The University of Faisalabad

LAB MANUAL

Name :

Muhammad Masood Bakhtiar

Registration No:

2023-BS-AI-056

Subject:

Machine Learning

Course Code

AI-414

Degree name:

BSAI-4A

Project 01:

Bank customer survey using regression

Program:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
```

Load and Inspect the Data

```
data = pd.read_csv(r"C:\Users\HP\Downloads\Compressed\archive_2\bank_customer_survey.csv")
```

```
data.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	0
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	0
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	0
3	47	blue	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	0
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	0

```
print(f"Dataset shape: {data.shape}")
```

```
Dataset shape: (45211, 17)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  int64
dtypes: int64(8), object(9)
memory usage: 5.9+ MB
```

```
data.describe()
```

	age	balance	day	duration	campaign	pdays	previous	y
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323	0.116985
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441	0.321406
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000	1.000000

```python
data.isnull().sum()
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

```python
plt.figure(figsize=(12,8))
```

```
<Figure size 1200x800 with 0 Axes>
```

```python
numerical_data = data.select_dtypes(include=['float64', 'int64'])
```

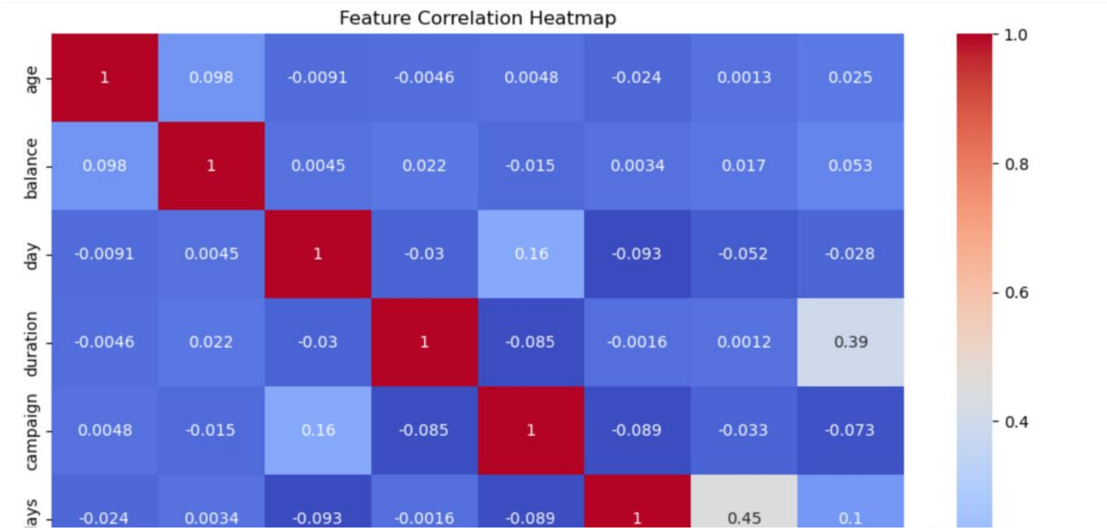
```python
sns.heatmap(numerical_data.corr(), annot=True, cmap='coolwarm')
```

