In [ ]:
```
# Introduction: Detecting Fraudulent Insurance Claims (with focus on Using Mlflow f

# Insurance fraud is a multi-billion dollar problem that affects both insurance com
# This project aims to develop a machine learning model to accurately identify pote
# By leveraging a dataset of claim information, we can predict which claims are mos

# In this notebook, we will:
# 1.  Explore and clean the data to understand its characteristics.
# 2.  Engineer relevant features to improve model performance.
# 3.  Train and evaluate several classification models.
# 4.  Identify the model with the best predictive capability by comparing them usin
```

In [427…
```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
import mlflow
import mlflow.sklearn
```

In [428…
```
df = pd.read_csv("/mnt/c/users/micha/downloads/insurance_claims/insurance_claims/in
```

In [ ]:
```
# Initial Data Exploration
# The dataset contains 40 columns, including both categorical and numerical feature
# Before proceeding, it's crucial to understand the data types, identify missing va
# This will ensure the data is in a suitable format for machine learning algorithms
# The first few rows give us a glimpse into the structure of the data.
```

In [429…
```
df.head()
```

Out[429…

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	poli
0	328	48	521585	2014-10-17	OH	250/500	
1	228	42	342868	2006-06-27	IN	250/500	
2	134	29	687698	2000-09-06	OH	100/300	
3	256	41	227811	1990-05-25	IL	250/500	
4	228	44	367455	2014-06-06	IL	500/1000	

5 rows × 40 columns

In [430…
```
df.isna().sum()
```

Out[430…

```
months_as_customer              0
age                             0
policy_number                   0
policy_bind_date                0
policy_state                    0
policy_csl                      0
policy_deductable               0
policy_annual_premium           0
umbrella_limit                  0
insured_zip                     0
insured_sex                     0
insured_education_level         0
insured_occupation              0
insured_hobbies                 0
insured_relationship            0
capital-gains                   0
capital-loss                    0
incident_date                   0
incident_type                   0
collision_type                  0
incident_severity               0
authorities_contacted          91
incident_state                  0
incident_city                   0
incident_location               0
incident_hour_of_the_day        0
number_of_vehicles_involved     0
property_damage                 0
bodily_injuries                 0
witnesses                       0
police_report_available         0
total_claim_amount              0
injury_claim                    0
property_claim                  0
vehicle_claim                   0
auto_make                       0
auto_model                      0
auto_year                       0
fraud_reported                  0
_c39                         1000
dtype: int64
```

In [ ]:
```
#Removed some of the columns that were not useful for our purpose
```

In [431…
```
df = df.drop(["_c39","policy_number","policy_bind_date","insured_zip","incident_loc
```

In [ ]:
```
#let's check out the number of missing values in our dataset
```

In [432…
```
missing_values = ["?", "N/A", "-", "NULL"]
(df.isin(missing_values)).sum()
```

Out[432…

```
months_as_customer              0
age                             0
policy_state                    0
policy_csl                      0
policy_deductable               0
policy_annual_premium           0
umbrella_limit                  0
insured_sex                     0
insured_education_level         0
insured_occupation              0
insured_hobbies                 0
insured_relationship            0
capital-gains                   0
capital-loss                    0
incident_date                   0
incident_type                   0
collision_type                178
incident_severity               0
authorities_contacted           0
incident_state                  0
incident_city                   0
incident_hour_of_the_day        0
number_of_vehicles_involved     0
property_damage               360
bodily_injuries                 0
witnesses                       0
police_report_available       343
total_claim_amount              0
injury_claim                    0
property_claim                  0
vehicle_claim                   0
auto_make                       0
auto_model                      0
auto_year                       0
fraud_reported                  0
dtype: int64
```

In [1]:

```
# Am I simply going to remove all those valuable rows only because they have missin
# is not dealing with the missing values. it is Classification with a focus on usin
```

In [433…

```
df = df[~df.isin(missing_values).any(axis=1)]
```

In [434…

```
(df.isin(missing_values)).sum()
```

Out[434…

```
months_as_customer            0
age                           0
policy_state                  0
policy_csl                    0
policy_deductable             0
policy_annual_premium         0
umbrella_limit                0
insured_sex                   0
insured_education_level       0
insured_occupation            0
insured_hobbies               0
insured_relationship          0
capital-gains                 0
capital-loss                  0
incident_date                 0
incident_type                 0
collision_type                0
incident_severity             0
authorities_contacted         0
incident_state                0
incident_city                 0
incident_hour_of_the_day      0
number_of_vehicles_involved   0
property_damage               0
bodily_injuries               0
witnesses                     0
police_report_available       0
total_claim_amount            0
injury_claim                  0
property_claim                0
vehicle_claim                 0
auto_make                     0
auto_model                    0
auto_year                     0
fraud_reported                0
dtype: int64
```

In [435…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 340 entries, 0 to 992
Data columns (total 35 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           340 non-null    int64  
 1   age                          340 non-null    int64  
 2   policy_state                 340 non-null    object 
 3   policy_csl                   340 non-null    object 
 4   policy_deductable            340 non-null    int64  
 5   policy_annual_premium        340 non-null    float64
 6   umbrella_limit               340 non-null    int64  
 7   insured_sex                  340 non-null    object 
 8   insured_education_level      340 non-null    object 
 9   insured_occupation           340 non-null    object 
 10  insured_hobbies              340 non-null    object 
 11  insured_relationship         340 non-null    object 
 12  capital-gains                340 non-null    int64  
 13  capital-loss                 340 non-null    int64  
 14  incident_date                340 non-null    object 
 15  incident_type                340 non-null    object 
 16  collision_type               340 non-null    object 
 17  incident_severity            340 non-null    object 
 18  authorities_contacted        340 non-null    object 
 19  incident_state               340 non-null    object 
 20  incident_city                340 non-null    object 
 21  incident_hour_of_the_day     340 non-null    int64  
 22  number_of_vehicles_involved  340 non-null    int64  
 23  property_damage              340 non-null    object 
 24  bodily_injuries              340 non-null    int64  
 25  witnesses                    340 non-null    int64  
 26  police_report_available      340 non-null    object 
 27  total_claim_amount           340 non-null    int64  
 28  injury_claim                 340 non-null    int64  
 29  property_claim               340 non-null    int64  
 30  vehicle_claim                340 non-null    int64  
 31  auto_make                    340 non-null    object 
 32  auto_model                   340 non-null    object 
 33  auto_year                    340 non-null    int64  
 34  fraud_reported               340 non-null    object 
dtypes: float64(1), int64(15), object(19)
memory usage: 95.6+ KB
```

In [ ]:
```
#now we want to investigate our categorical columns
#and see how many unique values each of these columns have
```

In [436…
```
cat_cols = df.select_dtypes(include=['object', 'category']).columns

