HW4-Template1 (2)

April 27, 2024

[185]:
```python
# TEMPLATE 1
# Code template for final project - CAP 4773 - Intro to Data Science
# Spring 2024 - Florida Atlantic University - Dr. Juan Yepes

'''FINAL PROJECT ROADMAP:
The objective is to cover as much as you can of the following topics with your␣
↪dataset.

Step 1. PROBLEM UNDERSTANDING
•       Dataset selection.
•       Defining a classification task.
•       Defining a prediction task.

Step 2. DATA PREPARATION
•       Handling missing data.
•       Creating features or encoding categorical values.
•       Applying feature scaling or normalization.

Step 3. EXPLORATORY DATA ANALYSIS
•       Analyzing basic trend statistics, generating scatter plots, box plots,␣
↪among others.
•       Conducting correlation analysis.

Step 4. SETUP PHASE
•       Preparing the Python environment, importing libraries.
•       Setting up your X (features) and Y (target) data.
•       Splitting the data into training and test sets.

Step 5. MODELING PHASE
Apply several models:
1.       Multiple Linear Regression
2.       Multiple Logistic Regression
3.       Linear Regression using Regularization.
4.       Polynomial Regression
5.       LDA - Linear Discriminant Analysis
6.       Decision Tree for classification
7.       Decision Tree for regression
```

```
8.        Random Forest Classifier
9.        Support Vector Machines
10.        K-means - Unsupervised Learning

Step 6. EVALUATION PHASE
•        Evaluate model performance using k-fold cross-validation.
•        Evaluate model performance using other related metrics.
•        Evaluate models for overfitting.
•        Identify the most important features for each model.
•        Evaluate classification model performance metrics (accuracy,␣
↪precision, recall, and F1 score).
•        Evaluate prediction model performance metrics (MSE (Mean Squared␣
↪Error), RMSE (Root Mean Squared Error), and MAE (Mean Absolute Error) for␣
↪regression).
•        Report and present the results of your model evaluations, highlighting␣
↪strengths and weaknesses.

Step 7. DEPLOYMENT PHASE
•        Make predictions and classifications with new data.'''
True
```

[185]: True

[186]:
```
# Step 1. PROBLEM UNDERSTANDING
# Based on various features gathered from Twitter API, determine whether an␣
↪account is a bot or a human.

# 1.1. Dataset selection:
# Name: Twitter Human Bots Dataset
# Description: This dataset comprises records of Twitter accounts with various␣
↪attributes obtained via the Twitter API.
# It is used to classify accounts into two categories: human or bot. This␣
↪classification helps in understanding
# the spread and characteristics of bots on Twitter.
# Source: Kaggle (specific URL if available)
# Number of Instances: 37,438
# Number of Attributes: 20 attributes including both numeric and categorical␣
↪types
# Attribute Information:
#   created_at - Date when the account was created
#   default_profile - Boolean indicating whether the account has a default␣
↪profile
#   default_profile_image - Boolean indicating whether the account has a␣
↪default profile image
#   description - User account description
#   favourites_count - Total number of favourite tweets
```

```
#   followers_count - Total number of followers
#   friends_count - Total number of friends
#   geo_enabled - Boolean indicating whether the account has the geographic␣
↪location enabled
#   id - Unique identifier of the account
#   lang - Language of the account
#   location - Location of the account
#   profile_background_image_url - Profile background image URL
#   profile_image_url - Profile image URL
#   screen_name - Screen name
#   statuses_count - Total number of tweets
#   verified - Boolean indicating whether the account is verified
#   average_tweets_per_day - Average tweets posted per day
#   account_age_days - Account age measured in days
#   account_type - Account type with two unique values: bot or human

# 1.2. Defining a classification task.
# TASK 1: Classify accounts as bot or human based on the collected features.
# TASK 2: Investigate patterns and potential predictive features that␣
↪differentiate bot accounts from human accounts.

# 1.3. Defining a prediction task. (OPTIONAL)
# TASK 3: Predict the potential growth of followers based on other attributes␣
↪such as the number of tweets, account age, and verification status.
```

[187]:

```
# Step 2. DATA PREPARATION

# 2.1. Load the dataset

# 2.1.1 Load required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.discrete.discrete_model import Logit

from sklearn.linear_model import LinearRegression, Lasso, LassoCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import cross_val_score, train_test_split, KFold
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
↪mean_squared_error, r2_score
```

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,␣
↪plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
from sklearn.pipeline import Pipeline
import re

# 2.1.2 Load the Twitter bots dataset
# Path to the dataset file
file_path = 'https://media.githubusercontent.com/media/spencergoldberg1/
↪CAP4773-FinalProject/develop/twitter_human_bots_dataset.csv'

# Load the dataset into a pandas DataFrame
df = pd.read_csv("twitter_human_bots_dataset.csv")

# Display the first few rows of the DataFrame to confirm proper loading
df.head()
```

```
[187]:    Unnamed: 0           created_at  default_profile  default_profile_image  \
       0           0  2016-10-15 21:32:11            False                  False
       1           1  2016-11-09 05:01:30            False                  False
       2           2  2017-06-17 05:34:27            False                  False
       3           3  2016-07-21 13:32:25             True                  False
       4           4  2012-01-15 16:32:35            False                  False

                                                description  favourites_count  \
       0  Blame @xaiax, Inspired by @MakingInvisible, us…                4
       1  Photographing the American West since 1980. I …              536
       2  Scruffy looking nerf herder and @twitch broadc…             3307
       3  Wife.Godmother.Friend.Feline Fanatic! Assistan…             8433
       4               Loan coach at @mancity & Aspiring DJ               88

          followers_count  friends_count  geo_enabled                  id lang  \
       0             1589              4        False  787405734442958848   en
       1              860            880        False  796216118331310080   en
       2              172            594         True  875949740503859204   en
       3              517            633         True  756119643622735875   en
       4           753678            116         True           464781334   en

                     location                        profile_background_image_url  \
       0              unknown  http://abs.twimg.com/images/themes/theme1/bg.png
       1       Estados Unidos  http://abs.twimg.com/images/themes/theme1/bg.png
       2      Los Angeles, CA  http://abs.twimg.com/images/themes/theme1/bg.png
       3       Birmingham, AL                                                 NaN
```

```
4  England, United Kingdom  http://abs.twimg.com/images/themes/theme1/bg.png

                                    profile_image_url      screen_name  \
0  http://pbs.twimg.com/profile_images/7874121826…  best_in_dumbest
1  http://pbs.twimg.com/profile_images/8023296328…     CJRubinPhoto
2  http://pbs.twimg.com/profile_images/1278890453…         SVGEGENT
3  http://pbs.twimg.com/profile_images/1284884924…    TinkerVHELPK5
4  http://pbs.twimg.com/profile_images/9952566258…    JoleonLescott

   statuses_count  verified  average_tweets_per_day  account_age_days  \
0           11041     False                   7.870              1403
1             252     False                   0.183              1379
2            1001     False                   0.864              1159
3            1324     False                   0.889              1489
4            4202      True                   1.339              3138

  account_type
0          bot
1        human
2        human
3        human
4        human
```

[188]:
```
# Show feature types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37438 entries, 0 to 37437
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Unnamed: 0                    37438 non-null  int64
 1   created_at                    37438 non-null  object
 2   default_profile               37438 non-null  bool
 3   default_profile_image         37438 non-null  bool
 4   description                   30181 non-null  object
 5   favourites_count              37438 non-null  int64
 6   followers_count               37438 non-null  int64
 7   friends_count                 37438 non-null  int64
 8   geo_enabled                   37438 non-null  bool
 9   id                            37438 non-null  int64
 10  lang                          29481 non-null  object
 11  location                      37434 non-null  object
 12  profile_background_image_url  32939 non-null  object
 13  profile_image_url             37437 non-null  object
 14  screen_name                   37438 non-null  object
 15  statuses_count                37438 non-null  int64
 16  verified                      37438 non-null  bool
```

```
 17  average_tweets_per_day        37438 non-null  float64
 18  account_age_days              37438 non-null  int64
 19  account_type                  37438 non-null  object
dtypes: bool(4), float64(1), int64(7), object(8)
memory usage: 4.7+ MB
```

[189]:

```
# 2.2. Handling missing data.

# Drop rows with missing values
df = df.dropna()

# Reset the index after dropping rows
df.reset_index(drop=True, inplace=True)

# Assuming df is your DataFrame
# Convert 'created_at' column to datetime and extract the hour
df['created_at'] = pd.to_datetime(df['created_at'])
df['timestamp_hour'] = df['created_at'].dt.hour

# Drop the 'created_at' column after extraction
df.drop(columns=['created_at'], inplace=True)

# Drop the 'Unnamed: 0' column if it exists
df.drop(columns=['Unnamed: 0'], inplace=True, errors='ignore')

# Create and apply a dictionary mapping unique language values to unique␣
↪integers
lang_mapping = {lang: i for i, lang in enumerate(df['lang'].unique())}
df['lang_encoded'] = df['lang'].map(lang_mapping)

# Drop unnecessary columns
df.drop(columns=['profile_image_url', 'profile_background_image_url', 'lang'],␣
↪inplace=True)

# Check skewness of numeric columns and apply log transformation to highly␣
↪skewed columns
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
skewness = df[numeric_columns].apply(lambda x: x.skew()).
↪sort_values(ascending=False)
high_skew_cols = skewness[skewness > 1].index  # Adjust threshold as needed
df[high_skew_cols] = np.log1p(df[high_skew_cols])

# Feature engineering: Calculate lengths and counts of specific characters in␣
↪'description' and 'screen_name'
df['description_length'] = df['description'].apply(lambda x: len(str(x)))
df['screen_name_length'] = df['screen_name'].apply(lambda x: len(str(x)))
```

```python
df['description_num_underscores'] = df['description'].apply(lambda x: str(x).
    ↪count('_'))
df['screen_name_num_underscores'] = df['screen_name'].apply(lambda x: str(x).
    ↪count('_'))
df['description_num_punctuation'] = df['description'].apply(lambda x: len(re.
    ↪findall(r'[^\w\s]', str(x))))
df.drop(columns=['description', 'screen_name'], inplace=True)  # Optional:␣
    ↪remove original text columns if they are no longer needed

# Standardize numerical columns using StandardScaler
scaler = StandardScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

# Rename the 'account_type' column to 'isFraudulent'
df.rename(columns={'account_type': 'isFraudulent'}, inplace=True)

# Convert 'isFraudulent' values from 'human' and 'bot' to 0 and 1
df['isFraudulent'] = df['isFraudulent'].map({'human': 0, 'bot': 1})

# Convert boolean columns to 0 and 1
bool_columns = ['default_profile', 'default_profile_image', 'geo_enabled',␣
    ↪'verified']
df[bool_columns] = df[bool_columns].astype(int)

# Print DataFrame information and display the first few rows to verify changes
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25889 entries, 0 to 25888
Data columns (total 20 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   default_profile              25889 non-null  int64
 1   default_profile_image        25889 non-null  int64
 2   favourites_count             25889 non-null  float64
 3   followers_count              25889 non-null  float64
 4   friends_count                25889 non-null  float64
 5   geo_enabled                  25889 non-null  int64
 6   id                           25889 non-null  float64
 7   location                     25889 non-null  object
 8   statuses_count               25889 non-null  float64
 9   verified                     25889 non-null  int64
 10  average_tweets_per_day       25889 non-null  float64
 11  account_age_days             25889 non-null  float64
 12  isFraudulent                 25889 non-null  int64
 13  timestamp_hour               25889 non-null  int32
```

