(f). Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

Below is an implementation outline for a Gradient Boosting Machine (GBM) using Python, along with hyperparameter tuning and feature importance exploration. I'll use the **XGBoost** library for this demonstration.

Implementation Outline

1. **Load the Data**: Load a dataset for demonstration (e.g., the UCI dataset or a Kaggle dataset).
2. **Preprocessing**: Handle missing values, encode categorical variables, and split the dataset.
3. **Model Training**: Use XGBoost for training.
4. **Hyperparameter Tuning**: Tune hyperparameters using grid search or random search.
5. **Feature Importance**: Extract and visualize feature importance.

Code :-Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBClassifier, plot_importance
import matplotlib.pyplot as plt
```

Step 1: Load Titanic dataset

```
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)
```

Step 2: Preprocess the dataset

```
data = data.drop(["Name", "Ticket", "Cabin"], axis=1)   Drop irrelevant features
data = pd.get_dummies(data, columns=["Sex", "Embarked"], drop_first=True)   One-hot encoding
data = data.fillna(data.median())   Handle missing values
X = data.drop("Survived", axis=1)
y = data["Survived"]
```

Step 3: Split data into training and test sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 4: Initialize the XGBoost classifier

```
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
```

Step 5: Define hyperparameter grid for tuning

```
param_grid = {
'n_estimators': [100, 200],
   'max_depth': [3, 6, 9],
   'learning_rate': [0.01, 0.1, 0.2],
   'subsample': [0.8, 1.0],
   'colsample_bytree': [0.8, 1.0]
}
```

Step 6: Perform grid search with cross-validation

```
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=3, scoring='accuracy',
verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Step 7: Display the best parameters and accuracy

```
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)
```

Step 8: Train the model with the best parameters

best_xgb = grid_search.best_estimator_

best_xgb.fit(X_train, y_train)

Step 9: Evaluate on the test set

accuracy = best_xgb.score(X_test, y_test)

print("Test Set Accuracy:", accuracy)

Step 10: Plot feature importance

plot_importance(best_xgb)

plt.title("Feature Importance")

plt.show()

```
[Running] python -u "C:\Users\Admin\AppData\Local\Temp\tempCodeRunnerFile.python"
Fitting 3 folds for each of 72 candidates, totalling 216 fits
C:\Users\Admin\AppData\Local\Programs\Python\Python38\lib\site-packages\xgboost\core.py:158: UserWarning: [12:45:38]
WARNING:
C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0c55ff5f71b100e98-1\xgboost\xgboost-ci-windows\src\lea
rner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 200, 'subsample': 0.8}
Best Cross-Validation Accuracy: 0.8230389202094339
C:\Users\Admin\AppData\Local\Programs\Python\Python38\lib\site-packages\xgboost\core.py:158: UserWarning: [12:45:38]
WARNING:
C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0c55ff5f71b100e98-1\xgboost\xgboost-ci-windows\src\lea
rner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Test Set Accuracy: 0.7988826815642458
```