for col in cat_cols:
    print(f"{col}: {df[col].nunique()} unique values")
```

```
policy_state: 3 unique values
policy_csl: 3 unique values
insured_sex: 2 unique values
insured_education_level: 7 unique values
insured_occupation: 14 unique values
insured_hobbies: 20 unique values
insured_relationship: 6 unique values
incident_date: 60 unique values
incident_type: 2 unique values
collision_type: 3 unique values
incident_severity: 3 unique values
authorities_contacted: 4 unique values
incident_state: 7 unique values
incident_city: 7 unique values
property_damage: 2 unique values
police_report_available: 2 unique values
auto_make: 14 unique values
auto_model: 39 unique values
fraud_reported: 2 unique values
```

In [ ]:
```
#we decided to remove the auto model because of the high number of unique values (H
```

In [437…
```
df = df.drop(["auto_model"],axis=1)
```

In [438…
```
df
```

Out[438…

	months_as_customer	age	policy_state	policy_csl	policy_deductable	policy_annual_pr
0	328	48	OH	250/500	1000	
2	134	29	OH	100/300	2000	
5	256	39	OH	250/500	1000	
8	27	33	IL	100/300	500	
11	447	61	OH	100/300	2000	
...	...	...	...	...	...	
980	245	40	IL	500/1000	1000	
984	163	36	IN	250/500	1000	
988	295	46	IN	100/300	500	
991	257	44	OH	100/300	1000	
992	94	26	IN	100/300	500	

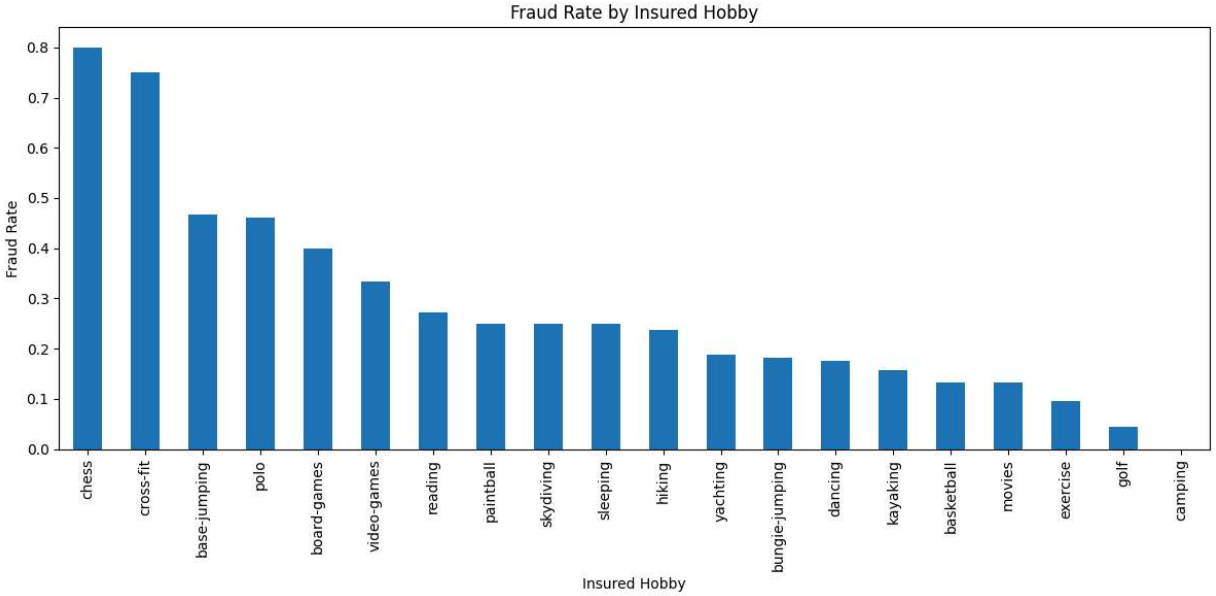
340 rows × 34 columns

In [ ]:
```
# let's explore the correlation between hobbies and fraud.
```

In [439…

```
import matplotlib.pyplot as plt
df['fraud_binary'] = df['fraud_reported'].map({'Y': 1, 'N': 0})
fraud_rate_by_hobby = df.groupby('insured_hobbies')['fraud_binary'].mean().sort_val
print(fraud_rate_by_hobby)

fraud_rate_by_hobby.plot(kind='bar', figsize=(12, 6), title='Fraud Rate by Insured
plt.ylabel('Fraud Rate')
plt.xlabel('Insured Hobby')
plt.tight_layout()
plt.show()
```

```
insured_hobbies
chess              0.800000
cross-fit          0.750000
base-jumping       0.466667
polo               0.461538
board-games        0.400000
video-games        0.333333
reading            0.272727
paintball          0.250000
skydiving          0.250000
sleeping           0.250000
hiking             0.238095
yachting           0.187500
bungie-jumping     0.181818
dancing            0.176471
kayaking           0.157895
basketball         0.133333
movies             0.133333
exercise           0.095238
golf               0.043478
camping            0.000000
Name: fraud_binary, dtype: float64
```



In [440…

```
#We got a useful categorical feature here. insured_hobbies does correlate with fra
```

In [ ]:
```
#let's divide the hobby list into two groups of risky vs not risky hobbies. This si
#from chess to videogames -> risky
#from reading to camping -> not risky
risky = ['chess', 'cross-fit', 'base-jumping', 'polo','board-games', 'video-games']
```

In [ ]:
```
#we added a new column for that
```

In [441…
```
df['hobby_risk'] = df['insured_hobbies'].apply(lambda x: 'risky' if x in risky else
```

In [442…
```
df.head()
```

Out[442…

	months_as_customer	age	policy_state	policy_csl	policy_deductable	policy_annual_pre
0	328	48	OH	250/500	1000	14
2	134	29	OH	100/300	2000	14
5	256	39	OH	250/500	1000	13
8	27	33	IL	100/300	500	14
11	447	61	OH	100/300	2000	11

5 rows × 36 columns

In [ ]:
```
#now that we have the new column with only two categories (risky-not risky), let's
```

In [443…
```
df = df.drop("insured_hobbies",axis=1)
```

In [444…
```
df.head()
```

Out[444…

	months_as_customer	age	policy_state	policy_csl	policy_deductable	policy_annual_pre
0	328	48	OH	250/500	1000	14
2	134	29	OH	100/300	2000	14
5	256	39	OH	250/500	1000	13
8	27	33	IL	100/300	500	14
11	447	61	OH	100/300	2000	11

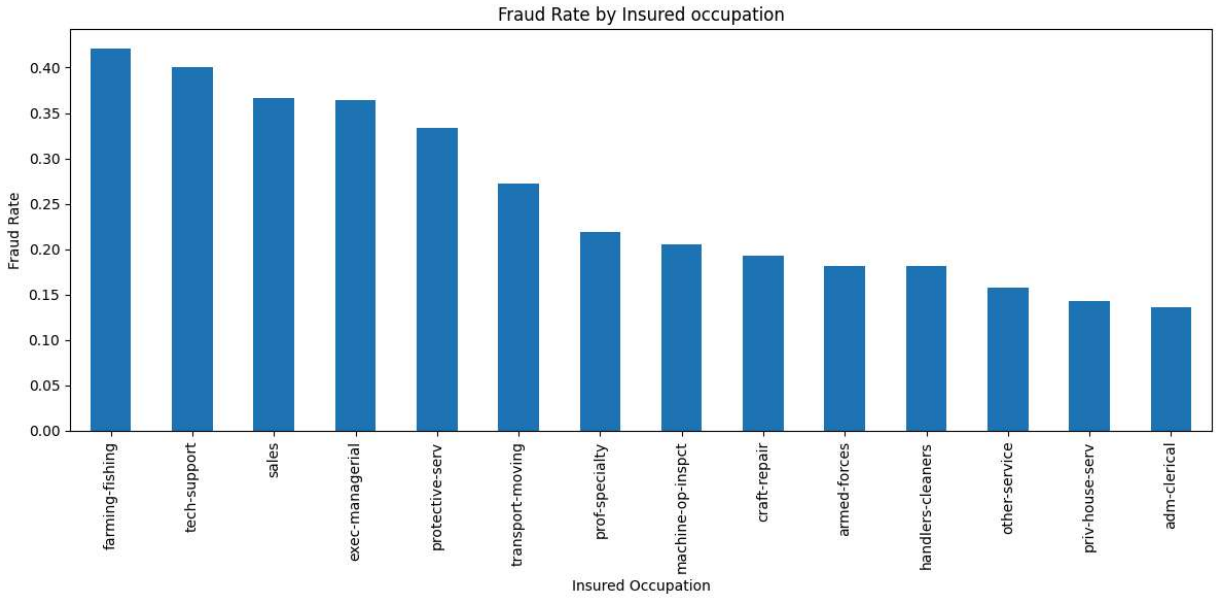
5 rows × 35 columns

In [ ]:
```
#let's do the same process. but this time for the insured occupation
```

In [445…
```
fraud_rate_by_occupation = df.groupby('insured_occupation')['fraud_binary'].mean().
print(fraud_rate_by_occupation)

fraud_rate_by_occupation.plot(kind='bar', figsize=(12, 6), title='Fraud Rate by Ins
```

```
plt.ylabel('Fraud Rate')
plt.xlabel('Insured Occupation')
plt.tight_layout()
plt.show()
```

```
insured_occupation
farming-fishing      0.421053
tech-support         0.400000
sales                0.366667
exec-managerial      0.363636
protective-serv      0.333333
transport-moving     0.272727
prof-specialty       0.218750
machine-op-inspct    0.205882
craft-repair         0.192308
armed-forces         0.181818
handlers-cleaners    0.181818
other-service        0.157895
priv-house-serv      0.142857
adm-clerical         0.136364
Name: fraud_binary, dtype: float64
```



In [ ]:
```
#this time instead of listing the high risk jobs (just like what we did for hobbies
#fraud rate will be considered risky.
#PS: this section needs further investigation (the rates look similar, I might cons
```

In [446…
```
risky_occupations=[]
for insured_occupation,fraud_rate in fraud_rate_by_occupation.items():
    if fraud_rate>=0.3:
        risky_occupations.append(insured_occupation)
print(risky_occupations)
```

```
['farming-fishing', 'tech-support', 'sales', 'exec-managerial', 'protective-serv']
```

In [447…
```
df['occupation_risk'] = df['insured_occupation'].apply(lambda x: 'risky' if x in ri
```

In [448…
```
df.head()
```

Out[448…

	months_as_customer	age	policy_state	policy_csl	policy_deductable	policy_annual_pre
0	328	48	OH	250/500	1000	14
2	134	29	OH	100/300	2000	14
5	256	39	OH	250/500	1000	13
8	27	33	IL	100/300	500	14
11	447	61	OH	100/300	2000	11

5 rows × 36 columns

In [449…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 340 entries, 0 to 992
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           340 non-null    int64  
 1   age                          340 non-null    int64  
 2   policy_state                 340 non-null    object 
 3   policy_csl                   340 non-null    object 
 4   policy_deductable            340 non-null    int64  
 5   policy_annual_premium        340 non-null    float64
 6   umbrella_limit               340 non-null    int64  
 7   insured_sex                  340 non-null    object 
 8   insured_education_level      340 non-null    object 
 9   insured_occupation           340 non-null    object 
 10  insured_relationship         340 non-null    object 
 11  capital-gains                340 non-null    int64  
 12  capital-loss                 340 non-null    int64  
 13  incident_date                340 non-null    object 
 14  incident_type                340 non-null    object 
 15  collision_type               340 non-null    object 
 16  incident_severity            340 non-null    object 
 17  authorities_contacted        340 non-null    object 
 18  incident_state               340 non-null    object 
 19  incident_city                340 non-null    object 
 20  incident_hour_of_the_day     340 non-null    int64  
 21  number_of_vehicles_involved  340 non-null    int64  
 22  property_damage              340 non-null    object 
 23  bodily_injuries              340 non-null    int64  
 24  witnesses                    340 non-null    int64  
 25  police_report_available      340 non-null    object 
 26  total_claim_amount           340 non-null    int64  
 27  injury_claim                 340 non-null    int64  
 28  property_claim               340 non-null    int64  
 29  vehicle_claim                340 non-null    int64  
 30  auto_make                    340 non-null    object 
 31  auto_year                    340 non-null    int64  
 32  fraud_reported               340 non-null    object 
 33  fraud_binary                 340 non-null    int64  
 34  hobby_risk                   340 non-null    object 
 35  occupation_risk              340 non-null    object 
dtypes: float64(1), int64(16), object(19)
memory usage: 98.3+ KB
```

In [ ]:
```
#dropping some columns that are not going to be useful anymore
```

In [450…
```
df = df.drop(["incident_date","insured_occupation"],axis=1)
```

In [451…
```
df = df.drop("fraud_reported",axis=1)
```

In [ ]:
```
#grouping the columns into two lists of numerical and cactegorical for future easy
```

In [452…
```
num_cols = df.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = df.select_dtypes(include=["object"]).columns.tolist()
```

In [453…
```
print(cat_cols)
```

```
['policy_state', 'policy_csl', 'insured_sex', 'insured_education_level', 'insured_relationship', 'incident_type', 'collision_type', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_city', 'property_damage', 'police_report_available', 'auto_make', 'hobby_risk', 'occupation_risk']
```

In [ ]:
```
#investigating the unique values in some of the categorical columns
```

In [454…
```
df["incident_severity"].unique()
```

Out[454…
```
array(['Major Damage', 'Minor Damage', 'Total Loss'], dtype=object)
```

In [455…
```
df["insured_education_level"].unique()
```

Out[455…
```
array(['MD', 'PhD', 'High School', 'College', 'Masters', 'JD',
       'Associate'], dtype=object)
```

In [456…
```
df["property_damage"].unique()
```

Out[456…
```
array(['YES', 'NO'], dtype=object)
```

In [457…
```
df[cat_cols]
```

Out[457…

	policy_state	policy_csl	insured_sex	insured_education_level	insured_relationship	inci
0	OH	250/500	MALE	MD	husband	Sing
2	OH	100/300	FEMALE	PhD	own-child	Mu
5	OH	250/500	FEMALE	PhD	unmarried	Mu
8	IL	100/300	FEMALE	PhD	own-child	Sing
11	OH	100/300	FEMALE	High School	other-relative	Mu
…	…	…	…	…	…	
980	IL	500/1000	MALE	PhD	unmarried	Sing
984	IN	250/500	MALE	MD	husband	Sing
988	IN	100/300	FEMALE	High School	wife	Sing
991	OH	100/300	MALE	MD	other-relative	Sing
992	IN	100/300	MALE	MD	husband	Mu

340 rows × 16 columns

In [ ]:
```
#grouping the categorical columns into two groups of nominal and ordinal for future
```

In [458…
```
nominal_columns = [item for item in cat_cols if item not in ['insured_education_lev
ordinal_columns = ['insured_education_level', 'incident_severity']
```

In [ ]:
```
#The following code takes the categorical (text-based) columns in my data and conve
#It does this using a technique called "one-hot encoding." The result is a new tabl
```

In [459…
```
onehot = OneHotEncoder(drop='first', sparse_output=False)
nominal_data = df[nominal_columns]
encoded_nominal = onehot.fit_transform(nominal_data)
encoded_nominal_df = pd.DataFrame(
    encoded_nominal,
    columns=onehot.get_feature_names_out(nominal_columns),
    index=df.index
)
encoded_nominal_df
```

Out[459…

	policy_state_IN	policy_state_OH	policy_csl_250/500	policy_csl_500/1000	insured_sex_M
0	0.0	1.0	1.0	0.0	
2	0.0	1.0	0.0	0.0	
5	0.0	1.0	1.0	0.0	
8	0.0	0.0	0.0	0.0	
11	0.0	1.0	0.0	0.0	
...	...	...	...	...	
980	0.0	0.0	0.0	1.0	
984	1.0	0.0	1.0	0.0	
988	1.0	0.0	0.0	0.0	
991	0.0	1.0	0.0	0.0	
992	1.0	0.0	0.0	0.0	

340 rows × 45 columns

In [ ]:
```
# Let's deal with the ordinal columns. The text-based columns whose values have an
#We need to turn them into numbers so that the machine learning algorithms deal bet
```

In [460…
```
education_order = {
    'High School': 0,
    'Associate': 1,
    'College': 2,
    'Masters': 3,
    'JD': 4,
    'MD': 5,
    'PhD': 6
}

# We Defined mapping for incident severity
severity_order = {
    'Minor Damage': 0,
    'Major Damage': 1,
    'Total Loss': 2
    }


df['insured_education_level_encoded'] = df['insured_education_level'].map(education
df['incident_severity_encoded'] = df['incident_severity'].map(severity_order)
```

In [461…
```
df.head()
```

Out[461…

	months_as_customer	age	policy_state	policy_csl	policy_deductable	policy_annual_pre
0	328	48	OH	250/500	1000	14
2	134	29	OH	100/300	2000	14
5	256	39	OH	250/500	1000	13
8	27	33	IL	100/300	500	14
11	447	61	OH	100/300	2000	11

5 rows × 35 columns

In [ ]:
```
#from here all i do is to get rid of the original unuseful columns and glue all the
```

In [462…
```
df_combined = df.drop(columns=nominal_columns)  # drop original columns
```

In [463…
```
df_combined
```

Out[463…

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit
0	328	48	1000	1406.91	0
2	134	29	2000	1413.14	5000000
5	256	39	1000	1351.10	0
8	27	33	500	1442.99	0
11	447	61	2000	1137.16	0
…	…	…	…	…	…
980	245	40	1000	1361.45	0
984	163	36	1000	1503.21	0
988	295	46	500	1672.88	0
991	257	44	1000	1280.88	0
992	94	26	500	722.66	0

340 rows × 21 columns

In [464…
```
df_combined = df_combined.join(encoded_nominal_df)    # join encoded columns
```

In [465…
```
df_combined = df_combined.drop(["insured_education_level", "incident_severity"],axi
```

In [466…
```
df_combined
```

Out[466…

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit
0	328	48	1000	1406.91	0
2	134	29	2000	1413.14	5000000
5	256	39	1000	1351.10	0
8	27	33	500	1442.99	0
11	447	61	2000	1137.16	0
...	...	...	...	...	...
980	245	40	1000	1361.45	0
984	163	36	1000	1503.21	0
988	295	46	500	1672.88	0
991	257	44	1000	1280.88	0
992	94	26	500	722.66	0

340 rows × 64 columns

In [467…

```
df_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 340 entries, 0 to 992
Data columns (total 64 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   months_as_customer                    340 non-null    int64
 1   age                                   340 non-null    int64
 2   policy_deductable                     340 non-null    int64
 3   policy_annual_premium                 340 non-null    float64
 4   umbrella_limit                        340 non-null    int64
 5   capital-gains                         340 non-null    int64
 6   capital-loss                          340 non-null    int64
 7   incident_hour_of_the_day              340 non-null    int64
 8   number_of_vehicles_involved           340 non-null    int64
 9   bodily_injuries                       340 non-null    int64
 10  witnesses                             340 non-null    int64
 11  total_claim_amount                    340 non-null    int64
 12  injury_claim                          340 non-null    int64
 13  property_claim                        340 non-null    int64
 14  vehicle_claim                         340 non-null    int64
 15  auto_year                             340 non-null    int64
 16  fraud_binary                          340 non-null    int64
 17  insured_education_level_encoded       340 non-null    int64
 18  incident_severity_encoded             340 non-null    int64
 19  policy_state_IN                       340 non-null    float64
 20  policy_state_OH                       340 non-null    float64
 21  policy_csl_250/500                    340 non-null    float64
 22  policy_csl_500/1000                   340 non-null    float64
 23  insured_sex_MALE                      340 non-null    float64
 24  insured_relationship_not-in-family    340 non-null    float64
 25  insured_relationship_other-relative   340 non-null    float64
 26  insured_relationship_own-child        340 non-null    float64
 27  insured_relationship_unmarried        340 non-null    float64
 28  insured_relationship_wife             340 non-null    float64
 29  incident_type_Single Vehicle Collision  340 non-null  float64
 30  collision_type_Rear Collision         340 non-null    float64
 31  collision_type_Side Collision         340 non-null    float64
 32  authorities_contacted_Fire            340 non-null    float64
 33  authorities_contacted_Other           340 non-null    float64
 34  authorities_contacted_Police          340 non-null    float64
 35  incident_state_NY                     340 non-null    float64
 36  incident_state_OH                     340 non-null    float64
 37  incident_state_PA                     340 non-null    float64
 38  incident_state_SC                     340 non-null    float64
 39  incident_state_VA                     340 non-null    float64
 40  incident_state_WV                     340 non-null    float64
 41  incident_city_Columbus                340 non-null    float64
 42  incident_city_Hillsdale               340 non-null    float64
 43  incident_city_Northbend               340 non-null    float64
 44  incident_city_Northbrook              340 non-null    float64
 45  incident_city_Riverwood               340 non-null    float64
 46  incident_city_Springfield             340 non-null    float64
 47  property_damage_YES                   340 non-null    float64
 48  police_report_available_YES           340 non-null    float64
 49  auto_make_Audi                        340 non-null    float64
 50  auto_make_BMW                         340 non-null    float64
```

```
 51  auto_make_Chevrolet                  340 non-null    float64
 52  auto_make_Dodge                      340 non-null    float64
 53  auto_make_Ford                       340 non-null    float64
 54  auto_make_Honda                      340 non-null    float64
 55  auto_make_Jeep                       340 non-null    float64
 56  auto_make_Mercedes                   340 non-null    float64
 57  auto_make_Nissan                     340 non-null    float64
 58  auto_make_Saab                       340 non-null    float64
 59  auto_make_Suburu                     340 non-null    float64
 60  auto_make_Toyota                     340 non-null    float64
 61  auto_make_Volkswagen                 340 non-null    float64
 62  hobby_risk_risky                     340 non-null    float64
 63  occupation_risk_risky                340 non-null    float64
dtypes: float64(46), int64(18)
memory usage: 172.7 KB
```

In [468…
```
num_cols
```

Out[468…
```
['months_as_customer',
 'age',
 'policy_deductable',
 'policy_annual_premium',
 'umbrella_limit',
 'capital-gains',
 'capital-loss',
 'incident_hour_of_the_day',
 'number_of_vehicles_involved',
 'bodily_injuries',
 'witnesses',
 'total_claim_amount',
 'injury_claim',
 'property_claim',
 'vehicle_claim',
 'auto_year',
 'fraud_binary']
```

In [469…
```
df_combined
```

Out[469…

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit
0	328	48	1000	1406.91	0
2	134	29	2000	1413.14	5000000
5	256	39	1000	1351.10	0
8	27	33	500	1442.99	0
11	447	61	2000	1137.16	0
...	...	...	...	...	...
980	245	40	1000	1361.45	0
984	163	36	1000	1503.21	0
988	295	46	500	1672.88	0
991	257	44	1000	1280.88	0
992	94	26	500	722.66	0

340 rows × 64 columns

In [ ]:
```
#let's start using mlflow and do some machine learning
```

In [472…
```
from sklearn.model_selection import train_test_split
```

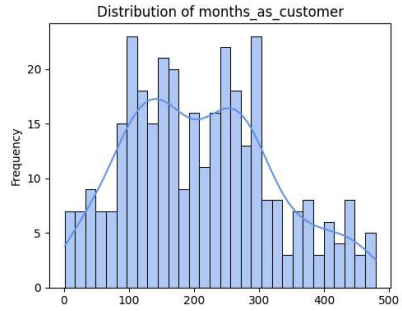
In [473…
```
from sklearn.preprocessing import StandardScaler
```

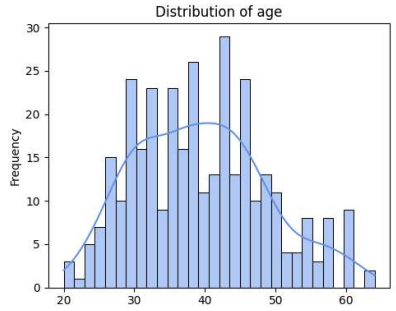
In [474…
```
y = df_combined['fraud_binary']
X = df_combined.drop(columns=['fraud_binary'])
```

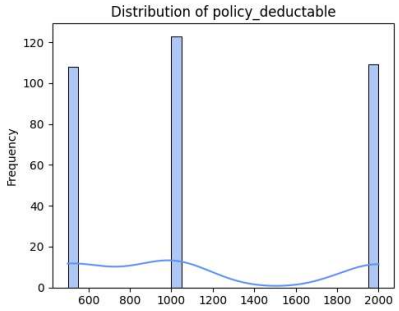
In [ ]:
```
# An investigation into which numerical numbers need scaling
```

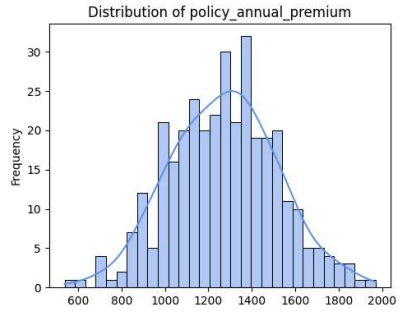
In [475…
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set up the grid
n_cols = 3
n_rows = (len(num_cols) + n_cols - 1) // n_cols
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows))

# Flatten the axes array
axes = axes.flatten()

# Loop through num_cols and plot
for i, col in enumerate(num_cols):
    sns.histplot(df_combined[col], ax=axes[i], kde=True, bins=30, color='cornflower
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel('')
```

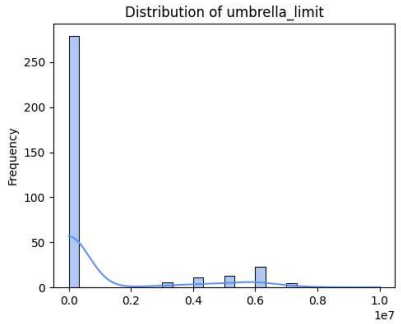
```
    axes[i].set_ylabel('Frequency')

# Turn off unused axes
for j in range(i + 1, len(axes)):
    axes[j].set_visible(False)

plt.tight_layout()
plt.show()
```

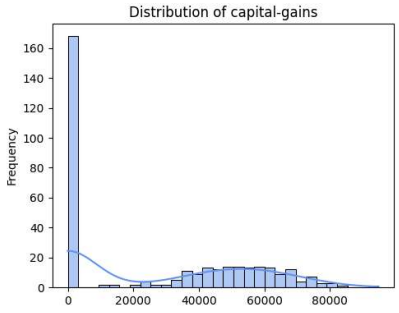

Distribution of months_as_customer
Frequency

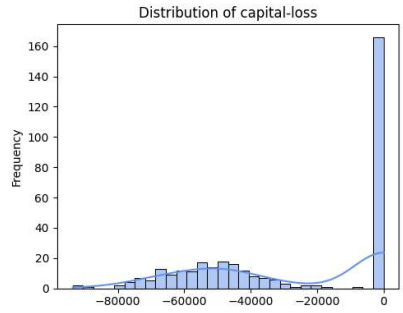

Distribution of age
Frequency

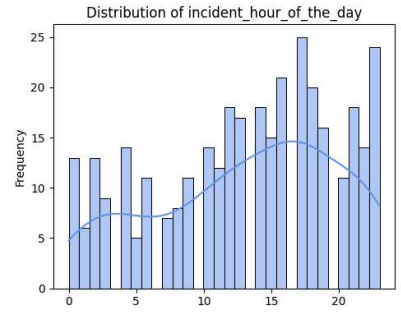

Distribution of policy_deductable
Frequency

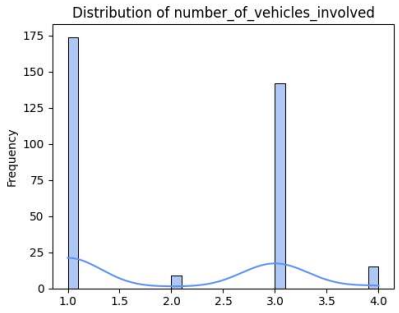

Distribution of policy_annual_premium
Frequency

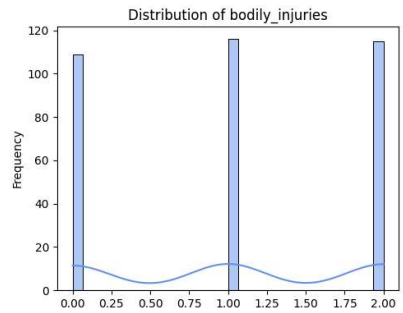

Distribution of umbrella_limit
Frequency
1e7

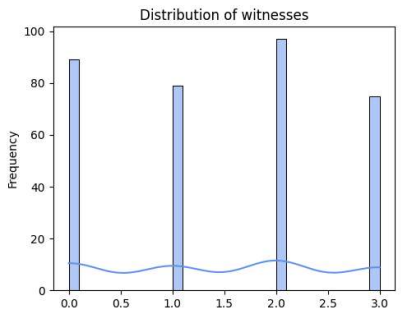

Distribution of capital-gains
Frequency

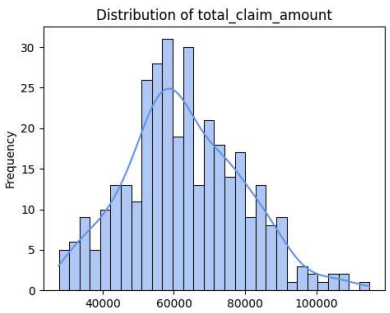

Distribution of capital-loss
Frequency

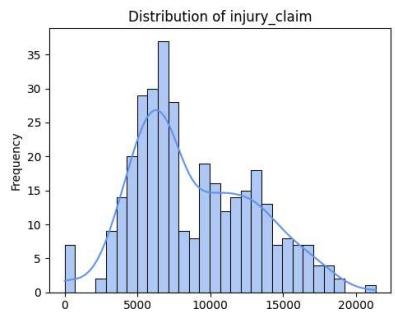

Distribution of incident_hour_of_the_day
Frequency

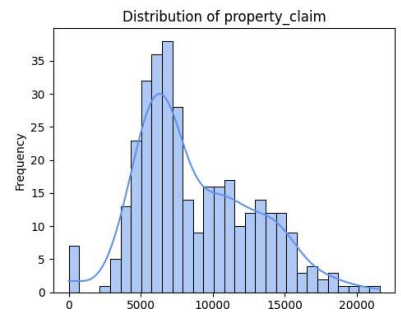

Distribution of number_of_vehicles_involved
Frequency

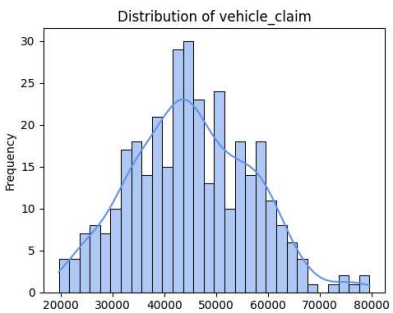

Distribution of bodily_injuries
Frequency

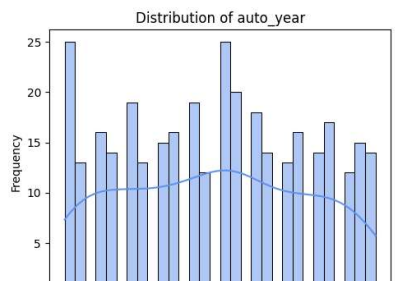

Distribution of witnesses
Frequency

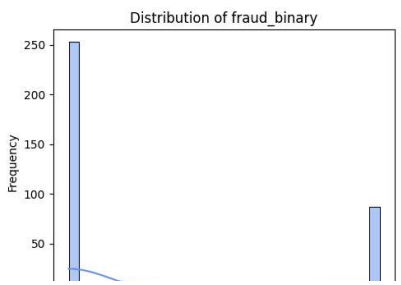

Distribution of total_claim_amount
Frequency


Distribution of injury_claim
Frequency


Distribution of property_claim
Frequency


Distribution of vehicle_claim
Frequency


Distribution of auto_year
Frequency


Distribution of fraud_binary
Frequency

In [ ]:
```
#after seeing the plots the columns are selected
```

In [476…
```
ct = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['months_as_customer', 'age','policy_annual_premi
                                   'capital-loss','total_claim_amount','injury_clai
                                   'auto_year'])        # scale numeric

    ],
    remainder='passthrough'  # or 'passthrough' to keep untouched columns
)
```

In [477…
```
import xgboost as xgb
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [ ]:
```
#defining the models for the pipeline
#we used Logistic Regression - Random Forest and the beautiful XGBoost
```

In [478…
```
models = {
    "Logistic Regression": LogisticRegression(max_iter=400),
    "Random Forest": RandomForestClassifier(),
    "XGBoost": xgb.XGBClassifier(eval_metric='mlogloss')
}
```

In [479…
```
from sklearn.pipeline import Pipeline
```

In [480…
```
from sklearn.metrics import classification_report
```

In [ ]:
```
#creating the pipeline- training the model - evaluating them and logging the result
```

In [481…
```
def log_model(model_name, model, X_train, y_train, X_test, y_test):
    mlflow.set_experiment("My Experiment") #it's just the name for mlflow to be abl
    with mlflow.start_run():

        mlflow.log_param("model", model_name) # let's Log the model parameters

        # now create the pipeline with the column transformer and classifier
        pipeline = Pipeline([
            ('preprocessor', ct),    # Apply preprocessing (scaling) to the data
            ('classifier', model)     # The classifier model
        ])


        pipeline.fit(X_train, y_train) # Training


        y_pred = pipeline.predict(X_test) #Evaluating the model
        #accuracy = accuracy_score(y_test, y_pred)
        report = classification_report(y_test, y_pred, output_dict=True)
```

```
        # Log the output metrics like accuracy, f1 score macro, recall for class 0
        mlflow.log_metrics({
            'accuracy': report['accuracy'],
            'recall_class_0': report['0']['recall'],
            'recall_class_1': report['1']['recall'],
            'f1_score_macro': report['macro avg']['f1-score']
        })


        mlflow.sklearn.log_model(pipeline, "model", input_example=X_test[:1]) # als
```

In [ ]:
```
#split the data set into 70/30 - training the models and logging them each
```

In [482…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta


# Train and log each model
for model_name, model in models.items():
    log_model(model_name, model, X_train, y_train, X_test, y_test)
```

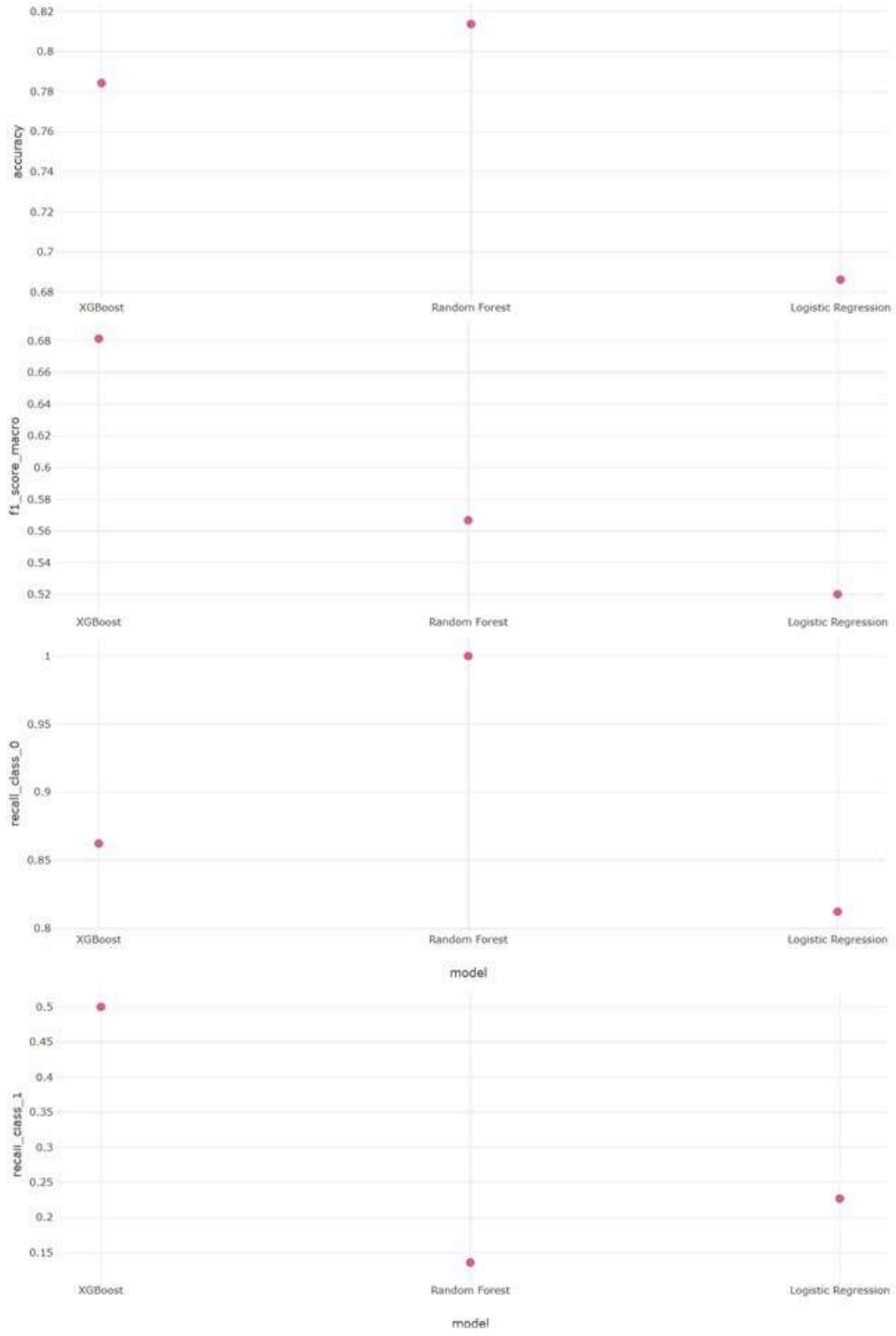
```
/home/loonycorn/mlflow/lib/python3.12/site-packages/sklearn/linear_model/_logistic.p
y:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/home/loonycorn/mlflow/lib/python3.12/site-packages/mlflow/types/utils.py:452: UserW
arning: Hint: Inferred schema contains integer column(s). Integer columns in Python
cannot represent missing values. If your input data contains missing values at infer
ence time, it will be encoded as floats and will cause a schema enforcement error. T
he best way to avoid this problem is to infer the model schema based on a realistic
data sample (training dataset) that includes missing values. Alternatively, you can
declare integer columns as doubles (float64) whenever these columns may have missing
values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/late
st/models.html#handling-integers-with-missing-values>`_ for more details.
  warnings.warn(
/home/loonycorn/mlflow/lib/python3.12/site-packages/mlflow/types/utils.py:452: UserW
arning: Hint: Inferred schema contains integer column(s). Integer columns in Python
cannot represent missing values. If your input data contains missing values at infer
ence time, it will be encoded as floats and will cause a schema enforcement error. T
he best way to avoid this problem is to infer the model schema based on a realistic
data sample (training dataset) that includes missing values. Alternatively, you can
declare integer columns as doubles (float64) whenever these columns may have missing
values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/late
st/models.html#handling-integers-with-missing-values>`_ for more details.
  warnings.warn(
/home/loonycorn/mlflow/lib/python3.12/site-packages/mlflow/types/utils.py:452: UserW
arning: Hint: Inferred schema contains integer column(s). Integer columns in Python
cannot represent missing values. If your input data contains missing values at infer
ence time, it will be encoded as floats and will cause a schema enforcement error. T
he best way to avoid this problem is to infer the model schema based on a realistic
data sample (training dataset) that includes missing values. Alternatively, you can
declare integer columns as doubles (float64) whenever these columns may have missing
values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/late
st/models.html#handling-integers-with-missing-values>`_ for more details.
  warnings.warn(
/home/loonycorn/mlflow/lib/python3.12/site-packages/mlflow/types/utils.py:452: UserW
arning: Hint: Inferred schema contains integer column(s). Integer columns in Python
cannot represent missing values. If your input data contains missing values at infer
ence time, it will be encoded as floats and will cause a schema enforcement error. T
he best way to avoid this problem is to infer the model schema based on a realistic
data sample (training dataset) that includes missing values. Alternatively, you can
declare integer columns as doubles (float64) whenever these columns may have missing
values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/late
st/models.html#handling-integers-with-missing-values>`_ for more details.
  warnings.warn(
/home/loonycorn/mlflow/lib/python3.12/site-packages/mlflow/types/utils.py:452: UserW
arning: Hint: Inferred schema contains integer column(s). Integer columns in Python
cannot represent missing values. If your input data contains missing values at infer
ence time, it will be encoded as floats and will cause a schema enforcement error. T
he best way to avoid this problem is to infer the model schema based on a realistic
data sample (training dataset) that includes missing values. Alternatively, you can
declare integer columns as doubles (float64) whenever these columns may have missing
```

```
values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-with-missing-values>`_ for more details.
  warnings.warn(
/home/loonycorn/mlflow/lib/python3.12/site-packages/mlflow/types/utils.py:452: UserWarning: Hint: Inferred schema contains integer column(s). Integer columns in Python cannot represent missing values. If your input data contains missing values at inference time, it will be encoded as floats and will cause a schema enforcement error. The best way to avoid this problem is to infer the model schema based on a realistic data sample (training dataset) that includes missing values. Alternatively, you can declare integer columns as doubles (float64) whenever these columns may have missing values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-with-missing-values>`_ for more details.
  warnings.warn(
```

In [ ]:
```
#here is the result I saved as a photo captured from Mlflow
```

In [ ]:
```
from IPython.display import HTML

HTML('<img src="F.jpg" width="800">')
```

Out[ ]:


accuracy
0.82
0.8
0.78
0.76
0.74
0.72
0.7
0.68
XGBoost
Random Forest
Logistic Regression
f1_score_macro
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.54
0.52
XGBoost
Random Forest
Logistic Regression
recall_class_0
1
0.95
0.9
0.85
0.8
XGBoost
Random Forest
Logistic Regression
model
recall_class_1
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
XGBoost
Random Forest
Logistic Regression
model

In [ ]: