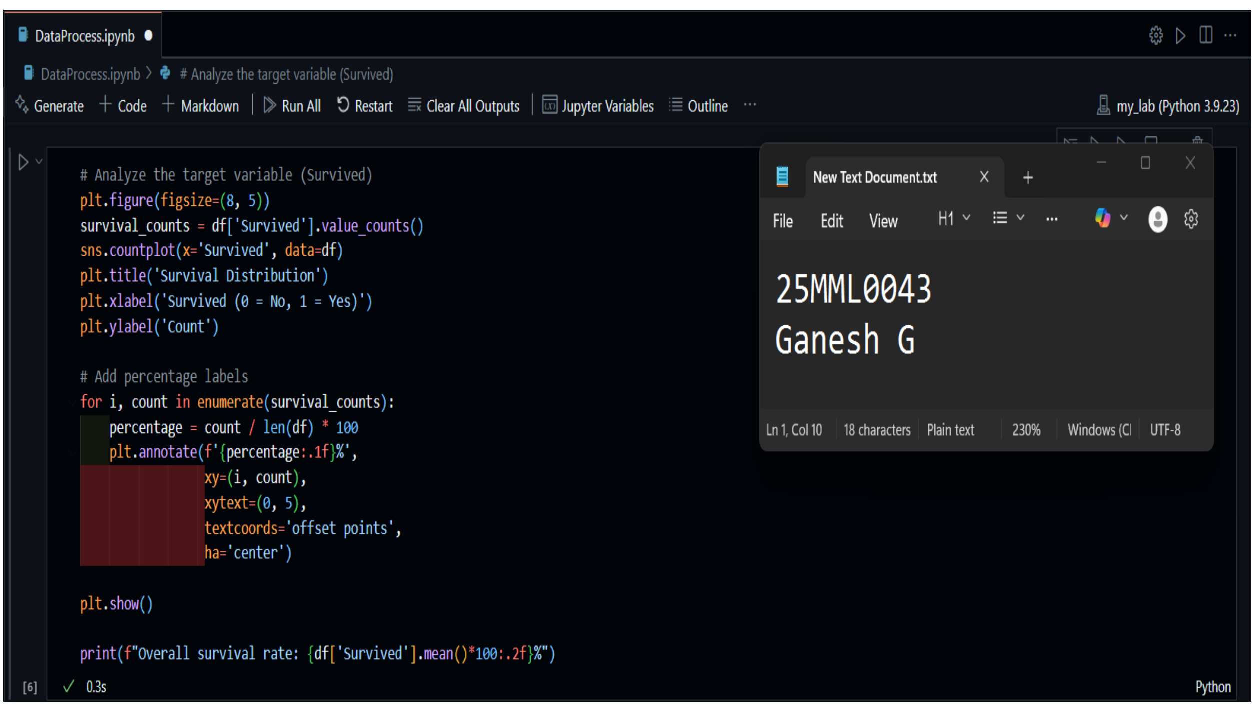


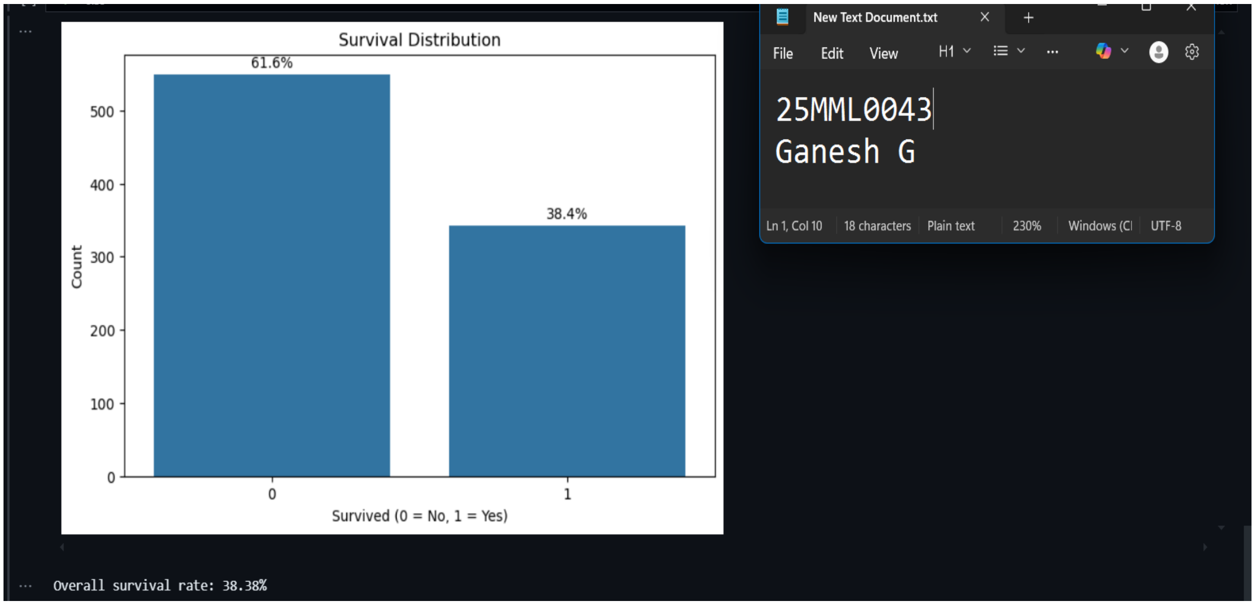
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Analyzing Missing Values:

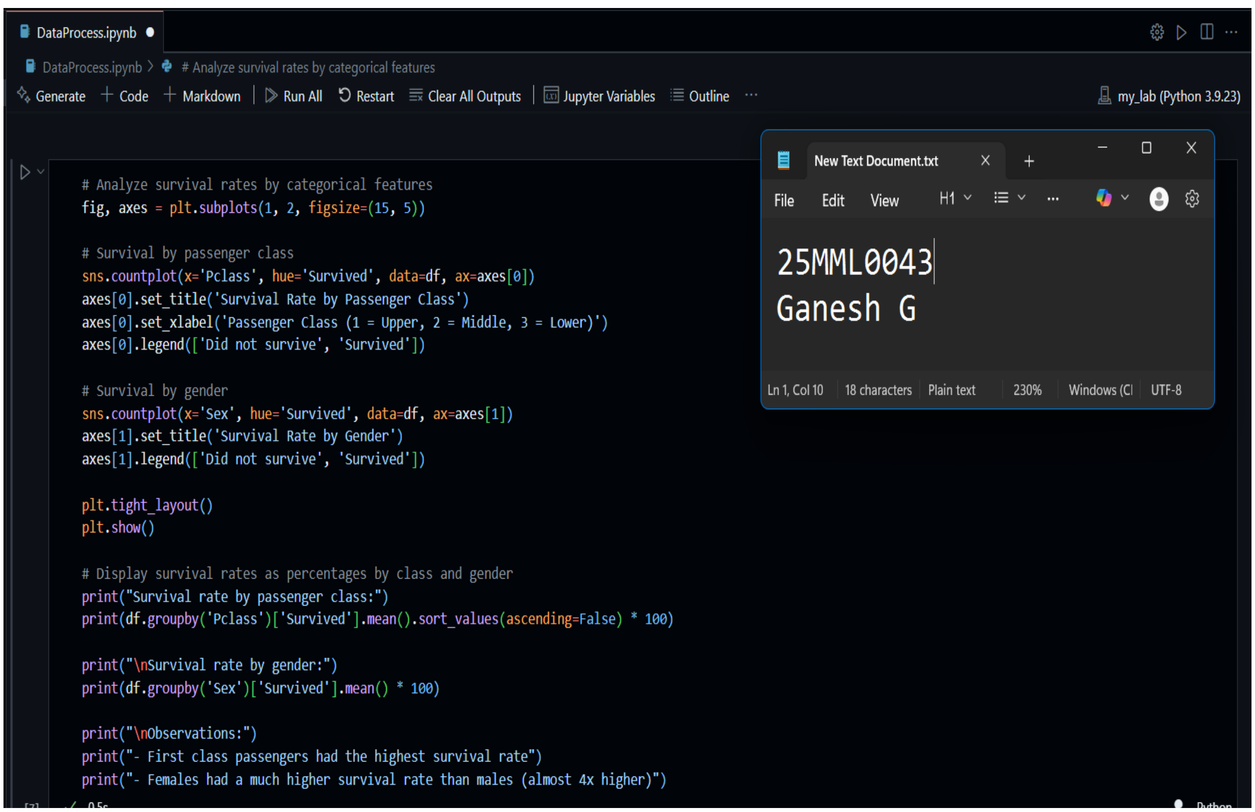


Exploratory Data Analysis (EDA):





