

# **EXPLANATORY DOCUMENT**

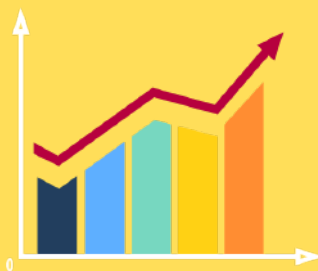
## **FEATURE ENGINEERING AND STORAGE FOR MACHINE LEARNING USING SNOWFLAKE**

**SUHANI N GAONKAR**

**22BDS0273**

**VIT, VELLORE**

**AI/ML INTERN CANDIDATE**



# FEATURE ENGINEERING



Feature engineering is the process of selecting, modifying, and creating variables (features) from raw data that make machine learning models work better. In other words, it is how we turn messy, real-world data into meaningful inputs that a model can actually learn from.

## IMPORTANCE OF FEATURE ENGINEERING

- **Machine learning models are only as good as the data they are trained on.**
- **Well-designed features improve model accuracy, reduce noise, and help models generalize better.**
- **In real projects, feature engineering often matters more than the choice of algorithm**

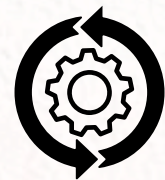
## COMMON TECHNIQUES

- **Normalization/Scaling:** Adjusting numerical values (e.g., age, salary) to a standard range so one feature doesn't dominate others.
- **Encoding Categorical Variables:** Converting text fields (e.g., gender, location) into numeric form for models.
- **Binning:** Grouping continuous variables into ranges (e.g., age groups: child, adult, senior).

- **Aggregations:** Summarizing data over time or across groups (e.g., average purchase per customer).
- **Time-based Features:** Using timestamps to create new signals (e.g., day of week, time since last purchase).

# USING SNOWFLAKE FOR DATA STORAGE & PROCESSING

## SNOWFLAKE AS A DATA PLATFORM



- Snowflake is a cloud-based data warehouse that can store and process both structured data (tables, rows, columns) and semi-structured data (JSON, Parquet, Avro).
- It separates storage and compute, which means you can scale them independently and pay only for what you use.
- Ideal for enterprise settings where multiple teams need reliable, centralized data.

## STORAGE AND COMPUTATION OF DATA IN SNOWFLAKE

- **Storage Layer:** Snowflake compresses and encrypts all data automatically. Tables can store structured rows and semi-structured JSON blobs.
- **Compute Layer (Warehouses):** Scalable virtual clusters that run your SQL queries. You can spin them up/down automatically, so you only pay for what you use.
- **Services Layer:** Handles security, optimization, metadata, and role-based access control.

# EXAMPLE SQL IN SNOWFLAKE

```
-- Step 1: Extract raw passenger data with filters
SELECT PASSENGERID, SEX, AGE, FARE
FROM TITANIC_DATASET
WHERE AGE IS NOT NULL AND FARE > 0;

-- Step 2: Create a cleaned version of the dataset
CREATE OR REPLACE TABLE TITANIC_CLEANED AS
SELECT
    PASSENGERID,
    COALESCE(AGE, 28) AS AGE,      -- Fill missing ages with median = 28
    COALESCE(FARE, 14.5) AS FARE,  -- Fill missing fares with median = 14.5
    CASE WHEN SEX = 'male' THEN 1 ELSE 0 END AS SEX_ENCODED
FROM TITANIC_DATASET;
```

- Extracts four key columns — PassengerID, Sex, Age, and Fare from the Titanic dataset.
- Filters the data to keep only passengers with a valid Age and a Fare greater than 0.
- Creates a new cleaned table named TITANIC\_CLEANED for the processed data.
- Fills missing Age values with 28, which is the median age of passengers.
- Fills missing Fare values with 14.5, which is the median fare.
- Converts Sex into numeric format: male = 1, female = 0, making it suitable for ML models.



# HOW SNOWFLAKE INTEGRATES WITH ML PIPELINES



## Snowpark ML

- A built in Snowflake library that allows you to train and run models directly inside Snowflake, without exporting data.

## External Integrations

- **Databricks** : Can read Snowflake tables and apply Spark ML for large-scale machine learning.
- **AWS SageMaker** : Can fetch features from Snowflake using connectors for training and deployment of ML models.