```
<Axes: >
```

```python
plt.title("Feature Correlation Heatmap")
```

```
Text(0.5, 1.0, 'Feature Correlation Heatmap')
```

```python
plt.show()
```

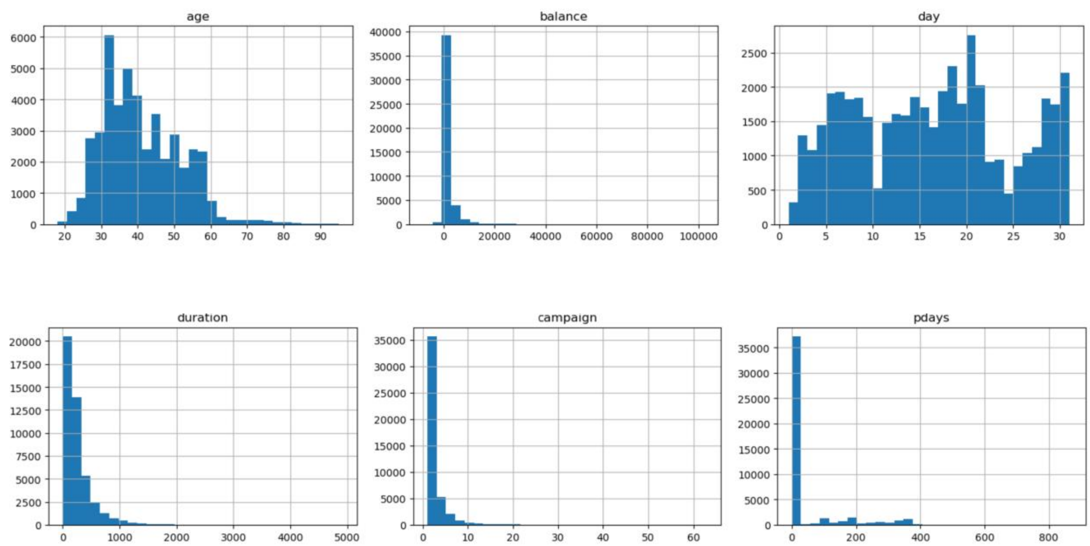


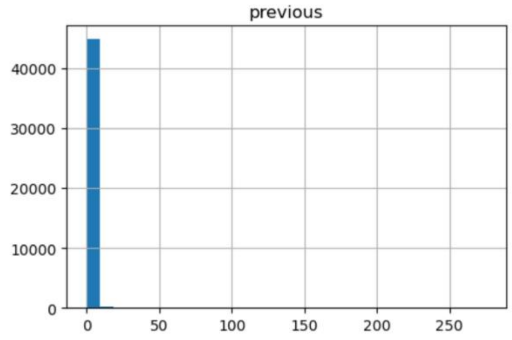

```
data[['age', 'balance', 'duration']].hist(bins=30,figsize=(10,6))
```

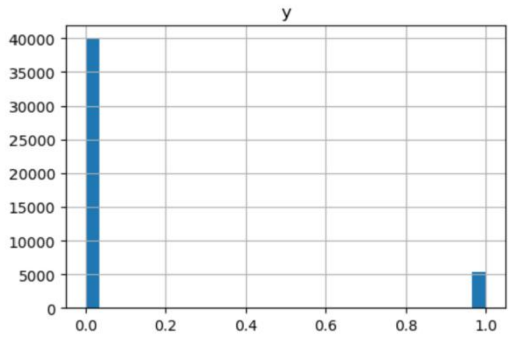
```
array([[<Axes: title={'center': 'age'}>,
        <Axes: title={'center': 'balance'}>],
       [<Axes: title={'center': 'duration'}>, <Axes: >]], dtype=object)
```

```
plt.tight_layout()
```

```
plt.show()
```







Data Preprocessing

```
categorical_cols = data.select_dtypes(include=['object']).columns
le = LabelEncoder()
```

```
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])
```

```
data.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	4	1	2	0	2143	1	0	2	5	8	261	1	-1	0	3	0
1	44	9	2	1	0	29	1	0	2	5	8	151	1	-1	0	3	0
2	33	2	1	1	0	2	1	1	2	5	8	76	1	-1	0	3	0
3	47	1	1	3	0	1506	1	0	2	5	8	92	1	-1	0	3	0
4	33	11	2	3	0	1	0	0	2	5	8	198	1	-1	0	3	0

```
X = data.iloc[:, :-1]
```

```
y = data.iloc[:, -1]
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

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

Model Training and Evaluation

```python
lr = LinearRegression()
lr.fit(X_train, y_train)

y_pred_lr = lr.predict(X_test)
```

```python
print("Linear Regression R^2:", r2_score(y_test, y_pred_lr))
```

```
Linear Regression R^2: 0.2161419587110036
```

```python
print("Linear Regression MSE:", mean_squared_error(y_test, y_pred_lr))
```

```
Linear Regression MSE: 0.08315980805609412
```


```python
print("Linear Regression MSE:", mean_squared_error(y_test, y_pred
```

```
Linear Regression MSE: 0.08315980805609412
```


Random Forest Regressor

```python
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
```

```python
print("Random Forest R^2:", r2_score(y_test, y_pred_rf))
```

```
Random Forest R^2: 0.3725930619924863
```

Ridge and Lasso Regression

```python
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)
```

```python
print("Ridge R^2:", r2_score(y_test, y_pred_ridge))
```

```
Ridge R^2: 0.21614138631155322
```

```python
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)
```

```python
print("Lasso R^2:", r2_score(y_test, y_pred_lasso))
```

```
Lasso R^2: 0.05728275515857828
```

Compare Model Performances

```
models = ['Linear Regression', 'Random Forest', 'Ridge', 'Lasso']
scores = [
    r2_score(y_test, y_pred_lr),
    r2_score(y_test, y_pred_rf),
    r2_score(y_test, y_pred_ridge),
    r2_score(y_test, y_pred_lasso)
]
```