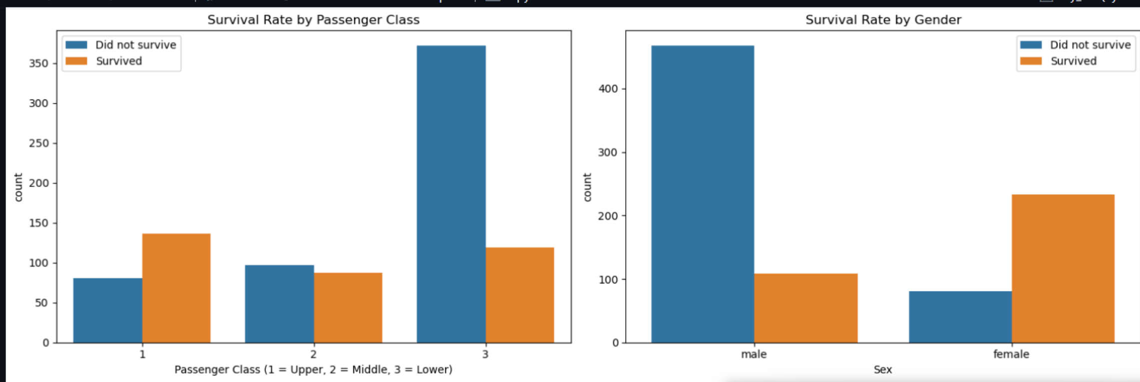
Categorical Features Analysis:



DataProcess.ipynb

DataProcess.ipynb > # Analyze survival rates by categorical features

Generate | + Code | + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | my_lab (Python 3.9.23)




```
Survival rate by passenger class:
Pclass
1    62.962963
2    47.282609
3    24.236253
Name: Survived, dtype: float64

Survival rate by gender:
Sex
female    74.203822
male      18.890815
Name: Survived, dtype: float64
```

New Text Document.txt

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CR | UTF-8

DataProcess.ipynb

DataProcess.ipynb > # Analyze numerical features: Age and Fare

Generate | + Code | + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | my_lab (Python 3.9.23)

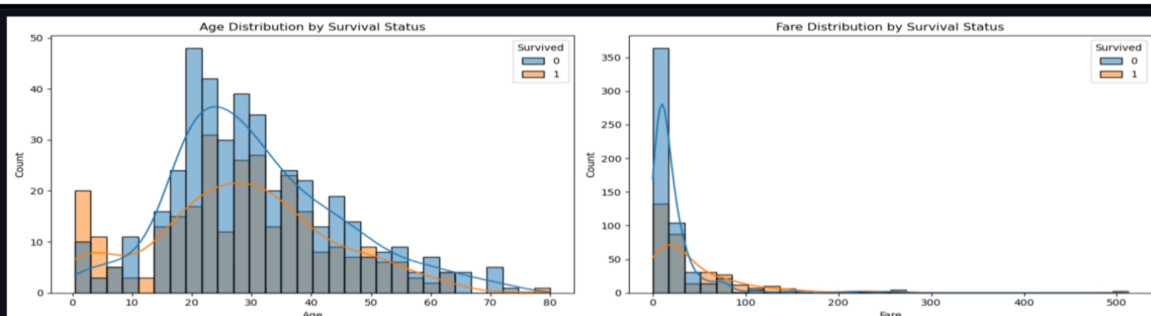
```python
# Analyze numerical features: Age and Fare
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Age distribution by survival status
sns.histplot(data=df, x='Age', hue='Survived', kde=True, bins=30, ax=axes[0])
axes[0].set_title('Age Distribution by Survival Status')

# Fare distribution by survival status
sns.histplot(data=df, x='Fare', hue='Survived', kde=True, bins=30, ax=axes[1])
axes[1].set_title('Fare Distribution by Survival Status')

plt.tight_layout()
plt.show()

# Create correlation matrix for numerical features
numeric_cols = df.select_dtypes(include=[np.number]).columns
correlation = df[numeric_cols].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Numerical Features')
plt.show()

print("Observations from correlation matrix:")
print("- Fare has a positive correlation with survival (0.26)")
print("- Pclass has a negative correlation with survival (-0.34)")
print("- Age has a slight negative correlation with survival (-0.07)")
```

