Fraudulantés

Average citizens by day - taking down the worlds fraudsters by night

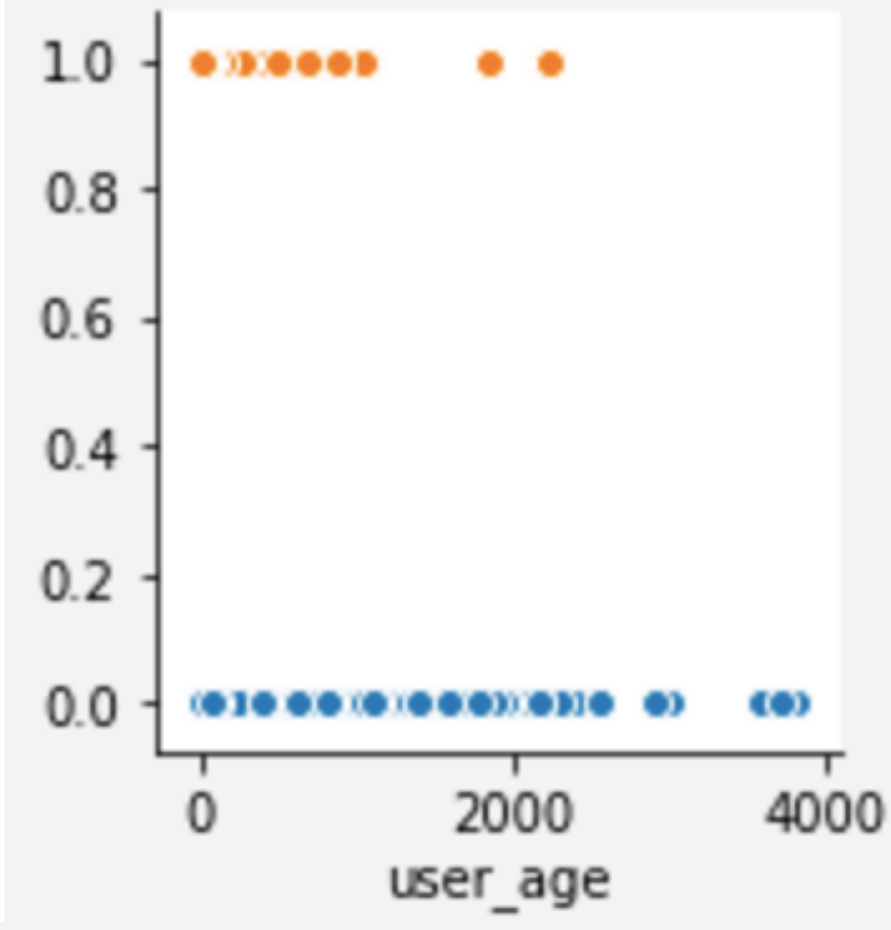
Brought in part to you by - Rand, Ryan, Tyler and Alex

HEROKU SERVER FRAUD

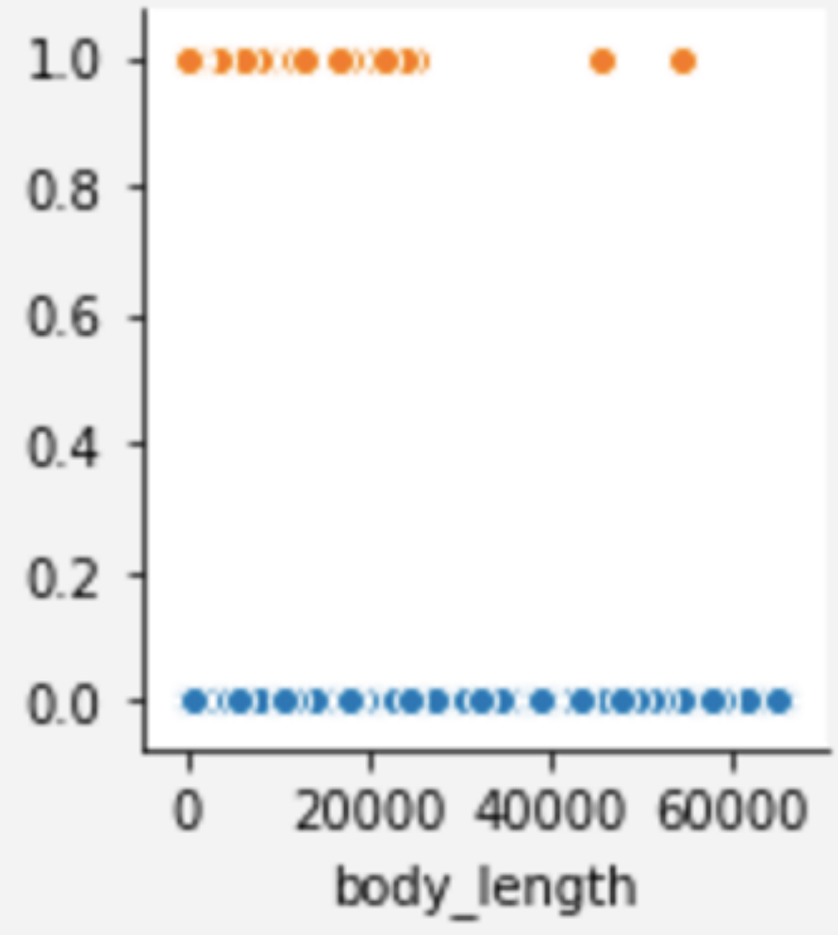
EDA

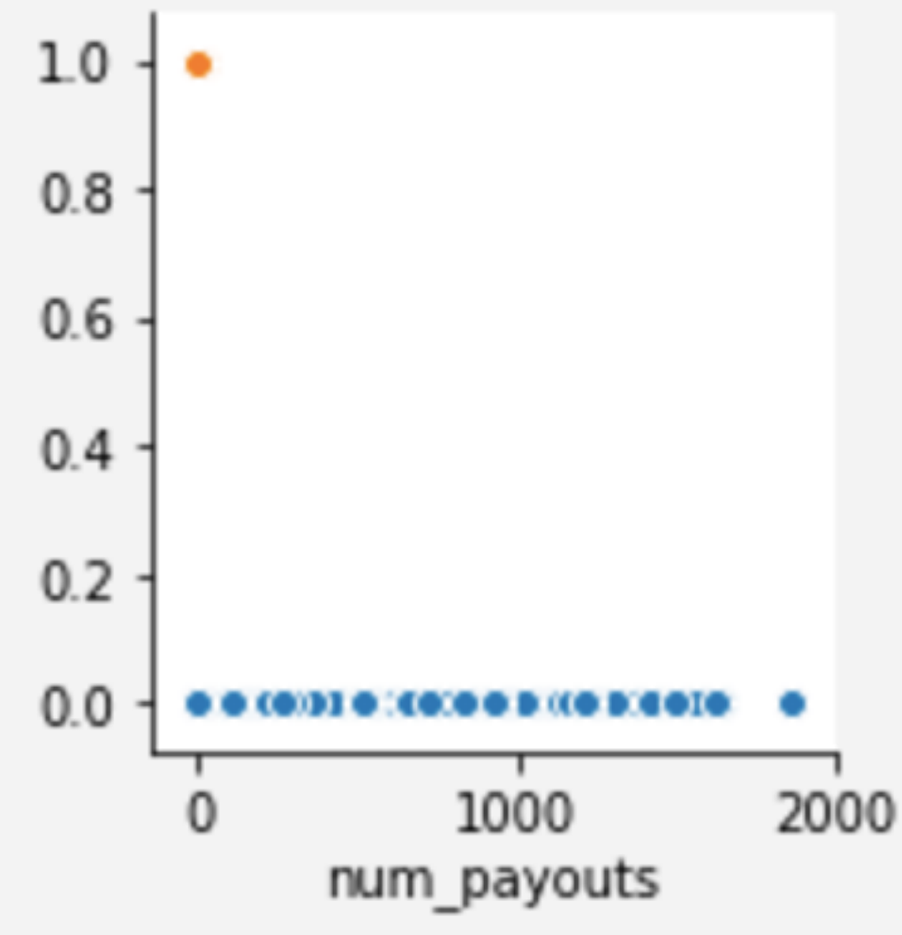
	0	1
acct_type	fraudster_event	premium
approx_payout_date	1266062400	1296720000
body_length	3852	3499
channels	5	0
country	US	US
currency	USD	USD
delivery_method	0	1
description	<p><a href="http://s432.photobucket.com/albums...	<p>Join us for a quick, one-night, community-
email_domain	gmail.com	ruf.org
event_created	1262739706	1293832670
event_end	1265630400	1296288000
event_published	1.26311E+09	1.29383E+09
event_start	1265594400	1296255600
fb_published	0	0
gts	0	868.02
has_analytics	0	0
has_header	1	0
has_logo	0	1
listed	y	n
name	99 HOUR "NO SLEEP" SUPER BOWL CELEBRITY WEEKEN...	Winthrop RUF Winter Getaway
name_length	60	27
num_order	0	23

	0	1
num_payouts	0	1
object_id	527017	786878
org_desc		<p>Since 1987,
org_facebook	0	0
org_name	Party Starz Ent & Diverse Int'l	RUF at Winthrop
org_twitter	0	12
payee_name		RUF
payout_type	0	1
previous_payouts	[]	[{'name': 'RUF',
sale_duration	29	28
sale_duration2	33	28
show_map	1	0
ticket_types	[{'event_id': 527017, 'cost': 25.0,	[{'event_id':
user_age	36	149
user_created	2009-11-30 20:45:50	2010-08-04
user_type	1	3
venue_address	717 Washington Avenue	
venue_country	US	US
venue_latitude	25.7775	32.7766
venue_longitude	-80.1334	-79.9309
venue_name	INK Nightclub - South Beach	The Charleston, SC
venue_state	FL	SC
hour	20	17



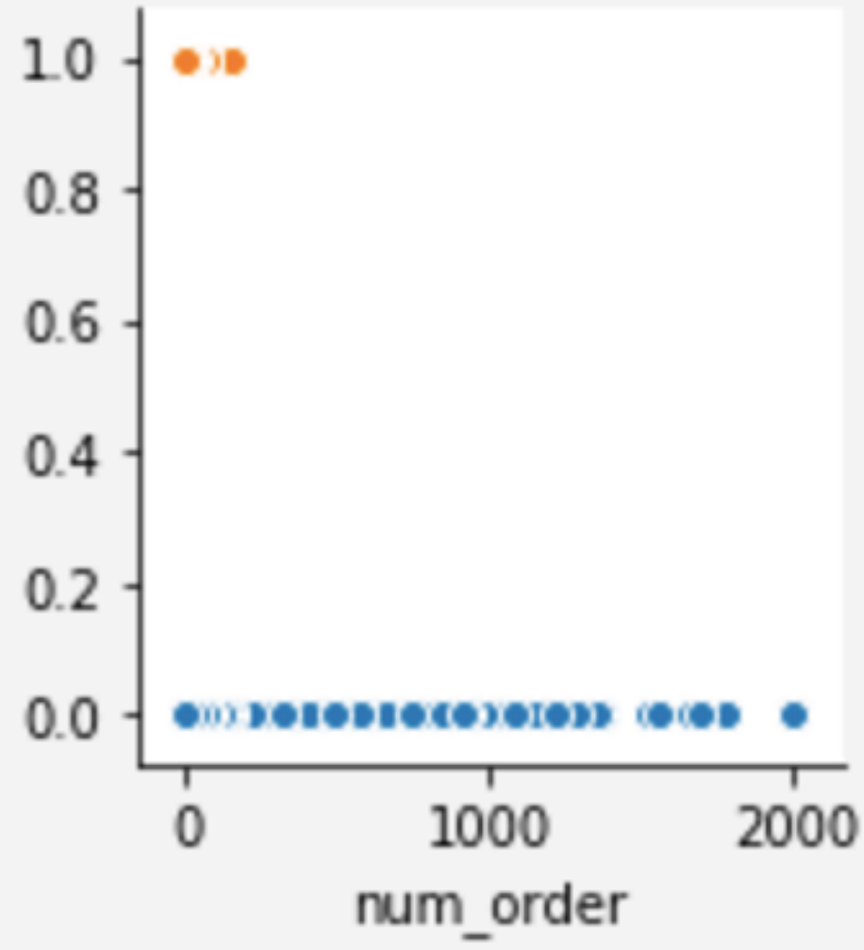
Distinguishable Pairplot Fraud & Feature Interactions

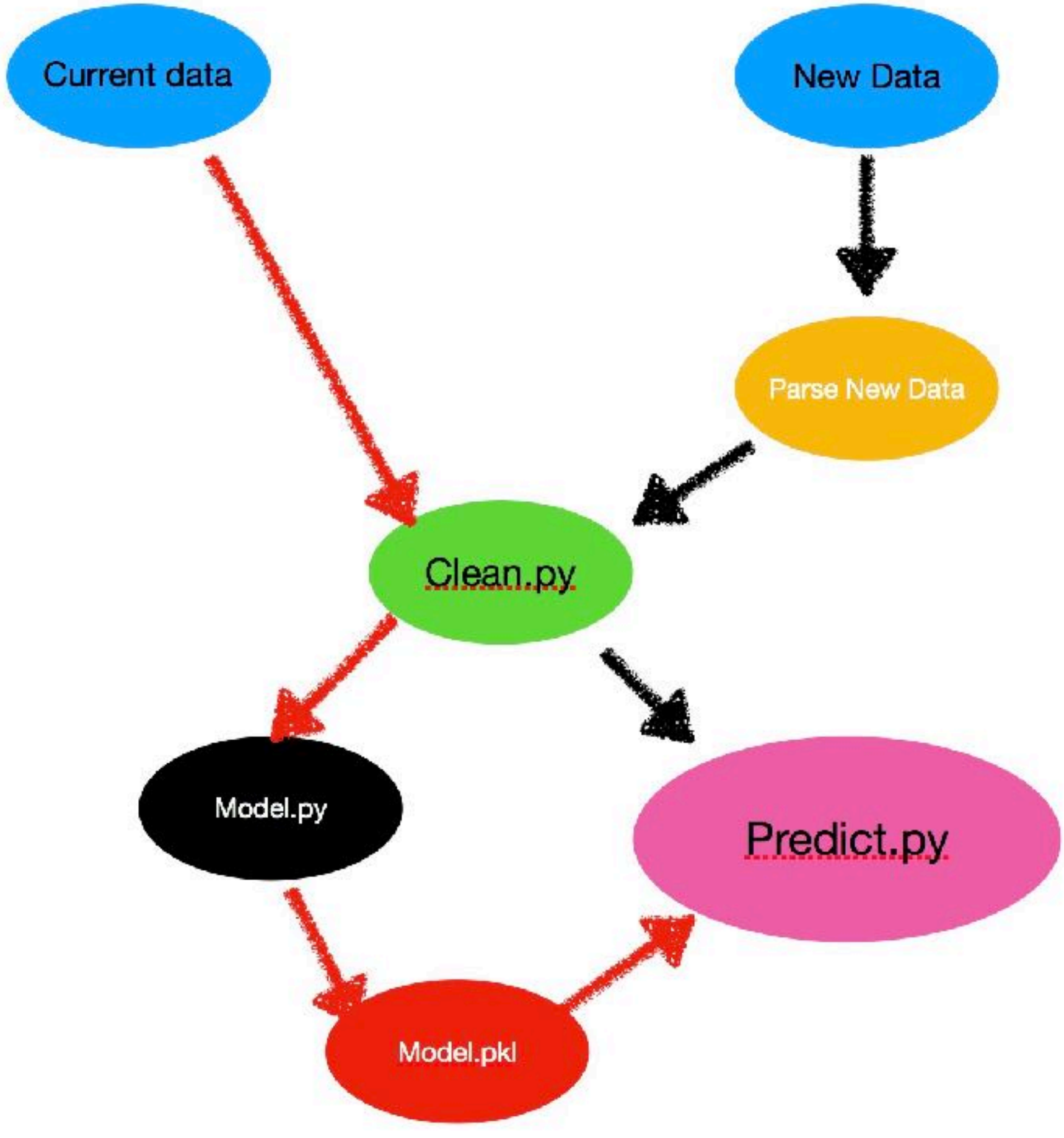




Features Not Seen

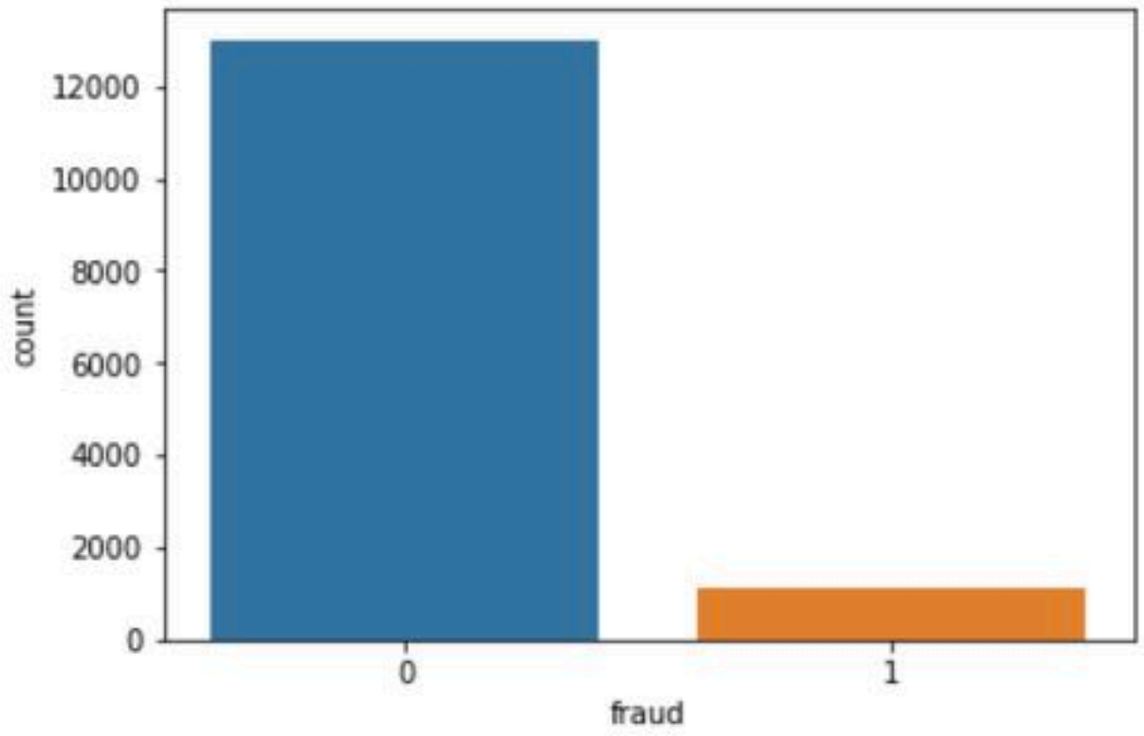
- user_type
- body_length
- name_length
- gts
- previous_payouts



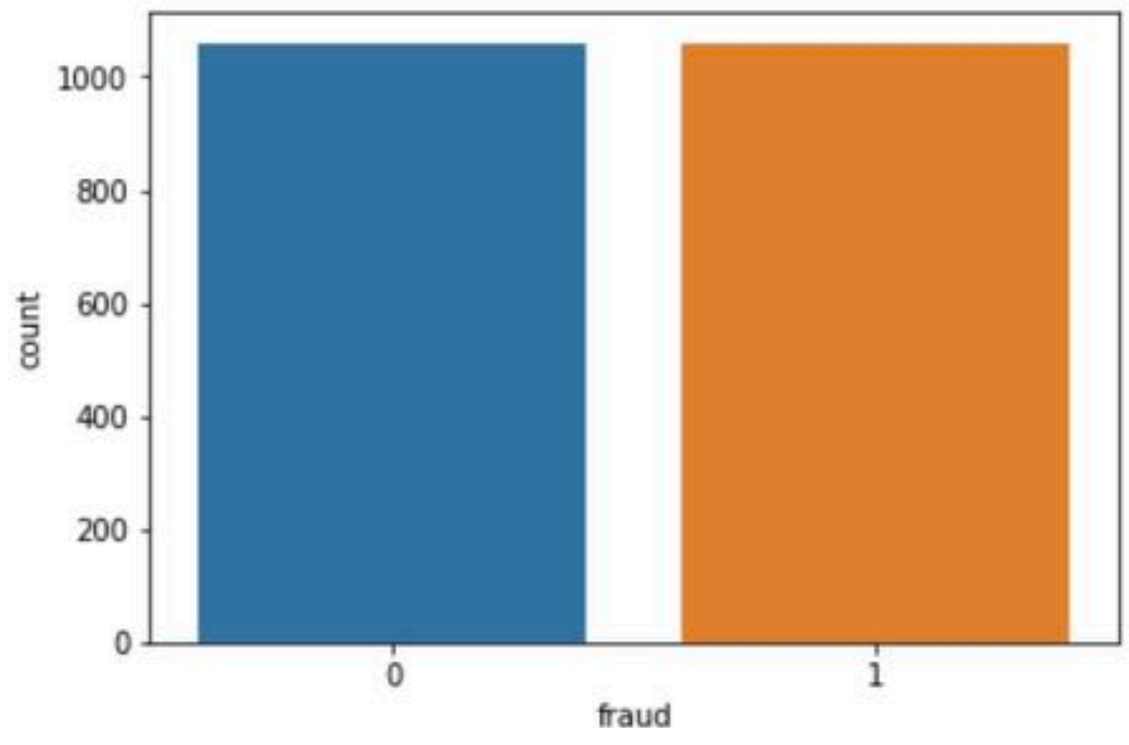

Current data
New Data
Parse New Data
Clean.py
Model.py
Predict.py
Model.pkl

Finding a Balance

data imbalance fraud vs non-fraud



RandomUnderSampler



Modeling the data

- Naive Bayes
- Logistic Regression
- Gradient Boost

Naive Bayes

Naive Bayes

```
In [ ]:
1 from sklearn.naive_bayes import GaussianNB
2 gnb = GaussianNB()
3 gauss_fit = gnb.fit(X_train, y_train)
4 gauss_pred = gnb.predict(X_test)
5 print_stuff(gauss_fit, gauss_pred)
```

```
classification report

             precision    recall  f1-score   support

          0       0.99      0.65      0.78       918
          1       0.17      0.94      0.29        72

avg / total       0.93      0.67      0.75       990

confusion matrix

[[594 324]
 [  4  68]]

Score: 0.668686868687
```

F1 Score: .75

F1 score

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Confusion Matrix

	Predicted: NOT FRAUD	Predicted: FRAUD			
Isn't Fraud	594	4	598	99%	Specificity
Is Fraud	324	68	392	17.3%	Recall
	918	72	990	66.9%	Accuracy
		94.4%	328	33.1%	Error Rate
		Precision		39.6%	Prevalence

Logistic Regression

Logistic Regression

```
1 from sklearn.linear_model import LogisticRegression
2 lr = LogisticRegression()
3 lr_fit = lr.fit(X_train, y_train)
4 lr_pred = lr.predict(X_test)
5 print_stuff(lr_fit, lr_pred)
```

```
classification report

             precision    recall  f1-score   support

          0       0.99      0.82      0.90       918
          1       0.29      0.93      0.44        72

avg / total       0.94      0.83      0.87       990

confusion matrix

[[755 163]
 [  5  67]]

Score: 0.830303030303
```

F1 Score: .83

F1 score

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Confusion Matrix

	Predicted: NOT FRAUD	Predicted: FRAUD			
Isn't Fraud	755	5	760	99%	Specificity
Is Fraud	163	67	230	29.1%	Recall
	918	72	990	83.0%	Accuracy
		93.1%	168	17.0%	Error Rate
		Precision		23.2%	Prevalence

Decision Tree

Decision Tree Classifier

```
In [ ]:
1 from sklearn.tree import DecisionTreeClassifier
2 dtree = DecisionTreeClassifier()
3 dtree_fit = dtree.fit(X_train, y_train)
4 dtree_pred = dtree.predict(X_test)
5 print_stuff(dtree_fit, dtree_pred)
```

```
classification report

             precision    recall  f1-score   support

          0       0.99      0.88      0.93       917
          1       0.36      0.89      0.52        73

avg / total       0.94      0.88      0.90       990

confusion matrix

[[803 114]
 [  8  65]]

Score: 0.876767676768
```

F1 Score: .88

F1 score

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Confusion Matrix

	Predicted: NOT FRAUD	Predicted: FRAUD			
Isn't Fraud	803	8	811	99%	Specificity
Is Fraud	114	65	179	36.3%	Recall
	917	73	990	87.7%	Accuracy
		89.0%	122	12.3%	Error Rate
		Precision		18.1%	Prevalence

Gradient Boost

F1 Score: .93

F1 score

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Gradient Boosting Classifier

```
:
1 from sklearn.ensemble import GradientBoostingClassifier
2 GBC = GradientBoostingClassifier(n_estimators=150, max_depth=4)
3
4 GBC_fit = GBC.fit(X_train, y_train)
5 GBC_pred = GBC.predict(X_test)
6 print_stuff(GBC_fit, GBC_pred)
```

```
classification report

             precision    recall  f1-score   support

          0       0.99      0.93      0.96       917
          1       0.51      0.86      0.64        73

avg / total       0.95      0.93      0.94       990

confusion matrix

[[857  60]
 [ 10  63]]

Score: 0.929292929293
```

Confusion Matrix

	Predicted: NOT FRAUD	Predicted: FRAUD			
Isn't Fraud	857	10	867	99%	Specificity
Is Fraud	60	63	123	51.2%	Recall
	917	73	990	92.9%	Accuracy
		86.3%	70	7.1%	Error Rate
		Precision		12.4%	Prevalence

Random Forest

F1 Score: .94

F1 score

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Random Forest Classifier

]:
```
1 from sklearn.ensemble import RandomForestClassifier
2 k = 150
3 rfc = RandomForestClassifier(n_estimators=k)
4 rfc_fit = rfc.fit(X_train, y_train)
5 rfc_pred = rfc.predict(X_test)
6 print_stuff(rfc_fit, rfc_pred)
```

```
classification report

             precision    recall  f1-score   support

          0       1.00      0.93      0.96       918
          1       0.53      0.94      0.68        72

avg / total       0.96      0.94      0.94       990

confusion matrix

[[858  60]
 [  4  68]]

Score: 0.935353535354
```

Confusion Matrix

	Predicted: NOT FRAUD	Predicted: FRAUD			
Isn't Fraud	858	4	862	100%	Specificity
Is Fraud	60	68	128	53.1%	Recall
	918	72	990	93.5%	Accuracy
		94.4%	64	6.5%	Error Rate
		Precision		12.9%	Prevalence

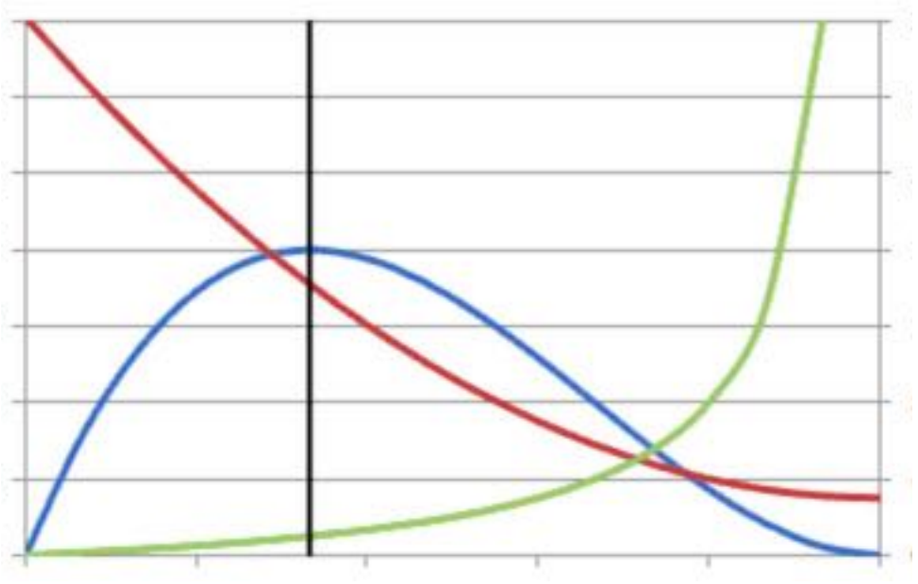
COST ANALYSIS

Average Payment per customer: $1100

Average Churn for investigated Customer: 1 in 100

Average Cost of False Positive: $220

Average Cost False Negative: $1100



$85k, avg $1.5k, avg/transaction

Confusion Matrix

	Predicted: NOT FRAUD	Predicted: FRAUD			
Isn't Fraud	858	4	862	100%	Specificity
Is Fraud	60	68	128	53.1%	Recall
	918	72	990	93.5%	Accuracy
		94.4%	64	6.5%	Error Rate
		Precision		12.9%	Prevalence

Cost Matrix

	Predicted: NOT FRAUD	Predicted: FRAUD		
Isn't Fraud	0	400	400	Loss from FP
Is Fraud	132	74800	74932	Saving from TP
Potential Risk, 60 Customers				

990	Customers	
858	Non Fraudulent	TN
60	Fraud Not Caught	FN
4	Not Fraud Accused	FP
68	Fraud Caught	TP

```
sample_amount = df.previous_payouts.iloc[4]
2
sample_amount[1]
```

Out[70]:

```
{'address': '249 A Street',
 'amount': 20.0,
 'country': 'US',
 'created': '2011-01-26 01:11:29',
 'event': 1206495,
 'name': 'Arts and Business Council or Greater Boston',
 'state': 'MA',
 'uid': 8205253,
 'zip_code': '02210'}
```

```
df['total_dols'].mean()
Out[106]:
85546.06

df['avg_dols'].mean()
Out[107]:
1561.34
```

```
def get_amount_tot_dols(test):
    '''
    list from one record containing previous payments
    '''
    count = len(test)
    amount = []
    amount_t = 0
    length = len(test)
    if count == 0: return 0
    else:
        for i in range(count):
            amount.append(test[i]['amount'])
            amount_t += float(test[i]['amount'])

        return amount_t
def get_amount_avg_dols(test):
    '''
    list from one record containing previous payments
    '''
    count = len(test)
    amount = []
    amount_t = 0
    length = len(test)
    if count == 0: return 0
    else:
        for i in range(count):
            amount.append(test[i]['amount'])
            amount_t += float(test[i]['amount'])

        return round(amount_t/count)

def get_amount_no_trans(test):
    '''
    list from one record containing previous payments
    '''
    count = len(test)
    amount = []
    amount_t = 0
    length = len(test)
    if count == 0: return 0
    else:
        for i in range(count):
            amount.append(test[i]['amount'])
```