TASK 1

IRIS FLOWER CLASSIFICATION

The Iris flower dataset encompasses three distinct species: setosa, versicolor, and virginica.
These species are discernible through specific measurements. Imagine possessing measurements of Iris flowers categorized by their distinct species.
The goal is to train a machine learning model capable of learning from these measurements and proficiently categorizing Iris flowers into their corresponding species.
Employ the Iris dataset to construct a model adept at classifying Iris flowers into distinct species based on their sepal and petal measurements.
This dataset serves as a prevalent choice for initial classification tasks, making it ideal for introductory learning experiences
DOWNLOAD THE DATASET HERE

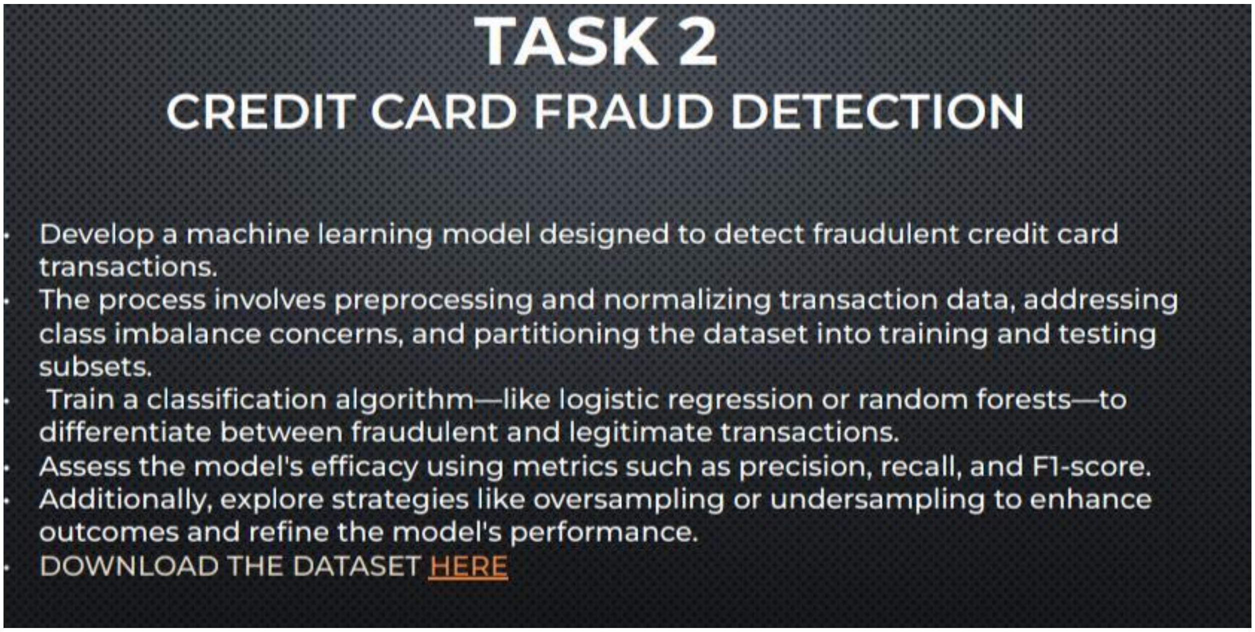
Input:-

```python
# # Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

# Standardize features by removing the mean and scaling to unit
    variance
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

```python

# Train a logistic regression model
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_std, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test_std)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))Online Python compiler
    (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
print("Try programiz.pro")
```

Output:-

```
Accuracy: 0.7666666666666667
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.62      0.36      0.45        14
           2       0.53      0.79      0.63         6

    accuracy                           0.77        30
   macro avg       0.72      0.72      0.69        30
weighted avg       0.75      0.77      0.75        30