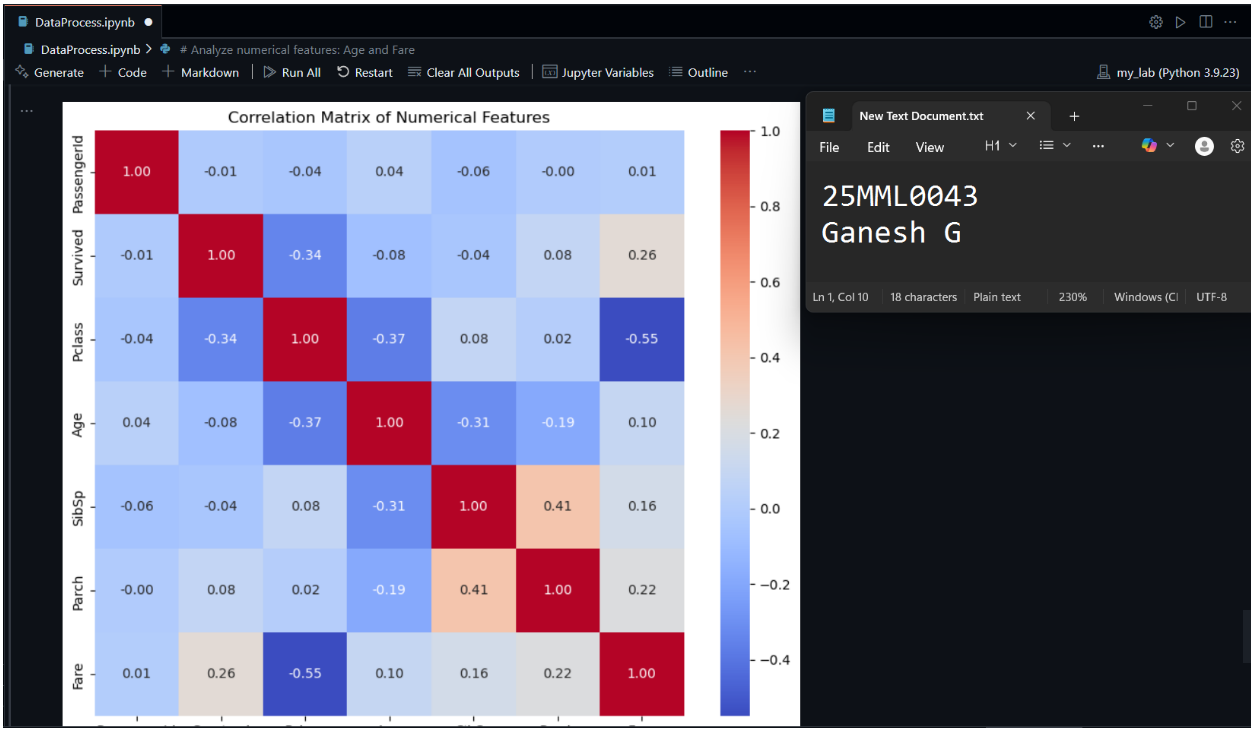
[8] ✓ 1.5s Python

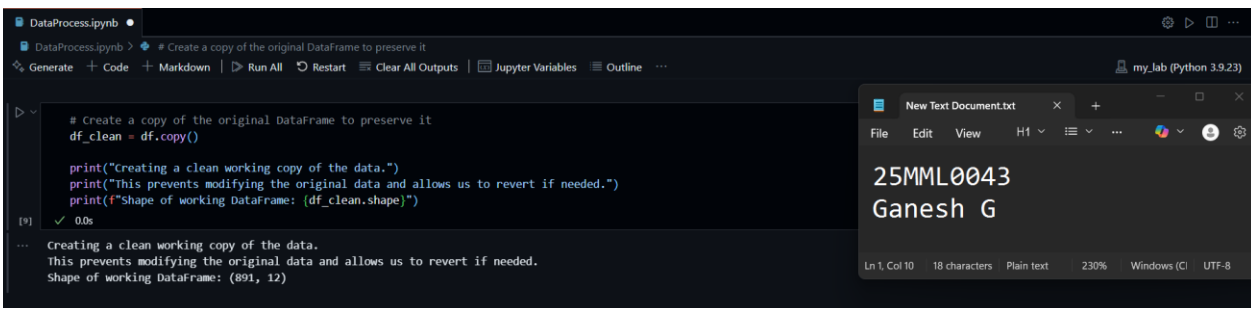
New Text Document.txt

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CR | UTF-8



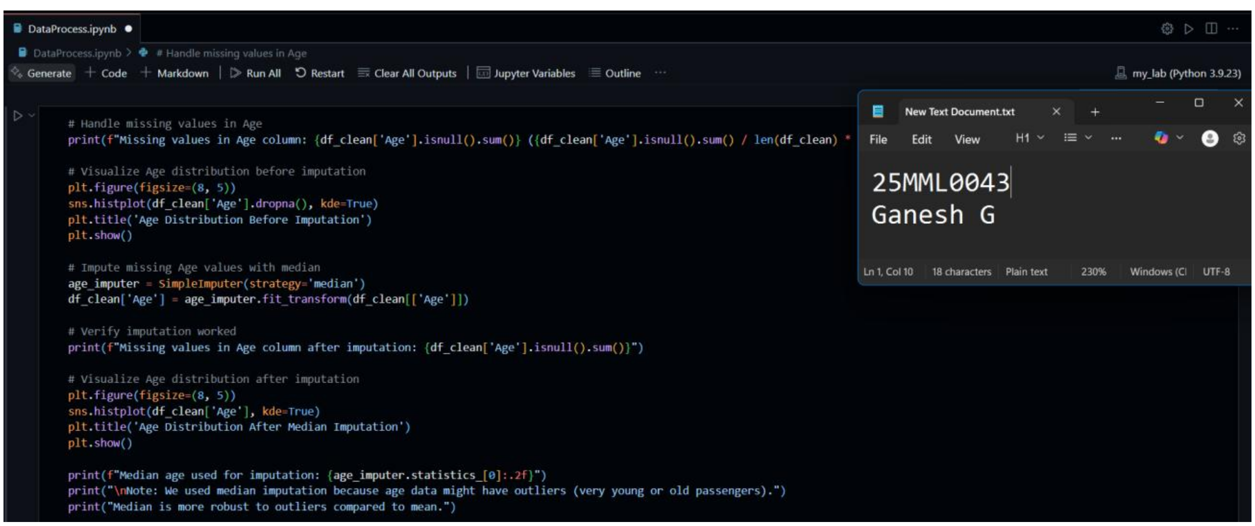





```python
# Create a copy of the original DataFrame to preserve it
df_clean = df.copy()

print("Creating a clean working copy of the data.")
print("This prevents modifying the original data and allows us to revert if needed.")
print(f"Shape of working DataFrame: {df_clean.shape}")
```

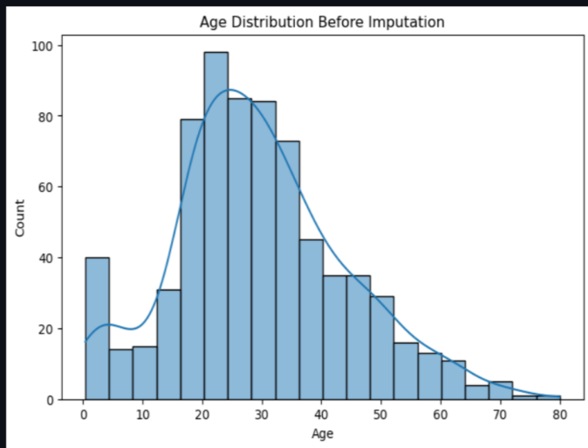
[9] ✓ 0.0s

```
Creating a clean working copy of the data.
This prevents modifying the original data and allows us to revert if needed.
Shape of working DataFrame: (891, 12)
```



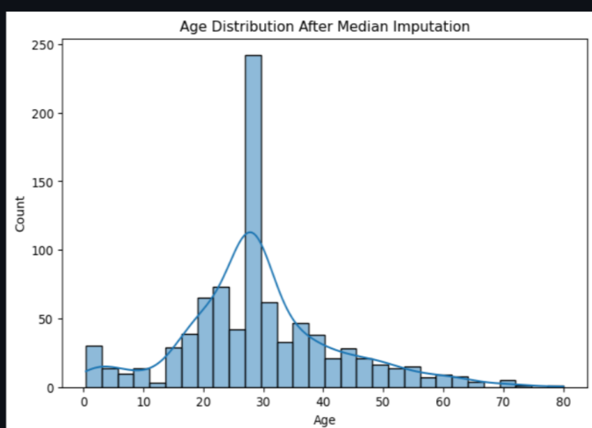

```python
# Handle missing values in Age
print(f"Missing values in Age column: {df_clean['Age'].isnull().sum()} ({df_clean['Age'].isnull().sum() / len(df_clean) *

# Visualize Age distribution before imputation
plt.figure(figsize=(8, 5))
sns.histplot(df_clean['Age'].dropna(), kde=True)
plt.title('Age Distribution Before Imputation')
plt.show()

# Impute missing Age values with median
age_imputer = SimpleImputer(strategy='median')
df_clean['Age'] = age_imputer.fit_transform(df_clean[['Age']])

# Verify imputation worked
print(f"Missing values in Age column after imputation: {df_clean['Age'].isnull().sum()}")

# Visualize Age distribution after imputation
plt.figure(figsize=(8, 5))
sns.histplot(df_clean['Age'], kde=True)
plt.title('Age Distribution After Median Imputation')
plt.show()

print(f"Median age used for imputation: {age_imputer.statistics_[0]:.2f}")
print("\nNote: We used median imputation because age data might have outliers (very young or old passengers).")
print("Median is more robust to outliers compared to mean.")
```

```
Missing values in Age column: 177 (19.87%)
```



25MML0043
Ganesh G

```
Missing values in Age column after imputation: 0
```



25MML0043
Ganesh G

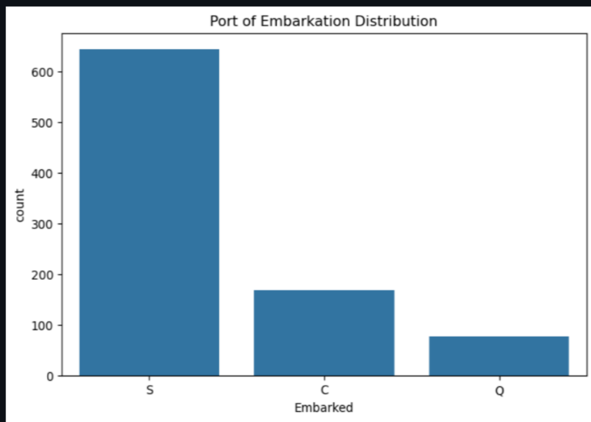

```
Median age used for imputation: 28.00

Note: We used median imputation because age data might have outliers (very young or old passengers).
Median is more robust to outliers compared to mean.
```

```python
# Handle missing values in Embarked column
print(f"Missing values in Embarked column: {df_clean['Embarked'].isnull().sum()} ({df_clean['Embarked'].isnull().sum() / len(df_clean) * 100:.2f}%)")

# Analyze Embarked distribution
plt.figure(figsize=(8, 5))
embarked_counts = df_clean['Embarked'].value_counts()
sns.countplot(x='Embarked', data=df_clean)
plt.title('Port of Embarkation Distribution')
plt.show()

print("Most common embarkation port:", df_clean['Embarked'].mode()[0])

# Impute missing Embarked values with most frequent value
embarked_imputer = SimpleImputer(strategy='most_frequent')
df_clean['Embarked'] = embarked_imputer.fit_transform(df_clean[['Embarked']]).ravel()

# Verify imputation worked
print(f"Missing values in Embarked column after imputation: {df_clean['Embarked'].isnull().sum()}")

print("\nNote: For categorical variables like 'Embarked', using the most frequent value (mode)")
print("is a common imputation strategy, especially when the missing percentage is low.")
```

✓ 0.2s Python

25MML0043
Ganesh G

```
Missing values in Embarked column: 2 (0.22%)
```




```
Most common embarkation port: S
Missing values in Embarked column after imputation: 0

Note: For categorical variables like 'Embarked', using the most frequent value (mode)
is a common imputation strategy, especially when the missing percentage is low.
```

New Text Document.txt

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CI | UTF-8

DataProcess.ipynb

DataProcess.ipynb > # Handle missing values in Cabin column

Generate | + Code | + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | my_lab (Python 3.9.23)

```python
# Handle missing values in Cabin column
print(f"Missing values in Cabin column: {df_clean['Cabin'].isnull().sum()} ({df_clean['Cabin'].isnull().sum() / len(df_clean) * 100:.2f}%)")

# Since a high percentage of Cabin values are missing, dropping the column is reasonable
df_clean.drop(columns=['Cabin'], inplace=True)

print("Cabin column dropped due to high percentage of missing values.")
print("When over 70% of values are missing, the information gain may not be worth the imputation effort.")

# Check for any remaining missing values
remaining_missing = df_clean.isnull().sum()
print("\nRemaining missing values after handling:")
print(remaining_missing[remaining_missing > 0])
```

[12] 0.0s

```
Missing values in Cabin column: 687 (77.10%)
Cabin column dropped due to high percentage of missing values.
When over 70% of values are missing, the information gain may not be worth the imputation effort.

Remaining missing values after handling:
Series([], dtype: int64)
```

New Text Document.txt

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CI | UTF-8

DataProcess.ipynb

DataProcess.ipynb > # Feature Engineering: Creating new meaningful features

Generate | + Code | + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline | my_lab (Python 3.9.23)

```python
# Feature Engineering: Creating new meaningful features

# 1. Create a Family Size feature
print("Creating new feature: 'FamilySize'")
df_clean['FamilySize'] = df_clean['SibSp'] + df_clean['Parch'] + 1  # +1 for the passenger themselves

# Analyze the new feature
plt.figure(figsize=(10, 6))
sns.countplot(x='FamilySize', hue='Survived', data=df_clean)
plt.title('Survival Rate by Family Size')
plt.xlabel('Family Size')
plt.ylabel('Count')
plt.legend(['Did not survive', 'Survived'])
plt.show()

# Calculate survival rate by family size
family_survival = df_clean.groupby('FamilySize')['Survived'].mean().sort_values(ascending=False)
print("Survival rate by family size:")
print(family_survival * 100)

print("\nObservation: Family sizes of 1-3 had higher survival rates than very large families.")
print("This suggests that small to medium-sized families had advantages during the evacuation.")
```

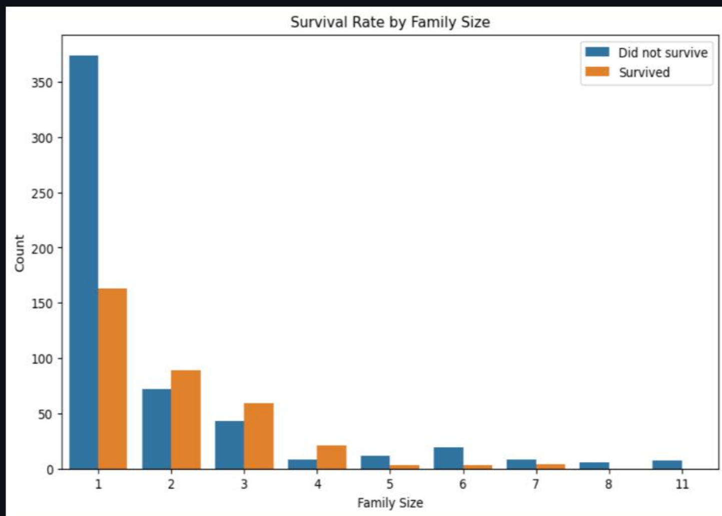
[13] 0.3s Python

New Text Document.txt

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CI | UTF-8

```
Creating new feature: 'FamilySize'
```



New Text Document.txt

File Edit View

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CRLF) | UTF-8

```
Survival rate by family size:
FamilySize
4    72.413793
```

```python
# 2. Create an IsAlone feature
print("Creating new feature: 'IsAlone'")
df_clean['IsAlone'] = (df_clean['FamilySize'] == 1).astype(int)

# Visualize survival rate by IsAlone
plt.figure(figsize=(8, 5))
sns.countplot(x='IsAlone', hue='Survived', data=df_clean)
plt.title('Survival Rate by Traveling Alone Status')
plt.xlabel('Traveling Alone (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.legend(['Did not survive', 'Survived'])
plt.show()

# Calculate survival rate by IsAlone
alone_survival = df_clean.groupby('IsAlone')['Survived'].mean()
print("Survival rate by traveling alone status:")
print(alone_survival * 100)

print("\nObservation: Passengers traveling with family had a higher survival rate than those traveling alone.")
print("This binary feature simplifies the family size information for the model.")
```

[14] ✓ 0.3s Python

File Edit View

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CRLF) | UTF-8

DataProcess.ipynb

DataProcess.ipynb > # 2. Create an IsAlone feature

Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline

```python
# 2. Create an IsAlone feature
print("Creating new feature: 'IsAlone'")
df_clean['IsAlone'] = (df_clean['FamilySize'] == 1).astype(int)

# Visualize survival rate by IsAlone
plt.figure(figsize=(8, 5))
sns.countplot(x='IsAlone', hue='Survived', data=df_clean)
plt.title('Survival Rate by Traveling Alone Status')
plt.xlabel('Traveling Alone (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.legend(['Did not survive', 'Survived'])
plt.show()

# Calculate survival rate by IsAlone
alone_survival = df_clean.groupby('IsAlone')['Survived'].mean()
print("Survival rate by traveling alone status:")
print(alone_survival * 100)

print("\nObservation: Passengers traveling with family had a higher survival rate than those traveling alone.")
print("This binary feature simplifies the family size information for the model.")
```

[14] ✓ 0.3s Python

New Text Document.txt

File Edit View

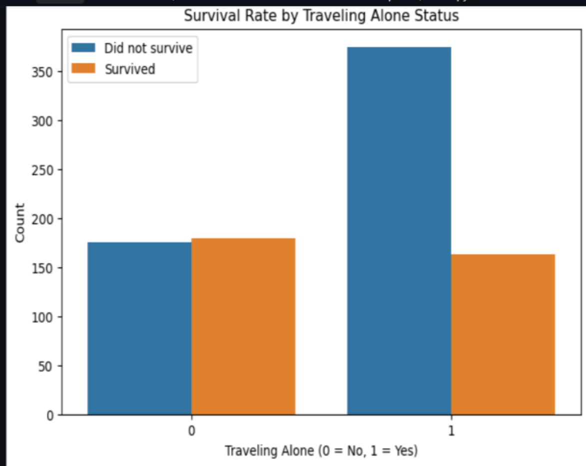
25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (CRLF) | UTF-8

DataProcess.ipynb

DataProcess.ipynb > # 2. Create an IsAlone feature

Generate | Code | Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline



New Text Document.txt

File Edit View

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (Cl | UTF-8

```
Survival rate by traveling alone status:
IsAlone
0    50.564972
1    30.353818
Name: Survived, dtype: float64

Observation: Passengers traveling with family had a higher survival rate than those traveling alone.
This binary feature simplifies the family size information for the model.
```

DataProcess.ipynb

DataProcess.ipynb > # 3. Extract titles from passenger names

Generate | Code | Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | Outline

New Text Document.txt

File Edit View

25MML0043
Ganesh G

Ln 1, Col 10 | 18 characters | Plain text | 230% | Windows (Cl | UTF-8

```python
# 3. Extract titles from passenger names
print("Extracting titles from passenger names...")
df_clean['Title'] = df_clean['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

# Display the different titles and their frequencies
title_counts = df_clean['Title'].value_counts()
print("Titles extracted from names:")
print(title_counts)

# Group rare titles to prevent overfitting
print("\nGrouping rare titles...")
title_mapping = {
    "Mr": "Mr",
    "Miss": "Miss",
    "Mrs": "Mrs",
    "Master": "Master",
    "Dr": "Rare",
    "Rev": "Rare",
    "Col": "Rare",
    "Major": "Rare",
    "Mlle": "Miss",  # Mademoiselle in French
    "Mme": "Mrs",    # Madame in French
    "Ms": "Miss",
    "Lady": "Rare",
    "Sir": "Rare",
    "Capt": "Rare",
    "Countess": "Rare",
    "Jonkheer": "Rare",
    "Don": "Rare",
    "Dona": "Rare"
}
df_clean['Title'] = df_clean['Title'].map(title_mapping)
```

```
Extracting titles from passenger names...
Titles extracted from names:
Title
Mr          517
Miss        182
Mrs         125
Master       40
Dr            7
Rev           6
Mlle          2
Major         2
Col           2
Countess      1
Capt          1
Ms            1
Sir           1
Lady          1
Mme           1
Don           1
Jonkheer      1
Name: count, dtype: int64

Grouping rare titles...

Titles after grouping:
...
Mrs       126
Master     40
Rare       23
Name: count, dtype: int64
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

<module 'matplotlib.pyplot' from 'c:\\Users\\Ganesh\\anaconda3\\envs\\my_lab\\lib\\site-packages\\matplotlib\\pyplot.py'>
```

25MML0043
Ganesh G

Encode Categorical Variables:

```python
from sklearn.preprocessing import OneHotEncoder

# Encode 'Sex' and 'Embarked' using one-hot encoding
encoder = OneHotEncoder(sparse_output=False, drop='first')
encoded_sex_embarked = encoder.fit_transform(df[['Sex', 'Embarked']])
encoded_df = pd.DataFrame(
    encoded_sex_embarked,
    columns=encoder.get_feature_names_out(['Sex', 'Embarked'])
)

# Remove the original columns and add the encoded columns
df.drop(columns=['Sex', 'Embarked'], inplace=True)
df = pd.concat([df, encoded_df], axis=1)
```

[16] 0.0s

25MML0043
Ganesh G

Scale Numerical Features:

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
numerical_features = ['Age', 'Fare']
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

[17] 0.0s

25MML0043
Ganesh G

Select Features and Target Variable:

```python
features = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex_male', 'Embarked_Q', 'Embarked_S']
target = 'Survived'

X = df[features]
y = df[target]
```

[18] 0.0s

25MML0043
Ganesh G

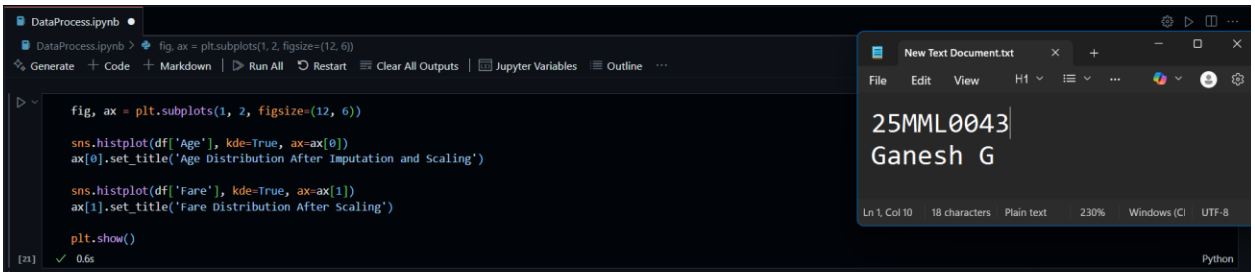
Split the Dataset

```python
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, random_state=42)
```

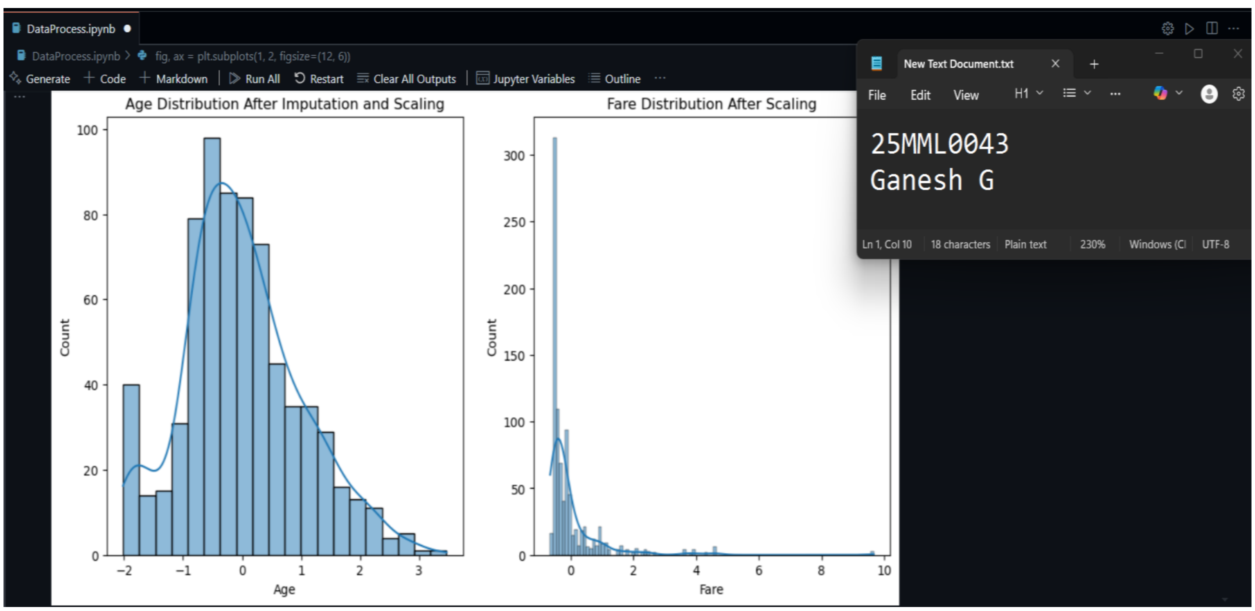
```python
print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

```
Training set shape: (712, 8)
Testing set shape: (179, 8)
```


Visualize the Distribution of Numerical Features:



```python
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

sns.histplot(df['Age'], kde=True, ax=ax[0])
ax[0].set_title('Age Distribution After Imputation and Scaling')

sns.histplot(df['Fare'], kde=True, ax=ax[1])
ax[1].set_title('Fare Distribution After Scaling')

plt.show()
```

LAB – 2

AIM:

To implement logistic regression for binary classification using the MNIST dataset, and train a model to classify whether a given handwritten digit image is a '0' or a '1'.

INTRODUCTION:

Logistic Regression is a widely used algorithm for binary classification tasks. It models the probability that a given input belongs to a particular class. In this lab, we'll use logistic regression to classify handwritten digits from the MNIST dataset — specifically to distinguish between the digits **'0'** and **'1'**. The MNIST dataset contains thousands of 28x28 pixel grayscale images of handwritten digits from 0 to 9. By focusing on just two classes, we can understand how logistic regression works in a straightforward binary setting. This lab will walk through loading the dataset, preprocessing the data, training the model, and evaluating its performance using accuracy and confusion matrices.

REAL WORLD EXAMPLE:

Security Access System (Known vs Unknown Faces):

1. Image Input: A webcam or CCTV captures a **grayscale or color face image** of the person at the entrance.
2. Preprocessing: Resize all face images to the same size (e.g., 64×64).
3. Labelling Data: Known as (1) and Unknown as (0)
4. Model Training with Logistic Regression
5. Prediction & Real-Time Use: At the door, the system captures and preprocesses a face image, flattens it into a vector, and feeds it into a logistic regression model. If the prediction is 1, access is granted; if 0, a security alert is triggered.

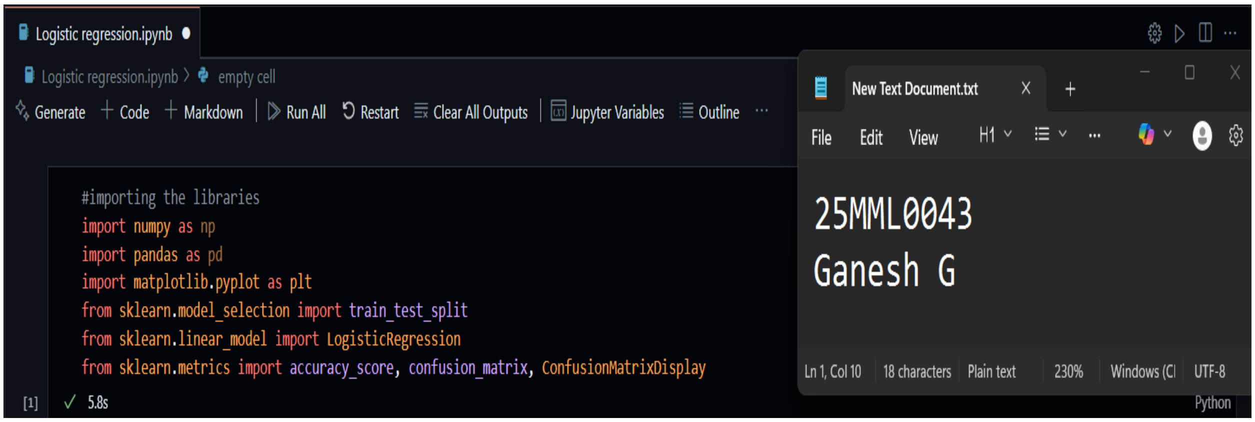
ALGORITHM:

1. Begin by importing the necessary libraries such as NumPy, pandas, matplotlib, and modules from scikit-learn.
2. Load the MNIST dataset using the fetch_openml() function from sklearn.datasets.
3. Filter the dataset to include only the images labeled as '0' and '1', since this is a binary classification task.
4. Normalize the pixel values of the images by dividing by 255 to bring all values into the range [0, 1].
5. Split the dataset into training and testing sets using train_test_split() to prepare for model training and evaluation.
6. Initialize the logistic regression model using LogisticRegression() from sklearn.linear_model.
7. Train the model by fitting it to the training data using the .fit() method.

8. Predict the labels for the test data using the .predict() method.
9. Evaluate the model's performance using metrics such as accuracy score and the confusion matrix.
10. Visualize the confusion matrix and some sample predictions to better understand how the model is performing.

IMPLEMENTATION AND RESULTS:

Import Libraries



Load and preprocess data

Data Normalization and Training

25MML0043
Ganesh G

```python
#Data normalization - any image that should cross 255 pixel
X = X/255.0
```
[4] ✓ 0.0s

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
#or
#X_train, X_test, y_train, y_test = train_test_split(x, y, train_size=0.8, random_state=42)
```
[5] ✓ 0.2s Python

```python
#Training the logistic regression model for 1000 times
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```
[6] ✓ 0.4s Python

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```python
#Making the predictions and storing in y(dependent variable)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```
[7] ✓ 0.0s Python

Making the predictions

Logistic regression.ipynb > #Making the predictions and storing in y(dependent variable)

New Text Document.txt

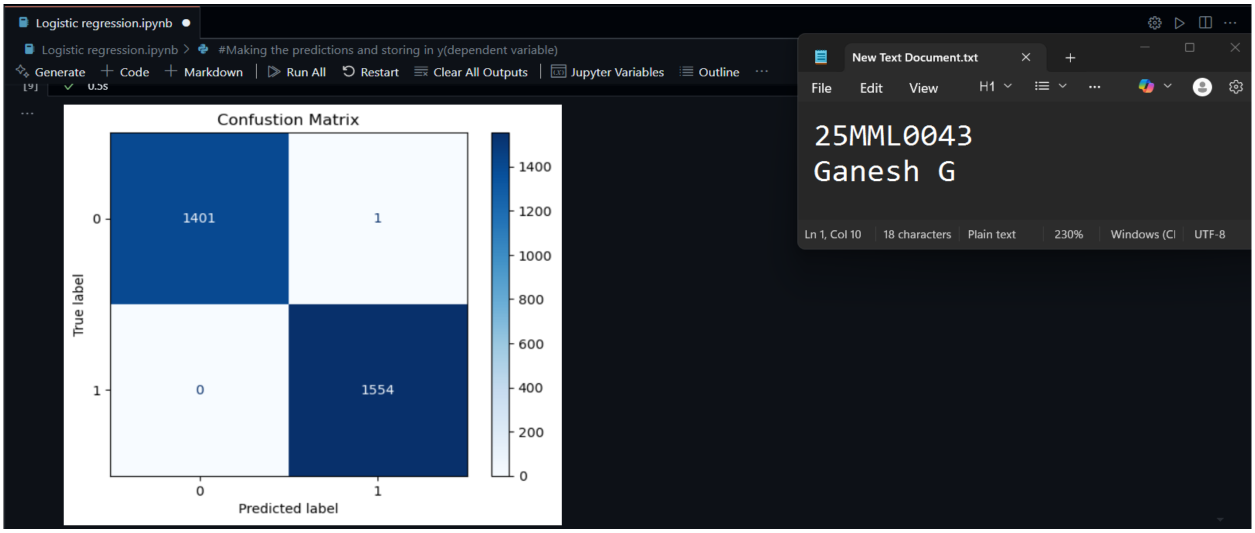
25MML0043
Ganesh G

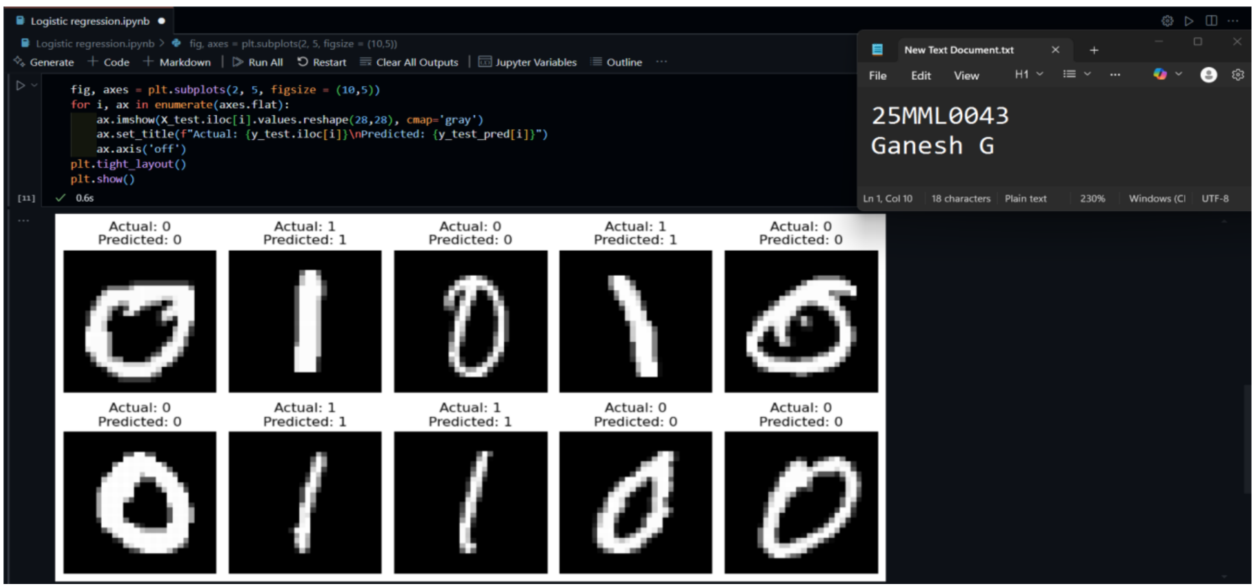
```python
#Making the predictions and storing in y(dependent variable)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```
[7] ✓ 0.0s

```python
#printing the training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Training accracy: ", train_accuracy*100)
print("Testing accuracy: ", test_accuracy*100)
```
[8] ✓ 0.0s Python

```
Training accracy:  99.99154262516915
Testing accuracy:  99.96617050067658
```

```python
#confustion matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)
ConfusionMatrixDisplay(conf_matrix).plot(cmap='Blues')
plt.title("Confustion Matrix")
plt.show()
```
[9] ✓ 0.5s Python


Logistic regression.ipynb
#Making the predictions and storing in y(dependent variable)
Generate
Code
Markdown
Run All
Restart
Clear All Outputs
Jupyter Variables
Outline
Confustion Matrix
1401
1
0
1554
True label
Predicted label
1400
1200
1000
800
600
400
200
0
New Text Document.txt
File
Edit
View
25MML0043
Ganesh G
Ln 1, Col 10
18 characters
Plain text
230%
UTF-8


Logistic regression.ipynb
fig, axes = plt.subplots(2, 5, figsize = (10,5))
Generate
Code
Markdown
Run All
Restart
Clear All Outputs
Jupyter Variables
Outline
fig, axes = plt.subplots(2, 5, figsize = (10,5))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test.iloc[i].values.reshape(28,28), cmap='gray')
    ax.set_title(f"Actual: {y_test.iloc[i]}\nPredicted: {y_test_pred[i]}")
    ax.axis('off')
plt.tight_layout()
plt.show()
0.6s
Actual: 0 Predicted: 0
Actual: 1 Predicted: 1
Actual: 0 Predicted: 0
Actual: 1 Predicted: 1
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 1 Predicted: 1
Actual: 1 Predicted: 1
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
New Text Document.txt
File
Edit
View
25MML0043
Ganesh G
Ln 1, Col 10
18 characters
Plain text
230%
UTF-8

LAB – 3

AIM:

Predicting gold prices using Linear Regression Model involves data collection, preprocessing, model training, and evaluation. Below is a detailed example using Python and Scikit-learn to predict gold prices.

INTRODUCTION:

Gold price prediction is an important task in financial analytics, where historical data is used to forecast future prices. Linear regression is a simple yet powerful algorithm that models the relationship between input variables (such as oil price, exchange rates, stock indices, etc.) and a continuous target variable (gold price). In this experiment, we will use historical gold price data, apply preprocessing, and train a linear regression model to make predictions.

REAL WORLD EXAMPLE:

Predicting Used Car Prices Using Linear Regression:

1. Data Collection: The company collects a dataset of thousands of used car listings from their platform
2. Data Preprocessing
3. Train Linear Regression Model
4. Prediction Example: Suppose a seller enters the following car info
5. Model Evaluation: Mean Absolute Error (MAE), Mean Squared Error (MSE)

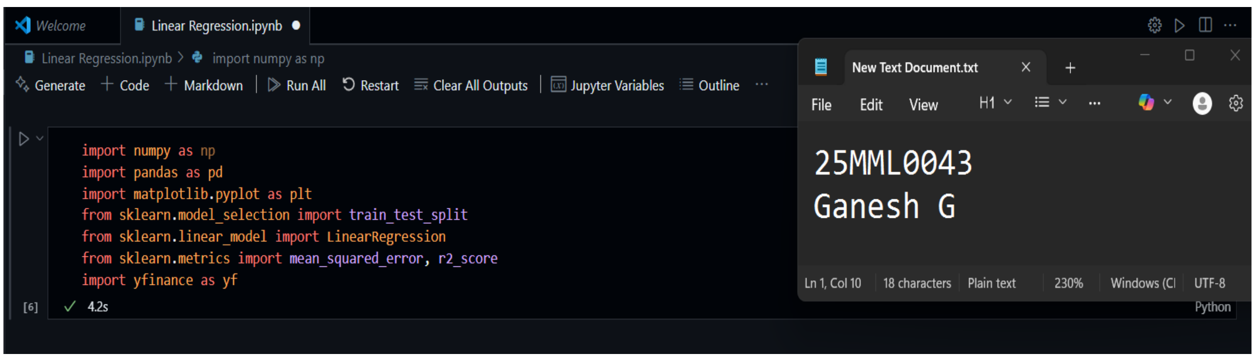
ALGORITHM:

1. Begin by importing the necessary libraries including pandas, NumPy, matplotlib, seaborn, and Scikit-learn modules for regression and evaluation.
2. Load the historical gold price dataset into a DataFrame.
3. Convert the date column to datetime format and set it as the index for time-series structure (if applicable).
4. Explore and visualize the data to understand trends and detect any missing values or anomalies.
5. Create lag features, such as using the previous day's gold price to predict the current day's price.
6. Drop rows with missing values caused by lagging.
7. Define the input features (X) and target variable (y) for the regression model.

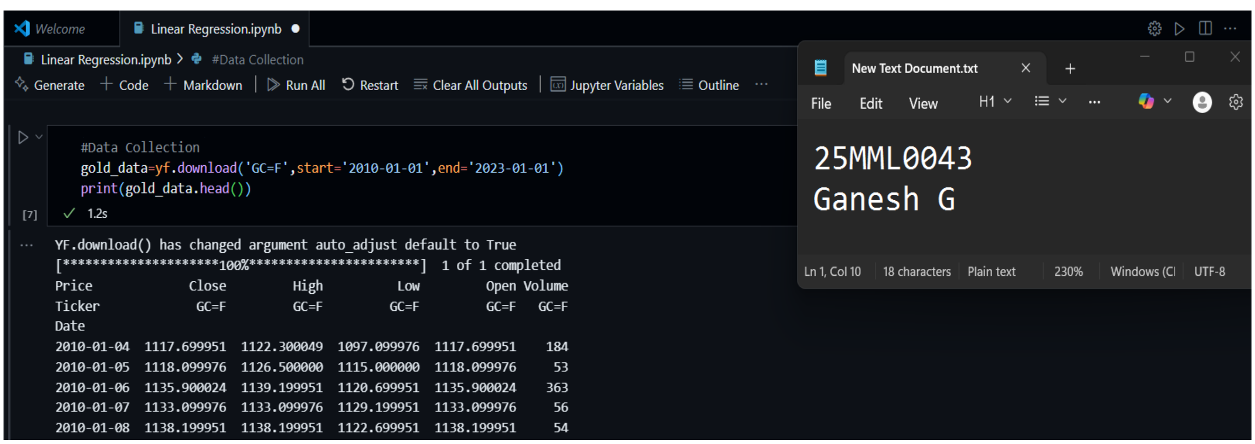
8. Split the dataset into training and testing sets using train_test_split() to evaluate model performance.
9. Initialize and train the LinearRegression model using the training data.
10. Use the trained model to predict gold prices on the test dataset.
11. Evaluate the model using metrics such as Mean Squared Error (MSE) and R-squared (R^2) score.
12. Plot and compare the actual and predicted prices to visually assess model performance.

<u>IMPLEMENTATION AND RESULTS:</u>

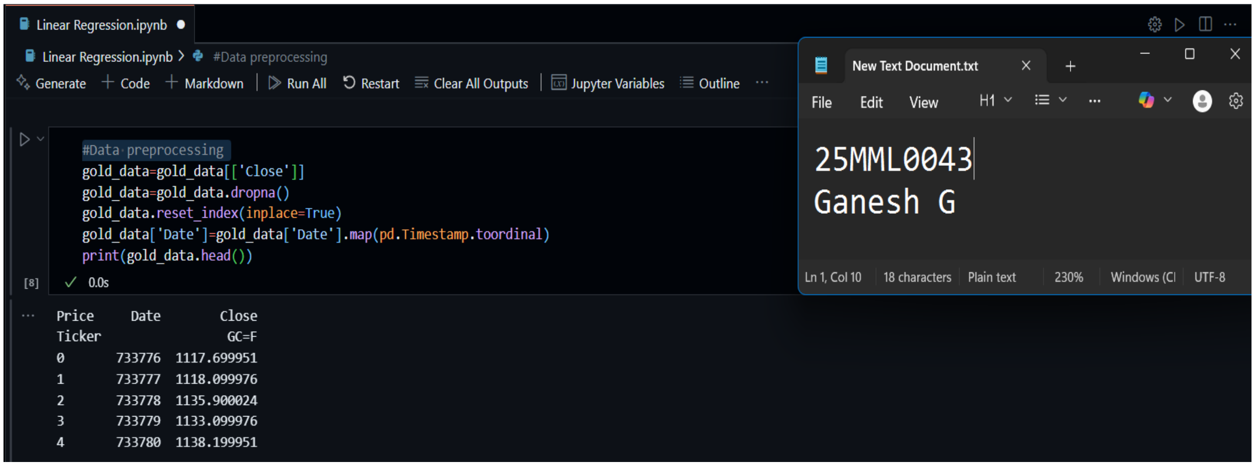
Import Libraries



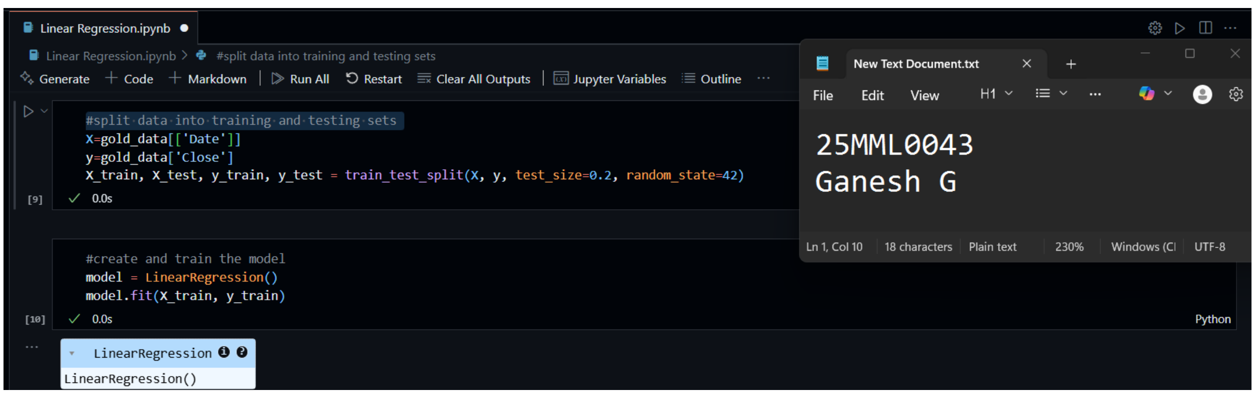
Data Collection



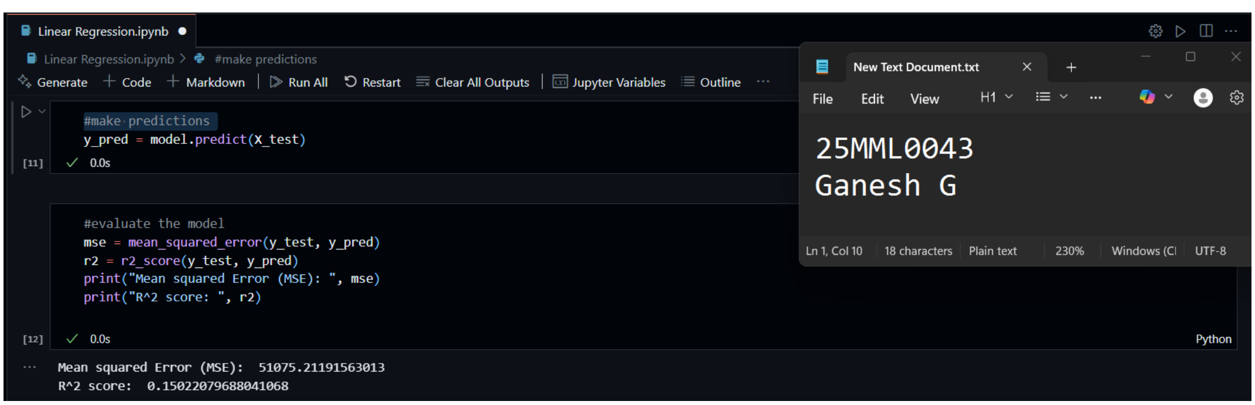
Data preprocessing



split data into training and testing sets



make predictions



visualize the results