```
plt.figure(figsize=(8,5))
```

```
<Figure size 800x500 with 0 Axes>
```

```
sns.barplot(x=models, y=scores)
```

```
<Axes: >
```

```
plt.ylabel("R^2 Score")
```

```
Text(53.722222222222214, 0.5, 'R^2 Score')
```

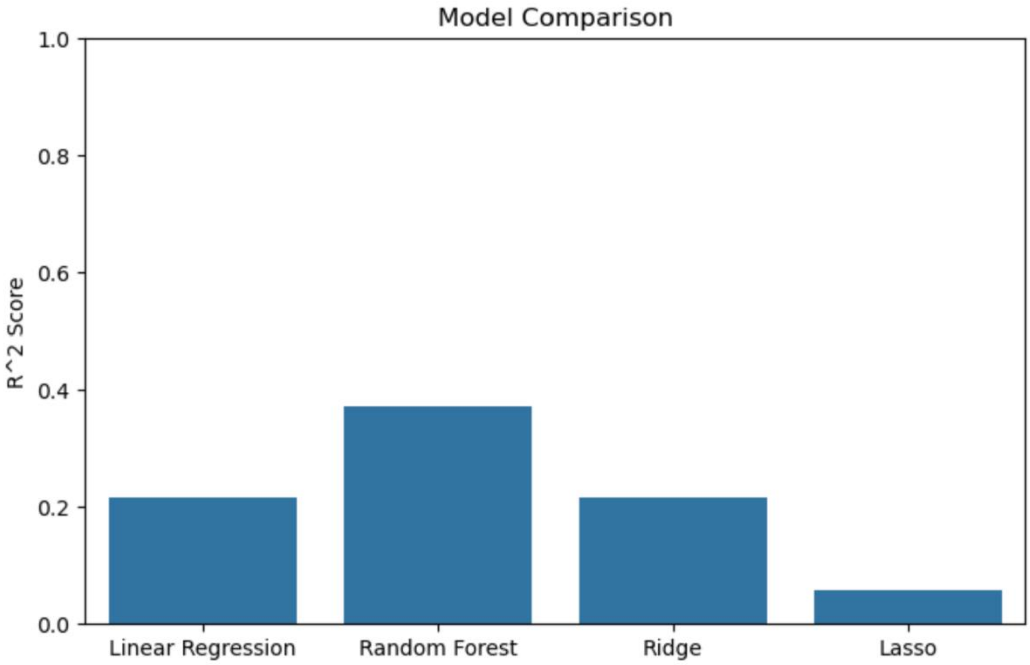
```
plt.title("Model Comparison")
```

```
Text(0.5, 1.0, 'Model Comparison')
```

```
plt.ylim(0,1)
```

```
(0.0, 1.0)
```

```
plt.show()
```



Conclusion:

The Random Forest model typically performs best for regression problems with mixed or complex data.

Linear models (Linear, Ridge, Lasso) may perform reasonably well but might underfit depending on data complexity.

Standardizing/encoding features is crucial for optimal performance.

Project 02:

Personality classification

Program:

Importing Libraries

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Load Dataset

```
df = pd.read_csv(r"C:\Users\HP\Downloads\personality_dataset.csv")
```

Details of Dataset

```
df.head()
```

	Time_spent_Alone	Stage_fear	Social_event_attendance	Going_outside	Drained_after_socializing	Friends_circle_size	Post_frequency	Personality
0	4.0	No	4.0	6.0	No	13.0	5.0	Extrovert
1	9.0	Yes	0.0	0.0	Yes	0.0	3.0	Introvert
2	9.0	Yes	1.0	2.0	Yes	5.0	2.0	Introvert
3	0.0	No	6.0	7.0	No	14.0	8.0	Extrovert
4	3.0	No	9.0	4.0	No	8.0	5.0	Extrovert