## Why This Matters

- Ensures the same feature transformations (like filling missing values or encoding categories) are used consistently across both training and inference.
- Reduces errors, avoids data leakage, and improves collaboration between data and ML teams.

# FEATURE STORE

- **A feature store is like a library of machine learning features. Instead of every data scientist cleaning and engineering data separately, all features are stored in one central place.**
- **It ensures that when we train a model and later use that model for predictions, we are using the exact same definitions of features.**
- **Think of it as the bridge between raw data in databases and ML models in production**

## NEED OF FEATURE STORE

### **Consistency**

Without a feature store, the same feature (like Age Group) might be calculated differently during training and inference. A feature store ensures uniform definitions.

### **Reusability**

Once a feature such as Family Size is created, it can be reused across multiple models and teams without duplicating work.

### **Governance**

Makes it easy to track where a feature came from, who created it, and which models are using it. This is important for enterprise-level projects.

### **Efficiency**

Saves time by centralizing work. Data engineers build features once, and data scientists can focus on modeling instead of repeating the same cleaning steps.

# COMPARISION OF VARIOUS FEATURE STORES



Feature Store	Best For	Storage & Serving	Strengths	Limitations
Snowflake Feature Store	Teams already using Snowflake	Features stored in Snowflake tables; batch reads (near real-time possible with caching)	Simple SQL based workflows, strong governance, time-travel support	Not a native online (real-time) store
AWS SageMaker Feature Store	AWS-native ML pipelines	Online store (DynamoDB) + Offline store (S3)	Tight integration with SageMaker, real-time serving, fully managed	Locked into AWS ecosystem
Databricks Feature Store	Spark and MLflow users	Delta Lake (batch/stream) + MLflow integration	Handles large-scale data, streaming features, experiment tracking	Requires Spark/Databricks environment, online serving less direct





**Load raw Titanic data into Snowflake.**



**Explore and clean data**



**Perform feature engineering**



**Store processed features into Feature Store.**



**Retrieve features into Snowpark Notebook.**



**Train Logistic Regression model.**



**Evaluate accuracy and predictions.**



**Visualize results**



# IMPLEMENTATION OF FEATURE ENGINEERING WITH SNOWFLAKE AND FEATURE STORE



## Dataset Used

The dataset used is the Titanic passenger dataset. It contains both numeric and categorical columns that are useful for predicting survival.

- **ID and Label:** PASSENGERID (unique identifier), SURVIVED (0 = did not survive, 1 = survived)
- **Demographics:** SEX, AGE
- **Socio-economic:** PCLASS (passenger class), FARE, CABIN
- **Family Information:** SIBSP (number of siblings/spouses aboard), PARCH (number of parents/children aboard)
- **Other:** EMBARKED (boarding port), NAME, TICKET

## ENVIRONMENT SETUP

My Workspace > titanic.sql

```
1      -- Create database
2      CREATE OR REPLACE DATABASE TITANIC_DB;
3
4      -- Use the database
5      USE DATABASE TITANIC_DB;
6
7      -- Create schema
8      CREATE OR REPLACE SCHEMA PUBLIC;
9
10     -- Create warehouse (compute resource)
11     CREATE OR REPLACE WAREHOUSE TITANIC_WH
12         WITH WAREHOUSE_SIZE='SMALL'
13         AUTO_SUSPEND=60
14         AUTO_RESUME=TRUE;
15
16     -- Activate warehouse
17     USE WAREHOUSE TITANIC_WH;
```

# EXTRACT: FETCH RAW DATA

```
-- Peek at the first 10 rows
SELECT * FROM TITANIC_DATASET LIMIT 10;
```

Table Chart

10 rows 170ms

PASSENGERID	SURVIVED	PCLASS	NAME	SEX	AGE	SIBSP	PARCH	TICKET	FARE	CABIN	EMBARKED
1	0	3	Allen, Mr. William Henry	male	35.00	0	0	373450	8.0500	null	S
2	1	1	Cummings, Mrs. John Bradley	female	38.00	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.00	0	0	STON/O2. 310	7.9250	null	S
4	1	1	Futrelle, Mrs. Jacques Heath	female	35.00	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.00	0	0	373450	8.0500	null	S
6	0	3	Moran, Mr. James	male	null	0	0	330877	8.4583	null	Q
7	0	1	McCarthy, Mr. Timothy J	male	54.00	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leon	male	2.00	3	1	349909	21.0750	null	S
9	1	3	Johnson, Mrs. Oscar W (Elis)	female	27.00	0	2	347742	11.1333	null	S
10	1	2	Nasser, Mrs. Nicholas (Adele)	female	14.00	1	0	237736	30.0708	null	C

The Extract step means pulling raw data from Snowflake for the first time to inspect it. In this case, we are previewing the Titanic dataset to check its structure, columns, and sample values before moving on to profiling and transformation.

