Credit Card Fraud Detection with Machine Learning Comprehensive Model

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
```

In [2]:
```
data = pd.read_csv('card_transdata.csv')

data.head()
```

Out[2]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_
0	57.877857	0.311140	1.945940	
1	10.829943	0.175592	1.294219	
2	5.091079	0.805153	0.427715	
3	2.247564	5.600044	0.362663	
4	44.190936	0.566486	2.222767	

Clean Dataset

Drop missing values

In [3]:
```
missing_values = data.isnull().any(axis=1)
print('Rows with Missing Values: ')
print(missing_values)
```

```
Rows with Missing Values:
0         False
1         False
2         False
3         False
4         False
          ...
999995    False
999996    False
999997    False
999998    False
999999    False
Length: 1000000, dtype: bool
```

In [ ]:

Check for duplicated Rows

In [4]:
```
duplicate_rows = data[data.duplicated()]
print("Duplicated Rows:")
print(duplicate_rows)

data.dropna(axis=0, inplace=True)

data.drop_duplicates(inplace=True)
```

```
Duplicated Rows:
Empty DataFrame
Columns: [distance_from_home, distance_from_last_transaction, ratio_t
o_median_purchase_price, repeat_retailer, used_chip, used_pin_number,
online_order, fraud]
Index: []
```

In [ ]:

Analyze Fraud for used_chip, used_pin_number

Create New Data Frame for used_chip

In [5]:
```
chippin_df = data[["used_chip", 'used_pin_number', 'fraud']]

chippin_df.head()
```

Out[5]:

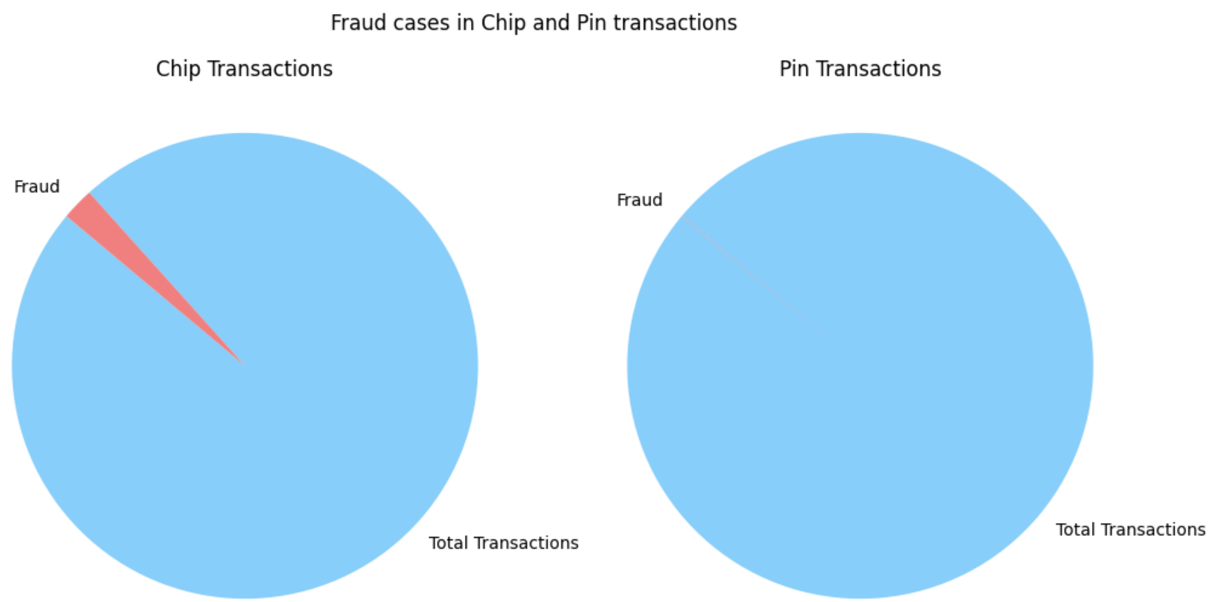
	used_chip	used_pin_number	fraud
0	1.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	1.0	0.0	0.0
4	1.0	0.0	0.0

In [6]:
```
total_transactions = len(chippin_df)
total_fraud = chippin_df['fraud'].sum()

fraud_by_chip = chippin_df[chippin_df['used_chip']==1.0]['fraud'].sum(

fraud_by_pin = chippin_df[chippin_df['used_pin_number']==1.0]['fraud']
```

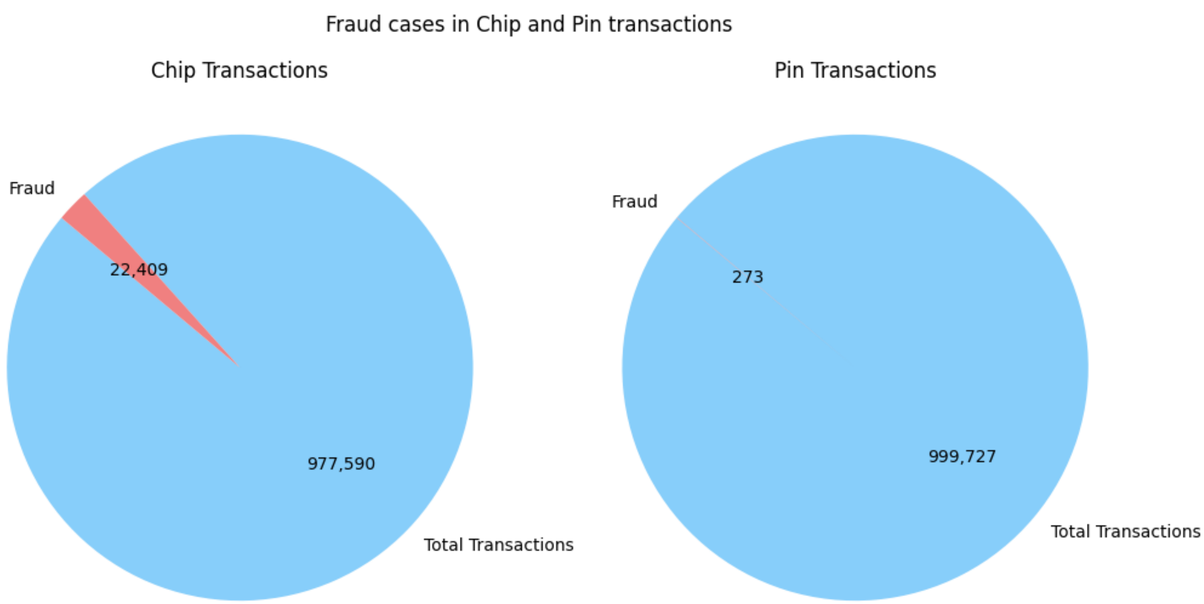
In [7]:
```
print("Total transactions:", total_transactions)
print("Total fraud cases:", total_fraud)