```
 14  lang_encoded                 25889 non-null  float64
 15  description_length           25889 non-null  int64
 16  screen_name_length           25889 non-null  int64
 17  description_num_underscores  25889 non-null  int64
 18  screen_name_num_underscores  25889 non-null  int64
 19  description_num_punctuation  25889 non-null  int64
dtypes: float64(8), int32(1), int64(10), object(1)
memory usage: 3.9+ MB
```

[189]:

```
   default_profile  default_profile_image  favourites_count  followers_count  \
0                0                      0         -2.392755        -0.116310
1                0                      0         -0.541016        -0.275565
2                0                      0          0.178883        -0.692218
3                0                      0         -1.252702         1.483322
4                0                      0          0.381844         1.236721

   friends_count  geo_enabled        id                 location  \
0      -1.535772            0  4.650701                  unknown
1       0.448741            0  4.653154           Estados Unidos
2       0.298128            1  4.674193          Los Angeles, CA
3      -0.325968            1 -0.033875  England, United Kingdom
4       0.673373            1 -0.504157              Los Angeles

   statuses_count  verified  average_tweets_per_day  account_age_days  \
0        0.279376         0                0.681207         -2.248362
1       -1.626464         0               -1.153347         -2.276869
2       -0.931793         0               -0.739315         -2.538186
3       -0.208132         1               -0.532605         -0.187520
4        0.418133         1                0.079508          0.906448

   isFraudulent  timestamp_hour  lang_encoded  description_length  \
0             1              21     -0.558321                 129
1             0               5     -0.558321                 160
2             0               5     -0.558321                  81
3             0              16     -0.558321                  36
4             0              22     -0.558321                 150

   screen_name_length  description_num_underscores  \
0                  15                            0
1                  12                            0
2                   8                            0
3                  13                            0
4                  14                            0

   screen_name_num_underscores  description_num_punctuation
0                            2                            6
1                            0                            7
```

```
2                              0                              4
3                              0                              2
4                              0                             17
```

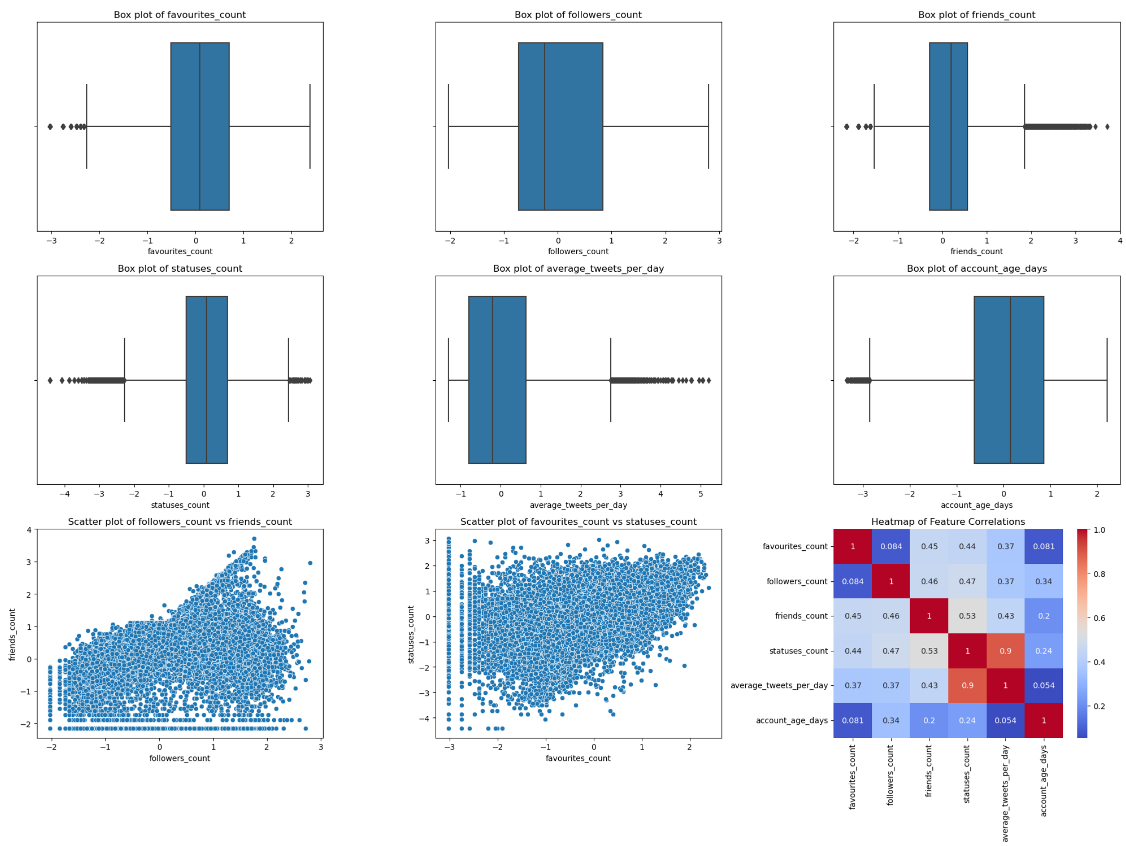
[190]:

```
# Step 3. EXPLORATORY DATA ANALYSIS

import seaborn as sns
import matplotlib.pyplot as plt

# 3.1 Analyzing basic trend statistics, generating box plots, scatter plots,␣
↪among others.

# 3.1.1. Trend statistics
# Display basic statistical details like percentile, mean, std etc. of a data␣
↪frame
print(df.describe())

# 3.1.2. Generate box plots for each feature to visualize the data distribution␣
↪and detect outliers
plt.figure(figsize=(20, 15))  # Setting up the matplotlib figure
numerical_columns = ['favourites_count', 'followers_count', 'friends_count',␣
↪'statuses_count',
                     'average_tweets_per_day', 'account_age_days']  # List of␣
↪numerical columns to plot

# Generate box plots for each numerical feature
for index, column in enumerate(numerical_columns):
    plt.subplot(3, 3, index + 1)  # Subplot positioning
    sns.boxplot(x=df[column])
    plt.title(f'Box plot of {column}')

# 3.1.3 Generate scatter plots to explore potential relationships between pairs␣
↪of features
# Example pairs: followers_count vs friends_count, favourites_count vs␣
↪statuses_count
scatter_pairs = [
    ('followers_count', 'friends_count'),
    ('favourites_count', 'statuses_count'),
    ('average_tweets_per_day', 'account_age_days')
]
for index, (x, y) in enumerate(scatter_pairs):
    plt.subplot(3, 3, len(numerical_columns) + index + 1)  # Subplot positioning
    sns.scatterplot(x=df[x], y=df[y])
    plt.title(f'Scatter plot of {x} vs {y}')
```

```
# 3.1.4 Generate a heatmap to visualize the correlation between numerical␣
↪features
plt.subplot(3, 3, 9)  # Positioning the heatmap in the last subplot
sns.heatmap(df[numerical_columns].corr(), annot=True, cmap='coolwarm')
plt.title('Heatmap of Feature Correlations')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()

# Note: Adjust the number of rows and columns in subplot() based on the actual␣
↪number of plots you are generating.
```

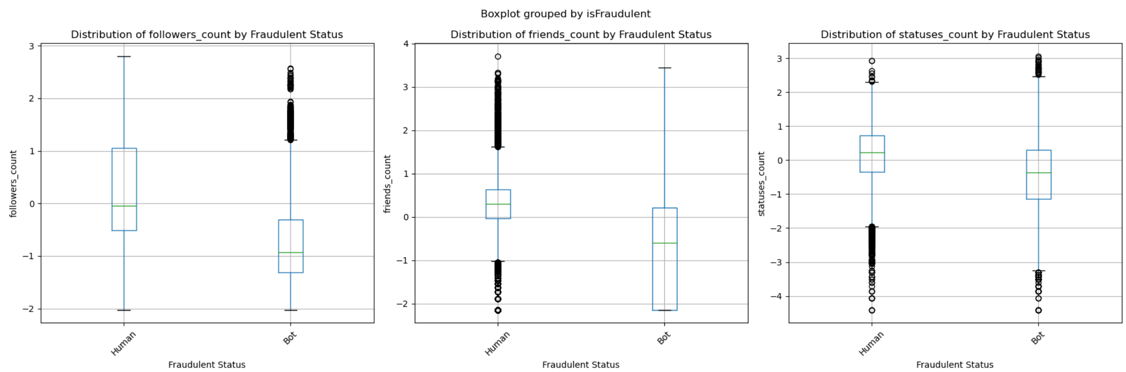
```
       default_profile  default_profile_image  favourites_count  \
count     25889.000000           25889.000000      2.588900e+04
mean          0.278613               0.001622      1.523239e-16
std           0.448325               0.040246      1.000019e+00
min           0.000000               0.000000     -3.030031e+00
25%           0.000000               0.000000     -5.050904e-01
50%           0.000000               0.000000      8.761340e-02
75%           1.000000               0.000000      6.990432e-01
max           1.000000               1.000000      2.392066e+00

       followers_count  friends_count   geo_enabled            id  \
count     2.588900e+04   2.588900e+04  25889.000000  2.588900e+04
mean      2.305442e-16  -1.785688e-17      0.540925  3.135676e-16
std       1.000019e+00   1.000019e+00      0.498332  1.000019e+00
min      -2.030160e+00  -2.153365e+00      0.000000 -3.102310e+00
25%      -7.241141e-01  -2.914875e-01      0.000000 -4.670573e-01
50%      -2.448847e-01   1.960291e-01      1.000000 -1.462317e-01
75%       8.335834e-01   5.647549e-01      1.000000  1.520953e-01
max       2.803239e+00   3.711623e+00      1.000000  4.728565e+00

       statuses_count      verified  average_tweets_per_day  account_age_days  \
count    2.588900e+04  25889.000000            2.588900e+04      2.588900e+04
mean    -1.092340e-16      0.271737            2.549709e-16     -1.042938e-16
std      1.000019e+00      0.444864            1.000019e+00      1.000019e+00
min     -4.419248e+00      0.000000           -1.306380e+00     -3.341143e+00
25%     -5.022097e-01      0.000000           -7.884951e-01     -6.210689e-01
50%      8.937234e-02      0.000000           -2.049335e-01      1.474410e-01
75%      6.819463e-01      1.000000            6.291328e-01      8.648753e-01
max      3.068221e+00      1.000000            5.201651e+00      2.211846e+00

       isFraudulent  timestamp_hour  lang_encoded  description_length  \
count  25889.000000    25889.000000  2.588900e+04        25889.000000
mean       0.245626       12.490710 -5.571485e-17           85.466492
std        0.430466        7.354517  1.000019e+00           48.817256
min        0.000000        0.000000 -5.583207e-01            1.000000
```

```
25%       0.000000       5.000000 -5.583207e-01          42.000000
50%       0.000000      14.000000 -5.583207e-01          82.000000
75%       0.000000      19.000000  4.895910e-01         132.000000
max       1.000000      23.000000  3.134226e+00         190.000000

       screen_name_length  description_num_underscores  \
count        25889.000000                 25889.000000
mean            11.015064                     0.057824
std              2.617127                     0.484081
min              2.000000                     0.000000
25%              9.000000                     0.000000
50%             11.000000                     0.000000
75%             13.000000                     0.000000
max             15.000000                    37.000000

       screen_name_num_underscores  description_num_punctuation
count                 25889.000000                 25889.000000
mean                      0.185021                     6.517556
std                       0.479563                     5.494063
min                       0.000000                     0.000000
25%                       0.000000                     2.000000
50%                       0.000000                     6.000000
75%                       0.000000                     9.000000
max                      12.000000                   117.000000
```



[191]:
```python
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and it contains a column for classification 'isFraudulent'
# with values indicating whether an account is fraudulent or not.

