In [4]:
```
%run Imports.ipynb
```

Introduction

This report uses logistic regression to determine the log odds relationship between the presence of government responses to covid-19 and the presence of cases of the virus.

Import dataset.

In [21]:
```
df=Json('df1: government responses to covid19')
df.excel('Gov_Responses2Covid19_last.xlsx', 'Dataset')
df1=json_storage['df1: government responses to covid19'][1]
df1
```

Out[21]:

	country	geoid	iso	d	cases	deaths	school	school_local	domestic	domestic_local	...	wage	credit	taxc	taxd	export	rate	Rigidity_Public_Health	Economic_Measures	population_2019	continent
0	Aruba	AW	ABW	2020-01-01 00:00:00	0.0	0.0	0.0	0.0	NaN	NaN	...	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.000000	106310.0	America
1	Aruba	AW	ABW	2020-01-02 00:00:00	0.0	0.0	0.0	0.0	NaN	NaN	...	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.000000	106310.0	America
2	Aruba	AW	ABW	2020-01-03 00:00:00	0.0	0.0	0.0	0.0	NaN	NaN	...	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.000000	106310.0	America
3	Aruba	AW	ABW	2020-01-04 00:00:00	0.0	0.0	0.0	0.0	NaN	NaN	...	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.000000	106310.0	America
4	Aruba	AW	ABW	2020-01-05 00:00:00	0.0	0.0	0.0	0.0	NaN	NaN	...	0.0	0.0	0.0	0.0	0.0	0.0	NaN	0.000000	106310.0	America
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
62695	Hong Kong	HK	HKG	2020-09-29 00:00:00	NaN	NaN	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	1.0	1.0	1.0	0.461539	0.714286	NaN	NaN
62696	Hong Kong	HK	HKG	2020-09-30 00:00:00	NaN	NaN	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	1.0	1.0	1.0	0.461539	0.714286	NaN	NaN
62697	Macau	MO	MAC	2020-09-30 00:00:00	NaN	NaN	1.0	0.0	0.0	0.0	...	1.0	1.0	1.0	0.0	0.0	1.0	0.250000	0.714286	NaN	NaN
62698	Hong Kong	HK	HKG	2020-10-01 00:00:00	NaN	NaN	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	1.0	1.0	1.0	0.461539	0.714286	NaN	NaN
62699	Macau	MO	MAC	2020-10-01 00:00:00	NaN	NaN	1.0	0.0	0.0	0.0	...	1.0	1.0	1.0	0.0	0.0	1.0	0.250000	0.714286	NaN	NaN

62700 rows × 43 columns

Data Mining

In [123…
```
df1.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62700 entries, 0 to 62699
Data columns (total 43 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   country                 62700 non-null  object 
 1   geoid                   62425 non-null  object 
 2   iso                     62700 non-null  object 
 3   d                       62700 non-null  object 
 4   cases                   57750 non-null  float64
 5   deaths                  57750 non-null  float64
 6   school                  45758 non-null  float64
 7   school_local            45758 non-null  float64
 8   domestic                55275 non-null  float64
 9   domestic_local          55275 non-null  float64
 10  travel                  55275 non-null  float64
 11  travel_partial          55275 non-null  float64
 12  travel_dom              55275 non-null  float64
 13  travel_dom_partial      55275 non-null  float64
 14  curf                    55275 non-null  float64
 15  curf_partial            55275 non-null  float64
 16  mass                    55275 non-null  float64
 17  mass_partial            55275 non-null  float64
 18  elect                   14834 non-null  float64
 19  elect_partial           14834 non-null  float64
 20  sport                   55275 non-null  float64
 21  sport_partial           55275 non-null  float64
 22  rest                    55275 non-null  float64
 23  rest_local              55275 non-null  float64
 24  testing                 55275 non-null  float64
 25  testing_narrow          55275 non-null  float64
 26  masks                   51459 non-null  float64
 27  masks_partial           51459 non-null  float64
 28  surveillance            55275 non-null  float64
 29  surveillance_partial    55275 non-null  float64
 30  state                   55275 non-null  float64
 31  state_partial           55275 non-null  float64
 32  cash                    54450 non-null  float64
 33  wage                    54450 non-null  float64
 34  credit                  54450 non-null  float64
 35  taxc                    54450 non-null  float64
 36  taxd                    54450 non-null  float64
 37  export                  54450 non-null  float64
 38  rate                    54450 non-null  float64
 39  Rigidity_Public_Health  55275 non-null  float64
 40  Economic_Measures       54450 non-null  float64
 41  population_2019         57750 non-null  float64
 42  continent               61875 non-null  object 
dtypes: float64(38), object(5)
memory usage: 20.6+ MB
```

Data Cleaning

Cleans government responses to covid 19 dataframe by replacing undesired values with desired ones, and droping undesired columns.

In [22]:
```
df1=df1.replace(np.nan, 0)
df1=df1.drop('geoid', axis=1)
df1=df1.drop('iso', axis=1)
df1=df1.drop('country', axis=1)
df1=df1.drop('continent', axis=1)
```

```
df1=df1.drop('d', axis=1)
df1=df1.drop('deaths', axis=1)
df1=df1.drop('population_2019', axis=1)
df1=df1.drop('school_local', axis=1)
df1=df1.drop('domestic_local', axis=1)
df1=df1.drop('travel_partial', axis=1)
df1=df1.drop('travel_dom_partial', axis=1)
df1=df1.drop('curf_partial', axis=1)
df1=df1.drop('mass_partial', axis=1)
df1=df1.drop('elect_partial', axis=1)
df1=df1.drop('sport_partial', axis=1)
df1=df1.drop('rest_local', axis=1)
df1=df1.drop('testing_narrow', axis=1)
df1=df1.drop('masks_partial', axis=1)
df1=df1.drop('surveillance_partial', axis=1)
df1=df1.drop('state_partial', axis=1)
df1=df1.drop('Rigidity_Public_Health', axis=1)
df1=df1.drop('Economic_Measures', axis=1)
df1
```

Out[22]:

	cases	school	domestic	travel	travel_dom	curf	mass	elect	sport	rest	...	masks	surveillance	state	cash	wage	credit	taxc	taxd	export	rate
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
62695	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0
62696	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0
62697	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	...	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0
62698	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0
62699	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	...	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0

62700 rows × 21 columns

The model will be determining whether cases of the virus were present. Makes all values where cases were above 0 into 1.

In [27]:

```
df1['cases'].values[df1['cases'] > 0] = 1
```

Government Regulation Features

The Governments' responses to COVID19 are the measures implemented by governments worldwide in response to the Coronavirus pandemic. There are two types of measures: public health measures and economic measures. .

The variables are :

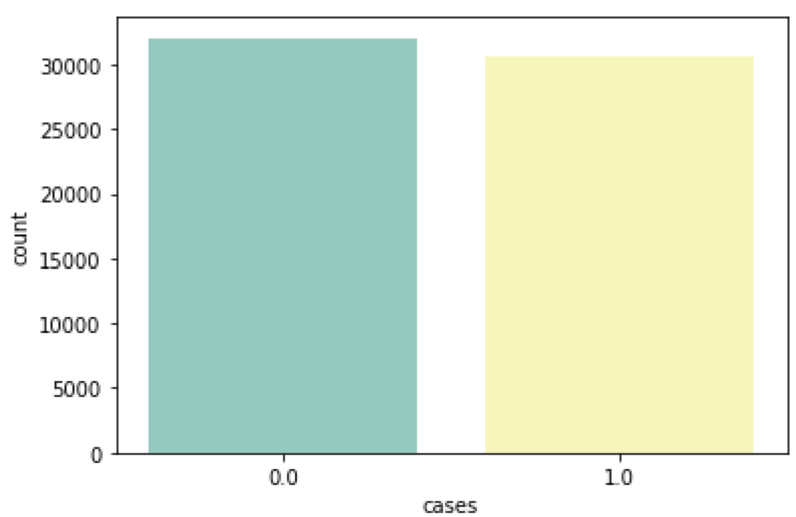
- cases: binary variable equal to 1 if there were cases of SARS-CoV-2 and 0 otherwise;
- school: binary variable equal to 1 if schools were closed and 0 otherwise;
- domestic: binary variable equal to 1 if there was a domestic lockdown and 0 otherwise;

- travel: binary variable equal to 1 if travel restrictions were implemented and 0 otherwise;
- travel_dom: binary variable equal to 1 if travel restrictions within the country (e.g. inter-region travels) were implemented and 0 otherwise;
- curf: binary variable equal to 1 if a curfew was implemented and 0 otherwise;
- mass: binary variable equal to 1 if bans on mass gatherings were implemented and 0 otherwise;
- elect: binary variable equal to 1 if some elections were postponed and 0 otherwise;
- sport: binary variable equal to 1 if bans on sporting and large events were implemented and 0 otherwise;
- rest: binary variable equal to 1 if restaurants were closed and 0 otherwise;
- testing: binary variable equal to 1 if there was a public testing policy and 0 otherwise;
- surveillance: binary variable equal to 1 if mobile app or bracelet surveillance was implemented and 0 otherwise;
- masks: binary variable equal to 1 if the obligations to wear masks in public spaces was implemented and 0 otherwise;
- state: binary variable equal to 1 if the state of emergency is declared and 0 otherwise;
- cash: binary variable equal to 1 if cash transfers are implemented and 0 otherwise;
- wage: binary variable equal to 1 if wage support is implemented and 0 otherwise;
- credit: binary variable equal to 1 if credit schemes are implemented and 0 otherwise;
- taxc: binary variable equal to 1 if tax credits are implemented and 0 otherwise;
- taxd: binary variable equal to 1 if tax delays are implemented and 0 otherwise;
- export: binary variable equal to 1 if supports to importers or exporters are implemented and 0 otherwise;
- rate: binary variable equal to 1 if the Central Bank lowered the interest rates and 0 otherwise;

Data Exploration

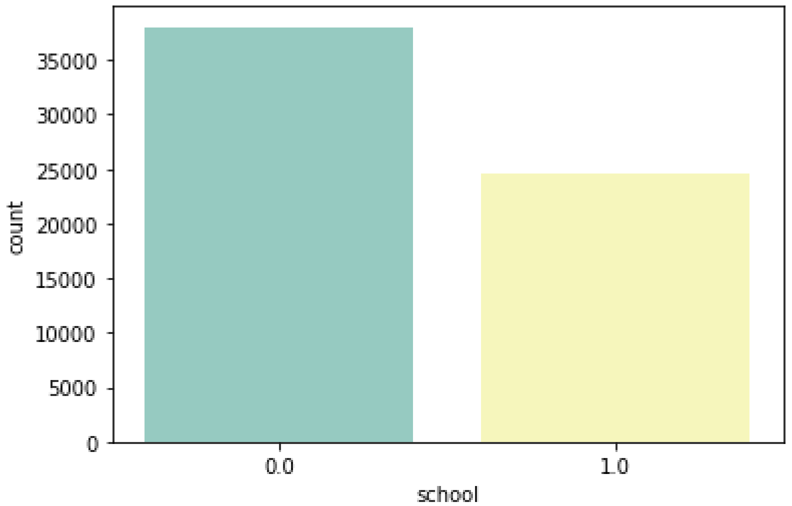
In [39]:
```
sns.countplot(df1['cases'], palette='Set3')
```

Out[39]:
```
<AxesSubplot:xlabel='cases', ylabel='count'>
```



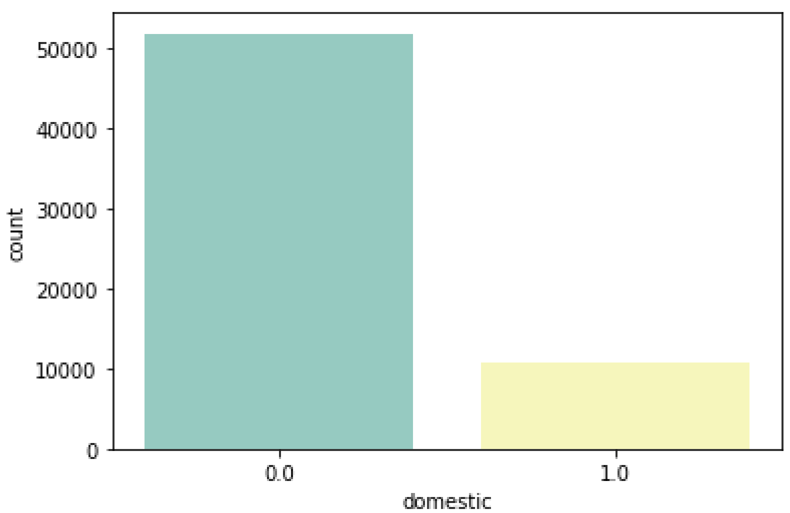
In [34]:
```
sns.countplot(df1['school'], palette='Set3')
```