print("Fraud cases using chip: {} out of {}".format(fraud_by_chip, tot
print("Fraud cases using pin: {} out of {}".format(fraud_by_pin, total
```

```
Total transactions: 1000000
Total fraud cases: 87403.0
Fraud cases using chip: 22410.0 out of 1000000
Fraud cases using pin: 273.0 out of 1000000
```

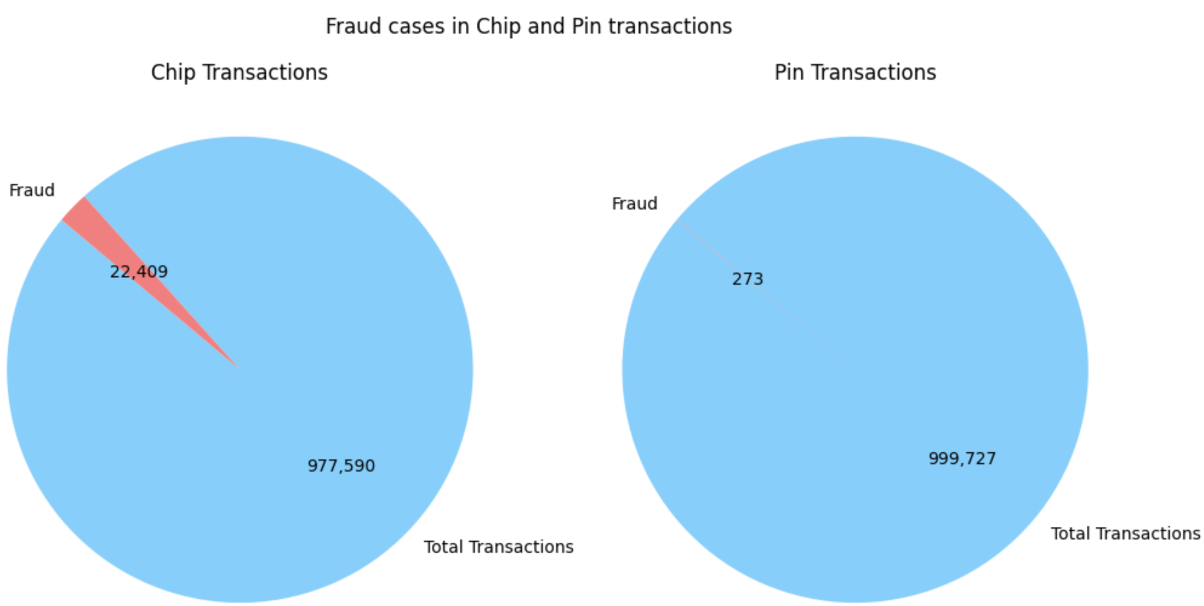
In [59]:
```
### Visualize the Fraud by chip and fraud by pin Data using Matplot Li
import plotly.express as px
#Pieplot #1
labels_chip = ["Total Transactions", "Fraud"]
sizes_chip = [total_transactions - fraud_by_chip, fraud_by_chip]
colors_chip = ["lightskyblue", "lightcoral"]

labels_pin = ["Total Transactions", "Fraud"]
sizes_pin = [total_transactions - fraud_by_pin, fraud_by_pin]
colors_pin = ["lightskyblue", "lightcoral"]

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.pie(sizes_chip, labels=labels_chip, colors=colors_chip, startangle
plt.axis("equal")
plt.title("Chip Transactions")

#Pieplot#2
plt.subplot(1, 2, 2)
plt.pie(sizes_pin, labels=labels_pin, colors=colors_pin, startangle=14
plt.axis("equal")
plt.title("Pin Transactions")
plt.suptitle("Fraud cases in Chip and Pin transactions")