- Used to preview the first 10 rows of the Titanic dataset.
- Helps confirm the available columns like PASSENGERID, SURVIVED, PCLASS, SEX, AGE, FARE, etc.
- Lets us quickly check the data format and values before cleaning.



# DATA EXPLORATION

```
33  -- Peek at the first 10 rows
34  SELECT * FROM TITANIC_DATASET LIMIT 10;
35
36  -- Count missing values
37  SELECT
38      SUM(CASE WHEN AGE IS NULL THEN 1 ELSE 0 END) AS missing_age,
39      SUM(CASE WHEN FARE IS NULL THEN 1 ELSE 0 END) AS missing_fare,
40      SUM(CASE WHEN CABIN IS NULL THEN 1 ELSE 0 END) AS missing_cabin,
41      SUM(CASE WHEN EMBARKED IS NULL THEN 1 ELSE 0 END) AS missing_embarked,
42      SUM(CASE WHEN SEX IS NULL THEN 1 ELSE 0 END) AS missing_sex
43  FROM TITANIC_DATASET;
44
45  -- Quick stats for AGE and FARE
46  SELECT
47      MEDIAN(AGE) AS median_age,
48      AVG(AGE) AS avg_age,
49      STDDEV(AGE) AS sd_age,
50      MEDIAN(FARE) AS median_fare,
51      AVG(FARE) AS avg_fare,
52      STDDEV(FARE) AS sd_fare
53  FROM TITANIC_DATASET;
54
55  -- Most common embarkation port
56  SELECT EMBARKED, COUNT(*) AS cnt
57  FROM TITANIC_DATASET
58  GROUP BY EMBARKED
59  ORDER BY cnt DESC
60  LIMIT 3;
61
```

## Count missing values

Checks how many entries are missing in important columns: AGE, FARE, CABIN, EMBARKED, and SEX. This helps identify which features need imputation or special handling.

## Quick stats for AGE and FARE

Calculates median, average, and standard deviation for AGE and FARE. The median values (around 28 for age and 14.5 for fare) are later used to fill missing data.

## Most common embarkation port

Finds which embarkation port occurs most often. The most frequent port (usually “S”) is used to replace missing embarkation values.

# FEATURE ENGINEERING

```
CREATE OR REPLACE TABLE TITANIC_FEATURES AS
SELECT
    PASSENGERID,
    SURVIVED,
    -- Encode Sex
    CASE
        WHEN SEX = 'male' THEN 1
        WHEN SEX = 'female' THEN 0
        ELSE NULL
    END AS SEX_ENCODED,
    -- Passenger class
    PCLASS,
    -- Age (imputed)
    COALESCE(AGE, 28) AS AGE,
    CASE
        WHEN COALESCE(AGE, 28) < 18 THEN 'child'
        WHEN COALESCE(AGE, 28) BETWEEN 18 AND 35 THEN 'young_adult'
        WHEN COALESCE(AGE, 28) BETWEEN 36 AND 55 THEN 'adult'
        ELSE 'senior'
    END AS AGE_GROUP,
    -- Family size
    (SIBSP + PARCH) AS FAMILY_SIZE,
    CASE
        WHEN (SIBSP + PARCH) = 0 THEN 'alone'
        WHEN (SIBSP + PARCH) <= 3 THEN 'small_family'
        ELSE 'large_family'
    END AS FAMILY_CATEGORY,
    -- Title extraction
    COALESCE(REGEXP_SUBSTR(NAME, '\\w+\\.\\.'), 'Unknown') AS TITLE,
    -- Fare per person (imputed)
    COALESCE(FARE, 14.5) / NULLIF((SIBSP + PARCH + 1), 0) AS FARE_PER_PERSON,
    -- Embarked (imputed)
    COALESCE(EMBARKED, 'S') AS EMBARKED,
    -- Cabin binary indicator
    CASE WHEN CABIN IS NULL THEN 0 ELSE 1 END AS HAS_CABIN
FROM TITANIC_DATASET;
-- Preview engineered features
SELECT * FROM TITANIC_FEATURES LIMIT 10;
```

