In [1]:
```
import pickle
```

In [7]:
```
X = pickle.load( open( "save.p", "rb" ) )
```

In [8]:
```
X.shape
```

Out[8]: (590538, 22)

In [9]:
```
y = pickle.load( open( "save2.p", "rb" ) )
# favorite_color is now { "lion": "yellow", "kitty": "red" }
```

In [10]:
```
y.shape
```

Out[10]: (590538,)

In [99]:
```
X_train = pickle.load(open( "X_train.p", "rb" ) )
```

In [100…
```
X_test = pickle.load(open( "X_test.p", "rb" ) )
```

In [87]:
```
y_train = pickle.load(open( "y_train.p", "rb" ) )
```

In [15]:
```
y_test = pickle.load(open( "y_test", "rb" ) )
```

In [16]:
```
X_val = pickle.load(open( "X_val", "rb" ) )
```

In [17]:
```
y_val = pickle.load(open( "y_val", "rb" ) )
```

In [19]:
```
y_val[57034]
```

Out[19]: 1

In [ ]:
```

```

In [20]:
```
import pandas as pd
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
import seaborn as sns
%config InlineBackend.figure_formats = ['svg']
```

```
plt.rcParams['figure.figsize'] = (9, 6)
sns.set(context='notebook', style='whitegrid', font_scale=1.2)
```

In [28]:
```
################
## IMBALANCED ##
################
```

In [21]:
```
import statsmodels.api as sm
```

In [26]:
```
# For this first example, we'll employ statsmodels
lm_1 = sm.Logit(y_train,  # with statsmodels, `y` comes first
                sm.add_constant(X_train[['hour']]))  # and then `x`
lm_1 = lm_1.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.152238
         Iterations 7
```

In [29]:
```
X_train.shape
```

Out[29]:
```
(354322, 22)
```

In [30]:
```
y_train.shape
```

Out[30]:
```
(354322,)
```

In [27]:
```
lm_1.summary()
```

Out[27]:

Logit Regression Results

Dep. Variable:	isFraud	**No. Observations:**	354322
Model:	Logit	**Df Residuals:**	354320
Method:	MLE	**Df Model:**	1
Date:	Sun, 07 Feb 2021	**Pseudo R-squ.:**	0.0006584
Time:	13:53:21	**Log-Likelihood:**	-53941.
converged:	True	**LL-Null:**	-53977.
Covariance Type:	nonrobust	**LLR p-value:**	3.433e-17

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.1761	0.018	-175.841	0.000	-3.211	-3.141
hour	-0.0100	0.001	-8.490	0.000	-0.012	-0.008

In [31]:
```
from sklearn.linear_model import LogisticRegression
```



```
lm_1 = LogisticRegression(solver='newton-cg',  # For comparison, use the same so
```

```
                         C=100000)  # No regularization

lm_1.fit(X_train[['hour']], y_train)
```

Out[33]:
```
LogisticRegression(C=100000, solver='newton-cg')
```

In [34]:
```
print('intercept: ', round(lm_1.intercept_[0], 4))
print('hour coef: ', round(lm_1.coef_[0][0], 4))
```

```
intercept:  -3.1761
hour coef:  -0.01
```

In [35]:
```
df_eval = X_test.copy()
df_eval['pred'] = lm_1.predict(X_test[['hour']])
#df_eval.loc[:, 'pred'] = df_eval['pred'].astype('category')
df_eval['correct_pred'] = df_eval['pred'] == y_test
```

In [36]:
```
df_eval.head()
```

Out[36]:

	P_emaildomain_gmail	P_emaildomain_none	P_emaildomain_yahoo	ProductCD_C	Produc
458896	0	0	0	0.0	
535881	0	0	0	0.0	
353211	1	0	0	0.0	
502338	1	0	0	0.0	
425874	1	0	0	0.0	

5 rows × 24 columns

In [37]:
```
df_eval.loc[df_eval['correct_pred']== False]
```

Out[37]:

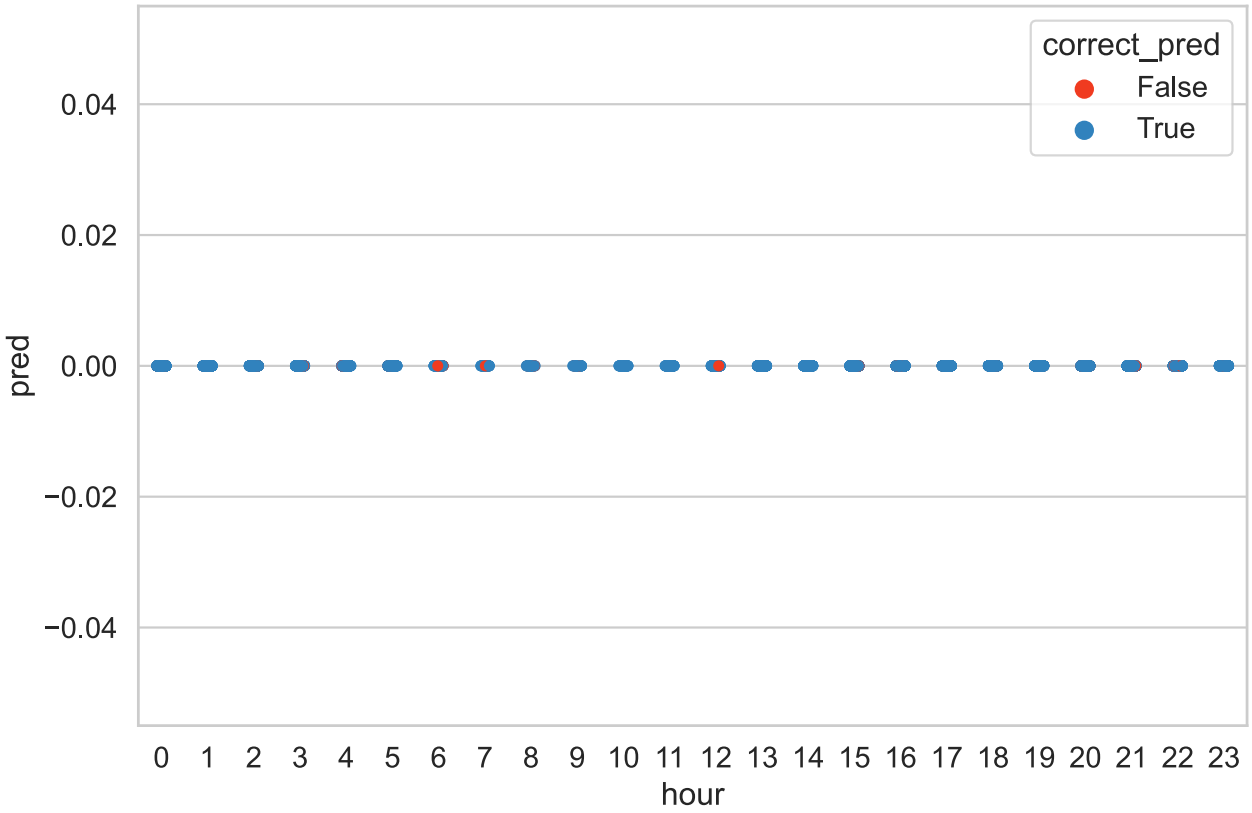
	P_emaildomain_gmail	P_emaildomain_none	P_emaildomain_yahoo	ProductCD_C	Produc
172731	1	0	0	1.0	
16356	0	0	0	1.0	
375739	1	0	0	0.0	
432486	1	0	0	1.0	
32558	1	0	0	0.0	
...	...	...	...	...	
349541	0	0	0	1.0	
144173	1	0	0	1.0	
483114	0	0	0	0.0	
			0	0.0	

	P_emaildomain_gmail	P_emaildomain_none	P_emaildomain_yahoo	ProductCD_C	Produc
144008	0	0	0	1.0	

4064 rows × 24 columns

In [ ]:

In [38]:
```
sns.stripplot(data=df_eval.sample(10000),
              x='hour',
              y='pred',
              hue='correct_pred',
              palette={False: '#f03b20', True: '#3182bd'});
```



In [40]:
```
lm_1.predict_proba(X_test[['hour']])[:5]
```

Out[40]:
```
array([[0.95992481, 0.04007519],
       [0.96427459, 0.03572541],
       [0.96723487, 0.03276513],
       [0.96427459, 0.03572541],
       [0.96594963, 0.03405037]])
```