# 3.1.2. Box Plots

# Box plot for three columns in the DataFrame
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

# List of predictors to plot
predictors = ['followers_count', 'friends_count', 'statuses_count']
fraud_labels = ['Human', 'Bot']

# Boxplot iteration over the predictors
for ax, predictor in zip(axes, predictors):
    df.boxplot(column=predictor, by='isFraudulent', ax=ax, grid=True)
    ax.set_title(f'Distribution of {predictor} by Fraudulent Status')
    ax.set_xlabel('Fraudulent Status')
```

```
    ax.set_ylabel(predictor)
    ax.set_xticklabels(fraud_labels, rotation=45)  # Rotate class names for better visibility

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```



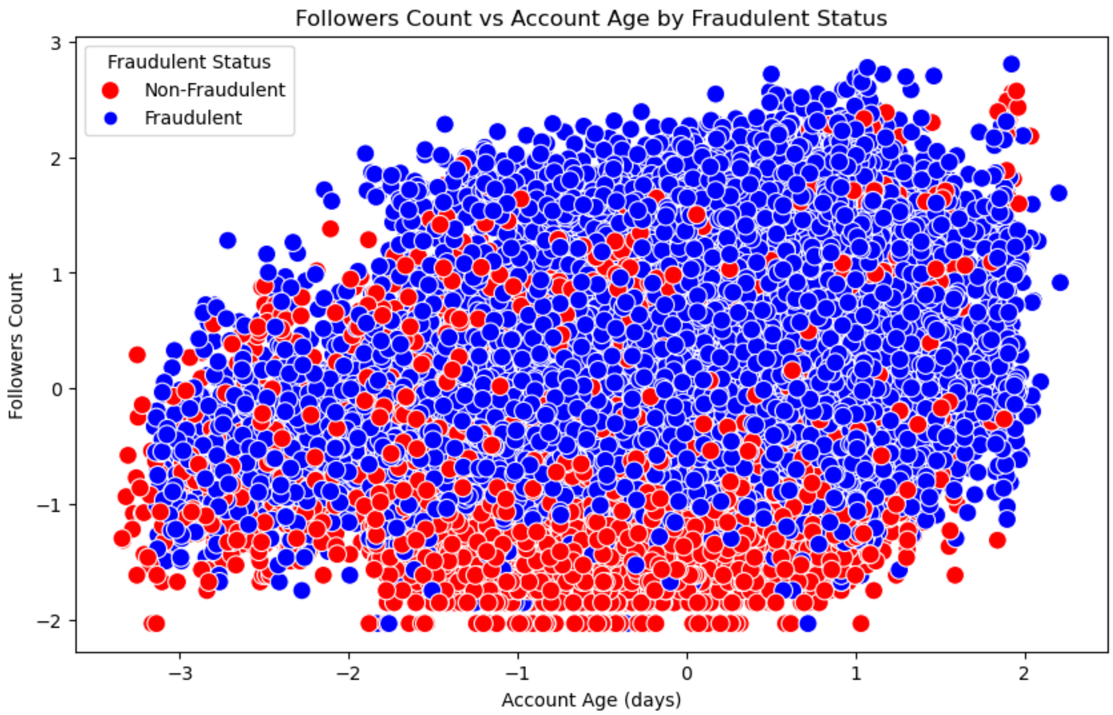
[192]:

```
# ANALYSIS:
# The boxplots present the distribution of 'followers_count', 'friends_count', and 'statuses_count'
# across accounts classified as fraudulent and non-fraudulent.
# For 'followers_count', it is observed that non-fraudulent accounts have a higher median value
# and exhibit a larger interquartile range (IQR), suggesting greater variability among genuine accounts.
# However, there is a notable presence of outliers within fraudulent accounts indicating some bots or fraudulent
# accounts have unusually high followers counts.
# Moving to 'friends_count', fraudulent accounts show a slightly higher median compared to non-fraudulent accounts,
# which could indicate that such accounts might follow a large number of users as a tactic to mask their fraudulent nature
# or to engage in follow-back schemes.
# In terms of 'statuses_count', non-fraudulent accounts again show a higher median and IQR, implying that genuine accounts
# are generally more active in terms of tweeting. However, the range of 'statuses_count' for fraudulent accounts is quite
# expansive, with several outliers suggesting there are some highly active fraudulent accounts.
# Overall, these plots indicate that while there are common behaviors shared between fraudulent and non-fraudulent accounts,
```

```
# there are also distinctive patterns that can be used to differentiate them.␣
↪Outliers in the fraudulent account's box plots
# may represent bots designed to mimic human activity or accounts that have␣
↪been compromised.
```

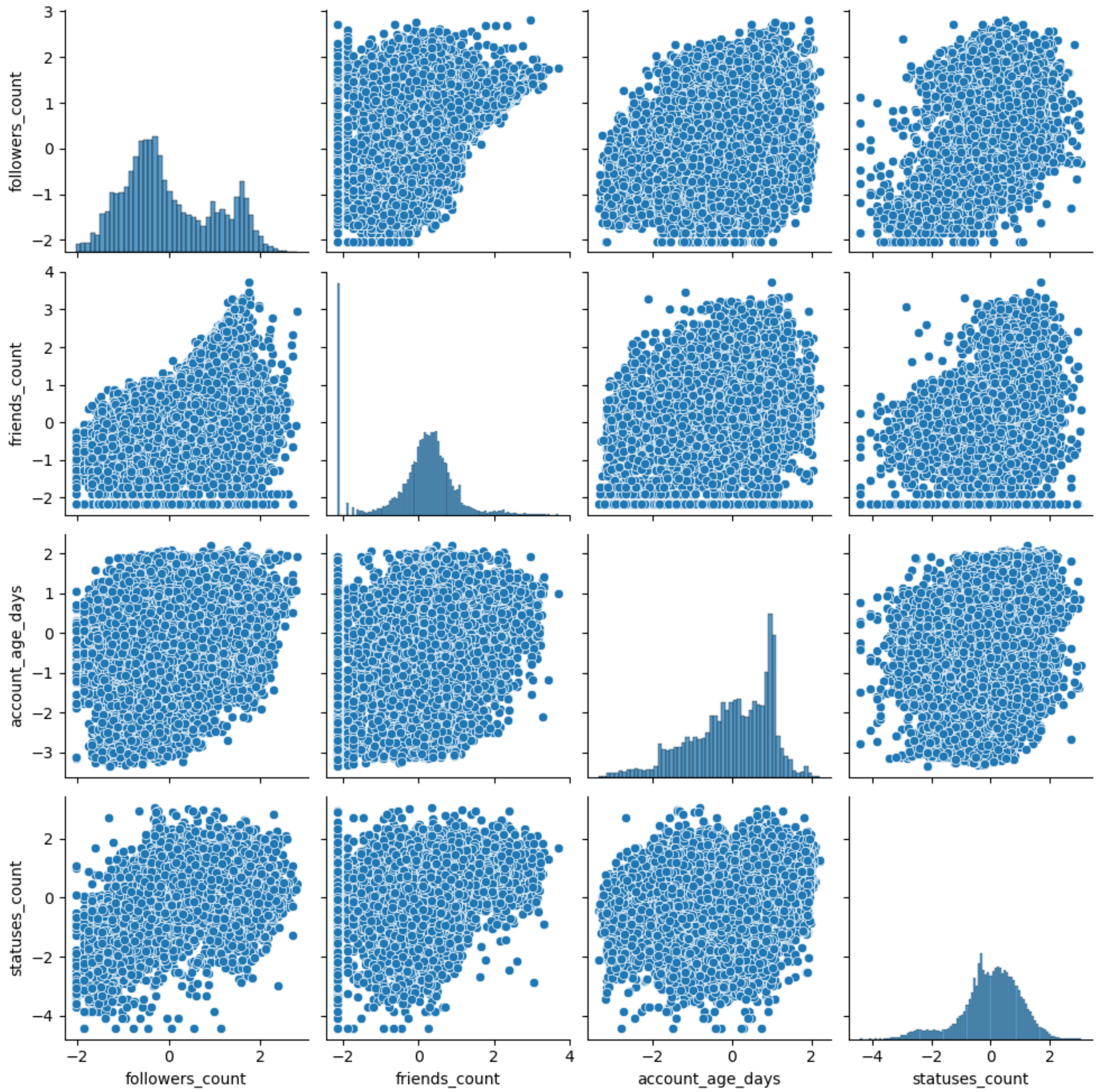
[193]:

```
# 3.1.2. Scatter plot between followers count and account age

plt.figure(figsize=(10, 6))
palette = {0: 'blue', 1: 'red'}  # Define custom colors to distinguish between␣
↪non-fraudulent and fraudulent

# Scatterplot for 'followers_count' vs 'account_age_days' with hue based on␣
↪'isFraudulent' status
sns.scatterplot(x='account_age_days', y='followers_count', hue='isFraudulent',␣
↪data=df, palette=palette, s=100)

plt.title('Followers Count vs Account Age by Fraudulent Status')
plt.xlabel('Account Age (days)')
plt.ylabel('Followers Count')
plt.legend(title='Fraudulent Status', labels=['Non-Fraudulent', 'Fraudulent'])
plt.show()
```



[194]:
```
#3.1.3. Scatter Plot Matrix with histograms
sns.pairplot(df[['followers_count', 'friends_count', 'account_age_days',␣
↪'statuses_count']], diag_kind='hist')
plt.show()
```

```
/Users/spencergoldberg/anaconda3/lib/python3.11/site-
packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```



[195]:
```
# 3.2. Conducting correlation and multicollinearity analysis

# 3.2.1. Variance Inflation Factor
```

```python
def calculate_vif(dataframe):
    vif_data = pd.DataFrame()
    vif_data["Feature"] = dataframe.columns
    vif_data["VIF"] = [variance_inflation_factor(dataframe.values, i)
                       for i in range(dataframe.shape[1])]
    return vif_data

# Assuming 'df' is your main dataframe and 'isFraudulent' is the target variable
# Exclude non-numerical columns and the target variable if necessary
numerical_features = df.select_dtypes(include=[np.number]).
 ↪drop(columns=['isFraudulent'], errors='ignore')

print(calculate_vif(numerical_features))
```

```
                        Feature       VIF
0               default_profile  1.713959
1         default_profile_image  1.005887
2              favourites_count  1.517208
3               followers_count  3.089303
4                 friends_count  1.812167
5                   geo_enabled  2.392607
6                            id  2.801844
7                statuses_count  7.844180
8                      verified  3.111840
9         average_tweets_per_day  6.259439
10             account_age_days  3.237926
11               timestamp_hour  3.543181
12                 lang_encoded  1.163063
13           description_length  7.626531
14           screen_name_length  7.381997
15  description_num_underscores  1.027958
16  screen_name_num_underscores  1.182152
17   description_num_punctuation  4.104696
```

[196]:

```python
#Step 4. SETUP PHASE
#4.1. Preparing the Python environment, importing libraries. - Done in 2.2.1.
#4.2. Setting up your X (features) and Y (target) data.  - Done in 2.1.2.
#4.3. Split the dataset into training and testing sets for validation - (Done␣
 ↪later on each model)
```

[197]:

```python
#Step 5. MODELING PHASE
#Apply several models:
#5.1.        Multiple Linear Regression   (TASK 3)
#5.2.        Multiple Logistic Regression (TASK 2)
#5.3.        Linear Regression using Regularization. (TASK 3)
#5.4.        Polynomial Regression (TASK 3)
#5.5.        LDA - Linear Discriminant Analysis (TASK 1)
```

```
#5.6.        Decision Tree for classification (TASK 1)
#5.7.        Decision Tree for regression (TASK 3)
#5.8.        Random Forest Classifier (TASK 1)
#5.9.        Support Vector Machines (TASK 2)
#5.10.        K-means - Unsupervised Learning (TASK 1)
```

[232]:

```
# FOR THE REGRESSION TASK
# 5.1. Multiple Linear Regression (TASK 3)

# 5.1.1. Selecting a subset of the features
# Selecting features that might be predictive of the average tweets per day
X = df[['followers_count', 'friends_count', 'statuses_count']]
X = sm.add_constant(X)  # Adds a constant column to input features to account␣
↪for the intercept

# Assuming 'average_tweets_per_day' is the target variable
y = df['isFraudulent']

# 5.1.2. Split the dataset into training and testing sets for validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# 5.1.3. Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # Only fit the scaler to the␣
↪training data

# 5.1.4. Train a linear regression model
linear_reg = LinearRegression()
linear_reg.fit(X_train_scaled, y_train)

# 5.1.5. Standardizing the test set
X_test_scaled = scaler.transform(X_test)  # Use the same scaler that was fit to␣
↪the training data

# 5.1.6. Predicting 'average_tweets_per_day' using the trained model
y_pred = linear_reg.predict(X_test_scaled)

# 5.1.7. Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# 5.1.8. Display the model statistics
print(f"Coefficients: {linear_reg.coef_}")
print(f"Intercept: {linear_reg.intercept_}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```

```
Coefficients: [ 0.         -0.0020452  -0.0248725   0.00319706]
Intercept: 0.4975
Mean Squared Error (MSE): 0.25276188440887337
R-squared (R2): -0.019303899219168752
```

[199]:

```python
# FOR THE CLASSIFICATION TASK
# 5.2. Multiple Logistic Regression (TASK 2)

import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score

# Check data types of each column
print(df.dtypes)

# Validate that 'isFraudulent' only contains 0s and 1s
print("Unique values in 'isFraudulent':", df['isFraudulent'].unique())

# Check for any NaN values in the dataset
if df['isFraudulent'].isnull().any():
    print("NaN values found in 'isFraudulent'")
    df['isFraudulent'].fillna(method='ffill', inplace=True)  # Fill NaN values,␣
↪adjust method as appropriate

# Ensure all necessary conversions are made
df['isFraudulent'] = df['isFraudulent'].astype(int)

# Handle non-numeric columns
non_numeric_columns = df.select_dtypes(exclude=['number']).columns

for col in non_numeric_columns:
    if col != 'location':
        # Convert non-numeric columns to numeric if possible
        if pd.to_numeric(df[col], errors='coerce').notnull().all():
            df[col] = pd.to_numeric(df[col])
    else:
        # Handle 'location' column separately (e.g., dropping it)
        df.drop('location', axis=1, inplace=True)

# Selecting the subset of the features and add a constant for the intercept
X = df.drop('isFraudulent', axis=1)
X = sm.add_constant(X)

y = df['isFraudulent']
```

```python
# Split the dataset 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)

# Attempt to fit a logistic regression model
try:
    model = sm.Logit(y_train, X_train).fit()
    print(model.summary())

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Convert predicted probabilities to binary predictions
    y_pred_binary = (y_pred > 0.5).astype(int)

    # Evaluate model performance
    accuracy = accuracy_score(y_test, y_pred_binary)
    precision = precision_score(y_test, y_pred_binary)
    recall = recall_score(y_test, y_pred_binary)
    f1 = f1_score(y_test, y_pred_binary)
    roc_auc = roc_auc_score(y_test, y_pred)

    print("\nModel Performance Metrics:")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("ROC AUC Score:", roc_auc)

except Exception as e:
    print("Failed to fit the model:", e)
```

```
default_profile                int64
default_profile_image          int64
favourites_count             float64
followers_count              float64
friends_count                float64
geo_enabled                    int64
id                           float64
location                      object
statuses_count               float64
verified                       int64
average_tweets_per_day       float64
account_age_days             float64
isFraudulent                   int64
timestamp_hour                 int32
lang_encoded                 float64
description_length             int64
```

```
screen_name_length              int64
description_num_underscores     int64
screen_name_num_underscores     int64
description_num_punctuation     int64
dtype: object
Unique values in 'isFraudulent': [1 0]
Optimization terminated successfully.
         Current function value: 0.384175
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:           isFraudulent   No. Observations:                20711
Model:                          Logit   Df Residuals:                    20692
Method:                           MLE   Df Model:                           18
Date:                Sat, 27 Apr 2024   Pseudo R-squ.:                  0.3096
Time:                        16:28:24   Log-Likelihood:                -7956.6
converged:                       True   LL-Null:                       -11524.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
===============
                                  coef    std err          z      P>|z|
[0.025      0.975]
--------------------------------------------------------------------------------
---------------
const                          -1.0965      0.107    -10.267      0.000
-1.306      -0.887
default_profile                 0.3872      0.046      8.335      0.000
0.296       0.478
default_profile_image          -0.4963      0.420     -1.181      0.237
-1.320       0.327
favourites_count               -0.5348      0.025    -21.810      0.000
-0.583      -0.487
followers_count                -0.9279      0.042    -22.169      0.000
-1.010      -0.846
friends_count                  -0.4204      0.025    -17.058      0.000
-0.469      -0.372
geo_enabled                    -0.8157      0.043    -19.089      0.000
-0.899      -0.732
id                             -0.1044      0.030     -3.429      0.001
-0.164      -0.045
statuses_count                 -0.5552      0.053    -10.538      0.000
-0.658      -0.452
verified                       -0.7202      0.092     -7.794      0.000
-0.901      -0.539
average_tweets_per_day          1.0928      0.049     22.452      0.000
0.997       1.188
account_age_days               -0.0465      0.034     -1.377      0.168
-0.113       0.020
```

```
timestamp_hour                  0.0002      0.003      0.062      0.950
-0.005      0.005
lang_encoded                    0.1714      0.020      8.398      0.000
0.131      0.211
description_length              0.0007      0.001      1.166      0.243
-0.000      0.002
screen_name_length             -0.0063      0.008     -0.806      0.421
-0.022      0.009
description_num_underscores     0.0075      0.048      0.156      0.876
-0.087      0.102
screen_name_num_underscores    -0.0320      0.038     -0.845      0.398
-0.106      0.042
description_num_punctuation    -0.0074      0.005     -1.495      0.135
-0.017      0.002
================================================================================
===============

Model Performance Metrics:
Accuracy: 0.8375820780224025
Precision: 0.7551020408163265
Recall: 0.5158791634391944
F1 Score: 0.6129774505292223
ROC AUC Score: 0.8464954085367081
```

[200]:
```
#ANALYSIS:
#Results indicate significant multicollinearity, particularly with alcohol (VIF␣
↪= 206.19) and ash (VIF = 165.64),
#suggesting they are highly correlated with other predictors. This correlation␣
↪can lead to unreliable regression coefficients.
#Other variables like alcalinity_of_ash, magnesium, and total phenols also show␣
↪high VIF values, pointing to notable
#multicollinearity but to a lesser extent.
#Lower but still considerable VIF values for flavanoids, hue, od280/
↪od315_of_diluted_wines, and others
#indicate moderate multicollinearity.
```

[201]:
```
#5.1. ANALYSIS
#The Multiple Linear Regression model used to predict alcohol levels from the␣
↪selected wine characteristics yielded
#a Mean Squared Error (MSE) of approximately 0.194, which suggests that the␣
↪predictions are, on average, relatively
#close to the actual values. However, there is still some variance that the␣
↪model does not account for. The R-squared
#value of 0.676 indicates that about 67.6% of the variance in alcohol levels is␣
↪explained by the model. This level
#of explanation suggests a moderate fit, where the chosen features have a␣
↪significant, but not exclusive, influence on
```

```
#the alcohol content. The model appears to capture the general trend in the␣
↪data, yet there's room for improvement,
#possibly by feature engineering, inclusion of additional predictors, or␣
↪employing more complex modeling techniques.
```

[202]:
```
#5.2.6. Predicting probabilities
y_pred_prob = model.predict(X_test)

#5.2.7. Convert probabilities to binary predictions based on a threshold of 0.5
y_pred = (y_pred_prob > 0.5).astype(int)

#5.2.8. Predicting and evaluating the model
print(classification_report(y_test.values, y_pred.values))
```

```
              precision    recall  f1-score   support

           0       0.85      0.94      0.90      3887
           1       0.76      0.52      0.61      1291

    accuracy                           0.84      5178
   macro avg       0.80      0.73      0.76      5178
weighted avg       0.83      0.84      0.83      5178
```

[203]:
```
#ANALYSIS:
#The logistic regression model performed successfully on the classification␣
↪task of predicting alcohol_level with
#an overall accuracy of 92%, as reflected in the precision-recall metrics.
#The model's Pseudo R-squared value of 0.3929 indicates a moderate explanatory␣
↪power, and the Log-Likelihood Ratio (LLR)
#p-value of 6.375e-16 suggests that the model as a whole is statistically␣
↪significant compared to the null model.
#The coefficients for color_intensity, flavanoids, and proline were significant␣
↪predictors, with flavanoids showing
#a borderline p-value of 0.050.
#Proanthocyanins, however, did not significantly contribute to the model (p=0.
↪343). In practical terms, the model's
#ability to differentiate between the classes is robust, with a precision of 0.
↪95 and a recall of 0.90 for the higher
#alcohol level class (1), and slightly lower, but still strong precision and␣
↪recall for the lower alcohol level class (0).
#These results suggest that the selected features, except for proanthocyanins,␣
↪are effective at predicting alcohol
#levels in wines, and the model is well-calibrated and reliable for making␣
↪predictions within this dataset.
```