- Creates a new table called TITANIC\_FEATURES with engineered and cleaned features.
- Age: missing values filled with 28 (imputation) and grouped into child, young adult, adult, and senior (binning).
- Fare: missing values filled with 14.5 (imputation) and converted into fare per person (scaling/normalization).
- Embarked: missing values filled with “S” (imputation with mode).
- Sex: converted into numbers, male as 1 and female as 0 (encoding categorical variable).



- Family features: SIBSP + PARCH gives FAMILY\_SIZE (feature creation) and FAMILY\_CATEGORY labels it as alone, small family, or large family (categorical grouping/binning).
- Title: extracted from NAME (e.g., Mr., Mrs., Miss) (text parsing / feature extraction).
- Cabin: converted into a binary flag, 1 if cabin info is present, 0 if missing (binary indicator creation).
- Finally, previews 10 rows and checks distinct categories to validate the new features (data validation).

Chart

10 rows 427ms

PASSENGERID	# SURVIVED	# SEX_ENCODED	# PCLASS	# AGE	AGE_GROUP	# FAMILY_SIZE	FAMILY_CATEGORY	TITLE	FARE_PER_PERSON	EMBARKE	HAS_CABIN
1	0	1	3	22.00	young_adult	1	small_family	Mr.	3.6250000000	S	0
2	1	0	1	38.00	adult	1	small_family	Mrs.	35.6416500000	C	1
3	1	0	3	26.00	young_adult	0	alone	Miss.	7.9250000000	S	0
4	1	0	1	35.00	young_adult	1	small_family	Mrs.	26.5500000000	S	1
5	0	1	3	35.00	young_adult	0	alone	Mr.	8.0500000000	S	0
6	0	1	3	28.00	young_adult	0	alone	Mr.	8.4583000000	Q	0
7	0	1	1	54.00	adult	0	alone	Mr.	51.8625000000	S	1
8	0	1	3	2.00	child	4	large_family	Master.	4.2150000000	S	0
9	1	0	3	27.00	young_adult	2	small_family	Mrs.	3.7111000000	S	0
10	1	0	2	14.00	child	1	small_family	Mrs.	15.0354000000	C	0

## STORING IN SNOWFLAKE FEATURE STORE

```
-- Create schema for feature store
CREATE OR REPLACE SCHEMA FEATURE_STORE;

-- Save engineered table
CREATE OR REPLACE TABLE FEATURE_STORE.TITANIC_FEATURES_FINAL AS
SELECT * FROM TITANIC_FEATURES;

COMMENT ON TABLE FEATURE_STORE.TITANIC_FEATURES_FINAL IS
'Engineered Titanic dataset with missing data handled, ready for ML pipelines';

-- | Confirm table creation
SHOW TABLES IN SCHEMA FEATURE_STORE;

-- Preview stored features
SELECT * FROM FEATURE_STORE.TITANIC_FEATURES_FINAL LIMIT 10;
```

- Creates a schema `FEATURE_STORE` to store ML-ready features separately from raw data.
- Saves the engineered dataset as `TITANIC_FEATURES_FINAL`, the final cleaned version for modeling.
- Adds a comment describing the table as missing-data handled and ready for ML pipelines.
- Confirms successful creation with `SHOW TABLES`.
- Previews the first 10 rows to verify that engineered features are stored correctly.
- Makes Snowflake function as a Feature Store, ensuring consistency and reusability of features.

## MODEL TRAINING

# USING SNOWFLAKE NOTEBOOK FOR MODEL TRAINING (SNOWPARK INTEGRATED)

Python ▾ as cell1

```
1 from snowflake.snowpark import Session
2 import pandas as pd
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.model_selection import train_test_split
5 from sklearn.metrics import accuracy_score
```

Python ▾ as cell4

0.4s ▶ ⌵ ⋮

```
1 from snowflake.snowpark import Session
2 import pandas as pd
3
4 # Use default session injected by Snowflake Notebook
5 session = Session.builder.getOrCreate()
6
7 # Load the engineered Titanic features
8 df = session.table("FEATURE_STORE.TITANIC_FEATURES_FINAL").to_pandas()
9
10 print("Data loaded from Feature Store")
11 print(df.head())
12
```