In [41]:
```
df_eval['probability_fraud'] = lm_1.predict_proba(X_test[['hour']])[:, 1]
```

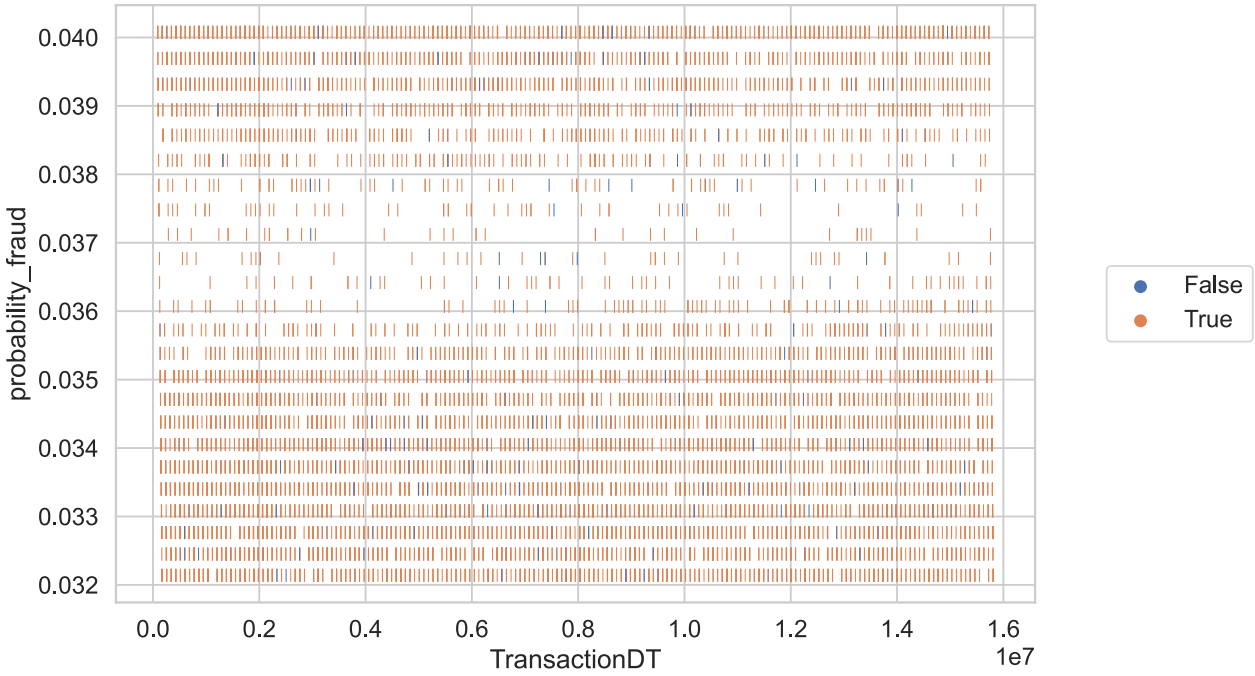


```
ple(10000),
                x='TransactionDT',
```

```
                    y='probability_fraud',
                    hue='correct_pred',
                    #y_jitter=0.2,
                    #x_jitter=0.2,
                    marker='|',
                    s=50);

g.legend(loc='right', bbox_to_anchor=(1.25, 0.5), ncol=1)
```