plt.show()
```



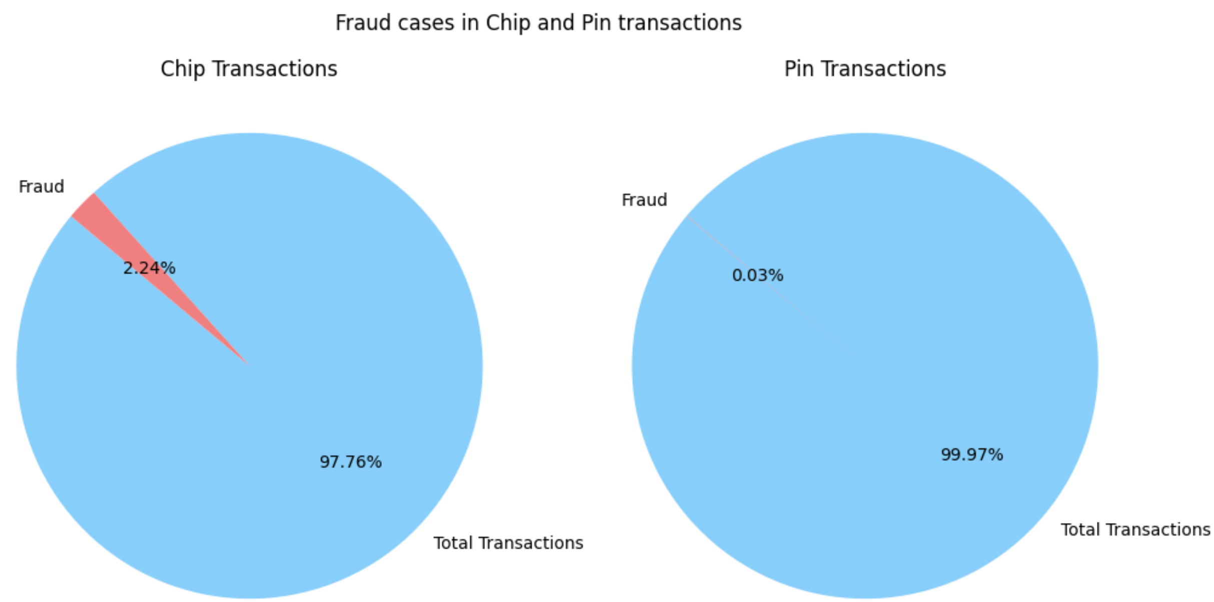
In [60]:
```
total_transactions = 1000000
fraud_by_chip = 22410
fraud_by_pin = 273

import matplotlib.pyplot as plt

# Pie plot #1
labels_chip = ["Total Transactions", "Fraud"]
sizes_chip = [total_transactions - fraud_by_chip, fraud_by_chip]
colors_chip = ["lightskyblue", "lightcoral"]

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.pie(sizes_chip, labels=labels_chip, colors=colors_chip, startangle
plt.axis("equal")
plt.title("Chip Transactions")

#Pieplot#2
plt.subplot(1, 2, 2)
plt.pie(sizes_pin, labels=labels_pin, colors=colors_pin, startangle=14
plt.axis("equal")
plt.title("Pin Transactions")
plt.suptitle("Fraud cases in Chip and Pin transactions")

plt.show()


plt.show()
```



In [61]:
```
### Visualize the Fraud by chip and fraud by pin Data using Matplot Li
import plotly.express as px
#Pieplot #1
labels_chip = ["Total Transactions", "Fraud"]
sizes_chip = [total_transactions - fraud_by_chip, fraud_by_chip]
colors_chip = ["lightskyblue", "lightcoral"]

labels_pin = ["Total Transactions", "Fraud"]
sizes_pin = [total_transactions - fraud_by_pin, fraud_by_pin]
colors_pin = ["lightskyblue", "lightcoral"]

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.pie(sizes_chip, labels=labels_chip, colors=colors_chip, startangle
plt.axis("equal")
plt.title("Chip Transactions")

##Pieplot#2
plt.subplot(1, 2, 2)
plt.pie(sizes_pin, labels=labels_pin, colors=colors_pin, startangle=14
plt.axis("equal")
plt.title("Pin Transactions")
plt.suptitle("Fraud cases in Chip and Pin transactions")

plt.show()
```



In [64]:
```
import matplotlib.pyplot as plt

#Pieplot #1
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.pie(sizes_chip, labels=labels_chip, colors=colors_chip, startangle
plt.axis("equal")
plt.title("Chip Transactions")

clever_autopct = lambda pct: f"{pct:.2f}%" if pct > 0 else ""
##Pieplot#2
plt.subplot(1, 2, 2)
plt.pie(sizes_pin, labels=labels_pin, colors=colors_pin, startangle=14
plt.axis("equal")
plt.title("Pin Transactions")
plt.suptitle("Fraud cases in Chip and Pin transactions")

plt.show()
```



In [ ]:

In [11]:
```
total_transactions-fraud_by_pin
```

Out[11]: 999727

In [ ]:

Analyzing Repeat Fraud Patterns

Create Repeat Retailer Dataframe

In [12]:
```
repeat_retailer_df = data[['repeat_retailer', 'fraud']]

repeat_retailer_df
```

Out[12]:

	repeat_retailer	fraud
0	1.0	0.0
1	1.0	0.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0
...	...	...
999995	1.0	0.0
999996	1.0	0.0
999997	1.0	0.0
999998	1.0	0.0
999999	1.0	0.0

1000000 rows × 2 columns

In [13]:
```
fraud_sequences = []
current_sequence = []

for index, row in repeat_retailer_df.iterrows():
    repeat_retailer, is_fraud = row['repeat_retailer'], row['fraud']

    if is_fraud ==1:
        if current_sequence:
            fraud_sequences.append(current_sequence.copy())
        current_sequence=[]
    else:
        current_sequence.append("Repeat Retailer" if repeat_retailer =
for i, sequence in enumerate(fraud_sequences[:10], start=1):
    print(f"Fraud Sequence {i}: {', '.join(sequence)}")
```

```
Fraud Sequence 1: Repeat Retailer, Repeat Retailer, Repeat Retailer
```

```
Fraud Sequence 1: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retailer, R
epeat Retailer, No Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer
Fraud Sequence 2: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retailer, R
epeat Retailer, No Repeat Retailer, Repeat Retailer
Fraud Sequence 3: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer
Fraud Sequence 4: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer
Fraud Sequence 5: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, No Repeat Retailer, Repeat Retailer, Repeat Retaile
r, Repeat Retailer, Repeat Retailer, Repeat Retailer
Fraud Sequence 6: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer
Fraud Sequence 7: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer, No Repeat Retailer, Repeat Retaile
r, No Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retai
ler, Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retail
er, Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retaile
r, Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retaile
r, Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retaile
r, Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retaile
r, Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retaile
r, No Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retai
ler, Repeat Retailer, Repeat Retailer, Repeat Retailer, No Repeat Ret
ailer, Repeat Retailer, No Repeat Retailer, Repeat Retailer
Fraud Sequence 8: No Repeat Retailer, Repeat Retailer
Fraud Sequence 9: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer, No Repeat Retailer, Repeat Retailer
Fraud Sequence 10: Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retailer, R
epeat Retailer, No Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer, Repeat Retailer, Repeat Retailer, R
epeat Retailer, No Repeat Retailer, Repeat Retailer, Repeat Retailer,
Repeat Retailer, Repeat Retailer
```

In [ ]:

Finding Correlation Between Transaction Amount and Fraud

Create DataFrame for Coorelation

In [14]:
```
correlation_df = data[['ratio_to_median_purchase_price', 'fraud']]