Data loaded from Feature Store

	PASSENGERID	SURVIVED	SEX_ENCODED	...	FARE_PER_PERSON	EMBARKED	HAS_CABIN
0	1	0	1	...	3.62500	S	0
1	2	1	0	...	35.64165	C	1
2	3	1	0	...	7.92500	S	0
3	4	1	0	...	26.55000	S	1
4	5	0	1	...	8.05000	S	0

[5 rows x 12 columns]


+ Python + SQL + Markdown

cell1  
cell4  
cell2  
cell3  
cell5  
cell6  
cell7  
cell8




- The code starts by importing required libraries like Snowpark Session for connecting to Snowflake and pandas for handling data.
- A Snowpark session is created using `Session.builder.getOrCreate` which automatically uses the active Snowflake connection in the notebook.
- The engineered Titanic features are stored in the Feature Store table called `FEATURE_STORE.TITANIC_FEATURES_FINAL`.
- The table is accessed using `session.table` and then converted into a pandas DataFrame with `.to_pandas` for machine learning tasks.
- A preview of the data is shown with `df.head`, which confirms that important columns such as `PASSENGERID`, `SURVIVED`, `SEX_ENCODED`, `AGE`, `FARE_PER_PERSON`, and `HAS_CABIN` are available.
- This process demonstrates how features can be extracted from the Feature Store and made ready for model training and evaluation directly inside the Snowflake notebook.

## SELECTING FEATURES, LABELS, AND SPLITTING DATA

Python  as cell2

```
1
2 # 2. Select Features & Labels
3
4 feature_cols = ["SEX_ENCODED", "PCLASS", "AGE", "FAMILY_SIZE", "FARE_PER_PERSON", "HAS_CABIN"]
5
6 # Define target label column
7 label_col = "SURVIVED"
8
9 # Separate features (X) and label (y)
10 X = df[feature_cols].fillna(0) # fill missing values with 0
11 y = df[label_col]
```

Python  as cell3

0.0s   

```
1 # 3. Train/Test Split
2 from sklearn.model_selection import train_test_split
3
4 # Split data into training (70%) and testing (30%)
5 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
6
7 print("Train size:", X_train.shape[0])
8 print("Test size:", X_test.shape[0])
9
```

Train size: 623

Test size: 268

+ Python + SQL + Markdown

- First, the important input features are chosen from the dataset: SEX\_ENCODED, PCLASS, AGE, FAMILY\_SIZE, FARE\_PER\_PERSON, and HAS\_CABIN.
- The target label column is defined as SURVIVED, which indicates whether the passenger survived or not.
- The features (X) and the label (y) are separated into two different sets, with missing values in features filled with 0.
- The data is then split into training and testing sets using train\_test\_split from scikit-learn.
- 70 percent of the data is assigned to the training set, while 30 percent is kept aside for testing the model.
- The printed output shows the sizes of both sets: 623 records in the training set and 268 records in the test set.

# TRAINING AND EVALUATING THE LOGISTIC REGRESSION MODEL

Python ▾ as cell5

```
1 # 4. Train Logistic Regression Model
2 from sklearn.linear_model import LogisticRegression
3
4 # Initialize logistic regression model
5 model = LogisticRegression(max_iter=500)
6
7 # Train on training set
8 model.fit(X_train, y_train)
9
10 print(" Logistic Regression model training complete")
11
```

Logistic Regression model training complete

+ Python

+ SQL

+ Markdown

Python ▾ as cell6


```
1 # 5. Evaluate Model Accuracy
2 from sklearn.metrics import accuracy_score
3
4 # Predict on test set
5 preds = model.predict(X_test)
6
7 # Calculate accuracy
8 acc = accuracy_score(y_test, preds)
9
10 print(f" Logistic Regression trained. Accuracy: {acc:.4f}")
11
```

Logistic Regression trained. Accuracy: 0.8246



- The logistic regression model is chosen because it is well-suited for binary classification tasks such as predicting survival (yes or no).
- A logistic regression model is initialized with a maximum number of iterations set to 500, ensuring the algorithm converges during training.
- The model is trained using the training dataset (X\_train, y\_train), learning the relationship between passenger features and survival outcome.
- Once training is complete, a confirmation message is printed to indicate successful model training.
- The trained model is then used to make predictions on the test dataset (X\_test).
- Accuracy is calculated by comparing predicted values with actual survival outcomes in y\_test, and the result shows that the model achieved about 82 percent accuracy.