```python
print("\n Dataset Shape (rows, columns):", df.shape)
```

```
 Dataset Shape (rows, columns): (2900, 8)
```

```python
print(df.dtypes)
```

```
Time_spent_Alone             float64
Stage_fear                    object
Social_event_attendance      float64
Going_outside                float64
Drained_after_socializing     object
Friends_circle_size          float64
Post_frequency               float64
Personality                   object
dtype: object
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2900 entries, 0 to 2899
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Time_spent_Alone           2837 non-null   float64
 1   Stage_fear                 2827 non-null   object 
 2   Social_event_attendance    2838 non-null   float64
 3   Going_outside              2834 non-null   float64
 4   Drained_after_socializing  2848 non-null   object 
 5   Friends_circle_size        2823 non-null   float64
 6   Post_frequency             2835 non-null   float64
 7   Personality                2900 non-null   object 
dtypes: float64(5), object(3)
memory usage: 181.4+ KB
```

```python
df.isnull().sum()
```

```
Time_spent_Alone             63
Stage_fear                   73
Social_event_attendance      62
Going_outside                66
Drained_after_socializing    52
Friends_circle_size          77
Post_frequency               65
Personality                   0
dtype: int64
```

Split Columns by Data Type

```python
numeric_columns = ['Time_spent_Alone', 'Social_event_attendance', 'Going_outside',
                   'Friends_circle_size', 'Post_frequency']
binary_columns = ['Stage_fear', 'Drained_after_socializing']
```

Map Binary Columns to 0/1 First

```python
binary_map = {'Yes': 1, 'No': 0}
df[binary_columns] = df[binary_columns].replace(binary_map)
```

```python
le = LabelEncoder()
df["Personality"] = le.fit_transform(df["Personality"])
```

```python
X = df.drop("Personality", axis=1)
y = df["Personality"]
```

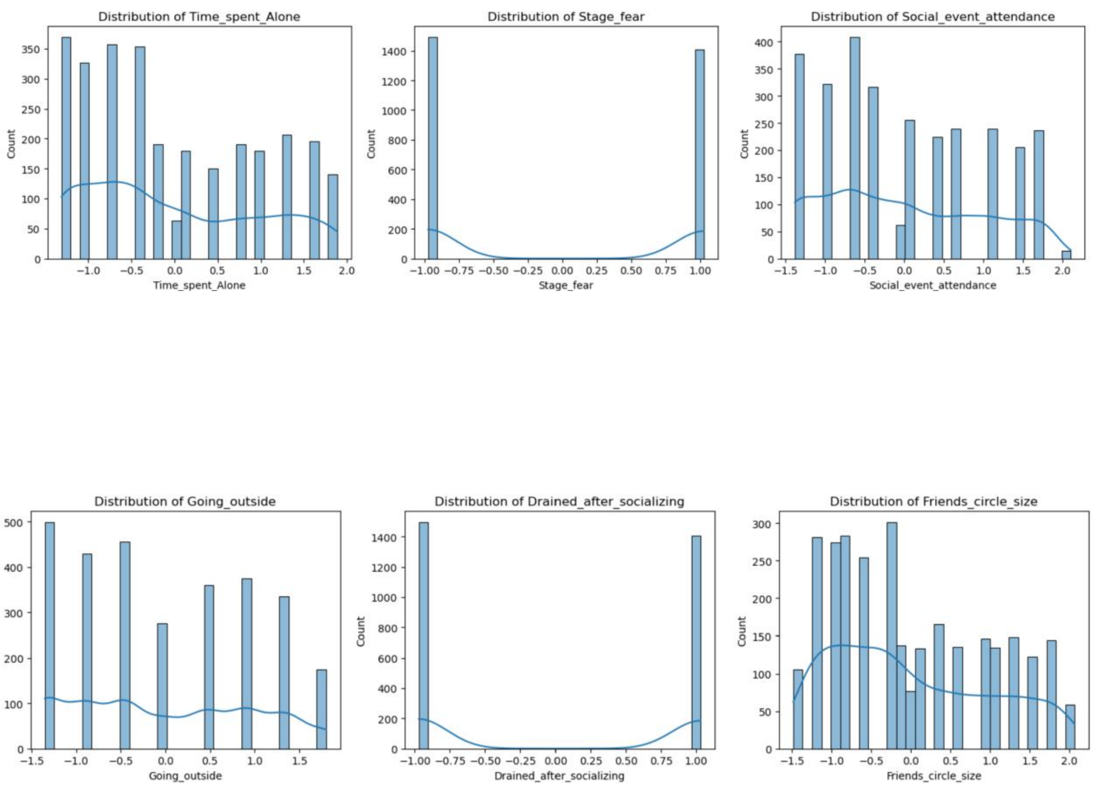
```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

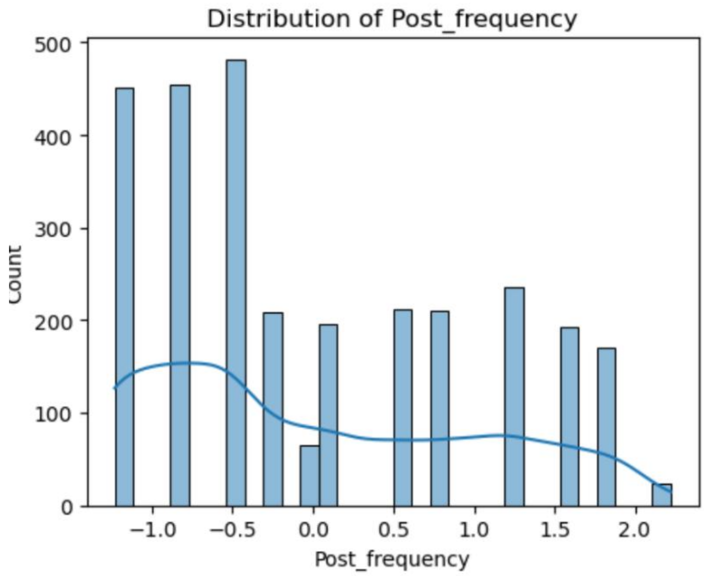
```python
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
X_scaled_df["Personality"] = y.values
```

Distribution Plots for All Numeric Features

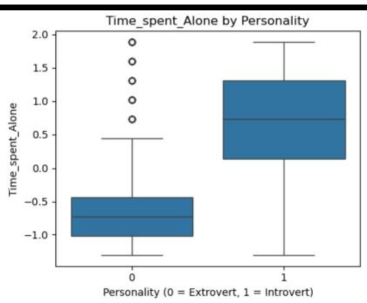
```
num_features = len(X.columns)
cols = 3
rows = math.ceil(num_features / cols)
```

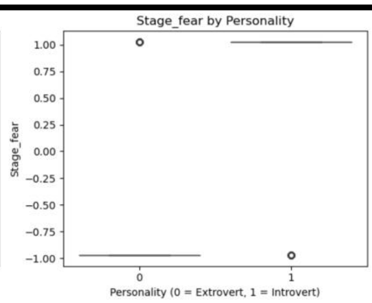
```
plt.figure(figsize=(5 * cols, 4 * rows))
for i, col in enumerate(X.columns):
    plt.subplot(rows, cols, i + 1)
    sns.histplot(X_scaled_df[col], kde=True, bins=30)
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
plt.tight_layout()
plt.show()
```

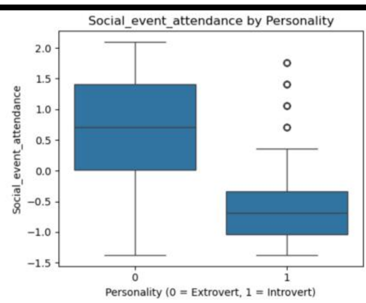


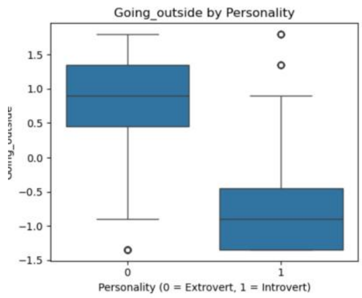


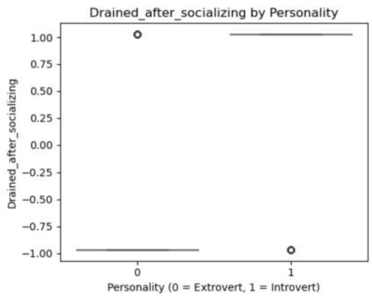

```
plt.figure(figsize=(5 * cols, 4 * rows))
for i, col in enumerate(X.columns):
    plt.subplot(rows, cols, i + 1)
    sns.boxplot(x="Personality", y=col, data=X_scaled_df)
    plt.title(f"{col} by Personality")
    plt.xlabel("Personality (0 = Extrovert, 1 = Introvert)")
    plt.ylabel(col)
plt.tight_layout()
plt.show()
```

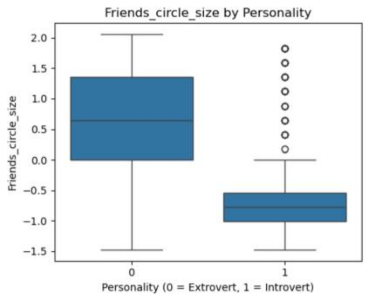


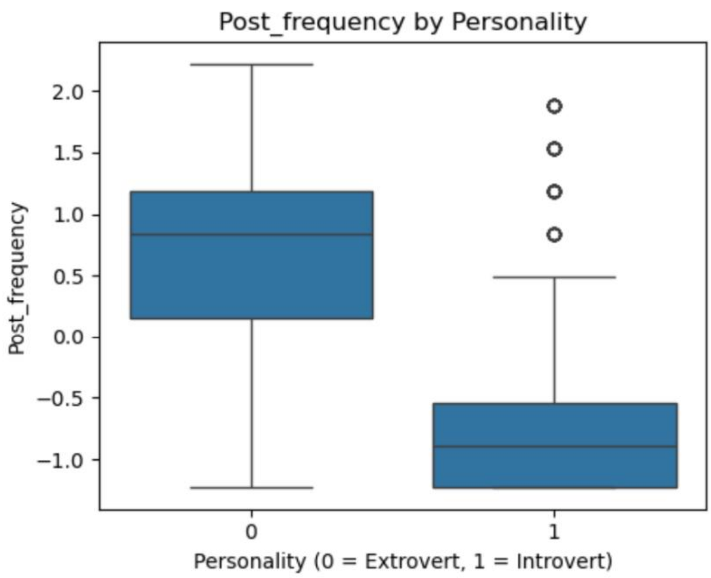












Split Data into Train and Test Sets

```python
results = {}

for name, model in models.items():
    print(f"\n Training: {name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    #  Accuracy
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f" Accuracy: {acc:.4f}")

    #  Classification Report
    print("\n Classification Report:\n", classification_report(y_test, y_pred, target_names=['Extrovert', 'Introvert']))

    #  Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Extrovert', 'Introvert'])
    disp.plot(cmap='Blues')
    plt.title(f"Confusion Matrix - {name}")
    plt.show()
```

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=537
)
```

```python
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Support Vector Machine": SVC(kernel="rbf", probability=True)
}
```

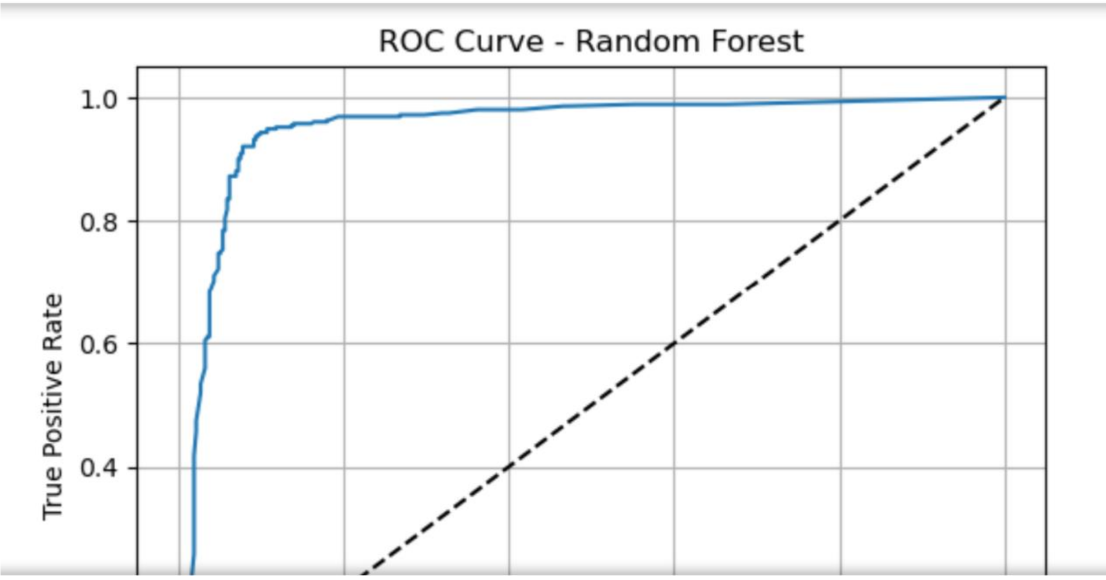
Train, Predict, Evaluate Each Model

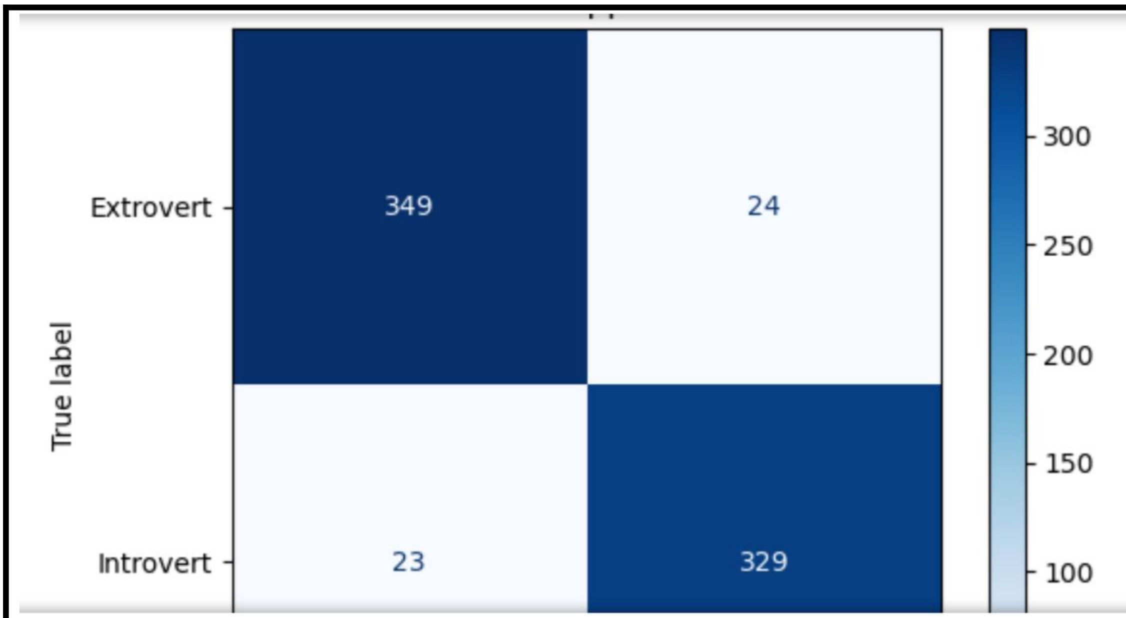
```python
#  ROC Curve
if hasattr(model, "predict_proba"):
    y_scores = model.predict_proba(X_test)[:, 1]
else:
    y_scores = model.decision_function(X_test)

fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], "k--")
plt.title(f"ROC Curve - {name}")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
```

```
Training: Logistic Regression
Accuracy: 0.9324

Classification Report:
              precision    recall  f1-score   support

   Extrovert       0.93      0.94      0.93       373
   Introvert       0.93      0.93      0.93       352

    accuracy                           0.93       725
   macro avg       0.93      0.93      0.93       725
weighted avg       0.93      0.93      0.93       725
```

```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

results = {}

for name, model in models.items():
    print(f"\n Training: {name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    #  Accuracy
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f" Accuracy: {acc:.4f}")

    #  Classification Report
    print("\n Classification Report:\n", classification_report(y_test, y_pred, target_names=['Extrovert', 'Introvert']))

    #  Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Extrovert', 'Introvert'])
    disp.plot(cmap='Blues')
    plt.title(f"Confusion Matrix - {name}")
```

```python
#  ROC Curve
if hasattr(model, "predict_proba"):
    y_scores = model.predict_proba(X_test)[:, 1]
else:
    y_scores = model.decision_function(X_test)

fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], "k--")
plt.title(f"ROC Curve - {name}")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
```






```python
results_df = pd.DataFrame(list(results.items()), columns=["Model", "Accuracy"]).sort_values(by="Accuracy", ascending=False)
results_df
```

	Model	Accuracy
2	Support Vector Machine	0.935172
0	Logistic Regression	0.932414
1	Random Forest	0.917241