correlation_df.head()
```

Out[14]:

	ratio_to_median_purchase_price	fraud
0	1.945940	0.0
1	1.294219	0.0
2	0.427715	0.0
3	0.362663	0.0
4	2.222767	0.0

In [15]:
```
correlation = correlation_df["ratio_to_median_purchase_price"].corr(cc

print(f"Coorelation between transaction amount and fraud:  {correlatic
```

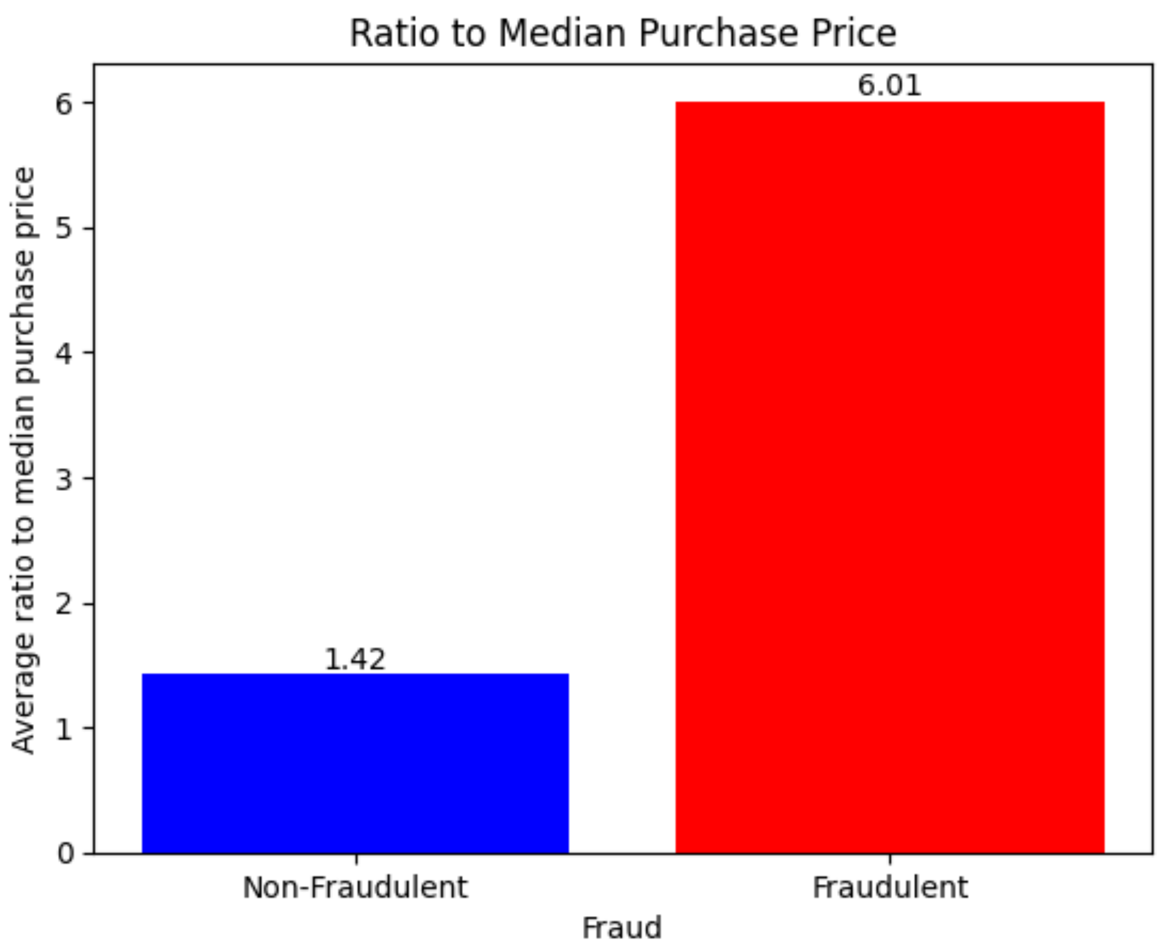
```
Coorelation between transaction amount and fraud:  0.4623047222882586
4
```

In [16]:
```
avgnonfraudtransaction = correlation_df[correlation_df["fraud"]==0]['r
avgfraudtransaction = correlation_df[correlation_df["fraud"]==1]['rati
print(f"Average ratio to median purchase price for non fraudulent tran
print(f"Average ratio to median purchase price for fraudulent transact
```

```
Average ratio to median purchase price for non fraudulent transactio
n:  1.423641855458059
Average ratio to median purchase price for fraudulent transaction:
6.006323490486969
```

Visualize the Data using MatPlotlib for Ratio to Median Purchase Price

In [17]:
```
import plotly.express as px

import matplotlib.pyplot as plt
import pandas as pd

data = {'categories': ['Non-Fraudulent', 'Fraudulent'],
        'average_ratio': [1.423, 6.006]}
df = pd.DataFrame(data)

fig, ax = plt.subplots()

ax.bar(df['categories'], df['average_ratio'], color=['blue', 'red'])

for i, v in enumerate(df['average_ratio'].round(2)):
    ax.text(i, v, str(v), ha='center', va='bottom')

ax.set_title('Ratio to Median Purchase Price')
ax.set_xlabel('Fraud')
ax.set_ylabel('Average ratio to median purchase price')

