1.) Import the Credit Card Fraud Data From CCLE

```
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import numpy as np
```

```
drive.mount('/content/gdrive/', force_remount = True)
```

```
Mounted at /content/gdrive/
```

```
df = pd.read_csv("/content/gdrive/MyDrive/Econ441B/fraudTest.csv")
```

```
df.head()
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	a
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.

5 rows × 23 columns

```
df.columns
```

```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category',
       'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip',
       'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud'],
      dtype='object')
```

```
df.dtypes
```

```
Unnamed: 0                 int64
trans_date_trans_time     object
cc_num                     int64
merchant                  object
category                  object
amt                      float64
first                     object
last                      object
gender                    object
street                    object
city                      object
state                     object
zip                        int64
lat                      float64
long                     float64
city_pop                   int64
job                       object
dob                       object
trans_num                 object
unix_time                  int64
merch_lat                float64
merch_long               float64
is_fraud                   int64
dtype: object
```

```
df['dob'] = pd.to_datetime(df['dob'])
```

```
df['dob'].dt.year.head()
```

```
0    1968
1    1990
2    1970
3    1987
4    1955
Name: dob, dtype: int64
```

```
df['year_birth'] = df['dob'].dt.year
```

```
df['customer_classification'] = df['year_birth'].copy()
# df["customer_classification"] = df["customer_classification"].astype(int)
```

```
# ya = 0, midd = 1, eld = 2
```

```
df.loc[df['customer_classification'] < 1960, 'customer_classification'] = 2
df.loc[(df['customer_classification'] <= 1985) & (df['customer_classification']>= 1960), 'cus
df.loc[(df['customer_classification'] <= 2005) & (df['customer_classification']>= 1986), 'cus


df.loc[df['customer_classification'] == 2, 'customer_classification'] = 'elderly'
df.loc[df['customer_classification'] == 1, 'customer_classification'] = 'middle_aged'
df.loc[df['customer_classification'] == 0, 'customer_classification'] = 'young_adult'
```

I created a customer classification variable here based on the customers date of birth. My rational was that I wanted to see if certain age brackets, such as the elderly, got targeted more by fraudsters. This variable will be used in the model seen in the coming problems.

```
df
```

st	last	gender	street	...	city_pop	job	dob	
eff	Elliott	M	351 Darlene Green	...	333497	Mechanical engineer	1968-03-19	2da90c7d74bd46a0caf3777
ne	Williams	F	3638 Marsh Union	...	302	Sales professional, IT	1990-01-17	324cc204407e99f51b0d6ca
ley	Lopez	F	9333 Valentine Point	...	34496	Librarian, public	1970-10-21	c81755dbbbea9d5c77f0943

32941

```
df_copy = df.copy()
```

552

2.) Select four columns to use as features (one just be trans_date_trans)

... ...

```
df_select = df[["trans_date_trans_time", "customer_classification", "amt", "city_pop", "is_fr
```

Estates planner 02-13

```
df_select.columns
```

```
Index(['trans_date_trans_time', 'customer_classification', 'amt', 'city_pop',
       'is_fraud'],
      dtype='object')
```

Islands 11-29

3.) Create your own variable out of trans_date. Create dummies for factor vars

830

```
type(df_select["trans_date_trans_time"][0])
```

```
str
```

```
df_select["trans_date_trans_time"] = pd.to_datetime(df_select["trans_date_trans_time"])
```

```
<ipython-input-14-99f721e4ce0f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  df_select["trans_date_trans_time"] = pd.to_datetime(df_select["trans_date_trans_time"]
```

```
dir(df_select["trans_date_trans_time"][0])
```

```
 'freq',
 'freqstr',
 'fromisocalendar',
 'fromisoformat',
 'fromordinal',
 'fromtimestamp',
 'hour',
 'is_leap_year',
 'is_month_end',
 'is_month_start',
 'is_quarter_end',
 'is_quarter_start',
 'is_year_end',
 'is_year_start',
 'isocalendar',
 'isoformat',
 'isoweekday',
 'max',
 'microsecond',
 'min',
 'minute',
 'month',
 'month_name',
 'nanosecond',
 'normalize',
 'now',
 'quarter',
 'replace',
 'resolution',
 'round',
 'second',
 'strftime',
 'strptime',
 'time',
 'timestamp',
 'timetuple',
 'timetz',
 'to_datetime64',
 'to_julian_date',
 'to_numpy',
 'to_period',
 'to_pydatetime',
 'today',
 'toordinal',
 'tz',
 'tz_convert',
 'tz_localize',
 'tzinfo',
 'tzname',
 'utcfromtimestamp',
```

```
 'utcnow',
 'utcoffset',
 'utctimetuple',
 'value',
 'week',
 'weekday',
 'weekofyear',
 'year']
```

```
df_select["time_var"] = [i.hour for i in df_select["trans_date_trans_time"]]
```

```
<ipython-input-15-f84357779188>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  df_select["time_var"] = [i.hour for i in df_select["trans_date_trans_time"]]
```

```
X = pd.get_dummies(df_select, ["customer_classification"]).drop(["trans_date_trans_time", "is_
y = df["is_fraud"]
```

```
X.head()
```

	amt	city_pop	time_var	customer_classification_elderly	customer_classification_r
0	2.86	333497	12	0	
1	29.84	302	12	0	
2	41.28	34496	12	0	
3	60.05	54767	12	0	
4	3.19	1126	12	1	

```
resample_X = X
resample_y = y
```

5.) Train a Logistic regression.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_normalized = scaler.fit_transform(resample_X)
```

```
from sklearn.linear_model import LogisticRegression
```

```
log_reg = LogisticRegression().fit(X_normalized, resample_y)
```

6.) The company you are working for wants to target at a False Positive rate of 5%. What threshold should you use? (Use oversampled data)

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
import numpy as np

# train and fit logistic regression model
log_reg = LogisticRegression().fit(X_normalized, resample_y)

# predict class probabilities
probs = log_reg.predict_proba(X_normalized)[:, 1]

# calculate TPR and FPR for different threshold values
fpr, tpr, thresholds = roc_curve(resample_y, probs)

# target a specific false positive rate
target_fpr = 0.05

# find the threshold that corresponds to the target FPR
idx = np.where(fpr <= target_fpr)
threshold = thresholds[idx][-1]

# make predictions using the threshold
y_pred = probs >= threshold
```

```
print('Threshold:',threshold)
```

```
Threshold: 0.005278875377888082
```

If the company we are working for wants to target a False Positive rate of 5%, I found that we should use a 0.005278875377888082 threshold. Anything smaller than this results in a FP % greater than 5%, which is what we wanted to avoid. However, with the threshold this low, I believe we are adding a lot of potential bias to the model

7.) If the company makes .02*amt on True transactions and

```
df_temp = df_select.copy()
```

```
df_temp.head()
```

	trans_date_trans_time	customer_classification	amt	city_pop	is_fraud	time_var
0	2020-06-21 12:14:25	middle_aged	2.86	333497	0	12
1	2020-06-21 12:14:33	young_adult	29.84	302	0	12
2	2020-06-21 12:14:53	middle_aged	41.28	34496	0	12
3	2020-06-21 12:15:15	young_adult	60.05	54767	0	12
4	2020-06-21 12:15:17	elderly	3.19	1126	0	12

```
df_temp["pred"] = log_reg.predict(resample_X)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:443: UserWarning: X has feature r
  warnings.warn(
```

```
df_temp = df_temp[["pred", "is_fraud", "amt"]]
```

```
df_temp.loc[df_temp['pred'] == 1].count()
```

```
pred        27991
is_fraud    27991
amt         27991
dtype: int64
```

```
df_temp.loc[df_temp['pred'] == 0].count()
```

```
pred        527728
is_fraud    527728
amt         527728
dtype: int64
```

```
df_temp.head()
```

	pred	is_fraud	amt
0	0	0	2.86
1	0	0	29.84

```
df_tempc = df_temp.copy()

df_tempc['Altered_amt'] = df_tempc['amt'].copy()


df_tempc.loc[(df_tempc['pred'] == 0) & (df_tempc['is_fraud'] ==0), 'Altered_amt'] *= 1.2
df_tempc.loc[(df_tempc['pred'] == 1) & (df_tempc['is_fraud'] ==0), 'Altered_amt'] *= -1
df_tempc.loc[(df_tempc['pred'] == 1) & (df_tempc['is_fraud'] ==1), 'Altered_amt'] *= -1
df_tempc.loc[(df_tempc['pred'] == 0) & (df_tempc['is_fraud'] ==1), 'Altered_amt'] *= -1


df_tempc.head()
```

	pred	is_fraud	amt	Altered_amt
0	0	0	2.86	3.432
1	0	0	29.84	35.808
2	0	0	41.28	49.536
3	0	0	60.05	72.060
4	0	0	3.19	3.828

```
round(df_tempc['Altered_amt'].sum(),2)
```

```
31393011.84
```

If the company makes .02*amt on True transactions and loses -amt on False, the company would make $31,393,011.84. This number was calculated on the assumption that we only make money if there's no fraud and we predict no fraud, while we lose the full amount in every other scenario.

8.) Using Logistic Regression Lasso to inform you. Would you use the selected features in a trusted prediction model?

```
# If most or all your variables go to 0 => Your data is garbage
# The regularization will tell us if our model has significance
# This of using coefficient strength similar to r^2
```

```
lasso_logreg = LogisticRegression(penalty='l1', solver='liblinear')
lasso_logreg.fit(X_normalized, resample_y)
coefs = lasso_logreg.coef_
```

```
coefs
```

```
array([[ 0.2841281 , -0.10558985,  0.185432  ,  0.13599199,  0.        ,
        -0.02016318]])
```

As seen above, the most of the coefficients are nonzero, which indicates that the model has significance and we could use the selected features in a trusted prediction model.