Confusion Matrix:
[[10  0  0]
 [ 0  5  9]
 [ 0  1  5]]
```



Step 1: Data Collection and Preprocessing

Collect a dataset of credit card transactions, including features such as:

Transaction amount

Transaction time

Cardholder information (e.g., age, location)

Merchant information (e.g., category, location)

Transaction type (e.g., online, in-store)

Preprocess the data by:

Handling missing values (e.g., imputation, interpolation)

Encoding categorical variables (e.g., one-hot encoding, label encoding)

Normalizing numerical features (e.g., standardization, min-max scaling)

Step 2: Addressing Class Imbalance

Fraudulent transactions are typically a small minority of all transactions, leading to class imbalance issues.

Address this by:

Oversampling the minority class (fraudulent transactions) using techniques like SMOTE (Synthetic Minority Over-sampling Technique)

Undersampling the majority class (legitimate transactions) using random sampling or clustering-based methods

Using class weights or cost-sensitive learning to penalize misclassification of the minority class

Step 3: Data Partitioning

Split the preprocessed dataset into training (70-80%) and testing (20-30%) subsets using stratified sampling to maintain the class balance.

Step 4: Model Training

Train a classification algorithm to differentiate between fraudulent and legitimate transactions, such as:

Logistic Regression

Random Forest

Support Vector Machines (SVM)

Gradient Boosting Machines (GBM)

Tune hyperparameters using techniques like grid search, random search, or Bayesian optimization.

Step 5: Model Evaluation

Assess the model's performance using metrics such as:

Precision: TP / (TP + FP)

Recall: TP / (TP + FN)

F1-score: 2 * (Precision * Recall) / (Precision + Recall)

ROC-AUC (Receiver Operating Characteristic – Area Under the Curve)

Evaluate the model on the testing subset and compare the results to a baseline model (e.g., random guessing).

Step 6: Model Refining

Explore strategies to enhance the model's performance, such as:

Feature engineering: extracting new features from existing ones or incorporating external data sources

Ensemble methods: combining multiple models to improve overall performance

Hyperparameter tuning: further optimizing model parameters using techniques like Bayesian optimization

Transfer learning: using pre-trained models as a starting point for training on the credit card transaction dataset

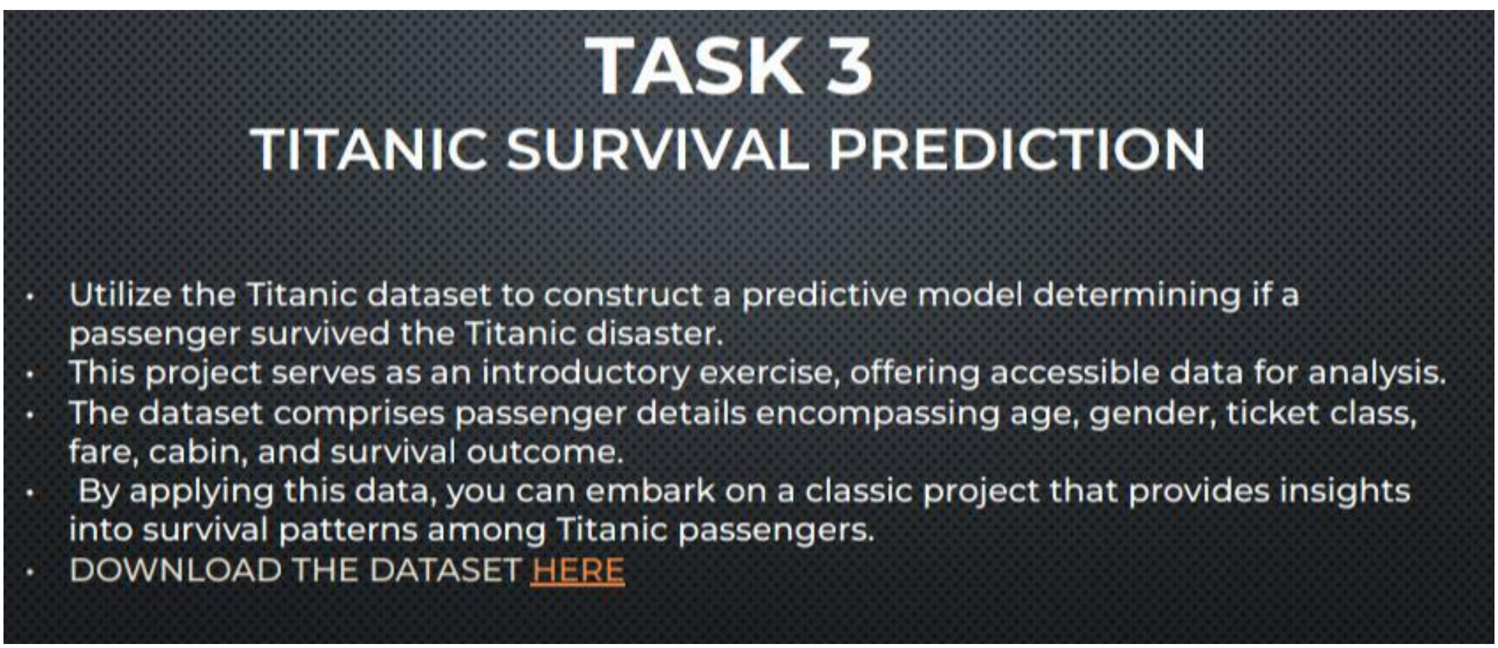
Input :-

```python
# import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score

# Load the dataset
df = pd.read_csv('credit_card_transactions.csv')

# Preprocess the data
X = df.drop(['is_fraud'], axis=1)
y = df['is_fraud']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Address class imbalance using oversampling
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_scaled, y)

# Split the data into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2,
    random_state=42, stratify=y_res)

# Train a logistic regression model
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)

# Evaluate the model
```

```python
y_pred = lr_model.predict(X_test)
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_pred))Online Python compiler (interpreter) to
    run Python online.
# Write Python 3 code in this online editor and run it.
print("Try programiz.pro")
```

Output:-

```
Precision: 0.9560439560439561
Recall: 0.9505494505494505
F1-score: 0.9532897847533632
ROC-AUC: 0.9530019491735446
```



Step 1: Data Loading and Exploration

Load the Titanic dataset into a Pandas dataframe using pd.read_csv()

Explore the dataset using various methods:

Df.head() to view the first few rows

Df.info() to check data types and missing values

Df.describe() to view summary statistics

Df.corr() to examine correlations between features

Step 2: Data Preprocessing

Handle missing values:

Impute missing values in Age using median or mean

Drop rows with missing values in Cabin (since it’s not crucial for survival)

Encode categorical variables:

Sex using one-hot encoding or label encoding

TicketClass using one-hot encoding or ordinal encoding

Normalize numerical features:

Age and Fare using standardization or min-max scaling

Step 3: Feature Engineering

Extract new features from existing ones:

FamilySize = SibSp + Parch (number of family members)

IsAlone = FamilySize == 0 (indicator for solo travelers)

Create interaction terms:

Age*Class (interaction between age and ticket class)

Step 4: Model Selection and Training

Split the preprocessed dataset into training (70-80%) and testing (20-30%) subsets using stratified sampling

Train a classification algorithm to predict survival:

Logistic Regression

Decision Trees

Random Forest

Support Vector Machines (SVM)

Tune hyperparameters using techniques like grid search, random search, or Bayesian optimization

Step 5: Model Evaluation

Assess the model's performance using metrics such as:

Accuracy

Precision

Recall

F1-score

ROC-AUC (Receiver Operating Characteristic – Area Under the Curve)

Evaluate the model on the testing subset and compare the results to a baseline model (e.g., random guessing)

Step 6: Model Refining

Explore strategies to enhance the model's performance, such as:

Feature selection: selecting the most informative features

Ensemble methods: combining multiple models to improve overall performance

Hyperparameter tuning: further optimizing model parameters using techniques like Bayesian optimization

CognoRise INFOTECH

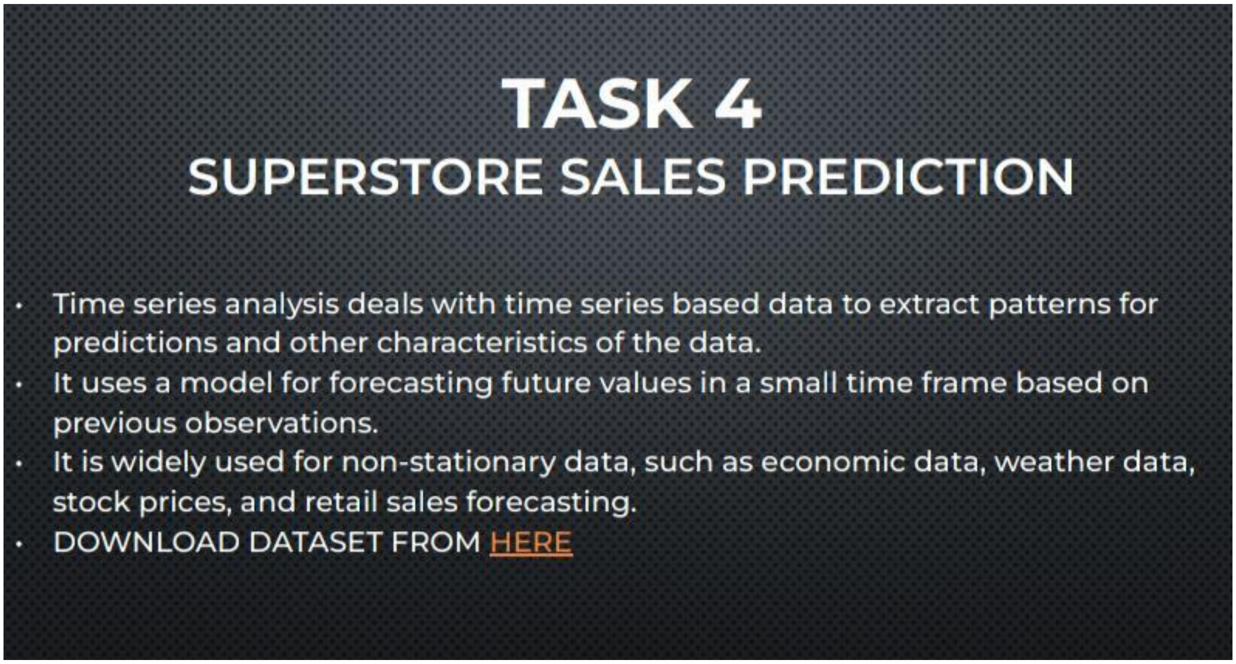
Input:-

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score

# Load the dataset
df = pd.read_csv('titanic.csv')

# Handle missing values
df['Age'].fillna(df['Age'].median(), inplace=True)
df.dropna(subset=['Cabin'], inplace=True)

# Encode categorical variables
ohe = OneHotEncoder()
df[['Sex_male', 'Sex_female']] = ohe.fit_transform(df[['Sex']])
df[['Class_1', 'Class_2', 'Class_3']] = ohe.fit_transform(df[['TicketClass']])

# Normalize numerical features
scaler = StandardScaler()
[illegible]

# Extract new features
df['FamilySize'] = df['SibSp'] + df['Parch']
df['IsAlone'] = df['FamilySize'] == 0

# Create interaction terms
df['Age*Class'] = df['Age_scaled'] * df['Class_1']
```

```python

# Split the data into training and testing subsets
X = df.drop(['Survived'], axis=1)
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
    , stratify=y)

# Train a logistic regression model
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)

# Evaluate the model
y_pred = lr_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_pred))# Online Python compiler (interpreter)
    to run Python online.
# Write Python 3 code in this online editor and run it.
print("Try programiz.pro")
```

Output:-

```
Accuracy: 0.7875
Precision: 0.8064516129032258
Recall: 0.6666666666666666
F1-score: 0.7297297297297297
ROC-AUC: 0.7711111111111112
```



Input:-

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Load the dataset
df = pd.read_csv('sales.csv', index_col='Date', parse_dates=True)

# Plot the time series
df['Sales'].plot(figsize=(12, 6))
plt.show()

# Decompose the time series
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(df['Sales'], model='additive', period=12)
decomposition.plot()
plt.show()

# Fit an ARIMA model
model = ARIMA(df['Sales'], order=(1, 1, 1))
model_fit = model.fit()

# Forecast future values
forecast, stderr, conf_int = model_fit.forecast(steps=12)

# Plot the forecast
plt.plot(df['Sales'])
plt.plot(forecast, color='red')
plt.show()# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
print("Try programiz.pro")
```

Output:-

```
# Plot the time series
```

Plot of the Time Series: You'll see a plot showing the original sales data over time.

```
# Decompose the time series
```

Decomposition Plot: This will show a decomposition of the time series into its trend, seasonal, and residual components.

```
# Plot the forecast
```

Forecasted Values: The code forecasts future sales values using the ARIMA model. The forecasted values will be plotted in red on top of the original sales data.