Out[34]:
```
<AxesSubplot:xlabel='school', ylabel='count'>
```



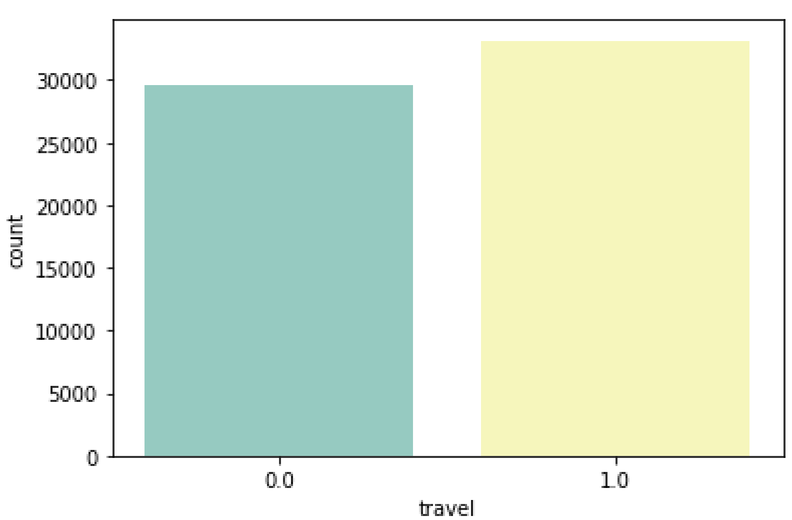

```
In [43]: sns.countplot(df1['domestic'], palette='Set3')
```

```
Out[43]: <AxesSubplot:xlabel='domestic', ylabel='count'>
```



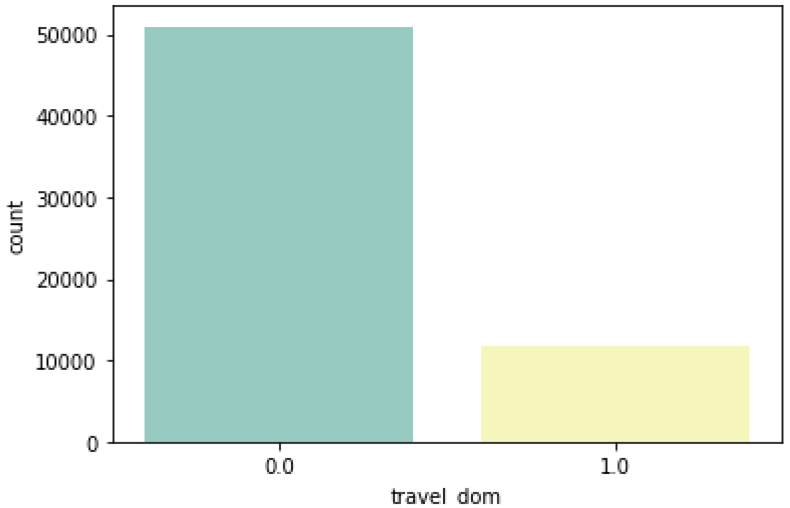

```
In [44]: sns.countplot(df1['travel'], palette='Set3')
```

```
Out[44]: <AxesSubplot:xlabel='travel', ylabel='count'>
```



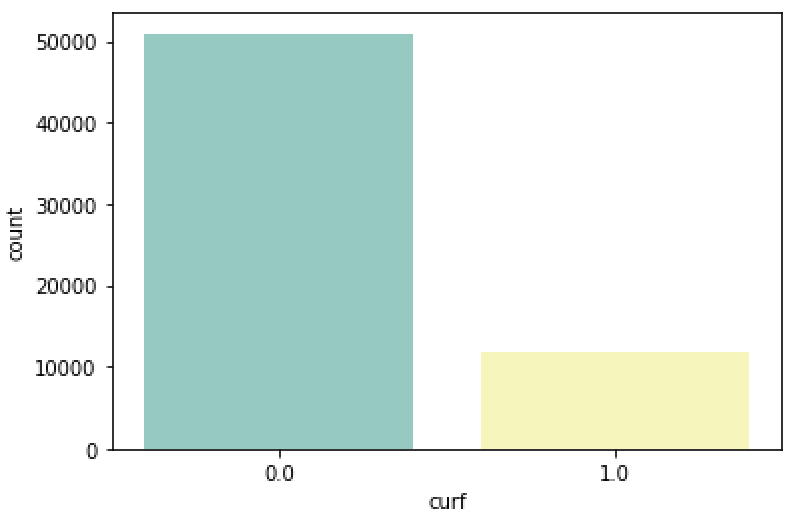

```
In [45]: sns.countplot(df1['travel_dom'], palette='Set3')
```

Out[45]: <AxesSubplot:xlabel='travel_dom', ylabel='count'>



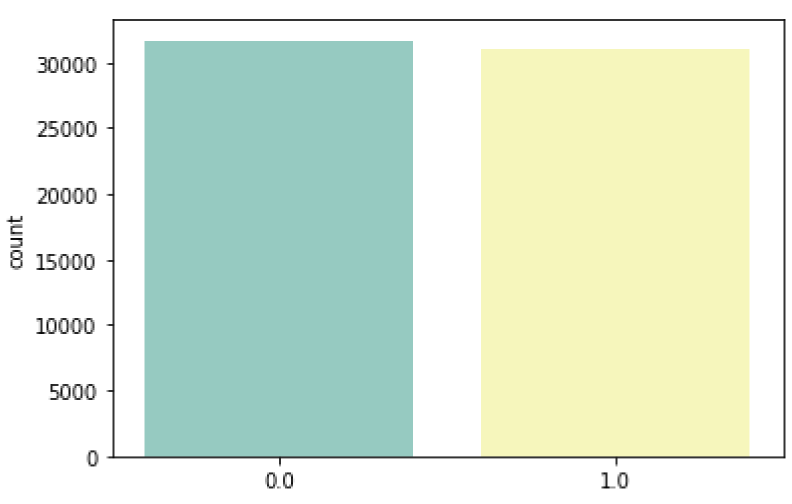
In [46]:
```
sns.countplot(df1['curf'], palette='Set3')
```

Out[46]: <AxesSubplot:xlabel='curf', ylabel='count'>

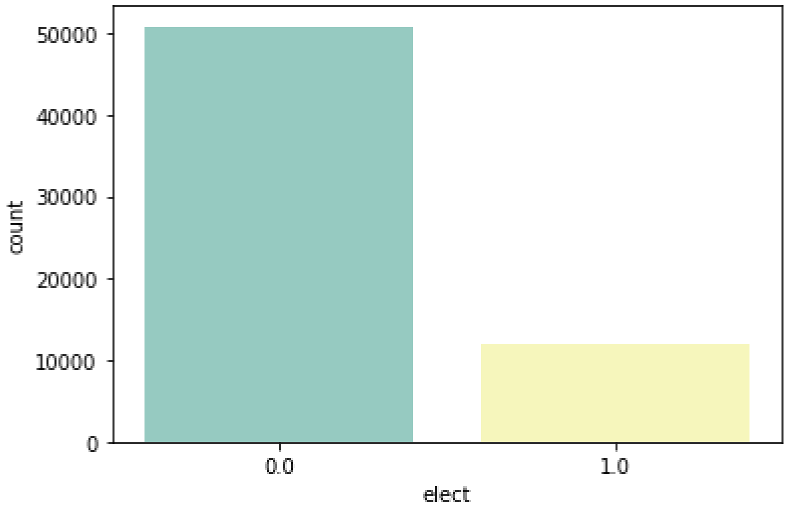


In [47]:
```
sns.countplot(df1['mass'], palette='Set3')
```

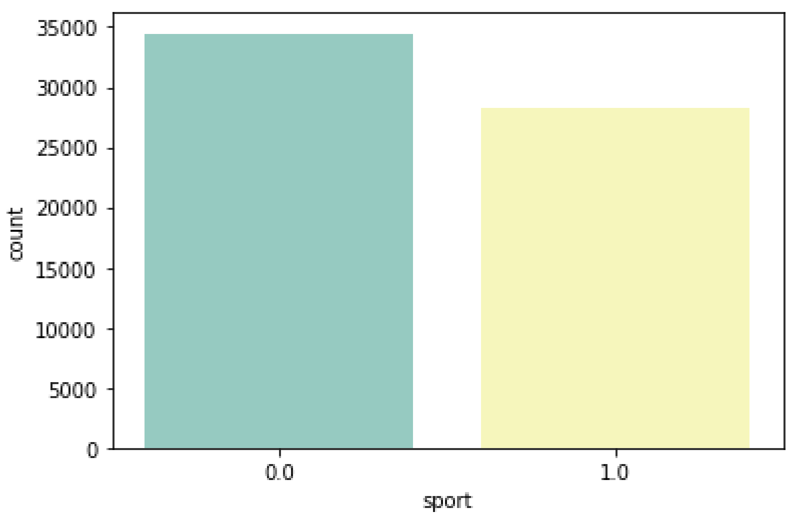
Out[47]: <AxesSubplot:xlabel='mass', ylabel='count'>



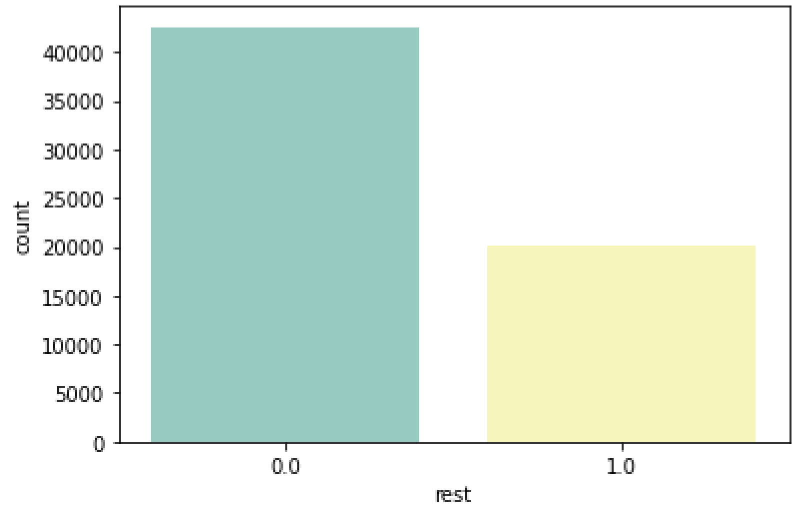
In [48]:
```
sns.countplot(df1['elect'], palette='Set3')
```
Out[48]: <AxesSubplot:xlabel='elect', ylabel='count'>



In [49]:
```
sns.countplot(df1['sport'], palette='Set3')
```
Out[49]: <AxesSubplot:xlabel='sport', ylabel='count'>

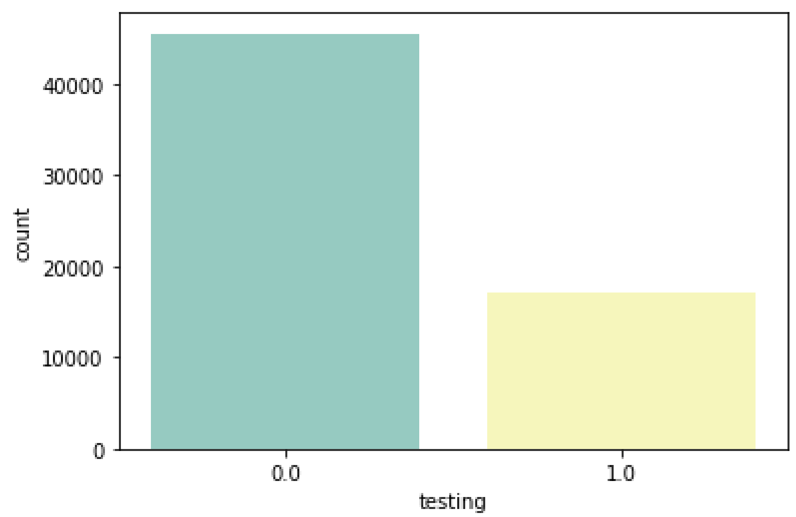


In [50]:
```
sns.countplot(df1['rest'], palette='Set3')
```
Out[50]: <AxesSubplot:xlabel='rest', ylabel='count'>



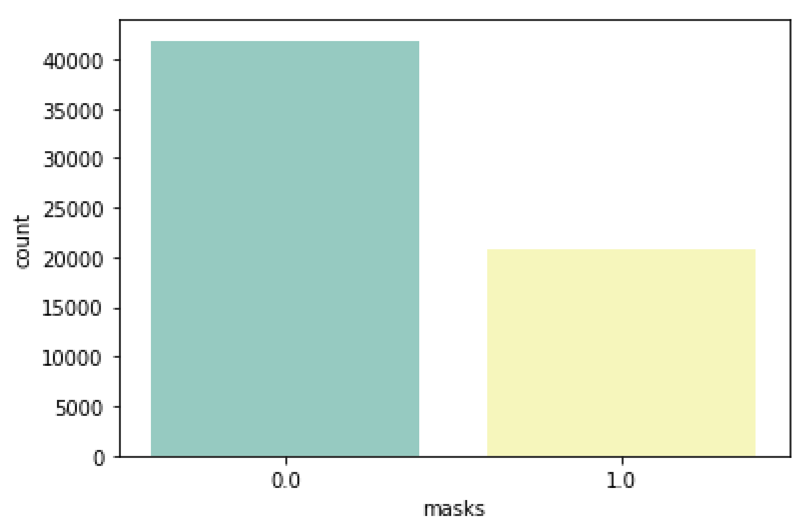
In [51]:
```
sns.countplot(df1['testing'], palette='Set3')
```

Out[51]: <AxesSubplot:xlabel='testing', ylabel='count'>



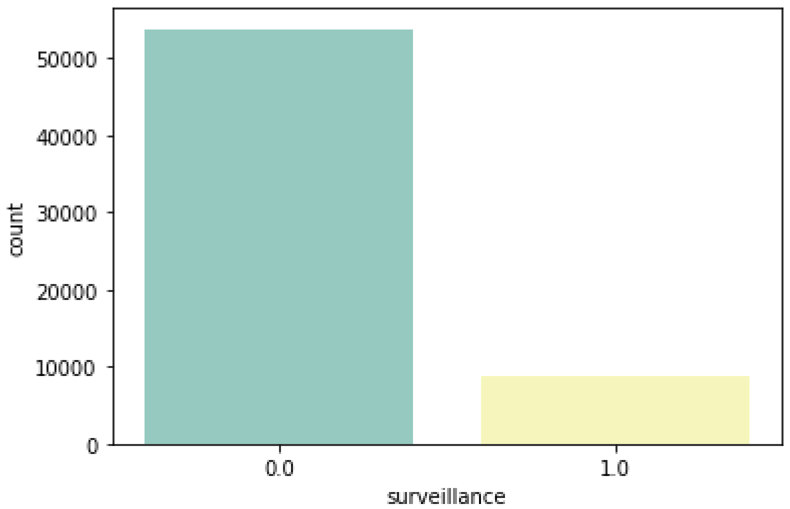
In [52]:
```
sns.countplot(df1['masks'], palette='Set3')
```

Out[52]: <AxesSubplot:xlabel='masks', ylabel='count'>



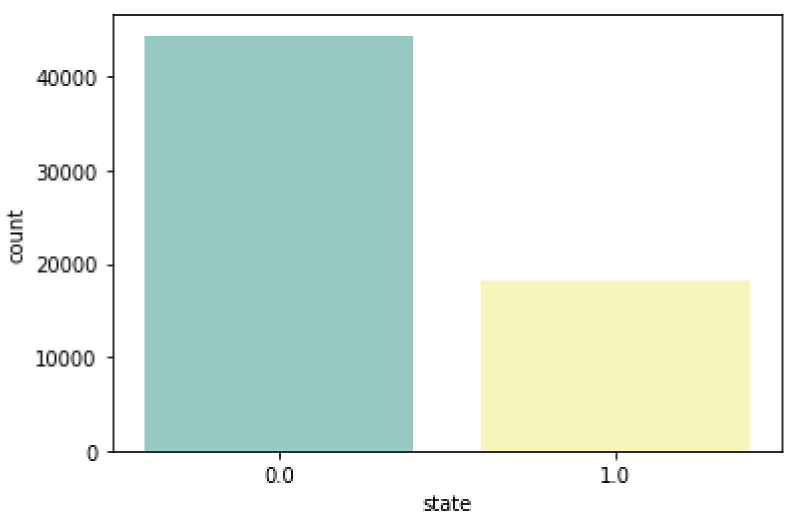
In [53]:
```
sns.countplot(df1['surveillance'], palette='Set3')
```

Out[53]: <AxesSubplot:xlabel='surveillance', ylabel='count'>



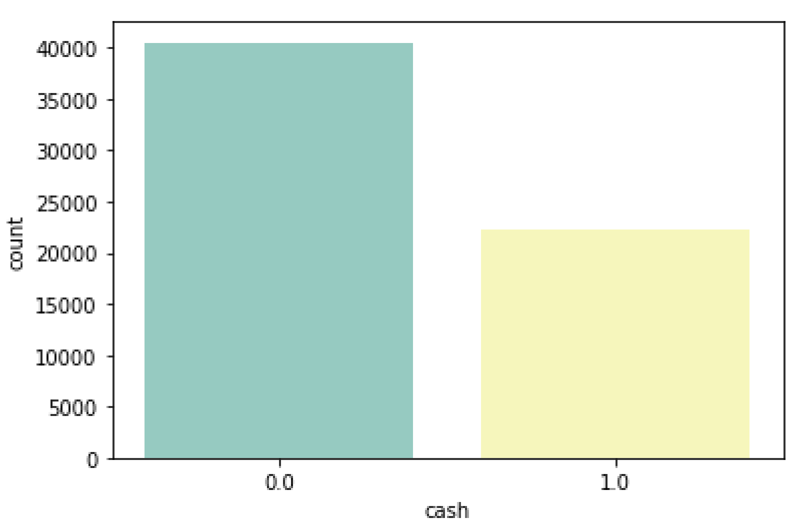
In [54]:
```
sns.countplot(df1['state'], palette='Set3')
```

Out[54]: <AxesSubplot:xlabel='state', ylabel='count'>

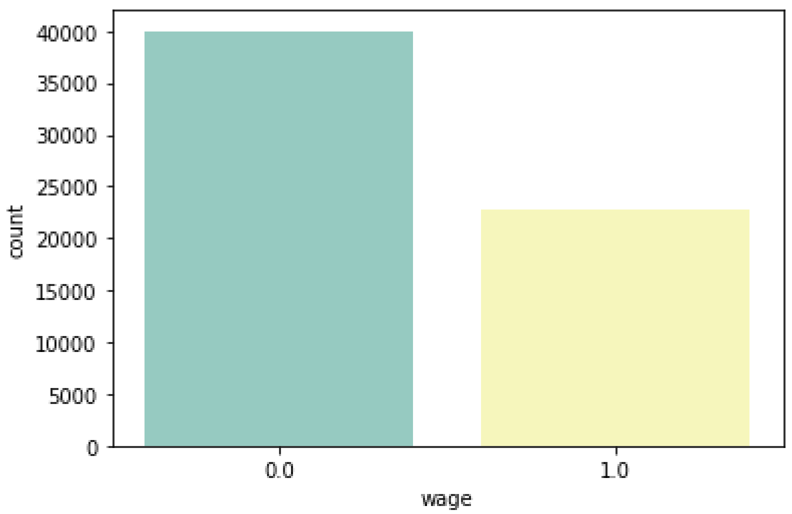


In [55]:
```
sns.countplot(df1['cash'], palette='Set3')
```

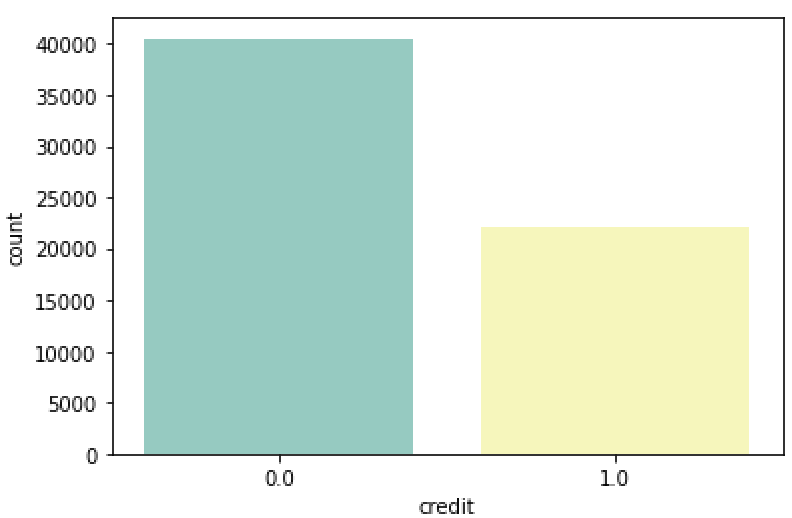
Out[55]: <AxesSubplot:xlabel='cash', ylabel='count'>



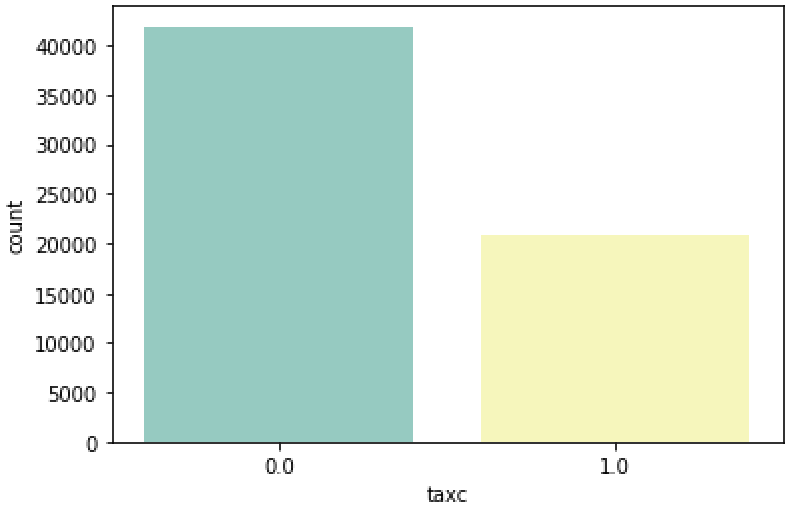
In [56]:
```
sns.countplot(df1['wage'], palette='Set3')
```
Out[56]: <AxesSubplot:xlabel='wage', ylabel='count'>



In [57]:
```
sns.countplot(df1['credit'], palette='Set3')
```
Out[57]: <AxesSubplot:xlabel='credit', ylabel='count'>

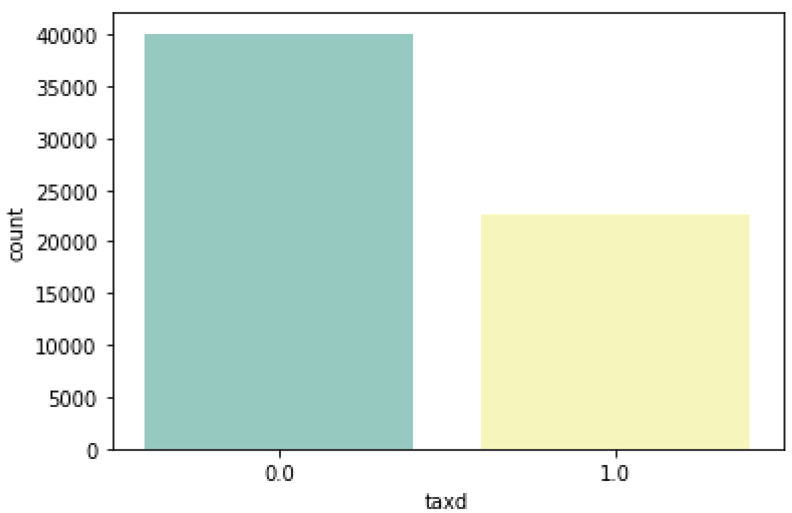


In [58]:
```
sns.countplot(df1['taxc'], palette='Set3')
```
Out[58]: <AxesSubplot:xlabel='taxc', ylabel='count'>



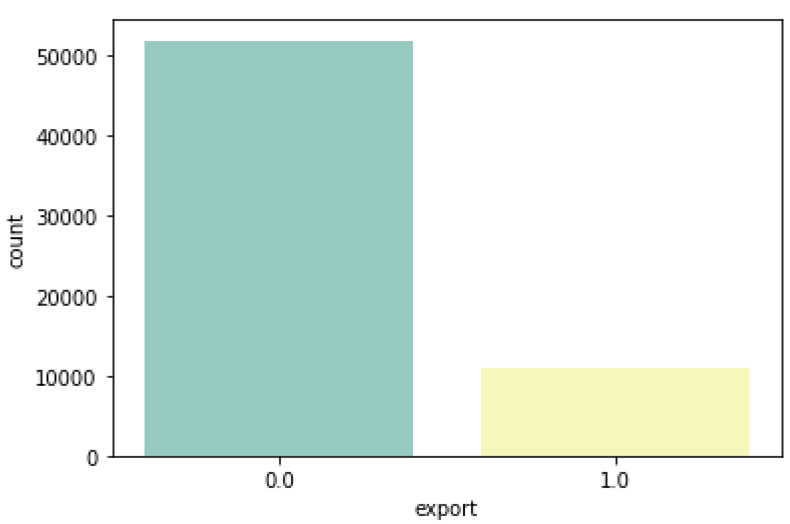
In [59]:
```
sns.countplot(df1['taxd'], palette='Set3')
```

Out[59]: <AxesSubplot:xlabel='taxd', ylabel='count'>



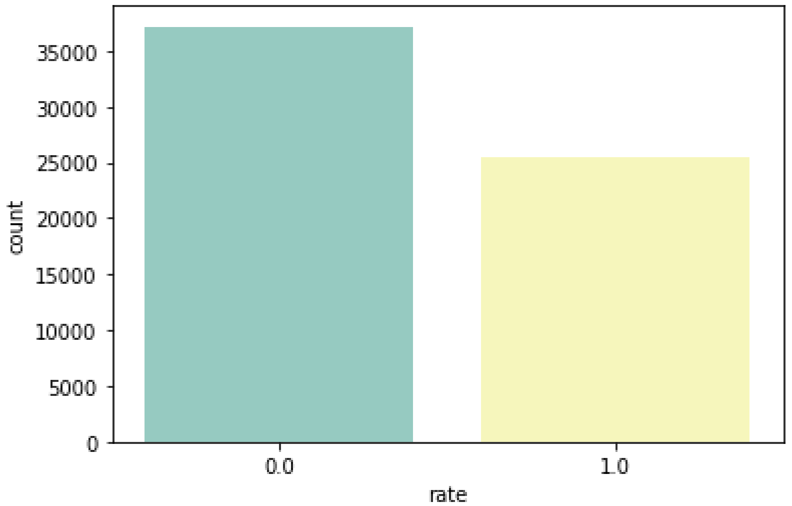
In [60]:
```
sns.countplot(df1['export'], palette='Set3')
```

Out[60]: <AxesSubplot:xlabel='export', ylabel='count'>



In [61]:
```
sns.countplot(df1['rate'], palette='Set3')
```

Out[61]: <AxesSubplot:xlabel='rate', ylabel='count'>



Feature Engineering

Makes the dependent variable y be the cases and the independent variables X be the government regulations. Then, splits data, 80/20, into a train test split.

In [63]:
```
y=df1['cases']
X=df1.drop(['cases'], axis=1)
```

Mean encoding solves for the value of the target dependent variable, which is conditional on the mean of the corresponding feature independent variable. μ is the encoded mean, n is the number of values, $\bar{x}$ is the estimated mean, m is a weight, and w is the original mean.

$$\mu = \frac{n * \bar{x} + m * w}{n + m}$$

In [81]:
```
ce_target = ce.TargetEncoder()

ce_target.fit(X, y)

ce_target.transform(X, y)
```

Out[81]:

	school	domestic	travel	travel_dom	curf	mass	elect	sport	rest	testing	masks	surveillance	state	cash	wage	credit	taxc	taxd	export	rate
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
62695	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0
62696	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0
62697	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0
62698	1.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0

	school	domestic	travel	travel_dom	curf	mass	elect	sport	rest	testing	masks	surveillance	state	cash	wage	credit	taxc	taxd	export	rate
62699	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0

62700 rows × 20 columns

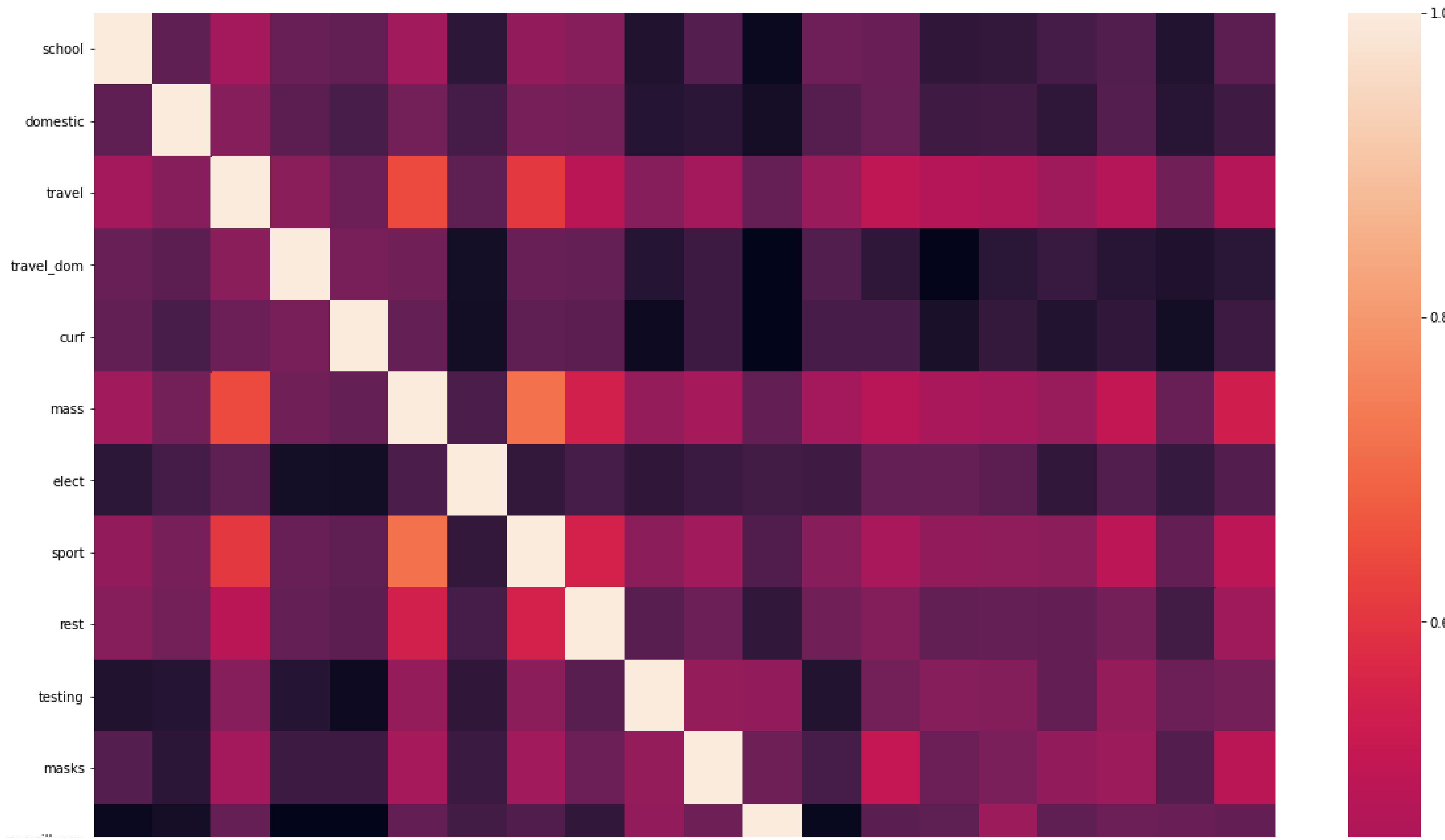
In [82]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

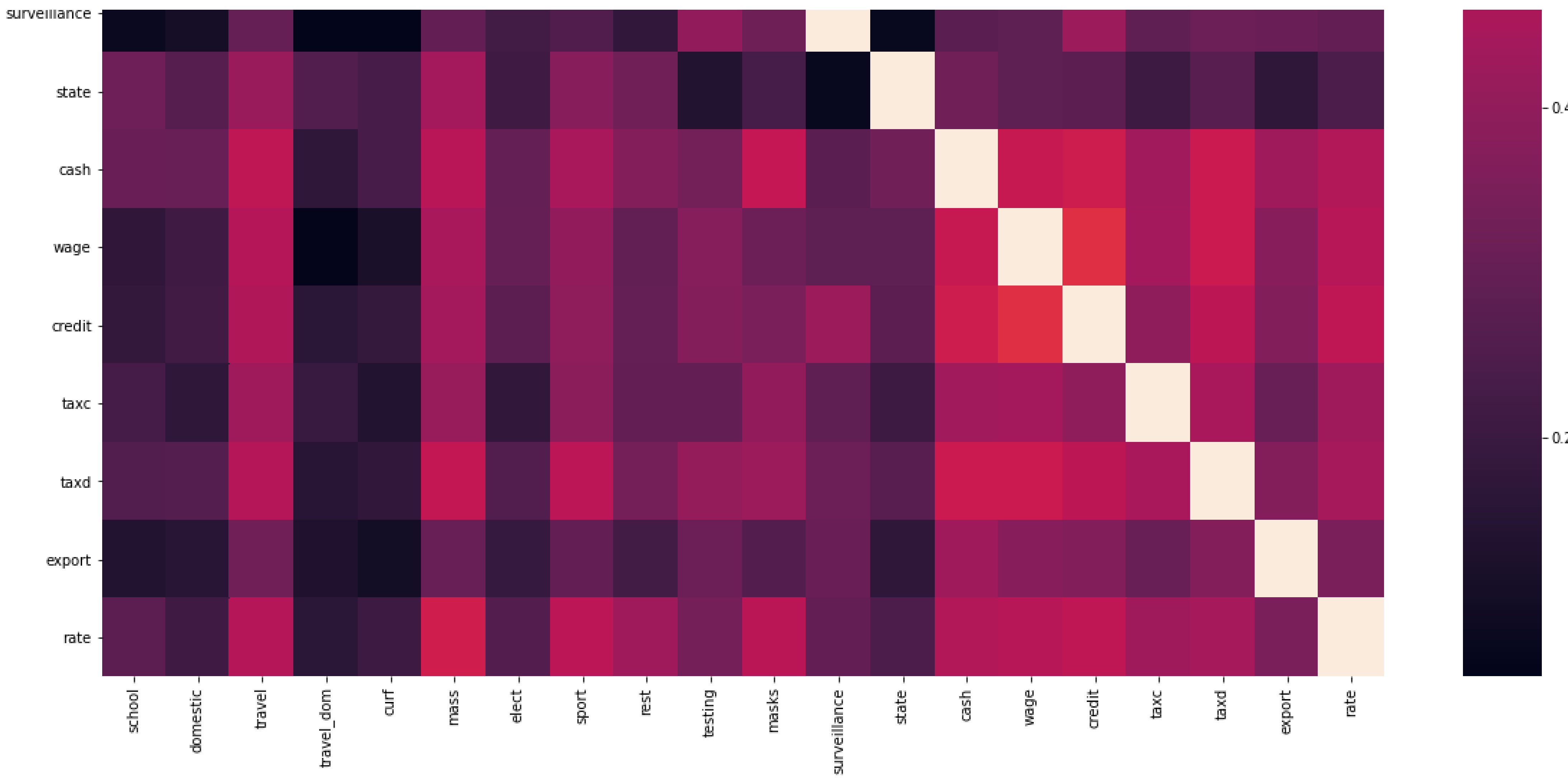
Binary logistic regression assumsions are a binary dependent variable, independent observations, linear continuous variables, no strongly influential outliers, independent variables with no multicollinearity, and and a large sample size. The dependent variable is zero or one, the observations are not repeated measurements made on each experimental unit or matched -- when pairs of data are matched based on similar features, there are no continuous variables, the data are zeros and ones so there are no outliers, and the data has 50160 rows.