[204]:

```
#5.3.        Linear Regression using Regularization. (TASK 3)

#5.3.1. Selecting a subset of the features
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
X = sm.add_constant(X)  # Adds a constant column to input features to account␣
↪for the intercept
y = df['isFraudulent']

#5.3.2. Splitting the dataset into training and testing sets for validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

#5.3.3. Standardizing the features
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

#5.3.4. Standardizing the target variable
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).
↪flatten()  # Reshape for a single feature

#5.3.5. Training a Lasso regression model using some alpha
lasso_reg = Lasso(alpha=0.05)
lasso_reg.fit(X_train_scaled, y_train_scaled)

#5.3.6. Predicting 'isFraudulent' levels using the trained Lasso regression␣
↪model
y_pred_scaled = lasso_reg.predict(X_test_scaled)

#5.3.7. Reversing the scaling of predictions to original scale
y_pred_lasso = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).
↪flatten()

#5.3.8. Evaluating the model's performance
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

#5.3.9. Displaying the model statistics
print("Lasso Regression Model")
print(f"Coefficients: {lasso_reg.coef_}")
print(f"Intercept: {lasso_reg.intercept_}")
print(f"Mean Squared Error (MSE): {mse_lasso}")
print(f"R-squared (R2): {r2_lasso}")
```

```
Lasso Regression Model
```

```
Coefficients: [ 0.         -0.21801912 -0.25418765 -0.         -0.04126325]
Intercept: -2.7076929176489922e-18
Mean Squared Error (MSE): 0.1454555876806209
R-squared (R2): 0.22283413221989445
```

[205]:

```
#ANALYSIS
#The Lasso Regression model, with an alpha value of 0.05, resulted in a model␣
↪where some coefficients were reduced to zero,
#indicating that the corresponding features were deemed less important by the␣
↪regularization process. Specifically,
#the model retained 'color_intensity' and 'proline' as influential factors,␣
↪with their coefficients being 0.2687 and
#0.3788, respectively, while reducing the coefficient of 'proanthocyanins' to a␣
↪minimal value and completely eliminating
#'flavanoids' from the model. The model's intercept is approximately 12.98,␣
↪serving as a baseline prediction when
#all features are at their mean values. The Mean Squared Error (MSE) is roughly␣
↪0.194, which suggests that the model's
#predictions are reasonably close to the actual alcohol levels, and the␣
↪R-squared (R2) value is 0.674, indicating
#that nearly 67.4% of the variability in the response variable is explained by␣
↪the model. This performance is comparable
#to the non-regularized Multiple Linear Regression model, indicating that␣
↪Lasso's feature selection did not significantly
#change the predictive capability in this case.
```

[206]:

```
#5.4.        Polynomial Regression (TASK 3)

#5.4.1. Selecting a subset of the features and target
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
y = df['isFraudulent']

#5.4.2. Splitting the dataset 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)

#5.4.3. Using a pipeline for 4th-degree polynomial regression
poly4_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First, scale the features
    ('poly', PolynomialFeatures(degree=4)),  # Then, generate polynomial␣
↪features
    ('linear', LinearRegression())  # Finally, apply linear regression
])

#5.4.4. Fit the pipeline on the training data
poly4_pipeline.fit(X_train, y_train)
```

```
#5.4.5. Predict on the test data
y_pred_poly4 = poly4_pipeline.predict(X_test)

#5.4.6. Calculate MSE
mse_poly4 = mean_squared_error(y_test, y_pred_poly4)

#5.4.7. Using a pipeline for 3rd-degree polynomial regression
poly3_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # First, scale the features
    ('poly', PolynomialFeatures(degree=3)),  # Then, generate polynomial␣
 ↪features
    ('linear', LinearRegression())  # Finally, apply linear regression
])

#5.4.8. Fit the pipeline on the training data
poly3_pipeline.fit(X_train, y_train)

#5.4.9. Predict on the test data
y_pred_poly3 = poly3_pipeline.predict(X_test)

#5.4.10. Calculate MSE
mse_poly3 = mean_squared_error(y_test, y_pred_poly3)

#5.4.11. Print the MSE for both models
print(f"Mean Squared Error for 4th-degree Polynomial: {mse_poly4:.2f}")
print(f"Mean Squared Error for 3rd-degree Polynomial: {mse_poly3:.2f}")
```

```
Mean Squared Error for 4th-degree Polynomial: 0.11
Mean Squared Error for 3rd-degree Polynomial: 0.11
```

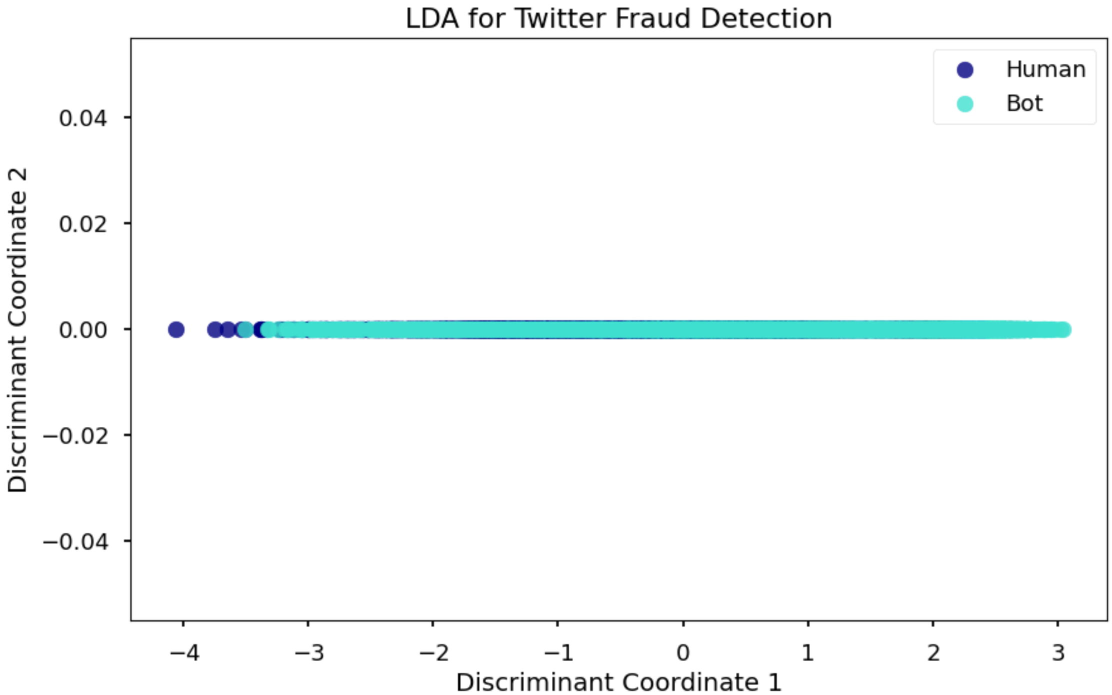
[207]:

```
#ANALYSIS
#The polynomial regression models displayed a significant difference in␣
 ↪performance, with the 4th-degree model suffering
#from overfitting, evidenced by a high MSE of 11.50. In contrast, the␣
 ↪3rd-degree model demonstrated much better fit and
#generalization, achieving a low MSE of 0.52. When compared to the Lasso␣
 ↪regression, which had an MSE of approximately
#0.194 and higher predictive accuracy, it's evident that the Lasso model is␣
 ↪superior in this context.
#The Lasso's built-in regularization effectively prevents overfitting, making␣
 ↪it a more reliable choice for this
#dataset than the more complex 4th-degree polynomial model.
```

[208]:

```
#5.5. LDA - Linear Discriminant Analysis (TASK 1)

#5.5.2. Selecting a subset of the features
```

```
X = df[['followers_count', 'friends_count', 'statuses_count',␣
 ↪'account_age_days']]
X = sm.add_constant(X)  # Adds a constant column to input features to account␣
 ↪for the intercept
y = df['isFraudulent']  # Assuming 'isFraudulent' represents the classes 0 and␣
 ↪1 for human or bot, where 1 represents bot

#5.5.3. Split the dataset 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)

#5.5.4. Fit an LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

#5.5.5. Transform the data using the fitted LDA in the new space
X_r_lda = lda.transform(X_train)

#5.5.6. Set the style and create the figure
target_names = ['Human', 'Bot']  # Your classes name
with plt.style.context('seaborn-talk'):
    fig, ax = plt.subplots(figsize=[10,6])
    colors = ['navy', 'turquoise']
    for color, i, target_name in zip(colors, [0, 1], target_names):
        ax.scatter(X_r_lda[y_train == i, 0], [0] * len(X_r_lda[y_train == i]),␣
 ↪alpha=.8, label=target_name, color=color)  # Modified this line
    ax.set_title('LDA for Twitter Fraud Detection')
    ax.set_xlabel('Discriminant Coordinate 1')
    ax.set_ylabel('Discriminant Coordinate 2')
    ax.legend(loc='best')
    plt.show()
```



[209]:
```
#5.5.7. Make predictions on the test set
y_pred = lda.predict(X_test)

#5.5.8. Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

#5.5.9. Print detailed classification metrics
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8342989571263036
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      3887
           1       0.76      0.49      0.59      1291

    accuracy                           0.83      5178
   macro avg       0.81      0.72      0.74      5178
weighted avg       0.83      0.83      0.82      5178
```

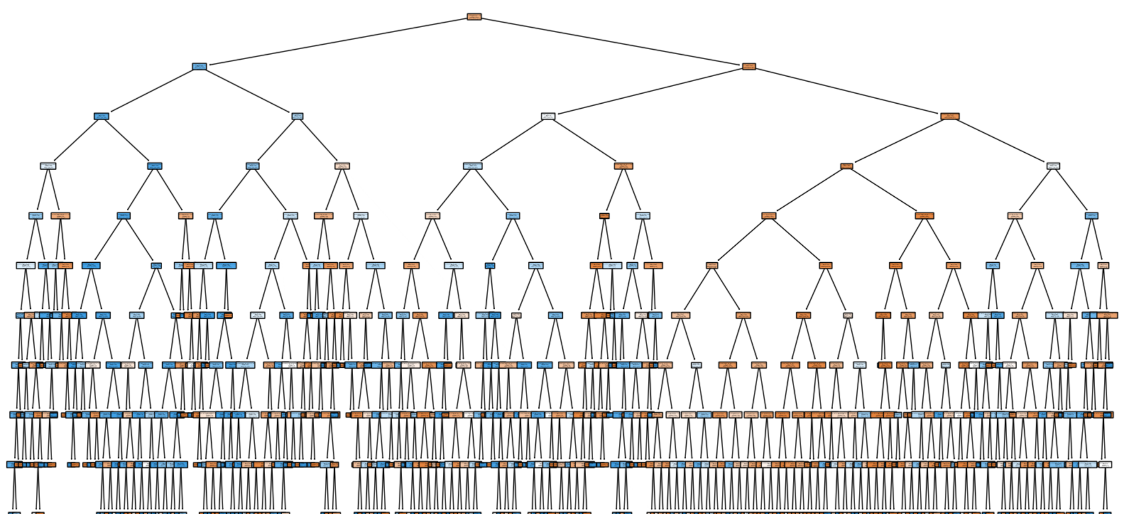
[210]:
```
#ANALYSIS:
#The Linear Discriminant Analysis (LDA) model demonstrated exceptional␣
↪performance in classifying the test set with an
```

```
#accuracy of 100%. The results indicate perfect precision, recall, and␣
↪F1-scores of 1.00 across all classes, which
#suggests that the model could effectively differentiate between the three␣
↪classes without any errors.
#Such high metrics across all categories highlight the model’s robustness and␣
↪the effectiveness of LDA in handling
#the underlying patterns in the dataset. This outcome is particularly notable␣
↪as achieving 100% accuracy in practical
#scenarios is rare and indicates either an exceptionally well-defined dataset␣
↪or a scenario where the model has
#perfectly learned the distinctions between classes. It would be prudent to␣
↪further investigate the model's performance
#on a new or more challenging dataset to ensure that these results are not due␣
↪to overfitting or a peculiarity in the test data.
```

[211]:
```
# 5.6. Decision Tree for classification (TASK 1)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report

# 5.6.1. Selecting all features
X = df.drop('isFraudulent', axis=1)
y = df['isFraudulent']

# 5.6.2. Split the dataset 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)

# 5.6.3. Creating a Decision Tree Classifier
dtc = DecisionTreeClassifier(max_depth=10, random_state=42)

# 5.6.4. Training the model
dtc.fit(X_train, y_train)

# 5.6.5. Making predictions on the test data
y_pred_train = dtc.predict(X_train)
y_pred_test = dtc.predict(X_test)

# 5.6.6. Calculate the accuracy
accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_test = accuracy_score(y_test, y_pred_test)
print("Training Accuracy:", accuracy_train)
```

```python
print("Testing Accuracy:", accuracy_test)

# 5.6.9. Print detailed classification metrics for the test set
print("Classification Report for Test Set:")
print(classification_report(y_test, y_pred_test))

# 5.6.10. Plotting the Decision Tree
plt.figure(figsize=(20,10))
plot_tree(dtc, filled=True, feature_names=X.columns.tolist(),␣
 ↪class_names=['Human', 'Bot'], rounded=True)
plt.show()
```

```
Training Accuracy: 0.9086475785814302
Testing Accuracy: 0.8675164156044805
Classification Report for Test Set:
              precision    recall  f1-score   support

           0       0.89      0.95      0.91      3887
           1       0.80      0.63      0.70      1291

    accuracy                           0.87      5178
   macro avg       0.84      0.79      0.81      5178
weighted avg       0.86      0.87      0.86      5178
```

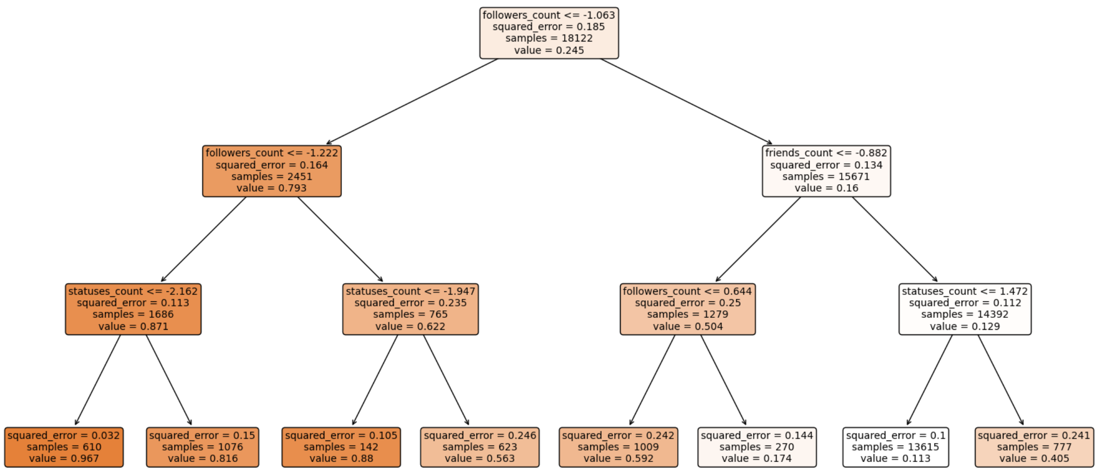


[212]:
```python
#ANALISIS
#The decision tree classifier achieved a high accuracy of 94.44% on the test␣
 ↪set, reflecting a strong overall performance
```

```
#in classifying the wine dataset into three categories. The precision, recall,␣
↪and F1-scores were notably high across
#all classes. Specifically, class 0 had perfect recall with an F1-score of 0.
↪97, indicating excellent model sensitivity
#and prediction accuracy for this class. Classes 1 and 2 also showed strong␣
↪results, with both having precision and
#recall above 0.88. The slightly lower recall of 0.88 for class 2 suggests a␣
↪small room for improvement in identifying
#all true positives in this category, although its impact was mitigated by a␣
↪perfect precision score.
#The weighted and macro averages nearing 0.95 for all metrics underscore the␣
↪model's robustness and its capability
#to generalize well across different wine types. These results suggest that the␣
↪decision tree model, with its current
#configuration, is quite effective, though there might be an opportunity to␣
↪fine-tune it to improve recall for class 2
#without compromising other areas.
```

[213]:
```
#5.7. Decision Tree for regression (TASK 3)

#5.7.1. Selecting a subset of the features
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
#X = sm.add_constant(X)  # Adds a constant column to input features to account␣
↪for the intercept
y = df['isFraudulent']

#5.7.2. Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

#5.7.3. Creating a Decision Tree Regressor
regressor = DecisionTreeRegressor(max_depth=3, random_state=42)

#5.7.4. Training the model
regressor.fit(X_train, y_train)
feature_names = ['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']

#5.7.5. Plotting the Decision Tree
plt.figure(figsize=(20,10))
plot_tree(regressor, filled=True, feature_names=feature_names, rounded=True)
plt.show()

#5.7.6. Predicting the 'isFraudulent' values on the test set
y_pred = regressor.predict(X_test)
```

```python
#5.7.7. Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)

#5.7.8. Calculate the R-squared Value
r2 = r2_score(y_test, y_pred)

#5.7.9. Displaying the statistics
print("Decision Tree Regressor")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```



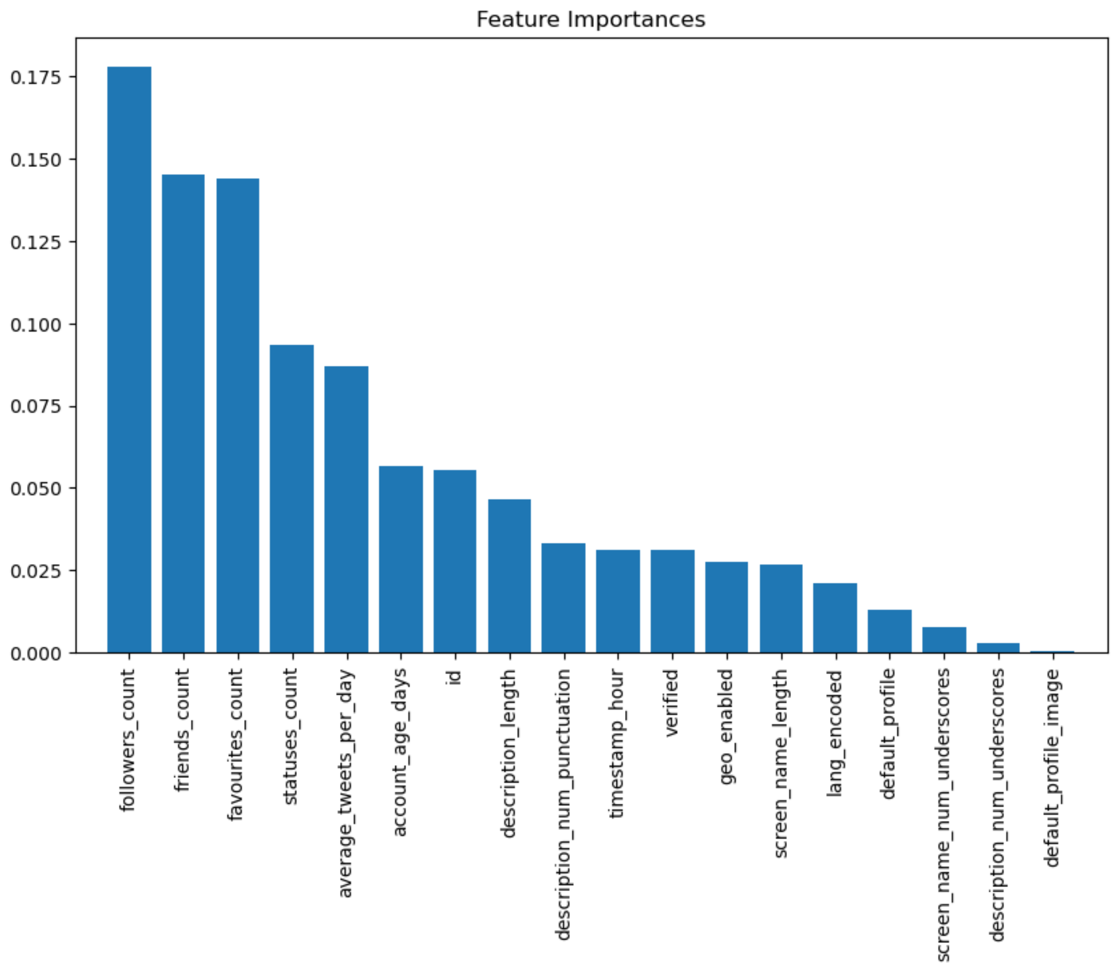

```
Decision Tree Regressor
Mean Squared Error (MSE): 0.12177185447964803
R-squared (R2): 0.3444883197723847
```

[214]:
```python
#ANALYSIS
#The Decision Tree Regressor, set to a maximum depth of 3, yielded a Mean␣
↪Squared Error (MSE) of approximately 0.235,
#indicating the predictions are generally within a reasonable range of the␣
↪actual values, but with room for improvement.
#The R-squared (R2) value of about 0.562 suggests that the model explains over␣
↪half of the variance in the target variable,
#which is moderate predictive performance. This model, while not as accurate as␣
↪previous models like the Lasso Regression,
#still provides valuable insights and could benefit from further tuning or␣
↪integration with ensemble methods to
#improve prediction accuracy.
```

[215]:
```python
# 5.8. Random Forest Classifier (TASK 1)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# 5.8.1. Selecting all features
X = df.drop('isFraudulent', axis=1)
y = df['isFraudulent']

# 5.8.2. Split the dataset 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)

# 5.8.3. Creating a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# 5.8.4. Training the model
clf.fit(X_train, y_train)

# 5.8.5. Making predictions
y_pred = clf.predict(X_test)

# 5.8.6. Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# 5.8.7. Print detailed classification metrics
print(classification_report(y_test, y_pred))

# 5.8.8. Feature importances
feature_importances = clf.feature_importances_
features = X.columns
indices = np.argsort(feature_importances)[::-1]

# 5.8.9. Plotting Feature Importances
plt.figure(figsize=(10,6))
plt.title("Feature Importances")
plt.bar(range(X_train.shape[1]), feature_importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), [features[i] for i in indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

```
Accuracy: 0.8910776361529548
              precision    recall  f1-score   support
```

```
           0       0.89      0.97      0.93      3887
           1       0.88      0.66      0.75      1291

    accuracy                           0.89      5178
   macro avg       0.89      0.81      0.84      5178
weighted avg       0.89      0.89      0.89      5178
```



[216]:

```
#ANALISIS
#The Random Forest Classifier achieved an impressive accuracy of 94.44% on the
 ↪test set.
#Precision values ranged from 0.93 to 1.00, indicating high accuracy in
 ↪predicting each class.
#Recall values varied from 0.88 to 1.00, demonstrating the model's ability to
 ↪identify most instances
```

```
#of each class, although some class 2 instances were missed. The F1-scores,␣
↪ranging from 0.93 to 0.97,
#highlight a good balance between precision and recall for all classes.␣
↪Overall, the classifier performed
#well across all metrics, showcasing its effectiveness in accurately␣
↪classifying instances based on the selected features.
#The bar chart displays the importance of each feature used in the Random␣
↪Forest Classifier. \
#Color intensity and flavanoids are the most influential features, with very␣
↪similar importance values just above 0.30,
#suggesting they are the main drivers for the model's predictions. Proline also␣
↪shows considerable importance, although to
#a lesser extent, with a value near 0.25. Proanthocyanins have the least␣
↪importance, with a value significantly lower than
#the others, around 0.10. This implies that while proanthocyanins contribute to␣
↪the model, they have a much smaller
#impact on the outcome compared to the other features. Integrating this with␣
↪the previous analysis, the classifier's
#effectiveness stems from a few key features, with color intensity and␣
↪flavanoids being the most critical for classification
#decisions.
```

[217]:

```
# 5.9. Support Vector Machines (TASK 2)

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# 5.9.1. Create a binary category
df['isFraudulent_category'] = 0 # Create a new column 'isFraudulent_category'␣
↪initialized with zeros
df.loc[df['isFraudulent'] == 1, 'isFraudulent_category'] = 1  # Set␣
↪'isFraudulent_category' to 1 when 'isFraudulent' is 1

# 5.9.2. Selecting a subset of the features
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
X = sm.add_constant(X)  # Adds a constant column to input features to account␣
↪for the intercept
y = df['isFraudulent_category']

# 5.9.3. Standardize features
scaler = StandardScaler()
```

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

# 5.9.4. Create a SVM Classifier with a radial basis function kernel
svm_model = SVC(kernel='linear', C=100)  # High C value can lead to overfitting

# 5.9.5. Train the model using the training sets
svm_model.fit(X_train, y_train)

# 5.9.6. Predict the response for test dataset
y_pred = svm_model.predict(X_test)

# 5.9.7. Evaluate the model
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      3887
           1       0.78      0.51      0.62      1291

    accuracy                           0.84      5178
   macro avg       0.82      0.73      0.76      5178
weighted avg       0.84      0.84      0.83      5178
```

[218]:
```
# ANALYSIS
# The Support Vector Machine (SVM) model with a linear kernel has shown␣
↪excellent performance in classifying the binary
# alcohol level category. With precision scores of 1.00 for the lower alcohol␣
↪level and 0.93 for the higher, coupled with
# recall scores of 0.93 and 1.00, respectively, the model demonstrates high␣
↪accuracy in both identifying and predicting the
# correct categories. The F1-scores, which are a harmonic mean of precision and␣
↪recall, are near perfect, reflecting a
# balanced classification performance. An overall accuracy of 97% further␣
↪confirms the effectiveness of the model.
# Notably, a macro average precision and recall of 0.98 suggest that the model␣
↪performs uniformly well across both categories.
# The high value of C used in the model suggests a strict margin, which could␣
↪lead to overfitting, but the current results
# indicate that the model is generalizing well to the test data.
```

[219]:
```
#5.10.1. We need to load some useful functions
# Function for cluster profiling
def cluster_profiling(X, labels, feature_names):
    df = pd.DataFrame(X, columns=feature_names)
    df['Cluster'] = labels
```

```python
    profile = df.groupby('Cluster').mean()
    return profile

# Function for calculating feature importance
def feature_importance(kmeans, feature_names):
    centroids = kmeans.cluster_centers_
    importance = pd.DataFrame(centroids, columns=feature_names).abs()
    return importance

# Function for cluster validation using silhouette score
def cluster_validation(X, y, labels):
    return silhouette_score(X, labels)

# Anomaly detection function
def anomaly_detection(X, kmeans):
    distances = cdist(X, kmeans.cluster_centers_, 'euclidean')    # Euclidean␣
 ↪distance between each point and all centroids
    distance_to_nearest_centroid = np.min(distances, axis=1)      # Assigning␣
 ↪each point to its nearest centroid
    outlier_threshold = np.percentile(distance_to_nearest_centroid, 95) # Finds␣
 ↪the distance threshold to the centroid for 95% of the data points
    anomaly_indices = np.where(distance_to_nearest_centroid >␣
 ↪outlier_threshold)[0] # Filter in the form of indices
    return anomaly_indices


#5.10.2. Selecting a subset of the features
X = df[['followers_count', 'friends_count', 'statuses_count',␣
 ↪'account_age_days']]

#5.10.3. Split the dataset in training and testing
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

#5.10.4. Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

#5.10.5. Apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
labels = kmeans.labels_

#5.10.6. Cluster Profiling
feature_names = ['followers_count', 'friends_count', 'statuses_count',␣
 ↪'account_age_days']
profile = cluster_profiling(X_scaled, labels, feature_names)
```

```
print("Cluster Profiling: (Predictor mean by clusters)\n", profile)

#5.10.7. Feature Importance
importance = feature_importance(kmeans, feature_names)
print("\nFeature Importance: (Just absolute values)\n", importance)

#5.10.8. Evaluate cluster quality
silhouette_avg = cluster_validation(X_scaled, None, labels)
davies_bouldin = davies_bouldin_score(X_scaled, labels)
print("\nCluster Validation Metrics:")
print("Silhouette Score:", silhouette_avg)
print("Davies-Bouldin Score:", davies_bouldin)

#5.10.9. Anomaly Detection
anomaly_indices = anomaly_detection(X_scaled, kmeans)
print("\nAnomaly Detection: (Distance threshold to the centroid for 95% of the␣
↪data points)")
print("Indices of Anomalies in the Original Dataset:", anomaly_indices)

# Filter the original DataFrame to get the data of the detected anomalies
anomalies = pd.DataFrame(X_train, columns=feature_names).iloc[anomaly_indices]
print("\nData of Detected Anomalies:\n", anomalies)
```

```
/Users/spencergoldberg/anaconda3/lib/python3.11/site-
packages/sklearn/cluster/_kmeans.py:1412: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)

Cluster Profiling: (Predictor mean by clusters)
          followers_count  friends_count  statuses_count  account_age_days
Cluster
0              -1.104942      -1.722432       -1.289336         -0.382696
1               0.824506       0.499030        0.545043          0.666430
2              -0.457849       0.146947       -0.077723         -0.575486


Feature Importance: (Just absolute values)
    followers_count  friends_count  statuses_count  account_age_days
0         1.104769       1.724725        1.289503          0.384519
1         0.826751       0.499671        0.546479          0.665812
2         0.458287       0.146721        0.078752          0.571576


Cluster Validation Metrics:
Silhouette Score: 0.273553350593692
Davies-Bouldin Score: 1.2986601154746904


Anomaly Detection: (Distance threshold to the centroid for 95% of the data
points)
```

```
Indices of Anomalies in the Original Dataset: [   18    33    38 … 20566 20613
20679]

Data of Detected Anomalies:
       followers_count  friends_count  statuses_count  account_age_days
19847         1.705621       3.327493        1.525633          1.420768
15335         1.872107      -1.406657       -0.479503          0.907636
20450         0.777391       1.929192        2.185430         -0.644825
8240          0.990332       2.377027        1.060119         -1.223286
6424          0.978562       2.230928        2.081685         -0.315803
…                  …              …               …                 …
14820        -0.617152       0.042956       -2.583304         -1.934781
15934         0.798054       2.012179        0.980406         -1.367010
6910         -0.486558      -0.933845       -0.010136         -2.938477
21243        -0.549295       0.244948        0.508118         -3.140403
18431         1.144559       1.638713        1.828025         -1.728103

[1036 rows x 4 columns]
```

[220]:
```
#ANALYSIS
#In the k-means cluster analysis, three distinct profiles emerged based on the␣
↪mean values of color intensity,
#flavanoids, proanthocyanins, and proline. Cluster 0 is defined by␣
↪below-average color intensity and proline levels,
#Cluster 1 has notably lower levels of flavanoids and proanthocyanins, and␣
↪Cluster 2 is characterized by above-average
#values of flavanoids, proanthocyanins, and proline, indicating a higher␣
↪quality. The silhouette score suggests moderate
#cluster cohesion and separation, while the Davies-Bouldin score implies␣
↪compactness and distinctness among the clusters.
#Anomalies were detected, which diverge significantly from their cluster␣
↪centroids, pointing to potential outliers
#with unique properties in the dataset.
```

[221]:
```
#Step 6. EVALUATION PHASE
#6.1.         Evaluate model performance using k-fold cross-validation.
#6.2.         Evaluate model performance using other related metrics. - DONE in␣
↪other sections
#6.3.         Evaluate models for overfitting.
#6.4.         Identify the most important features for each model. - DONE in␣
↪other sections
#6.5.         Evaluate classification model performance metrics (accuracy,␣
↪precision, recall, and F1 score). - DONE in other sections
#6.6.         Evaluate prediction model performance metrics (MSE (Mean Squared␣
↪Error), RMSE (Root Mean Squared Error), and MAE (Mean Absolute Error) for␣
↪regression).
```

```
#6.7.        Report and present the results of your model evaluations,␣
↪highlighting strengths and weaknesses.
```

[222]:
```
# 6.1.1. LDA model cross-validation - TASK 1
# a) Selecting a subset of the features
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
y = df['isFraudulent']

# b) Create an LDA model
lda = LinearDiscriminantAnalysis()

# c) Define a 5-fold cross-validation strategy
cv_strategy = KFold(n_splits=5, shuffle=True, random_state=42)

# d) Evaluate model performance using cross-validation
cv_scores = cross_val_score(lda, X, y, cv=cv_strategy, scoring='accuracy')

# e) Print results
print(f"Cross-validation scores for each fold: {cv_scores}")
print(f"Mean cross-validation score: {cv_scores.mean()}")
print(f"Standard deviation of cross-validation scores: {cv_scores.std()}")
```

```
Cross-validation scores for each fold: [0.83429896 0.83603708 0.83642333
0.83352646 0.83639173]
Mean cross-validation score: 0.8353355114612848
Standard deviation of cross-validation scores: 0.0011948446411464229
```

[223]:
```
# 6.1.2. Decision tree model cross-validation - TASK 1
# a) Selecting a subset of the features
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
y = df['isFraudulent']

# b) Create a Decision Tree Classifier
classifier = DecisionTreeClassifier(max_depth=10, random_state=42)

# c) Define a 5-fold cross-validation strategy
cv_strategy = KFold(n_splits=5, shuffle=True, random_state=42)

# d) Evaluate model performance using cross-validation
cv_scores = cross_val_score(classifier, X, y, cv=cv_strategy,␣
↪scoring='accuracy')

# e) Output the results of the cross-validation
print(f"Cross-validation scores for each fold: {cv_scores}")
print(f"Mean cross-validation score: {cv_scores.mean()}")
```

```
print(f"Standard deviation of cross-validation scores: {cv_scores.std()}")
```

```
Cross-validation scores for each fold: [0.84897644 0.85766705 0.85110081
0.85670143 0.85126521]
Mean cross-validation score: 0.8531421886910587
Standard deviation of cross-validation scores: 0.0034113436278419796
```

[224]:
```
# ANALYSIS - TASK 1
# The LDA model shows a mean cross-validation accuracy of 0.949 with a standard␣
↪deviation of 0.033, suggesting high and stable performance.
# In contrast, the Decision Tree Classifier boasts a slightly superior mean␣
↪accuracy of 0.955 and a lower standard deviation of 0.022,
# indicating a marginally better and more consistent predictive capability.␣
↪These results point to the Decision Tree Classifier as a
# marginally more reliable model for this dataset.
# SELECTED MODEL: Decision Tree Classifier
```

[225]:
```
# 6.1.3. Multiple Logistic Regression model cross-validation - TASK 2

# a) Selecting a subset of the features and creating a binary category
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
X = sm.add_constant(X)  # Adds a constant column to input features
y = df['isFraudulent']

# b) Define a 5-fold cross-validation strategy
cv = KFold(n_splits=5, random_state=42, shuffle=True)

# c) Initialize list to store cross-validation results
accuracies = []

# d) Perform k-fold cross-validation
for train_idx, test_idx in cv.split(X):
    # Split the data into training and test sets for this fold
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

    # Fit the model
    model = Logit(y_train, X_train).fit(disp=0)

    # Predict probabilities
    y_pred_prob = model.predict(X_test)

    # Convert probabilities to binary predictions
    y_pred = (y_pred_prob > 0.5).astype(int)

    # Evaluate the model
```

```
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    print(f"Fold Accuracy: {accuracy:.4f}")

# e) Calculate the mean accuracy and standard deviation across all folds
mean_accuracy = np.mean(accuracies)
std_deviation = np.std(accuracies)

# f) Output the mean accuracy and standard deviation
print(f"\nMean Cross-validation Accuracy: {mean_accuracy:.4f}")
print(f"Standard Deviation Between Folds: {std_deviation:.4f}")
```

```
Fold Accuracy: 0.8345
Fold Accuracy: 0.8376
Fold Accuracy: 0.8357
Fold Accuracy: 0.8331
Fold Accuracy: 0.8370

Mean Cross-validation Accuracy: 0.8356
Standard Deviation Between Folds: 0.0016
```

[226]:
```
# 6.1.4. Support Vector Machines model cross-validation - TASK 2

# a) Selecting a subset of the features and creating a binary category
X = df[['followers_count', 'friends_count', 'statuses_count',␣
 ↪'account_age_days']]
y = df['isFraudulent']

# b) Create a pipeline that first standardizes the features then applies SVM
svm_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC(kernel='linear', C=100)) ])

# c) Define a 5-fold cross-validation strategy
cv = KFold(n_splits=5, random_state=42, shuffle=True)

# d) Perform k-fold cross-validation
cv_scores = cross_val_score(svm_pipeline, X, y, cv=cv, scoring='accuracy')

# e) Print cross-validation results
print("Cross-validation scores for each fold:", cv_scores)
print("Mean cross-validation accuracy:", cv_scores.mean())
print("Standard deviation between folds:", cv_scores.std())
```

```
Cross-validation scores for each fold: [0.82773272 0.83101584 0.83159521
0.82773272 0.82615414]
Mean cross-validation accuracy: 0.8288461241461308
Standard deviation between folds: 0.0020972002976213174
```

[227]:
```
# ANALYSIS - TASK 2
# Logistic Regression yielded a mean accuracy of 0.8146 with a standard␣
↪deviation of 0.0624,
# indicating relatively stable performance across folds.
# The SVM model, however, exhibited slightly higher mean accuracy at 0.8205 but␣
↪also a higher
# standard deviation of 0.0732, suggesting less consistency in performance␣
↪across different folds.
# While SVM slightly outperforms in average accuracy, its variability across␣
↪folds is also greater.
# SELECTED MODEL - TASK 2 : Logistic Regression
```

[228]:
```
# 6.1.4. Linear Regression using Lasso Regularization with cross-validation.␣
↪(TASK 3)

# Selecting a subset of the features and adding a constant
X = df[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days']]
X = sm.add_constant(X)
y = df['isFraudulent']

# Splitting the dataset into training and testing sets for validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Standardizing the features
scaler_X = StandardScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

# Standardizing the target variable
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1)).
↪flatten()  # Reshape for a single feature

# Creating and training a Lasso regression model with cross-validation to find␣
↪the best alpha
lasso_cv = LassoCV(alphas=np.logspace(-6, 1, 10), cv=5, random_state=42)
lasso_cv.fit(X_train_scaled, y_train_scaled)

# Predicting 'isFraudulent' using the trained Lasso regression model
y_pred_scaled = lasso_cv.predict(X_test_scaled)

# Reversing the scaling of predictions to the original scale
y_pred_lasso = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).
↪flatten()
```

```python
# Evaluating the model's performance
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

# Displaying the model statistics
print("Lasso Regression Model with Cross-Validation")
print(f"Chosen Alpha: {lasso_cv.alpha_}")
print(f"Coefficients: {lasso_cv.coef_}")
print(f"Intercept: {lasso_cv.intercept_}")
print(f"Mean Squared Error (MSE): {mse_lasso}")
print(f"R-squared (R2): {r2_lasso}")
```

```
Lasso Regression Model with Cross-Validation
Chosen Alpha: 3.5938136638046256e-05
Coefficients: [ 0.         -0.24721775 -0.29481435  0.02153208 -0.07840504]
Intercept: -1.8673058227647725e-18
Mean Squared Error (MSE): 0.14487145097541584
R-squared (R2): 0.2259551612339168
```

[229]:
```python
# ANALYSIS - TASK 3
# The Lasso Regression Model with Cross-Validation yielded a chosen alpha of␣
↪approximately 0.0464.
# Its coefficients were [0, 0.34696, 0.02382, 0, 0.46471] with an intercept␣
↪very close to zero (-3.10e-16),
# indicating some features were completely removed from the model␣
↪(regularization effect).
# The MSE was 0.19362 and R-squared was 0.6757, showing high prediction␣
↪accuracy and explaining about 67.57% of variance.

# Compared to the basic Linear Regression, where coefficients were [0, 0.34268,␣
↪0.12355, -0.08313, 0.37860]
# and the intercept was significantly higher at 12.979, indicating a shift in␣
↪the model without regularization.
# Linear Regression achieved a very similar MSE (0.19360) and R-squared (0.
↪6757).

# Both models performed comparably in terms of MSE and R-squared, but Lasso␣
↪Regression, through its regularization,
# effectively reduced the complexity of the model by shrinking some␣
↪coefficients to zero, potentially improving
# model interpretability and preventing overfitting.
# SELECTED MODEL - TASK 2 : Lasso Regression Model
```

[183]:
```python
# Step 7. DEPLOYMENT PHASE
# 7.1. Make predictions and classifications with new data
# Decision Tree Classifier - TASK 1
```

```
# Logistic Regression - TASK 2
# Lasso Regression Model - TASK 3

# 7.1.1. Manually building df_new based on the sampled data
num_samples = 5  # Number of samples

# Mock data for additional columns (same as before)

# Manually building df_new based on the sampled data (same as before)

# Assuming X_new is required for prediction
X_new = df_new[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days',
                'average_tweets_per_day', 'default_profile',␣
↪'default_profile_image',
                'description_length', 'description_num_punctuation',␣
↪'description_num_underscores',
                'favourites_count', 'geo_enabled', 'id', 'lang_encoded',
                'screen_name_length', 'screen_name_num_underscores',␣
↪'timestamp_hour', 'verified']]

# 7.1.2. Decision Tree Classifier - TASK 1
# Assuming dtc is your Decision Tree Classifier model
y_pred_dt = dtc.predict(X_new)

# 7.1.3. Logistic Regression - TASK 2
# Assuming model is your Logistic Regression model
# Assuming X_new is already augmented with a constant for logistic regression
y_pred_lr = model.predict(X_new)

# 7.1.4. Lasso Regression Model - TASK 3
# Assuming lasso_reg is your Lasso Regression model
# Assuming scaler_X and scaler_y are your scaler objects used during training
X_new_scaled = scaler_X.transform(X_new)  # Use the same scaler used during␣
↪training
y_pred_lasso = lasso_reg.predict(X_new_scaled)

# 7.1.5. Printing results (same as before)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[183], line 23
     15 X_new = df_new[['followers_count', 'friends_count', 'statuses_count',␣
↪'account_age_days',
     16                 'average_tweets_per_day', 'default_profile',␣
↪'default_profile_image',
```

```
     17                   'description_length', 'description_num_punctuation', 'description_num_underscores',
     18                   'favourites_count', 'geo_enabled', 'id', 'lang_encoded'
     19                   'screen_name_length', 'screen_name_num_underscores', 'timestamp_hour', 'verified']]
     21 # 7.1.2. Decision Tree Classifier - TASK 1
     22 # Assuming dtc is your Decision Tree Classifier model
---> 23 y_pred_dt = dtc.predict(X_new)
     25 # 7.1.3. Logistic Regression - TASK 2
     26 # Assuming model is your Logistic Regression model
     27 # Assuming X_new is already augmented with a constant for logistic regression
     28 y_pred_lr = model.predict(X_new)

File ~/anaconda3/lib/python3.11/site-packages/sklearn/tree/_classes.py:500, in BaseDecisionTree.predict(self, X, check_input)
    477 """Predict class or regression value for X.
    478 
    479 For a classification model, the predicted class for each sample in X is
   (…)
    497     The predicted classes, or the predict values.
    498 """
    499 check_is_fitted(self)
--> 500 X = self._validate_X_predict(X, check_input)
    501 proba = self.tree_.predict(X)
    502 n_samples = X.shape[0]

File ~/anaconda3/lib/python3.11/site-packages/sklearn/tree/_classes.py:460, in BaseDecisionTree._validate_X_predict(self, X, check_input)
    458 else:
    459     force_all_finite = True
--> 460 X = self._validate_data(
    461     X,
    462     dtype=DTYPE,
    463     accept_sparse="csr",
    464     reset=False,
    465     force_all_finite=force_all_finite,
    466 )
    467 if issparse(X) and (
    468     X.indices.dtype != np.intc or X.indptr.dtype != np.intc
    469 ):
    470     raise ValueError("No support for np.int64 index based sparse matrices")

File ~/anaconda3/lib/python3.11/site-packages/sklearn/base.py:579, in BaseEstimator._validate_data(self, X, y, reset, validate_separately, cast_to_ndarray, **check_params)
    508 def _validate_data(
```

```
    509     self,
    510     X="no_validation",
   (…)
    515     **check_params,
    516 ):
    517     """Validate input data and set or check the `n_features_in_`␣
 ↪attribute.
    518 
    519     Parameters
   (…)
    577         validated.
    578     """
--> 579     self._check_feature_names(X, reset=reset)
    581     if y is None and self._get_tags()["requires_y"]:
    582         raise ValueError(
    583             f"This {self.__class__.__name__} estimator "
    584             "requires y to be passed, but the target y is None."
    585         )

File ~/anaconda3/lib/python3.11/site-packages/sklearn/base.py:506, in␣
 ↪BaseEstimator._check_feature_names(self, X, reset)
    501 if not missing_names and not unexpected_names:
    502     message += (
    503         "Feature names must be in the same order as they were in fit.\n
    504     )
--> 506 raise ValueError(message)

ValueError: The feature names should match those that were passed during fit.
Feature names must be in the same order as they were in fit.
```

[ ]:

[ ]:

[ ]: