```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, mean_squared_error, mean_absolute_error

# Models
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.svm import SVC, SVR
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.naive_bayes import GaussianNB

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

# Display the first few rows of the dataframe
df.head()
```

ĸ	sibsp	zero	zero.1	zero.2	zero.3	zero.4	...	zero.12	zero.13	zero.14	Pclass
)	1	0	0	0	0	0	...	0	0	0	3
1	1	0	0	0	0	0	...	0	0	0	1
1	0	0	0	0	0	0	...	0	0	0	3
1	1	0	0	0	0	0	...	0	0	0	1
)	0	0	0	0	0	0	...	0	0	0	3

```
print(df.columns)
```

```
Index(['Passengerid', 'Age', 'Fare', 'Sex', 'sibsp', 'zero', 'zero.1',
       'zero.2', 'zero.3', 'zero.4', 'zero.5', 'zero.6', 'Parch', 'zero.7',
       'zero.8', 'zero.9', 'zero.10', 'zero.11', 'zero.12', 'zero.13',
       'zero.14', 'Pclass', 'zero.15', 'zero.16', 'Embarked', 'zero.17',
       'zero.18', '2urvived'],
      dtype='object')
```

Handle missing values

```
# Handle missing values (if any)
df = df.dropna()

# Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

df.head()
```

	Passengerid	Age	Fare	Sex	sibsp	zero	zero.1	zero.2	zero.3	zero.4	...
0	1	22.0	7.2500	0	1	0	0	0	0	0	...
1	2	38.0	71.2833	1	1	0	0	0	0	0	...
2	3	26.0	7.9250	1	0	0	0	0	0	0	...
3	4	35.0	53.1000	1	1	0	0	0	0	0	...
4	5	35.0	8.0500	0	0	0	0	0	0	0	...

5 rows × 28 columns

split the data

```
X = df.drop('2urvived', axis=1)
y = df['2urvived']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

standarlize the feature variable

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Classification Model

```
def evaluate_classification_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    return {'accuracy': accuracy, 'precision': precision, 'recall': recall, 'f1_score': f1}
```

Train and evaluate

```
# Initialize models
classification_models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree Classifier': DecisionTreeClassifier(),
    'Random Forest Classifier': RandomForestClassifier(),
    'Support Vector Classifier': SVC(),
    'K-Nearest Neighbors Classifier': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB()
}

# Evaluate models
classification_results = {}
for model_name, model in classification_models.items():
    classification_results[model_name] = evaluate_classification_model(model, X_train, X_test, y_train, y_test)

classification_results
```

```
{'Logistic Regression': {'accuracy': 0.8320610687022901,
  'precision': 0.8265663605856105,
  'recall': 0.8320610687022901,
  'f1_score': 0.8196274561417092},
 'Decision Tree Classifier': {'accuracy': 0.8282442748091603,
  'precision': 0.824794107740595,
  'recall': 0.8282442748091603,
  'f1_score': 0.8261674417996165},
 'Random Forest Classifier': {'accuracy': 0.8625954198473282,
  'precision': 0.8604637764655049,
  'recall': 0.8625954198473282,
  'f1_score': 0.8545461416370647},
 'Support Vector Classifier': {'accuracy': 0.8702290076335878,
  'precision': 0.872570712099421,
  'recall': 0.8702290076335878,
  'f1_score': 0.8606212161095025},
 'K-Nearest Neighbors Classifier': {'accuracy': 0.851145038167939,
  'precision': 0.8477122275638563,
  'recall': 0.851145038167939,
  'f1_score': 0.8418699795439805},
 'Naive Bayes': {'accuracy': 0.8129770992366412,
  'precision': 0.8033948880268629,
  'recall': 0.8129770992366412,
  'f1_score': 0.7998789372835173}}
```

Regression model

```
def evaluate_regression_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)

    return {'mse': mse, 'rmse': rmse, 'mae': mae}
```

Train and evaluate

```
# Initialize models
regression_models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree Regressor': DecisionTreeRegressor(),
    'Random Forest Regressor': RandomForestRegressor(),
    'Support Vector Regressor': SVR(),
    'K-Nearest Neighbors Regressor': KNeighborsRegressor()
}

# Evaluate models
regression_results = {}
for model_name, model in regression_models.items():
    regression_results[model_name] = evaluate_regression_model(model, X_train, X_test, y_train, y_test)

regression_results
```

```
{'Linear Regression': {'mse': 0.1271509495397618,
  'rmse': 0.3565823180413771,
  'mae': 0.2753783467030288},
 'Decision Tree Regressor': {'mse': 0.17938931297709923,
  'rmse': 0.42354375568186486,
  'mae': 0.17938931297709923},
 'Random Forest Regressor': {'mse': 0.09621603053435115,
  'rmse': 0.31018708956749175,
  'mae': 0.1784732824427481},
 'Support Vector Regressor': {'mse': 0.12101607192287066,
  'rmse': 0.34787364361628587,
  'mae': 0.21941974508668727},
 'K-Nearest Neighbors Regressor': {'mse': 0.12122137404580155,
  'rmse': 0.34816860002849415,
  'mae': 0.21068702290076338}}
```

result

```
# Combine classification and regression results
all_results = {
    'Classification Models': classification_results,
    'Regression Models': regression_results
}

# Print results
for model_type, results in all_results.items():
    print(f"Results for {model_type}:")
    for model_name, metrics in results.items():
        print(f"\n{model_name}:")
        for metric, value in metrics.items():
            print(f"{metric}: {value}")
        print("\n")
```

```
accuracy: 0.8129770992366412
precision: 0.8033948880268629
recall: 0.8129770992366412
f1_score: 0.7998789372835173


Results for Regression Models:

Linear Regression:
mse: 0.1271509495397618
rmse: 0.3565823180413771
mae: 0.2753783467030288



Decision Tree Regressor:
mse: 0.17938931297709923
rmse: 0.42354375568186486
mae: 0.17938931297709923



Random Forest Regressor:
mse: 0.09621603053435115
rmse: 0.31018708956749175
mae: 0.1784732824427481



Support Vector Regressor:
mse: 0.12101607192287066
rmse: 0.34787364361628587
mae: 0.21941974508668727



K-Nearest Neighbors Regressor:
mse: 0.12122137404580155
rmse: 0.34816860002849415
mae: 0.21068702290076338
```

determine best model

```
# Example logic to identify best model based on F1 score for classification and RMSE for regression
best_classification_model = max(classification_results, key=lambda x: classification_results[x]['f1_score'])
best_regression_model = min(regression_results, key=lambda x: regression_results[x]['rmse'])

print(f"Best Classification Model: {best_classification_model} with F1 Score: {classification_results[best_classification_model]['f1_sco
print(f"Best Regression Model: {best_regression_model} with RMSE: {regression_results[best_regression_model]['rmse']}")
```

```
Best Classification Model: Support Vector Classifier with F1 Score: 0.8606212161095025
Best Regression Model: Random Forest Regressor with RMSE: 0.31018708956749175
```