Out[66]: <matplotlib.legend.Legend at 0x7febf73722b0>

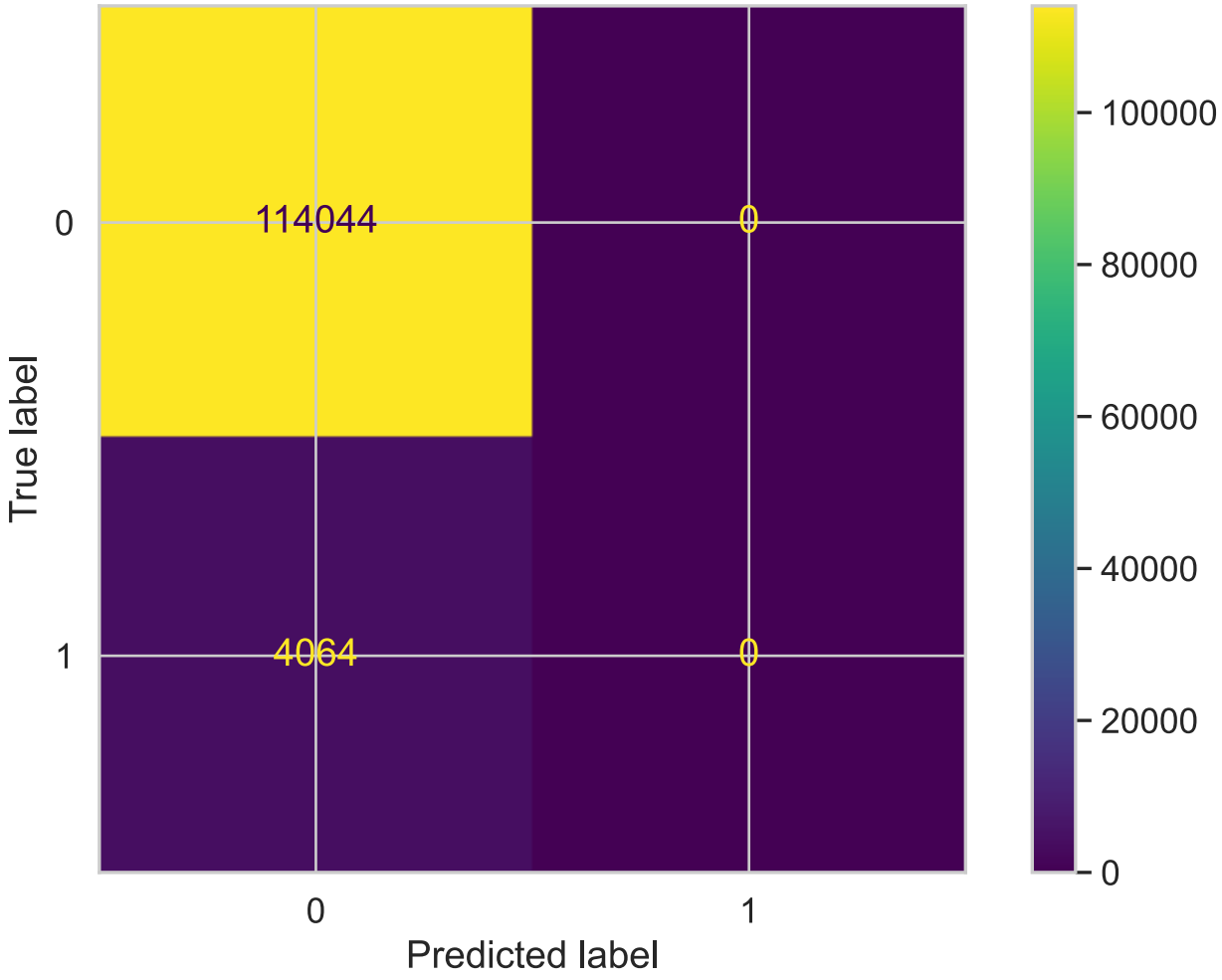


In [67]:
```
from sklearn.metrics import roc_auc_score, confusion_matrix, roc_curve
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
#df_eval['correct_pred'] = df_eval['pred'] == y_test
confusion_matrix(df_eval['in_sf'], df_eval['pred'])
```

In [69]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

In [70]:
```
#plt.grid(False)
cm = confusion_matrix(y_test, df_eval['pred'])
#plt.grid(False)
cm_display = ConfusionMatrixDisplay(cm).plot()
```

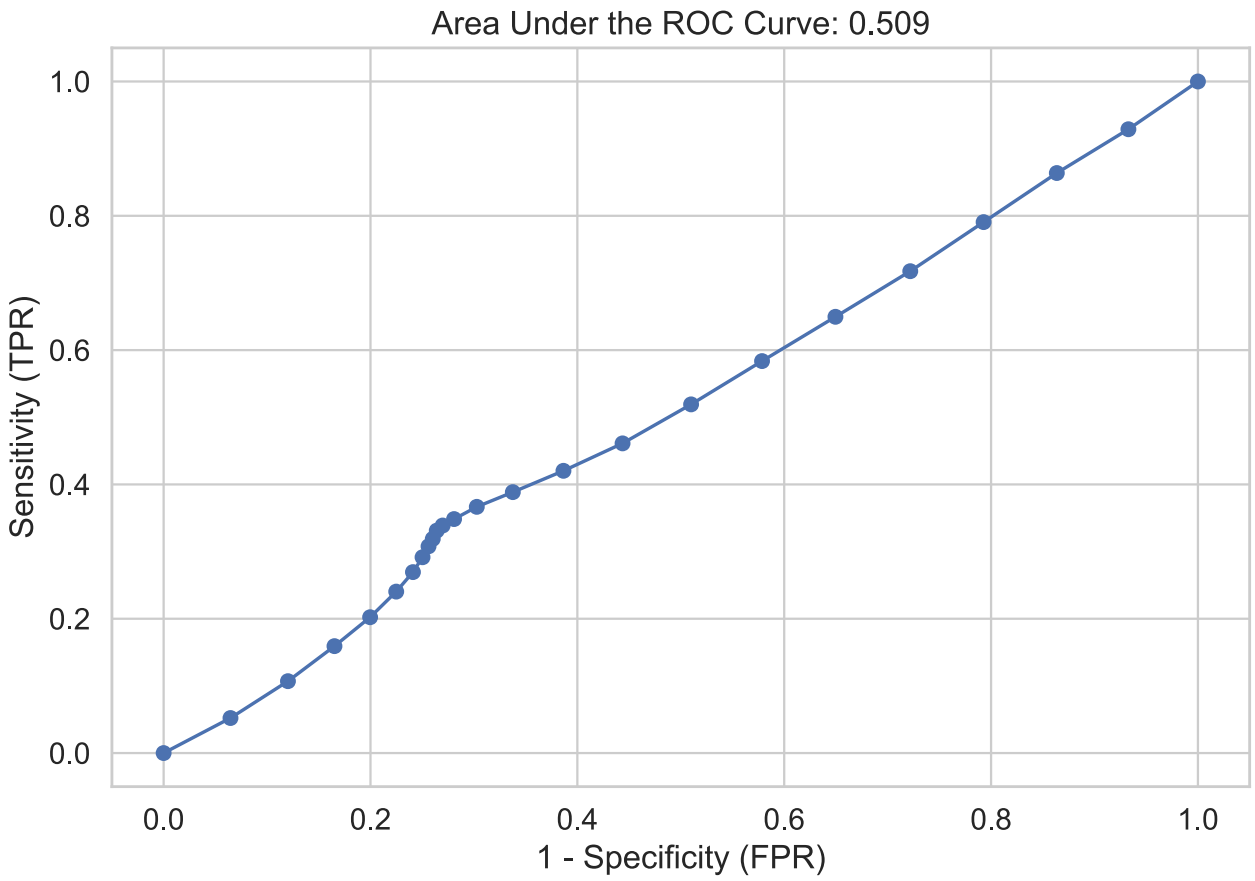


In [71]:
```
fpr, tpr, thresholds = roc_curve(y_test,
                                 df_eval['probability_fraud'])
```

In [72]:
```
def plot_roc(true, probas):
    auc = roc_auc_score(true, probas)

    plt.plot(fpr, tpr, marker='o')
    plt.xlabel('1 - Specificity (FPR)')
    plt.ylabel('Sensitivity (TPR)');
    plt.title(f"Area Under the ROC Curve: {round(auc, 3)}");
```

In [73]:
```
plot_roc(y_test, df_eval['probability_fraud'])
```

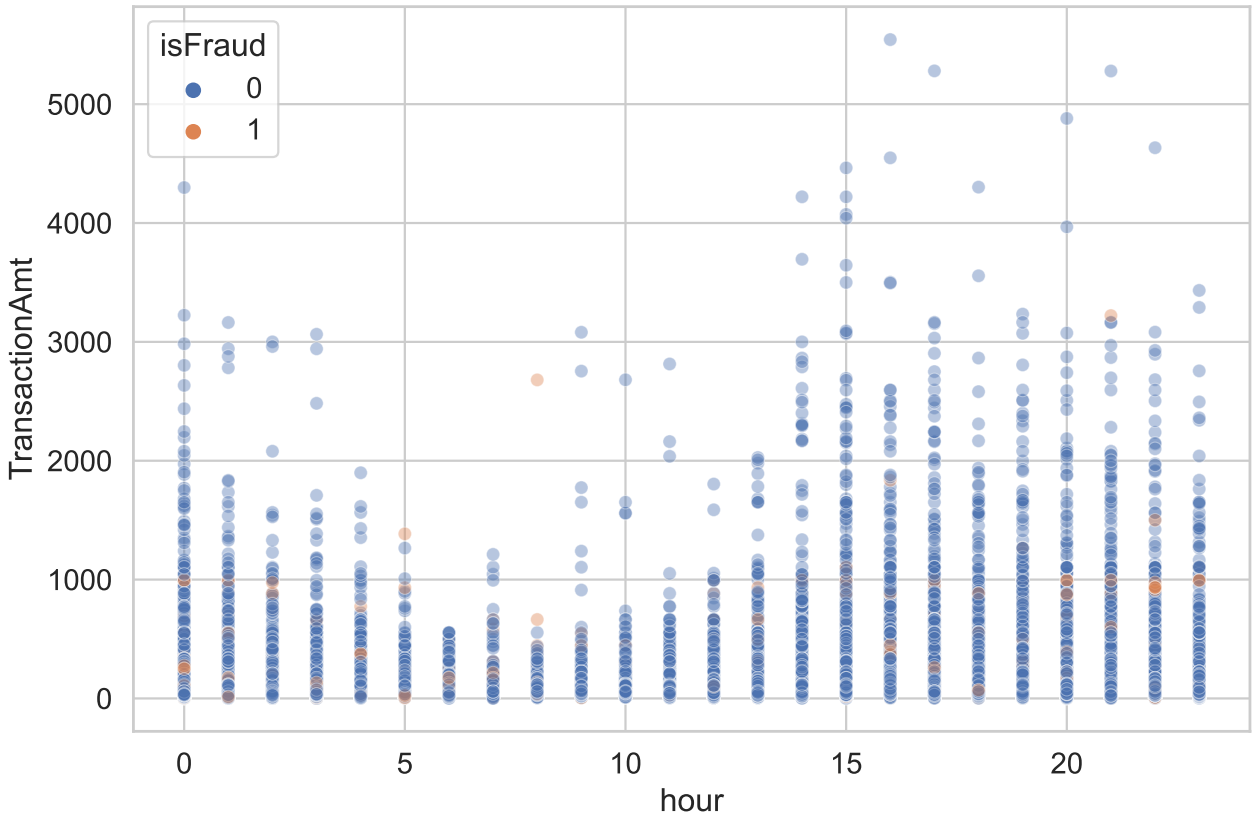


In [74]:
```
mask = tpr > 0.9
thresholds[mask].max()
```

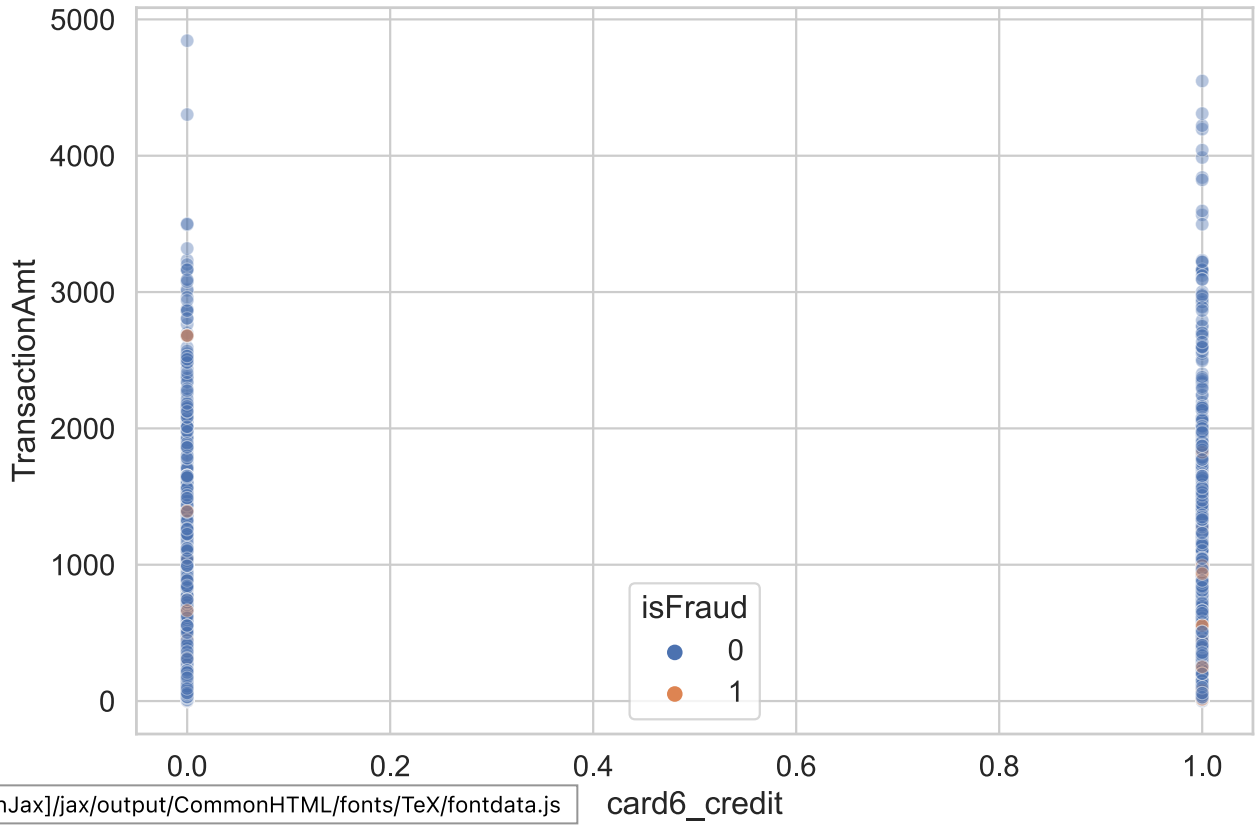
Out[74]:
```
0.03245122004427823
```

In [ ]:
```
##############################
## Multiple Log Regression ##
##############################
```

In [82]:
```
sns.scatterplot(data=X_train.sample(50000),
                x='hour',
                y='TransactionAmt',
                hue=y_train,
               alpha = .4);
```



In [81]:
```
sns.scatterplot(data=X_train.sample(50000),
                x='card6_credit',
                y='TransactionAmt',
                hue=y_train,
               alpha = .4);
```



In [101…
```
features = ['TransactionAmt', 'hour']

# Since we're using more than one feature, let's scale our features
scaler = StandardScaler()
```

In [102…
```
X_train = scaler.fit_transform(X_train[features])
y_train = y_train
```

In [103…
```
X_train[:4]
```

Out[103…
```
array([[-0.44689737,  0.80843146],
       [-0.30483636,  0.41409468],
       [-0.42540554, -1.03180684],
       [-0.32632819,  1.20276824]])
```

In [104…
```
lm_2 = LogisticRegression()  # We'll also regularize our features
```

In [105…
```
lm_2.fit(X_train, y_train)
```

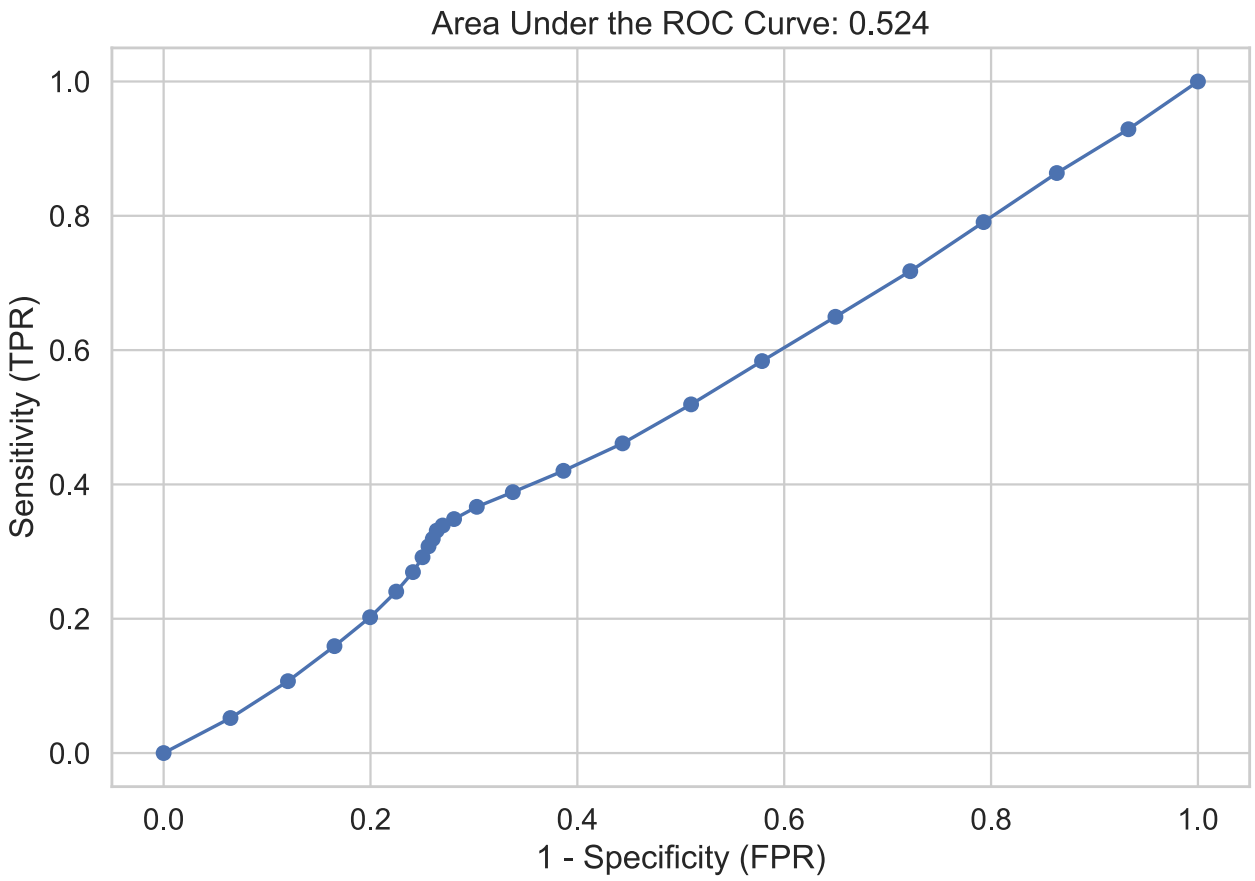
Out[105…
```
LogisticRegression()
```

In [106…
```
X_test = scaler.transform(X_test[features])
preds = lm_2.predict(X_test)
```

In [107…
```
confusion_matrix(y_test,
                 preds)
```

Out[107…
```
array([[114044,      0],
       [  4064,      0]])
```

In [108…
```
plot_roc(y_test, lm_2.predict_proba(X_test)[:, 1])
```



In [ ]:
```
####################################
## Multi-Class Logistic Regression ##
####################################
```

In [112…
```
from sklearn.preprocessing import OneHotEncoder
from sklearn import datasets
```

In [113…
```
lm_ovr = LogisticRegression(solver='newton-cg', multi_class='ovr')
lm_mn = LogisticRegression(solver='newton-cg', multi_class='multinomial')
```

In [115…
```
print(X_train.shape)
print(y_train.shape)
```

```
(354322, 2)
(354322,)
```

In [116…
```
lm_ovr.fit(X_train, y_train)
lm_mn.fit(X_train, y_train)
```

Out[116… `LogisticRegression(multi_class='multinomial', solver='newton-cg')`

In [117…
```
preds_ovr = lm_ovr.predict(X_test)
preds_mn = lm_mn.predict(X_test)
```

In [118…
```
confusion_matrix(y_test,
                 preds_ovr)
```

Out[118…
```
array([[114044,      0],
       [  4064,      0]])
```

In [119…
```
confusion_matrix(y_test,
                 preds_mn)
```

Out[119…
```
array([[114044,      0],
       [  4064,      0]])
```

In [120…
```
preds_proba_ovr = lm_ovr.predict_proba(X_test)
preds_proba_mn = lm_mn.predict_proba(X_test)
```

In [ ]:
```
def get_multiclass_aucs(labels, scores, name='One-vs-Rest', kind='ovr'):
    ohe = OneHotEncoder()
    labels_ohe = ohe.fit_transform(labels)
    labels_ohe = labels_ohe.toarray()

    print(f'Average: {roc_auc_score(labels_ohe, scores, multi_class=kind)}')

    auc_scores = roc_auc_score(labels_ohe, scores, multi_class=kind, average=Non
    auc_scores = {i:s for i, s in enumerate(auc_scores)}

    return auc_scores
```