Check for muticollinearity.

In [83]:
```
x_corr=X_train.corr(method='pearson')
plt.figure(figsize=[20, 20])
sns.heatmap(x_corr)
```

Out[83]: <AxesSubplot:>





In [26]:
```
correlation=[]
for columnName1, columnData1 in X_train.iteritems():
    for columnName2, columnData2 in X_train.iteritems():
        if abs(columnData1.corr(columnData2)) > .7:
            correlation.append((columnName1, columnName2, abs(columnData1.corr(columnData2))))
correlation
```

Out[26]:
```
[('school', 'school', 1.0),
 ('domestic', 'domestic', 0.9999999999999999),
 ('travel', 'travel', 1.0),
 ('travel_dom', 'travel_dom', 0.9999999999999999),
 ('curf', 'curf', 1.0),
 ('mass', 'mass', 0.9999999999999999),
 ('mass', 'sport', 0.7186910509080194),
 ('elect', 'elect', 1.0),
 ('sport', 'mass', 0.7186910509080194),
 ('sport', 'sport', 1.0),
 ('rest', 'rest', 1.0),
 ('testing', 'testing', 1.0),
 ('masks', 'masks', 1.0),
 ('surveillance', 'surveillance', 1.0),
 ('state', 'state', 1.0),
 ('cash', 'cash', 1.0),
 ('wage', 'wage', 1.0),
 ('credit', 'credit', 1.0),
 ('taxc', 'taxc', 1.0),
 ('taxd', 'taxd', 1.0),
```

```
 ('export', 'export', 1.0),
 ('rate', 'rate', 1.0)]
```

Remove multicollinear feature from X.

In [86]:
```
X_train=X_train.drop('sport', axis=1)
X_test=X_test.drop('sport', axis=1)
```

Logistic Regression Model

The Logistic function, logit(P), starts with setting the log odds equal to the parameters. Exponentiate both sides of the equation and cross multiply 1-P to get the logistic function.

$$ln(\frac{P}{1-P}) = \alpha + \beta x$$

$$\frac{P}{1-P} = e^{\alpha+\beta x}$$

$$P = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

The probability of class 1 is the logistic function and of class 0 is one minus the logistic function.

$$P(Class = 1|X = x) = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

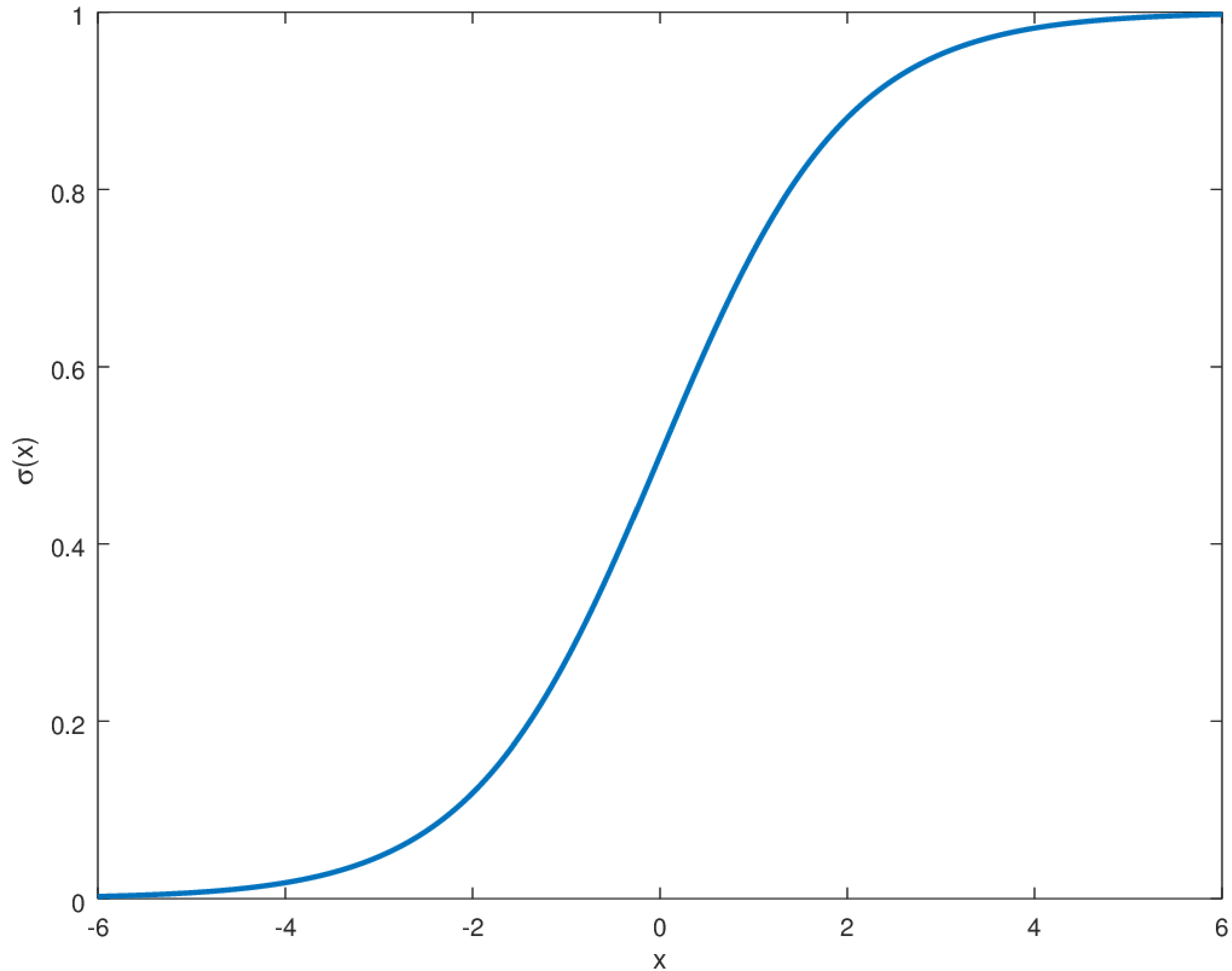
$$P(Class = 0|X = x) = 1 - \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

Take the inverse of P to get the desired sigmoid function.

$$\sigma(x) = P^{-1} = \frac{1}{1+e^{-(\alpha+\beta x)}}$$

In [27]:
```
Image(filename='sigmoid_function.png')
```

Out[27]:

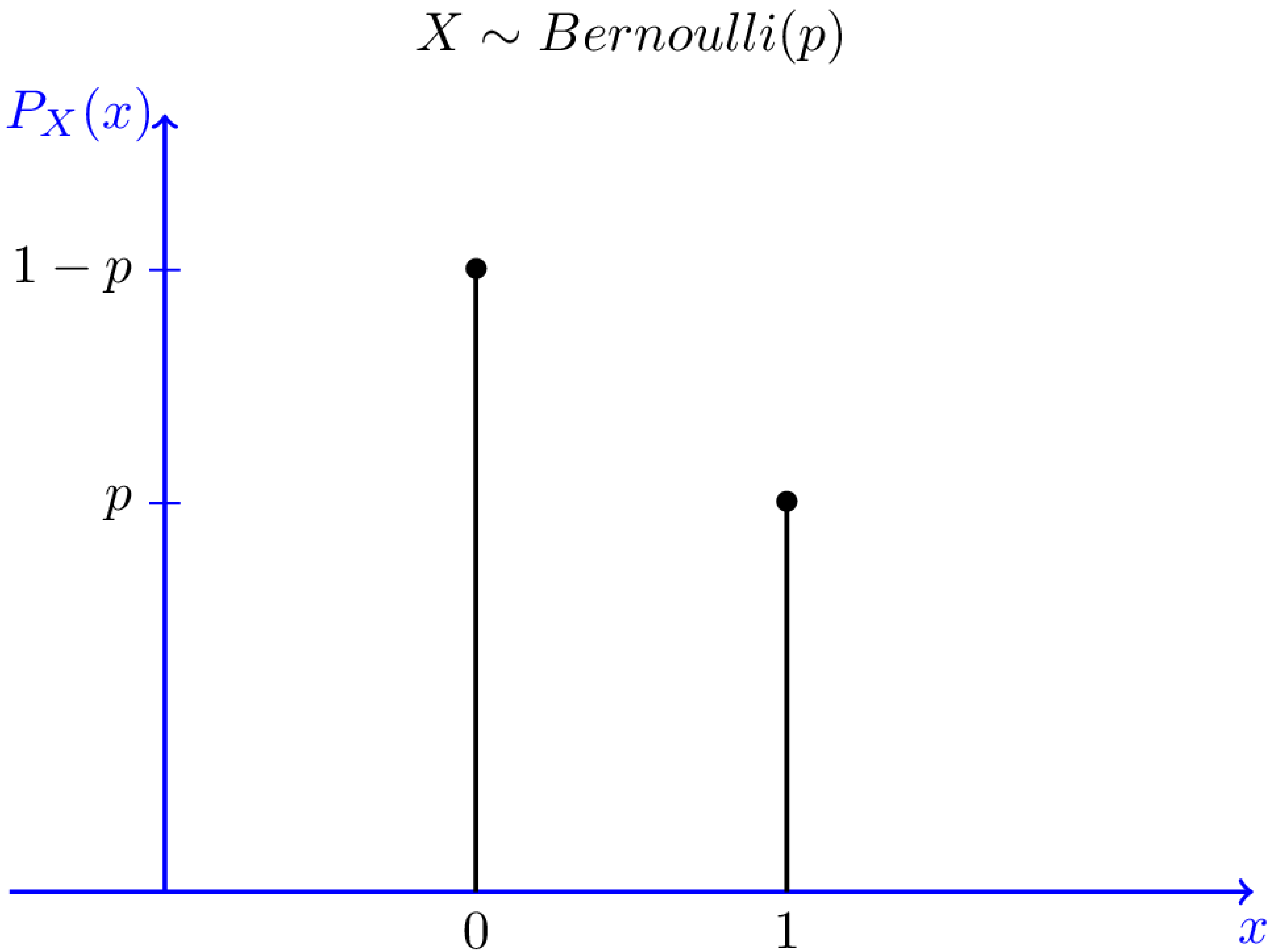


The predicted labels are binary so each label is a Bernoulli random variable from a Bernoulli probability mass function.

$$P(x;p) = \begin{cases} p & : x = 1 \\ 1-p & : x = 0 \end{cases}$$

In [28]:
```
Image(filename='bernoulli(p) color.png')
```

Out[28]:



MLE, maximum likelihood estimation, finds the parameters, θ, that maximize the log likelihood probability, a model comparitive metric, that a value belongs to a class.

$$MLE = \prod_i^N P(Y = y_i | X = x_i)$$

$$argmaxLL(\theta) = \sum_i^N y_i * log(\sigma(\theta^T * x_i)) + (1 - y_i) * log(1 - \sigma(\theta^T * x_i))$$

Solve for θ that maximizes the the log likelihood by solving for the partial derivative of the log likelihood with respect to θ and gradiently accend toward the maximum of the LL function.

$$\nabla \frac{\partial LL(\theta)}{\partial \theta_j} = \sum_i^N [y_i - \theta^T * x_i] x_{ij}$$

Iteratively accend toward maximum LL with an η stepsize.

$$\theta_j^{new} = \theta_j^{previous} + \eta * \nabla \frac{\partial LL(\theta^{previous})}{\partial \theta_j^{previous}}$$

The McFadden R squared, a model comparitive metric, is 1 minus the log likelihood of the full model, which is like the sum of squared residuals, devided by the log likelihood of the intercept model, which is like the total sum of squares.

$$R^2_{McF} = 1 - \frac{lnL(M_{full})}{lnL(M_{int})}$$

Conduct logistic regression model from statsmodels for regression results.

In [87]:
```
log_reg = sm.Logit(y_train, X_train).fit()
log_reg.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.572699
         Iterations 6
```

Out[87]:

Logit Regression Results

Dep. Variable:	cases	**No. Observations:**	50160
Model:	Logit	**Df Residuals:**	50141
Method:	MLE	**Df Model:**	18
Date:	Thu, 01 Apr 2021	**Pseudo R-squ.:**	0.1735
Time:	17:55:59	**Log-Likelihood:**	-28727.
converged:	True	**LL-Null:**	-34758.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
school	-0.2891	0.025	-11.447	0.000	-0.339	-0.240
domestic	0.5612	0.036	15.696	0.000	0.491	0.631
travel	-0.5174	0.031	-16.442	0.000	-0.579	-0.456
travel_dom	0.1109	0.033	3.333	0.001	0.046	0.176
curf	0.1589	0.032	4.966	0.000	0.096	0.222
mass	0.3878	0.033	11.909	0.000	0.324	0.452
elect	0.4917	0.032	15.245	0.000	0.428	0.555
rest	0.4321	0.030	14.302	0.000	0.373	0.491
testing	-0.0400	0.030	-1.327	0.184	-0.099	0.019

masks	0.7622	0.030	25.037	0.000	0.703	0.822
surveillance	0.3914	0.041	9.471	0.000	0.310	0.472
state	-0.1466	0.028	-5.220	0.000	-0.202	-0.092
cash	0.7558	0.033	23.231	0.000	0.692	0.820
wage	-0.0793	0.032	-2.464	0.014	-0.142	-0.016
credit	0.0880	0.032	2.730	0.006	0.025	0.151
taxc	-0.3708	0.029	-12.645	0.000	-0.428	-0.313
taxd	0.4579	0.031	14.907	0.000	0.398	0.518
export	0.1851	0.036	5.174	0.000	0.115	0.255
rate	-0.0738	0.030	-2.493	0.013	-0.132	-0.016

Remove statistically insignificant feature.

In [88]:
```
X_train=X_train.drop('testing', axis=1)
```

In [89]:
```
log_reg = sm.Logit(y_train, X_train)
model=log_reg.fit()
model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.572717
         Iterations 6
```

Out[89]:

Logit Regression Results

Dep. Variable:	cases	**No. Observations:**	50160
Model:	Logit	**Df Residuals:**	50142
Method:	MLE	**Df Model:**	17
Date:	Thu, 01 Apr 2021	**Pseudo R-squ.:**	0.1735
Time:	17:56:06	**Log-Likelihood:**	-28727.
converged:	True	**LL-Null:**	-34758.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
school	-0.2873	0.025	-11.390	0.000	-0.337	-0.238
domestic	0.5625	0.036	15.739	0.000	0.492	0.633
travel	-0.5209	0.031	-16.599	0.000	-0.582	-0.459
travel_dom	0.1094	0.033	3.288	0.001	0.044	0.175
curf	0.1604	0.032	5.018	0.000	0.098	0.223
mass	0.3825	0.032	11.832	0.000	0.319	0.446
elect	0.4915	0.032	15.241	0.000	0.428	0.555
rest	0.4286	0.030	14.236	0.000	0.370	0.488
masks	0.7553	0.030	25.180	0.000	0.696	0.814
surveillance	0.3820	0.041	9.381	0.000	0.302	0.462

state	-0.1439	0.028	-5.139	0.000	-0.199	-0.089
cash	0.7571	0.033	23.282	0.000	0.693	0.821
wage	-0.0826	0.032	-2.574	0.010	-0.145	-0.020
credit	0.0864	0.032	2.683	0.007	0.023	0.149
taxc	-0.3696	0.029	-12.610	0.000	-0.427	-0.312
taxd	0.4524	0.030	14.867	0.000	0.393	0.512
export	0.1811	0.036	5.080	0.000	0.111	0.251
rate	-0.0729	0.030	-2.464	0.014	-0.131	-0.015

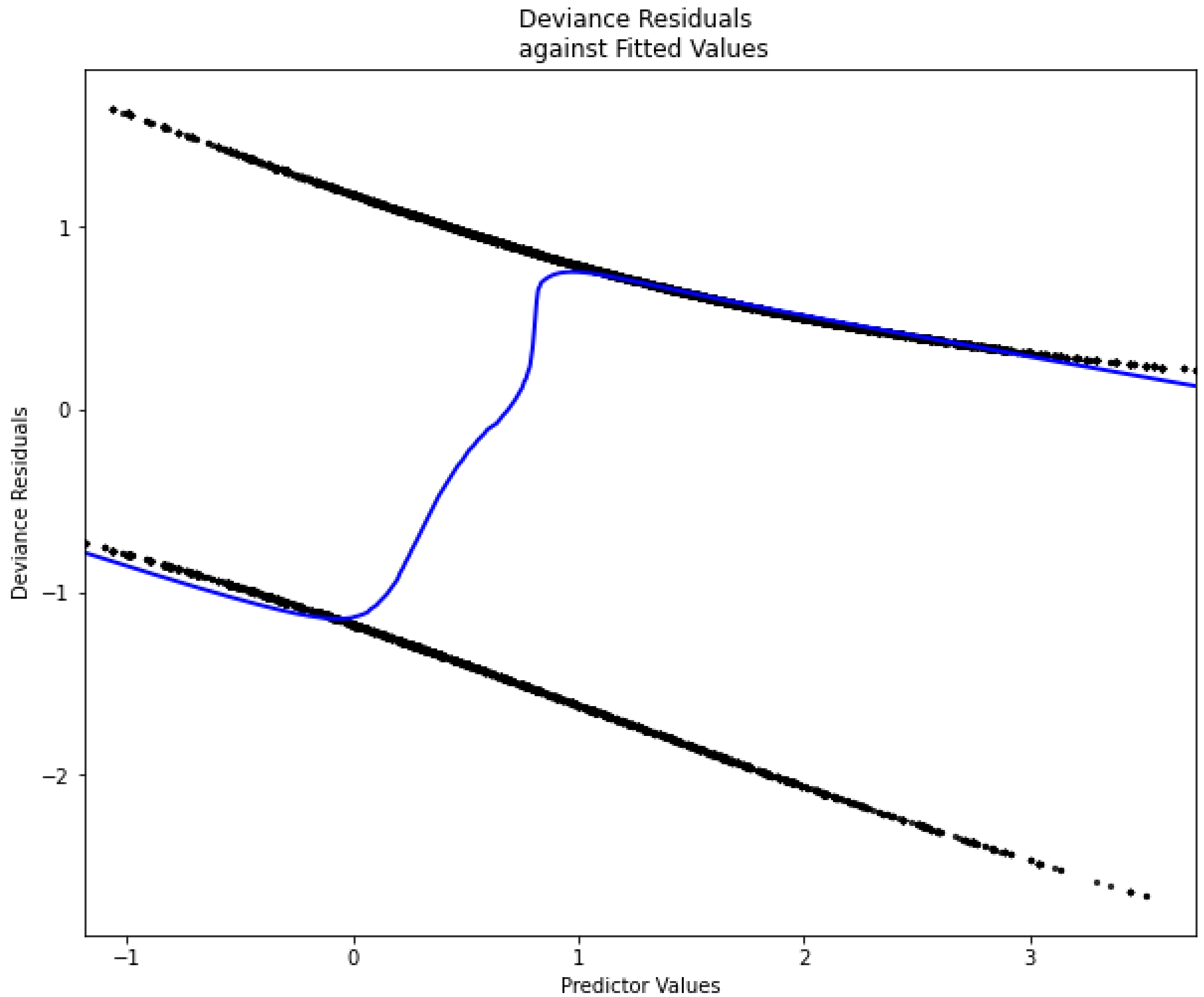
The presence of the independent variables increases or decreases the log odds of the presence of the dependent variable based on the sign of the parameter coefficients. The government regulations of school closures, travel restrictions, state of emergency declarations, wage support, tax credits, and interest rate lowering decreased the log odds of the presence of virus cases.

Goodness-of-fit tests determine whether the predicted probabilities deviate from the observed probabilities. Deviance is the difference of likelihoods between the fitted model and the residuals. 0 predicted residuals are negative and 1 predicted residuals are positive. µî are the fitted values and yi are the observed values.

$$DevianceResiduals = \sum_i^N \sqrt{2[yi * log(y_i/\hat{\mu}_i) + (n_i - y_i) * log(n_i - y_i/n_i - \hat{\mu}_i)]}$$

In [33]:

```
fig, ax = plt.subplots(1, figsize=(10, 8))
sns.regplot(model.fittedvalues, model.resid_dev, ax= ax,
            color="black", scatter_kws={"s": 5},
            line_kws={"color":"b", "alpha":1, "lw":2}, lowess=True)
plt.title("Deviance Residuals \n against Fitted Values")
plt.xlabel("Predictor Values")
plt.ylabel("Deviance Residuals")
plt.show()
```

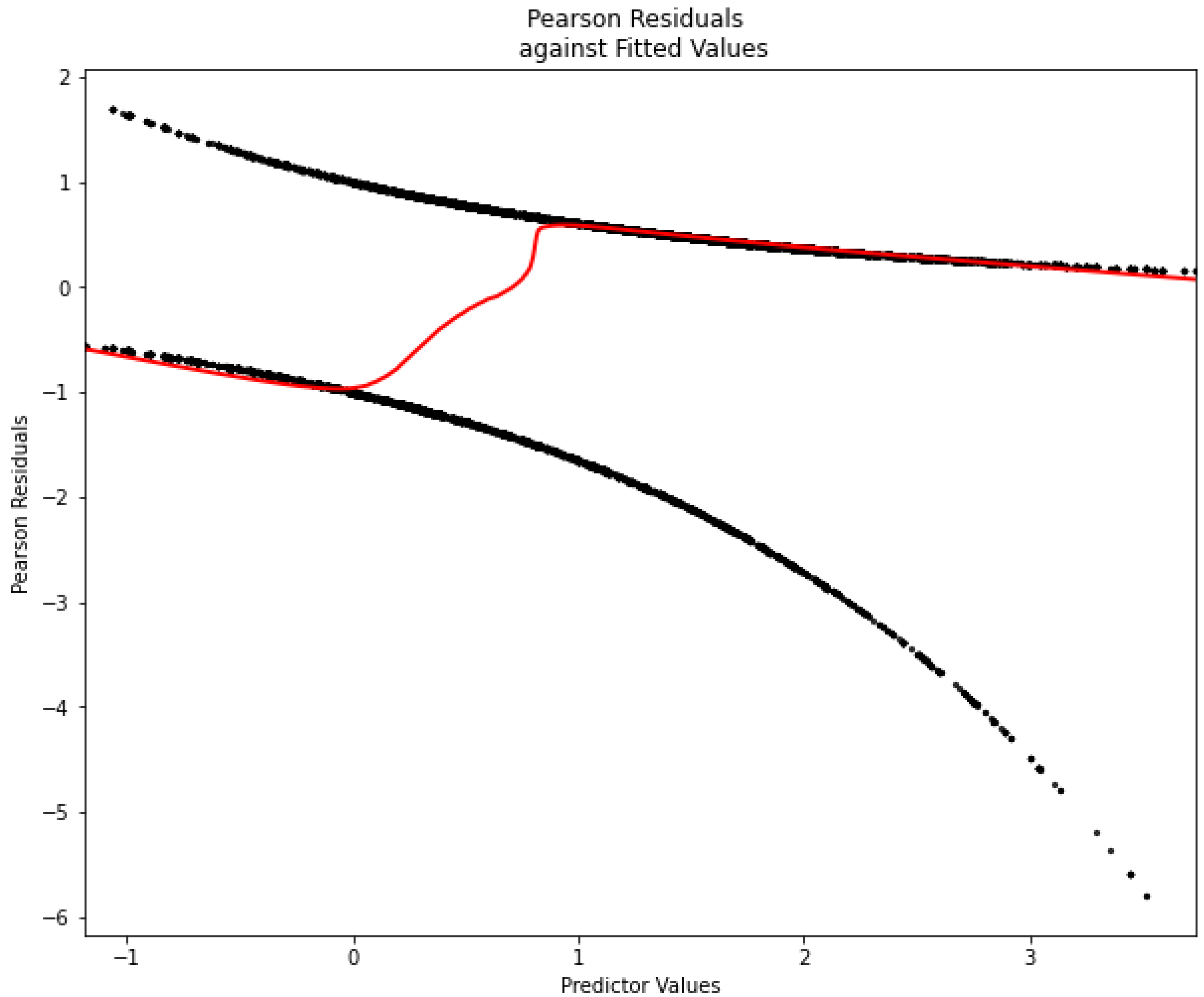


$$PearsonResiduals = \sum_i^N \frac{y_i - \hat{\mu}_i}{\sqrt{\hat{\mu}_i(n_i - \hat{\mu}_i)/n_i}}$$

In [34]:
```
fig, ax = plt.subplots(1, figsize=(10, 8))
sns.regplot(model.fittedvalues, model.resid_pearson, ax= ax,
            color="black", scatter_kws={"s": 5},
            line_kws={"color":"r", "alpha":1, "lw":2}, lowess=True)

plt.title("Pearson Residuals \n against Fitted Values")
plt.xlabel("Predictor Values")
plt.ylabel("Pearson Residuals")
```

Out[34]:
```
Text(0, 0.5, 'Pearson Residuals')
```



Pearson's chi-squared test is a goodness-of-fit test that determines whether categorical observed values, O, are consistent with their coresponding expected values, E.

$$\chi 2 = \sum_i^N \frac{(O_i - E_i)^2}{E_i}$$

$$E(x) = \sum_i^N x_i * p(x_i)$$

In [43]:

```
stat, p = chisquare(model.resid_pearson)#Null Hypothesis: no significant difference between the observed and the expected values
print('Stat:', stat, 'P-value:', p)
alpha=.05
if p>alpha:
    print('Don\'t reject null of no significant difference between the observed and the expected values.')
else:
    print('Reject null of no significant difference between the observed and the expected values.')
```

```
Stat: -145098.9073862549 P-value: 1.0
Don't reject null of no significant difference between the observed and the expected values.
```

Conduct logistic regression model from sklearn for classification results.

In [66]:

```
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)
y_hat_train = logistic_regression.predict(X_train)
```

To solve for the log odds ratio for each independant variable with the dependent variable, take the difference of the log odds, and then set the result as the exponent to base e.

$$odds_0 = \frac{p}{(1-p)}$$

$$odds_1 = \frac{(1-p)}{p}$$

$$ln(odds_0) - ln(odds_1) = ln(odds_0/odds_1)$$

$$e^{ln(odds_0/odds_1)} = odds_0/odds_1$$

The odds of each feature are the following more times likely to be present than cases.

In [130…

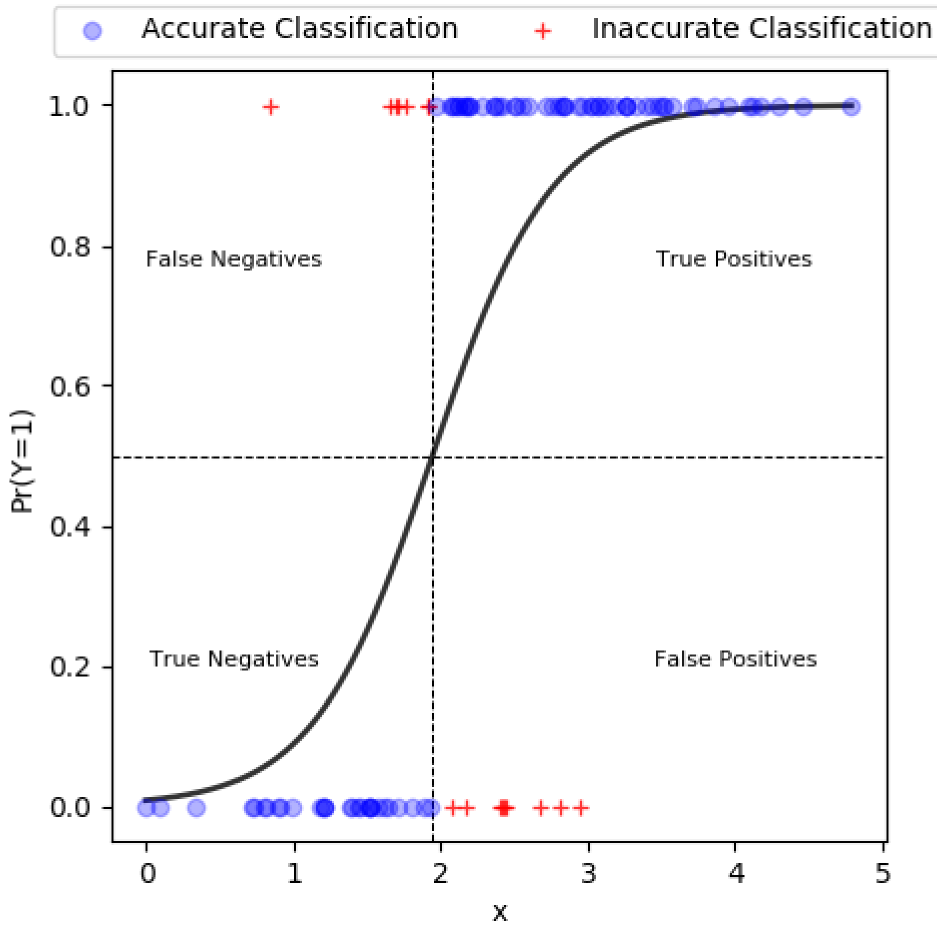
```
for i,l in zip(logistic_regression.coef_[0][1:],X_train.columns):
    odds_difference=i-logistic_regression.coef_[0][0]
    odds_ratio=np.exp(odds_difference)
    print(f'{l}: {odds_ratio}')
```

```
school: 1.051222393583342
domestic: 0.7545839686306944
travel: 0.7077537943155912
travel_dom: 0.7620597488754686
curf: 1.3555413658845121
mass: 1.6701484676348024
elect: 0.9105463643884978
rest: 0.7322113269115323
masks: 1.844595502211115
surveillance: 0.8135883192534
state: 0.6393028151928141
cash: 1.1134221818661332
wage: 0.7398151724479073
credit: 0.717032473923729
taxc: 0.5321042678388413
taxd: 1.0579507399508241
export: 0.7716020552223852
rate: 0.7598709410243987
```

In [45]:

```
Image(filename='lr.png')
```

Out[45]:

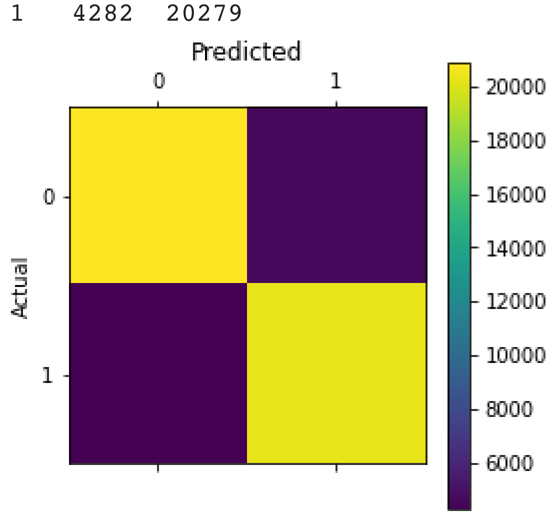


For the train set, the model has 20765 true positives, 20242 true negatives, 4834 false positives, and 4319 false negatives.

In [129…

```
con_mat(y_train, y_hat_train)
```

```
Confusion Matrix
----------------
        0      1
0   20913   4686
1    4282  20279
```



$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

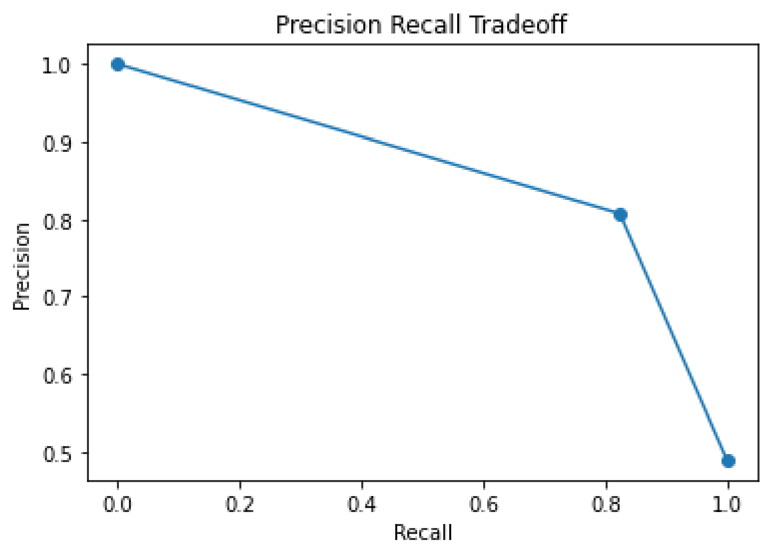
$$f1 = \frac{2 * precision * recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$specificity = \frac{TN}{TN + FP}$$

In [49]:
```
Metrics(y_train, y_hat_train)
```

```
Precision Score: 0.8072260328601053
Recall Score: 0.8241521110703962
F1 Score: 0.8156012651852449
Accuracy Score: 0.817523923444976
Specificity Score: 0.8278185297400733
```



$$TruePositiveRate = \frac{TP}{TP + FN}$$

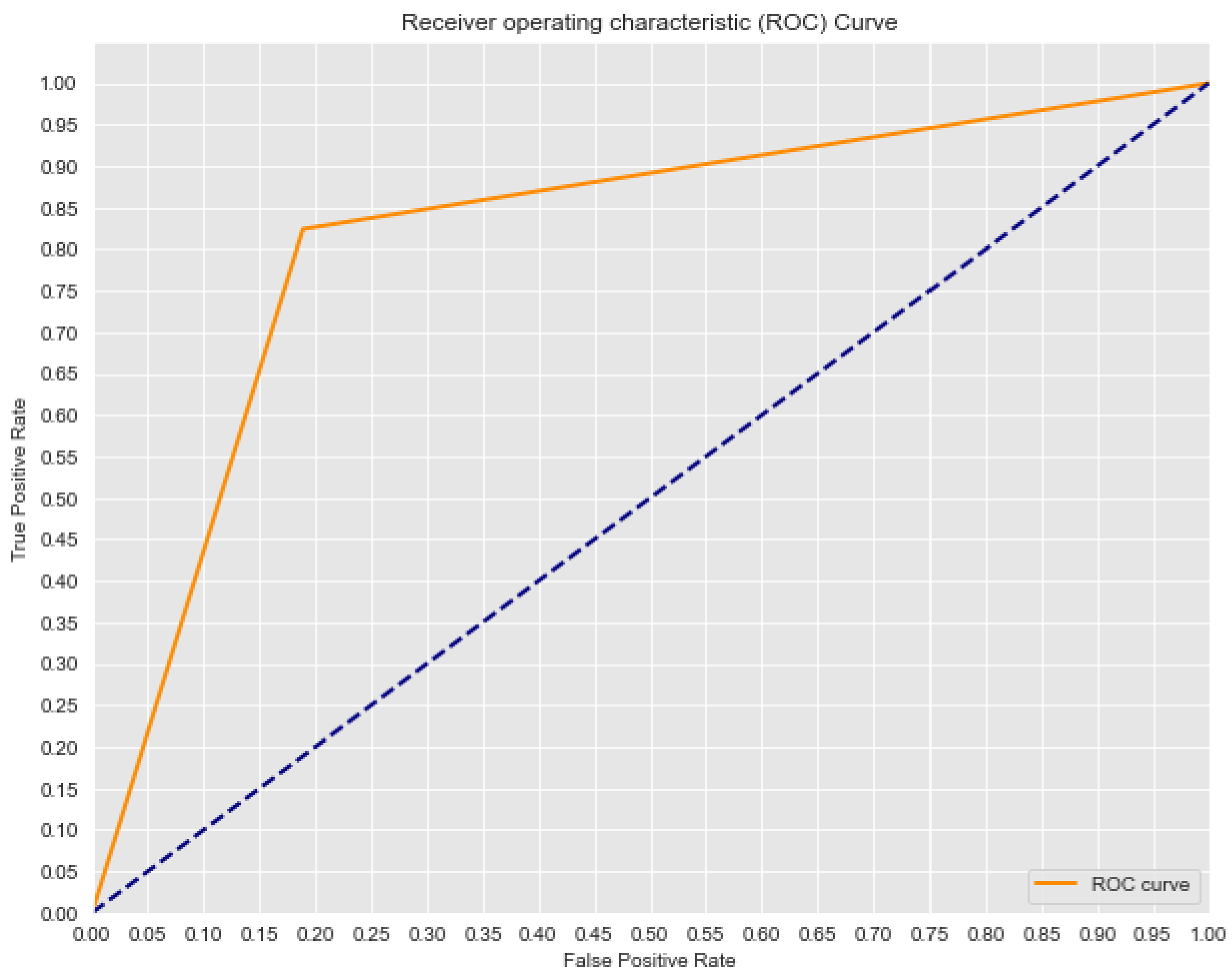
$$FalsePositiveRate = \frac{FP}{FP + TN}$$

$$ReceiverOperatingCharacteristic = \frac{TPR}{FPR}$$

$$AreaUnderCurve = \int_a^b TPR(FPR^{-1}(x))dx$$

In [60]:
```
roc(y_train, y_hat_train)
```

```
AUC: 0.8176583048418117
```



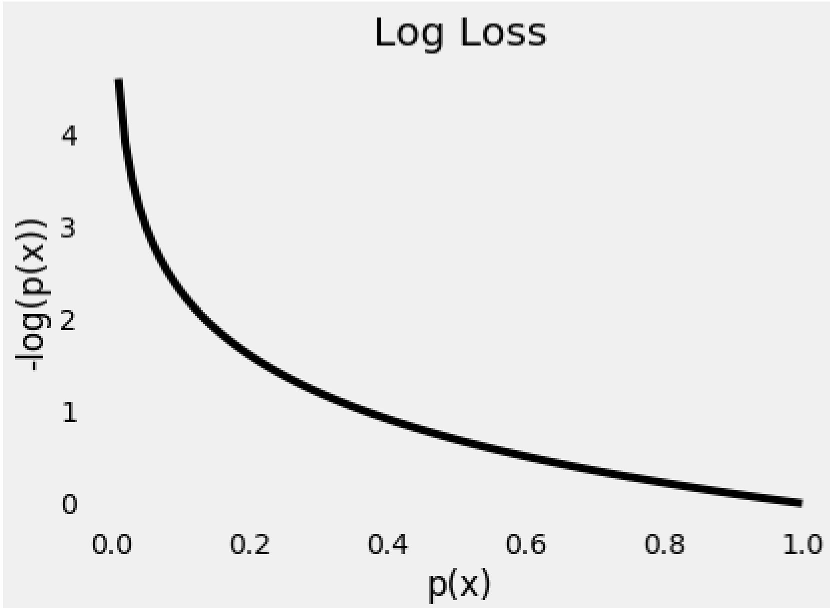
The log loss, a cost function, is the cross entropy between the distribution of the true labels and the predictions. Entropy measures unpredictability and cross entropy incorporates the entropy of the predicted distribution with of the true distribution. The log loss multiplies -1 by the log likelihood to identify that lower scores are better, devides the result by the sample size, and reults in the mean loss. As the loss approaches 0, the probability of correct classification increases.

$$NegativeLogLoss = -\frac{1}{N}\sum_{i}^{N} y_i * log(\sigma(\theta^T * x_i) + (1 - y_i) * log(1 - \sigma(\theta^T * x_i))$$

In [51]:

```
Image(filename='log_loss.png')
```

Out[51]:



In [52]:
```
cv_score = cross_val_score(logistic_regression, X_train, y_train, cv=5, scoring='neg_log_loss')
mean_cv_score = np.mean(cv_score)
print('Mean Cross Validation of Cost Function')
print(f"Negative Log Loss Score: {mean_cv_score}")
```

```
Mean Cross Validation of Cost Function
Negative Log Loss Score: -0.42917685692963775
```

Test model on test set.

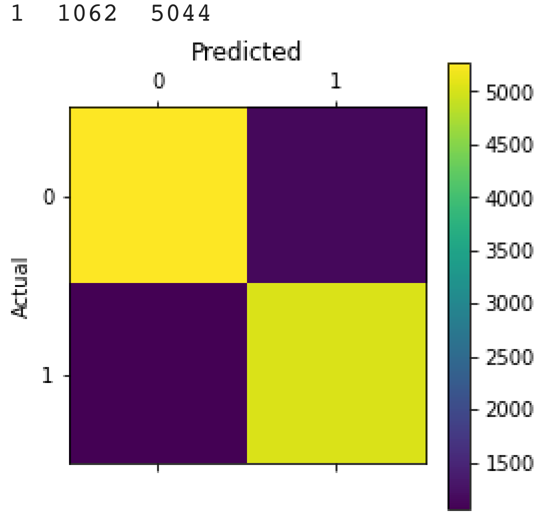
In [126…
```
logistic_regression.fit(X_test, y_test)

y_hat_test = logistic_regression.predict(X_test)
```

For the test set, the model has 5233 true positives, 5043 true negatives, 1201 false positives, and 1063 false negatives.

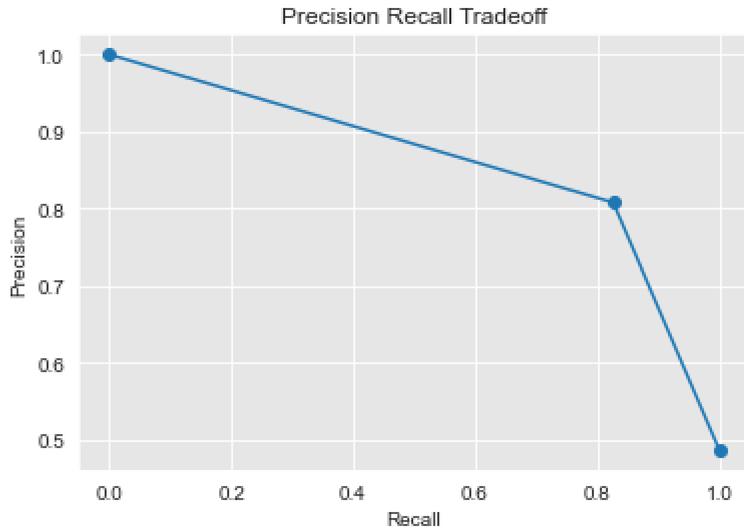
In [128…
```
con_mat(y_test, y_hat_test)
```

```
Confusion Matrix
----------------
      0     1
0  5277  1157
1  1062  5044
```



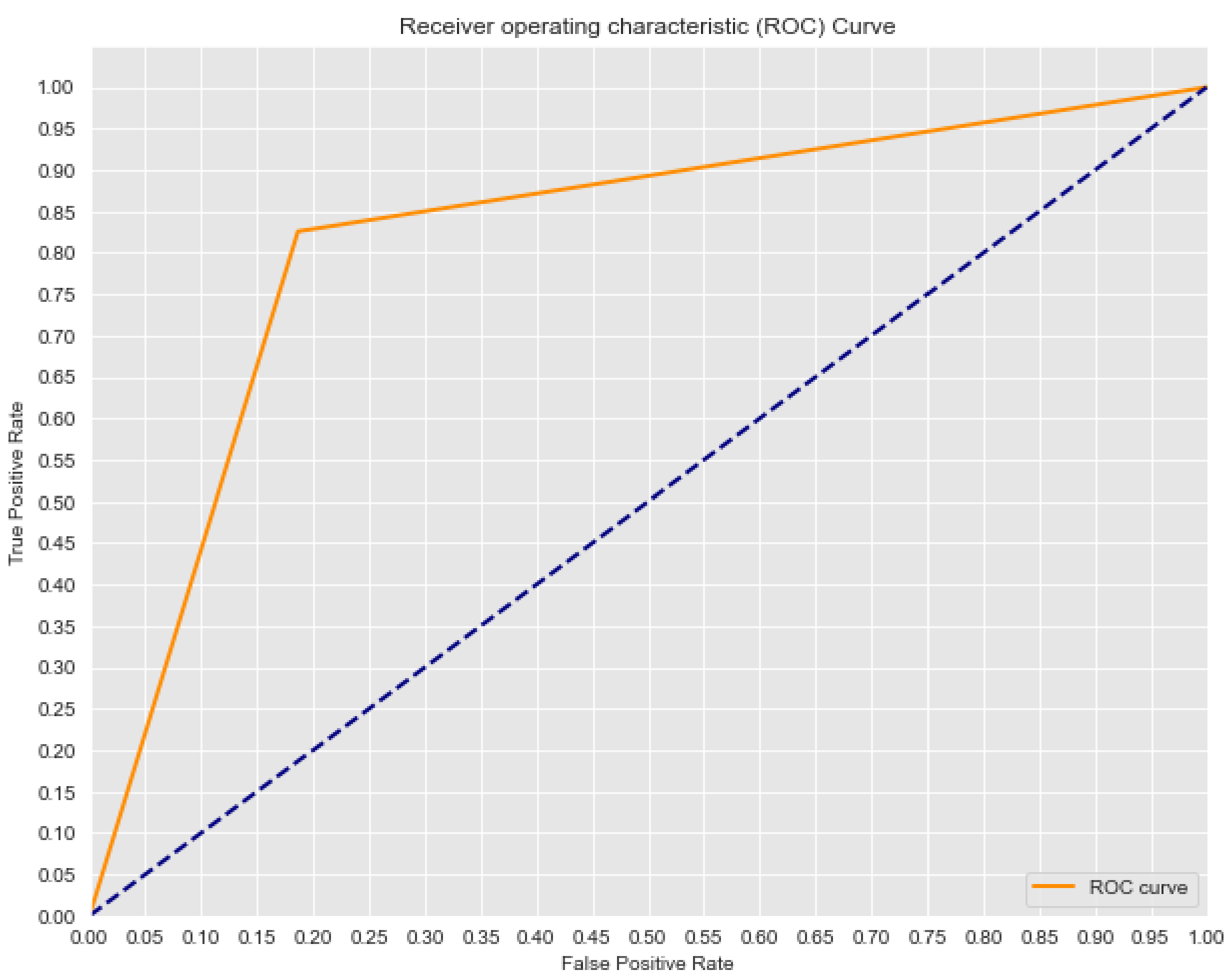
In [56]:
```
Metrics(y_test, y_hat_test)
```

```
Precision Score: 0.8076553491351698
Recall Score: 0.8259089420242385
F1 Score: 0.8166801619433198
Accuracy Score: 0.8194577352472089
Specificity Score: 0.8311626429479034
```



In [61]:
```
roc(y_test,y_hat_test)
```

```
AUC: 0.8196221738408417
```



Conclusion

The logit model suggests that the government regulations of school closures, travel restrictions, state of emergency declarations, wage support, tax credits, and interest rate lowering decreased the log odds of the presence of virus cases. Government regulations such as disallowing public gatherings and mandating wearing masks did not decrease the log odds of the presence of virus cases. The virus travels in sneezed or coughed droplets of mucus or saliva, is airborne for a few moments, and then lands on a surface. Instead of wearing masks and preventing gatherings, carrying a handkerchiefs in which people could sneeze or cough and sanitizing areas were people gather would be sufficient in preventing the spread of SARS-CoV-2.

In [ ]:
```

```