plt.show()
```



Analyzing Fraud Casses Online Transactions

Create a Data Frame from online order and fraud

In [18]:
```
data = pd.read_csv('card_transdata.csv')

data.head()

online_order_df = data[['online_order','fraud']]

online_order_df
```

Out[18]:

	online_order	fraud
0	0.0	0.0
1	0.0	0.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0
...	...	...
999995	0.0	0.0
999996	0.0	0.0
999997	1.0	0.0
999998	1.0	0.0
999999	1.0	0.0

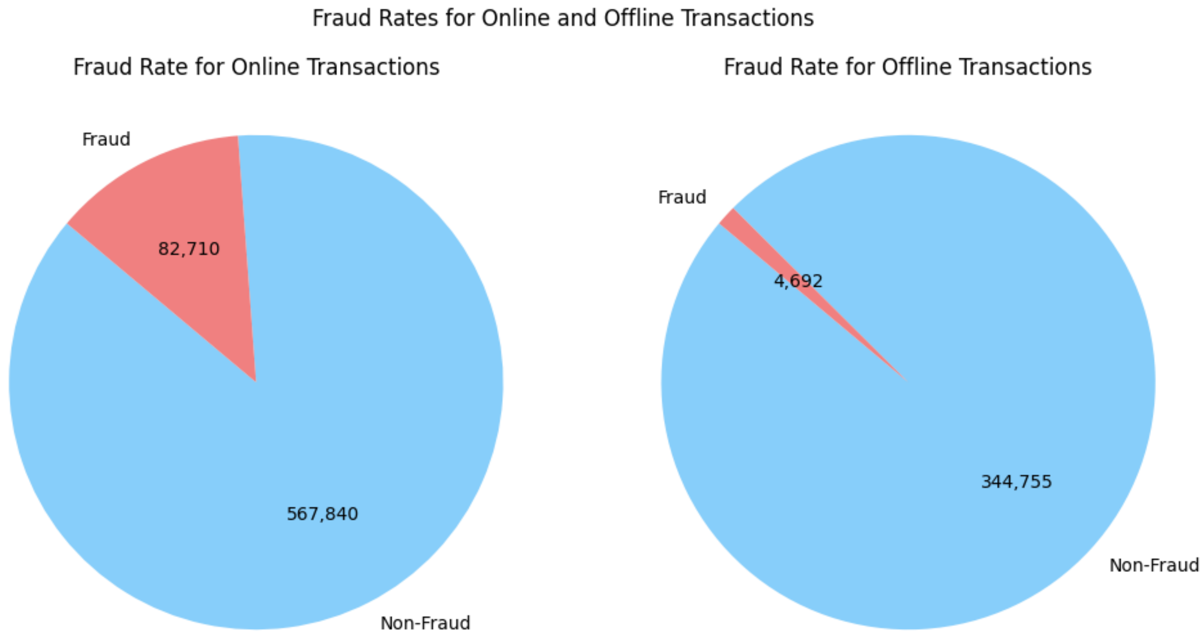
1000000 rows × 2 columns

In [ ]:

In [19]:
```
# Online Orders
total_online_orders = online_order_df["online_order"].sum()
total_online_fraud = online_order_df[(online_order_df['fraud']==1)&(or
fraud_rate_online = total_online_fraud/total_online_orders

# Offline Orders
total_offline_orders = len(online_order_df) - total_online_orders
total_offline_fraud = online_order_df[(online_order_df['fraud']==1)&(c
fraud_rate_offline = total_offline_fraud/total_offline_orders

print(f"Fraud rate for online transactions: {fraud_rate_online: .2%} (

print(f"Fraud rate for offline transactions: {fraud_rate_offline: .2%}
```

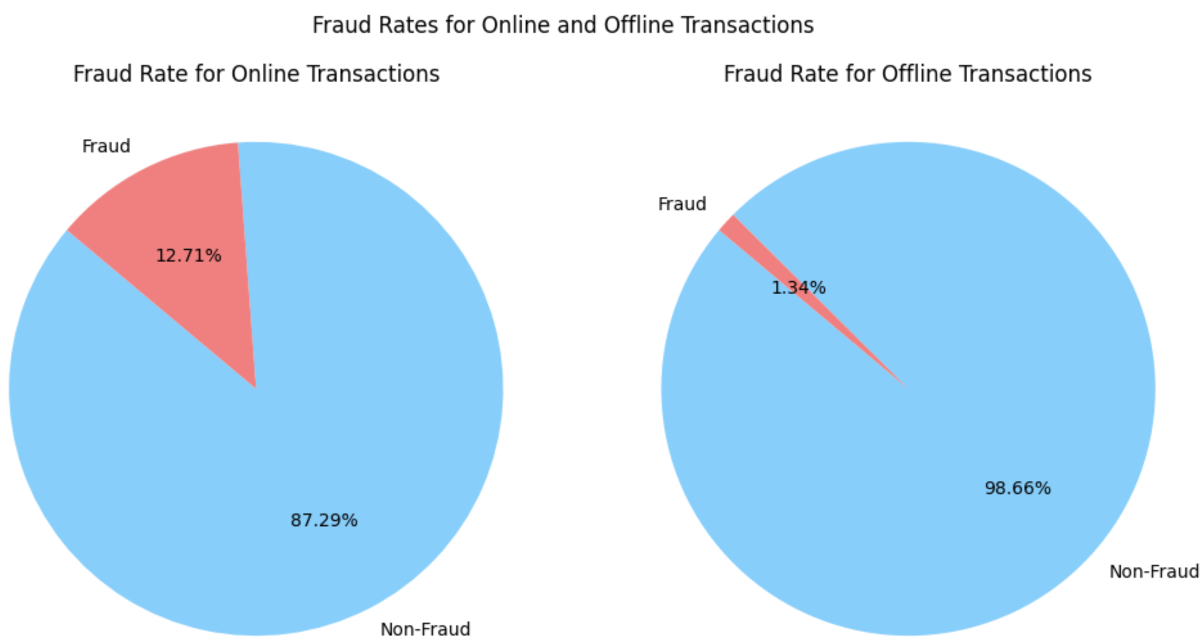
```
Fraud rate for online transactions:  12.71% (82711 cases out of 65055
2.0 online transactions)
Fraud rate for offline transactions:  1.34% (4692 cases out of 34944
8.0 offline transactions)
```

In [20]:
```
fraud_online = 82711
total_online = 650552.0
fraud_offline = 4692
total_offline = 349448.0

import matplotlib.pyplot as plt

labels_online = ["Non-Fraud", "Fraud"]
sizes_online = [total_online - fraud_online, fraud_online]
colors_online = ["lightskyblue", "lightcoral"]

labels_offline = ["Non-Fraud", "Fraud"]
sizes_offline = [total_offline - fraud_offline, fraud_offline]
colors_offline = ["lightskyblue", "lightcoral"]

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.pie(sizes_online, labels=labels_online, colors=colors_online, star
plt.axis("equal")
plt.title("Fraud Rate for Online Transactions")

plt.subplot(1, 2, 2)
plt.pie(sizes_offline, labels=labels_offline, colors=colors_offline, s
plt.axis("equal")
plt.title("Fraud Rate for Offline Transactions")

plt.suptitle("Fraud Rates for Online and Offline Transactions")

plt.show()
```


Fraud Rates for Online and Offline Transactions
Fraud Rate for Online Transactions
Fraud
82,710
567,840
Non-Fraud
Fraud Rate for Offline Transactions
Fraud
4,692
344,755
Non-Fraud

In [65]:

```
fraud_online = 82711
total_online = 650552.0
fraud_offline = 4692
total_offline = 349448.0

import matplotlib.pyplot as plt

labels_online = ["Non-Fraud", "Fraud"]
sizes_online = [total_online - fraud_online, fraud_online]
colors_online = ["lightskyblue", "lightcoral"]

labels_offline = ["Non-Fraud", "Fraud"]
sizes_offline = [total_offline - fraud_offline, fraud_offline]
colors_offline = ["lightskyblue", "lightcoral"]

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.pie(sizes_online, labels=labels_online, colors=colors_online, star
plt.axis("equal")
plt.title("Fraud Rate for Online Transactions")

plt.subplot(1, 2, 2)
plt.pie(sizes_offline, labels=labels_offline, colors=colors_offline, s
plt.axis("equal")
plt.title("Fraud Rate for Offline Transactions")

plt.suptitle("Fraud Rates for Online and Offline Transactions")

plt.show()
```



In [ ]:

In [ ]:

In [ ]:

Conducting Feature Selection with Random Forest

In [21]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

In [22]:
```
data = pd.read_csv('card_transdata.csv')

data.head()
```

Out[22]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_
0	57.877857	0.311140	1.945940	
1	10.829943	0.175592	1.294219	
2	5.091079	0.805153	0.427715	
3	2.247564	5.600044	0.362663	
4	44.190936	0.566486	2.222767	

Create our X and y variables

In [23]:
```
X = data.drop("fraud", axis=1) # the x variable is all of the columns
y = data['fraud']  # target column
```

Train, Test, Split

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

Scale the Data wuith Standard Scaler

In [ ]:

In [ ]:

Random Forest Classifier

In [25]:
```
rf_classifier = RandomForestClassifier(random_state=42)
```

Fit Random Forest Classifier

In [26]:
```
rf_classifier.fit(X_train, y_train)
```

Out[26]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Feature Importance

In [27]:
```
feature_importances = pd.Series(rf_classifier.feature_importances_,inc
print("Ranked Feature Importance: ")
print(feature_importances)
```

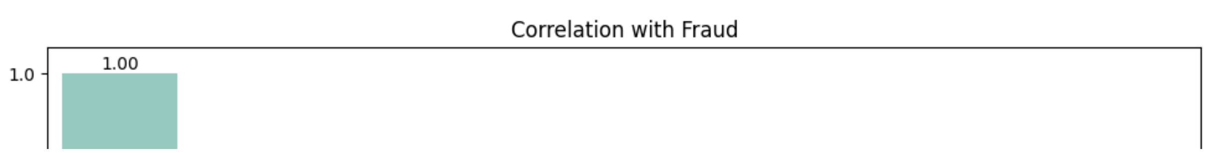
```
Ranked Feature Importance:
ratio_to_median_purchase_price    0.527171
online_order                      0.169382
distance_from_home                0.134910
used_pin_number                   0.063928
used_chip                         0.052078
distance_from_last_transaction    0.045711
repeat_retailer                   0.006820
dtype: float64
```

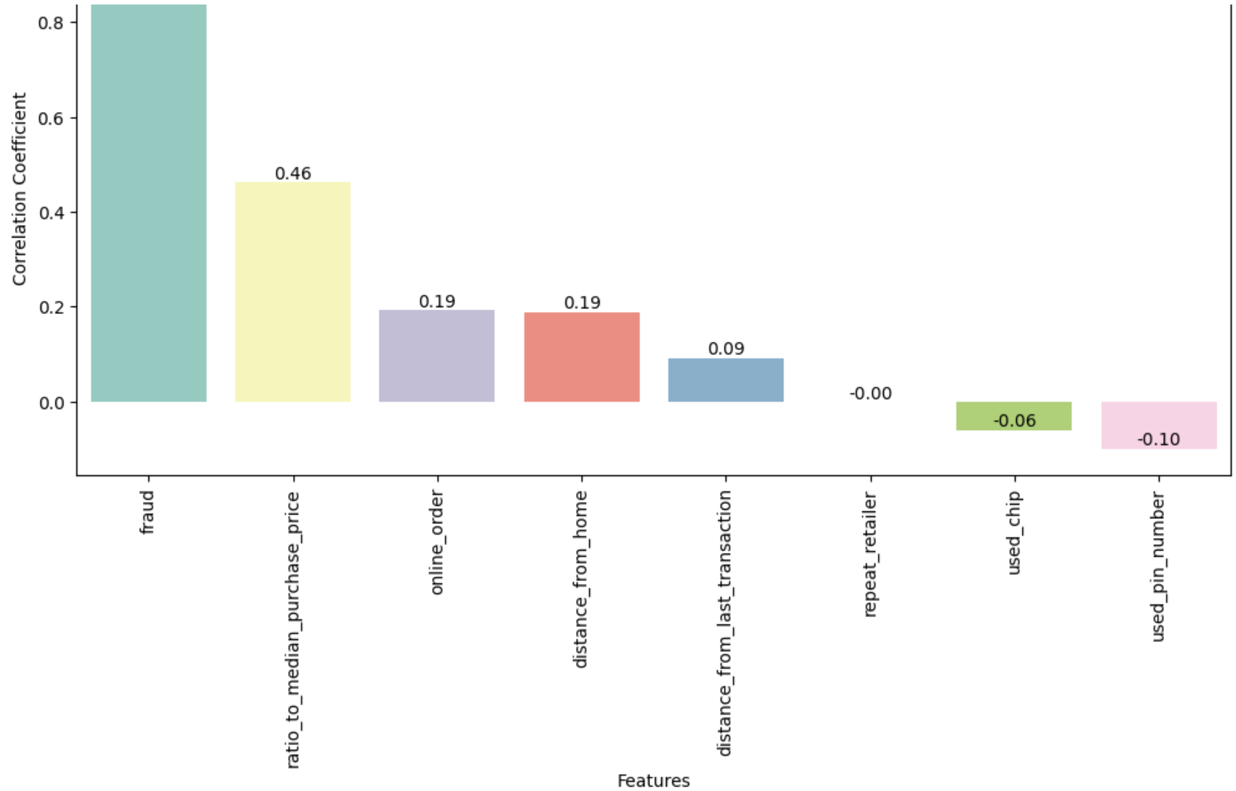
In [28]:
```
numeric_columns = data.select_dtypes(include=['float', 'int']).columns
corr = data[numeric_columns].corr()['fraud'].sort_values()
corr
```

Out[28]:
```
used_pin_number                  -0.100293
used_chip                        -0.060975
repeat_retailer                  -0.001357
distance_from_last_transaction    0.091917
distance_from_home                0.187571
online_order                      0.191973
ratio_to_median_purchase_price    0.462305
fraud                             1.000000
Name: fraud, dtype: float64
```

In [29]:
```
import seaborn as sns

numeric_columns = data.select_dtypes(include=['float', 'int']).columns
corr = data[numeric_columns].corr()['fraud'].sort_values(ascending = F

plt.figure(figsize=(12, 6))
ax = sns.barplot(x=corr.index, y=corr.values, palette='Set3')  # Set c
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
ax.set_title('Correlation with Fraud')
ax.set_xlabel('Features')
ax.set_ylabel('Correlation Coefficient')

for i, v in enumerate(corr.values):
    ax.text(i, v, f'{v:.2f}', ha='center', va='bottom')

plt.show()
```

```
/var/folders/_1/gsvjgb894xx9rw70yr1spt000000gn/T/ipykernel_50577/3728
568272.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be r
emoved in v0.14.0. Assign the `x` variable to `hue` and set `legend=F
alse` for the same effect.

  ax = sns.barplot(x=corr.index, y=corr.values, palette='Set3')  # Se
t custom palette
/var/folders/_1/gsvjgb894xx9rw70yr1spt000000gn/T/ipykernel_50577/3728
568272.py:8: UserWarning: set_ticklabels() should only be used with a
fixed number of ticks, i.e. after set_ticks() or using a FixedLocato
r.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```





In [ ]:

Building Credit Card Fraud Detection Model with Random Forest

Create Random Sample

In [30]:
```
new_transaction_features = data.sample(1).drop('fraud',axis = 1)
print("\nRandomly sampled features for new transaction:")
print(new_transaction_features)

prediction = rf_classifier.predict(new_transaction_features)
print("\nPrediction for new transaction")
print("Fraud" if prediction[0] == 1 else "Legitimate")
```

```
Randomly sampled features for new transaction:
        distance_from_home  distance_from_last_transaction  \
877170           26.274697                        0.958592

        ratio_to_median_purchase_price  repeat_retailer  used_chip  \
877170                        1.660521              1.0        0.0

        used_pin_number  online_order
877170              0.0           1.0

Prediction for new transaction
Legitimate
```

Create our X and y variables

In [31]:
```
X = data.drop("fraud", axis=1) # the x variable is all of the columns
y = data['fraud']  # target column
```

Fit Random Forest Classifier

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.


rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

new_transaction_features = data.sample(1).drop('fraud',axis = 1)
print("\nRandomly sampled features for new transaction:")
print(new_transaction_features)

prediction = rf_classifier.predict(new_transaction_features)
print("\nPrediction for new transaction")
print("Fraud" if prediction[0] == 1 else "Legitimate")
```

```
Randomly sampled features for new transaction:
        distance_from_home  distance_from_last_transaction  \
978622            1.402091                        4.780455

        ratio_to_median_purchase_price  repeat_retailer  used_chip  \
978622                        0.703732              0.0        0.0

        used_pin_number  online_order
978622              0.0           1.0

Prediction for new transaction
Legitimate
```

In [33]:
```
new_transaction_features1 = pd.DataFrame({
    'distance_from_home':[85],
    'distance_from_last_transaction':[75],
    'ratio_to_median_purchase_price':[5.1],
    'repeat_retailer':[0],
    'used_chip':[1],
    'used_pin_number':[0],
    'online_order':[0]
})

prediction = rf_classifier.predict(new_transaction_features1)
print("\nPrediction for new transaction")
print("Fraud" if prediction[0] == 1 else "Legitimate")
```

```
Prediction for new transaction
Fraud
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Building A Credt Card Detection Model with Logistic Regression

In [34]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import accuracy_score
```

In [35]:
```
data = pd.read_csv('card_transdata.csv')

data.head()
```

Out[35]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_
0	57.877857	0.311140	1.945940	
1	10.829943	0.175592	1.294219	
2	5.091079	0.805153	0.427715	
3	2.247564	5.600044	0.362663	
4	44.190936	0.566486	2.222767	

Create X and y values

In [36]:
```
X = data.drop("fraud", axis=1)
y = data['fraud']
```

Test, Train, Split

In [37]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

Scale the Data

In [38]:
```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

logreg_classifier = LogisticRegression(max_iter= 1000, random_state=42
logreg_classifier.fit(X_train_scaled, y_train)

new_transaction_features1 = pd.DataFrame({
    'distance_from_home':[89],
    'distance_from_last_transaction':[15],
    'ratio_to_median_purchase_price':[2.3],
    'repeat_retailer':[1],
    'used_chip':[0],
    'used_pin_number':[1],
    'online_order':[0]
})

prediction = logreg_classifier.predict(scaler.transform(new_transactic

print("\nPrediction for New Transaction:")
print("Fraud" if prediction[0]==1 else "Legitimate")
```

```
Prediction for New Transaction:
Legitimate
```

Accuracy Metrics for Logistic Regression

In [39]:
```
y_pred = logreg_classifier.predict(X_test_scaled)
precision = precision_score(y_test,y_pred)
recall = recall_score(y_test,y_pred)
f1 = f1_score(y_test,y_pred)
accuracy = accuracy_score(y_test,y_pred)


print("\nEvaluation Metrics:")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"f1 score: {f1:.4f}")
print(f"accuracy: {accuracy:.4f}")
```

```
Evaluation Metrics:
Precision: 0.8900
Recall: 0.5975
f1 score: 0.7150
accuracy: 0.9585
```

In [ ]:
```

```

Building Credit CardFraudDetection Model with SVM

In [40]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
```

In [41]:
```python
data = pd.read_csv('card_transdata.csv').sample(1000, random_state=42)

data.head()
```

Out[41]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	re
987231	0.929509	1.296477	0.361110	
79954	0.611179	0.208295	3.118884	
567130	3.956062	0.529194	1.579942	
500891	21.798902	0.019399	11.416909	
55399	3.310635	1.707802	2.028915	

In [42]:
```python
X = data.drop("fraud", axis=1) # the x variable is all of the columns
y = data['fraud']  # target column
```

Scale Data with Standard Scaler

In [43]:
```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
svm_classifier = SVC(kernel = "linear", probability=True, random_state
calibrated_svm = CalibratedClassifierCV(svm_classifier)
calibrated_svm.fit(X_scaled,y)
distance_from_home = float(input("Enter Distance From Home: "))
distance_from_last_transaction = float(input("Enter Distance From Last
ratio_to_median_purchase_price = float(input("Enter Ratio to Median Pu
repeat_retailer = int(input("Enter Repeat Retailer (0 or 1): "))
used_chip = int(input("Enter Used Chip (0 or 1): "))
used_pin_number = int(input("Enter Used Pin Number (0 or 1): "))
online_order = int(input("Enter Online Order (0 or 1): "))
```

```
Enter Distance From Home: 78
Enter Distance From Last Transaction: 62
Enter Ratio to Median Purchase Price: 7.2
Enter Repeat Retailer (0 or 1): 1
Enter Used Chip (0 or 1): 1
Enter Used Pin Number (0 or 1): 0
Enter Online Order (0 or 1): 1
```

Create DataFrame from User Input

In [48]:
```
new_transaction_features1 = pd.DataFrame({
    'distance_from_home':[distance_from_home],
    'distance_from_last_transaction':[distance_from_last_transaction],
    'ratio_to_median_purchase_price':[ratio_to_median_purchase_price],
    'repeat_retailer':[repeat_retailer],
    'used_chip':[used_chip],
    'used_pin_number':[used_pin_number],
    'online_order':[online_order]
})
scaled_transaction = scaler.transform(new_transaction_features1)
```

Make Predictions

In [49]:
```
prediction = calibrated_svm.predict(scaled_transaction)
probability_of_fraud = calibrated_svm.predict_proba(scaled_transactior
print("\nPrediction for New Transaction:")
print("Fraud" if prediction[0] == 1 else "Legitimate")
print(f"Probability of Fraud: {probability_of_fraud * 100:.2f}%")
```

```
Prediction for New Transaction:
Fraud
Probability of Fraud: 70.82%
```

Acuracy Metric for SVC

In [50]:
```
y_pred = calibrated_svm.predict(X_test_scaled)
precision = precision_score(y_test,y_pred)
recall = recall_score(y_test,y_pred)
f1 = f1_score(y_test,y_pred)
accuracy = accuracy_score(y_test,y_pred)


print("\nEvaluation Metrics:")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"f1 score: {f1:.4f}")
print(f"accuracy: {accuracy:.4f}")
```

```
Evaluation Metrics:
Precision: 0.8998
Recall: 0.2918
f1 score: 0.4406
accuracy: 0.9354
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```
## Random Forest Accuracy
```

In [51]:

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

y_pred = rf_classifier.predict(X_test_scaled)
precision = precision_score(y_test,y_pred)
recall = recall_score(y_test,y_pred)
f1 = f1_score(y_test,y_pred)
accuracy = accuracy_score(y_test,y_pred)


print("\nEvaluation Metrics:")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"f1 score: {f1:.4f}")
print(f"accuracy: {accuracy:.4f}")
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/base.py:439: User
Warning: X does not have valid feature names, but RandomForestClassif
ier was fitted with feature names
  warnings.warn(


Evaluation Metrics:
Precision: 0.7889
Recall: 0.0654
f1 score: 0.1207
accuracy: 0.9170
```