## INSPECTING PREDICTIONS

Python  as cell7

```
1 # 6. Inspect Predictions (instead of saving model)
2 results = pd.DataFrame({
3     "PassengerID": df.iloc[y_test.index].PASSENGERID.values,
4     "Actual": y_test.values,
5     "Predicted": preds
6 })
7
8 print(" Sample Predictions:")
9 print(results.head(10))
10
```

Sample Predictions:

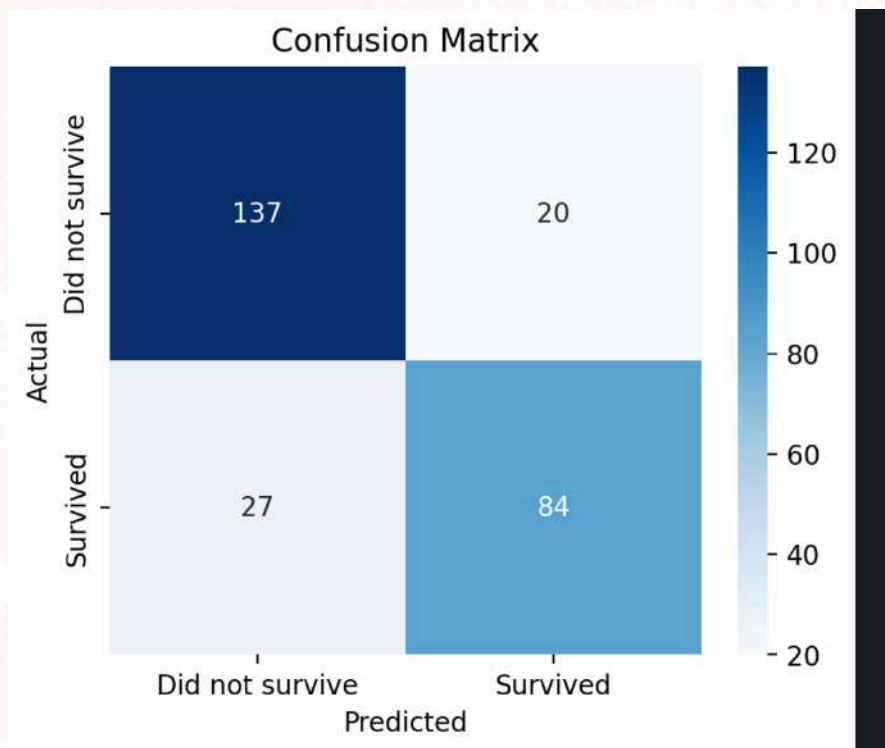
	PassengerID	Actual	Predicted
0	710	1	0
1	440	0	0
2	841	0	0
3	721	1	1
4	40	1	1
5	291	1	1
6	301	1	1
7	334	0	0
8	209	1	1
9	137	1	1

- The predictions made by the model are compared against the actual survival outcomes for passengers, displayed along with their PassengerID.
- A sample of 10 rows is printed to quickly check how well the predicted values match the actual results.

# CONFUSION MATRIX

Python ▾ as cell8

```
1 # 7. Confusion Matrix & Classification Report
2 from sklearn.metrics import confusion_matrix, classification_report
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 # Confusion matrix
7 cm = confusion_matrix(y_test, preds)
8
9 # Plot heatmap
10 plt.figure(figsize=(5,4))
11 sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
12             xticklabels=["Did not survive", "Survived"],
13             yticklabels=["Did not survive", "Survived"])
14 plt.xlabel("Predicted")
15 plt.ylabel("Actual")
16 plt.title("Confusion Matrix")
17 plt.show()
18
19
```



- A confusion matrix is created to compare actual survival outcomes with the model's predictions, showing how many were correctly or incorrectly classified.
- The heatmap visualization makes it easier to interpret results: true positives (survived and predicted survived), true negatives (did not survive and predicted did not survive), and the errors (false positives and false negatives).
- From the matrix: 137 passengers were correctly predicted as not surviving, 84 correctly predicted as surviving, while 20 and 27 cases were misclassified.